Machine learning methods for preference aggregation

Hanna Kujawska

Dissertation for the Master's degree (MSc)
at the University of Bergen, Norway

2019

Dissertation date: November 7, 2019

Year: 2019
Title: Machine learning methods for preference aggregation
Author: Hanna Kujawska

Scientific environment

The author has carried out the research reported in this dissertation at the Research Group on Optimization, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Bergen, Norway and at the Research Group on Logic, Information and Interaction, Department of Information Science and Media Studies, Faculty of Social Sciences, University of Bergen, Norway.

Acknowledgements

I would first like to thank my thesis advisors: Associate Professor Marija Slavkovik of the Department of Information Science and Media Studies at the University in Bergen and Professor Jan-Joachim Rückmann of the Department of Informatics at the University in Bergen in Norway. Prof.Marija Slavkovik consistently allowed this paper to be my own work but steered me in the right direction whenever she thought I needed it.

I would also like to acknowledge Prof. Jan-Joachim Rückmann as the second reader of this thesis, and I am gratefully indebted to him for his very valuable comments on this thesis and support on building overall strategy.

Finally, I must express my very profound gratitude to my parents, my husband Filip and our son Leonard for providing me with unfailing support and continuous encouragement throughout my study period and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author
Hanna Kujawska

Abstract

Preference aggregation is the process of combining multiple preferences orders into one global ranking. The top-ranked alternative is called the winner. Many aggregation methods have been considered in the literature. Some methods, like Borda count, require polynomial time, with respect to the input, to find the winner. For others, like for the Kemeny and Dodgson methods, the winners are computationally hard to compute.

We explored experimentally if machine learning algorithms can be used to predict the winner of Borda, Kemeny and Dodgson voting rules, effectively trading computational complexity for (in)accuracy.

Machine learning models were trained using two datasets: a real-world Spotify dataset and a synthetic dataset, both of profiles of size $N = 20$ alternatives and $V = 25$ voters. Four different methods for converting profiles into data sets were considered. The experimental study compared several supervised machine learning models (among others XGBoost, Gradient Boost, Support Vector Machine, Stochastic Gradient Descent (SGD) classifiers). Using less than 0.1% of all the possible profiles for the training set, models were found that predict: (i) Borda winner with the XGBoost classifier with accuracy of 100%, (ii) Kemeny winner(s) with the SGD classifier with 85% accuracy; and (iii) Dodgson winner(s) with the Gradient Boost classifier with 89% accuracy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Aggregation of individual preferences towards a collective choice has been studied originally in social choice theory (mutual decision making and voting systems) [Brandt et al. (2016), F. Rossi and Walsh (2011)]. A voting rule is a preference aggregation technique called a social choice function (SCF). Computational complexity of determining the output of a voting rule imports a concept from theoretical computer science to computational social choice [Chevaleyre et al. (2007). Recently machine learning (ML) methods are being used to automatically learn the ranking techniques, including these typically designed for human-generated preference data [Chu and Ghahramani (2005), Procaccia A.D. (2009)]. While this area, known as *learning-to-rank* (LTR), usually lack the theoretical support of the social choice, they perform excellently in empirical studies and handle large data, placing no restrictions on the data type [Volkovs (2013)].

Individual preference is a linear ordering, also called ranking or *ranked list* of alternatives, i.e. a rank-ordered list over the set of alternatives. Thus, a ranking vector is a permutation of the integers 1 through n, where each integer gives the rank position of the corresponding alternative. *Rank aggregation* is the process of using mathematical techniques, like voting rules, to create one robust *aggregated rank* from several individually ranked lists. In this thesis, the prediction of the winner in the aggregated rank appears as an interdisciplinary task linking voting theory and ML, using methods from statistics, optimization and data mining. *Preference learning* (PL) is a sub-field in ML that handles datasets with ordinal relations [Farrugia et al. (2015), Hüllermeier and Fürnkranz (2011)].

Learning the result of the aggregated preferences in ML consist of discovering the preference model that is fitted to the given preference data as the training data, to predict the final aggregated ranking from a new set of preferences [Corrente et al. (2013)]. Specifically, in this thesis, we are concerned with the *ML-based winner determination problem* (WDP) - we applied supervised ML to voting methods to predict a *winner* as a top-ranked alternative.

1.2 Motivation

We can observe many diverse problems with ordinal relations, including image retrieval [Yang Hu et al. (2008)], web pages ranking [Dwork et al. (2001), Renda and Straccia (2003)], protein selection [Ben-Bassat et al. (2018)], machine translation [Corrente et al. (2013)], game players ranking [Deng et al. (2014))] and many others.

In this thesis, we are interested in predicting the winner, since finding the winner in some voting rules is NP-hard problem [Dwork et al. (2001), Truchon (2008), Chevaleyre et al. (2007)]. Therefore the real question is: can we design mechanisms for preference aggregation that allow solving the WDP problem optimally and provably fast with traditional supervised ML techniques? Is there a time-efficient way to tackle the WDP? In this thesis, we are assuming only complete and strict orderings. This research, to the best knowledge of the author, was the first evaluation of the application of supervised ML algorithms that are provably fast (polynomial time in the size of the problem instance) and preferably successful in finding an optimal solution to the given problem instances. While computationally attractive, this approach may suffer in some cases. Thus, this work ought to significantly point also out negative results where traditional ML algorithms fail to find the desired output. Future work might compare the output of this research with works resulting from the LTR concepts or other approaches such as unsupervised ML algorithms or time-series models. Thus, this work outlined also avenues for future research.

1.3 Objectives

The purpose of this thesis is to investigate the learnability of the voting rules and evaluate the performance of ML models when predicting the top-alternative in a given aggregated preference rank.

We persuaded three fundamentally coherent research questions:

- How can we use traditional ML to predict winners of Borda, Kemeny and Dodgson voting rules?
- Which proportion of possible "data points" is sufficient, as the training dataset, for an efficient classification?
- What is the best way to represent preferences as a dataset?

We considered three different voting rules - Borda, Kemeny and Dodgson - because they belong to different complexity classes concerning winner determination problem:

- the Borda voting rule belongs to the class of score-based rules, solvable deterministically, in polynomial time [Wikipedia (2019b)] (for more details see Section 2.2.1);
- the Kemeny voting rule belongs to the class of distance-based rules and measures the distance between two linear orderings, by counting pairs of alternatives on which

they disagree, Kemeny rule returns the ranking(s) minimizing the distance; solvable in nondeterministic polynomial time [Brandt et al. (2016)] (for more details see Section 2.3.2);

- the Dodgson voting rule is computationally harder than Kemeny rule. For each preference ranking, determines the fewest number of pairwise swaps needed to make that candidate the Condorcet winner (an alternative that defeats every other alternative in the strict pairwise majority sense). The candidate(s) with the fewest swaps is (are) declared the winner(s); solvable in nondeterministic polynomial time [Brandt et al. (2016), Nurmi (2010)] (for more details see Section 2.3.3).

An important part of this thesis was to build a suitable training and testing pipeline to test different ML algorithms, with new parameters, and quickly and easily compare it to other models. A robust pipeline enabled us to find the best possible models.

We tested different ML models for four preferences' representations. We explored ten classifiers to maximize the predictive performance of the model. In particular, we compared ten ML classifiers, namely: Gradient Boosting, XGBoost, AdaBoost, Support Vector Machine (SVM), Naive Bayes, Neural Network, Decision Tree, Random Forest, Linear classifiers with Stochastic Gradient Descent (SGD) and Ridge classifier. We chose those classification algorithms based on the literature review [Li (2008). Lucchese et al. (2018)] and their availability in the *scikitlearn* library for Python. In the experimental study, we tested these models against actual real-world and large-scale synthetic collection of preferences comprising 20 alternatives and 25 voters.

1.4 Methodology

We empirically tested different supervised ML methods to assess the learning performance of three voting rules: Borda, Kemeny and Dodgson. Figure 1.1 illustrates the project framework.

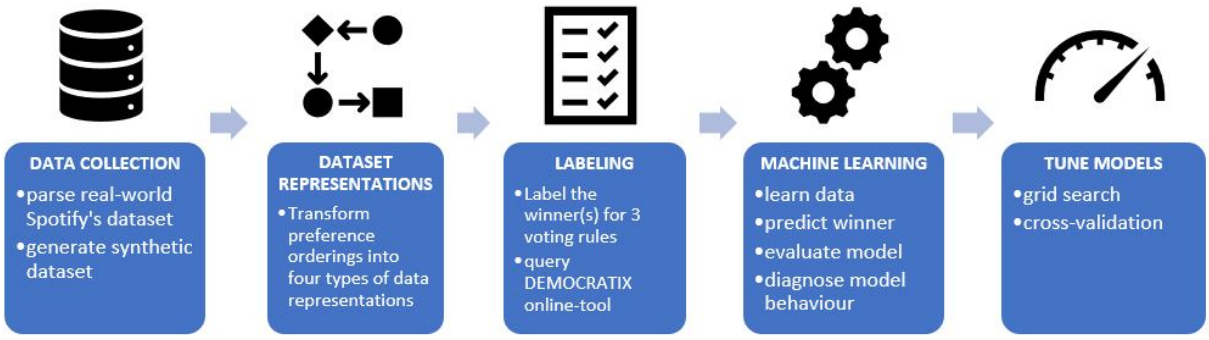


Figure 1.1: Project workflow.

Specifically, data collection included taking rankings from the following resources (step 1 in Figure 1.1):

- a real-world dataset from Spotify and
- high-dimensional synthetic dataset.

The ranking represents the set of agents having preferences over a set of alternatives. Since the ML algorithms require a certain format of the input space, we next represented the preferences in four dataset representations (step 2). Then, the studied representations have been labeled (step 3), i.e. for every preference of all agents (so-called *profile*), the mechanism outputted a winner(s). A winner becomes a label in ML model[1]. To extract the labels for NP-hard voting rules, as Kemeny and Dodgson, we applied aggregations algorithms and queried DEMOCRATIX (see Chapter 2.4), online tool evaluating preference profiles concerning various voting rules. Next, such labeled data have been introduced to the traditional ML algorithms (step 4). Finally, learning, prediction, tuning and evaluation processes respectively have been closing the entire proposed framework (steps 5 and 6).

These thesis calculations were performed in Python using related scientific libraries, such as scikit-learn, numpy, scipy, pandas, matplotlib, seaborn and selenium.

1.4.1 Success criteria

We tested three voting rules - Borda, Kemeny and Dodgson - against two datasets: Spotify with 361 samples and generated a dataset with 12360 samples. The following equation describes the total number of all possible profiles:

$$total\_num\_profiles = \frac{(|V| + |C|! - 1)!}{|C|!(|V| - 1)!}, \tag{1.1}$$

where: $|C| = 20$ is number of alternatives, and $|V| = 25$ is number of preference ranks (votes) per sample.

Using less than 0.01% of all the possible profiles for the training set, we found:

- different preference representations as a dataset that can be used to train ML classifiers,
- classification models that predict:
  - Borda winner(s) with the XGBoost classifier with accuracy of 100% accuracy,
  - Kemeny winner(s) with the SGD classifier with 85 % accuracy,
  - Dodgson winner(s) with the Gradient Boost classifier with 89% accuracy.

To choose the classification model with the best performance, we applied two different evaluation metrics: F1-score and accuracy. We didn't prioritize the algorithms' run-time. We used accuracy in particular for uniform class distribution in a high-dimensional synthetic dataset. The F1-score, as it considers false positives and false negatives (see Section 3.4.2 for more details), is especially useful for uneven class distribution, as we have in the Spotify dataset (some winners occur more often than others).

[1]Labeling algorithm included tie-breaking mechanism in case of multi-winner instances (see Chapter 4 for more details)

1.5 Literature review

The concept of *automated design of voting rules* approaches the problem of finding the winner of the voting rule. This process finds the (known) voting rule that captures specific properties/criteria if there does exist a prior set of specification of features that can be used to determine the winner. Developing a theory of automated design of voting rules by learning has been discussed by Procaccia A.D. (2009). This paper demonstrated the learnability results of *Probably Approximately Correct (PAC) learning* for two types of voting rules: scoring rules and voting trees. In score-based rules, the candidate receives points according to its position in the preferences of each voter. Voting rules represented by small trees select a candidate based on pairwise comparisons in an iterative process. Procaccia A.D. (2009) showed that the class of scoring rules is learnable efficiently, namely in polynomial time for all alternatives and voters by the PAC model. These results are the reason why we expected to get good results for the Borda method. In particular, we achieved a classification accuracy comparable to (or higher than) the results presented (Procaccia A.D. (2009)) - 100% accurate predictions. Procaccia A.D. (2009) also demonstrated the possibility of learning voting rules that can be represented as small voting trees. In general, a learning process of small voting trees requires to provide an exponential number of samples. Procaccia A.D. (2009) proved that polynomial training set suffices if the number of leaves is polynomial in the size of the set of alternatives.

Learning preference and rank aggregation is a subject of research interest in many areas, among others in the election winner determination problem in computational voting [Condorcet (1785), Borda (1781))], but also games' competitions ranking [Deng et al. (2014)], meta-search engines [Dwork et al. (2001), Renda and Straccia (2003)] or in meta-analytic bioinformatics [Patel et al. (2013)] .

Discovering the best ML classifier and tuning its hyperparameters can be a very time-consuming task. Donini et al. (2018), in their work, presented a voting problem with Random Neural Networks. In particular, the authors introduced an ensemble classifier method that exploits neural networks and voting theory and does not require neither domain knowledge nor the expertise in fine-tuning of the parameters. This work encouraged us to extend our ML models portfolio with neural network (NN) classifier: we tested a multi-layer perceptron (MLP). The tempting perspective of achieving a high level of predictive accuracy allowed us to receive in particular good results - 80% of accurate Dodgson winner(s) predictions trained in Representation 2. We found that NN requires a large amount of training data to optimize the model. Additionally, NN not converged to a single solution (are not deterministic). Maybe more complex NN having more layers and neurons could associate better each input variable with the output variable through the different weights.

Relevant techniques for learning and predicting from rank orderings might be in particular challenging, since it might involve the prediction of complex structures, such as weak or partial order relations, rather than single values [Ailon (2010), Kamishima et al. (2011)]. Complex learning tasks, such as predicting the winners of preference aggregation, differs strongly from the standard classification and regression ML tasks. In this work, **we pre-**

dicted a single value, and not complex structures, like partial preference orderings or weak rank relations. This work objective was not to establish the full rank-ordered solution of all alternatives (aggregated rank), as it takes place in Ali and Meila (2012) or LNR concept, but to predict the top alternative of the aggregated rank - the winner.

Incompleteness in the preferences of voters is prevalent in many real-world scenarios. Lang et al. (2012) considers voting problems in voting trees with incomplete preferences and weighted votes. Authors performed a sequence of pairwise comparisons in an iterative procedure, similarly to Procaccia A.D. (2009) work, where each comparison decision is made based on majority voting rule. In comparison to the work of Lang et al. (2012) or Ailon (2010) we also researched **learning and predicting preference models but for full (total) preference orderings**. Moreover, our training input space comprised explicit preference information, rather than data from diverse indirect (implicit) feedback or relative preferences. Thus, the format of input data might deviate from standard ML training data. In this research, the input is in the format of the list of nested vectors. However, ML-WDP input space format might comprise even more general types of information, such as relative preferences or different kinds of indirect feedback [Hüllermeier and Fürnkranz (2011)].

In the domain literature, we distinguish two main approaches to the learning rank aggregation problem. The first one considers *generative probabilistic model*. The generative probabilistic model approach comes from the fundamental work of Condorcet in the 18th century (Condorcet (1785)). Condorcet selects a winner by maximizing the likelihood of a candidate aggregate ranking. Condorcet's approach is based on principled Bayesian preference elicitation frameworks for collecting rank data. The statistical method such as *maximum likelihood estimation* (MLE) has been very popular in ML and computational social choice [Truchon (2008), Conitzer and Kalagnanam (2006), Conitzer and Sandholm (2005), Xia (2019)]. Recently this approach also addressing partial preferences has been developed by Conitzer and Xia (2009). Authors developed and discussed a model that assumes each partial ballot is a noisy observation of several pairwise comparisons of candidates. These findings enlighted us to also explored probabilistic classifiers - ML models enabling predicting the probability distribution over classes given input variables. We applied, for instance, the Naive Bayes classifier. We proved, based on findings, that they are highly scalable when tested against large amounts of data - 100% accurate prediction of Borda winners for all four preference representations, 81% of accurate predictions for Dodgson winners predictions trained in Representation 1.

The second main alternative is *the metric approach*. It is expressed by choosing a (quasi-) distance on the set of rankings, also called *swap distance* (described in the ensuing part of this work, see 2.3.1) and then finding a barycentric permutation, sometimes referred to as a *consensus* or *median ranking*, i.e. a ranking at minimum distance from the observed ones [Korba et al. (2017)]. This approach encompasses numerous methods, including the Kemeny aggregation. In the statistical setting, the Kemeny aggregation method can be interpreted as equivalent to the MLE approach under the noise model distinguished by Condorcet [Young (1988)], Korba et al. (2017), Ali and Meila (2012)]. In this work, we demonstrated empirically that ML algorithms could be used to predict the winner of Kemeny voting rule,

effectively trading computational complexity for 85% accuracy with the stochastic gradient descent classifier.

De Neve (2014) discussed the ideological change in the US and the economics of voting. Authors presented a voting model using independent variables, such as personal income growth rate, unemployment rate, GNP, taxes and inflation rate. In contrast to De Neve (2014) work, in this thesis, we demonstrated the possibility of learning the winner of voting rules from **no additional independent variables**, namely all data insights contributed from given preference ranks only.

1.5.1 Contributions

In the research:

- we extended the exploration of learnability of voting rules approximated by scoring-rules, such as Borda count (as partially covered in Procaccia A.D. (2009) work) by also investigating classification models for distance-based voting rules: Kemeny and Dodgson,
- we represented voting problems as ML problems by introducing four data representations that can be used to train ML classifiers,
- we converted preference dataset into PrefLib[2] format (library for preferences, popular in the computational voting community), allowing contributing data and helping extend this online resource,
- we introduced ML training and testing pipeline allowing us to quickly and easily generate performance metrics for different models and hypotheses,
- we assessed ten ML models' performance,
- we explored if ML models can be used for the optimization of intractable functions. Specifically, selecting a winner becomes an optimization problem.

1.6 Structure of the thesis

This thesis is organized as follows. In Chapter 2, we introduced the voting theory and presented the domain theoretical aspects of this work. We discussed three types of aggregating methods: Borda, Kemeny and Dodgson voting rules. Here also we introduced the practical (working) example.

In Chapter 3, we addressed the problem of ML and rank learnability. We provided an overview of several algorithms we used to predict the winner based on preference rankings.

In Chapter 4, we described the experimental study with ML methods for preference aggregation. More precisely, here we defined the ML-based winner determination problem.

[2]http://www.preflib.org

Within the experimental study, we describe the approach on how to represent preferences into the dataset in the learning process. We introduced four different preference representations and we discussed their advantages and disadvantages. Here also we analyzed the experimental result analysis concerning the evaluation metrics.

Finally, in Chapter 5, we concluded with a summary of the obtained results. We presented here the implications of major findings and also the recommendation for future research.

Chapter 2

Voting theory

2.1 Introduction

In this chapter we introduce the relevant concepts and definitions from computational voting theory. In this research we consider three voting rules: Borda's, Kemeny's and Dodgson's, described in following sections: 2.2.1, 2.3.3, 2.3.3. Borda method represents *score-based* rules, while Kemeny and Dodgson are *distance-based* rules. Borda's rule and Kemeny's rule are prominent examples of voting rules depending only on weighted pairwise majority comparisons (also known as type $C2$ functions) [Brandt et al. (2016)][1]. On the other hand, Dodgson rule is historically significant voting rule belonging to the class $C3$, meaning voting rule requiring strictly more information than a weighted directed graph, with computationally hard winner determination problems.

In this chapter, we also discussed the computational complexity of considered voting rules. In particular, Kemeny and Dodgson methods are NP-hard. Thus, here we described how we circumvented their intractability, namely by introducing DEMOCRATIX online tool for preference orderings.

Finally, we provided a working example to illustrate the differences between the results of voting rules better.

2.1.1 Notation and assumptions

Voting theory, in particular computational voting theory, which studies computational issues in voting, is a branch of *computational social choice* that recently gain probably the most attention in this field's literature [Xia (2019)]. Throughout this thesis, a *vote* is a linear order over the set of *alternatives (candidates)*. We referred to it as *preference ordering* or simply *rank*. First, each *voter (agents)* cast one vote. These votes together constitute a *profile*. Next, we apply a *voting rule* to the profile in order to distinct the winning alternative, i.e. *the winner*. We call this process **preference aggregation** for *winner determination problem*

[1] Here, pairwise comparisons can be easily represented by use of weighted directed graphs, where the weight of an edge from alternative x to alternative y is the number of voters who prefer x to y [F. Rossi and Walsh (2011)].

(WDP). Preference aggregation *model* is thus composed of three components: (i) input: data (complete preferences, not comparison), (ii) transformation (aggregation) and (iii) output: global ranking (optimal permutation), and so the winner (top alternative in consensus).

Now, formulating above assumptions we introduce following notations. Given is a finite set of alternatives (or candidates) $C = \{c_1, c_2, \ldots, c_m\}$, with $m \geq 2$, and a finite set of agents (voters) $V = \{v_1, v_2, \ldots, v_n\}$.

Definition 2.1.1. Preference ordering.
Each voter $i \in V$ is represented with her *preference relation* referred to as $\succ_i$, i.e. the order $\succ_i$ over the set C of alternatives that is [Brandt et al. (2016)]:

1. strict,
2. complete, if $c_1 \succ_i c_2$ or $c_2 \succ_i c_1$, for all $c_1 \neq c_2 \in C$,
3. transitive, if $c_1 \succ_i c_2$ and $c_2 \succ_i c_3$ then $c_1 \succ_i c_3$, for all $c_1, c_2, c_3 \in C$ and
4. antisymmetric, if $c_1 \succ_i c_2$ and $c_2 \succ_i c_1$, then $c_1 c_2$, for all $c_1, c_2 \in C$.

A voter i prefers candidate c over candidate c' if $c \succ_i c'$. The top-ranked candidate of $\succ$ is at position 1, the successor at position 2, while the last-ranked candidate is at position m.

Definition 2.1.2. Profile.
A collection of preference orders $P = (\succ_1, \ldots, \succ_n)$ for each voter $i \in V$ is called a preference *profile*. $\mathcal{L}(C)^n$ denotes the set of all such profiles for a given n. An election E is a pair $E = (C, P)$.

Definition 2.1.3. Voting rule.
A voting rule F is a mapping from an election E to a non-empty subset of the candidates $W \subseteq C$, i.e., the winners of the election [Kim (2017)].

Some distance-based voting rules, for instance Kemeny method, chooses the winning ranking that is the closest to the individual rankings based on the total number of pairwise switches. It means it requires the calculations of the distance from the given profile to the so-called *unanimous profile*.

Definition 2.1.4. Unanimous profiles.
A profile is called *unanimous* when all the preference orders in it are the same, namely $\succ_1 = \cdots = \succ_n$.

An example of a unanimous profile P for $C = \{a, b, c, d\}$ with $m = 4$ options and $n = 3$ agents (voters) is presented by 2.1:

$$P = \begin{pmatrix} b \succ_1 a \succ_1 d \succ_1 c \\ b \succ_2 a \succ_2 d \succ_2 c \\ b \succ_3 a \succ_3 d \succ_3 c \end{pmatrix} \tag{2.1}$$

Running example

For better illustration of the calculus, let's introduce now a preliminary example with one preference profile.

Example 2.1.1. *Let us consider an election E with four candidates $C = \{a; b; c; d\}$ ($|C| = 4$) and $|V| = 7$ voters. Table 4.5 presents the preference profile P of the V voters. Each row represents the preference order of a subset of voters, where the first column is the number of people who voted with this preference order, and the following column is the order of the vote. For instance, 3 voters have the preference order: $a \succ b \succ c \succ d$.*

Table 2.1: Preference profile example.

number of voters	*preference order*
3 voters	$a \succ b \succ c \succ d$
1 voter	$d \succ b \succ a \succ c$
1 voter	$d \succ c \succ a \succ b$
1 voter	$b \succ d \succ c \succ a$
1 voter	$c \succ d \succ b \succ a$

The Borda winner is b, the Kemeny winner is a and Dodgson winner is b. We discussed detailed calculus in the upcoming sections.

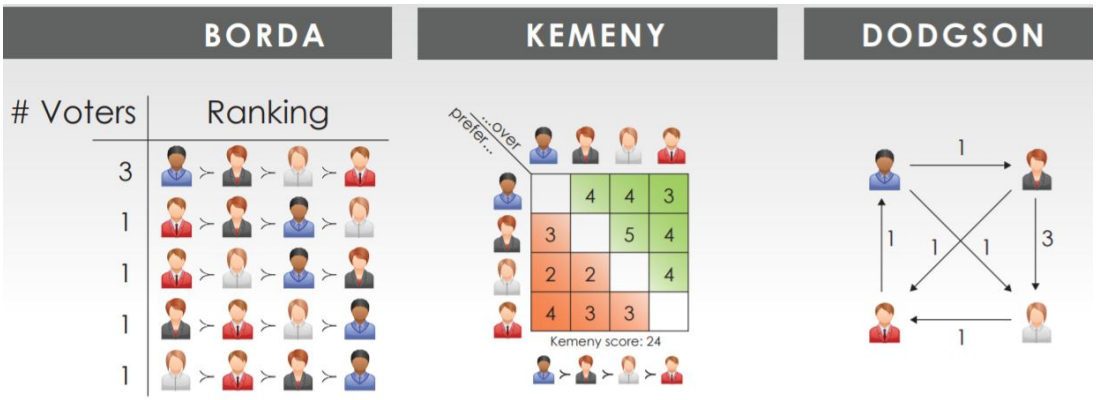


Figure 2.1: Borda, Kemeny and Dodgson winner for working example (Table 4.5).

2.1.2 Condorcet winner

This research aims to use preference aggregation to determine the winner. In the voting theory, a prevalent WDP concept consists of comparing each candidate with all the other candidates and seeing how many agents support each pair. For instance, we observe it in the examined Kemeny and Dodgson voting rules.

Definition 2.1.5. Condorcet winner.
Condorcet winner for the profile P is the alternative $c \in C$ that defeats every other alternative in the strict pairwise comparison. *Condorcet winner* is the candidate that has at least $\frac{n}{2}$ votes compared to any other candidate, where n is the number of preference orders in the profile.

Condorcet winner is unique. However, there are elections where a Condorcet winner might not exist. For instance, let us consider the set of candidates $C = \{a, b, c, d\}$ and the following profiles P and P' (see Equation 2.2):

$$P = \begin{pmatrix} a \succ_1 b \succ_1 c \succ_1 d \\ a \succ_2 b \succ_2 c \succ_2 d \\ b \succ_3 a \succ_3 d \succ_3 c \end{pmatrix} \qquad P' = \begin{pmatrix} a \succ'_1 b \succ'_1 c \succ'_1 d \\ d \succ'_2 b \succ'_2 a \succ'_2 c \\ c \succ'_3 d \succ'_3 b \succ'_3 a \end{pmatrix} \tag{2.2}$$

The Condorcet winner of the profile P is a candidate a, since for candidate a we have that: a defeats b with 2 votes ($\succ_1$ and $\succ_2$) ; a defeats c with 3 votes ($\succ_1$, $\succ_2$ and $\succ_3$); a defeats d with 3 votes ($\succ_1$, $\succ_2$ and $\succ_3$) - which in every case constitutes more than $1,5$ votes compared to any other candidate (at least $\frac{n}{2}$ votes, given $n = 3$ preference orders in profile P). Table 2.2 presents the full pairwise comparison count matrix, called also *popularity matrix*.

Table 2.2: Popularity matrix for profile P given by 2.2.

	$\succ a$	$\succ b$	$\succ c$	$\succ d$
a $\succ$	-	2	3	3
b $\succ$	1	-	3	3
c $\succ$	0	0	-	2
d $\succ$	0	0	1	-

Let us consider profile P' from equation 2.2. This profile does not have a Condorcet winner. Table 2.3 presents the full defeat matrix, where we see that no candidate receives at least $\frac{n}{2} = 1,5$ votes within pairwise comparison of all candidates.

Table 2.3: Popularity matrix for profile P' given by 2.2.

	$\succ a$	$\succ b$	$\succ c$	$\succ d$
a $\succ$	-	1 ($\succ_1$)	2 ($\succ_1$ and $\succ_2$)	1 ($\succ_1$)
b $\succ$	2 ($\succ_1$ and $\succ_3$)	-	2 ($\succ_1$ and $\succ_2$)	1 ($\succ_1$)
c $\succ$	1 ($\succ_3$)	1 ($\succ_3$)	-	2 ($\succ_1$ and $\succ_3$).
d $\succ$	2 ($\succ_2$ and $\succ_3$)	2 ($\succ_2$ and $\succ_3$)	1 ($\succ_2$)	-

The Algorithm 1 (modified from Charwat and Pfandler (2015)) presents encoding of the Condorcet rule. The first step is to determine the position of each candidate in the preference order, this contains full candidates set $\{1, \ldots, m\}$ (line 1). If $C_i \neq C_j$ holds in the preference

Require: $M = |C|, N = |V|, P$
1: $candidate(I) \leftarrow candnum(m), 1 \leq I \leq M.;$
2: $prefer(P, C_1, C_2) \leftarrow p(P, Pos_1, C_1), p(P, Pos_2, C_2), Pos_1 \leq Pos_2$
3: $preferCount(C_1, C_2) \leftarrow candidate(C_1; C_2), C_1 \neq C_2$
4: $noWinner(C) \leftarrow preferCount(C, \_, N), voternum(V), N \cdot 2 \leq V.$
5: $winner(C) \leftarrow candidate(C)$, not $noWinner(C)$
6: $anyWinner \leftarrow winner(\_)$
7: $\leftarrow$ not $anyWinner$
8: **return** winner(C)

Algorithm 1: Condorcet encoder

relation, $prefer(P, C_1, C_2)$ has been determined (line 2). This corresponds to this element of popularity matrix which compares the pair (C_1, C_2) in the profile P. The value of function $prf(c_1, c_2)$ is computed in the line 3. Next, the rule that candidate c_i cannot be a Condorcet winner has been encoded, if there is some other candidate c_j, such that $prf(c_i, c_j) \leq \frac{n}{2}$ (line 4). Here, $noWinner(C)$ is obtained if such a counterexample can be found for candidate C_i. The different cases, if no counterexample exists, then the Condorcet winner has been found (line 5). The final two steps ensure that no answer set is returned if no Condorcet winner exists (lines 6 and 7).

Table 2.4: Pairwise comparison matrix for profile P given in the Table 4.5

	$\succ a$	$\succ b$	$\succ c$	$\succ d$
a$\succ$	-	4	4	3
b$\succ$	3	-	5	4
c$\succ$	3	2	-	4
d$\succ$	4	3	3	-

Presented case in Table 4.5 has no Condercet winner, since none candidate collected at least $\frac{n}{2} = 3,5$ votes in pairwise comparison with all other candidates. Table 2.4 presents in details pairwise comparison matrix.

2.2 Score-based voting rules

Scoring rules form this class of voting rules that award points to alternatives according to their position in the preferences of the voters [Procaccia A.D. (2009)]. Well-known examples of scoring rules are Borda count, plurality and anti-plurality. *The plurality rule* selects the candidate who was ranked first by the most voters. Thus the vector of scoring weights is described by $\alpha = (1, 0, 0, \ldots, 0)$. Plurality is the voting rule commonly used in real-world elections. The main drawback of this rule is that it completely ignores all the information

given by the voter preferences besides for the top ranking. *Anti-plurality rule* is characterised by a vector of scoring weights is described by $\alpha = (1, 1, 1, \ldots, 1, 0)$.

2.2.1 Borda method

The Borda method, also known as **Borda count**, is *a positional scoring rule*. Each ranked candidate in C is associated with a score that is obtained by the position that candidate has in the preference orders of the profile. A candidate $c \in C$ receives $(m-1)$ points from each $\succ_i$ where it is top-ranked, $(m-2)$ points from all $\succ_i$ where it is second-ranked, and so on, receiving 0 points from the last ranked alternative. Thus formally, for a fixed number of candidates *m* Borda count is expressed by non-negative score vector (also known as *vector of scoring weights*) given by **??**:

$$\alpha = (m-1, m-2, \ldots, 1, 0). \tag{2.3}$$

Definition 2.2.1. Borda winner.
Borda winner for the profile P is the alternative $c \in C$ that has a maximal sum of points that he receives from all voters.

Borda winner is also called the winner with the highest *Borda score*. While it sometimes elects broadly acceptable candidates rather than those preferred by the majority, the Borda count is often described as a c*onsensus-based electoral system*, rather than a majoritarian one [Wikipedia (2019a)].

Borda's rule is a multi-winner method (also called the irresolute method), meaning more than one candidate can receive maximal Borda score for one profile P.

The Algorithm 2 (modified from Charwat and Pfandler (2015)) presents program solver for the Borda's rule. The first step is to determine the candidate relation, namely position in the rank. This relation contains all elements of candidates set $\{1, ..., m\}$ (line 1). The second step is to the score from set α each candidate in every preference relation according to her position (line 2). The next step sums the scores of overall candidate votes (line 3). Finally, the winner(s) is determined (line 4 and 5).

Require: $M = |C|, N = |V|, P$
1: $candidate(I) \leftarrow candnum(m), 1 \leq I \leq M.$;
2: $posScore(P, C, S) \leftarrow p(P, Pos, C), candnum(M), S := M{-}Pos$
3: $score(C, N) \leftarrow candidate(C), N{:} = \sum^{S} posScore(\_, C, S).$
4: $maxScore(M) \leftarrow M := max_S score(\_, S).$
5: $winner(C) \leftarrow candidate(C), score(C, M), maxScore(M)$
6: **return** winner(C)

Algorithm 2: Borda count encoder

In the running example given in Table 4.5 Borda count is represented by following score vector (*vector of scoring weights*): $\alpha = \{3; 2; 1; 0\}$. Borda scores $B(i)$, where $i \in C$, are

computed as follows:

$$B(i) = \begin{cases} B(a) = (3 \cdot 3) + (1 \cdot 1) + (1 \cdot 1) + (1 \cdot 0) + (1 \cdot 0) = 11 \\ \mathbf{B(b)} = (3 \cdot 2) + (1 \cdot 2) + (1 \cdot 0) + (1 \cdot 3) + (1 \cdot 1) = \mathbf{12} \\ B(c) = (3 \cdot 1) + (1 \cdot 0) + (1 \cdot 2) + (1 \cdot 1) + (1 \cdot 3) = 9 \\ B(d) = (3 \cdot 0) + (1 \cdot 3) + (1 \cdot 3) + (1 \cdot 2) + (1 \cdot 2) = 10 \end{cases} \tag{2.4}$$

The Borda winner is the candidate who collects the highest total number of points (highest cumulative score). Here the Borda winner is candidate ***b*** since this alternative received the highest Borda score from all agents (voters), which is equal to 12. Candidate b, therefore, becomes the desired output of this preference profile (also called label or ground truth) for supervised ML task.

2.3 Distance-based voting rules

We considered two distance-based rules: Kemeny and Dodgson. Those rules may be interpreted as minimizing a distance to *consensus* (minimization of disagreement). Both Kemeny and Dodgson methods use the same metric[2] to *compute dissimilarity or distance* between two rankings: *Kendall's tau distance* (see Subsection 2.3.1). In Kemeny and Dodgson methods, if Condorcet winner exists, it becomes the winner.

2.3.1 Swap distance / Kendall's tau metric

For two preference orders $\succ_i$ and $\succ_j$ we can define the *swap* distance, also called *Kendall distance* or *Kemeny distance*, as the minimal number of pairwise swaps, i.e. transposition of discordant pair, required to make the two linear orderings the same.

Definition 2.3.1. Discordant pair.
Discordant pair is adjacent pair of alternatives in disorder-rank that at least one alternative does not match the pattern-rank [Teknomo (2018)].

Pattern-rank has order or sequences that disorder-rank wants to achieve. Pattern-rank serves as an example, guide or goal that the disorder-rank will reach after several transformations or operations. Distance for ordinal variables measures the *minimum* number of operation steps to make from disorder-rank the pattern-rank. One can think of this metric as the number of flips one needs to perform on a ranking to turn it into the other, and it is sometimes referred to as *bubble-sort distance*[3].

Thus, the algorithm to compute Kendall distance metric is to count the *minimum* number of operation interchange (or transposition) of discordant pair:

[2] The most common practices among others not covered in this work are: normalized rank transformation, Spearman distance, Footrule distance, Cayley distance, Ulam distance and Minkowski distance.

[3] Closely-related Tau correlation coefficient is implemented in Scipy as scipy.stats.kendalltau.

1. choose **adjacent** pair on disorder rank that at least one of alternatives does not match to the corresponding alternative in the pattern-rank (i.e. discordant pair);

2. interchange the order of the pair.

Definition 2.3.2. Kendall tau distance (swap distance).
The swap distance d between two orders $\succ$ and $\succ'$ over a set of candidates C with $c_i, c_j \in C$ is defined as:

$$d(\succ, \succ') = |\{(c_i, c_j) : (c_i \succ c_j \text{ and } c_j \succ' c_i) \text{ or } (c_j \succ c_i \text{ and } c_i \succ' c_j)\}|. \tag{2.5}$$

We have that $d(\succ, \succ') = 0$ if and only if $\succ_1$ is the same order as $\succ'$, i.e. for *unanimous profile* given for instance by 2.1. Otherwise, swap distance is the number of pairwise comparisons on which two preference orders differ.

For instance, swap distance $d(\succ_1, \succ_2)$ between two following orders:

(1) $a \succ_1 b \succ_1 c \succ_1 d$ as pattern-vector and

(2) $b \succ_2 a \succ_2 d \succ_2 c$ as disordered-vector

is 2, we write $d(\succ_1, \succ_2) = 2$, because above two preference orders represented as pairwise comparisons are as follows:

(1) $\{a \succ_1 b, a \succ_1 c, a \succ_1 d, b \succ_1 c, b \succ_1 d, c \succ_1 d\}$

(2) $\{b \succ_2 a, a \succ_2 c, a \succ_2 d, b \succ_2 c, b \succ_2 d, d \succ_2 c\}$.

The two orders differ on 2 discordant pairs (pairwise comparisons), which are: $a \succ_1 b$ vs. $b \succ_2 a$ and $c \succ_1 d$ vs. $d \succ_2 c$, but no other pairs.
We define swap distance D between two profiles of preferences, such that $P = (\succ_1, ..., \succ_n)$ is a list of preference orders and $P' = (\succ'_1, ..., \succ'_n)$ is another list of preference orders, as follows:

$$D(P, P') = d_s(\succ_1, \succ'_1) + \cdots + d_s(\succ_n, \succ'_n). \tag{2.6}$$

For example, we have that swap distance between profiles given by 2.2 is 9, since $D(P, P') = 0 + 4 + 5 = 9$.

2.3.2 Kemeny rule

The Kemeny method is a *distance-based rule*. The collective preference order or a profile P is the order $\succ$ for which the sum of swap distances from $\succ$ to each $\succ_i \in P$ is minimal. This collective preference order is called **Kemeny ranking**, also known as *Kemeny consensus*, with respect to election E. In order to calculate Kemmeny winner we use *popularity matrix*. Pairwise comparison counts may be expressed by table representing sequence of choices such that most popular choices are in the top left corner of the matrix and the least popular in bottom right (e.g. see Figure 2.1 b). **Kemeny winners** are the top-ranked alternatives in a Kemeny consensus. Computing Kemeny consensus is NP-hard over 4 candidates[Young

(1988), Lang et al. (2012),Dwork et al. (2001)]. Formally, we can define the Kemeny rule as follows. Let $\mathcal{C}$ be the set of all total, strict and antisymmetric orders that can be constructed over a set of alternatives C.

$$\text{KemenyRanking}(P) = \arg\min_{\succ \in \mathcal{C}} \sum_{\succ_i \in P} d(\succ, \succ_i) \tag{2.7}$$

Thus, given a profile P of n preference orders over the set of candidates C, the Kemeny winner is:

$$\text{KemenyWinner}(P) = \begin{cases} \text{the Condorcet winner, if } P \text{ has a Condorcet winner} \\ \text{else, the top ranked alternative in profile } P', \\ \text{such that } D_s(P, P') \text{ is minimal} \\ \text{and } P' \text{ is a unanimous profile of } n \text{ preference orders.} \end{cases} \tag{2.8}$$

Therefore, to calculate the Kemeny winner we first need to consider the set X of all complete, strict, antisymmetric and transitive orders that can be constructed over C. For $C = \{a, b, c, d\}$ we have that X is given by Equation 2.9:

$$X = \left\{ \begin{array}{l} a \succ b \succ c \succ d \\ a \succ b \succ d \succ c \\ a \succ c \succ b \succ d \\ a \succ c \succ d \succ b \\ a \succ d \succ b \succ c \\ a \succ d \succ c \succ b \\ b \succ a \succ c \succ d \\ b \succ a \succ d \succ c \\ b \succ c \succ a \succ d \\ b \succ c \succ d \succ a \\ b \succ d \succ a \succ c \\ b \succ d \succ c \succ a \\ c \succ b \succ a \succ d \\ c \succ b \succ d \succ a \\ c \succ a \succ b \succ d \\ c \succ a \succ d \succ b \\ c \succ d \succ b \succ a \\ c \succ d \succ a \succ b \\ d \succ b \succ c \succ a \\ d \succ b \succ a \succ c \\ d \succ c \succ b \succ a \\ d \succ c \succ a \succ b \\ d \succ a \succ b \succ c \\ d \succ a \succ c \succ b \end{array} \right\} \tag{2.9}$$

We then need to consider the set U of all unanimous profiles for n agents constructed from orders in X. For $C = \{a, b, c, d\}$ there are 24 such profiles. Finally, we calculate the swap distance D from P to each of the profiles of U and find the profile $P' \in U$ for which $D(P, P')$ is minimal.

Require: $M = |C|, N = |V|, P$

1: $candidate(I) \leftarrow candnum(m), 1 \leq I \leq M.;$
2: $wrank(P, C_2, C_1) \leftarrow p(P, Pos_1, C_1), p(P, Pos_2, C_2), Pos_1 \leq Pos_2$
3: $wrankCount(C_2, C_21 \leftarrow candidate(C_1; C_2), C_1 \neq C_2$
4: $gpref(Pos, C) \leftarrow candidate(Pos; C), notnpref(Pos, C)..$
5: $npref(Pos, C) \leftarrow domain(Pos; C), notgpref(Pos, C)$
6: $\leftarrow gpref(Pos, C_1), gpref(Pos, C_2), C_1 \neq C_2$
7: $\leftarrow gpref(Pos_1, C), gpref(Pos_2, C), Pos_1 \neq Pos_2$
8: $occupied(Pos) \leftarrow gpref(Pos, \_)$
9: $\leftarrow domain(Pos)$, not $occupied(Pos)$
10: $rank(C_1, C_2) \leftarrow gpref(Pos_1, C_1), gpref(Pos_2, C_2), Pos_1 \leq Pos_2$
11: $gwrankC(C_1, C_2, N) \leftarrow rank(C_1, C_2), wrankC(C_1, C_2)$
12: $gwrankC(\_, \_)$
13: $winner(C) \leftarrow gpref(1, C)$
14: **return** winner(C)

Algorithm 3: Kemeny encoder

In Step 1 of Algorithm 3 (modified from Charwat and Pfandler (2015)), we obtain relation, which is used to identify candidates and their positions in preference relations (line 1). Next, we determine for each preference order the candidates C_2 that are worse-ranked than C_1 (line 2). After that, the algorithm sums the overall number of voters that do not prefer $C_2 \succ C_1$ (line 4). We guessed the preference relation by assigning to each candidate exactly one position (line 4-9). Notably, we the Kemeny relation rank has been found whenever $C_1 \succ C_2$ in the guessed preference relation (line 10). Next, we computed the number of votes that disagree on $C_1 \succ C_2$ (line 11). Then, we summed up all N, and in $gwrankC$ *Kemeny score* has been computed and (*Kemeny consensus*) determined (line 12). Finally, in the last step, we returned *Kemeny winner*, meaning the candidate ranked first in a Kemeny consensus (line 13).

For a given running example presented in Table 4.5, which has no Condorcet winner, we construct the popularity matrix to determine Kemeny winner. Kemeny consensus ranking $K = \sum_{i \in C} K(i)$, is computed based on (2.5) as follows:

$$K(i) = \begin{cases} \mathbf{K(a)} = 0 \cdot (a \succ a) + 4 \cdot (a \succ b) + 4 \cdot (a \succ c) + 3 \cdot (a \succ d) = \mathbf{11} \\ K(b) = 0 \cdot (b \succ b) + 5 \cdot (b \succ c) + 4 \cdot (b \succ d) = 9 \\ K(c) = 0 \cdot (c \succ c) + 4 \cdot (c \succ d) = 4 \\ K(d) = 0 \cdot (d \succ d) = 0 \end{cases} \tag{2.10}$$

Kemeny score for following preference ranking $a \succ b \succ c \succ d$, is equal to 24 $(11+9+4)$ since this is the sum of all the counts in an upper-right triangle (see Fig. **??** b). Now, we were calculating all the ranking scores, i.e. for all 24 possible ranking sequences, given by X (2.9). Based on the best Kemeny score, we determine the consensus ranking, which in this example is $a \succ b \succ c \succ d$ with the Kemeny score equal to 24. Therefore top-ranked candidate a is the Kemeny winner.

2.3.3 Dodgson rule

Dodgson voting rule elects Condorcet winner, as very favorable property, by making swaps of adjacent candidates in the votes such that there is a Condorcet winner. For this rule, the winner is the candidate that needs the minimum number of swaps.

Given a profile P of n preference orders over the set of candidates C, the Dodgson winner is:

$$\text{DodgsonyWinner}(P) = \begin{cases} \text{the Condorcet winner, if } P \text{ has a Condorcet winner} \\ \text{else, the top-ranked alternative in profile } P', \\ \text{such that } D_s(P, P') \text{ is minimal} \\ \text{and } P' \text{ is a profile of } n \text{ preference orders} \\ \text{that has a Condorcet winner.} \end{cases} \tag{2.11}$$

Here, first, we construct the set X of all complete, strict, antisymmetric and transitive orders that can be built over C. For $C = \{a, b, c, d\}$, we have the set X composed of 24 preference orders given by Equation 2.9. Next, we construct the set K of all profiles of n agents been built from orders in X that have a Condorcet winner[4]. Finally, we calculate the swap distance D from P to each of the profiles of K and find the profile P' for which $D(P; P')$ is minimal.

In the first two steps of Algorithm 4, we determine voters and the domain (positions and candidates). In steps 3 and 4, we guess the shifts in the votes. For a voter V the candidate at position Pos_1 will be shifted to Pos_2. One shift consists of $Pos_1 - Pos_2$ elementary exchanges, namely swaps, of adjacent candidates. At most one shift per voter (steps 5-6) to a better position (step 7) is allowed. The preference profile is now recomputed: the candidate C_1 is moved from Pos_1 to Pos_2 (step 8) and each candidate originally at Pos with $Pos_2 \leq Pos < Pos_1$ is shifted by one position downwards (step 9). In the newly computed votes nv, the shifted candidates are assigned to their new positions (step 10) and the remaining positions are filled with the respective candidates of the original vote (steps 11 and 12). Steps from 13 to 18 encode the computation of the Condorcet winner, similarly to Algorithm 1. Finally, step 19 minimizes the number of swaps.

[4]The brute-force way to create K is to construct all combinations of size n with repetitions from the elements of X and then eliminate those profiles that do not have a Condorcet winner.

Require: $M = |C|, N = |V|, P4$

1: $voter(I) \leftarrow voternum(N), 1 \leq I \leq N$
2: $candnum(I) \leftarrow candnum(M)$, $1 \leq I \leq M$
3: $shift(V, Pos_1, Pos_2) \leftarrow voter(V), domain(Pos_1; Pos_2)$, not $noshift(V, Pos_1, Pos_2)$
4: $noshift(V, Pos_1, Pos_2) \leftarrow voter(V), domain(Pos_1; Pos_2)$, not $shift(V, Pos_1, Pos_2)$
5: $\leftarrow shift(V, Pos_1, \_), shift(V, Pos'_1, \_), Pos1 \neq Pos'_1$
6: $\leftarrow shift(V, \_, Pos_2)$, $shift(V, \_, Pos'_2), Pos_2 \neq Pos'_2$
7: $\leftarrow shift(V, Pos_1, Pos_2), Pos_1 \leq Pos_2$
8: $sv(V, Pos_2, C_1) \leftarrow shift(V, Pos_1, Pos_2), v(V, Pos_1, C_1)$
9: $sv(V, PosShift, C) \leftarrow shift(V, Pos_1, Pos_2), v(V, Pos, C), Pos2 \leq Pos, Pos \leq Pos_1,$
$PosShift := Pos + 1$
10: $nv(V, PosShift, C) \leftarrow sv(V, PosShift, C)$
11: $occupied(V, Pos) \leftarrow sv(V, Pos, \_)$
12: $nv(V, Pos, C_1) \leftarrow v(V, Pos, C_1)$, not $occupied(V, Pos)$
13: $prefer(V, C_1, C_2) \leftarrow nv(V, Pos_1, C_1), nv(V, Pos_2, C_2), Pos_1 < Pos2$
14: $preferCnt(C_1, C_2, N) \leftarrow domain(C_1; C_2), C_1 \neq C_2, N := countprefer(\_, C_1, C_2)$
15: $noWinner(C) \leftarrow preferCnt(C, \_, N), voternum(V)$, $N \cdot 2 \leq V$
16: $winner(C) \leftarrow domain(C)$, not $noWinner(C)$
17: $anyWinner \leftarrow winner(\_)$
18: $\leftarrow notanyWinner$
19: **return** winner(C)

Algorithm 4: Dodgson rule encoder

Since in the given running example, presented in Table 4.5, is no Condorcet winner, we built a weighted directed graph to represent pairwise comparisons. Here, the weight of an edge from alternative a to the alternative b is the number of voters who prefer a over b. In the running example, we consider only two candidates: b and c, since only those collected at least $\frac{n}{2} = 3,5$ votes in pairwise comparison with all other candidates.

$$D(i) = \begin{cases} D(a) = 3 \text{ with one swap with } d, \\ \textbf{D(b)} = 5 \text{ with } \textbf{one} \text{ swap with } a, \\ D(c) = 5 \text{ with two swaps with } a \text{ and } b, \\ D(d) = 3 \text{ with two swaps with } b \text{ and } c, \end{cases} \tag{2.12}$$

Dodgson winner elects Condorcet winner as the candidate that needs the minimum number of swaps. Candidate b needs one swap of adjacent candidates in the votes, i.e. interchange with the candidate a, such that there is a Condorcet winner (collects at least four votes). Therefore the candidate b is the Dodgson winner in the running example.

2.4 DEMOCRATIX - label extraction tool

Voting rules determine a winner, namely top-ranked alternative in the aggregated rank. In ML, the winner is the label, more precisely desired output that classification models want to predict. The ongoing research phase was enlightened to search for available tools that would allow evaluating preference profiles, particularly for Kemeny and Dodgson voting rules, since they are NP-hard over four candidates [Dwork et al. (2001), Brandt et al. (2016)]. A valuable open-sourced and web-based tool called *DEMOCRATIX*[5] supports strict-order, complete preference profiles for following voting rules: Plurality, Veto, Simpson, Borda, Condorcet, Kemeny, Dodgson, Sluter, Bucklin, Copeland, Young and Black.

This approach ensured an automatic label extraction technique by domain-specific web crawling. It means we used this tool in the research for labeling Kemeny and Dodgson winner. Thus, to determine a ground truth (label): first, we connected to web data; next, we run the script sending selected profiles' database in required PrefLib format; and finally, parsed received data and saved it locally. The full DEMOCRATIX crawling framework is presented and described more detailed in Figure 4.2 in Chapter 4.

It is important to mention also the main drawback of using *DEMOCRATIX* tool for determining the winner. In some cases, for instance, when the solver of the tool has been interrupted or killed by a signal, it was necessary to rerun the web-crawling script, which happened in 2472 cases for Kemeny winner determination and 1842 cases for Dodgson winner determination. That might significantly increase the processing time for larger instances.

2.5 Tie-breaking rules

Typically, voting rules are accompanied by tie-breaking mechanisms. The subject literature includes, but is not limited to, four possible tie-breaking approaches [Obraztsova and Elkind. (2011); N. Mattei and Walsh (2014)]:

1. Use a fixed ordering of the alternatives (or a designated voter) to break all ties.
2. Use a randomized mechanism to break all ties.
3. Deal with set-valued outcomes directly.
4. Ignore or suppress the issue (assume no ties exist).

All the above approaches have their pros and cons [Aziz et al. (2013)]. Voting rules such as Borda, Kemeny and Dodgson are irresolute, meaning that in some settings might determine multi-winner result.

In this work, we applied explicitly *lexicographic tie-breaking rule* in addition to a voting rule to enforce that the size of the winner set W is equal to $k = 1$. In the *lexicographic tie-breaking rule*, ties are broken using a priority ordering on the candidates C, meaning if there

[5] http://democratix.dbai.tuwien.ac.at/examples/index.php

is a set of tied alternatives, the winner is selected as a candidate who is (alphabetically) first in the sequence, according to a fixed priority ordering. For instance, given the tied set of the multi-winner result: $\{a, b, c\}$ the lexicographic tie-breaking mechanism produces the output: $\{a\}$.

2.6 Conclusions

In this section, we presented the preliminaries and definitions from computational voting theory. We introduced three votings (rank aggregation) methods: Borda, Kemeny and Dodgson. We selected them based on the increasing level of computational complexity, i.e. Borda method is scored-based rule solvable in polynomial time, Kemeny and Dodgson are distance-based methods solvable in non-deterministic polynomial time. To label the winner in aggregated rank we used an available online tool called DEMOCRATIX. Finally, we presented a running example to better illustrate the calculus for all three considered voting rules.

Chapter 3

Machine learning techniques

In this chapter, we established notations for ML models. We presented an overview of popular supervised ML techniques for classification. We discussed also error analysis and model behavior diagnosis tools.

3.1 Introduction

ML enables computers' programs (algorithms) to learn from data alone instead of being explicitly programmed. Traditional ML field is composed of the following tree areas [Xia (2019)] (see Figure 3.1):

- supervised learning,
- unsupervised learning[1],
- reinforcement learning[2].

Next to that is Deep Learning field[3].

[1]Unsupervised learning goal is to find patterns and correlations in data. Unsupervised learning enables us to identify problems with little or no knowledge of what the results should look like since there are no given labels. The goal of unsupervised learning is *clustering* the data based on relationships/correlations among the variables in the data or find their hidden structure. It is possible to extract the structure from the dataset where there is not known the explicit effect of the variables. For instance, given a collection of 1,000,000 different stars pictures, clustering might derive the data structure and find a way to automatically group these stars into collections that are related or associated by different variables, such as stars' age, location or brightness. With unsupervised learning, there is no feedback based on the prediction results.

[2]The goal of reinforcement learning is to find the best actions to take to achieve goals or maximize rewards. The learning model is composed of four main components: decision process, reward system, learning series of steps, expecting an agent to learn by themselves by punishment and reward actions how to react in the environment.

[3] There is also semi-supervised learning. Semi-supervised learning is localized between supervised and unsupervised learning methods. The goal of semi-supervised learning is *classification*, and here, however, the input dataset contains labeled and unlabelled data. In a classification task, there must be labeled data, since classification is an ML method for identifying a new observation based on data training. In semi-supervised learning, there exists a small amount of labeled data and a large amount of unlabelled data. Thus, to enable the classification, the unlabelled data are being trained to learn the existing class.

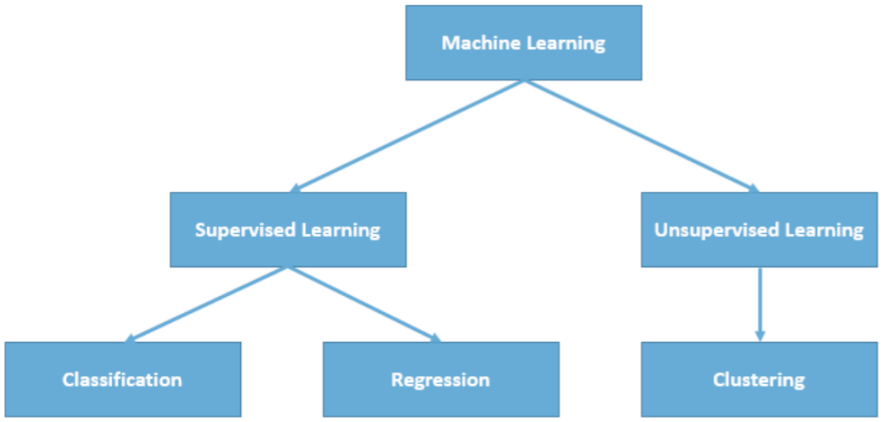


Figure 3.1: Traditional ML fields.

Supervised learning is the task of learning a function (hypothesis) that maps input data to output data based on given examples of input-to-output pairs. **Classification** occurs when the algorithm predicts a discrete result - the output is a category. In our case, we predicted the outcome category (winner, i.e. one of 20 possible alternatives - categories), so we are only interested in the supervised learning landscape of ML (see Figure 3.1).

The classification process is composed of two phases: first is **learning**, also referred to as *training*, and second called **predicting** (also known as *testing*). For model-tuning reasons one might introduce also **validation** phase, which is included in the learning phase. Figure 3.2 presents the road-map of processing data in the learning and predicting phase.

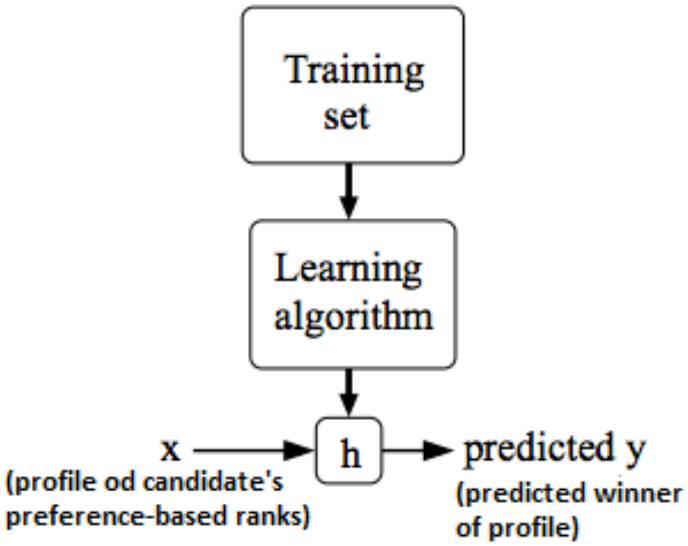


Figure 3.2: Model representation for supervised learning. Modified from Ng (2019).

We established notion of $x \in X$ to describe the *input* variables, also called *input features*, and $y \in Y$ to denote the *label* or *target variable* or *ground truth* that the ML model is trying to predict as the desired *output*. Thus, classification model learn a function $h(x)$, called a *hypothesis*, assigning the label to the given input.

Definition 3.1.1. Data point.
A pair (x, y) is called a *training example* or *data point* or *sample*. A data point is a vector of values, where each value is associated with a *feature*.

Features are used to build a factorized representation of an entity.

3.1.1 Voting as a supervised classification problem

To model voting rule as function $h(x) : X \rightarrow Y$ to be learned by supervised ML model, we took a profile to be a data point $x(i)$, all preference profiles as input space X, and the set of alternatives as the set of labels (or classes) Y to be associated with a profile (see Figure 3.3). While we have m-alternatives, this is m-class classification problem ($|m| = 20$). However, both the Borda and the Kemeny methods are irresolute voting rules, namely there can be more than one candidate tied as winners for one profile. Typically, a voting rule is accompanied with a tie-breaking mechanisms to account for such situations. We here address ties' problem by considering lexicographic tie-breaking rule for profiles that contain tied winners.

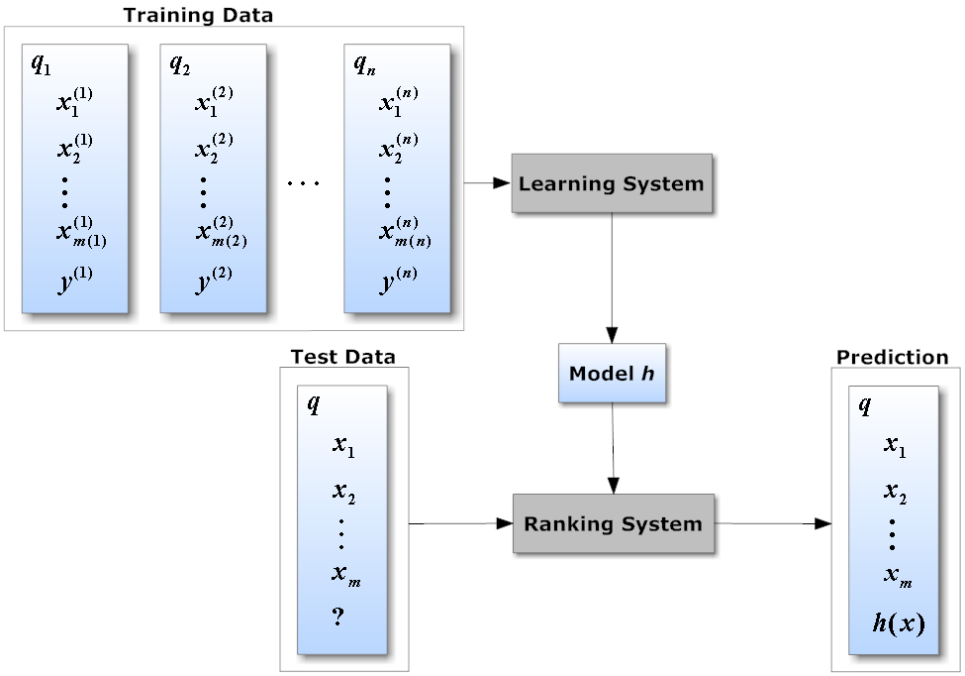


Figure 3.3: Learning to rank concept [miro.medium.com (2016)].

Therefore, we considered the following learning problem:
Given:

- a set of labels $C = \{C_i | i = 1, ..., c\}$
- a set of examples (profiles) $P = \{p_k | k = 1, ..., m\}$
- for each training example p_k: a set of voters $V = \{v_j | j = 1, ..., n\}$ with individually ordered preferences set $C = \{C_i | i = 1, ..., c\}$ ($p_k \in V \times C$).

Find: a function that selects the top-ranked alternative (label) $\{C_i|i = 1, ..., c\}$ of aggregated rank for any given example.

Classification here consist on assigning to each example a single class label C_i. In *multi-label classification* each training example p_k is associated with a subset $S_k \in C$ of possible labels. As pointed out before, we predicted a top-ranked label(alternative) in aggregated ranking (total order) of the labels. Thus, we assume that for each instance, there exists a total order of the labels, namely they form a transitive and asymmetric relation.

3.2 ML algorithms for classification

We now introduce the ML methods we used in our experiments. These are: XGBoost, Linear Support Vector Machines (SVM), Multilayer Perceptron, regularized linear classifiers with stochastic gradient descent (SGD)and Ridge classifier. These approaches were chosen through a process of trial and error experimentation starting from a large pool of all available ML classification methods in the *scikitlearn* library[4].

3.2.1 Generalized linear models

Generalized linear models (GLM) belong to popular predicting techniques as they are easy to implement. If we assume linearity between input variables and the output variable, these models generate robust predictions. A significant advantage of GLM models is an easy interpretation of the fitted coefficients. The downside of these models is that they not always fit the problem and can be too simple if there is no linearity between input and output variables. Also, if the input variables are highly correlated, the model performance can be quite poor. Logistic Regression is an example of classification models from the GLM group for a binary output variable problem (see Figure 3.4).

3.2.2 Support Vector Machines (SVM)

Support Vector Machines (SVM). Conceptually, a data point, for which all feature values are real numbers, can be seen as a point in hyperspace. Binary classification would then be the problem of finding a hyper-plane that separates the points from one class from the points of the other class. SVM's find this hyper-plane by considering the two closest data points from each class. Since not all datasets are separable with a hyper-plane, SVM's use *kernels* to transform the dataset into one that can be split by a hyperplane (Figure 3.5). We might apply both linear and nonlinear kernels. Here we used an SVM with a linear kernel. The efficiency of an ML method can be improved by tuning the so-called hyper-parameters. The SVM we used provides one hyper-parameter to tune: cost of the miss-classification of the data on the training process (C).

[4]https://scikit-learn.org/stable/

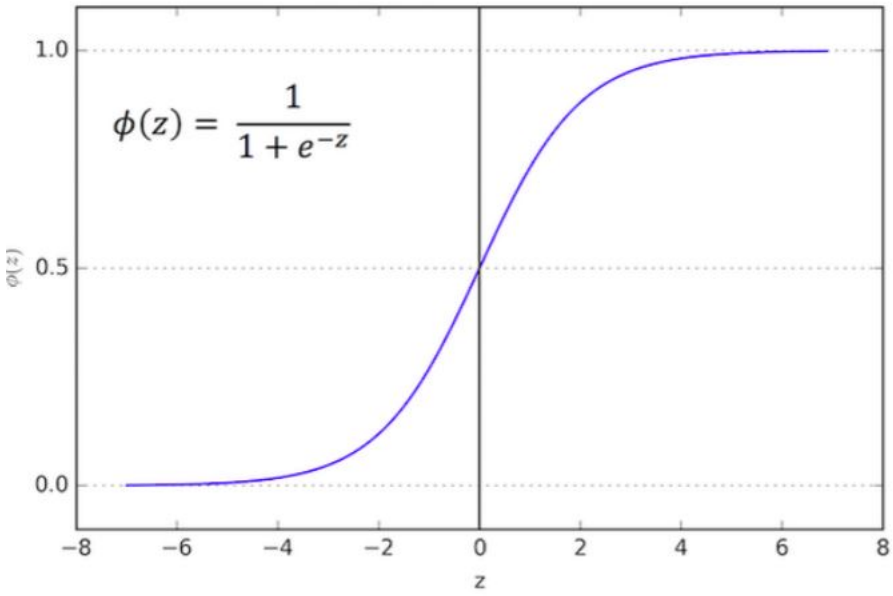


Figure 3.4: Logistic function [thefactmachine.com (2016)].

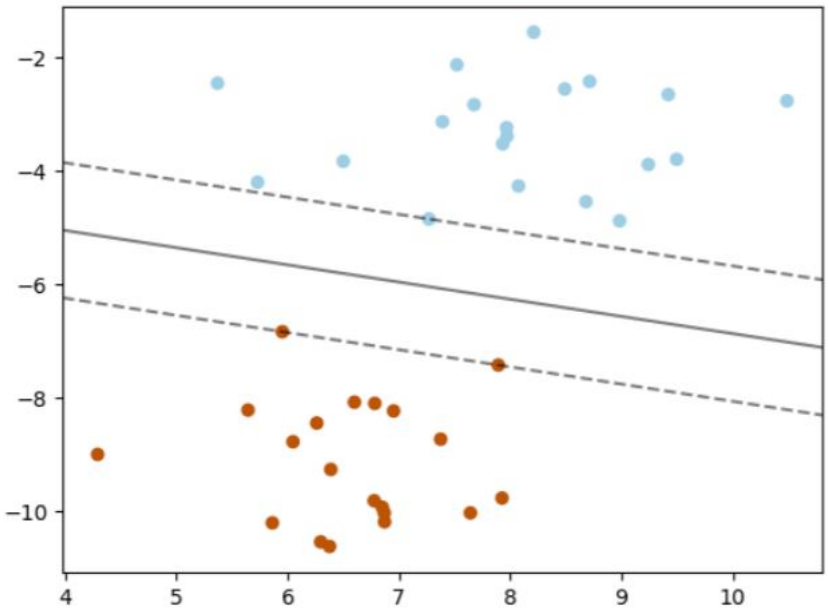


Figure 3.5: SVM's hyperplane for classification [scikit learn.org (2013a)].

3.2.3 Gradient Boosted Decision Trees (GB)

Gradient Boosted Decision Trees (GB) are among the most powerful and widely used models for supervised learning. Predicting a label can be done with a decision tree built using the training data (see Figure 3.6). To increase the prediction performance, GB builds an ensemble of decision trees serially: each new tree is built to correct the mistakes of the previous one. GB's offers a wide range of hyper-parameters that can be tuned to improve prediction performance, among else the number of trees ($n\_estimators$) and $learing\_rate$, which controls the degree to which each tree is allowed to correct the mistakes of the pre-

vious trees.

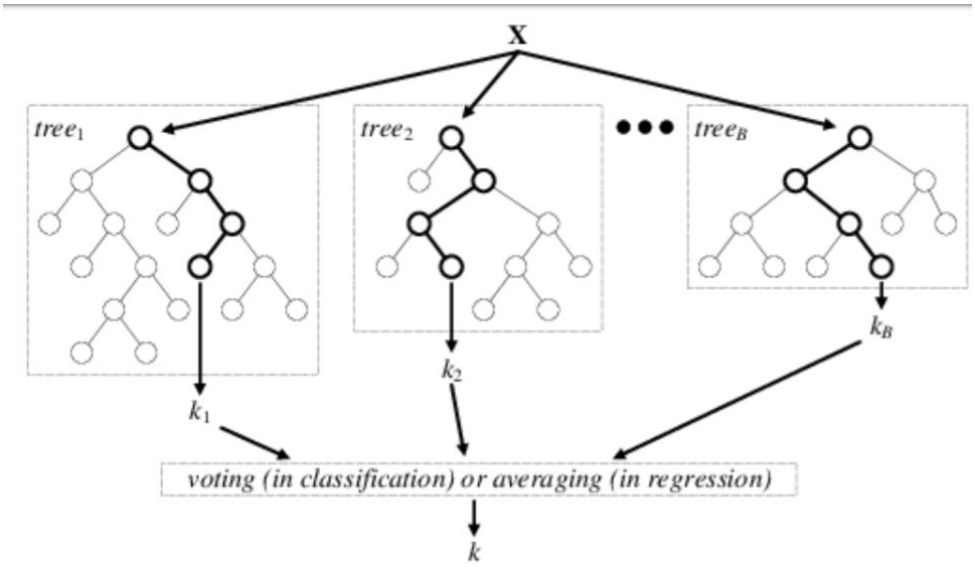


Figure 3.6: Random Forest technique [Verikas et al. (2016)].

3.2.4 Multilayer Perceptrons (MLP)

Multilayer Perceptrons (MLP) are feed-forward neural networks. Supervised learning tasks often apply MLPs. They learn to model the correlation (or dependencies) in two phases process: *forward* pass and backward pass. In the forward pass, the training data flow moves from the input layer through the hidden layers to the output layer (also called the visible layer); see Figure 3.7. There, the prediction (decision) of the output layer is measured against the target labels. The error can be measured in a variety of ways, e.g. root mean squared error (RMSE). In the *backward* pass, $backpropagation$ is used to make model parameters, i.e. *weigh* and *bias* adjustments relative to the error. That act of differentiation, based on any gradient-based optimization algorithm, gives us a landscape of error. During the convergence state, we adjust the parameters along the gradient, which minimizes the error of the model.

3.2.5 Regularized linear classifiers with stochastic gradient descent (SGD)

Regularized linear classifiers with stochastic gradient descent (SGD) SGD is a very efficient approach in the context of large-scale learning. For classification purposes, regularized linear classifiers use plain stochastic gradient descent learning routine which supports different regression *loss functions* (L), that measures model (miss-) fit and *penalties* (R), regularization term that penalizes model complexity. SGD is fitted with the training samples and the target values (class labels) for the training samples and each observation

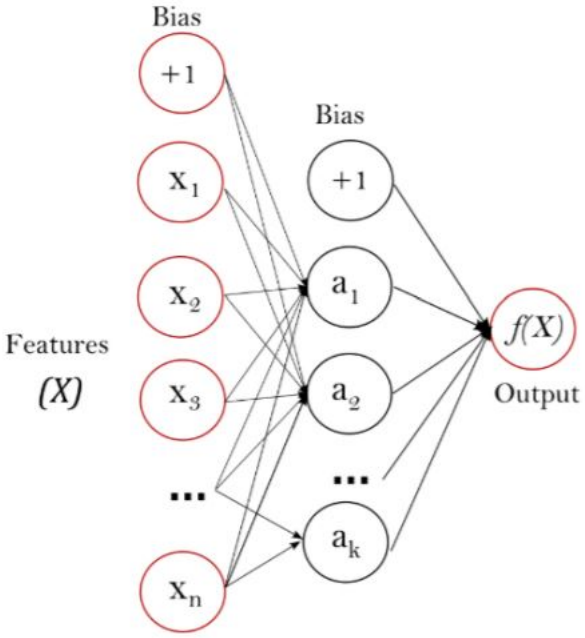


Figure 3.7: Multilayer perceptron diagram [scikit learn.org (2013b)].

updates the model parameters: $weights$ and $bias$ (also called offset or intercept). A common choice to find the model parameters is by minimizing the regularized training error. Stochastic Gradient Descent is sensitive to feature scaling, so it is highly recommended to scale the training data set before learning a linear scoring function with model parameters and intercept[5].

3.2.6 Naive Bayes classifier (NB)

Naive Bayes classifier (NB) belongs to the group of probabilistic classifiers. This group of ML models predicts the probability distribution over classes given input variables. In particular, NB bases on the' naive' assumption that every two different features are independent (Bayes' Theorem). For Gaussian Naive Bayes, the likelihood $P(x_i|y)$ of the features follows a Gaussian distributions (3.8). The main advantage of applying Naive Bayes classifiers is that they are highly scalable when presented with large amounts of data[6].

3.3 Data transformation

3.3.1 Dataset transformation: pre-processing and scaling

An important step before building the model is to analyze and pre-process the data to ensure that dataset is in a usable format when training and testing different ML models. The way of representing the types of features might have a significant effect on the performance of the

[5]https://scikit-learn.org/stable/modules/sgd.html
[6]They take an approximately linear time to train when adding features.

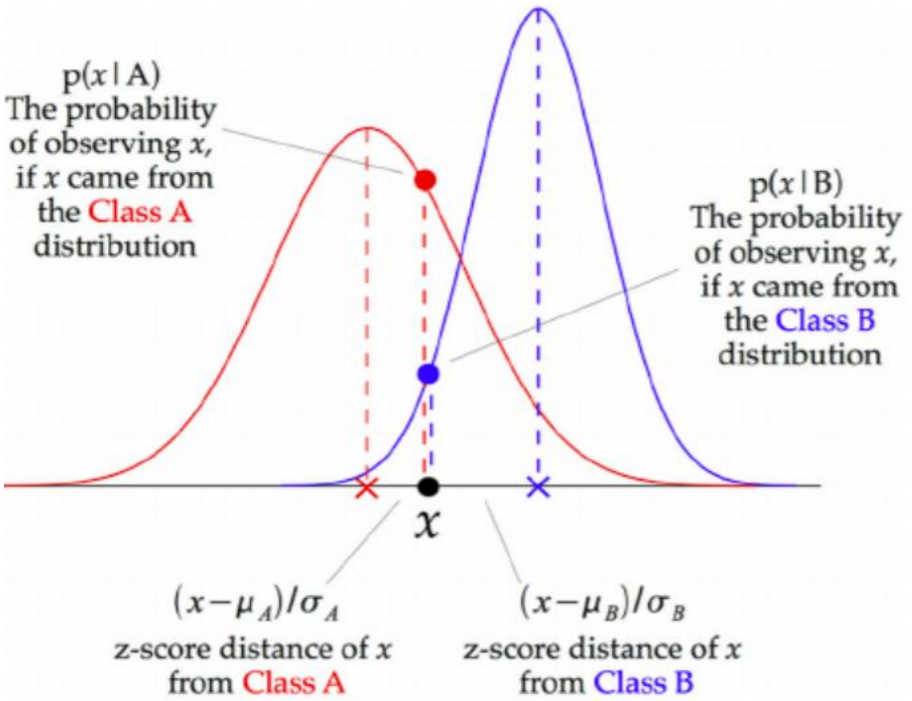


Figure 3.8: Gaussian Naive Bayes method [Raizada and Lee (2013)].

ML models. Learning a new representation of the data can sometimes improve the accuracy of supervised algorithms, or can lead to reduced memory and time consumption.

Data scaling

Neural networks and SVMs are very sensitive to the scaling of the data. Therefore, the common practice is to adjust the features, so the data representation is more suitable for these algorithms. **StandardScaler** from *scikit-learn* library ensures that for each feature the mean is 0, and the variance is 1. Thus, all features are brought to the same magnitude. The effect of rescaling the data is quite significant.

Handling skew data

Skew data are the imbalanced data, not uniformly distributed over all labels. In this research, we use classifiers, where each winner is a separate class. The baseline model consists of $N = 20$ different candidates. In the Spotify dataset, for some cases, there are very few profiles (observations) of the same winner (i.e., very few training examples per class.). That makes it hard to train most classifiers. Additionally, in the future, we might like to add a new candidate for the winner, without needing to retrain a large model.

Data augmentation

The Spotify dataset results in a very biased winner distribution, meaning contains a lot of one class winners, whereas the other class not many. If we continued with the future

extraction or learning process from this baseline model, the result would be overwhelmed by the likelihood of the candidate of the most frequently appearing class. Thus, we generated the dataset balanced in the sense of Borda score (symmetric, not biased) for 20 unique alternatives (classes).

3.3.2 Feature engineering

Feature engineering is an approach of representing a given dataset in the best way for particular application[7] (the most informative approach for learning process). Representing data in the right way can have a bigger influence on the performance of a supervised model than the exact parameters we choose. In practice, the features that are used (and the match between features and method) is often the most important piece in making an ML approach work well. Finding the transformation that works best for each combination of the dataset and model is considered as the most challenging step (non-intuitive approach). Tree-based models (such as decision trees, gradient boosted tree and random forest) or SVMs might often be able to discover complex, necessary dependencies themselves, and don't require transforming the data explicitly most of the time, in contrast to, in particular, families of linear models.

3.4 Testing

Below we presented different testing methods and metrics that we applied in this thesis as a foundation of optimizing the model.

3.4.1 Cross-validation testing

Cross-validation is one of the most commonly applied validation techniques used when training a model to avoid over-fitting. That is, in the situation when the model fits the training data very well but is not able to generalize to unseen before data. Figure 3.9 illustrate the principle of the cross-validation model training technique. The attribute of this method is k denoting the number of *folds* (also called *sections*) into which the whole training dataset is divided. For instance, given the model selection section with 4- folds cross-validation, the whole training set is arbitrarily divided into four equal folds. For a dataset with 20 instances and $k = 4$, the dataset is divided into set of 5 and 15 instances for k-folds: $1, 2, 3, 4$. In this case of the cross-validation, the step is as follows:

- 1 used as a validation set and 2, 3, 4 used as a training set,
- 2 used as a validation set and 1, 3,4 used as a training set,
- 3 used as a validating set and 1, 2, 4 used as a training set,

[7] It is one of the main tasks of data scientists and ML practitioners trying to solve real-world problems.

- 4 used as a validating set and 1, 2, 3 used as a training set.

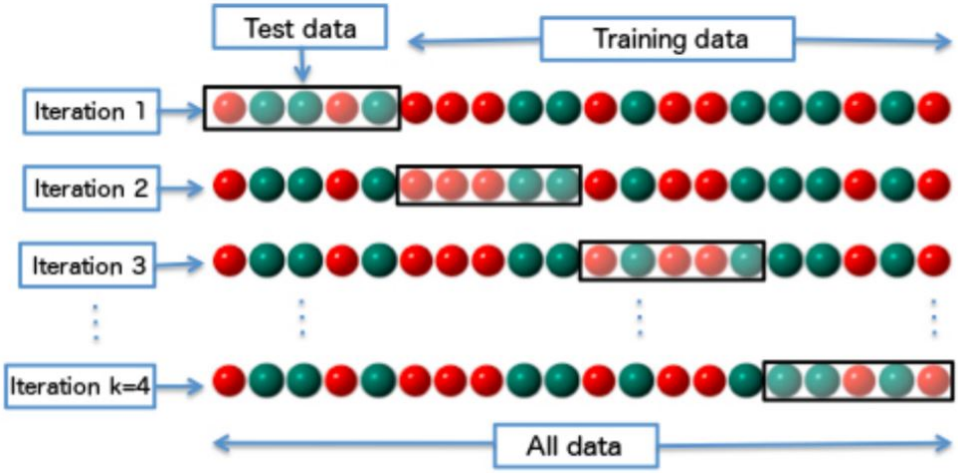


Figure 3.9: Diagram of the cross-validation model training technique [Wikipedia (2019c)].

Next, the validation error from each step is taken and the mean is used as a final validation error. Now, for datasets with the large training set and validation set, the model with the least validation error will be selected as the best model for the given dataset. In contrast, for a small dataset with a small training set and validation set, there might be again a case of overfitting problems. The main pitfall of using cross-validation technique is that the training time grows proportionally to the number of cross-validation iterations. Generally, it is worth sacrificing model training speed to reach a more robust final model that is less prone to over-fitting.

3.4.2 Evaluation metrics

Two different evaluation metrics have been used to assess the performance of ten classifiers: accuracy and F1-score.

- **accuracy** is a simple metric defining the performance of classification model based on the proportion of samples that a classifier correctly predicted:

$$accuracy = \frac{\#true}{\#total}, \tag{3.1}$$

  where:

  - $\#true$ is the number of samples correctly predicted by classifier,
  - $\#total$ is the total number of profiles (samples)

- **F1-score** is a more complex metric than the accuracy, in particular for interpreting a multi-label classification problem we have in this thesis (20 possible classes). F1-score is calculated for each class (category) separately and the average is taken as the final F1-score:

$$F1score = 2 * \frac{precision * recall}{precision * recall}, \tag{3.2}$$

where:

- $precision = \frac{t_p}{t_p+f_p}$, here:
  t_p is the number of true positive, namely the number of samples correctly predicted as positive; f_p is the number of false positive, namely number of samples wrongly predicted as positive
- $recall = \frac{t_p}{t_p+f_n}$, here:
  t_p is the number of true positive; f_n is the number of false negative, namely number of samples wrongly predicted as negative

To choose the ML classifier with the best performance, we applied both metrics, accuracy and F1-score. **These metrics are cost functions to be maximized**.

3.5 Evaluating learning algorithm

3.5.1 Error analysis

From the wide range of ten examined classification models predicting the winner for a new observation, only one model has been selected. We chose the model based on an evaluation metric when we analyzed errors. **The error is the percentage of examples for which the predicted top-ranked alternative was incorrect in comparison to the given label**. The goodness of the model to predict a new observation is called *generalization*. However, as a learning algorithm (model) fits a training set well, that does not mean it is a good classifier. There are two common pitfalls in the generalization process: under- and overfitting. *Underfitting* occurs when a simple hypothesis is trying to explain complex data. In this case, the training set contains already a lot of false predictions. On the other hand, selecting a complex hypothesis for the too-small data set can also compromise the data. In this case, the selected hypothesis works perfectly for the given data set. However, the problem arises when it is tested for new instances[8]. Thus, in the first place, we have to conduct explanatory data analysis of given datasets. This allows deciding if the dataset is big or small, biased or balanced, linear or nonlinear, simple or complex and so on, to eventually select a hypothesis describing the best fit between modeled function and given data. Model convergence is reflected in the evaluation of learning algorithms and takes place with error analysis.

General trouble-shooting process

Once the trouble-shooting process for errors in the model predictions is conducted by:

- providing more training examples - by the generation of high-dimensional synthetic dataset;

[8]The error of selected model as measured on the data set with which model trained the parameters is lower than the error on any other data set, in particular, testing set. Thus, the problem of over-fitting is easily visible on the later stage of the testing phase, where the selected model is tested with real-world data. Therefore, minimal error might lead to the false conclusion of best model selection.

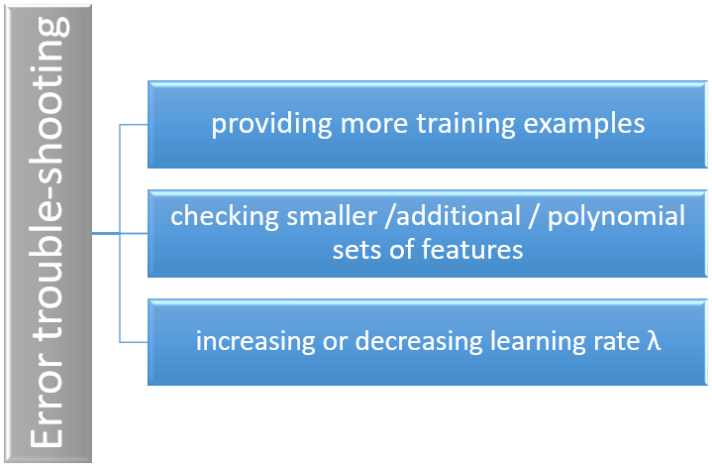


Figure 3.10: Errors trouble-shooting techniques. Modified from Ng (2019).

- checking smaller sets of features /additional features / polynomial features - by introducing five different feature representation based on the row vector of preferences;
- increasing or decreasing learning rate λ.

then the evaluation of the next (new) hypothesis began, see Figure 3.10.

3.5.2 Diagnosis tool: learning curves

A learning curve is a graphical representation of model learning performance[9] over experience or time. The learning curves were used in this research as an ML diagnostic tool for generalization behavior (under-fitting, over-fitting or good fit) of ML models. It diagnoses if the datasets (training and validation) are not relatively representative of the problem domain. Figure **??**, for instance, presents the learning curves of Borda model performance on the train and test sets learned in Representation 1.

The learning curve's shape was analyzed during the experimental study (see Chapter 4) to diagnose the behavior of ML models, in particular, to suggest possible improvement avenues for learning or performance process. In this diagnosis process **, we were looking at maximizing metric (accuracy/F1-score)**, meaning that higher relative scores on the y-axis indicate more or better learning.

3.5.3 Diagnosis tool: classification rapport

The error analysis is performed by using the confusion matrix tool, which shows the error rate, in particular false positives, for every label's configuration. Moreover, the tuning technique conducted to increase the performance (reduce the error rate) consists of additional checking if the winner model predictions are included in the original label, i.e. before breaking ties. This check, however, is only possible in case of irresolute voting rules, meaning those that can produce more than one winner in the aggregation process.

[9] algorithms that learn from a training dataset incrementally

3.6 Conclusions

In this chapter, we introduced preliminaries from ML theory. We defined voting as a supervised classification problem. We presented an overview of used supervised ML techniques for classification. We discussed diagnosis tools for (i) error analysis and (ii) model behavior, i.e. how to detect common pitfalls of learning from rank data, such as under-/overfitting and representations of the dataset for the problem domain. We presented also testing techniques and model evaluation metrics.

Chapter 4

Experimental study

4.1 Introduction

In this chapter, we described the experimental study. Figure 4.1 visualizes the general overview of the process' steps. Next, we introduced four different feature representations' approaches for forecasting (learning and predicting). Finally, we discussed the model tuning and evaluation to select the best classification model for winner prediction.

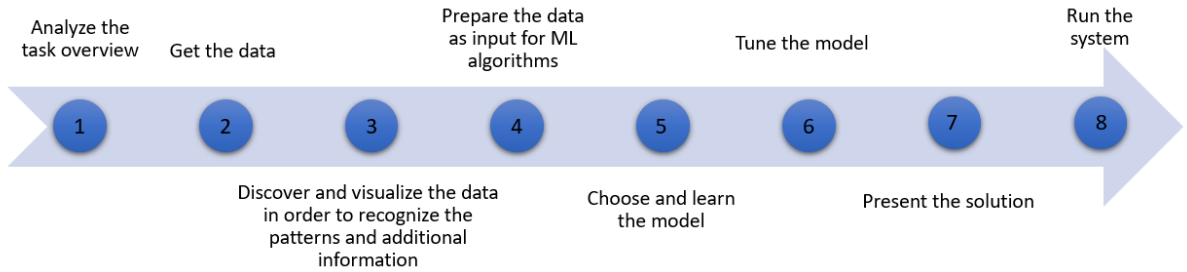


Figure 4.1: ML project's steps. Modified from Aziz et al. (2013).

Winner determination problem over 4 alternatives becomes NP-hard[1] [F. Rossi and Walsh (2011)]. We explored experimentally if machine learning algorithms can be used to predict the winner of Borda, Kemeny and Dodgson voting rules, effectively trading computational complexity for (in)accuracy. The winner determination problem in this setting is categorized into a supervised static classification task, as we fed the algorithms with labeled data off-line. The label was extracted from web-based DEMOCRATIX tool (see Section 2.4). The model prediction target is one-class label describing the winner selected by the corresponding voting model.

We measured model effectiveness with two different metrics: accuracy and F1-score (see Section 3.4.2). We represented the input of the model by a list of preference orderings (nested lists). The output is a single alternative.

[1] Nowadays there are solution for computing social choices, poll results and analysis, including probability theorems, when there are overlapping classes. The recent solutions have been been presented in the Section 1.5.

4.1.1 Machine learning pipeline

An important part of this thesis was to build a suitable training and testing pipeline to test ten classification models, with new parameters, and quickly and easily compare it to other models. A robust pipeline enabled us to find the best possible models. Implementation of this pipeline consisted of the following elements:

- we originally stored datasets in the Excel database. We imported CSV files into DataFrames in Python 3. This step allowed us to read and write to different tables easily. The pipeline reads the data from the database but does not write to it. Instead, we generated a new CSV file with saved results. Such an approach allowed us to run diverse possible versions of the model, with different parameters, in parallel. This strategy proves to be particularly useful when optimizing the model parameters.

- we applied Python 3 to write scripts for each component in the pipeline, using the following libraries: selenium for web-querying DEMOCRATIX tool; pandas, numpy, scipy for algebraic and statistical calculations; matplotlib and seaborn for visualization and analytics. This selection of many useful data analysis tools allowed us to extract data from the Excel database and use DataFrames to manipulate the data throughout the pipeline.

- we used *Scikitlearn* libraries for Python for modeling and prediction in the ML part. Easy implementation of various ML algorithms in this library allowed us to try and optimize examined ten classifiers.

- we stored model performance metrics in dedicated CSV file which allowed to keep track which models performed the best.

Before executing the model pipeline, we pre-processed the Spotify dataset (described more detailed in Section 4.3) and queried the web-based DEMOCRATIX tool with selenium library for Python. We used this labeling technique for identifying Kemeny and Dodgson winners in both datasets: Spotify and a synthetic one.

4.2 Datasets description

In the experimental study two datasets were used: (i) a real-world dataset of ranked lists of music tracks (songs) from Spotify[2] and (ii) a high-dimensional dataset of the synthetically generated ranks. Table 4.1 presents a comparison of the two datasets. Both datasets consist of profiles for $N = 20$ candidates with $V = 25$ ranked preferences (votes) per profile. The distinction between the datasets is in the number of profiles.

[2]https://spotifycharts.com/regional

Table 4.1: Dataset instances.

Instance	# candidates	# votes	# profiles (elections)
Spotify dataset	20	25	361
Synthetic dataset	20	25	12,360

Spotify dataset

The Spotify[3] dataset consists of daily top-200 music rankings for 63 countries in 2017[4]. Ranked are music tracks (songs) described by *position*, *track name*, *artis*t, *streams* and *URL*. The set of *candidates* Y[5] is thus the set of tracks and the *voters* X[6]. However, since all of the preference orders assumed to be (i)total and (ii) single-winner, and due to (iii)computational complexity and algorithms running time, it was necessary to reduce the dataset to $|C| = 20$ tracks and the *voters* to $|V| = 25$ countries[7] (meaning 25 permutations of 20 alternatives per each sample).

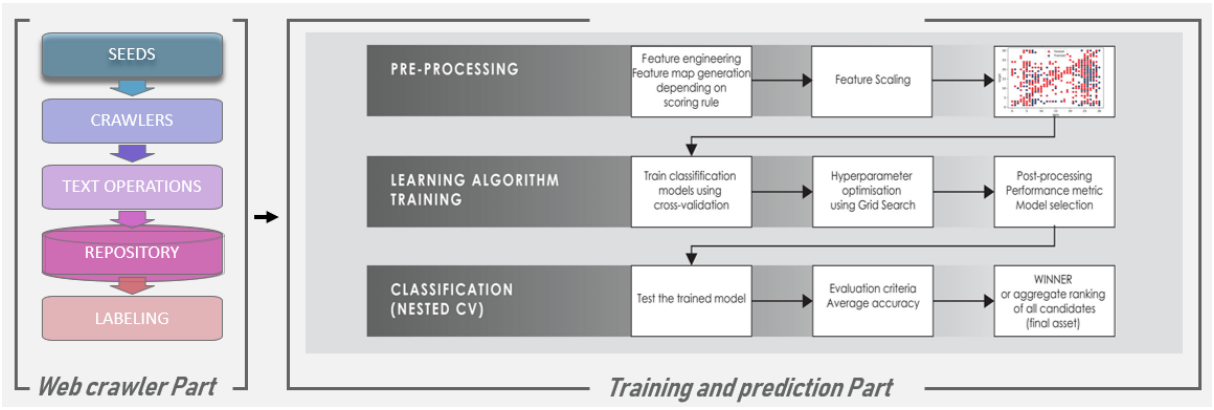


Figure 4.2: Steps of the framework.

[3]https://spotifycharts.com/regional

[4]Dataset is composed of 361 CSV files and takes 7 MB disc space. Each file represent the ranking of 200 top songs for each day and country within following columns: *position, track name, artist, streams, URL*

[5]The research aims to define the best classifier (also called predictor), such that when evaluating the function at profile x, one obtains the best prediction of its corresponding label y. Here y denotes *winner candidate*. When selecting the label's set, more precisely a list of the candidates (alternatives), it is important to considered unique tracks only, namely the combination of information from columns: *position*, *track name* and *artist*, since the *track name* alone not necessarily were unique. In the processing stage, a new *independent feature* was created from an ensemble of *track name* + *artist*, such that we stored the list of unique tracks in the dictionary, where the key was denoted by index and value by *track name* + *artist* value. Number of alternatives (songs/candidates) is equal to $N = 20$.

[6] *A single voter* preferences represents one ranking for one day in one country. We stored preference ranking as a list of codded in dictionary keys. This setting formulated 365 profiles, one for each day of the year.

[7]Retrieved and pre-processed data included the information about the position of the candidate in the rank, number of candidates in every rank N, number of ranks V in one observation (in the profile).

Synthetic dataset

We generated synthetic dataset in Python by creating permutations of $|C| = 20$ alternatives and joining them into $|V| = 25$ combination with repetitions. The following equation describes the total number of all possible to generate ranks:

$$total\_num\_profiles = \frac{(|V| + |C|! - 1)!}{|C|!(|V| - 1)!}, \tag{4.1}$$

where: $|C| = 20$ is number of alternatives, and $|V| = 25$ is number of preference ranks (votes) per sample.

In the newly created dataset, we copied the first 361 profiles from the Spotify dataset. For the next newly constructed profile, we immediately determined the Borda winner. In the synthetic dataset, we kept only those profiles that, together with already existing in dataset profiles, revealed an equal number of each of Borda's winners. Precisely, in a dataset of 12360 profiles and 20 alternatives, we created 6180 samples for each Borda winner. **Finally, we used less than 0.01% of all the possible to generate profiles** for 20 alternatives and 25 agents.

Borda, Kemeny and Dodgson dataset

For both instances from Table 4.1 we further created three datasets: separately for Borda, Kemeny and Dodgson voting model. Each dataset consists of preference ranks (the same for all voting rules) along with labels denoting the winner (unique for each voting rule). The labels determining the desired winner consist of the single alternative selected from the set of $N = 20$ values (number of candidates). Queering DEMOCRATIX[8] web-based application (see Section 2.4 for more details) led to the brute-forced determination of the winner (see Figure 4.2), and so create the dataset ready for supervised learning. **Ultimately, twenty four different datasets** (2 datasets $\cdot$ 3 voting rules $\cdot$ 4 representations $= 24$ datasets) **were established for exploration of learnability of Borda, Kemeny and Dodgson winner(s)**.

4.3 Data preparation and exploration

The process of the *explanatory data analysis* (EDA) allows deciding if the dataset is big or small, skewed or balanced, linear or nonlinear, simple or complex, to eventually select a classification model describing the best fit between modeled function and given data. EDA revealed that the Spotify profiles' labels are not uniformly distributed (referred to as *imbalanced* or *skewed* data), namely **not all candidates are winners of an approximately equal number of the profiles in the dataset.** For instance, Spotify dataset contains many profiles where the winner is candidate number 16 or candidate 18. **This was the first addressed problem that influenced the performance of the learned prediction model.**

[8] http://democratix.dbai.tuwien.ac.at/index.php

We handled this issue by generating a synthetic dataset with the same number of candidates $|C|$ and voters $|V|$ as in Spotify. However, in the newly built dataset, we kept only those profiles that ensured class-balanced output, i.e. the same number of each winner (see description of the second dataset instance in Table 4.1).

What important to recall now is that earlier to all our profile representations we introduced the pre-processing step, namely (i) lexicographic tie-breaking mechanism and (ii) dataset clearance step, meaning deleting not correctly labeled profiles, such as those with missing values or *nans* that failed during labeling when query DEMOCRATIX tool. Original Borda, Kemeny and Dodgson dataset' contained 12360 profiles. The number of Borda profiles didn't change since the DEMOCRATIX system did not query it, so Borda's dataset contained 12360 samples. Kemeny and Dodgson label were extracted based on the results from DEMOCRATIX. However, some profiles didn't reveal the winner. Instead, they were marked as not valid or interrupted during queering DEMOCRATIX web-application. We assumed it could be because of some of the profile didn't have a solution, i.e. consensus (winning aggregated rank). Thus, the pre-processing phase consists of final clearance, i.e. the elimination of the "time run-out" samples and also an implicit selection of a single winner based on the lexicographic tie-breaking mechanism results.

For Kemeny, after introducing a tie-breaking mechanism, the count didn't change; however, after final clearance, the number of profiles for training and prediction counted 10653. A final number of Dodgson's profiles used for training and predicting was 11754.

We split the dataset into a training, validation and testing sets ($70/15/15[\%]$), using the *Stratified ShuffleSplit* cross-validator from the Model Selection module of *scikit-learn* library[9], which creates a single training/testing set having equally balanced (stratified) classes. Table 4.2 presents the train, validation and test split of profiles in the generated dataset, so

Dataset	Training set	Validation set	Test set
Borda	6666	1429	1429
Kemeny	6921	1484	1483
Dodgson	7362	1578	1578

Table 4.2: Number of profiles used for training, validation and test set in generated dataset.

for instance in the Dodgson dataset 7362 data points were used for training, 1578 profiles for validation and 1578 profiles for testing.

The detailed final distribution of total and training samples per candidate is presented in Table 4.3 and illustrated in Figure 4.3(Borda winner's distribution), in Figure 4.4 (Kemeny winner's distribution) and in Figure 4.5 (Dodgson winner's distribution). For each candidate in its column, we see the number of profiles in which that candidate is the Borda /Kemeny /Dodgson winner and how many of the profiles in which the candidate is the winner are included in the training set.

[9]The seed for the random number generator during the split is equal to 42.

Table 4.3: Borda, Kemeny and Dodgson winner distribution in the generated dataset.

Candidate	as Borda winner	in training	as Kemeny winner	in training	as Dodgson winner	in training
1	483	338	597	418	483	338
10	358	250	299	209	551	386
11	481	337	419	293	511	358
12	307	215	322	225	460	322
13	491	344	397	278	542	379
14	544	381	401	281	519	363
15	522	365	387	271	499	349
16	404	283	381	267	1156	809
17	387	271	360	252	206	144
18	479	335	496	347	513	359
19	513	359	585	409	522	365
2	497	348	613	429	515	361
20	553	387	598	419	500	350
3	494	346	563	394	502	352
4	550	385	612	428	489	342
5	453	317	565	396	501	351
6	475	332	522	365	535	375
7	508	355	633	443	512	358
8	484	339	590	413	513	359
9	541	379	548	384	489	342

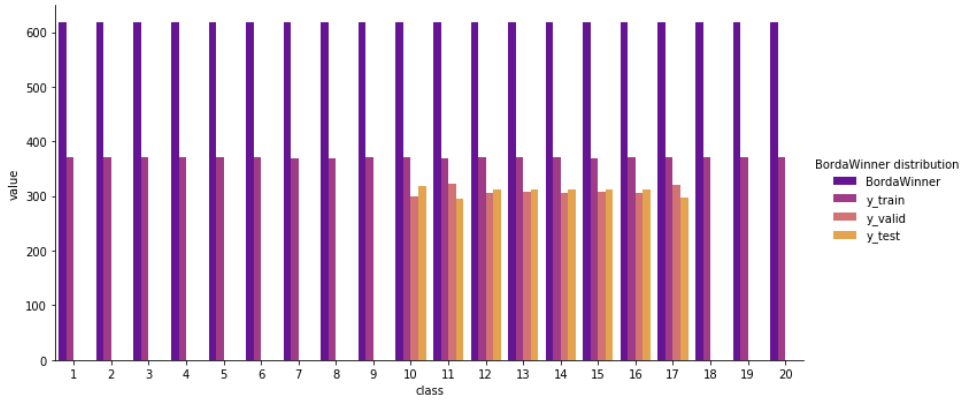


Figure 4.3: Borda winner's distribution.

4.4 Profile as data point

Profile is a unit (sample), single observation (instance) represented initially as a nested list of voters' preferences [10]. ML algorithms might not process data in the form of nested

[10] In case of Spotify dataset, it was the list of all countries' ranks on a single day. The number of profiles $X = 365$, which is equal to the number of days in the year, created the total input space. To this end, we considered the time horizon from 1-01-2017 to 31-12-2017 with the step of one day. In the case of the synthetic data set, the number of profiles was equal to $X = 12360$. Thanks to such architecture, we ensured the scalability of the solution. Precisely, a good base for scalability was not the set of rankings in the profile, but a set of voters in the profile.

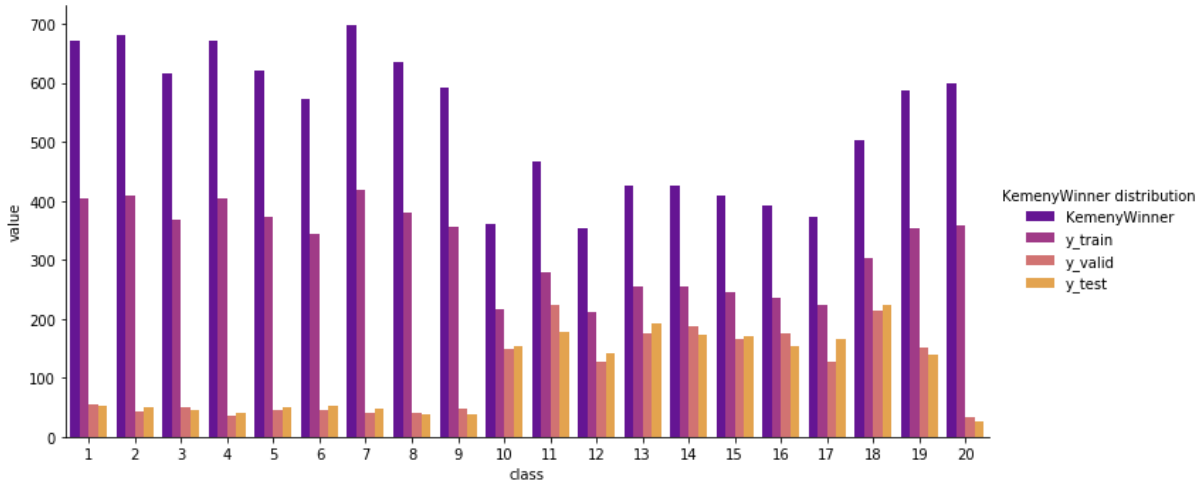


Figure 4.4: Kemeny's winner distribution.

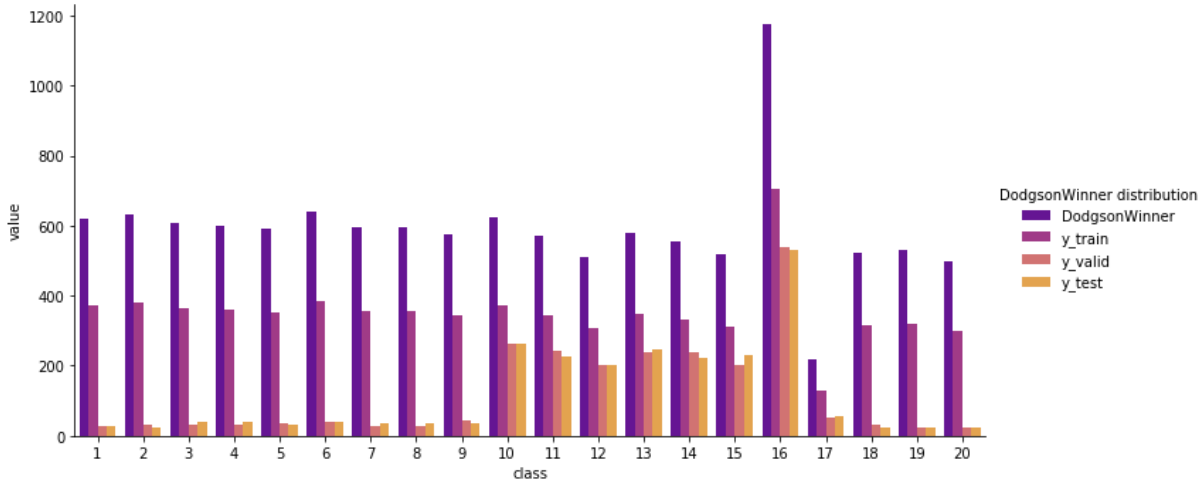


Figure 4.5: Dodgson's winner distribution.

vectors[11]. Therefore, there are several ways in which a winner determination problem can be represented as a labeled data point[12]. Being aware that the choice of representation can have a substantial impact on the winner prediction, we explored four different approaches. The approaches differ in what is considered a *feature*.

4.4.1 Representation 1: score factorisation

This factorised representation transforms original list of ranks into Borda count dataset, such that the **feature is set of the candidates** C and the **value of the feature** is **Borda**

[11]The idea behind future extraction is that it is possible to find a representation of the data that is better suited to analysis than the given raw representation

[12]A voting problem is typically a pair of $< C, V >$, where C is set of candidates, V is a collection (list) of voters each represented as preference order over C.

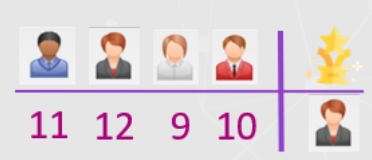


Figure 4.6: Working example factorized into Representation 1.

score[13] of the feature. The profile from Example 2.1.1 is given in Table 4.4 and Figure 4.6.

Table 4.4: The profile from Example 2.1.1 in Representation 1.

f-a	f-b	f-c	f-d	label
11	12	9	10	b

The shape of the new representation is:

$$(\#profiles, \#candidates).$$

Here, the shape of the Spotify model is $(361, 20)$ and the generated model is $(12360, 20)$.

The main drawback of this factorization is the lack of reduction in time consumption during the training phase and minor in memory (data) compression. Another disadvantage of this transformation is forced anonymity, namely lost information, e.g. about who voted for whom. From the other side, the main advantage of this approach is that the original profiles don't have to be the same size to transform the dataset to this feature representation. Moreover, once learned new representation of the data, the predictive model might be used for winner prediction of any new set of candidates. Finally, we fund this representation much more informative for further processing in terms of improved accuracy of supervised algorithms (we will discuss it in the next chapter).

Table 4.5: Preference profile example.

# Voters	Ranking
3	≻ ≻ ≻
1	≻ ≻ ≻
1	≻ ≻ ≻
1	≻ ≻ ≻
1	≻ ≻ ≻

number of voters	preference order
3 voters	$a \succ b \succ c \succ d$
1 voter	$d \succ b \succ a \succ c$
1 voter	$d \succ c \succ a \succ b$
1 voter	$b \succ d \succ c \succ a$
1 voter	$c \succ d \succ b \succ a$

4.4.2 Representation 2: occurrence factorisation

The **features** in this representation are **the set of candidates together with the position at which each candidate can be ranked**. In other words, this transformation consists of

[13] Depending on the position of the candidate in each rank n, Borda rule assigns $n-1$ points to a top ranked-choice, $n-2$ points to second ranked-choice, down to 0 points for a bottom-ranked alternative.

counting the occurrence of each candidate at each position of the ranking, such that the features are the combinations of the candidates at each rank position. The value of the feature is the number of votes for the candidate appearing at the featured position, so **the value of the feature is the number of times the candidate occurs in the particular position in the profile**. For example, if the set of candidates has four candidates, we obtain 16 features, one for each position in which each candidate can be ranked. The shape of the new representation is:

$$(\#profiles, |C| \cdot |C|).$$

Thus, for given $N = 20$ candidates and length of each vote (ranking) equal to 20 positions, we obtain 400 features. Here, the shape of the Spotify model is $(361, 400)$ and of the generated model: $(12360, 400)$. The profile from Example 2.1.1 is given in Table 4.6 and

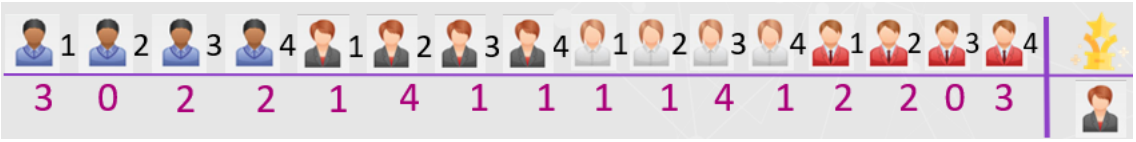


Figure 4.7: Working example factorized into Representation 2.

Figure 4.7.

Table 4.6: The profile from Example 2.1.1 in Representation 2.

f-a1	f-a2	f-a3	f-a4	f-b1	f-b2	f-b3	f-b4	f-c1	f-c2	f-c3	f-c4	f-d1	f-d2	f-d3	f-d4	label
3	0	2	2	1	4	1	1	1	1	4	1	2	2	0	3	b

The main drawback of this transformation is that we lose the information about the sequence of individual preference. On the other hand, we gain more information about hall profile complex dependencies. This presentation of given data leads to a significant reduction in time consumption, in particular in Kemeny winner prediction models. Moreover, learning this data transformation also improve the accuracy of supervised algorithms in comparison to making "most frequent" predictions or random predictions of Kemeny winners.

4.4.3 Representation 3: pairwise cumulative score

Here, the set of features is the set of unique pairs of candidates. The value of the feature is the number of voters that prefer the first candidate to the second. The profile from Example 2.1.1 is given in Table 4.7 and Figure 4.8 .

Table 4.7: The profile from Example 2.1.1 in Representation 3.

ab	ac	ad	ba	bc	bd	ca	cb	cd	da	db	dc	label
4	4	3	3	5	4	3	2	5	4	2	3	b

For given $N = 20$ candidates we have 380 possible combination (without repetition, with order).The shape of the representation is:

$$(\#profiles, \binom{|C|}{2}).$$

Here, shape of the Spotify model is: $(361, 380)$ and the generated model: $(12360, 380)$.

This transformation implies the reduction in time complexity. However, it didn't lead to an increase in the accuracy of the algorithms but stays on a similar level as making "most frequent" predictions or random predictions for Kemeny winners.

Figure 4.8: Working example factorized into Representation 3.

4.4.4 Representation 4: weighted sum

The feature is the set of the voters V and the value of the feature is the weighted sum of voter's rank. The profile from Example 2.1.1 is given in Table 4.8. The shape of the new representation is:

$$(\#profiles, \#voters).$$

Here, the shape of the Spotify model is $(361, 25)$ and the generated model is $(12360, 25)$.

Table 4.8: The profile from Example 2.1.1 in Representation 4.

f-v$_1$	f-v$_2$	f-v$_3$	f-v$_4$	f-v$_5$	f-v$_6$	f-v$_7$	label
180	180	180	187	189	187	189	b

4.5 Model selection - experiments and results.

In this section, we concentrated on experiments undertaken to select the classification models we used in the final model at different stages of our pipeline.

We needed a classification model for predicting the winner given the preferences, as a dataset, (i) in four different representations and (ii) for three voting rules. We tested and compared ten different classification models:

- Gradient Boosting Classifier
- XGBoost Classifier
- AdaBoost Classifier

- Support Vector Machine (SVM)
- Gaussian Naive Bayes
- Multilayer Perceptron
- Decision Tree Classifier
- Random Forest Classifier
- Linear classifiers with Stochastic Gradient Descent (SGD)
- Ridge classifier

4.5.1 Borda results

The results we obtained using the same parameters, keeping all other elements of the pipeline constant and training a new general classification model given preference Representation 1, are presented in Table 4.9 in columns: *Accuracy* and *F1-score*. In this table we presented also the results of execution *time for training and prediction* phase, and *precision/recall* evaluation metrics.

Table 4.9: Representation 1: Borda predictions performance after hyper-parameter tuning.

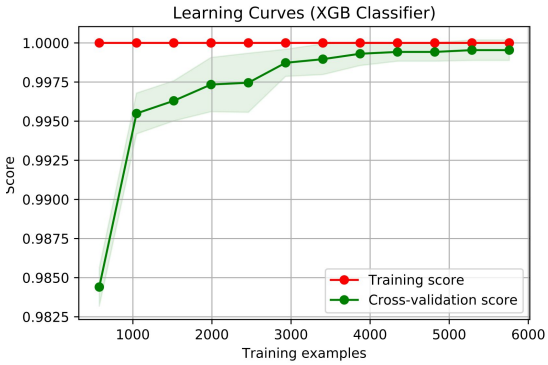
	Fitting time [s]	Prediction time [s]	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GaussianNB	0.02359819412	0.02658939362	100	100	100	100	0	0	100
XGBClassifier	3.550036907	0.04513978958	99.5	66.56	66.37	66.47	4	0.22	99.79
RandomForestClassifier	0.0721616745	0.005528211594	90.29	33.34	31.88	32.52	2	0.11	90.4
SGDClassifier	0.09017705917	0.002687692642	85.6	49.58	46.82	47.92	26	1.4	87
SVC(kernel='linear')	0.08874988556	0.01698827744	76.86	27	23.05	24.79	20	1.08	77.94
RidgeClassifier	0.02047753334	0.003262758255	70.01	26.25	21.1	21.95	5	0.27	70.28
DecisionTreeClassifier	0.02439689636	0.004003763199	66.07	49.94	57.14	52.33	0	0	66.07
RandomForestClassifier	0.03056502342	0.004910945892	4.91	25.04	17.54	19.37	44	2.37	57.28
LinearSVC(C=1.0)	0.8422548771	0.003510475159	52.32	25.82	16.05	16.64	38	2.05	54.37
AdaBoostClassifier	0.4562883377	0.02396583557	48.33	33.21	42.86	35.58	0	0	48.33
MLPClassifier	0.007092237473	0.02181887627	43.37	19.18	13	15.16	111	5.99	49.36
SVC(C=1, kernel='rbf')	0.150769949	0.01939749718	22.55	86.3	22.49	15.87	0	0	22.55

We obtain the best accuracy by using the Gaussian Naive Bayes model (100%) and XGBclassifier (99,5%). We noticed that top-performing classification models are belonging to a group of algorithms capable of generating probability predictions. Both models have been chosen to build final classification models. We took two models since 100% accurate predictions could be the result of the overfitting, and so this model would not generate well. Thus, we conducted a diagnosis of both models' behavior.

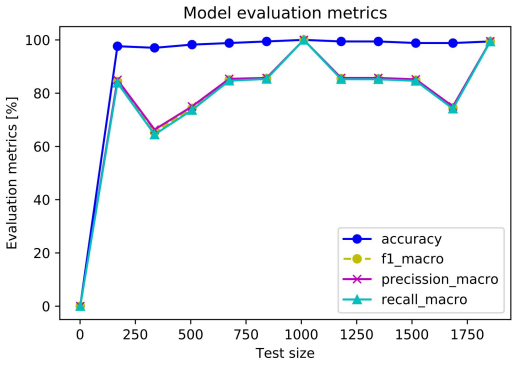
Moreover, Figure 4.9 illustrates the execution time for the training and prediction phase. The longest fitting time (ca. 3,5s) characterizes the XGBoost classifier, which in contrast, is very fast in prediction (ca. 0.045s).

The performance of the final model for Borda winners prediction was improved by:

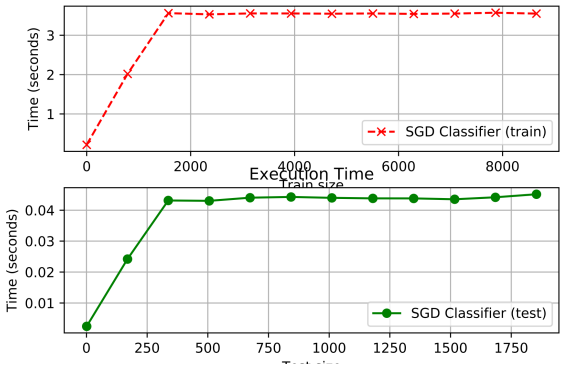
- testing classifiers against synthetic dataset (ensured winner-balanced training dataset),



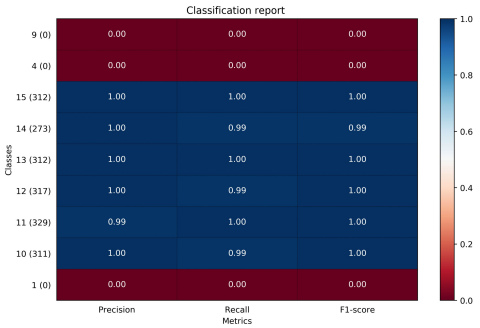
(a) Learning curves .



(b) Model evaluation metrics.



(c) Training and prediction execution time.



(d) Classification report.

Figure 4.9: XGBoost classifier learned in Representation 1.

- scaling features before the training and the testing
- checking if the predicted winner is in ties

The most accurate in Representation 1 is the Gaussian Naive Bayes classifier with 100% accuracy and 0.023s fitting time and almost the same 0.026s predicting time. Thus, we observed that the most prominent models' origin from a group of ensemble algorithms, like XGBoost, AdaBoost or Random forest classifiers, also had the highest F1-score. Here, we based model prediction on the cross-entropy of choices learned during the training phase. This approach is the reason why, e.g. XGBoost classifier had disproportional long learning process (fitting time over 3s). Additionally, the next model improvement step consisted of checking the ties. After the tie-breaking mechanism, we ensured a single-winner in the class. In reality, the voting rule could determine other winners that the model could predict correctly. We checked the ties and compared predicted the winner with the other winners before the tie-breaking mechanism. If predicted winners were present in the tie, we counted that sample as true positive. That improved the results by 0-6% of algorithm accuracy.

We built interesting insight by noticing that GaussianNaiveBayes tended to push probabilities to 0 or 1. The reason for it is because it assumes that features are conditionally independent given the class, which is the case in this dataset in Representation 1 containing not redundant features.

Table 4.10 presents detailed Borda's results for all ML models learned in Representation 2. We observed strong linearity in the Borda count system. Representation 2 also performed best for the algorithms taking into account prior probabilities of the classes, like Gaussian Naive Bayes classifier (by 100 % prediction accuracy) and accumulating the entropy of learned desired classes outcome (the information, a measure of the number of possible arrangements the class in a system can have), like XGBoost classifier (also by 100 % prediction accuracy).

Table 4.10: Representation 2: Borda predictions performance after hyper-parameter tuning.

	Fitting time [s]	Prediction time [s]	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	30.4573338	0.05381321907	100	100	100	100	0	0	100
GaussianNB	0.02711296082	0.02467918396	100	100	100	100	0	0	100
DecisionTreeClassifier	0.08309459686	0.004017114639	66.07	49.94	57.14	52.33	0	0	66.07
SVC($kernel =' linear'$)	1.517385006	0.1696600914	64.72	33.04	29.97	30.75	0	0	64.72
AdaBoostClassifier	1.4271245	0.04778265953	48.33	33.21	42.86	35.58	0	0	48.33
LinearSVC	2.075093031	0.004464149475	41.96	19.57	13.33	15.58	2	0.11	42.07
SGDClassifier	0.3524949551	0.004344463348	40.45	20.3	14.38	16.18	0	0	40.45
RidgeClassifier	0.05543828011	0.004752635956	30.04	17.12	9.09	11.45	52	2.8	32.84
MLPClassifier	0.03619837761	0.4821381569	29.29	11.43	9.74	9.87	1	0.05	29.34
SVC($kernel =' rbf'$)	1.579460859	0.174980402	22.55	86.3	22.49	15.87	0	0	22.55
RandomForestClassifier	0.1510174274	0.007100582123	17.31	9.4	5.2	6.4	151	8.14	25.45

Figure A.1 presents model evaluation metrics for Representation 2. Noticeable two algorithms XGBoost and Gaussian NB classifiers learned Representation 2 dataset extremely well (by 100%). The learning process of the XGBoost predictor was significantly longer (more than the 30s) than other models.

Tables 4.7 and 4.12 presents Representation 3 and Representation 4 performance respectively. Those representations were more descriptive for the tied samples, and here it improved the examined models' accuracy by extra $\tilde{4}$ % and 12% for MLP predictor. Again, the best classifiers - XGBoost and Gaussian NB -learned the linearity of the Borda score rule in Representation 3 and 4 with 100% accuracy.

Table 4.11: Representation 3:Borda predictions performance after hyper-parameter tuning.

	Fitting time [s]	Prediction time [s]	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	30.21589756	0.05239534378	100	100	100	100	0	0	100
GaussianNB	0.02364778519	0.02355194092	100	100	100	100	0	0	100
SVC(kernel='linear')	0.7123975754	0.1593894958	81.55	26.87	24.49	25.57	2	0.11	81.66
LinearSVC	9.615909815	0.004586696625	70.12	26.04	21.07	23.2	9	0.49	70.61
DecisionTreeClassifier	0.1124827862	0.003812789917	66.07	49.94	57.14	52.33	0	0	66.07
SGDClassifier	0.7037308216	0.007167100906	60.46	29.33	19.66	19.67	20	1.08	61.54
AdaBoostClassifier	1.689310789	0.04125356674	48.33	33.21	42.86	35.58	0	0	48.33
MLPClassifier	0.01814723015	0.2230367661	46.44	18.87	13.92	15.74	72	3.88	50.32
RandomForest(depth = 5)	0.1785812378	0.007026910782	45.47	19.74	13.66	15.47	76	4.1	49.57
RidgeClassifier	0.04805111885	0.004705429077	38.78	20.57	11.65	14.16	36	1.94	40.72
RandomForestClassifier	0.03068852425	0.006850004196	31.45	17.29	9.4	11.17	76	4.1	35.55
SVC(kernel='rbf')	1.510657787	0.1677224636	22.55	86.3	22.49	15.87	0	0	22.55

Table 4.12: Representation 4: Borda predictions performance after hyper-parameter tuning.

	Fitting time [s]	Prediction time [s]	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	3.655939102	0.03061032295	100	100	100	100	0	0	100
GaussianNB	0.01229357719	0.004845142365	100	100	100	100	0	0	100
DecisionTreeClassifier	0.02488136292	0.002355337143	66.07	49.94	57.14	52.33	0	0	66.07
AdaBoostClassifier	0.5383927822	0.02752375603	48.33	33.21	42.86	35.58	0	0	48.33
SVC(kernel='linear')	513.806108	0.01938676834	43.58	29.09	26.36	26.87	0	0	43.58
RandomForestClassifier	0.09772253036	0.00602555275	31.01	11.75	9.36	9.8	15	0.81	31.82
SVC(kernel='rbf')	0.1583929062	0.01837658882	22.55	86.3	22.49	15.87	0	0	22.55
MLPClassifier	0.006424665451	0.02417945862	12.57	7.17	3.78	4.89	218	11.76	24.33
LinearSVC	4.357069254	0.003664731979	10.14	5.11	3.41	3.14	13	0.7	10.84
RandomForest(depth = 5)	0.03180623055	0.005166292191	9.12	7.29	2.73	3.67	43	2.32	11.44
RidgeClassifier	0.01920890808	0.002331256866	7.93	6.8	2.41	3.11	8	0.43	8.36
SGDClassifier	0.1527459621	0.002839565277	0.16	1.59	0.14	0.25	0	0	0.16

4.5.2 Kemeny results

Table 4.13 shows the performance of the algorithms learning Kemeny labels by Representation 1. Here top predictors were: XGBoostCLassifier, Gaussian NB and Random Forest classifier with the after hyperparameter tuning performance ranging from 58,13 %, 50,37% and 51,19% respectively. The percentage of the correctly predicted winner involved in the ties varied from 5 % for Random Forest classifier up to ca.30% for AdaBoost classifier.

Table 4.13: Representation 1: Kemeny rule performance after hyper-parameter tuning.

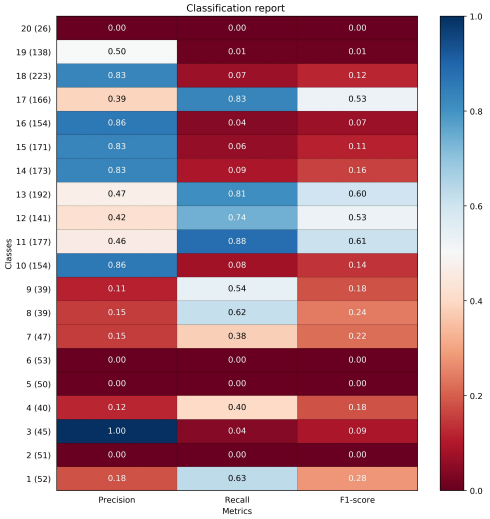
	Fitting time [s]	Prediction time [s]	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	4.137499571	0.07161974907	51.81	38.96	37.51	37.4	101	6.32	58.13
GaussianNB	0.01627016068	0.03955364227	44.43	35.39	36.51	34.88	95	5.94	50.37
RandomForestClassifier	0.2941424847	0.03845930099	42.55	34.31	35.42	33.47	138	8.64	51.19
SVC(kernel='linear')	1.134880781	0.05269598961	40.43	32.57	31.91	31.08	136	8.51	48.94
RidgeClassifier	0.008381843567	0.00386762619	35.48	36.86	33.36	28.43	205	12.83	48.31
LinearSVC	1.357146263	0.006160259247	35.23	40.81	27.36	24.62	97	6.07	41.3
RandomForest(depth = 5)	0.03054499626	0.00642156601	34.61	31.47	27.64	26.26	130	8.14	42.75
SGDClassifier	0.1166534424	0.007428407669	25.34	30.36	23.42	20.94	93	5.82	31.16
AdaBoostClassifier	0.8649594784	0.1009278297	24.97	21.46	18.59	17.7	472	29.54	54.51
MLPClassifier	0.004200696945	0.02492451668	18.46	19.86	19	15.02	253	15.83	34.29
SVC(kernel='rbf')	0.5110402107	0.07839155197	8.64	90.14	10.46	10.42	151	9.45	18.09
DecisionTreeClassifier	0.03039526939	0.02007961273	5.94	16.57	12.84	6.88	109	6.82	12.76

Table 4.14 presents the models performance trained by Representation2. The best results achieved XGBoost classifier with extremely long fitting time over 90 s and the accuracy 33%. The tie-breaking mechanism decreased the performance results up to 25% for the AdaBosst predictor.

Table 4.15 illustrates the models performance trained by Representation 3. Surprisingly, the SGDC classifier reached very high accuracy: 85%, after checking tied labels. That was a very good result taking into account that the accuracy of the first predictions was on the level of 2,4 % model accuracy (see Figure 4.10). Next, Random Forest classifier and SVM with *rbf* kernel reached the prediction results on the level of 66 and 64% of accuracy, respectively.

Table 4.16 presents results of algorithms trained in Representation 4. Random Forest

(a) Learning curves.



(b) Classification report.

Figure 4.10: SGD classifier results trained in Representation 3.

Table 4.14: Representation 2: Kemeny rule performance after hyper-parameter tuning.

	Fitting time [s]	Prediction time [s]	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
XGBClassifier	90.58778834	0.2741849422	21.53	24.42	21.8	18.02	188	11.76	33.29
LinearSVC	5.24428463	0.03603768349	19.27	19.89	19.28	16.71	149	9.32	28.59
SVC(kernel='linear')	1.168784142	0.1247189045	19.21	21.67	19.19	16.49	201	12.58	31.79
SGDClassifier	1.073603153	0.03670167923	17.83	24.42	18.43	16.14	91	5.69	23.52
RidgeClassifier	0.03056669235	0.004123210907	17.65	19.46	17.6	15.23	151	9.45	27.1
GaussianNB	0.01425242424	0.01847815514	16.21	22.71	18.24	14.7	189	11.83	28.04
RandomForestClassifier	0.09783053398	0.00573348999	9.64	15.38	10.75	9.09	278	17.4	27.04
SVC(kernel='rbf')	3.055214882	0.2706463337	8.64	90.14	10.46	10.42	151	9.45	18.09
MLPClassifier	0.06370639801	0.6993527412	7.32	11.25	8.93	6.38	241	15.08	22.4
AdaBoostClassifier	0.946659565	0.03643536568	6.7	15.65	7.98	5.46	402	25.16	31.86
RandomForest(depth = 5)	0.02654576302	0.005226135254	5.51	9.43	8.14	5.15	272	17.02	22.53
DecisionTreeClassifier	0.0720334053	0.003858089447	3.5	7.04	5.41	2.88	189	11.83	15.33

model predicted correctly 66,87% of tested samples after checking the results in the ties. This representation was not more informative then Representation 3 for predicting Kemeny winners.

4.5.3 Dodgson results

Dodgson voting rule is computationally even harder than Kemeny rule (NP-hard) since it requires strictly more information than a weighted directed graph [14].

Table 4.17 presents the performance of the algorithms learned in Representation 1 with Dodgson labels. Here the maximum accuracy received XGBoost classifier with 66,19%,

[14] To label dataset in a reasonable time, we decided to reduce the dataset to 20 alternatives (candidates) and 25 ranks (votes). However, we kept the same number of profiles, i.e. 12360.

Table 4.15: Representation 3: Kemeny rule performance after hyper-parameter tuning.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
SGDClassifier	2.44	0.12	5.0	0.24	1771	83.11	85.55
RandomForestClassifier	60.86	57.18	61.32	55.78	128	6.01	66.87
SVC(kernel='rbf')	60.58	95.26	61.82	71.49	82	3.85	64.43
GradientBoostingClassifier	59.5	55.43	58.66	54.09	79	3.71	63.21
XGBClassifier	26.94	38.13	28.27	25.07	175	8.21	35.15
RandomForest(depth = 5)	7.74	37.15	14.93	8.26	348	16.33	24.07
AdaBoostClassifier	6.29	7.04	7.32	5.04	232	10.89	17.18
DecisionTreeClassifier	5.02	10.71	6.6	3.65	252	11.83	16.85
GaussianNB()	6.34	8.28	7.72	5.37	219	10.28	16.62
RidgeClassifier	4.27	8.11	6.99	3.09	245	11.5	15.77
MLPClassifier	4.41	2.54	4.71	1.7	115	5.4	9.81
LinearSVC	6.62	0.33	5.0	0.62	15	0.7	7.32

Table 4.16: Representation 4: Kemeny rule performance after hyper-parameter tuning.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
RandomForest	60.86	57.18	61.32	55.78	128	6.01	66.87
SVC	60.58	95.26	61.82	71.49	82	3.85	64.43
GradientBoosting	58.94	54.92	58.14	53.59	88	4.13	63.07
XGBClassifier	26.94	38.13	28.27	25.07	175	8.21	35.15
RandomForest(max_depth=5)	7.74	37.15	14.93	8.26	348	16.33	24.07
AdaBoost	6.29	7.04	7.32	5.04	232	10.89	17.18
DecisionTree	5.02	10.71	6.6	3.65	252	11.83	16.85
GaussianNB	6.34	8.28	7.72	5.37	219	10.28	16.62
RidgeClassifier	4.27	8.11	6.99	3.09	245	11.5	15.77
SGDClassifier	10.46	0.52	5.0	0.95	22	1.03	11.49
MLPClassifier	3.94	2.58	5.2	2.07	158	7.41	11.35
LinearSVC	6.62	0.33	5.0	0.62	15	0.7	7.32

next GaussianNB with 62.5% accuracy (with a short fitting time 0.007 s) followed by Random Forest classifier achieving 61.54 % accuracy. We received those results after ties verification, which for Dodgson rule learned in Representation 1 consists of 0.45% (SVC(kernel='rbf') classifier) up to over 10% (MLP classifier) improvement in predictors accuracy.

Tables 4.18, 4.19 and 4.20 present the performance of the algorithms learned Dodgson's winners in Representation 2, 3 and 4, respectively.

Noticeable, the best classifier trained in Representation 1 was ensemble algorithm: Gradient Boosting with the accuracy of 89,37% (after miss-prediction analysis).

The next model, also from a group of ensemble algorithms: Random Forest, reached 86,52% and finally even from the same group: XGBoost classifier with the result of 81,46%. The robustness of false positives ranged between 1% (Decision Tree classifier) and 5,5% (SGD classifier).

The best performance of Dodson's winner prediction in Representation 2 reached Gradi-

Table 4.17: Representation 1: Dodgson rule performance after hyper-parameter tuning.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GradientBoostingClassifier	87.2	81.37	75.34	77.95	51	2.17	89.37
RandomForestClassifier	84.22	75.05	73.68	73.87	54	2.3	86.52
XGBClassifier	77.63	65.96	48.98	53.13	90	3.83	81.46
GaussianNB	66.31	42.34	41.84	41.63	120	5.1	71.41
RandomForest(depth = 5)	66.57	55.43	36.24	39.26	88	3.74	70.31
SVC(kernel='rbf')	68.06	97.07	61.14	73.34	10	0.43	68.49
RidgeClassifier	63.16	40.37	38.84	38.73	114	4.85	68.01
MLPClassifier	57.64	43.33	44.4	38.79	113	4.81	62.45
AdaBoostClassifier	55.98	35.27	36.61	35.32	117	4.98	60.96
SGDClassifier	50.91	27.14	30.64	23.01	129	5.49	56.4
LinearSVC	34.11	34.02	25.55	19.77	116	4.93	39.04
DecisionTreeClassifier	23.52	17.17	12.22	8.5	25	1.06	24.58

Table 4.18: Representation 2: Dodgson rule performance after hyper-parameter tuning.

	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GradientBoostingClassifier	80.77	68.75	73.07	70.46	59	2.51	83.28
MLPClassifier	77.16	63.44	71.98	66.57	68	2.89	80.05
XGBClassifier	69.72	53.67	57.13	54.12	97	4.13	73.85
SVC(kernel='rbf')	68.06	97.07	61.14	73.34	10	0.43	68.49
LinearSVC	62.91	51.52	55.26	50.62	125	5.32	68.23
RandomForestClassifier	61.93	52.59	62.03	52.66	141	6.0	67.93
RidgeClassifier	63.16	46.38	51.67	47.73	102	4.34	67.5
GaussianNB	57.59	42.61	51.65	44.99	120	5.1	62.69
SGDClassifier	53.21	48.54	40.55	37.82	93	3.96	57.17
AdaBoostClassifier	25.1	19.39	22.26	18.23	195	8.29	33.39
DecisionTreeClassifier	21.23	9.1	9.0	5.52	66	2.81	24.04
RandomForest(depth=5)	22.59	1.13	5.0	1.84	19	0.81	23.4

ent Boosting Classifier with accuracy by 83,38% (with the depreciation of the tie-breaking mechanism). Next, neural network classifier MLPC with 80,05% and finally, XGboost classifier was closing the top triad with 73,85% prediction accuracy. Dodgson's winner miss predictions analyzed by checking possible winner in ties allowed to increase the algorithms performance from 0.5(SVC) - 8,3% (AdaBoost).

The Representation 3 is the most effective in learning Dodgson's winners. The Gradient Boosting classifier learned in Representation 3 reached an accuracy of 89.41%, slightly higher than the same model trained in Representation 1.

Finally, the Dodgson WDP problem tackled by predictive models learned in Representation 4 reached 68,49% of accuracy by the SVC algorithm with *rgb* kernel. The number of false-positive in ties varied from 0,5% (SVC classifier) to 8,5% (Ridge classifier).

Table 4.19: Representation 3: Dodgson rule performance after hyper-parameter tuning.

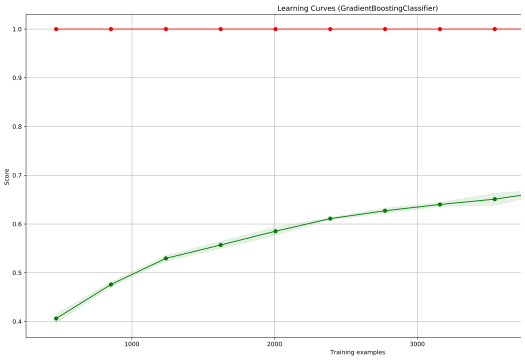
	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
GradientBoostingClassifier	87.28	80.85	77.62	78.91	50	2.13	89.41
XGBClassifier	85.92	77.83	78.41	77.9	66	2.81	88.73
RandomForestClassifier	71.08	59.15	69.32	60.71	105	4.47	75.55
GaussianNB	65.76	50.52	45.52	45.62	115	4.89	70.65
SVC(kernel='rbf')	68.06	97.07	61.14	73.34	10	0.43	68.491
RidgeClassifier	62.14	42.01	43.51	41.85	113	4.81	66.95
MLPClassifier	54.4	39.04	42.54	38.8	110	4.68	59.08
SGDClassifier	50.87	36.31	29.72	22.74	128	5.44	56.31
LinearSVC	41.56	46.0	19.17	20.69	51	2.17	43.73
RandomForest(depth = 5)	26.8	62.52	10.2	10.87	23	0.98	27.78
DecisionTreeClassifier	17.31	10.01	18.27	9.93	192	8.17	25.48
AdaBoostClassifier	15.06	11.9	14.83	9.21	172	7.32	22.38

Table 4.20: Representation 4: Dodgson rule performance after hyper-parameter tuning.

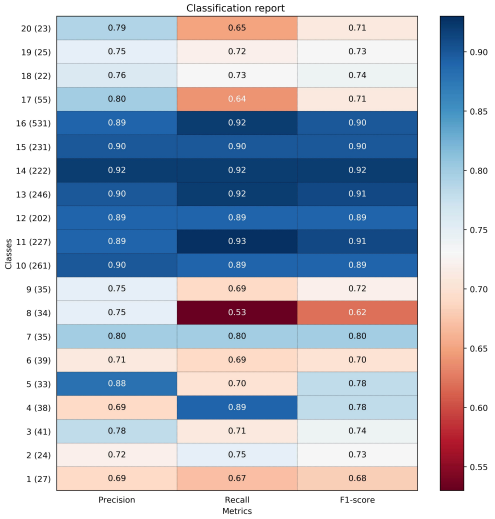
	Accuracy [%]	Precision [%]	Recall [%]	F1score [%]	# In ties [# samples]	ties [%]	Acc. with ties [%]
SVC(kernel='rbf'	68.06	97.07	61.14	73.34	10	0.43	68.49
RandomForestClassifier	61.12	50.79	60.55	51.39	140	5.95	67.07
GradientBoostingClassifier	58.23	48.46	57.31	49.4	73	3.11	61.34
XGBClassifier	32.41	31.67	23.27	21.26	125	5.32	37.73
SGDClassifier	22.59	1.13	5.0	1.84	19	0.81	23.4
RandomForest(depth = 5)	22.59	1.13	5.0	1.84	19	0.81	23.4
DecisionTreeClassifier	19.61	5.78	6.26	3.41	165	7.02	26.63
RidgeClassifier	19.27	3.83	6.58	2.91	200	8.51	27.78
GaussianNB	17.74	8.51	9.31	6.12	167	7.1	24.846
AdaBoostClassifier	15.65	8.91	8.29	5.6	184	7.83	23.48
LinearSVC	8.59	0.43	5.0	0.79	18	0.77	9.36
MLPClassifier	6.17	1.55	5.26	1.47	59	2.51	8.68

4.6 Discussion and important remarks

The winner determination problem as a supervised classification task was explored in this research very extensively, precisely selecting details of building the best approach for unique voting scheme predictions. The research exploration covered over 120 problem settings: three separated datasets (Borda, Kemeny and Dodgson voting rules) persuaded in four fundamentally different Representations; for each of those datasets, we applied ten traditional ML algorithms.



(a) Learning curves.



(b) Classification report.

Figure 4.11: Gradient Boosting classifier results trained in Representation 3.

4.6.1 Error analysis and diagnosing model behavior

Borda miss-prediction are reported in Figure A.1. This rapport showed the main classification metrics on a per-class basis: precision[15], recall[16] and F1-score[17]. To support more intuitive interpretation and eventually problem detection, in the report, we integrated numerical scores within the color-coded heatmap. The range of the heat maps is (0.0, 1.0). Such a solution facilitated the comparison of classification across different predictive models. In this research, we were looking for the weighted accuracy of F1-score in particular, since it should be used to compare the classifier's model instead of global accuracy. The prediction set not included all classes (red areas also marked as 0 in the parenthesis). The best classifiers for Representation 1 is the XGBoost classifier. A learning curve is a graphical representation of model learning performance[18] over experience or time. The learning curves were used here as an ML diagnostic tool for generalization behavior (underfitting, overfitting or good fit) of ML models and diagnose if the datasets (training and validation) are not relatively representative of the problem domain. Figure **??** presents the learning curves of model performance on the train and test sets.

The learning curve's shape was used to diagnose the behavior of ML models, in particular, to suggest possible improvement avenues for learning or performance processes.

[15]Precision is the classifier's ability not to label an instance positive that is negative. The problem statement: for all instances classified positive, what percent was correct?

[16]Recall is the classifier's ability to find all positive instances. The problem statement: for all positive instances, what percent was classified correctly?

[17]The F1 score is a weighted harmonic mean of precision and recall. F1 range from is 1.0 for the best 0.0 for the worst.

[18]algorithms that learn from a training dataset incrementally

In this diagnosis process, the optimized function was **maximizing accuracy and F1-score metrics**, meaning that higher relative scores on the y-axis indicate more or better learning.

Borda model behavior. We identified underfitting in the following models: SVM with RBF kernel (e) and Random Forest (d), where the training loss remains flat regardless of experience[19].

On the other hand, some of the algorithms were diagnosed with over-fitting. We identified over-fitted models by analyzing the learning curve of the training and validation loss. Learning curve shows under-fitting if: (i) the training loss continues to increase until the end of training, i.e. premature halt occurs, like examined MLP (h) and SGD (j) classifier; or (ii) validation loss increases in order to decrease again, like random forest (c) at its late state.

Finally, the learning curves' plot showed a good fit for XGBoost (a), linear SVM (f) or AdaBoost (g) classifiers, where we observed (i) the training loss decrease to the point of stability and (ii) the validation loss increase to stability point but remained a small gap with the training loss.

Figure A.1 presents train and validation learning curves for models learned in Representation 2. Here, the suitable model fit was presented by XGBoost, Linear SVM with SGD learning. Representation 2 training set was easier to learn my examined ML models comparing to Representation 1 (more algorithms achieved 100% accuracy for the training set at the same time increasing accuracy for test set).

Kemeny miss-predictions. The evaluation of the Kemeny miss-predictions consists of checking if a predicted winner, classified as false positive, is the true positive, meaning the label is the desired output of the considered voting rule.

Interestingly in Kemeny's winner's prediction case, Representation 3 and 4 were very well learnable by SGD classifier. Here, however, the accuracy of the original dataset was very low ca. 2% and the number of miss-prediction was very high, ca. 83 % were wrongly classified (false positive) since were correct. Kemeny WDP is an example of the irresolute voting rule, meaning it is a multi-winner aggregation method. After checking the robustness of the negative result for Kemeny winners, we identified that 83% of original miss-predictions were found in ties, meaning were correct but ignored by the designed lexicographic tie-breaking mechanism. That analysis yielded to better model accuracy: up to 85%. Next, the ensemble algorithm, such as Random Forest, trained in Representation 4 reached 68% of accurate Kemeny's winners' classifications.

Dodgson miss-predictions. The evaluation of the Dodgson miss-prediction is similar to Kemeny's. For the most efficient predictor - Gradient Boost classifier - learned in Representation 2, the range of false positive is 2,13%, meaning 50 labels were ignored during the lexicographic tie-breaking process. We don't interpret the miss-predictions concerning where in the Dodgson consensus rank is the wrongly predicted alternative since the Dodg-

[19] We identify under-fitted models by analyzing of the learning curve of the training loss. The learning curve shows under-fitting if : (i) the training loss continues to increase until the end of the training, i.e. premature halt occurs or (ii) the training loss remains flat regardless of training.

son full global rank is unknown and finding it is not a subject of this research. The goal of the study is learning and predicting the top-winning alternative in the three voting rules. Since we consider irresolute voting rules, if there is more than a single winner, we used a tie-breaking mechanism with the voting rule.

4.7 Conclusions

Experimental study demonstrated that Borda winners could be predicted with great accuracy, as expected from the theoretical evaluation for the learnability of scoring rules from Procaccia A.D. (2009). For Kemeny's rule, we encounter some more problems, obtaining in the base model roughly 38% of predictions accurately identifying the Kemeny winner. Further experiments led to a stratifying shuffled training set and ensured a balanced distribution of the candidates. Applied different preference representations enlightened new insights into the performance of the ML algorithms for rank aggregation.

Representation 1 led to an increase in the performance for Borda winner prediction by 100% accuracy with the XGBooster classifier. The main drawback of Representation 1 is the lack of reduction in time during the training phase and minor gain in memory (data) compression. However, we found that this representation is much more informative for further processing in terms of improved accuracy of supervised algorithms.

On the other hand, Representation 3 and 4 give the best results in predicting Kemeny winners, and it assured the performance on the level 20 times higher than a random prediction, i.e. ca. 85% accuracy with the SGD classifier.

During the error analysis and the model behavior diagnosis we encountered important challenges:

- overfitting problem - we recognized when evaluating estimator performance. The training accuracy was very high (up to 100% of learned examples was explainable by predictive model) and the test accuracy was much worse (for instance in Kemeny's winner case, the extremely insufficient application predicted correctly only 1% test data points)
- non-uniform distribution of labels in the Spotify dataset. This problem was handled by:
  - data augmentation
  - stratified shuffled split into training, validation and test sets
- model enhancement was handled by:
  - tuning the hyper-parameters of the predictive model
  - miss-predictions analysis

Since voting rules are irresolute, we applied a tie-breaking mechanism (i.e. selection as single label the first winner from the complete sequence of winning candidates). We conducted

the verification if the miss-predicted label was included in the first solution that, by implicit assumption, was excluded from the labels.

Those actions demonstrated a good effect on the problem generalization. The experimental study revealed that the obtained results are satisfying. For training we used less than 0.01% of all possible voting profiles for a given number of voters (25) and alternatives (20). We explored ten different ML classification models with various parameters to optimize the final prediction model's performance. Careful selection from a wide range of problem formulation setting led to the identification of the best predictive model for rank aggregation, increasing accuracy of (i) Borda winner with the XGBoost classifier by 100%, (ii) Kemeny winner(s) with the SGD classifier by 85%; and (iii) Dodgson winner(s) with the Gradient Boost classifier by 89%.

Chapter 5

Summary

5.1 Summary of major findings

The main objective of this thesis was to investigate the learnability of the voting rules and evaluate the performance of ML models when predicting the top-alternative in a given aggregated preference rank. We explored experimentally if machine learning algorithms can be used to predict the winner of Borda, Kemeny and Dodgson voting rules, effectively trading computational complexity for (in)accuracy.

Classification models were trained using two datasets: a real-world Spotify dataset and a synthetic dataset, both of profiles of size $N = 20$ alternatives and $V = 25$ voters. We compared ten supervised ML algorithms (among others SVM, XGBoost and GaussianNB) to identify the best final classification model.

We prepared the data for the learning algorithms by applying two pre-processing methods: data transformation and augmentation. Data transformation included engineering feature representation techniques to represent preferences as datasets that could be used to train ML models. Consequently, we factorized, trained and tested four fundamentally different preference representations for three voting rules: Borda, Kemeny and Dodgson. Data augmentation had to be performed due to the biased distribution of the Spotify winner(s)' classes and to optimize of ML models performance.We selected the final classification model based on the function to optimize, namely for maximizing of accuracy and F1-score (evaluation metrics). We applied augmentation and data transformation that led to improvement of prediction accuracy of the **Borda winner(s) with the XGBoost classifier with accuracy by 100 % ; Kemeny winner(s) with the SGD classifier with 85 % and Dodgson winner(s) with the Gradient Boost classifier with 89% accuracy.** Interestingly in Kemeny's winners' prediction case, Representation 3 and 4 were extremely well learnable by SGD classifier.

We empirically demonstrated that the Borda count, as a voting rule with linear dependencies, can be learned and predicted with high accuracy (>95%). Specifically, we reached 100 % accuracy with XGBoost classifier regardless of the Representation it was trained with.

We have been specifically interested in the NP-hard Kemeny and Dodgson voting rules.

The empirical research results for those rules demonstrate that the ensemble learning algorithms are by far better at reducing errors compared to the linear classifiers. Moreover, we identified the XGboost and GradientBoost models as the best performing predictive model and we proved it in successful and accurate predicting rank data.

We gained interesting insights from Kemeny miss-predictions analysis. Kemeny WDP is an example of an irresolute voting rule. Irresolute voting rule is the one that selects more than one winner in a tie. After checking the robustness of the negative result for Kemeny winners, we identified that 83% of original miss-predictions were found in ties, meaning were correct but ignored by the designed lexicographic tie-breaking mechanism. That analysis yielded to better model accuracy: up to 85%. Next, the ensemble algorithm, such as Random Forest, trained in Representation 4 reached 68% of accurate Kemeny's winners' classifications.

Finally, Dodgson's winner prediction is the most complicated voting rule out of the three explored in this research. Machine learning models reached a maximum of 89.47% by the Gradient Boosting model learned in Representation 3.

As a concluding remark, the experimental study revealed that the sequence of the votes in each profile doesn't play any role in contrast to the order of preference given by each vote - which must be kept while being the fundamental assumption for rank aggregation method.

To diagnose the learning and generalization behavior of ML models we used the learning curves - the graphical representations of the changes in learning performance over time in terms of experience. Performance curves (learning curves of model performance) were used to diagnose model dynamics' behavior, precisely: an under- or overfitting. We created dual learning curves for an ML model while learning both the training and validation datasets. Although the synthetic dataset was generated based on uniform distribution of the classes, and the test and validation set was selected with the stratified split, the overfit and underfit (premature learning hold) problem occurred in specific algorithms settings. Rank data problem seemed to be well fitted by a group of ensemble algorithms, such as: XGBoost classifier, AdaBoost and probabilistic algorithms such as the Gaussian Naive Bayes model.

Summarizing, the contributions of this thesis are:

- we provided analyses of winner miss-predictions in ties, in particular for NP-hard voting rules: Kemeny and Dodgson. For Kemeny, for instance, we identified that 83 % of original miss-predictions were found in ties, meaning classification model predicted them correctly but ignored due to designed lexicographic tie-breaking mechanism,

- we have built an easy-to-use training and testing pipeline that can efficiently be reproduced and reused to check the performance of new classification models,

- we converted Kemeny and Dodgson profiles into PrefLib format, which allows to contribute data and help extend the online library for preferences,

- we built easy-to-use web-querying script for high-dimensional dataset (selenium Python) crawling DEMOCRARIX online tool, it can be used for new online tools querying,

- we achieved a high classification accuracy for Borda scored-based rule - 100% this was expectable from the results of Procaccia A.D. (2009),
- we achieved reasonable accuracy for winner(s) predictions of the Kemeny (85 %) and Dodgson (89 %) voting rules,
- we explored ten different ML classification models with different parameters to optimize the final prediction model's performance,
- for training we used less than 0.01% of all possible voting profiles for a given number of voters (25) and alternatives (20), we assume that using more of the profiles for training would make the final classification model more robust and increase its performance,
- we identified four preference representations, as a dataset that ML classifiers used for training/validating/predicting, and we discussed their influence on the performance of the classifiers,
- we used cross-validation testing technique to select the best final classification model.

However, the thesis also pointed out these weaknesses:

- we used real-world Spotify dataset, that size was too small to hit a good level of classifiers accuracy, also in this settings, we easily encountered overfitting problem,
- preference ranks need to be transformed into one of introduced dataset representations to use a (pre-trained) final, classification model and predict the winner,
- the labeling process for NP-hard voting rules, like Kemeny and Dodgson, is very time-consuming but can be done in parallel for different profiles, for instance, in DEMOCRATIX online tool.

5.2 Future work

Apart from the prediction evaluation metrics used in this research - accuracy, confusion matrix and classification report (with which we computed precision, recall, and f-score) - in this setting, it might be worthwhile to look into the receiver operating characteristic (ROC) and AUC curves. An attractive alternative might be using the ranked probability score (RPS) metric, which is a measure of how similar two probability distributions are and is used as a way to evaluate the quality of a probabilistic prediction.

As a concluding remark, we noticed that pairwise comparison is a universal way to investigate the partial preferences, however *rank centrality* might be the interesting direction of future research as it is a simple and intuitive algorithm for rank aggregation itself.

Whereas the issue of computing (approximately) ranking medians has received much attention in the literature, just like statistical modeling of the variability of ranking data,

the generalization ability of practical ranking aggregation methods has not been studied in general probabilistic setup by describing optimal elements and establishing the learning rate bounds for empirical Kemeny ranking medians (Korba et al. (2017)).

Finally, an interesting avenue of research is using unsupervised learning methods instead, which would attempt to cluster the profiles. An interesting insight would be to study whether the profiles will be clusterable by the winner or different profile properties would stand out.

Appendices

A.1 Borda, Kemeny and Dodgson results

In this appendix we included selected results for (i) ten explored classification models in (ii) four preference representations for:
Borda winner(s) predictions:

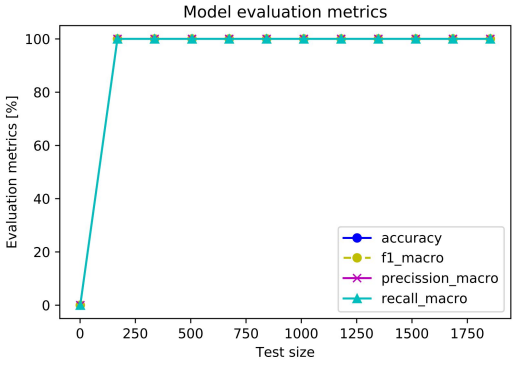
- train and validation learning curves for Representation 1,
- classification report for Representation 1,
- evaluation metric curves for Representation 2.

Kemeny winner(s) predictions:

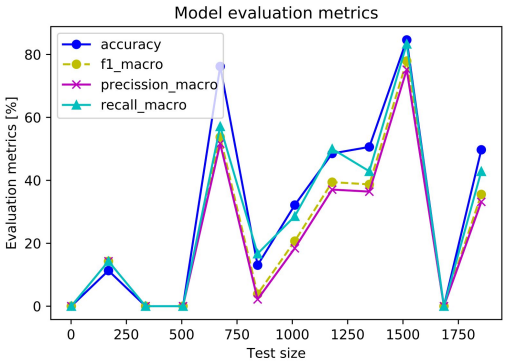
- learning curves for Representation 1, Representation 2 , Representation 3 and Representation 4
- classification report for Representation 1, Representation 2, Representation 3 and Representation 4.

Dodgson winner(s) predictions:

- learning curves for Representation 1, Representation 2, Representation 3 and Representation 4,
- classification report for Representation 1, Representation 2, Representation 3 and Representation 4.

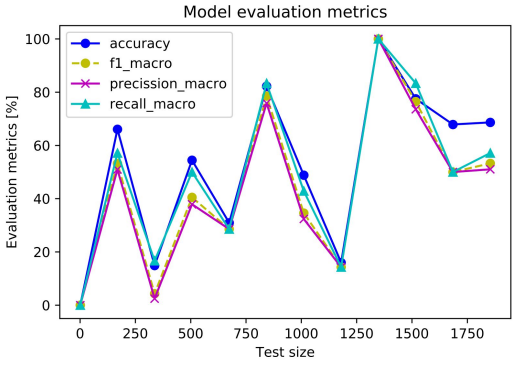


(a) XGBoost[a] classifier.

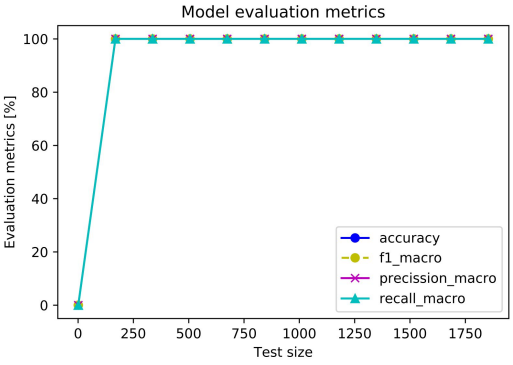


(b) Decision tree classifier.

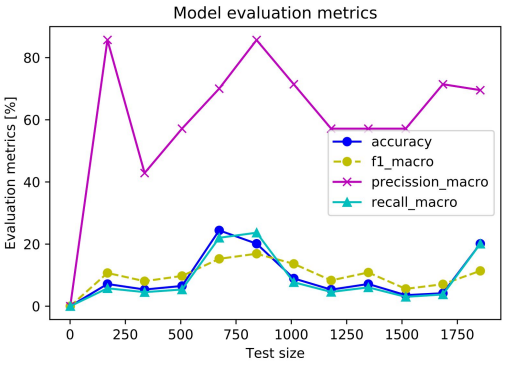
[a] eXtreme Gradient Boosting



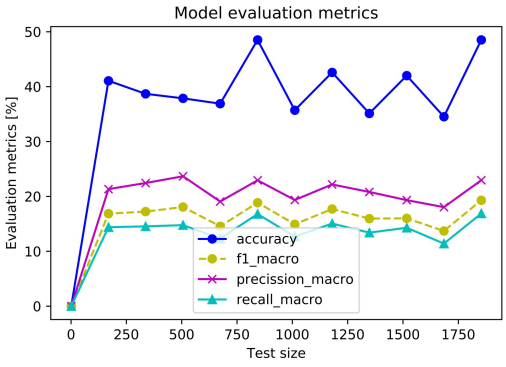
(c) Random forest classifier.



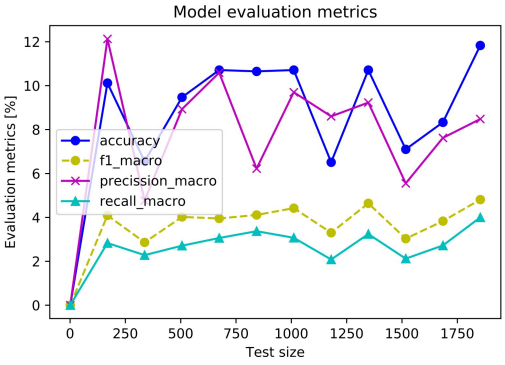
(d) Random forest classifier.



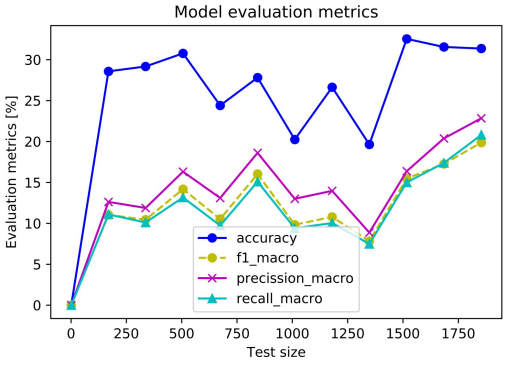
(e) Support Vector classifier with RBF kernel.



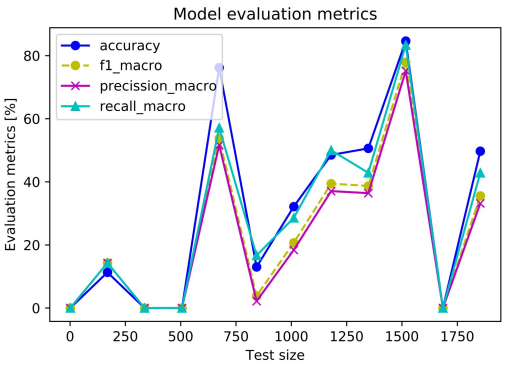
(f) Support Vector classifier with linear kernel.



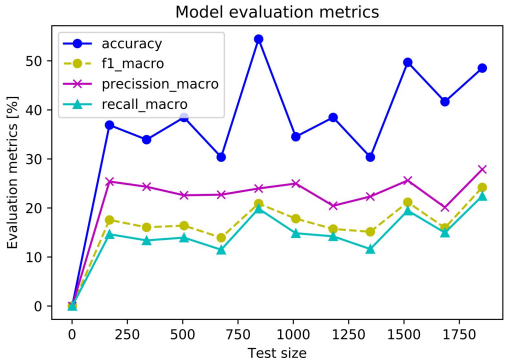
(g) AdaBoost classifier.



(h) Multi-layer perceptron classifier.



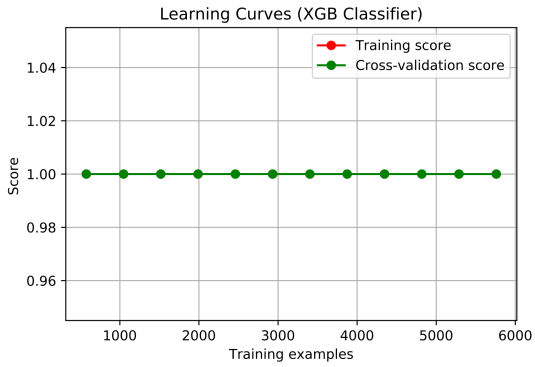
(i) Ridge classifier.



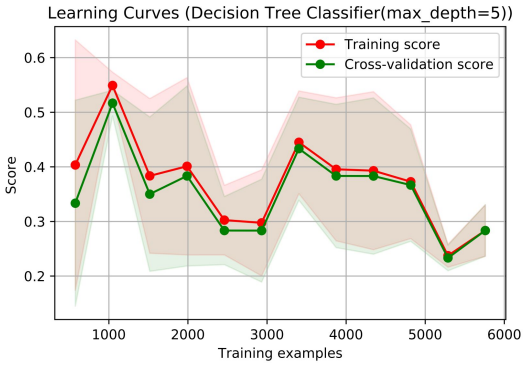
(j) Linear SVM with SGD[a] training.

[a] stochastic gradient descent (SGD) learning

Figure A.1: Model evaluation metrics for Representation 2.

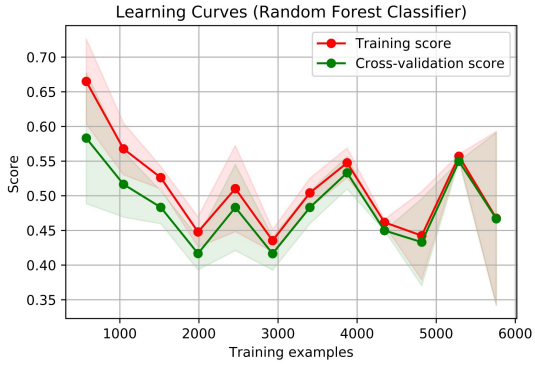


(a) XGBoost[a] classifier.

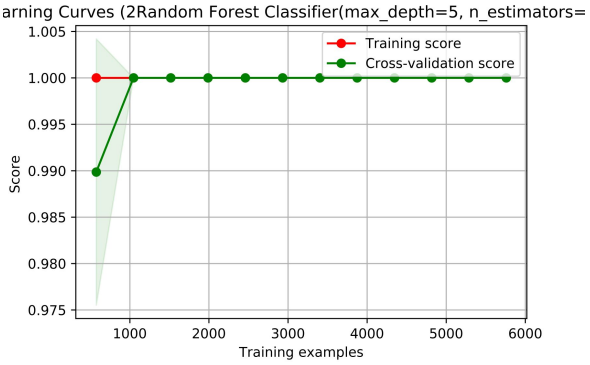


(b) Decision Tree classifier.

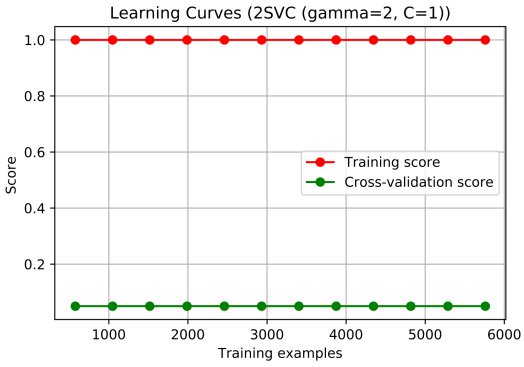
[a] eXtreme Gradient Boosting



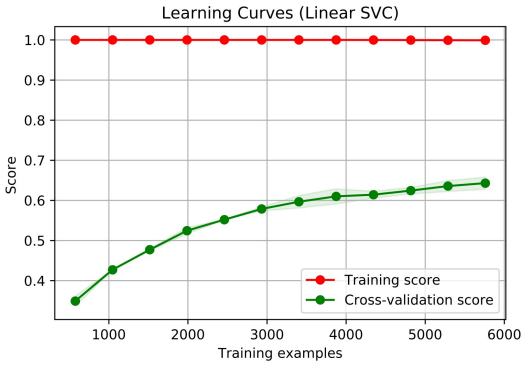
(c) Random Forest classifier.



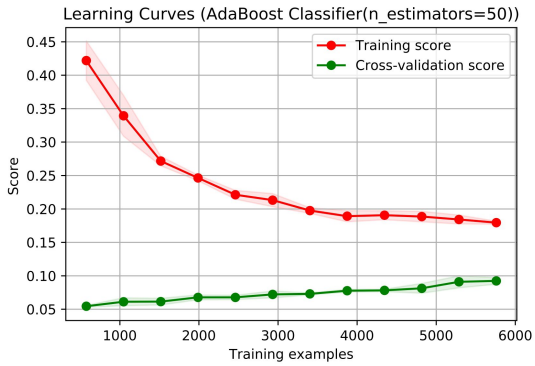
(d) Random Forest classifier.



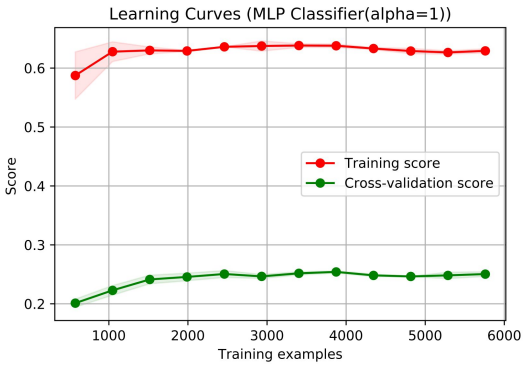
(e) Support Vector classifier with RBF kernel.



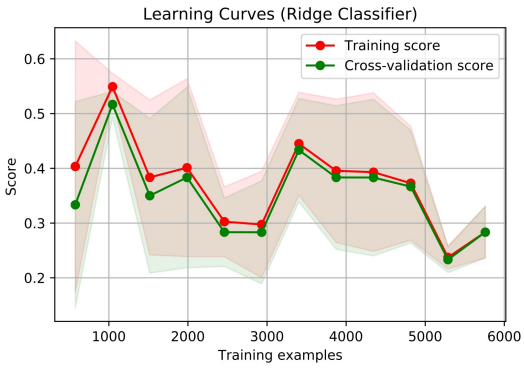
(f) Support Vector classifier with linear kernel.



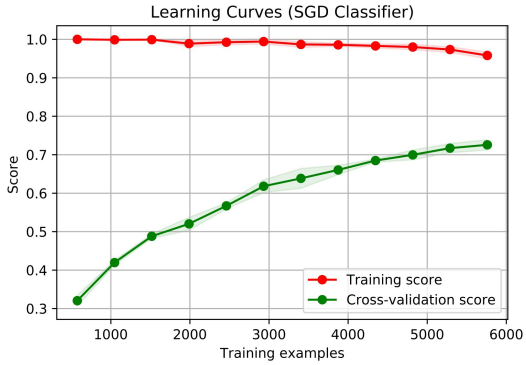
(g) AdaBoost classifier.



(h) MLP classifier.



(i) Ridge classifier.

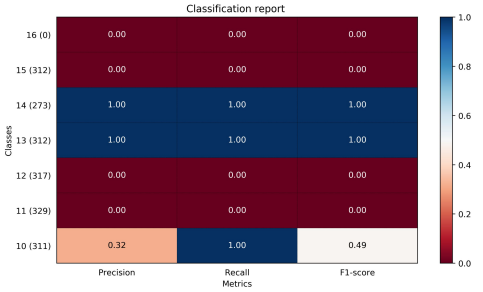


(j) Linear SVM with SGD[a] training.

[a] stochastic gradient descent (SGD) learning

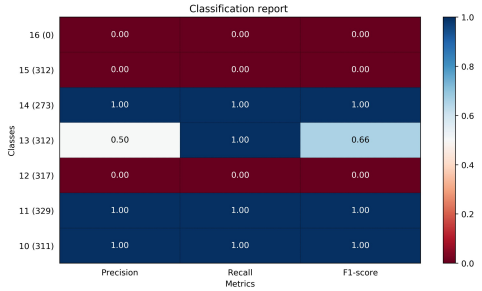
Figure A.2: Train and validation learning curves for models learned in Representation 2.

(a) XGBoost[a] classifier.

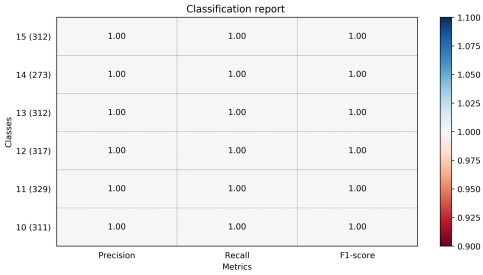


(b) Decision tree classifier.

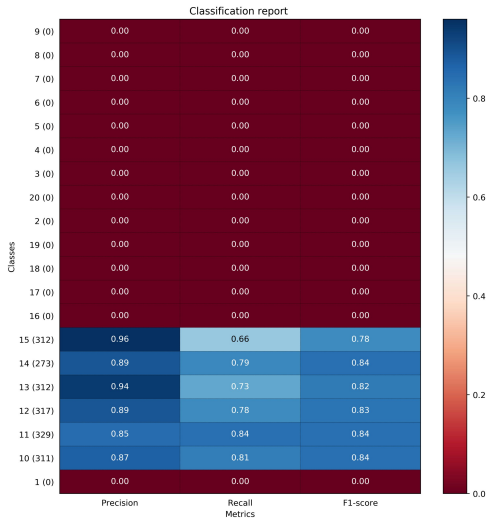
[a] eXtreme Gradient Boosting



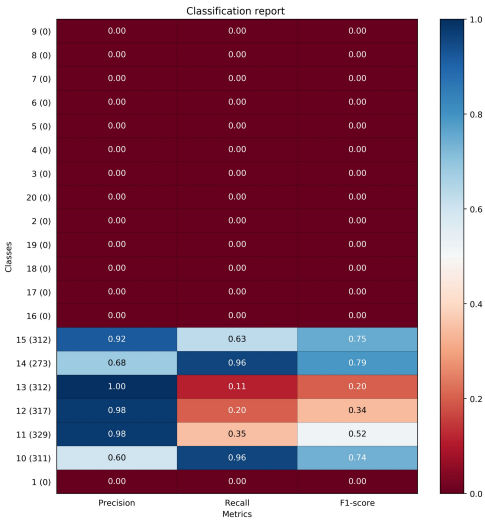
(c) Random forest classifier.



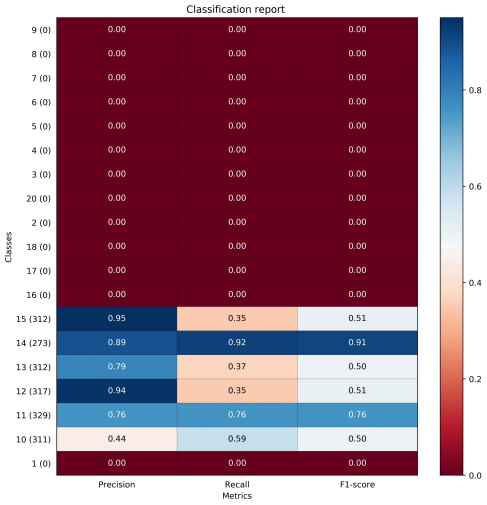
(d) Random forest classifier.



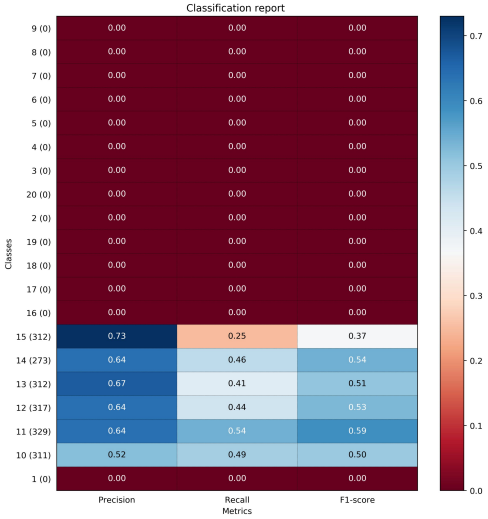
(e) Support Vector classifier with RBF kernel.



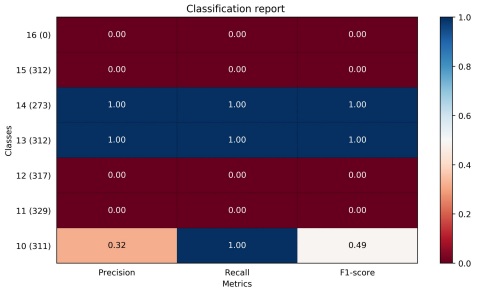
(f) Support vector classifier with linear kernel.



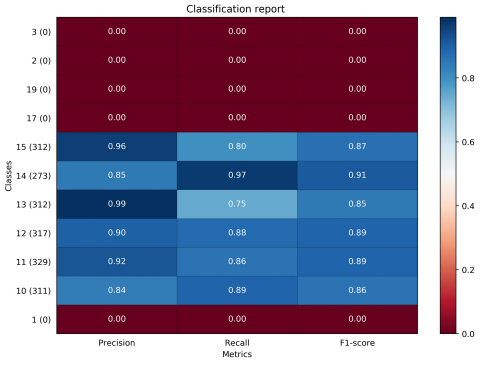
(g) AdaBoost classifier.



(h) Multi-layer perceptron classifier.



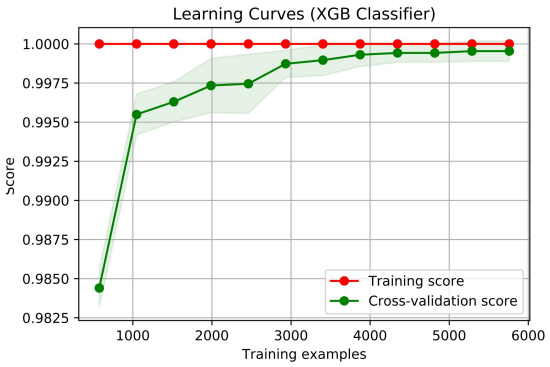
(i) Ridge classifier.



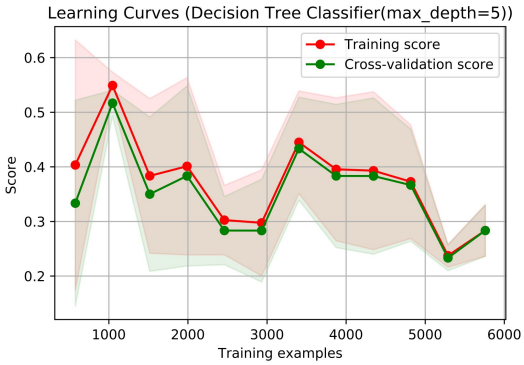
(j) Linear SVM with SGD[a] training.

[a] stochastic gradient descent (SGD) learning

Figure A.3: Classification rapport for models learned in Representation 1.

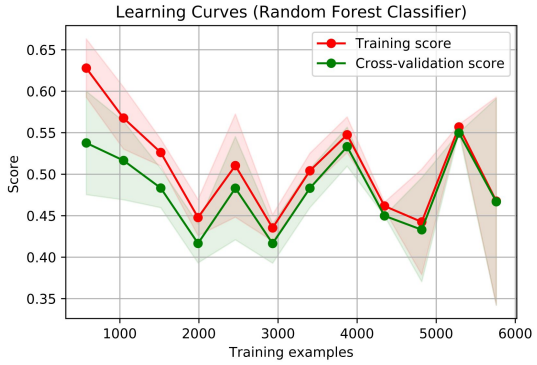


(a) XGBoost[a] classifier.

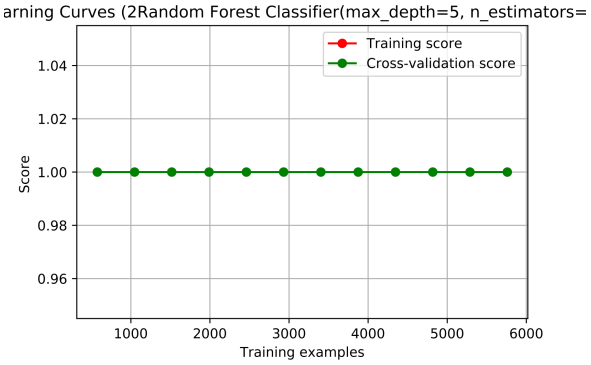


(b) Decision tree classifier.

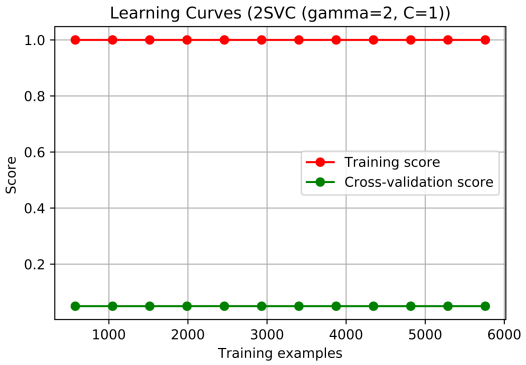
[a] eXtreme Gradient Boosting



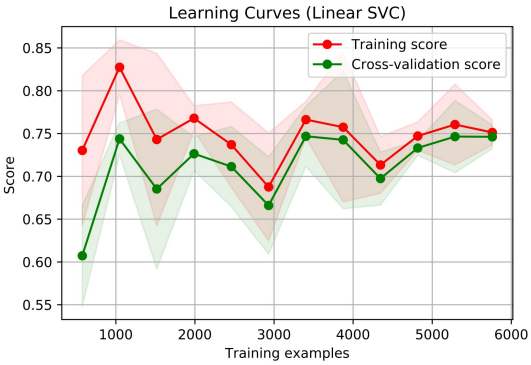
(c) Random forest classifier.



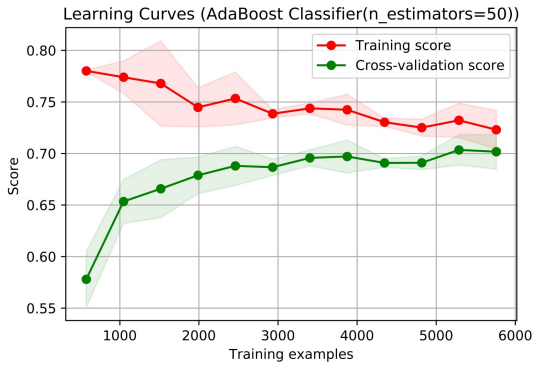
(d) Random forest classifier.



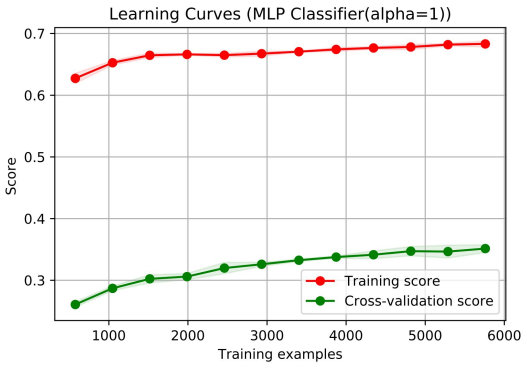
(e) Support Vector classifier with RBF kernel.



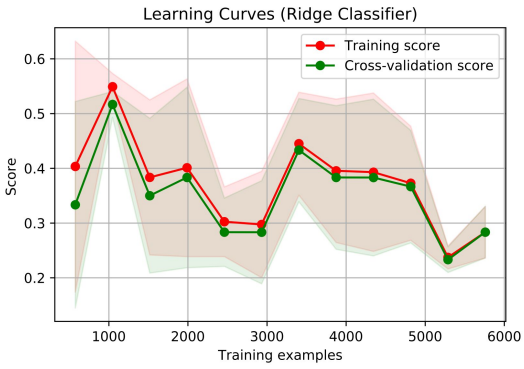
(f) Support Vector classifier with linear kernel.



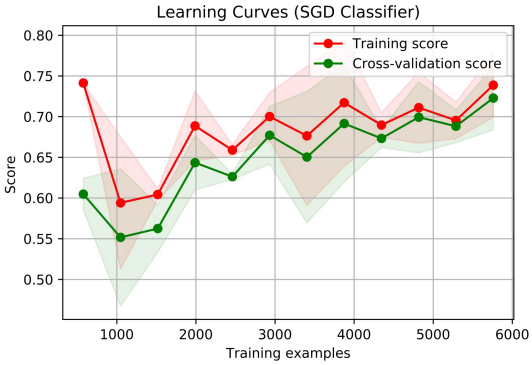
(g) AdaBoost classifier.



(h) Multi-layer perceptron classifier.



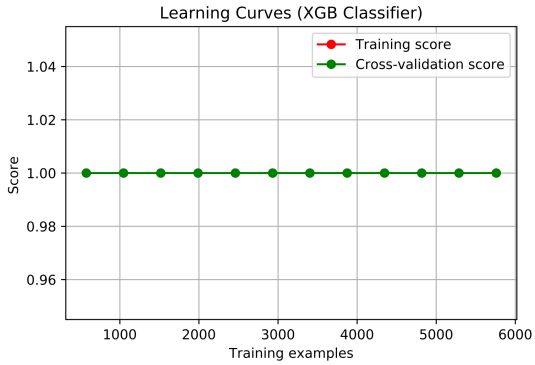
(i) Ridge classifier.



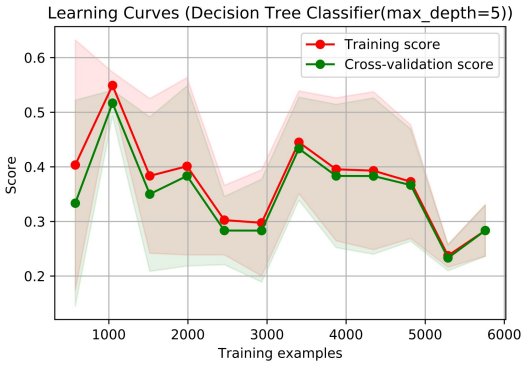
(j) Linear SVM with SGD[a] training.

[a] stochastic gradient descent (SGD) learning

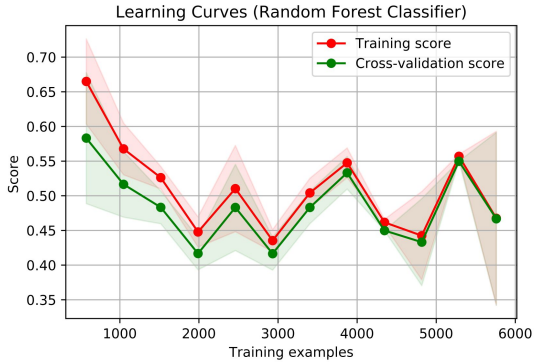
Figure A.4: Train and validation learning curves for models learned in Representation 1.



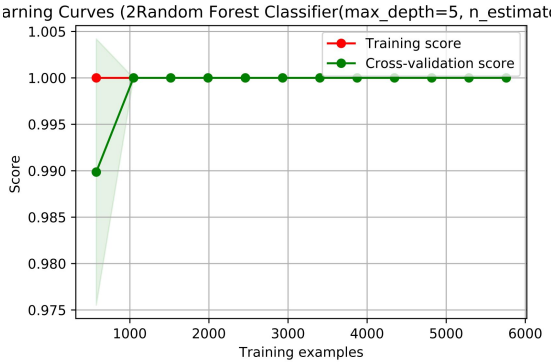
(a) XGBoost classifier.



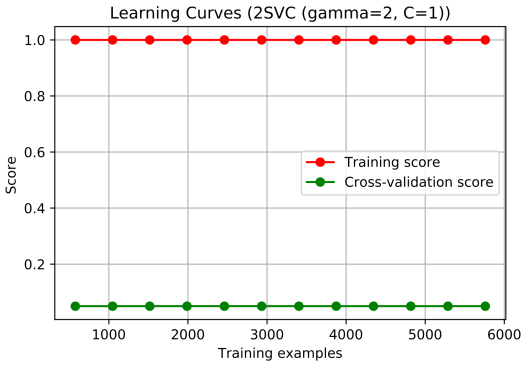
(b) Decision Tree classifier.



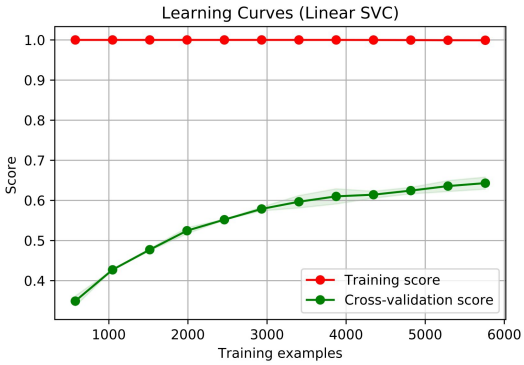
(c) Random Forest classifier.



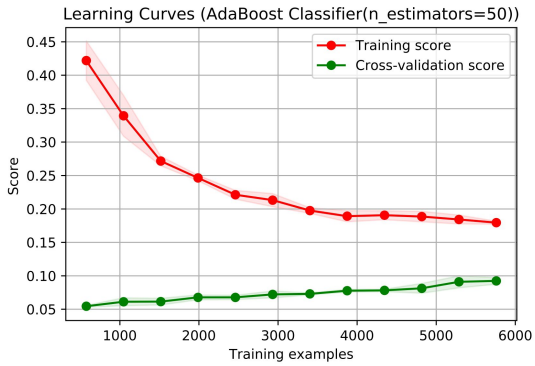
(d) Random Forest classifier.



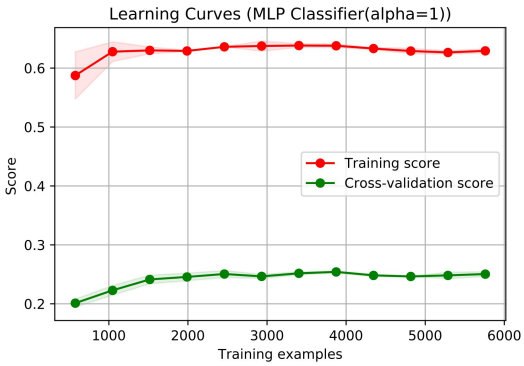
(e) Support Vector classifier with RBF kernel.



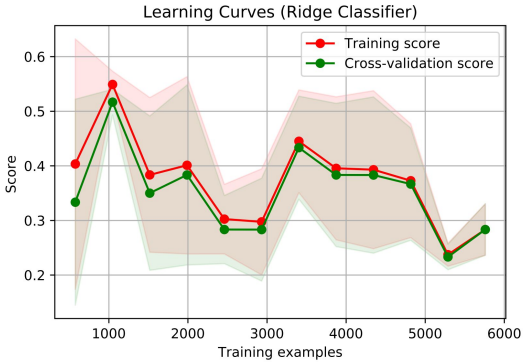
(f) Support Vector classifier with linear kernel.



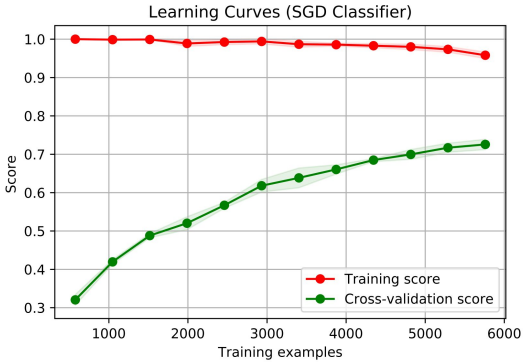
(g) AdaBoost classifier.



(h) MLP classifier.



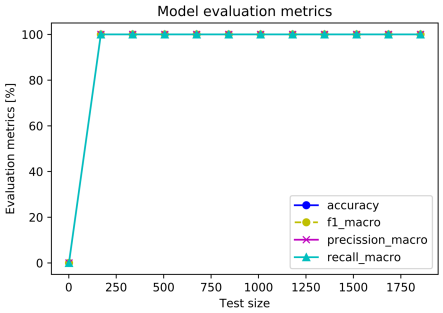
(i) Ridge classifier.



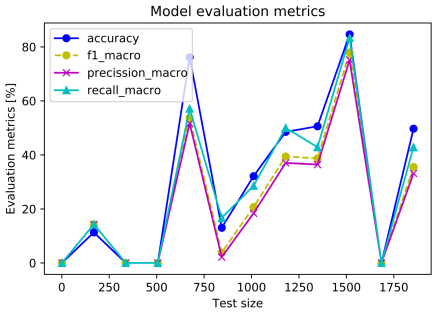
(j) Linear SVM with SGD[a] training.

[a] stochastic gradient descent (SGD) learning

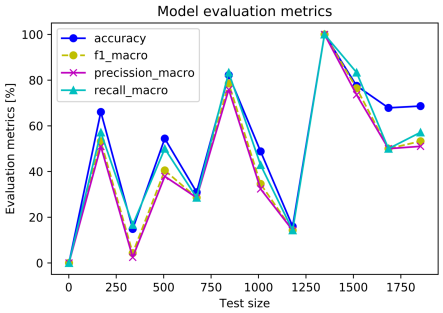
Figure A.5: Train and validation learning curves for models learned in Representation 2.



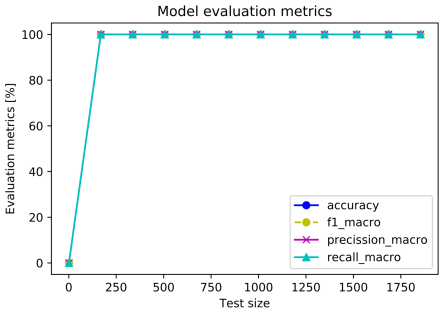
(a) XGBoost classifier.



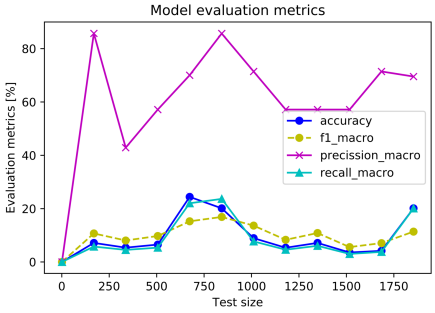
(b) Decision tree classifier.



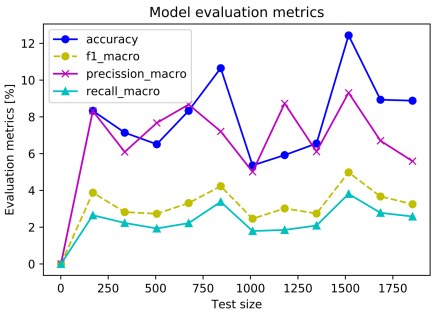
(c) Random forest classifier.



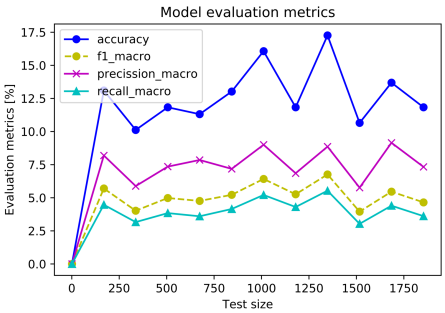
(d) Random forest classifier.



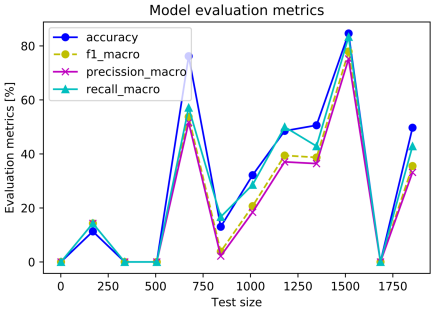
(e) Support Vector classifier with RBF kernel.



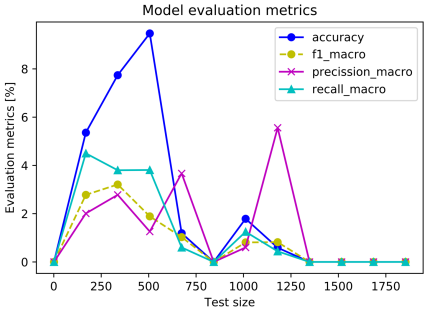
(f) AdaBoost classifier.



(g) MLP classifier.

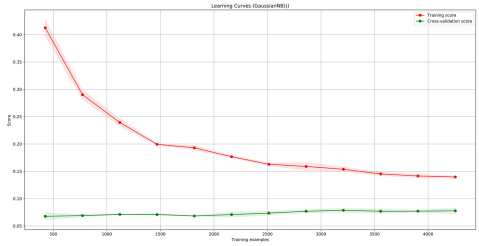


(h) Ridge classifier.

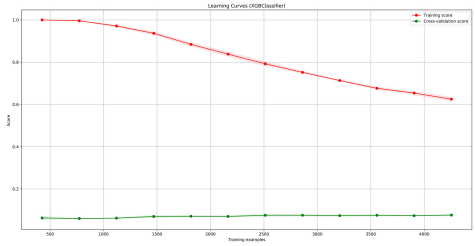


(i) Linear SVM with SGD

Figure A.6: Evaluation metrics (Representation 4|Borda winner(s) predictions).

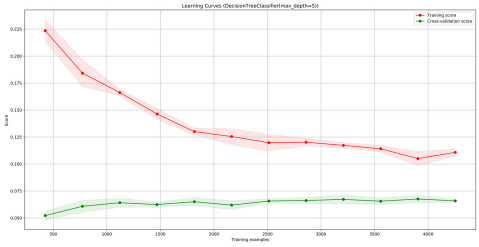


(a) Gaussian Naive Bayes classifier.

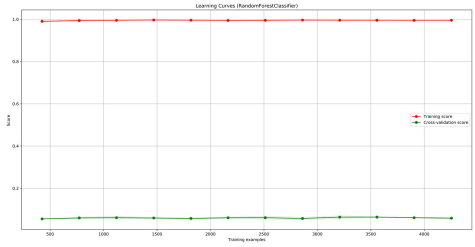


(b) XGBoost[a] classifier.

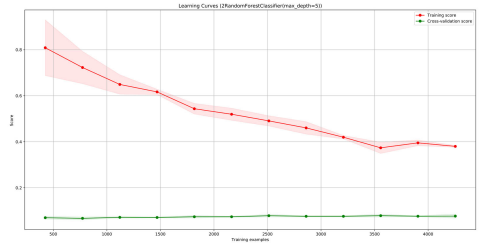
[a] eXtreme Gradient Boosting



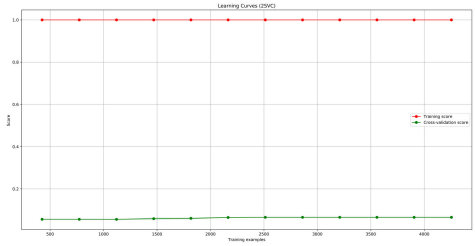
(c) Decision tree classifier.



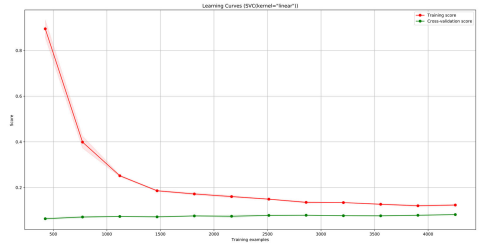
(d) Random forest classifier.



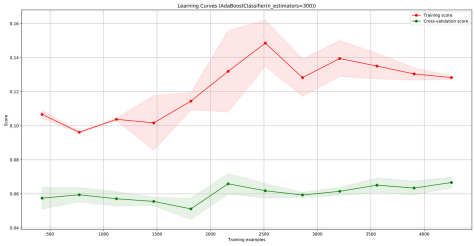
(e) Random forest classifier.



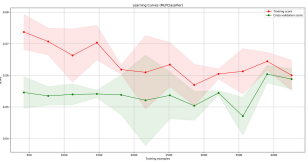
(f) Support Vector classifier with RBF kernel.



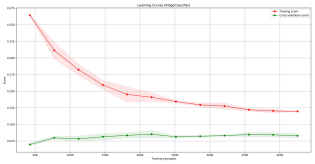
(g) Support Vector classifier with linear kernel.



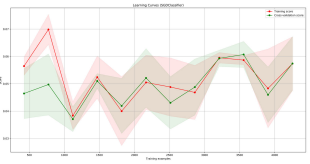
(h) AdaBoost classifier.



(i) MLP classifier.

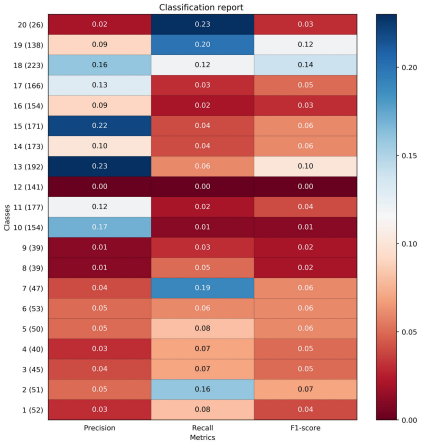


(j) Ridge classifier.

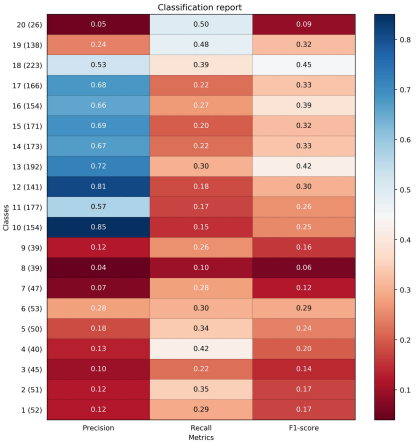


(k) Linear SVM with SGD

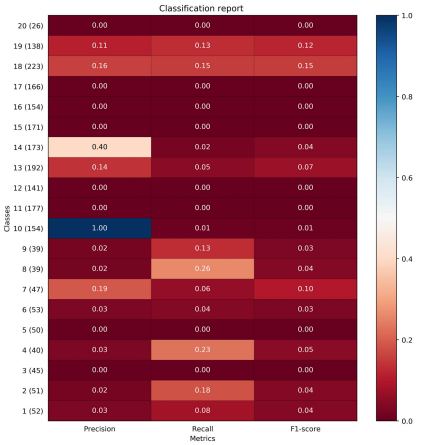
Figure A.7: Train and validation learning curves (Representation 1|Kemeny winner(s) predictions).



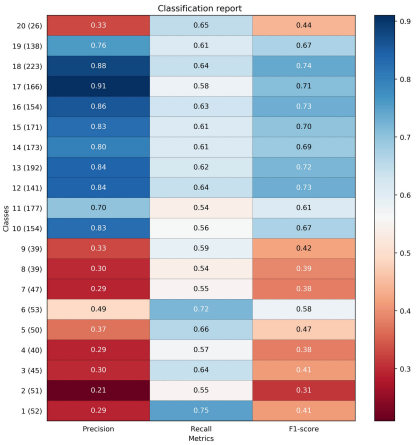
(a) Gaussian Naive Bayes classifier.



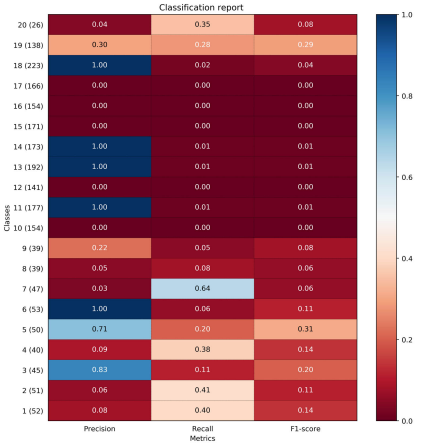
(b) XGBoost classifier.



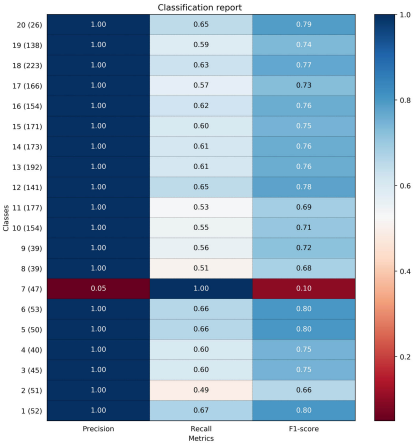
(c) Decision tree classifier.



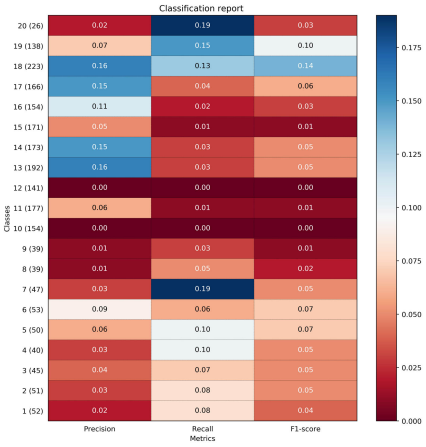
(d) Random forest classifier.



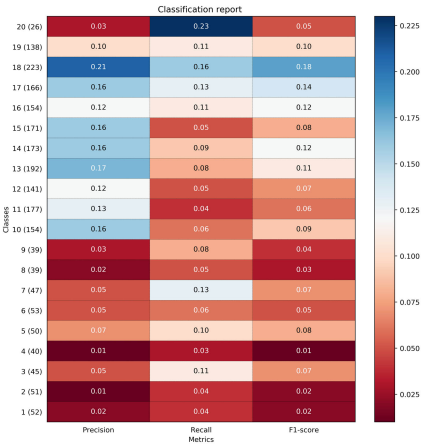
(e) Random forest classifier.



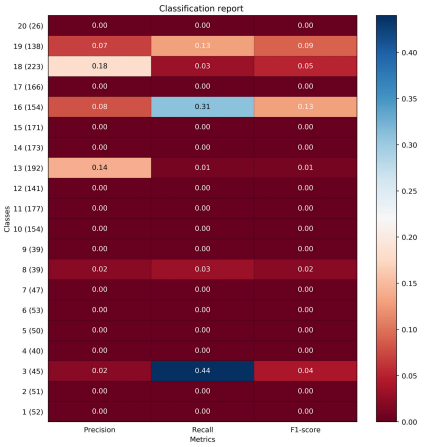
(f) Support Vector classifier with RBF kernel.



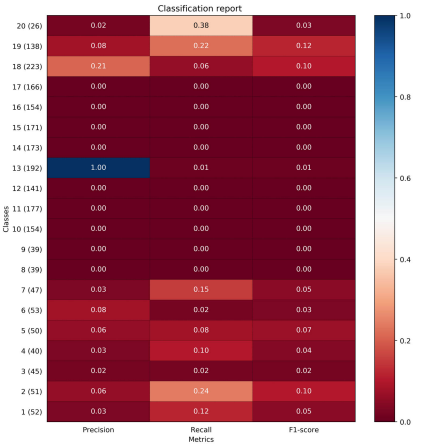
(g) Support Vector classifier with linear kernel.



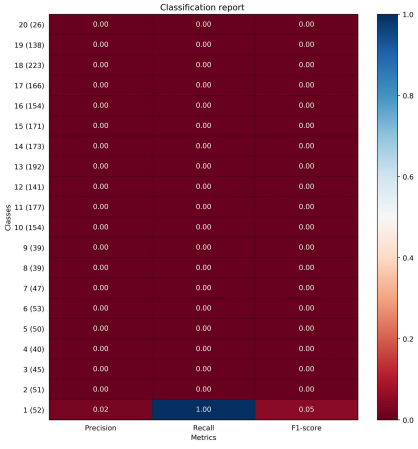
(h) AdaBoost classifier.



(i) MLP classifier.

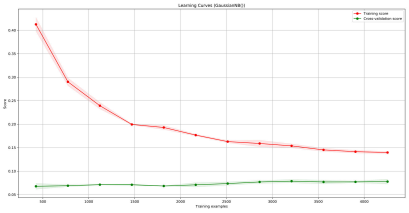


(j) Ridge classifier.

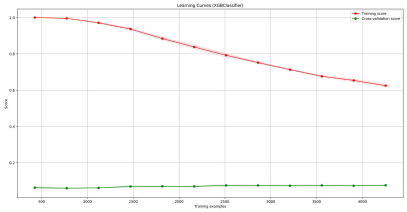


(k) Linear SVM with SGD

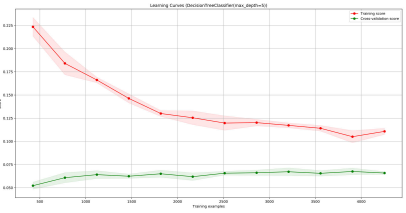
Figure A.8: Classification report (Representation 1|Kemeny winner(s) predictions).



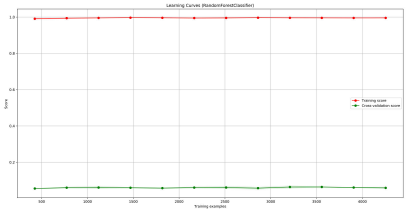
(a) Gaussian Naive Bayes classifier.



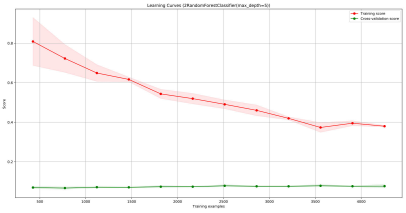
(b) XGBoost classifier.



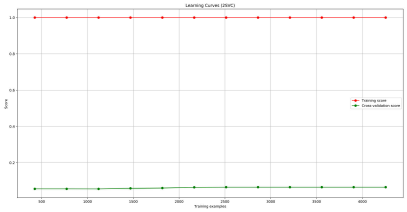
(c) Decision tree classifier.



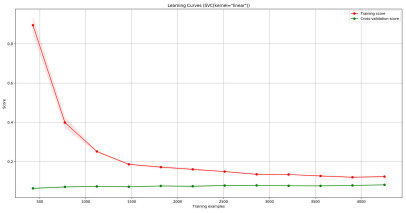
(d) Random forest classifier.



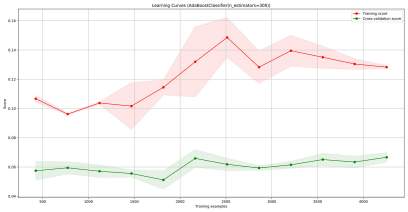
(e) Random forest classifier.



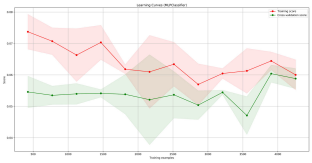
(f) Support Vector classifier with RBF kernel.



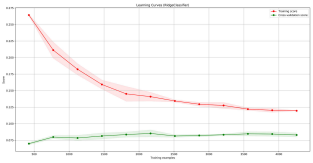
(g) Support Vector classifier with linear kernel.



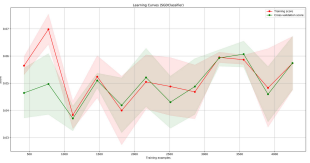
(h) AdaBoost classifier.



(i) MLP classifier.

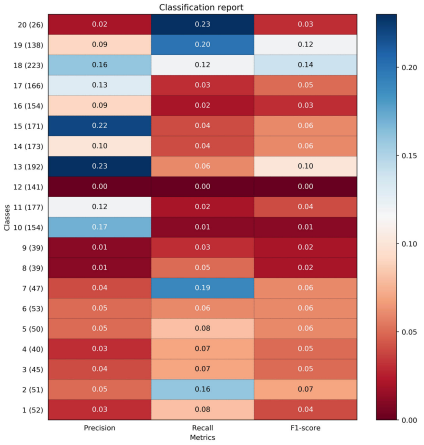


(j) Ridge classifier.

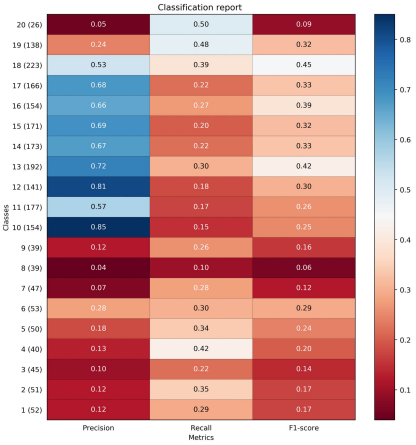


(k) Linear SVM with SGD

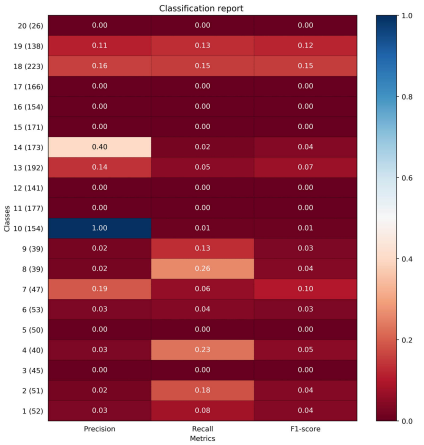
Figure A.9: Train and validation learning curves (Representation 2|Kemeny winner(s) predictions).



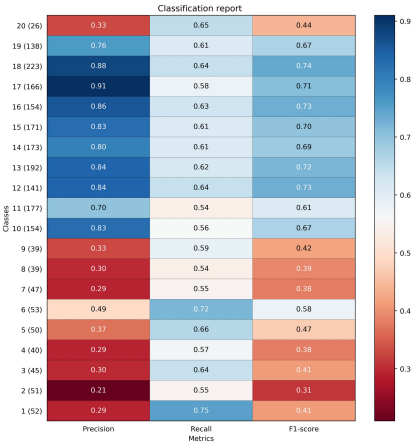
(a) Gaussian Naive Bayes classifier.



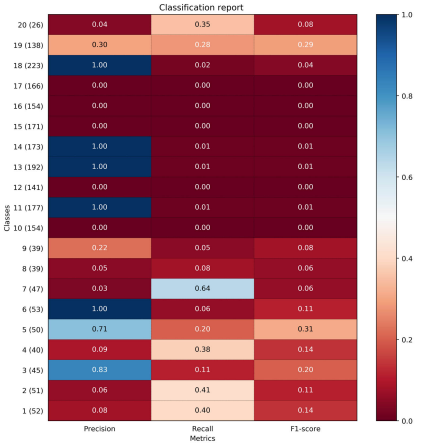
(b) XGBoost classifier.



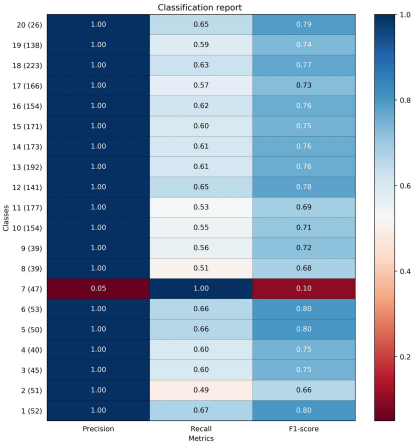
(c) Decision tree classifier.



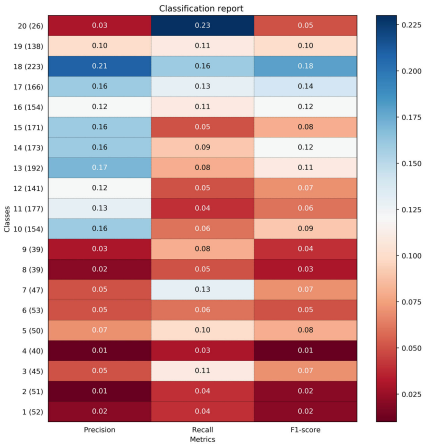
(d) Random forest classifier.



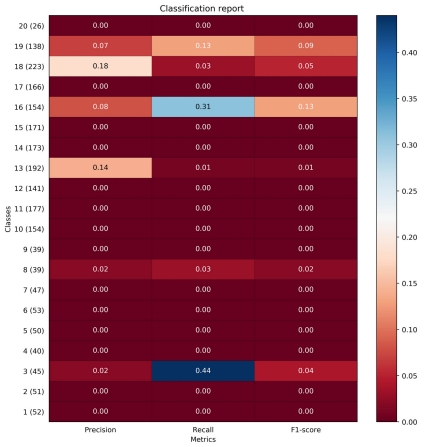
(e) Random forest classifier.



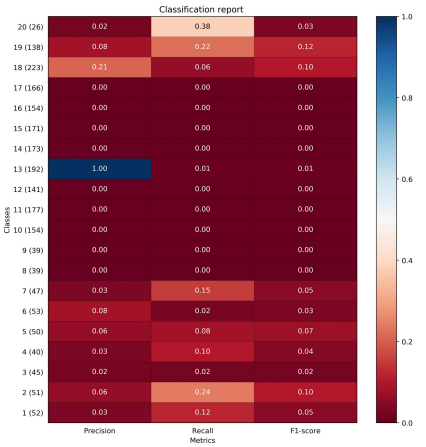
(f) Support Vector classifier with RBF kernel.



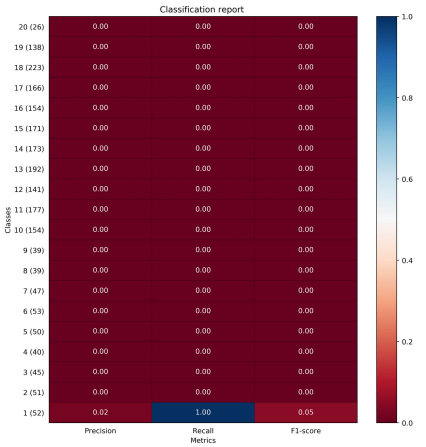
(g) AdaBoost classifier.



(h) MLP classifier.

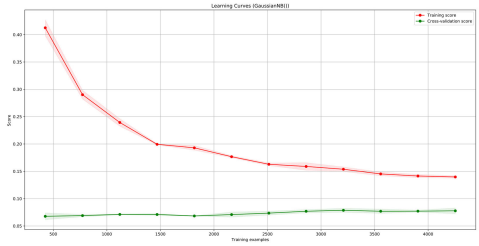


(i) Ridge classifier.

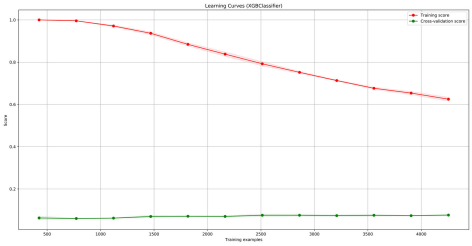


(j) Linear SVM with SGD

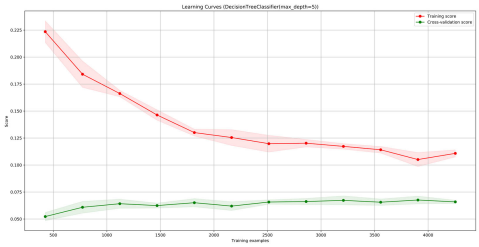
Figure A.10: Classification report (Representation 2|Kemeny winner(s) predictions).



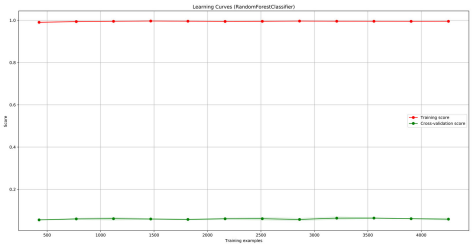
(a) Gaussian Naive Bayes classifier.



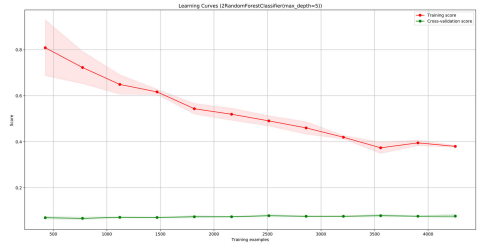
(b) XGBoost classifier.



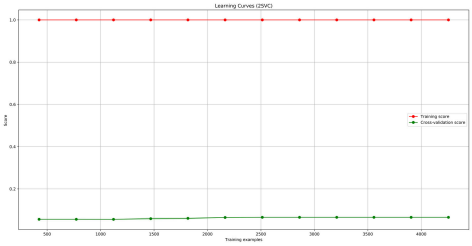
(c) Decision tree classifier.



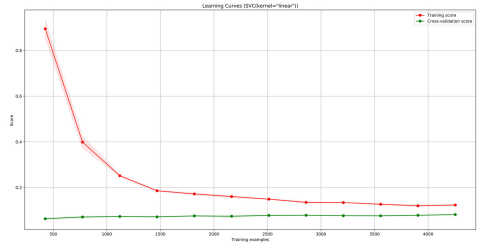
(d) Random forest classifier.



(e) Random forest classifier.

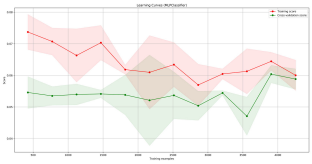


(f) Support Vector classifier with RBF kernel.

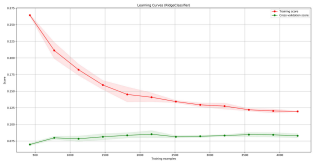


(g) Support Vector classifier with linear kernel.

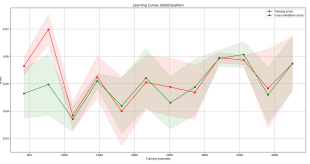
(h) AdaBoost classifier.



(i) MLP classifier.

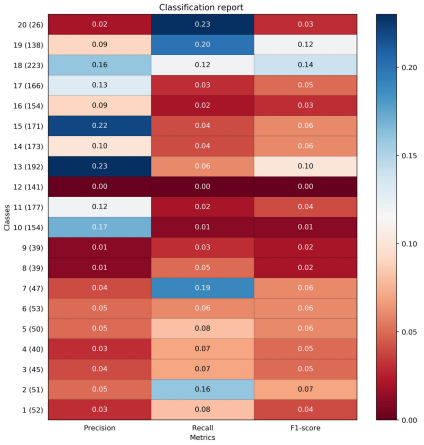


(j) Ridge classifier.

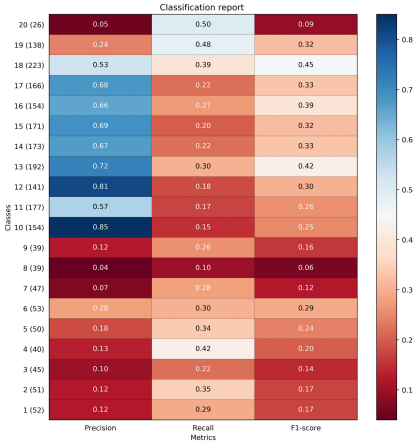


(k) Linear SVM with SGD

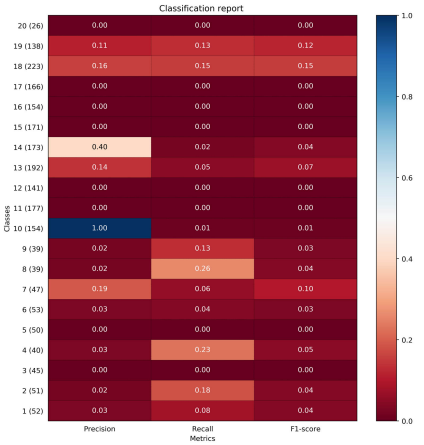
Figure A.11: Train and validation learning curves (Representation 3|Kemeny winner(s) predictions).



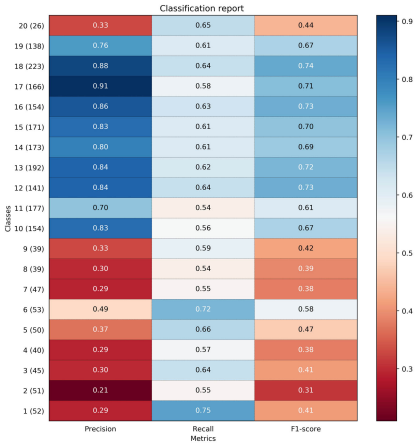
(a) Gaussian Naive Bayes classifier.



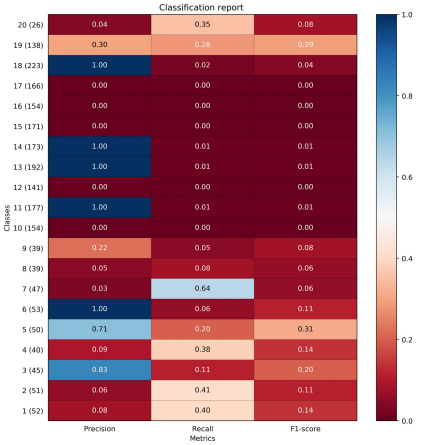
(b) XGBoost classifier.



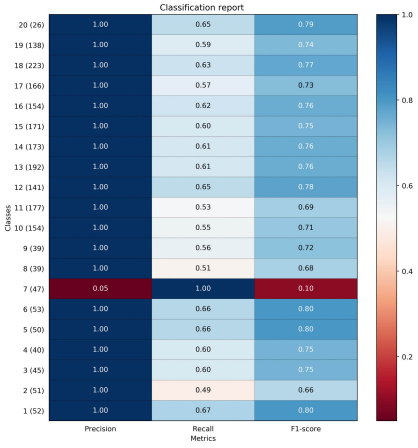
(c) Decision tree classifier.



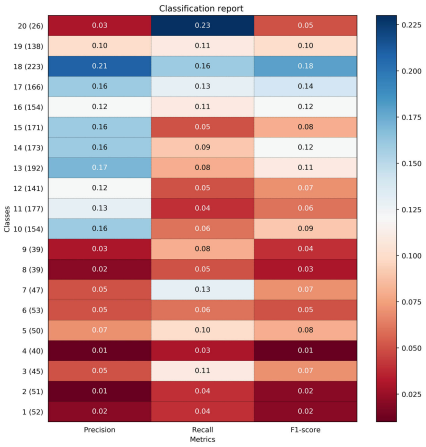
(d) Random forest classifier.



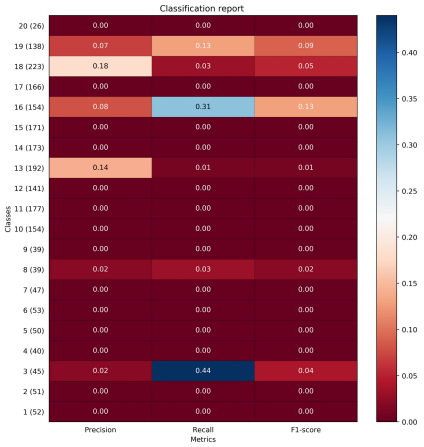
(e) Random forest classifier.



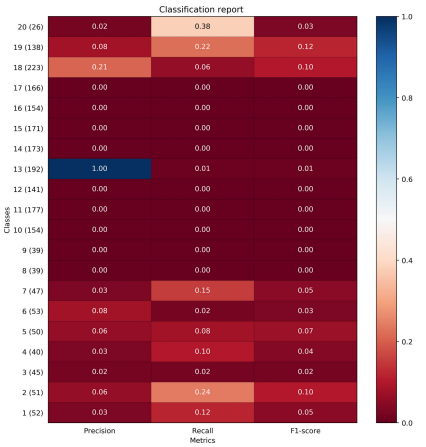
(f) Support Vector classifier with RBF kernel.



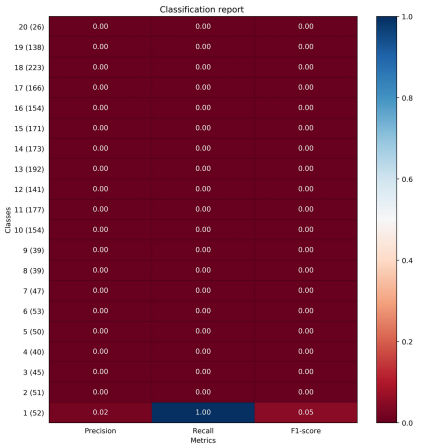
(g) AdaBoost classifier.



(h) MLP classifier.

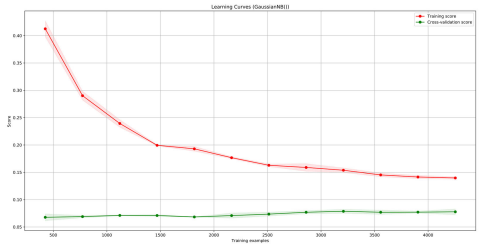


(i) Ridge classifier.

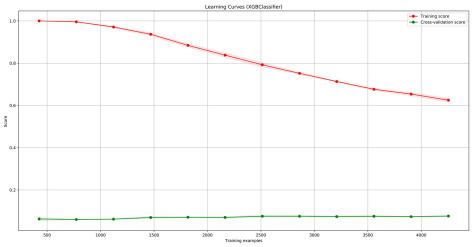


(j) Linear SVM with SGD

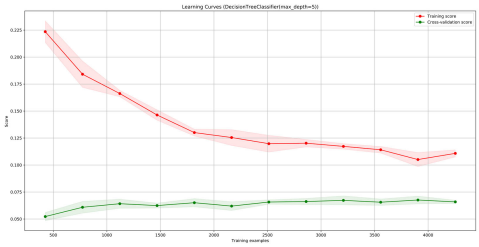
Figure A.12: Classification report (Representation 3|Kemeny winner(s) predictions).



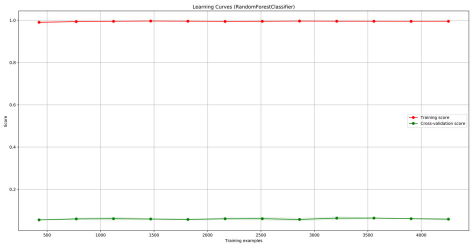
(a) Gaussian Naive Bayes classifier.



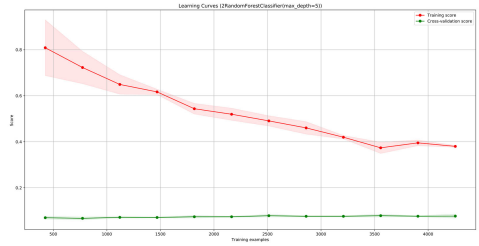
(b) XGBoost classifier.



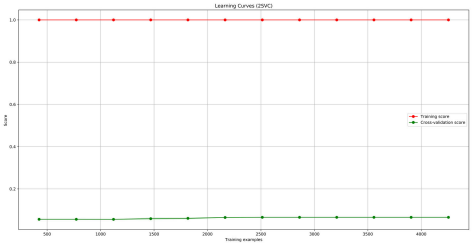
(c) Decision tree classifier.



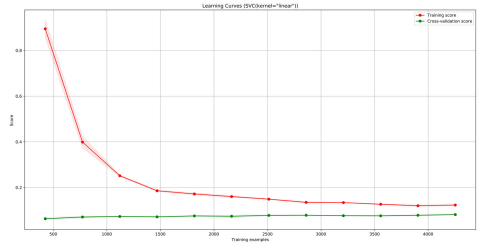
(d) Random forest classifier.



(e) Random forest classifier.

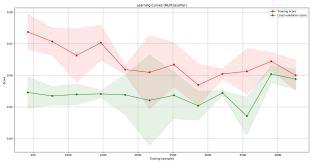


(f) Support Vector classifier with RBF kernel.

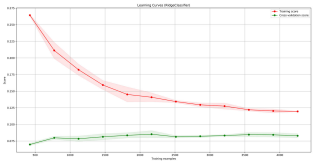


(g) Support Vector classifier with linear kernel.

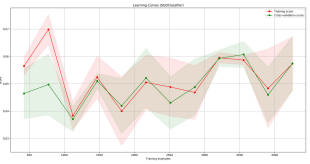
(h) AdaBoost classifier.



(i) MLP classifier.

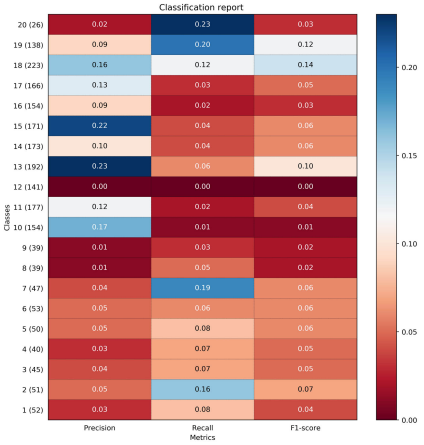


(j) Ridge classifier.

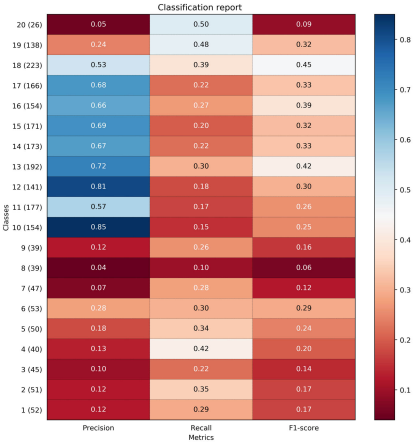


(k) Linear SVM with SGD

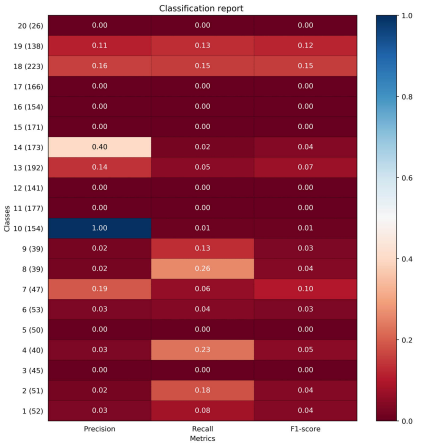
Figure A.13: Train and validation learning curves (Representation 4|Kemeny winner(s) predictions).



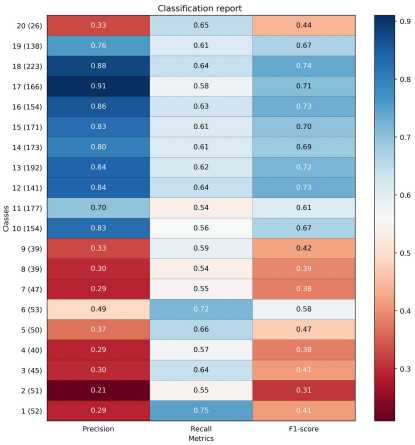
(a) Gaussian Naive Bayes classifier.



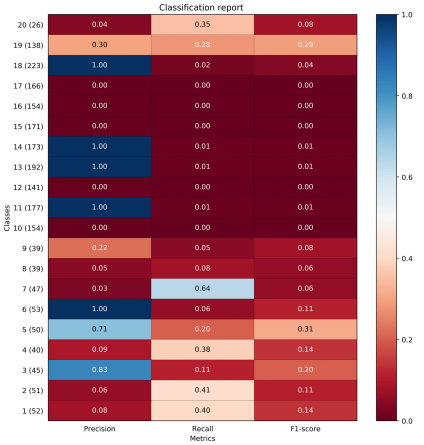
(b) XGBoost classifier.



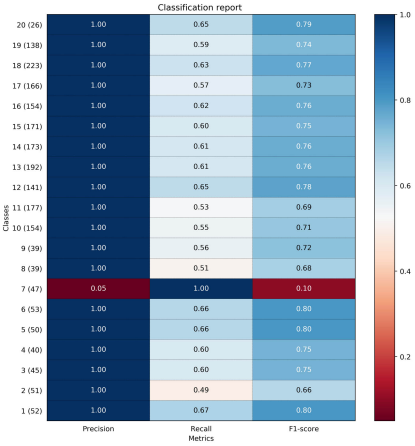
(c) Decision tree classifier.



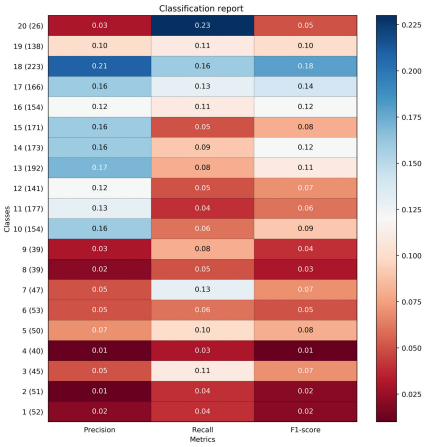
(d) Random forest classifier.



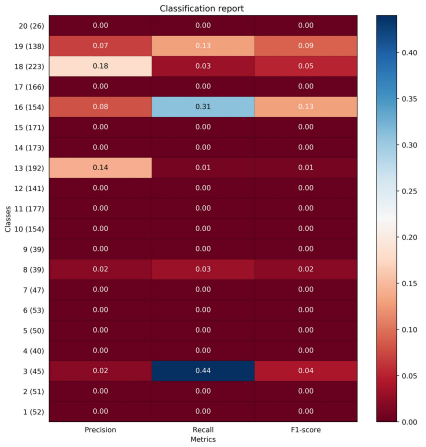
(e) Random forest classifier.



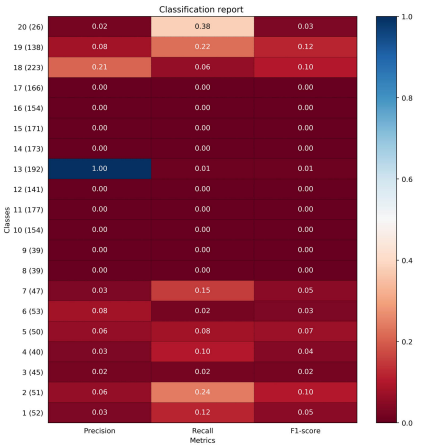
(f) Support Vector classifier with RBF kernel.



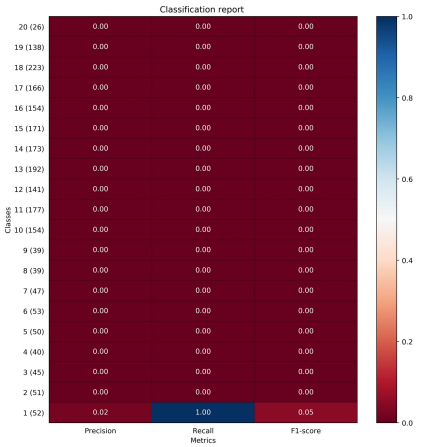
(g) AdaBoost classifier.



(h) MLP classifier.

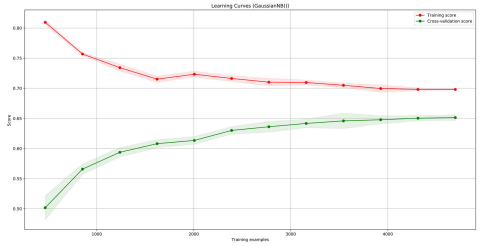


(i) Ridge classifier.

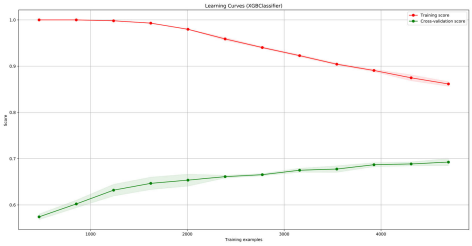


(j) Linear SVM with SGD

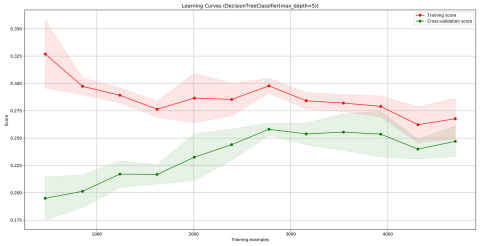
Figure A.14: Classification report (Representation 4|Kemeny winner(s) predictions).



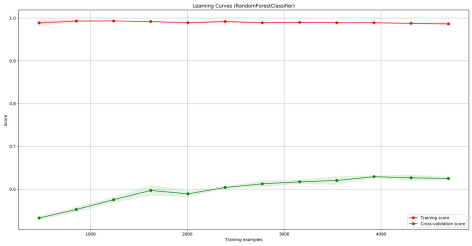
(a) Gaussian Naive Bayes classifier.



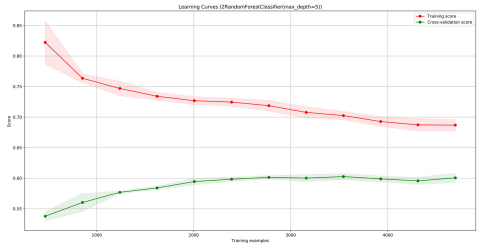
(b) XGBoost classifier.



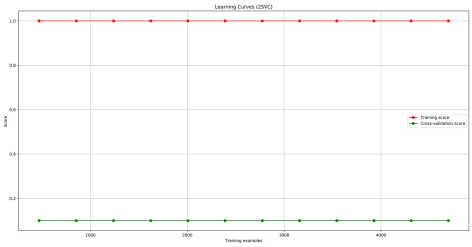
(c) Decision tree classifier.



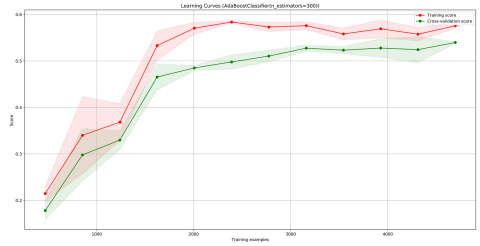
(d) Random forest classifier.



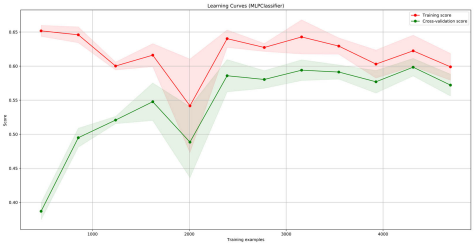
(e) Random forest classifier.



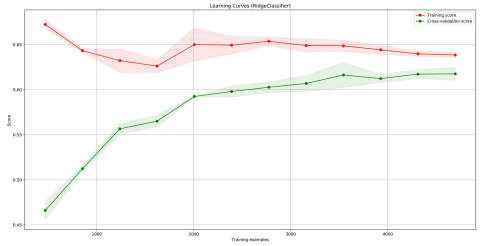
(f) Support Vector classifier with RBF kernel.



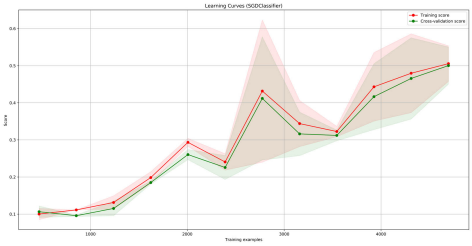
(g) AdaBoost classifier.



(h) MLP classifier.

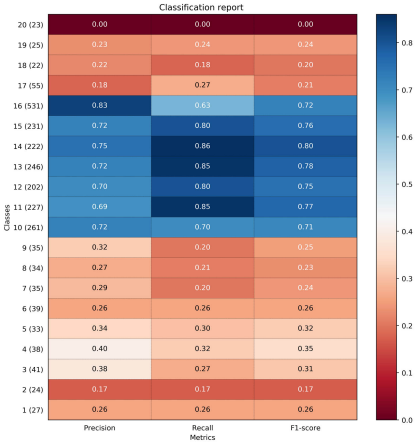


(i) Ridge classifier.

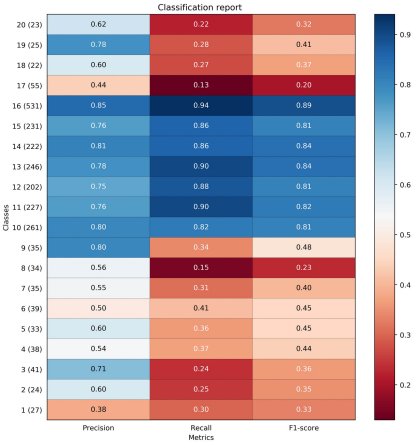


(j) Linear SVM with SGD

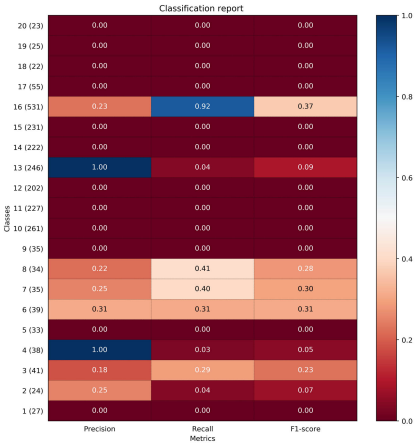
Figure A.15: Train and validation learning curves (Representation 1|Dodgson winner(s) predictions).



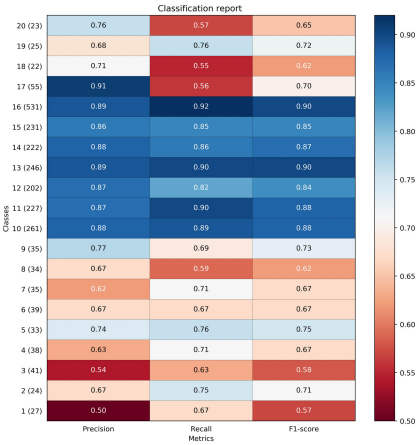
(a) Gaussian Naive Bayes classifier.



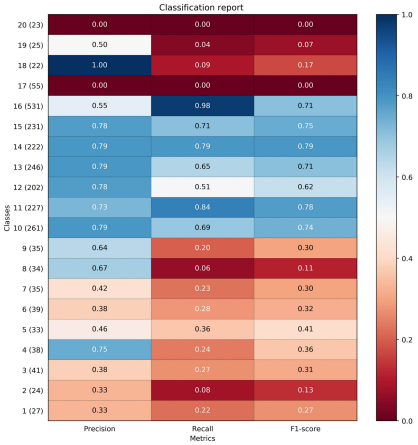
(b) XGBoost classifier.



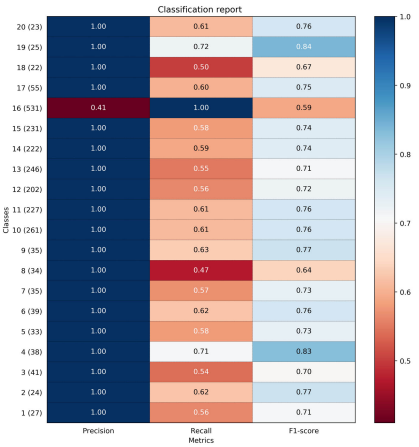
(c) Decision tree classifier.



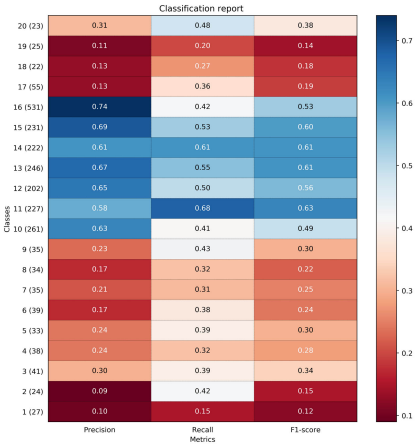
(d) Random forest classifier.



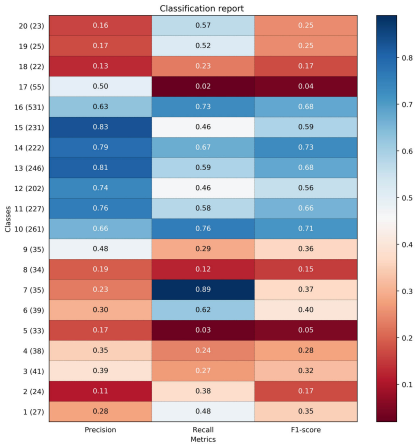
(e) Random forest classifier.



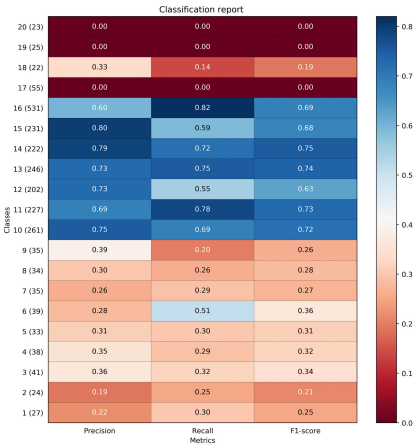
(f) Support Vector classifier with RBF kernel.



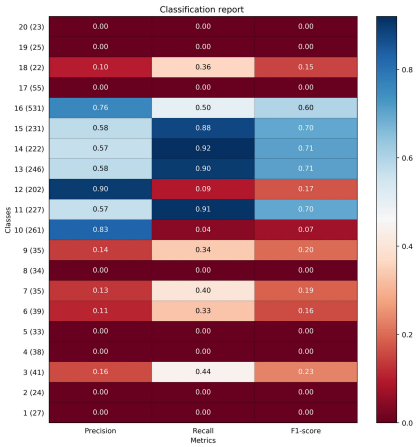
(g) AdaBoost classifier.



(h) MLP classifier.

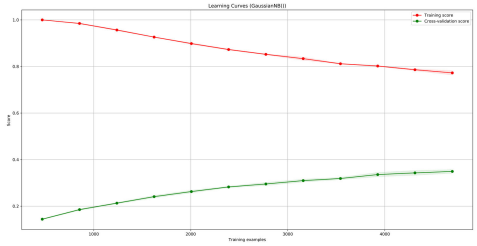


(i) Ridge classifier.

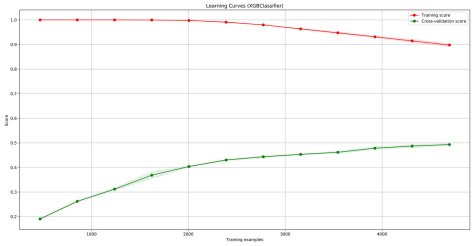


(j) Linear SVM with SGD

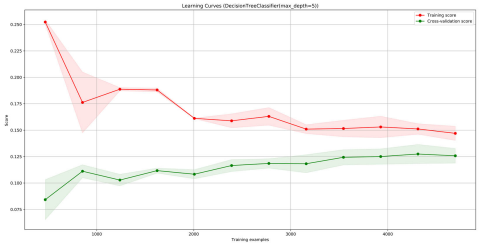
Figure A.16: Classification report (Representation 1|Dodgson winner(s) predictions).



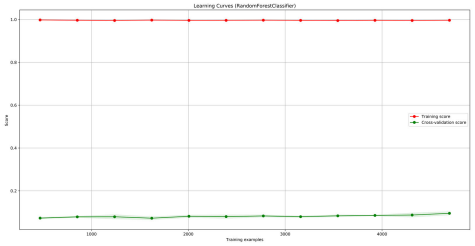
(a) Gaussian Naive Bayes classifier.



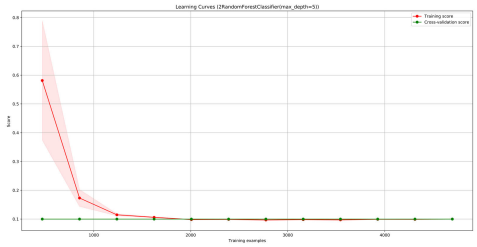
(b) XGBoost classifier.



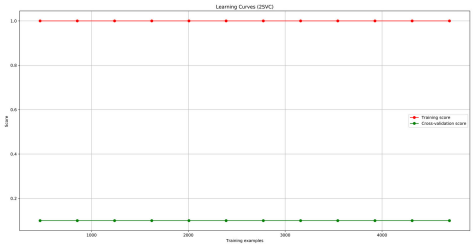
(c) Decision tree classifier.



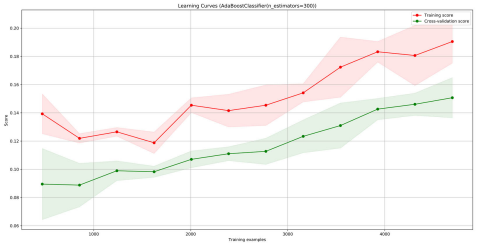
(d) Random forest classifier.



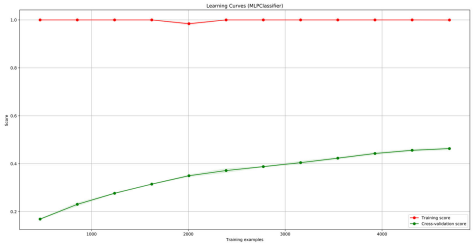
(e) Random forest classifier.



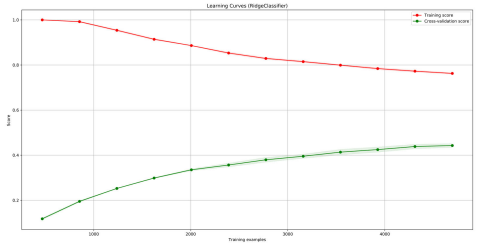
(f) Support Vector classifier with RBF kernel.



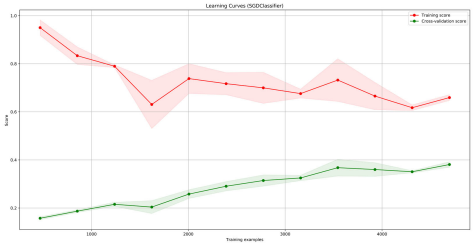
(g) AdaBoost classifier.



(h) MLP classifier.

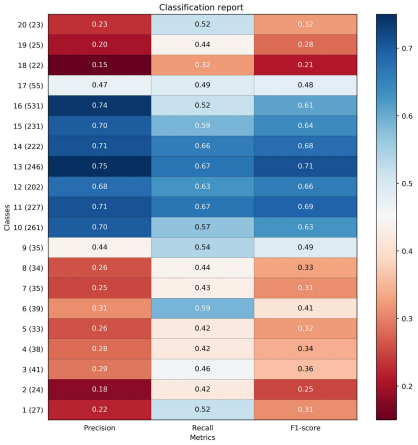


(i) Ridge classifier.

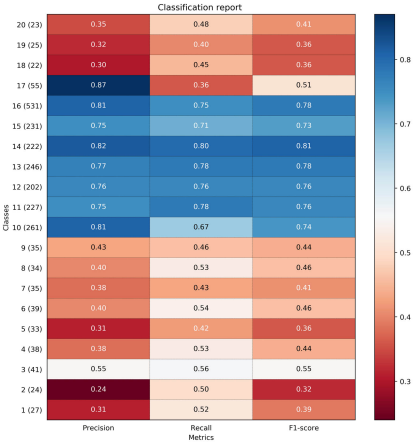


(j) Linear SVM with SGD

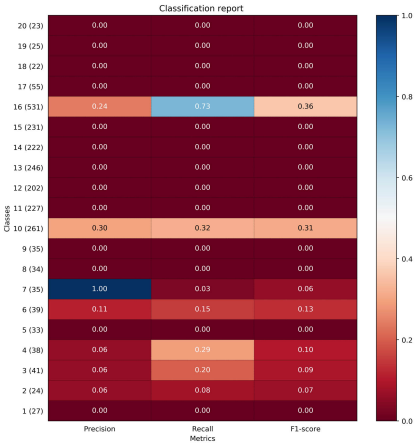
Figure A.17: Train and validation learning curves (Representation 2|Dodgson winner(s) predictions).



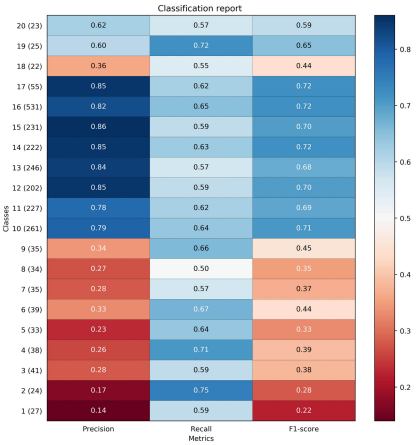
(a) Gaussian Naive Bayes classifier.



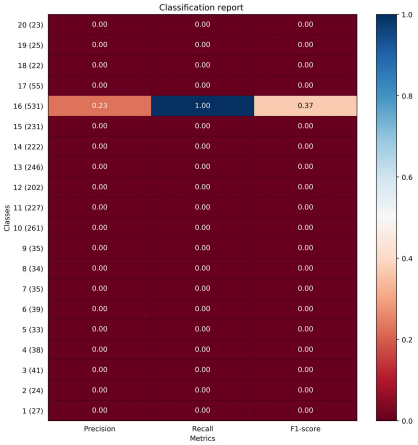
(b) XGBoost classifier.



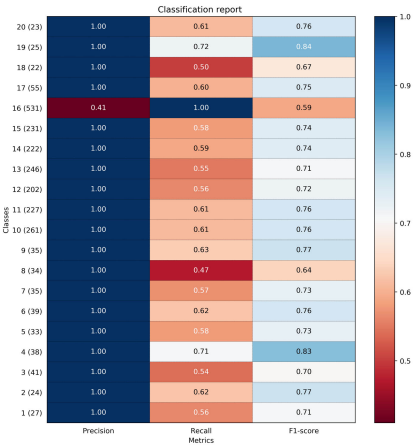
(c) Decision tree classifier.



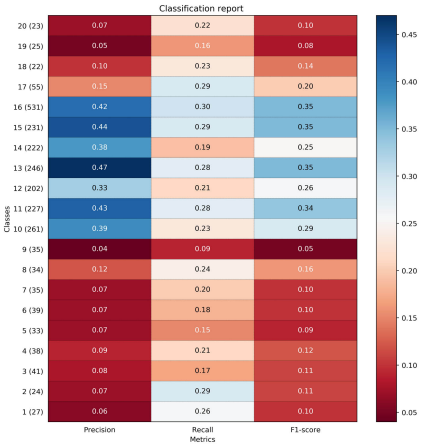
(d) Random forest classifier.



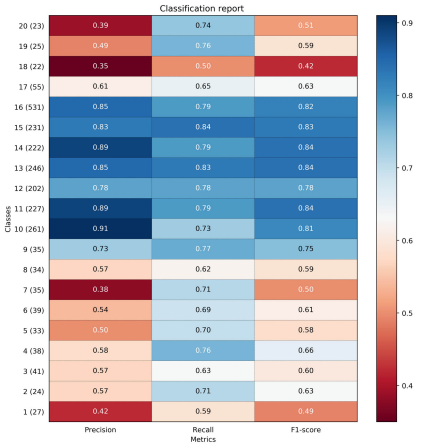
(e) Random forest classifier.



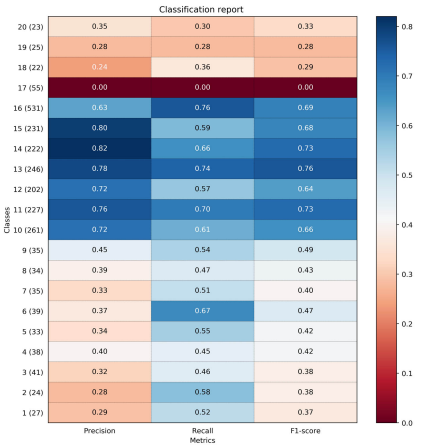
(f) Support Vector classifier with RBF kernel.



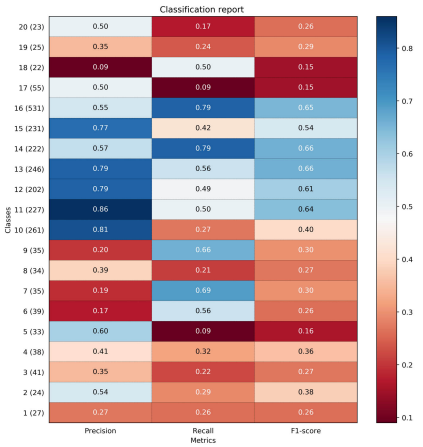
(g) AdaBoost classifier.



(h) MLP classifier.

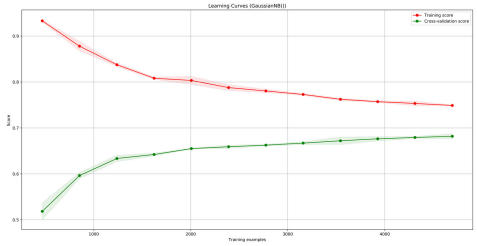


(i) Ridge classifier.

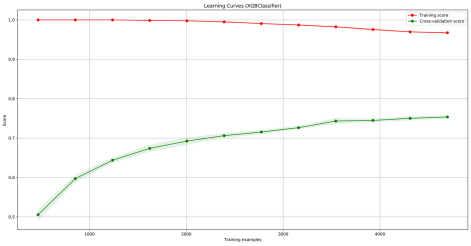


(j) Linear SVM with SGD

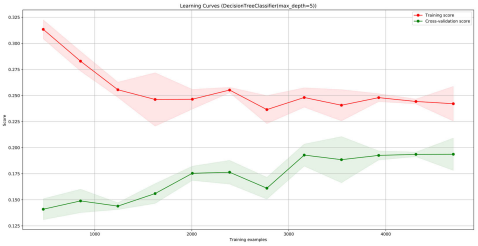
Figure A.18: Classification report (Representation 2|Dodgson winner(s) predictions).



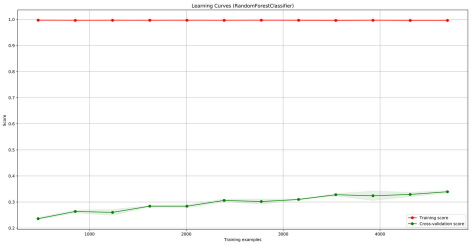
(a) Gaussian Naive Bayes classifier.



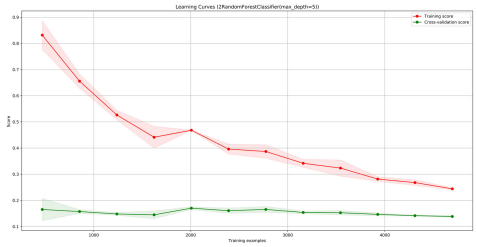
(b) XGBoost classifier.



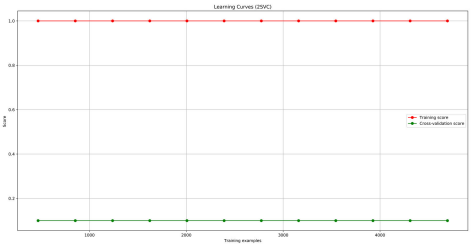
(c) Decision tree classifier.



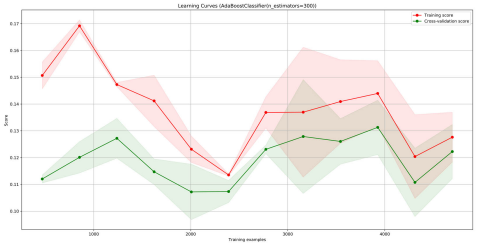
(d) Random forest classifier.



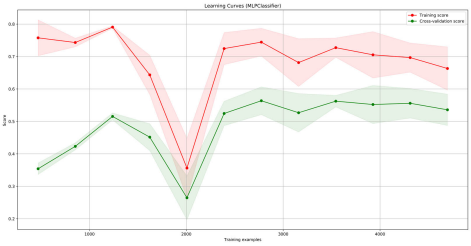
(e) Random forest classifier.



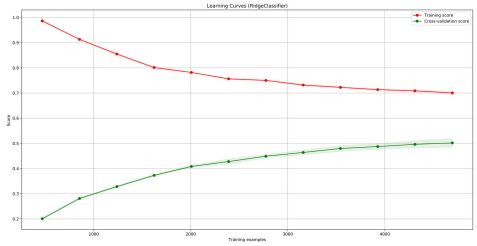
(f) Support Vector classifier with RBF kernel.



(g) AdaBoost classifier.



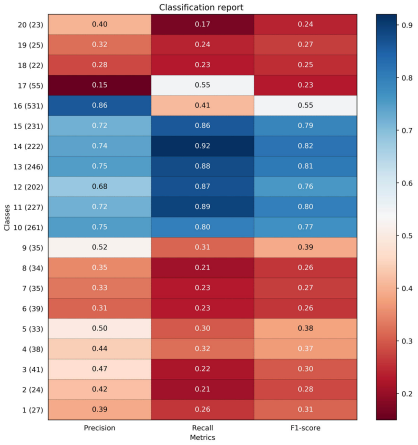
(h) MLP classifier.



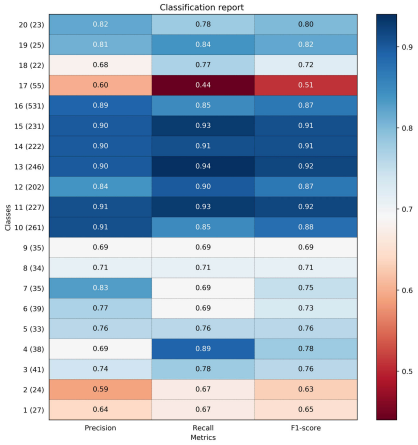
(i) Ridge classifier.

(j) Linear SVM with SGD

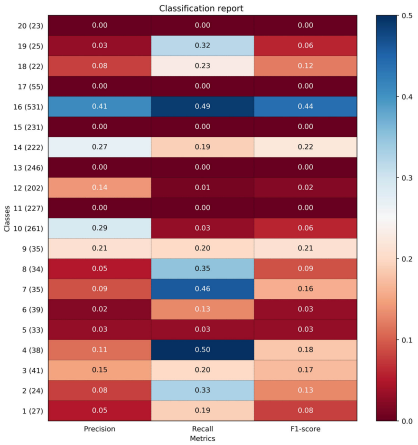
Figure A.19: Train and validation learning curves (Representation 3|Dodgson winner(s) predictions).



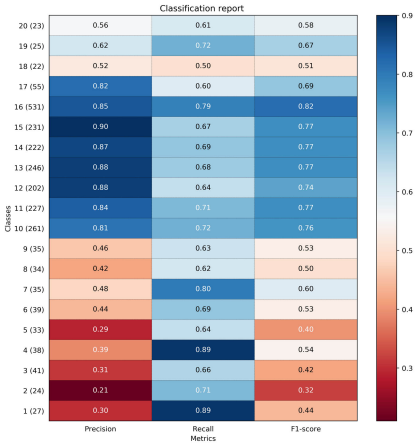
(a) Gaussian Naive Bayes classifier.



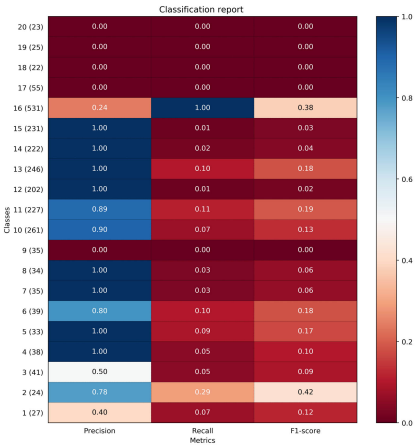
(b) XGBoost classifier.



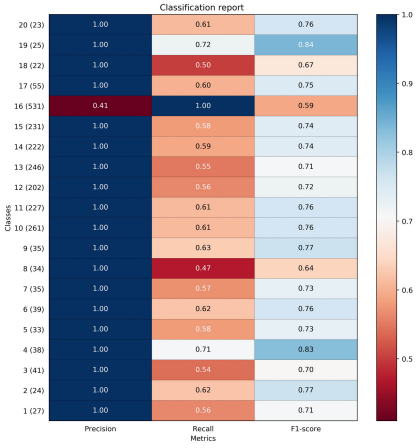
(c) Decision tree classifier.



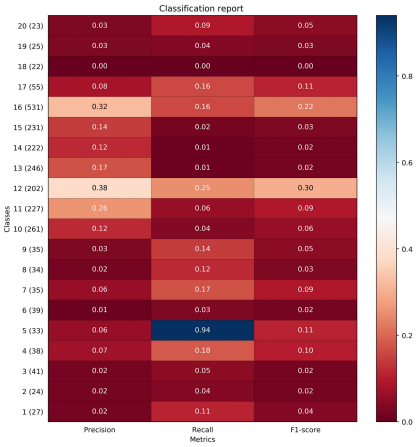
(d) Random forest classifier.



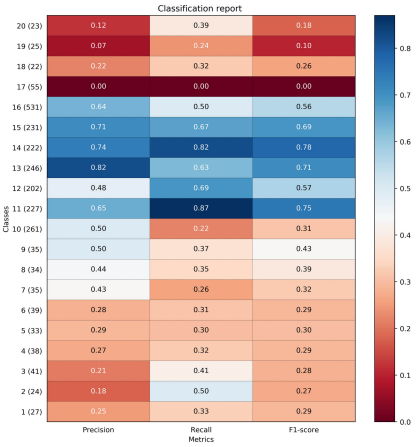
(e) Random forest classifier.



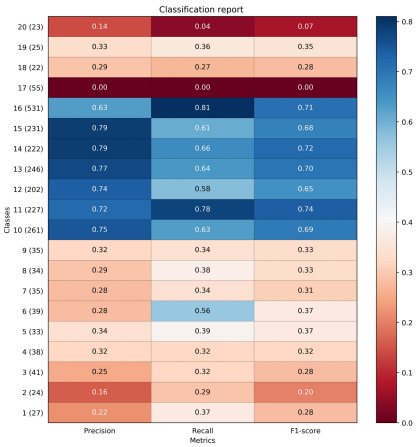
(f) Support Vector classifier with RBF kernel.



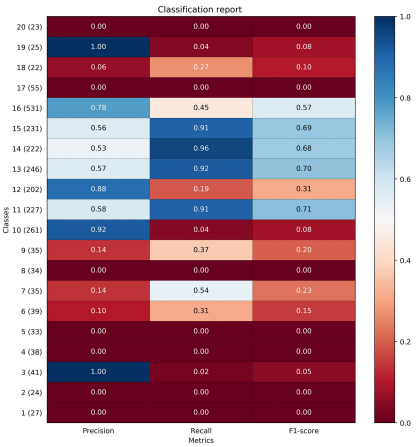
(g) AdaBoost classifier.



(h) MLP classifier.

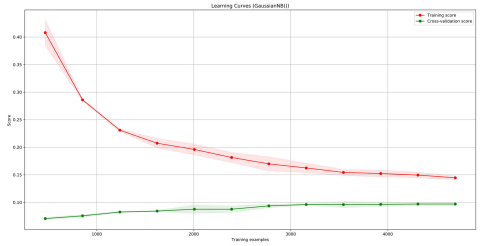


(i) Ridge classifier.

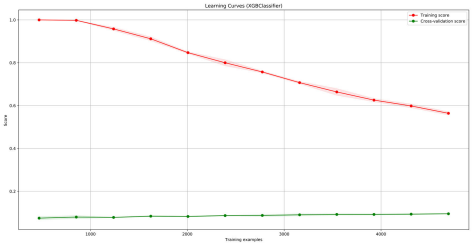


(j) Linear SVM with SGD

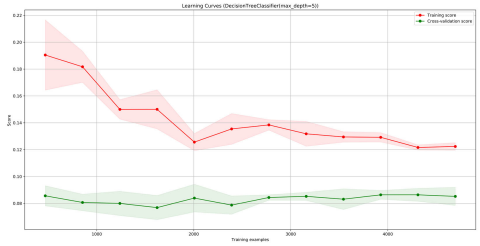
Figure A.20: Classification report (Representation 3|Dodgson winner(s) predictions).



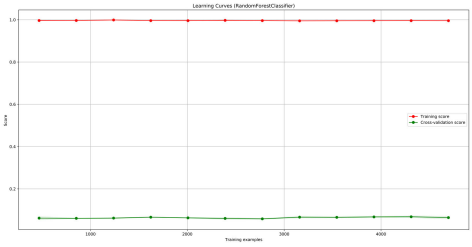
(a) Gaussian Naive Bayes classifier.



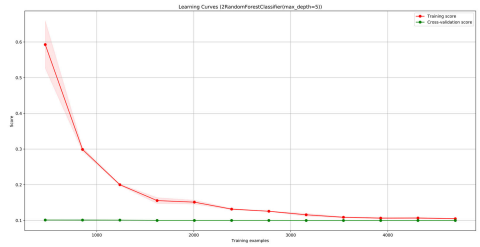
(b) XGBoost classifier.



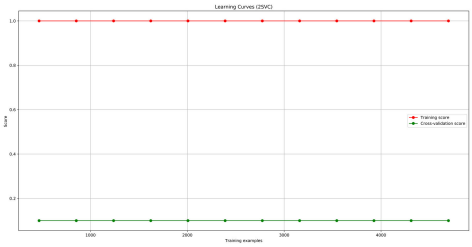
(c) Decision tree classifier.



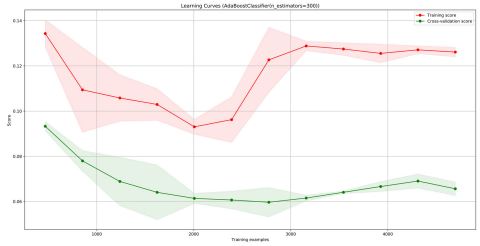
(d) Random forest classifier.



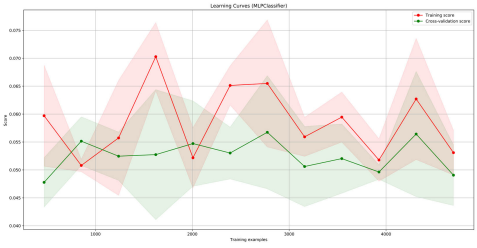
(e) Random forest classifier.



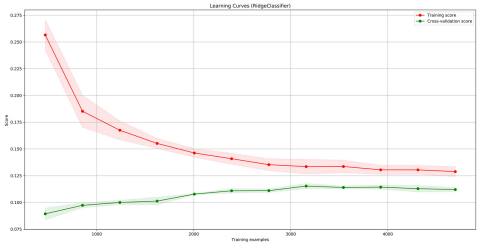
(f) Support Vector classifier with RBF kernel.



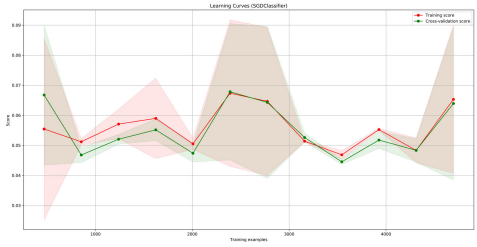
(g) AdaBoost classifier.



(h) MLP classifier.

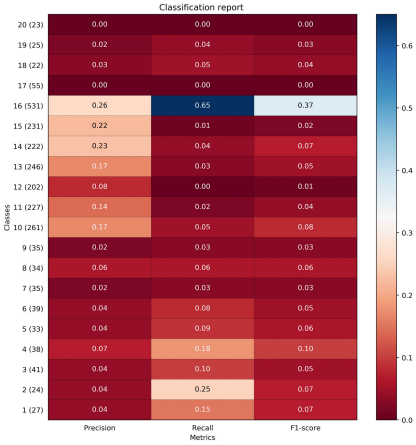


(i) Ridge classifier.

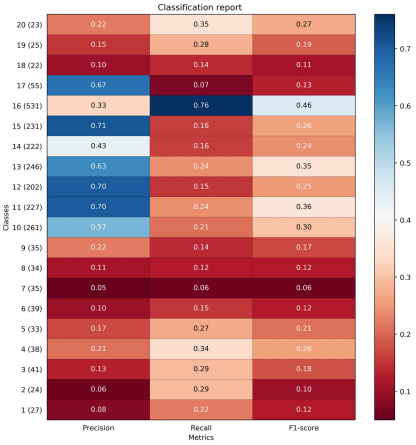


(j) Linear SVM with SGD

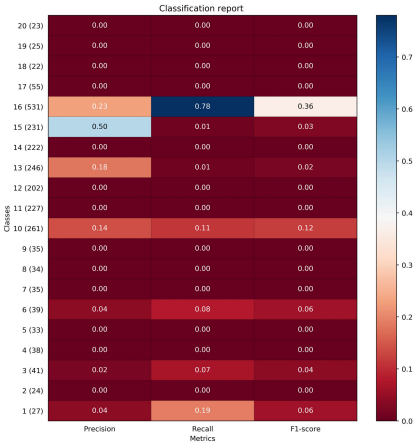
Figure A.21: Train and validation learning curves (Representation 4|Dodgson winner(s) predictions).



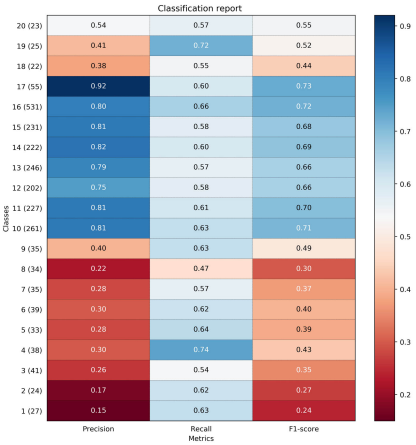
(a) Gaussian Naive Bayes classifier.



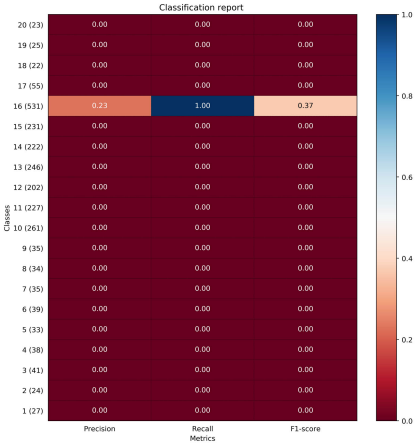
(b) XGBoost classifier.



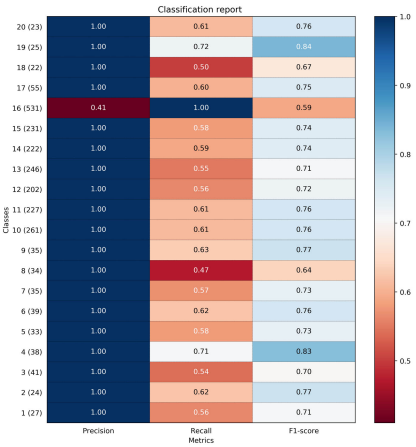
(c) Decision tree classifier.



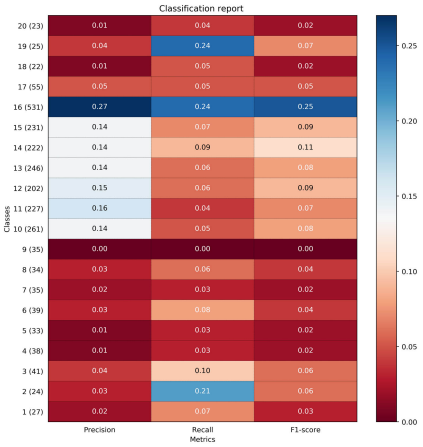
(d) Random forest classifier.



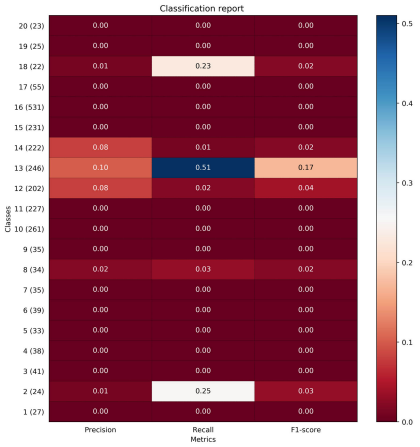
(e) Random forest classifier.



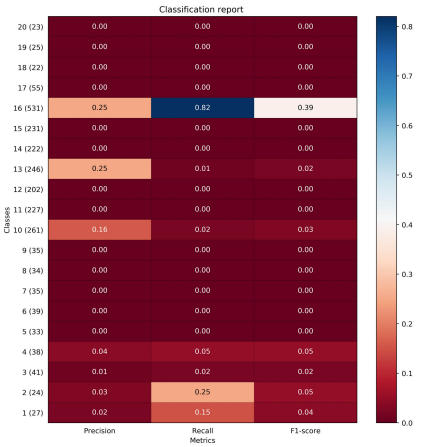
(f) Support Vector classifier with RBF kernel.



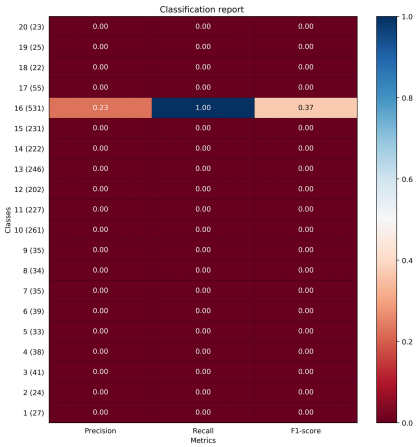
(g) AdaBoost classifier.



(h) MLP classifier.



(i) Ridge classifier.



(j) Linear SVM with SGD

Figure A.22: Classification report (Representation 4|Dodgson winner(s) predictions).

Bibliography

Ailon, N. (2010). Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica 57*(2), 284–300. 5, 6

Ali, A. and M. Meila (2012). Experiments with kemeny ranking: What works when? *Mathematical Social Sciences 64*, 28–40. 6

Aziz, H., S. Gaspers, N. Mattei, N. Narodytska, and T. Walsh (2013). Ties matter: Complexity of manipulation when tie-breaking with a random vote. *In Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, 74–80. xiii, 21, 37

Ben-Bassat, I., B. Chor, and Y. Orenstein (2018). A deep neural network approach for learning intrinsic protein-RNA binding preferences. *Bioinformatics 34*(17), i638–i646. 2

Borda, J. (1781). *Mémoire sur les élections au scrutin, Mémoire de l'Académie Royale. Histoire de l'Académie des Sciences*. 5

Brandt, F., V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia (2016). *Handbook of Computational Social Choice* (1st ed.). New York, NY, USA: Cambridge University Press. 1, 3, 9, 10, 21

Charwat, G. and A. Pfandler (2015). Democratix: A declarative approach to winner determination. In T. Walsh (Ed.), *Algorithmic Decision Theory*, Cham, pp. 253–269. Springer International Publishing. 12, 14, 18

Chevaleyre, Y., U. Endriss, J. Lang, and N. Maudet (2007). A short introduction to computational social choice. In J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plášil (Eds.), *SOFSEM 2007: Theory and Practice of Computer Science*, Berlin, Heidelberg, pp. 51–69. Springer Berlin Heidelberg. 1, 2

Chu, W. and Z. Ghahramani (2005). Preference learning with gaussian processes. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, New York, NY, USA, pp. 137–144. ACM. 1

Condorcet, N. (1785). Essai sur l'application del'analyse 'a la probabilit´e des d´ecisions rendues 'a lapluralit´e des voix. l'imprimerie royale, paris. 5, 6

Conitzer, V., D. A. and J. Kalagnanam (2006). Improved bounds for computing kemeny rankings. Volume 6, pp. 620–626. 6

Conitzer, V., R. M. and L. Xia (2009). Preference functions that score rankings andmaximum likelihood estimation. Volume 9, pp. 109–115. 6

Conitzer, V. and T. Sandholm (2005). Common voting rules as maximum likelihood estimators. Volume 9, Arlington, Virginia.AUAI Press, pp. 145–152. 6

Corrente, S., S. Greco, M. Kadziński, and R. Słowiński (2013). Robust ordinal regression in preference learning and ranking. *Machine Learning 93*(2), 381–422. 1, 2

De Neve, J. (2014). Ideological change and the economics of voting behavior in the US, 1920–2008.. *Elect Stud 34:27–3*. 7

Deng, K., S. Han, K. J. Li, and J. S. Liu (2014). Bayesian aggregation of order-based rank data. *Journal of the American Statistical Association 109*(507), 1023–1039. 2, 5

Donini, M., A. Loreggia, M. S. Pini, and F. Rossi (2018). Voting with random neural networks: a democratic ensemble classifier. In *RiCeRcA@AI*IA*. 5

Dwork, C., R. Kumar, M. Naor, and D. Sivakumar (2001). Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, New York, NY, USA, pp. 613–622. ACM. 2, 5, 17, 21

F. Rossi, K. B. V. and T. Walsh (2011). *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan Claypool Publishers. 1, 9, 37

Farrugia, V. E., H. P. Martínez, and G. N. Yannakakis (2015). The Preference Learning Toolbox. *arXiv e-prints*, arXiv:1506.01709. 1

Hüllermeier, E. and J. Fürnkranz (2011). Learning from label preferences. In T. Elomaa, J. Hollmén, and H. Mannila (Eds.), *Discovery Science*, Berlin, Heidelberg, pp. 2–17. Springer Berlin Heidelberg. 1, 6

Kamishima, T., H. Kazawa, and S. Akaho (2011). *A Survey and Empirical Comparison of Object Ranking Methods*, pp. 181–201. Berlin, Heidelberg: Springer Berlin Heidelberg. 5

Kim, S. (2017). Ordinal versus cardinal voting rules: A mechanism design approach. *Games and Economic Behavior 104*(C), 350–371. 10

Korba, A., S. Clémençon, and E. Sibony (2017). A learning theory of ranking aggregation. 6, 62

Lang, J., M. S. Pini, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh (2012). Winner determination in voting trees with incomplete preferences and weighted votes. *Autonomous Agents and Multi-Agent Systems 25*(1), 130–157. 6, 17

Li, P. (2008). Learning to rank using classification and gradient boosting. In *NIPS 2008*. 3

Lucchese, C., F. M. Nardini, R. Perego, S. Orlando, and S. Trani (2018). Selective gradient boosting for effective learning to rank. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, New York, NY, USA, pp. 155–164. ACM. 3

miro.medium.com (2016). Learning to rank concept. xiii, 25

N. Mattei, N. N. and T. Walsh (2014). How hard is it to control an election by breaking ties?. *In Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 1067–1068. 21

Ng, A. (2019). Machine learnig. Mhttp://cs229.stanford.edu/syllabus.html. [Online; accessed 12-04-2019]. xiii, 24, 34

Nurmi, H. (2010). e-democracy: A group decision and negotiation perspective. *Voting Theory*, 101–123. 3

Obraztsova, S. and E. Elkind. (2011). On the complexity of voting manipulation under randomized tie-breaking. *In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 319–324. 21

Patel, T., D. Telesca, R. Rallo, S. George, T. Xia, and A. E. Nel (2013). Hierarchical rank aggregation with applications to nanotoxicology. In *Journal of Agricultural, Biological, and Environmental Statistics*, pp. 18(2):159–177. 5

Procaccia A.D., Zohar A., P. Y. R. J. (2009). The learnability of voting rules. *Artificial Intelligence 173*(12–13), 1133–1149. 1, 5, 6, 7, 13, 57, 61

Raizada, R. and Y. S. Lee (2013). Diagram of the gaussian naive bayes model. Mhttps://www.researchgate.net/figure/Illustration-of-how-a-Gaussian-Naive-Bayes-GNB-classifier-works-For-each-data-point_fig1_255695722. [Online; accessed 12-06-2019]. xiii, 30

Renda, M. E. and U. Straccia (2003). Web metasearch: Rank vs. score based rank aggregation methods. In *Proc. of the 18th Annual ACM Symposium on Applied Computing*, Melbourne, Florida, pp. 841–846. ACM Press. 2, 5

scikit learn.org (2013a). Plot of the svm hyperplane. Mhttp://scikit-learn.org/stable/modules/svm.html. [Online; accessed 12-10-2019]. xiii, 27

scikit learn.org (2013b). Plot of the svm hyperplane. Mhttp://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Online; accessed 12-03-2019]. xiii, 29

Teknomo, K. (2018). Revoledu. [Online; accessed 22-09-2018]. 15

thefactmachine.com (2016). Logistic regression. Mhttp://www.thefactmachine.com/logistic-regression/. [Online; accessed 12-10-2019]. xiii, 27

Truchon, M. (2008). Borda and the maximum likelihood approach to vote aggregation. 2, 6

Verikas, A., E. Vaiciukynas, A. Gelzinis, and M. C. Olsson (2016). Architecture of the random forest model. Mhttps://www.researchgate.net/figure/Architecture-of-the-random-forest-model_fig1_301638643. [Online; accessed 12-01-2019]. xiii, 28

Volkovs, M. (2013). *Machine Learning Methods and Models for Ranking*. Ph. D. thesis, University of Toronto. 1

Wikipedia (2019a). Borda count. Mhttps://en.wikipedia.org/wiki/Borda_count. [Online; accessed 12-02-2019]. 14

Wikipedia (2019b). Definition of nondeterministic polynomial time. M[ttps://en.wikipedia.org/wiki/NP_(complexity). Accessed: 2019-10-11. 2

Wikipedia (2019c). Diagram of the cross-validation model training technique. M[https://en.wikipedia.org/wiki/Cross-validation_(statistics)1. Accessed: 2019-08-10. xiii, 32

Xia, L. (2019). Learning and decision-making from rank data. *Synthesis Lectures on Artificial Intelligence and Machine Learning 13*, 1–159. 6, 9, 23

Yang Hu, Mingjing Li, and Nenghai Yu (2008). Multiple-instance ranking: Learning to rank images for image retrieval. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8. 2

Young, H. P. (1988). Condorcet's theory of voting. american political science review. *82(4)*, 1231–1244. 6, 16