University of Bergen
Department of Informatics

Stem cell identification via RNA sequencing in breast cancer

Author: Cristóbal Giménez Devís
Supervisors: Pekka Parviainen

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June 2024

Abstract

Breast cancer is a complex, heterogeneous disease with distinct cancer subtypes. When the breast organ develops, it has a large pool of mammary stem cells. These cells have the potential to give rise to the various cell types that constitute the breast, as such, they are at risk for acquiring mutations, which can lead to transformed stem cells, thereby promoting the process of tumorigenesis. Among the breast cancer cells, increasing evidence points to the presence of a rare, small and heterogeneous subpopulation of cancerous cells termed as breast cancer stem cells (BCSCs), which have unlimited renewal capacity and are responsible for repopulating and giving rise to the heterogeneous, overly aggressive tumors. The recent technology of gene expression profiling like RNA-seq has helped some studies try to figure a way of identifying BCSCs, but there is still much work to be done. In this work we will use Machine Learning approach together with RNA-seq techonology in order to try to solve this task.

Acknowledgements

Thank you, Anagha, for your constant tutelage and dedication, for going out of your way to take on a risky project which could have easily failed.

Thank you Pekka, for providing us with knowledge, and experience in the field, they say some great ideas spark in the most mundane conversations, and so they did.

Thank you, mother. Thank you, father. Thank you, sister, as time passes, I realize more and more how nothing I ever made would have been possible without you.

Thank you, to those who saw me start but could not watch me end, wherever you are, keep watching over us.

" *You can never know everything, and part of what you know is always wrong. Perhaps even the most important part. A portion of wisdom lies in knowing that. A portion of courage lies in going on anyways.*" *Lan to Rand - The Eye of the World.*

Cristóbal Giménez Devís

Contents

List of Figures

Chapter 1

Introduction

Breast cancer is one of the most frequently diagnosed malignancies in the world, which has high rate of metastasis (propagation of cancer) and recurrence (cancer reappearing after treatment), leading to very low survival rate.

Among the breast cancer cells there is a very small and rare subset of cells called *Breast cancer stem cells* (BCSCs), which are very heterogeneous and have stem cell properties i.e. the capacity to self-renew and differentiate (transform into other cells).

This rare sub-population of tumor cells is characterized with strong tumorigenic capacity which can promote metastasis and recurrence, and therefore have been associated with both poor prognosis, and therapy resistance.

RNA-seq is a technology used to extract the gene expression sequences in a biological sample. In our case, RNA-seq is used to extract the sequences of genes from each cell individually.

Each sequence (represented as a vector) will contain the expression of the genes (the activity of the gene in the cell, represented with a numerical value), this, will be the data we use in this work.

From this point forward, we present a Machine Learning approach at trying to identify these rare populations of BCSCs with the gene expression sequences.

1.1 Motivation

Obtaining the cure for cancer has always been the end goal of modern medicine, and nowadays we are still far away from obtaining good and general results. So, any light that can be shined onto this issue is always welcomed.

The increasing power of Machine Learning systems and models have been useful to solve real world complex tasks in numerous areas, so, Machine Learning could also be useful in trying to solve the mystery of cancer.

1.2 Objectives

The main objective of this work is to correctly classify a given sequence of genes which represents the information of a cell, into two different classes:

- **Class 0**: This class will represent non-stem like cells.
- **Class 1**: This class will represent stem-like or progenitor cells (our main focus).

More precisely, we are trying to focus on breast cancer, so, our main objective is to correctly identify BCSCs from non-BCSCs.

Additionally, we add the following sub-objectives:

- Explore different machine learning models and methods to try to solve the task at hand.
- Interpret what the models learn in order to possible obtain any meaningful biological information.
- Measure the generalization of the models to different datasets, with different types of cells and stem cells.
- Check with real data, the claims of recent papers which have identified BCSC markers. And discuss the result.

1.3 Structure of the thesis

This thesis is divided in a total of 5 chapters, which are distributed in the following way:

- **Chapter 2**: In this chapter we will briefly discuss the state of the art in this task, along with describing the datasets used for this work.
- **Chapter 3**: Here, we will talk about the methodology used, a brief theoretical background in machine learning, and the models used will be explained in good detail.
- **Chapter 4**: Will present the experimental setup, along with the libraries used for the experiments and the hyperparameters of the models.
- **Chapter 5**: The results will be shown in this chapter, together with a discussion and a summary.
- **Appendix A**: Here we will show plots with additional results together with some discussion that is not in the scope of the main objective.

Chapter 2

State of the art and data description

Even though the main problem of this work still hasn't been resolved, there are some attempts at trying to solve it. In this chapter we will go over the two main algorithms which try to identify cancer stem cells.

2.1 CytoTRACE

One of the state of the art methods to identify BCSCs is the CytoTRACE [9] algorithm. CytoTRACE is based on the experimental finding that stemness (the likelihood of being a stem cell) is positively correlated with the amount of genes expressed in a cell (we say if say a gene is expressed if the value is higher than zero).

The algorithm computes the amount of genes expressed for each cell, then performs some extra steps, like making a nearest neighbor graph and applying non-negative least squares regression. The final result is a score for each cell between 0 and 1 (0, meaning unlikely to be a stem cell, and 1, very likely to be a stem cell).

2.2 ORIGINS

Another state of the art method is ORIGINS [26]. This algorithm uses protein-protein interaction (PPI) data to compute intra-cell gene interactions, in order to compute a score called activity level between 0 and 1 (0, meaning a less active cell, unlikely to be a stem cell, and 1, a more active cell, likely to be a stem cell).

2.2.1 Reimplementation of ORIGINS in Python

ORIGINS, unlike CytoTRACE, is only implemented in R programming language, since in this work we will be working in Python, we have to implement the R code of ORIGINS in Python and optimize it.

The PPI data used consists of a set of pairs $P = \{(g_i, g_j) \,|\, 1 \leq i, j \leq |P|\}$ of two genes, where each pair represents an interaction (gene g_i interacts with gene g_j and viceversa). ORIGINS then uses these pairs to build an adjacency matrix A of size $N_g \times N_g$, where N_g is the number of genes recorded in the dataset on which ORIGINS is applied, and sets the value 1 to the entries of the matrix where two genes interact.

Then the algorithm computes the activity of each cell by getting the interaction of all the genes of the same cell (intra-cell interaction) and sums them all:

Algorithm 1 ORIGINS pseudocode

Require: A ▷ Adjacency matrix.
Require: E ▷ Expression matrix (the data, cells by rows and genes by columns).
  $act \leftarrow []$ ▷ Empty list
  for $i \leftarrow 0$ to N_c **do** ▷ N_c is the number of cells
    $activity \leftarrow 0$
    for $j \leftarrow 0$ to N_g **do**
      $interaction \leftarrow 0$
      for $k \leftarrow 0$ to N_g **do**
        $interaction \leftarrow interaction + A[j,k] \cdot E[i,k]$
      end for
      $activity \leftarrow activity + E[i,j] \cdot interaction$
    end for
    $act[i] \leftarrow activity$
  end for
  $act \leftarrow act - min(act)/(max(act) - min(act))$ ▷ This is a broadcast operation

2.2.2 Algorithm rewriting

We can easily notice that this is a high time complexity algorithm, since it uses three for loops to access the data and operate. Not only that, the original algorithm loads the entire adjacency and expression matrix (data) in memory, which are both very sparse matrices (almost all elements are zero) and can take up tens of GB.

Also, the adjacency matrix has a shape of $N_g \times N_g$ however most of the genes that are recorded in the data do not exist in the PPI (the set of pairs of genes), this causes that there are **a lot of rows and columns in the adjacency matrix that are completely zero**, and thus, useless to the algorithm. Effectively reducing the shape of the adjancency matrix.

In our rewriting of the algorithm, the first step is to preprocess the adjacency matrix by removing all of those zero rows and columns and only leaving only the useful genes.

The second major rewriting is transforming the two internal loops into a dot matrix product to speed up the computing time, the rewriting results in:

Algorithm 2 ORIGINS rewriting

Require: A ▷ Adjacency matrix.
Require: E ▷ Expression matrix (the data, cells by rows and genes by columns).
  $act \leftarrow []$ ▷ Empty list
  for $i \leftarrow 0$ to N_c **do** ▷ N_c is the number of cells
    $interaction \leftarrow E[i,:] \cdot A$ ▷ $E[i,:]$ means the whole ith row of matrix E
    $activity \leftarrow interaction \cdot E[i,:]^T$ ▷ $E[i,:]^T$ means the ith row of E transposed
    $act[i] \leftarrow activity$
  end for
  $act \leftarrow act - min(act)/(max(act) - min(act)$ ▷ This is a broadcast operation

The third and final step is to convert both the expression and adjacency matrix into sparse matrices to reduce the spatial complexity of the algorithm, for this, we used the ***scipy*** library to convert into sparse matrices to save memory, and use fast and efficient dot products.

2.3 Data description

In this work, we will use different datasets from which our models will learn. We need different datasets because one of our objectives is to see if the models can capture any patterns in the data that are specific to cancer stem cells and then measure how good the pattern is by checking the generalization of the models across different datasets.

2.3.1 Breast cancer dataset (BCD)

The first dataset is data from breast cancer. We will refer to it as *breast cancer data* (*BCD*), and we need it since it's the core objective of this work. The data is obtained from [14] with contains approximately $240,000$ cells, of which, only $3,536$ are labeled as BCSC. Also, each cell has approximately $33,000$ genes (features) recorded.

One important remark, is that, with the current biological knowledge of breast cancer, there doesn't exist a ground truth, i.e. no dataset has all the BCSCs labeled, and some stem cells might be incorrectly labeled. So, we have to carry that error with us, since there is no way around it.

2.3.2 Hematopoietic dataset (HSD)

The second dataset we will use comes from cells in the human blood system (concretely, the hematopoietic system, which generates blood cells in the human body).

The reason why we need this data is because the differentiation process (how stem cells transform into other cell types) of hematopoietic stem cells (HSCs for short) is well studied and known, however the same cannot be said for BCSCs. This means, that unlike BCD, the labelling of this dataset is much more reliable.

So, we will use this data to see if models that learn on hematopoietic data where the differentiation process is clear, can learn any patterns that generalize well to identify BCSCs.

The HSD is obtained from [23] which contains approximately $266,000$ cells, of which $4,323$ are labeled as HSCs, also, each cell has approximatelly $27,000$ genes (features) recorded.

2.3.3 General dataset (General)

The third and last dataset comes from a collection of different cells and stem cells in the entire human body, hence, we will refer to this dataset as *General*.

This dataset contains a very small portion of HSCs, but not BCSCs, and the labelling is also reliable.

The data is obtained from [33] where the dataset contains approximately $1,273,000$ cells, from which $83,871$ are labeled as stem cells, this last dataset has genes $51,383$(features) recorded. we use this data to provide the models with many different types of stem cells to see if they can capture any pattern specific to stem cells and then generalize to other datasets.

Chapter 3

Methodology

Machine Learning is a branch of Artificial Intelligence whose purpose is to create statistical models that are able to learn patterns and relationships within the data they are provided with, in order to correctly make predictions. In this work, we will discuss and show how we used Machine Learning to solve the task at hand.

3.1 Theoretical Background

Before we continue, there are some aspects about Machine Learning that need to be elaborated to be able to understand our methodology better.

3.1.1 Supervised Learning

As previously described, we currently have both data obtained from different sources, and some of its medical annotation, which we will now refer to it as *labels*. This particular situation, in which we hold both the data and its labels to learn from, is a type of Machine Learning which is called Supervised Learning.

In Supervised Learning, we try to build a model that maps an input $x \in R^d$ to its correct label $y \in R$, by learning from the dataset X and its labels Y.

Mathematically speaking, we assume that there exists a function $f^*(x) : R^d \rightarrow R$ which perfectly maps inputs to the correct labels. However, such perfect function is

out of our reach, uncomputable, but we can make a Machine Learning model which approximates f^*, by learning from our dataset such that our model generalizes well to unseen datapoints (inputs not present in our dataset).

So, to summarize and formalize these concepts:

- We have a dataset X made of a number i of samples $X = \{x_i \mid 1 \leq i \leq |X|, x \in R^d\}$. And also a set of labels $Y = \{y_i \mid 1 \leq i \leq |Y|, y \in R\}$.
- We try to build a model $f(x) : R^d \rightarrow R$ which approximates f^* by learning from the datapoints in our dataset so that it generalizes well to any possible input.

In our work, we are trying to correctly differentiate BCSC from non-BCSC, the output of our model is a discrete label called a *class*, and our goal is to correctly classify a cell into **two** possible categories: Class 0 (non-BCSC) or Class 1 (BCSC). This, is translated as a **binary classification** problem, where all the labels $y \in \{0, 1\}$.

To proceed, we gather input-output pairs (x_i, y_i) and feed them to our model. With this, we obtain a formulation for the model we must build in our work:

$$f(x_i) : X \rightarrow \{0, 1\}$$

$$y_i = f(x_i)$$

3.1.2 Bias-Variance tradeoff

However, the mathematical formulation of our model is still not enough to begin approaching the problem. Theoretically, any model that follows the previous formulation would be acceptable, but in practice, reality is different.

Recall, that we are trying to fit the data in our present dataset, which means that every possible model we use will be different and will perform differently. But what makes some models better for certain datasets or tasks?

One of the first essential components of a model is **Bias**, which is defined as the difference between the **expected prediction of the model** and the **true underlying value being predicted**.

In essence, bias is the error caused by the assumptions the models makes about the data provided to it and its underlying patterns. On one hand, high bias models tend to make strong assumptions about the data (e.g. assuming the data is linear) and may fail to correctly capture the patterns because of oversimplification of the problem. On the other hand low bias models have little to no assumptions, but its possible that they end up overfitting (more on overfitting to follow).

$$Bias = \mathbb{E}\big[(f(x) - f^*(x))^2\big]$$

Another important component is **Variance**, which must not be mistaken with statistical variance σ^2. Variance refers to the **error fluctuation across different datasets**, if we have a model, and we trained the same model with different training datasets, the error measured is different for every one of those models, this is caused by **noise and random fluctuations in the data**.

$$Variance = \mathbb{E}\left[\mathbb{E}\Big[(f(x) - \mathbb{E}[f(x)])^2\Big]\right]$$

where $\mathbb{E}[f(x)]$ represents the average prediction of the model over all possible training datasets.

There is one last important concept, **irreducible error** ϵ. This, represents the error that cannot be reduced by any model, no matter how well-fitted or complex a model may be. This error comes from multiple sources, such as missing data or information, errors when measuring and taking data or even the natural variability of the task at hand (in our case, sources of irreducible error are: high variability in gene expression and missing genes in the sequences).

With all of this, we can decompose the expected error of a model in the following equation:

$$Error = Bias^2 + Variance + \epsilon$$

Now, the only thing that is left is to reduce Bias and Variance until the error is close to zero. However, it's not that simple, if we were to reduce Bias (low bias, high complexity

model), our model is likely to fit the noise in the training dataset, thus, increasing the Variance and the expected error. On the other hand, if we reduced Variance the models may fail to capture complex patterns in the data, increasing Bias and the expected error.

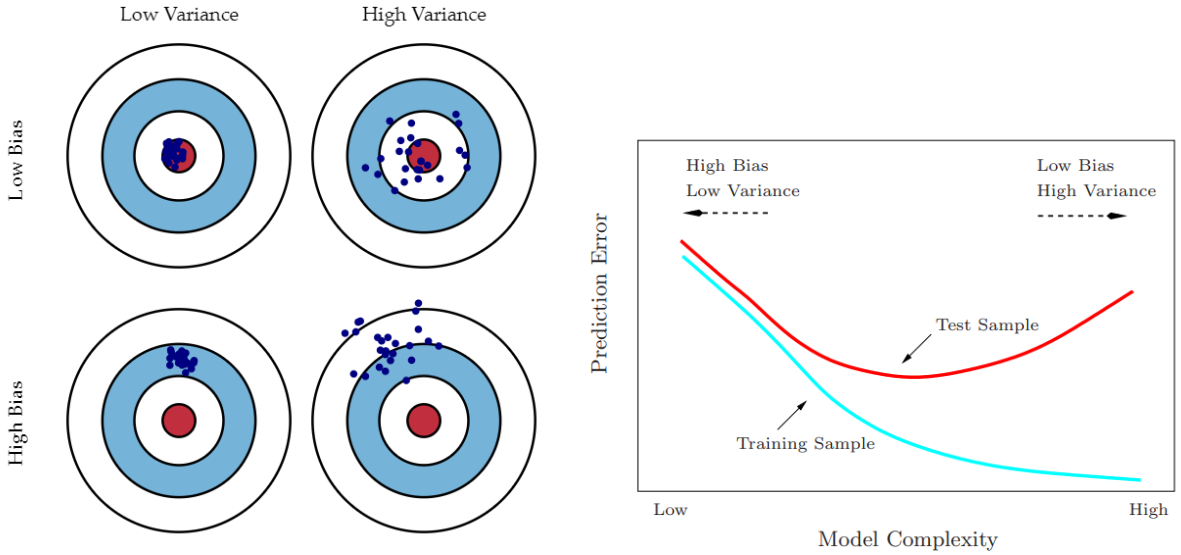


Figure 3.1: Illustration of Bias-Variance interaction.

So, the optimal approach is to find the perfect balance between Bias and Variance so that our model can fit the data and generalize well.

3.1.3 Overfitting and underfitting

Once we know about the error sources of our model, we must also understand two particular situations that occur when finding the optimal balance between Bias and Variance.

- **Underfitting**: Underfitting occurs when a model with high bias and low variance (simple model, with low complexity) fails to capture the patterns in the data and fit it properly, thus **failing to learn from the data**. Signs of underfitting include the training error being very high.
- **Overfitting**: Overfitting occurs when a model with low bias and high variance (complex model, prone to fit errors in the data) fits the training data and the noise, thus **failing to generalize**. Signs of overfitting include the training error being low and the test error being high.

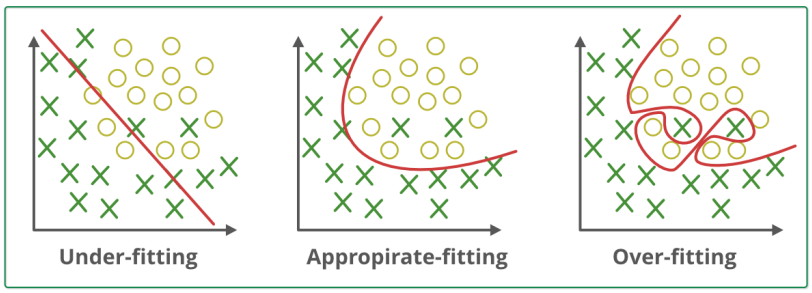


Figure 3.2: Illustration of Under and Overfitting.

3.1.4 Curse of dimensionality

In the machine learning field, there is a phenomenon that occurs when working with data that lives in high dimensional spaces, which can negatively impact the performance of the models, the **curse of dimensionality**.

When increasing the number of features (dimensions) in a dataset, the average distance between observations (datapoints) increases, making the dataset more sparse. This means, that if we want to keep the average distance constant we need to get more datapoints to fill the "empty space" in a high-dimensional sparse, dataset.

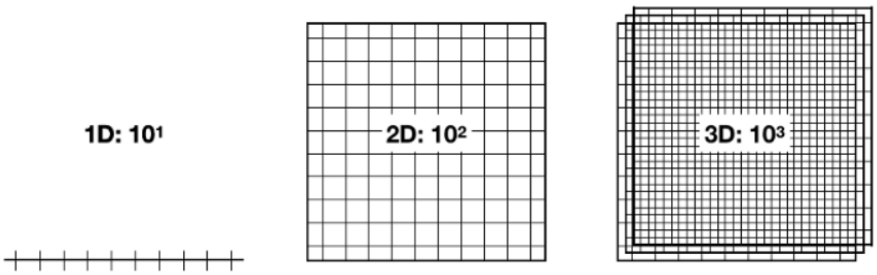


Figure 3.3: As the dimension increases, the number of datapoints needed grows exponentially.

This affects the main problem in this work, because we will be working with very high dimensional data, and we will require huge amounts of data (which are not available, or not many datasets exist). So this will also be a source of error.

3.2 Machine Learning models

We will now dive, with enough detail, into the different models that exist in Machine learning and have been used in this work.

3.2.1 Logistic Regression

Logistic regression is a supervised machine learning algorithm used for classification, where the goal is to predict the probability that an instance belongs to a given class or not.

Logistic regression consists in taking the continuous output of a **linear regressor**, and then applying a **sigmoid function** σ to it, in order to transform the output into a probability ranging from 0 to 1.

$$\sigma(x) = \frac{1}{1 - e^{-x}}$$

Just like in linear regression, we have a data matrix X of shape $n \times m$, where n is the number of samples and the m the number of features, and we want to find a vector of coefficients $w = (w_1, ..., w_m)$ and vector of biases $b = (b_1, ..., b_m)$ which minimize error. And we operate in (almost) the same way:

$$z = (\sum_{i=1}^{m} w_i \cdot x_i) + b$$

$$z = w \cdot x + b$$

but we add the extra step:

$$o = \sigma(z)$$

and thus, we transform the output into a probability of belonging to class 1.

3.2.2 Multilayer Perceptrons

A multilayer perceptron (MLP for short), consists of a set of **neurons** (also called **perceptrons**) which are interconnected and arranged by **layers**.

A **neuron** is an information-processing unit which is the basic building block of an MLP, it is composed of three elements:

- **Weights**: A set of weights which express the importance of the respective inputs to the output of the neuron. To operate, the input x_i connected to the neuron k is multiplied by the weight w.
- **Adder**: An adder for **summing the input signals**, weighted by the respective **weights** of the neuron; the operations here essentially constitute a **linear combiner**.

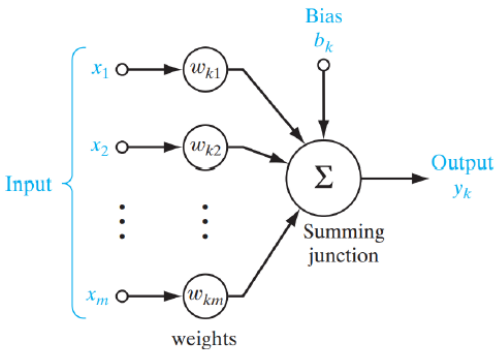


Figure 3.4: Illustration of a neuron (without activation function) .

The model of a neuron described above, also includes an externally applied **bias**, denoted by b_k. The bias b_k has the effect of increasing or lowering the total output of the adder, depending on whether it is positive or negative, respectively.

In mathematical terms, the neuron computes the output according to the following equation:

$$y_k = \left(\sum_{i=1}^{m} w_{ki} \cdot x_i \right) + b_k$$

Activation functions.

The neuron described above has the issue that it is only suitable to solve **linear problems**, and thus, cannot be applied to complex or non-linear problems. However there exists a solution for this, adding an **activation function** to the neuron.

An activation function, limits the amplitude of the output of a neuron, and is designed in such a way that it is usually a non-linear function, in order to allow the neurons to model non-linear patterns and relationships in the data. The only requirement for this activation function is that it must be **differentiable with respect to the input**.

The model of each neuron in the MLP now includes a nonlinear activation function which allows it to fit data in a non-linear way.

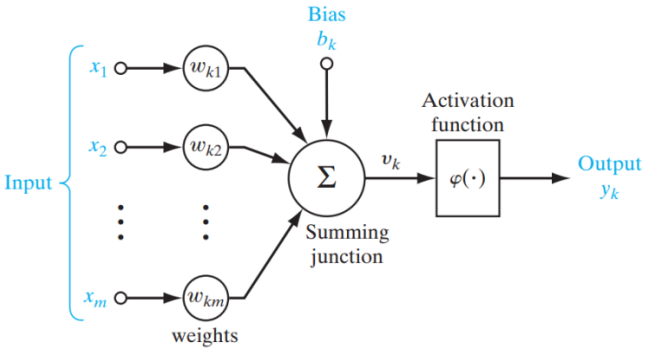


Figure 3.5: Illustration of a neuron (with activation function) .

Now the equation for the output changes to:

$$v_k = \left(\sum_{i=1}^{m} w_{ki} \cdot x_i \right) + b_k$$

$$y_k = \varphi(v_k)$$

where $(x_1, x_2, \ldots, x_m)$ are the input features; $(w_{k1}, w_{k2}, \ldots, w_{km})$ are the respective weights of neuron k; v_k is the linear combiner output due to the input signals; b_k is the bias and $\varphi(.)$ is the activation function.

Now, what's left is distributing all the neurons by **layers**. Each layer consists of a number m_i of neurons, where each neuron is connected to **all the neurons from**

the previous and the next layer. However, all the neurons from the same layer are independent (no computing dependencies) of each other. Finally, the model is organized in the following way:

- Input layer: This layer consists of neurons which receive the raw inputs.
- Hidden layers: These layers are responsible for the main computations of the MLP, the neurons extract features and patterns from the previous layers, combine them, and propagate the result forward.
- Output layer: The last layer of the model, which transforms the result from the hidden layers into a probability.

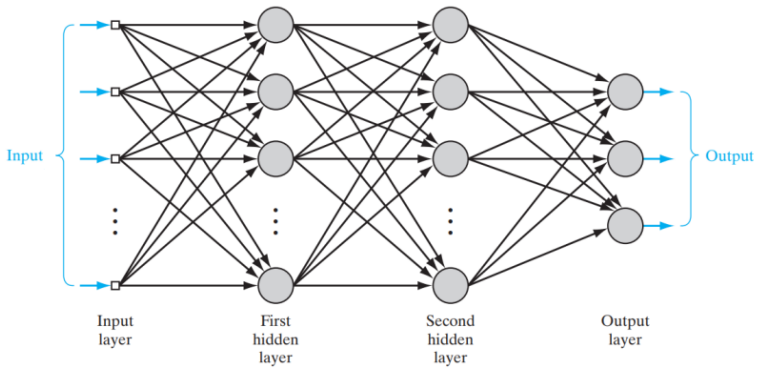


Figure 3.6: Illustration of an MLP .

3.2.3 Random Forests

Another model for classification in the machine learning field are Random Forests (RF for short), which are different from MLPs.

RFs are composed by a set of Decision Trees (DT for short), where each DT is a **graph** built as a **hierarchical spanning tree**. In each DT, the internal nodes represent the input features of the data; the edges (also called branches) represent the decision rules and each leaf node represents the outcome (in our work and study case a class).

DTs learn decision rules instead of hidden features like MLPs do. A decision rule consists of **selecting an input feature in a way that makes classifying the input data simpler**, this is done by **maximizing the information gain**.

Information gain measures how much information a feature provides us about a class. Each internal node creates a decision rule by spliting the data by the feature with the maximum information gain, which is computed using Shannon's entropy:

$$E(D) = \sum_{i=0}^{C=1} -p_i \cdot \log(p_i)$$

$$IG(D, F) = E(D) - \sum \frac{|D_s|}{|D|} \cdot E(D_s)$$

where E is Shannon's entropy, D is our dataset, F is an input feature, and D_s is the subset that results when splitting the dataset by feature F.

With all of this, we follow these steps:

- We create the **root node** of the tree.
- Each node that does not have a decision rule, splits the data according to a feature that maximizes information gain.
- However, we do not split when the data **belongs to only one class** (this means we reached a leaf node, also called a pure node).

Once there are no nodes left to split, we stop the process.

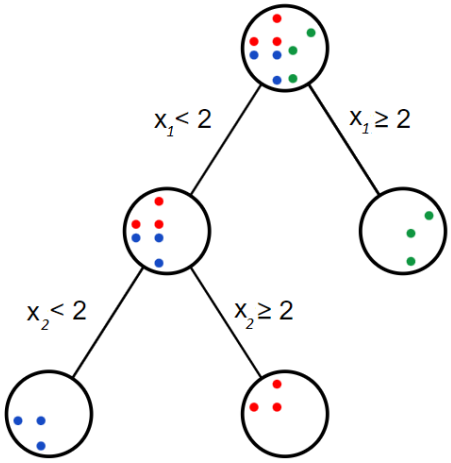


Figure 3.7: Illustration of a decision tree.

Now, to consolidate the definition of RF:

- We must gather a bunch of DTs in parallel, each DT will be independent from the rest.

- We feed to each tree a **fraction of the original dataset** so that each tree learns different decision rules. This ensures that each tree in slightly different, reducing Variance and globally improving the predictive power or performance.
- The result is the aggregation of the independent outputs of each DT.

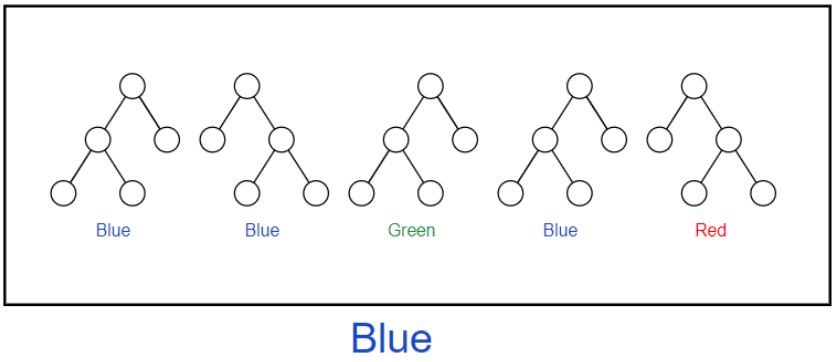


Figure 3.8: Illustration of a Random Forest (each DT learns different rules and predicts differently, but the global result when aggregating is correct, more on aggregating to follow).

3.3 Learning embeddings

When working with very high dimensional data, as we said before, distances between datapoints become very high, and the task of learning a model that maps inputs to outputs can carry a lot of error.

So, in Machine Learning, there are techniques that try to reduce this error by performing ***dimensionality reduction***. With this, we try to map the original dataset to a lower dimensional space, while trying to preserve as much relevant information as possible.

There are many techniques that can be used, in our case we decided to try to learn ***embeddings*** from the dataset. Embeddings [19] are a type of representation learning technique used in machine learning that encodes high-dimensional data to a lower-dimensional space. Each datapoint in the lower-dimensional space is then represented as a vector (called the embedding) and each vector is computed in a way that the information is preserved as well as possible.

3.3.1 "Meaningful" embeddings

The question of how to learn embeddings that preserve information inmediatly arises. In general, one cannot ensure that each of the **learned features** (dimensions in the low-dimensional space) holds any meaning or is related in any direct way to the original features.

However it is possible to add restrictions or formulate the problem in such a way that the embedding are computed in a useful way.

In our work, we have to distinguish between stem cells and non-stem cells, normally, one could think that **stem cells are similar to other stem cells and non-stem cells are similar to other non-stem cells**. But, we must not forget that in chapter 1 we mentioned that BCSCs are very heterogeneous. However, we could try **forcing the models to learn a representation in such a way that cells which have the same label have similar embeddings**.

In other words, we can make the models learn a representation (embedding) in which cells from the same class are similar to each other but very different from embeddings from other classes.

3.3.2 Metric Learning

First, since we want our embeddings to be "similar" to some other embeddings, we must define a measure of distance or similarity, in essence, a metric, which tells us how similar two embeddings are.

The standard approach would be to choose a standard distance metric (Euclidean, Manhattan, Cosine, etc.) using a priori knowledge of the domain. However, in this case, there's no concrete knowledge regarding BCSCs.

Metric learning aims at automatically constructing task-specific distance metrics from (sometimes weakly) supervised data, in a machine learning manner. The learned distance metric can then be used to perform classification. But in our case, we will use it to also learn embeddings.

When directly mapping to a lower-dimensional space randomly, the datapoints will be sparse, spread and without order. Since we want to capture similarities between datapoints, one could try, as we said previously, to preserve similarities by **making similar datapoints (called positives) closer together in the lower-dimensional space** and **different datapoints (called negatives) farther away in the low-dimensional space**, and that, is one of the core ideas of metric learning.

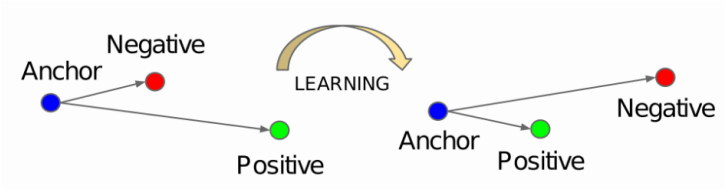


Figure 3.9: The core idea is bring together similar datapoints and push away different datapoints. The anchor is the reference datapoint (the one that is used to compute which are positives and negatives) .

ArcFace loss

The chosen metric to learn embeddings is the ArcFace [6] loss function. ArcFace tries to map every embedding (a vector) into a hypersphere (n-dimensional sphere), where **similar embeddings are closer together and different embeddings are far apart**.

The way this is done is by computing the angles between the embedding (a vector) and all the class vectors (can also be called the mean vectors of each class).

If the embedding has a higher projection (small angle) to the **class vector of the class the embedding belongs to** then we say it is **correctly projected**.

$$L_{ArcFace} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(cos(\theta_{y_i,i}+m))}}{e^{s(cos(\theta_{y_i,i}+m))}+\sum_{j\neq y_i}e^{s\,cos(\theta_{j,i})}}$$

Notice that we also add a margin m, this margin has the function of separating all the classes and penalizes deeply vectors that are not extremely close to the class vector.

Also, we add a scaling parameter s (called temperature), which encourages models to be confident in their predictions (in this case, the scaling penalizes a lot if the embedding is projected to the wrong class).

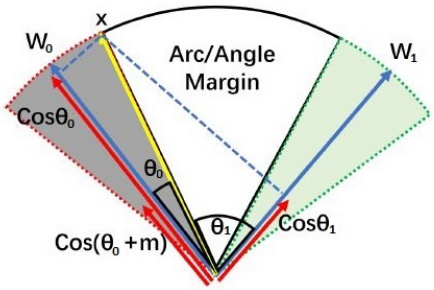


Figure 3.10: Illustration of ArcFace. X is the embedding, W_0 and W_1 are the class vectors. θ_0 and θ_1 are the angles between X and each class vector.

Batch normalization

In order to compute embeddings we used MLPs, with an extra addition, we added ***batch normalization*** which is present in almost all (if not all) metric learning architectures.

Batch normalization consists in normalizing the output of each layer of the MLP with the following equations:

$$B_n = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$
$$\mu_{mov} = \alpha\mu_{mov} + (1 - \alpha)\mu$$
$$\sigma_{mov} = \alpha\sigma_{mov} + (1 - \alpha)\sigma$$

Where γ and β are learnable parameters, and μ and σ are the mean, and standard deviation, respectively.

When evaluating a model, the mean μ and standard deviation σ are not computed, instead, the moving average and moving standard deviation is used, since they represent all the data and not just the current batch.

Batch normalization helps the training process as regularization, preventing overfitting and stabilizing learning.

3.4 Bagging

The last method we will use is called *bootstrap aggregating* (bagging for short). Bagging, is an ensembling machine learning technique which aims to reduce the Variance error of the models.

The first step is to select a base model for learning, in our case we will use as base models both MLPs and RFs.

Secondly, we perform random sampling with replacement on our dataset (randomly select a subset of samples from our original data, where each sample may appear more than once), to create numerous subsets (the same number as base models). This makes the models have access to different portions of the original data and learn different patterns.

The third step is to **independently** train each model with their corresponding subset of data.

Lastly, we **aggregate** the output of each model, in our case, the final output is the **average** of all the outputs.

Bagging helps reduce **overfitting** by reducing Variance, since the training data for each independent model is different. Not only that, it can improve the model accuracy and handles imbalanced data well (which is our case).

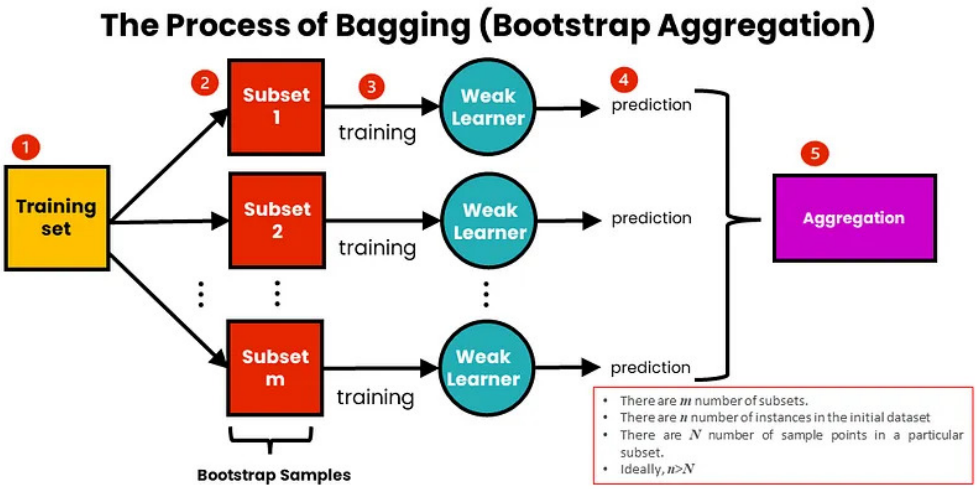


Figure 3.11: Illustration of bagging.

3.5 Performance measures

We must not forget that after training our model, we must have a method that can measure the performance of the model, how well it generalizes and how trustworthy the model's prediction are, for this, we must define a **performance measure**.

3.5.1 Precision and Recall

There are multiple performance measures, one of most common ones to use, is **accuracy**, however, in our work, we are using datasets which are severely **unbalanced**, thus, using accuracy can lead to **very optimistic and unrealistic results**. So, we will use **Precision** and **Recall**, we will define them according to the **four possible** predictions of our model:

- **True positive** (TP): When the model classifies a cell as a BCSC, and is correct in its prediction, we call this a **true positive**.
- **False positive** (FP): When the model classifies a cell as a BCSC, and is not correct in its prediction (it was not a BCSC) we call this a **false positive**.
- **True negative** (TN): When the model classifies a cell as **not** a BCSC, and is correct in its prediction we call this a **true negative**.
- **False negative** (FN): When the model classifies a cell **not** as a BCSC, and is not correct in its prediction (it was a BCSC) we call this a **false positive**.

Precision

Precision is defined as the fraction of correctly predicted positives.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Recall on the other hand is defined as the fraction of true positives the model was able to identify from the dataset.

$$Recall = \frac{TP}{TP + FN}$$

F1 - Score

F1 Score is the harmonic mean of the precision and recall, in such a way that gives an idea of both precision and recall at the same time.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

In our results we will be reporting the F1-Score for **only class 1 (BCSCs)**. And we will also report precision and recall curves, this curve shows the tradeoff between precision and recall for different thresholds. A high area under the curve represents both high recall and high precision, where high precision indicates a low false positive rate, and high recall means a low false negative rate.

3.5.2 ROC-AUC

Another performance measure is the Receiver Operating Characteristics (ROC) with Area Under the Curve (AUC).

ROC represents the recall with respect to the specificity of a classifier, it is a probability curve which represents the true positive rate versus the false positive rate according to different thresholds.

Specificity is essentially the same a precision but for class 0 (non BCSCs).

$$Specificity = \frac{TN}{TN + FP}$$

AUC measures separability, in other words, **how well our model is able to differentiate between classes**. The higher the AUC, the better the model is at predicting correctly classes 0 and 1.

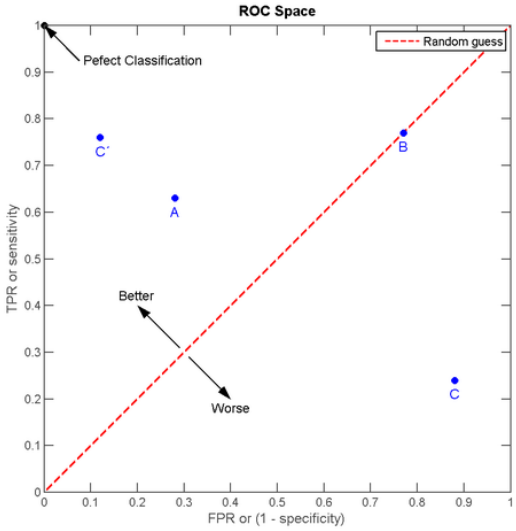


Figure 3.12: Example of a ROC-AUC (not real data).

Chapter 4

Experimental Setup

In this chapter we are going to describe how we trained our models, the framework used to operate and more.

4.1 Data management and input size issues

As previously describe before, we will be working with three different datasets, to then measure the generalization of all models across the datasets. However, there is one issue we must take care of before continuing.

Each dataset has a different number of recorded genes (features), and models like MLPs and RFs require a fixed input size which cannot change. Which means that **it is not possible to measure generalization to other datasets because each one would require a different input size**.

Fortunately there are genes (features) that are recorded in every dataset (however **some are not present in every dataset**). But, each time they are in a different position, which is also an issue, since MLPs and RFs also require that each feature must be always in the same position.

So, before the experiments, we must preprocess the data. We must come up with a way to create a **fixed input size for every dataset** and **make every gene appear in the same position everytime**.

4.2 Training formats

Our solution to this is to create what we called **training formats**. Essentially, we will assign to each gene a fixed position (and create a shared gene ordering for all datasets) and when first loading the data, all genes will be shuffled to the positions according to the ordering that is established by the training format, so that every gene stays in the same position everytime.

With this, we fix the gene order issue. We can also fix the input size problem if **we only take genes that are in the training format** (this means, ignore genes that are not in the ordering) and set **the missing genes in the format** as zero.

In our work, we have established three different formats:

- Stem cell genes format (we will refer to this format in the next chapter as StemMarkers): This format is composed of genes that are, according to biological literature, correlated with stemness (the presence of these genes can identify stem cells, and these special genes are given the name "markers"). The list of markers is extracted from [7], which contains 8547 genes. We create this format to see if **only using these genes is enough for models to correctly predict stem cells**.
- Non stem cell genes format (we will refer to this as NonStemMarkers): This format, on the other hand is composed of every gene that is not in [7]. We create this format to see if **the genes that are not in [7] contain any useful information to predict stem cells**.
- All genes format (we will refer to this as AllGenes): Here we use both the genes contained in Stem Markers format and Non Stem Markers format. We use this format to check if **giving models access to all genes improves the performance**.

4.3 Pipeline and hyperparameters

Now, let's go over the main framework and libraries used for our work:

- *PyTorch*: PyTorch is the standard machine learning library to train and evaluate models. We will use this library to create MLPs and compute embeddings.

- *PyTorch Lightning*: PyTorch Lightning is a deep learning framework that offers maximal flexibility without sacrificing performance, it optimizes the training loops, and has many options for model selection and logging results.
- *XGBoost*: XGBoost is one of the many extensions of PyTorch, this library in particular will be useful to create and train RFs.
- *PyTorch Metric Learning*: An extension of PyTorch which implements all of the necessary tools to train models using Metric Learning approaches, we will use it to compute embeddings.
- *Ensemble PyTorch*: Ensemble PyTorch is another extension which we will use to create bagging models using PyTorch as the base.
- *Scikit-learn*: A library which implements many basic functionalities, in our it will be useful for us since it implements simple Logistic Regressors and all the performance measures needed to evaluate our models.
- *SciPy*: SciPy is a library that offers algorithms for optimization,integration, interpolation, and many more problems. But we will only be interesed in using the **sparse matrices** formats to store our data matrices in order to consume far less memory.
- *Captum*: Captum is a extensible library for model interpretability built on PyTorch. It offers many methods to explain the models and help comprehend them internally. Captum will be useful to understand MLPs, where we will use *Integrated Gradients* [28], these are used to attribute the prediction of a model to its input features, which helps us uncover which features are most influential in the model.

4.3.1 Logistic Regressors

, There are no major hyperparameters to tune for logistic regressors, except the solver and the maximum iterations.

The solver used for fitting the data is "lbfgs" (which is the recommended and default option), and we have set the maximum iterations of the algorithm to 1000.

4.3.2 MLPs

When training MLPs, the overall design (architecture) affects performance, specially two choices:

- **Depth**: Depth refers to the number of hidden layers, an MLP with many hidden layers can capture complex relationships and patterns in the data, however, they are prone to overfitting and the ***vanishing gradient problem***.
- **Width**: Width refers to the number of neurons in a single layer. A layer with many neurons can learn more detailed internal representations of the data, but are also prone to overfitting.

In our work, not only we will try different combinations of width and depth, also, we will use MLPs with no activation functions and different training formats.

So, summarizing, the experiments on MLPs will try:

- Different training formats: StemMarkers, NonStemMarkers and AllGenes.
- Activation functions: If no activation function is used, the MLP becomes a linear model. In case an activation function is used it will be the ***LeakyReLU***.
- Width: The choice for width has been manually fixed. Different values from width will be drawn from a list in a logarithmic scale, considering also the training format chosen:
  - Stem Markers: In case this training format is chosen, the width is drawn from the list: [100, 250, 500, 1000, 2500, 5000, 8547], since 8547 is the maximum amount of genes correlated with BCSCs, taken from [7].
  - Non Stem Markers and All Genes: In case this training format is chosen, the width is drawn from the list: [100, 250, 500, 1000, 2500, 5000, 10000]
- Depth: The number of layers chosen will be between 1 and 4 layers, to avoid overfitting and vanishing gradient.

4.3.3 Random Forests

In the case of random forests, there are architecture choices similar to depth and width in MLPs, but slightly different:

- Depth: In this case, depth refers to number of nodes a tree can have. The more nodes a tree has, the more decision rules it must make, which can allow it to fit the training data better, but it can also overfit. The chosen experiments will use depth between 250 and 3000 nodes.

- Width: Width is not the same as in MLPs. Now, width refers to the number of trees working simultaneously, the more trees a RF has, the more it can average over more diverse decision boundaries, reducing Variance and improving performance. We will experiment using between 200 and 500 trees in parallel.
- Subsample ratio: One extra hyperparameter is the ratio of the original data that is given to each tree independently. Giving different portions of data to each tree can act as regularization and increase performance. In our case the ratios will be chosen from the list [0.5, 0.75, 1].

4.3.4 Embeddings

As we previously described before, we will use MLPs to learn embeddings. Which means, that the same hyperparameters used for MLPs will be used to learn the embeddings.

However, the architecture must be slightly modified, this time, two MLPs will be needed, one will compute embeddings, and the other one will predict a probability based on those embeddings.

The first MLP, called the ***Embedder*** consists of linear layers, with *LeakyReLU* activation function and batch normalization. The second MLP only has linear layers and *LeakyReLU* activation functions.

The experiments will try multiple values for the width, depth and training formats for the embedder.

In the case of the second MLP, the width has to be same one chosen for the embedder in order to match the shapes of the weight matrices, so, only the depth will be changed.

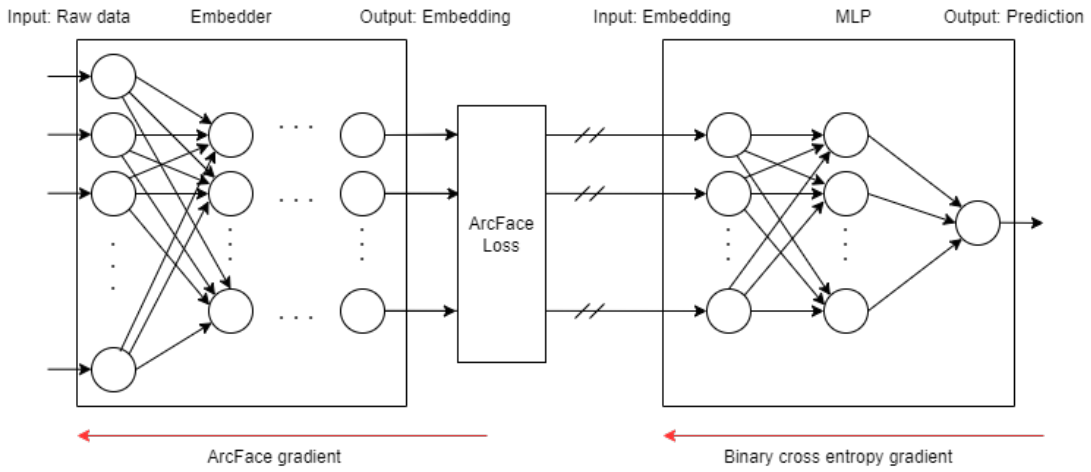


Figure 4.1: Embedder - MLP architecutre (the // symbol represents the stop-gradient operator).

Batch Miner

When working with Metric Learning loss functions and approaches it is also required to select a *mining* strategy.

Essentially, each time we compute embeddings from a batch of data, the loss can be computed differently according on how we feed the triplets (anchor, positive and negative) to the loss function. Selecting which triplets to feed to the loss function is called *mining*, and can be summarized into three cases:

- **Easy triplets**: triplets which have a loss of 0, because the distance from the **positive** to the **anchor** is less than the distance from the **negative** to the **anchor**.

$$D_f(A, P) + m < D_f(A, N)$$

  where m is the margin.
- **Semi-hard triplets**: triplets where the negative is not closer to the anchor than the positive, but which still have positive loss.

$$D_f(A, P) < D_f(A, N) < D_f(A, P) + m$$

- **Hard triplets**: triplets where the negative is closer to the anchor than the positive, with very high loss.

$$D_f(A, N) < D_f(A, P)$$

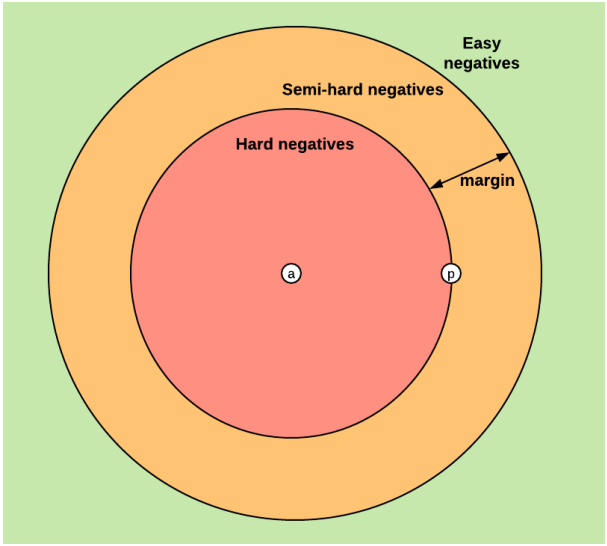


Figure 4.2: Illustration of batch mining, depending on where we place the negative example, the triplet becomes an easy, semi-hard or hard triplet.

In our work, we decided to work only with hard triplets, since according to [11] it usually yields better results. Which means that when mining the triplets, if there are any hard triplets then we have $loss > 0$, otherwise, if there are no hard triplets then $loss = 0$.

4.3.5 Bagging

One of the last methods we will try in this work is *bagging*. As previously described before, bagging consists of aggregating the output of a set of classifiers, so, naturally the architecture and number of classifier will directly affect the performance.

- Architecture: For our experiments, the chosen base model with be both a linear MLP and a non-linear MLP. However, since this may consume a gigantic amount of memory, each model will only have only hidden layer where the number of neurons will be chosen according to the training format:
  - StemMarkers: In case this training format is chosen, the width is drawn from the list: [100, 250, 500, 1000, 2500, 5000, 8547].
  - NonStemMarkers and AllGenes: In case this training format is chosen, the width is drawn from the list: [100, 250, 500, 1000, 2500, 5000, 10000].
- Number of classifiers: Each new classifier we add helps reduce the overall Variance, however, each new classifier consumes a lot of memory. In our case, we will only experiment with 5 to 10 independent classifiers.

4.3.6 Learning rate optimizer and Callbacks

Some models, like MLPs, learn from the data by minimizing a ***loss function***, in our case, the loss function chosen for all the models who need one is the ***binary cross entropy***. More specifically, in PyTorch is the ***BCEWithLogitsLoss***.

When minimizing a loss function, the weights and biases of the MLP change according to the **gradient** of the loss functions with respect to the weights and biases. However, this change must be regulated with a ***learning rate***.

The learning rate directly affects the speed at which the models learn, if the learning rate is too low, the training process may take a huge amount of time to converge (if it does converge at some point). When the learning rate is too high, the weights and biases

change drastically every iteration (epoch) and is very unlikely to reach a local minimum in the loss function. To avoid this, we use a **learning rate optimizer** which carefully computes the optimal learning rate to use every time we need to update the weights and biases.

The optimizer used in this work is **AdamW**, this optimizer is a version of **Adam** with L2 regularization, but with added weight decay to avoid overfitting and improve generalization since Adam has been observed to produce models which don't have good generalization power [16].

Early stopping callback

In order to save time and computing resources we will include an ***early stopping*** callback in every experiment.

After each epoch, we obtain the F1-Score for both the training and validation data, early stopping consists on finishing the training process if the performance on validation data doesn't improve after a set number of epochs (called patience). We have set patience to be 5 epochs, which means, if performance on validation data doesn't improve 5 epochs after the best performance obtained, training stops, and we keep the weights of the previous 5 epochs.

Chapter 5

Results

Finally, we will show the results obtained, along with a discussion.

5.1 Performance on the same dataset

After training, we must measure how well our models generalize, and compare the performance between models.

Here, we want to answer the following questions:

- How well do the models fit the data.
- How well the models generalize to the validation and test partitions.
- Interpret the hidden features the models learn in order to extract any meaningful biological information.

In this section we will show multiple tables with the F1-Score of the models, each entry of the table will be a combination of the training format (StemMarkers, AllGenes and NonStemMarkers) used and the dataset used for training (BCD, HSD, General).

Also, we will analyze the feature importance of each model and discuss the results.

However, **we will only discuss the results and importance in BCD**, in order to not artificially extend this section. **The rest of the figures and discussion** will be written in the **Appendix A**

Note: The training, validation and test partition **are always the same for all models**.

5.1.1 Logistic Regressors

Here we show the results obtained with logistic regressors:

	Training F1	Validation F1	Test F1
StemMarkers - BCD	1,000	0,900	0,920
AllGenes - BCD	**1,000**	**0,911**	**0,922**
NonStemMarkers - BCD	1,000	0,890	0,890
StemMarkers - HSD	1,000	0,919	0,930
AllGenes - HSD	**1,000**	**0,939**	**0,949**
NonStemMarkers - HSD	1,000	0,935	0,946
StemMarkers - General	0,773	0,753	0,762
AllGenes - General	**0,860**	**0,815**	**0,822**
NonStemMarkers - General	0,791	0,755	0,761

Figure 5.1: F1-Score of logistic regressors (best results in red).

The simplest model possible, a logistic regressor, already performs very well in both BCD and in HSD, however, struggles when fitting the data in the General dataset (the training F1 is not very high).

When the model is only trained with stem cell markers (genes used to isolate and identify stem cells), it also performs well, which confirms that the **stem cell markers are predictive of cancer stem cells** (at least, some, maybe not all).

On the other hand, when using only non-stem cell markers, it also has good performance:

- In BCD: The performance is worse, but still good, meaning that **non-stem cell markers are also predictive of BCSCs** (at least, some, maybe not all).
- In HSD: The performance is better than StemMarkers in both validation and test.
- In General: The performance is almost equal.

In all datasets, when the models has access to all the recorded genes (AllGenes format), the validation F1 is better than in the other two formats, which hints that combining the information provided by all the genes, cancer stem cells can be distinguished better.

Feature Importance

Now, let's analyze the feature importance of the logistic regressors, essentially, we take a look inside the model and get **which internal features has the model learned to identify the stem cells**. In the case of logistic regressor we only need to take the **weights** (or coefficients) the model learns and see which contribute positively or negatively, if a coefficient is positive it means that the probability of being a BCSC increases if the value of the gene also increases, if the coefficient is negative, it means that the probability decreases if the value increases.

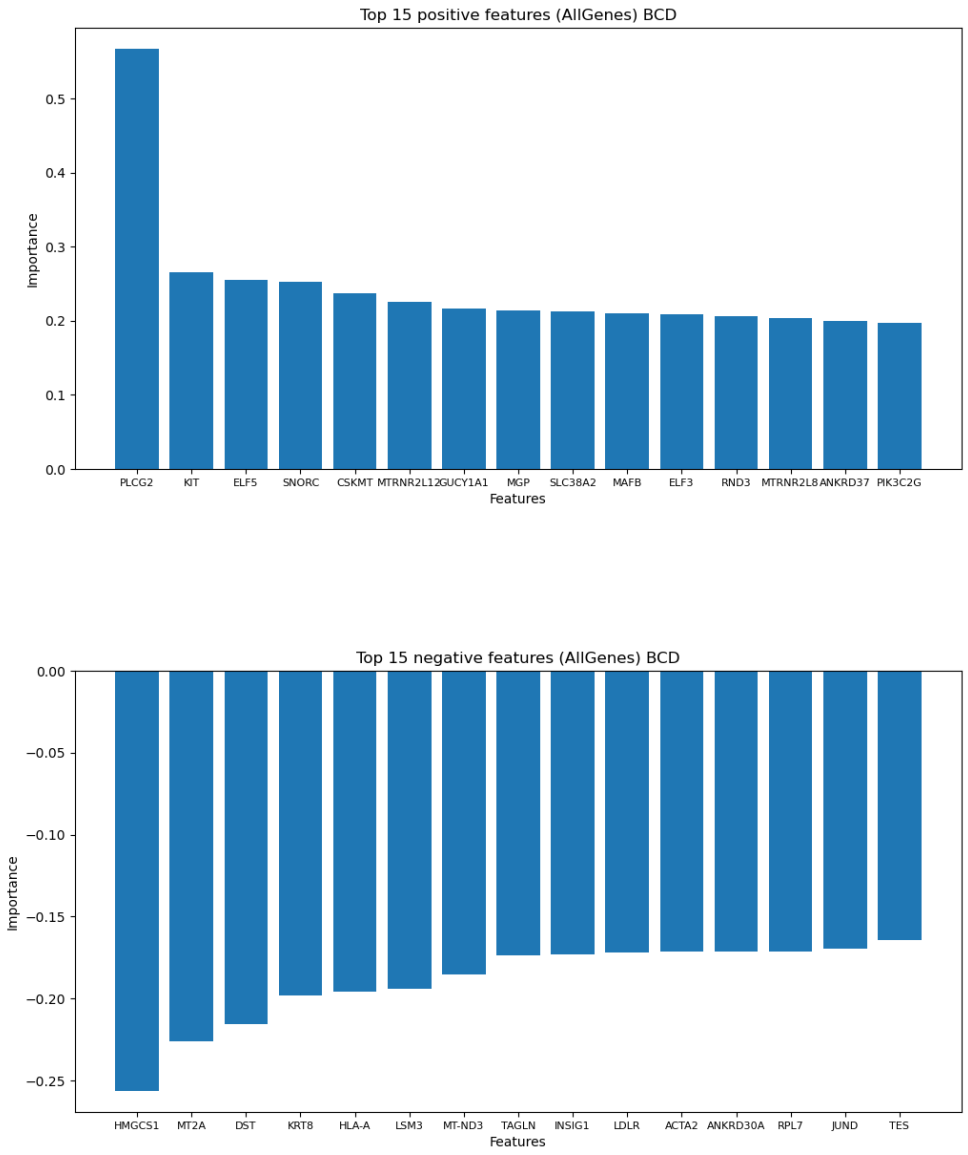


Figure 5.2: Feature importance by logistic regressors on BCD (positive importance, on top, negative importance on bottom).

We can see that, for the logistic regressor, the gene PLCG2 is by far the most important gene that predicts BCSCs, since the coefficient is higher than 0.5. This gene has already been identified as related to breast cancer and other diseases in [12].

On the flip side, the gene HMGCS1 predicts non-BCSCs. This gene is known, to be a BCSC marker, and essential for cancer stem cell activities in breast cancer in general such as they detail in the study [5]

5.1.2 MLPs

Now, let's look at the results obtained by linear MLPs:

	Training F1	Validation F1	Test F1
StemMarkers - BCD	0,990	0,909	0,918
AllGenes - BCD	**0,973**	**0,916**	**0,931**
NonStemMarkers - BCD	0,978	0,902	0,902
StemMarkers - HSD	0,999	0,935	0,933
AllGenes - HSD	**0,991**	**0,936**	**0,940**
NonStemMarkers - HSD	0,999	0,924	0,938
StemMarkers - General	0,790	0,772	0,782
AllGenes - General	**0,868**	**0,827**	**0,836**
NonStemMarkers - General	0,819	0,774	0,779

Figure 5.3: F1-Score of linear MLPs (best results in red).

And the results obtained with non-linear MLPs:

	Training F1	Validation F1	Test F1
StemMarkers - BCD	0,999	0,912	0,913
AllGenes - BCD	**0,974**	**0,911**	**0,918**
NonStemMarkers - BCD	0,977	0,900	0,913
StemMarkers - HSD	0,999	0,935	0,933
AllGenes - HSD	**0,985**	**0,938**	**0,943**
NonStemMarkers - HSD	0,996	0,932	0,939
StemMarkers - General	0,948	0,884	0,886
AllGenes - General	**0,950**	**0,896**	**0,898**
NonStemMarkers - General	0,949	0,877	0,876

Figure 5.4: F1-Score of non-linear MLPs (best results in red).

We can already tell that linear MLPs, have better test F1 than the logistic regressors in general (not all cases), which means that they generalize a little better. However, the non-linear MLPs are a little worse.

Similarly, when using the datasets BCD and HSD, they both perform well, however, the linear one struggles to fit the data in the General dataset, whereas the non-linear MLP has no issue (high training F1). Also, we can see that in all datasets, non-stem cell markers also has good performance.

Feature importance

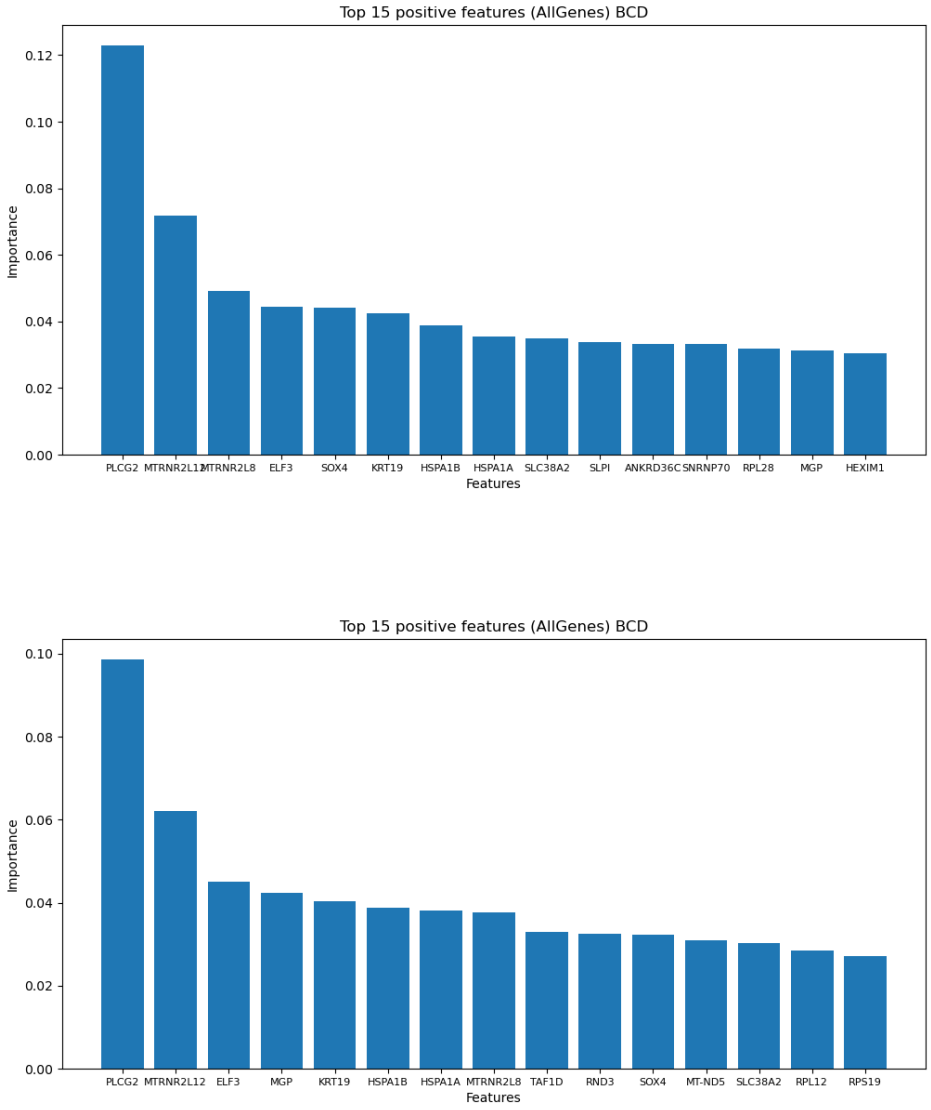
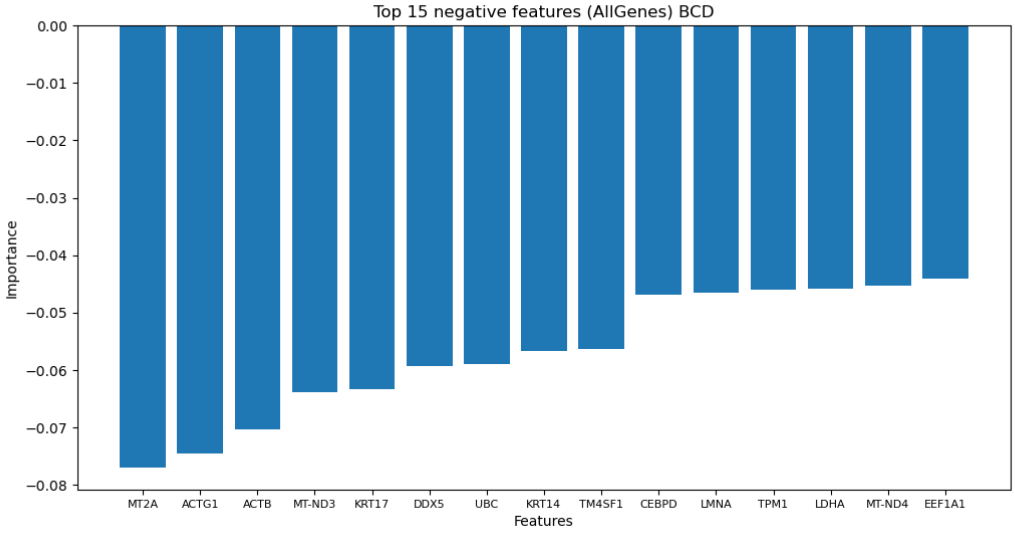


Figure 5.5: Positive feature importance by both linear (top) and non-linear (bottom) MLPs in BCD.

Just like the logistic regressors, the gene PLCG2 is the most predictive of stem cells, however we must note than in the case of MLPs the importance value is not a coefficient, now it is an **average of the integrated gradients**. Even though they can be interpreted in the same way:

- If the gradient is positive: If the value of the gene increases, then the probability of being a BCSC also increases.
- If the gradient is negative: If the value of the gene increases, then the probability of being a BCSC decreases.

Also, the gene SOX4 appears in the top 15 most importante genes, which has been identified as a very important marker of BCSCs in [21].



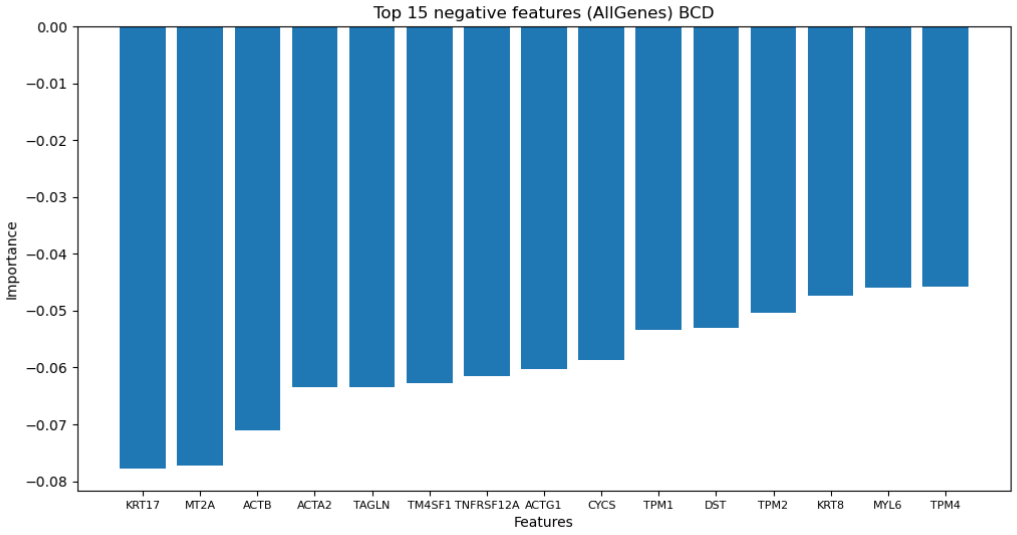


Figure 5.6: Negative feature importance by both linear (top) and non-linear (bottom) MLPs in BCD.

With respect to the negative importance, the gene MT2A is the most important gene with a negative importance (for the linear MLP), since this is an integrated gradient, this means that if the value of MT2A is high, the probability of being a BCSC is low, if the value is low however, the probability of BCSC is high. This result is similar to [30] is which the observe that the gene MT2A has low expression (low value) in both gastric and colorectal cancer.

For the non-linear MLP, the gene KRT17 takes the first place, leaving MT2A as the second most important gene with negative importance. Studies confirm that reduced expression (low value) of KRT17 predicts poor prognosis in breast cancer [25].

5.1.3 Random Forest

	Training F1	Validation F1	Test F1
StemMarkers - BCD	0,956	0,781	0,787
AllGenes - BCD	**0,958**	**0,823**	**0,832**
NonStemMarkers - BCD	0,957	0,786	0,807
StemMarkers - HSD	0,951	0,901	0,896
AllGenes - HSD	**0,952**	**0,929**	**0,931**
NonStemMarkers - HSD	0,951	0,918	0,919
StemMarkers - General	0,916	0,796	0,805
AllGenes - General	**0,929**	**0,832**	**0,835**
NonStemMarkers - General	0,910	0,796	0,798

Figure 5.7: Results obtained with Random Forests (best results in red).

Random Forests fit the data quite well, however, in some cases they fail to generalize well (validation and test F1 is low compared to training F1).

Again, when the RFs have access to all genes, the performance increases (in the case of HSD, RFs are close to MLPs or logistic regressors).

Feature Importance

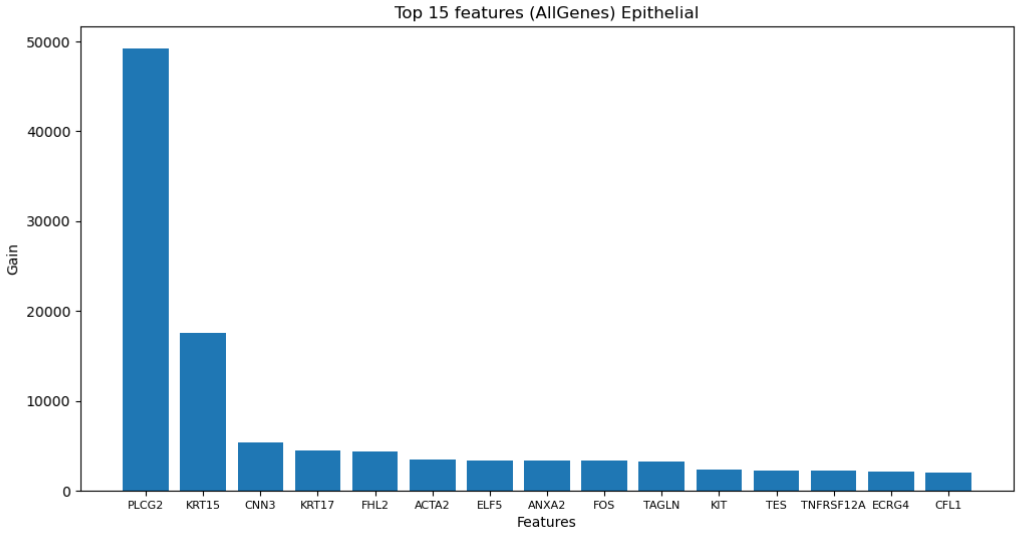


Figure 5.8: Average information gain across the random forest.

Once again, the gene PLCG2 is by far the gene that is most predictive of BCSCs. We can can see KRT17 again as the fourth most important gene. And in this case, the importance is the **average information gain of each gene across all nodes in the random forest**.

5.1.4 Embedder and MLP

Now, the results of the combination of embeddings and an MLP.

	Training F1	Validation F1	Test F1
StemMarkers - BCD	0,992	0,896	0,909
AllGenes - BCD	**0,999**	**0,916**	**0,911**
NonStemMarkers - BCD	0,983	0,897	0,907
StemMarkers - HSD	0,987	0,919	0,921
AllGenes - HSD	**0,988**	**0,922**	**0,943**
NonStemMarkers - HSD	0,986	0,935	0,929
StemMarkers - General	0,844	0,809	0,813
AllGenes - General	**0,913**	**0,850**	**0,851**
NonStemMarkers - General	0,778	0,752	0,754

Figure 5.9: Results for the combination of embeddings and an MLP (best results in red).

The validation and test F1 here vary, as they are sometimes lower or higher than the rest of the other models.

In the case of BCD, the embedder and MLP architecture performs worse than linear and non-linear MLPs. With the other datasets, sometimes the embedder performs better than linear MLPs but worse than non-linear MLPs.

This drop in performance could be attributed to the training and learning of the embeddings, since it's not uncommon that models learn embeddings which **both compress and lose some information**.

However, we would like to also analyze what this can tell us about the structure of the data:

- In both BCD and HSD, the test F1 for the training formats StemMarkers and NonStemMarkers are very similar. This potentially indicates that both formats hold the sufficient information to distinguish well between cancer stem cells. But this doesn't mean that **the features both formats have are independent of each other**, it could happen that are genes correlated with each other across both formats, so, each format holds similar predictive power.
- In the General dataset, the StemMarkers format achieves more test F1 than NonStemMarkers, which means that in this dataset, the StemMarkers hold more predictive information than the other one in this dataset.

Feature Importance and Embedding space

Now, let's see what the embedder learned and how the embeddings look (in BCD).

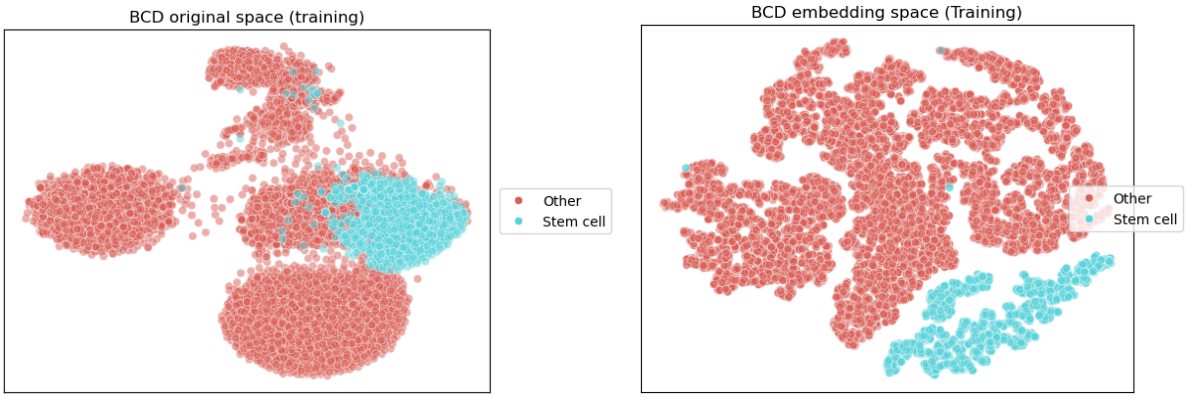


Figure 5.10: Original data space (left) versus learned embedding space (right).

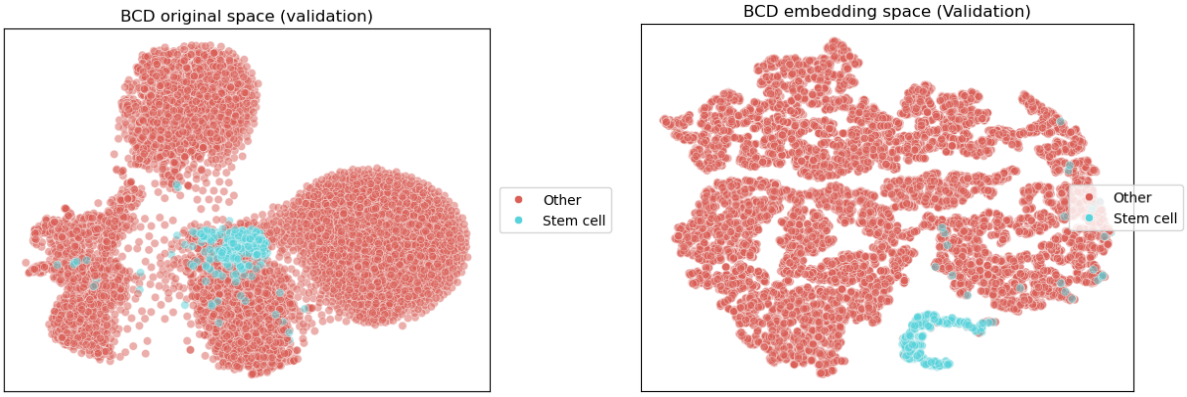


Figure 5.11: Original data space (left) versus learned embedding space (right).

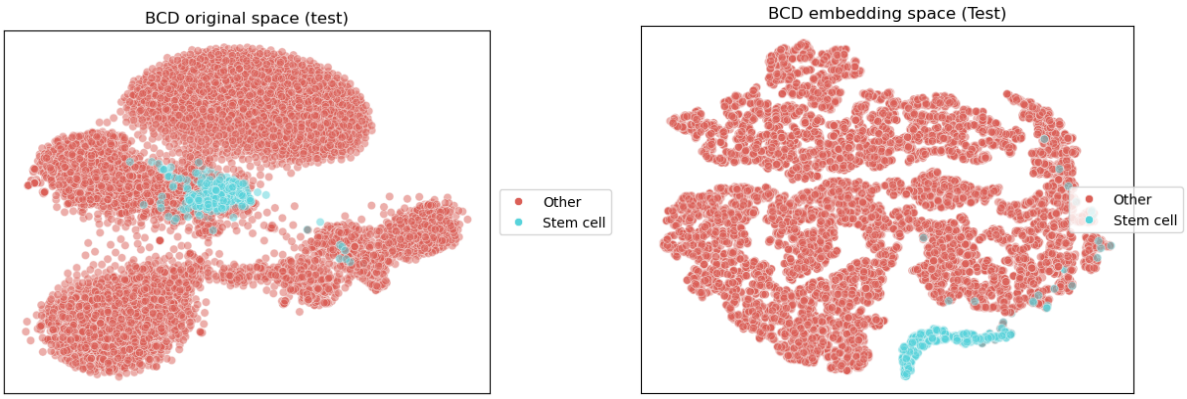
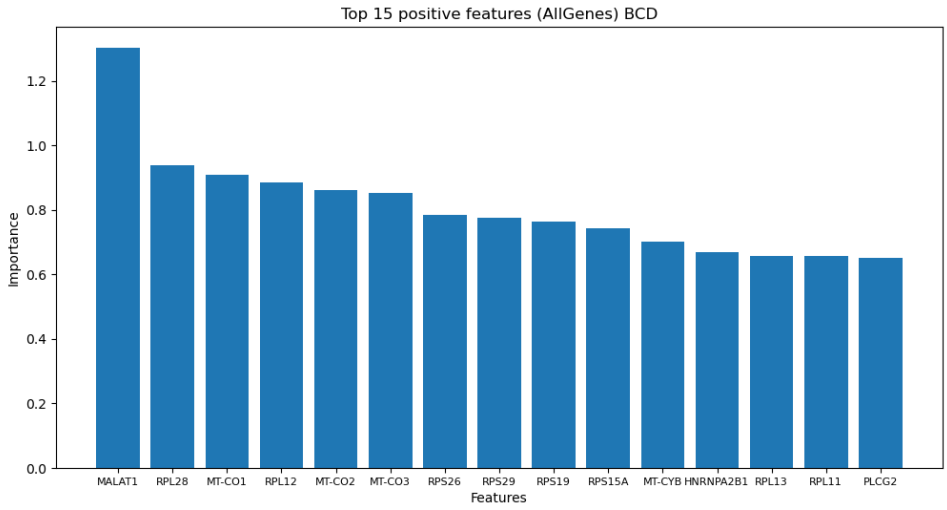


Figure 5.12: Original data space (left) versus learned embedding space (right).

We can see the embedder learns to separate well both classes (even though there are some cells that are not correctly separated).

Feature importance



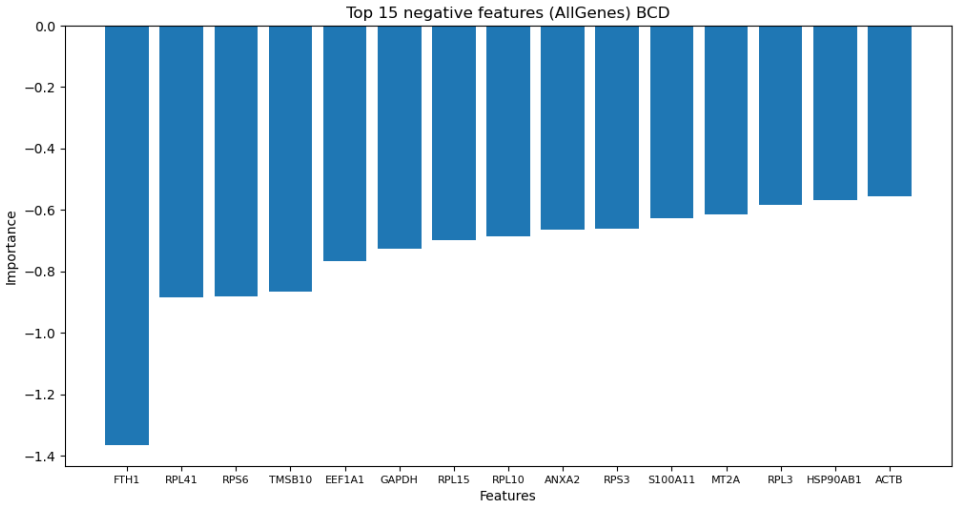


Figure 5.13: Feature importance by embedder and MLP on BCD (positive importance, on top, negative importance on bottom).

Here, the gene PLCG2 appears again, however, as the 15th most important gene, the first place is now taken by MALAT1, which is a gene known to induce metastasis in breast caancer [13], and thus correlated to cancer stem cells.

In the negative importance, we see FTH1 take the lead. Recall that this means that if the value of FTH1 is high then the probability of BCSC is low. Turns out, that the study [1] ,finds that FTH1 acts as a tumor suppressor in breast cancer, and that overexpression (FTH1 having a very high value) of FTH1 often associates with good prognosis (disease developing favorably, or getting better) in breast cancer.

5.1.5 Bagging

	Training F1	Validation F1	Test F1
StemMarkers - BCD	0,987	0,917	0,928
AllGenes - BCD	**0,984**	**0,923**	**0,935**
NonStemMarkers - BCD	0,992	0,913	0,916
StemMarkers - HSD	0,985	0,929	0,940
AllGenes - HSD	**0,982**	**0,942**	**0,952**
NonStemMarkers - HSD	0,987	0,935	0,946
StemMarkers - General	**0,935**	**0,878**	**0,880**
AllGenes - General	0,937	0,868	0,872
NonStemMarkers - General	0,778	0,752	0,754

Figure 5.14: Results obtained using bagging (best results in red).

When using Bagging, the method outperforms every other one, this boost in performance (although a very small one) can be attributed to bagging reducing Variance (the error in the dataset), but not to the architecture of the models, since the base models were MLPs and bagging neither reduces nor increases Bias.

However, captun does not support integrated gradients for Bagging so we cannot see the feature importance in this case.

5.2 Performance across datasets

Now, another objective of our work, is to measure the transferability, how well do the models generalize to other datasets.

First, let's get our baseline in order to compare to it, we run both CytoTRACE and ORIGINS algorithm on all three datasets, and measure the F1-Score.

	F1 Score
CytoTRACE (BCD)	0,00094
CytoTRACE (HSD)	0,02246
CytoTRACE (General)	0,00002
ORIGINS (BCD)	0,00058
ORIGINS (HSD)	0,08331
ORIGINS (General)	0,00000

Figure 5.15: F1-Score obtained by CytoTRACE and ORIGINS on all three datasets.

Now, we will gather models trained on one dataset and get the F1-Score on a different dataset (e.g. get a model trained on HSD and measure the F1 on BCD). In order to see if the patterns the models learn are transferable to the other dataset, or see if they have anything in common, potentially indicating that cancer stem cells have a common pattern or not.

Note: In this chapter all the results of transfering to will be listed, since we considered them important to discuss.

5.2.1 HSD to BSD

	F1
StemMarkers HSD to BCD (LinearMLP)	0,0028
StemMarkers HSD to BCD (Non-LinearMLP)	0,0011
StemMarkers HSD to BCD (Random Forest)	0,0000
StemMarkers HSD to BCD (Embedder)	0,0000
StemMarkers HSD to BCD (Bagging)	0,0000
StemMarkers HSD to BCD (Logistic)	**0,0285**

Figure 5.16: F1 Score resulting when transfering from HSD to BSD (StemMarkers).

	F1
AllGenes (Linear) HSD to BCD	**0,0598**
AllGenes (NonLinear) HSD to BCD	0,0011
AllGenes (RF) HSD to BCD	0,0000
AllGenes (Embedder) HSD to BCD	0,0000
AllGenes (Bagging) HSD to BCD	0,0000
AllGenes (Logistic) HSD to BCD	0,0289

Figure 5.17: F1 Score resulting when transfering from HSD to BSD (AllGenes).

	F1
NonStemMarkers (Linear) HSD to BCD	**0,0298**
NonStemMarkers (NonLinear) HSD to BCD	0,0038
NonStemMarkers (RF) HSD to BCD	0,0000
NonStemMarkers (Embedder) HSD to BCD	0,0000
NonStemMarkers (Bagging) HSD to BCD	0,0000
NonStemMarkers (Logistic) HSD to BCD	0,0288

Figure 5.18: F1 Score resulting when transfering from HSD to BSD (NonStemMarkers).

We can see that, again, the AllGenes format is better when generalizing to other datasets, and the linear MLP is the model that best generalizes.

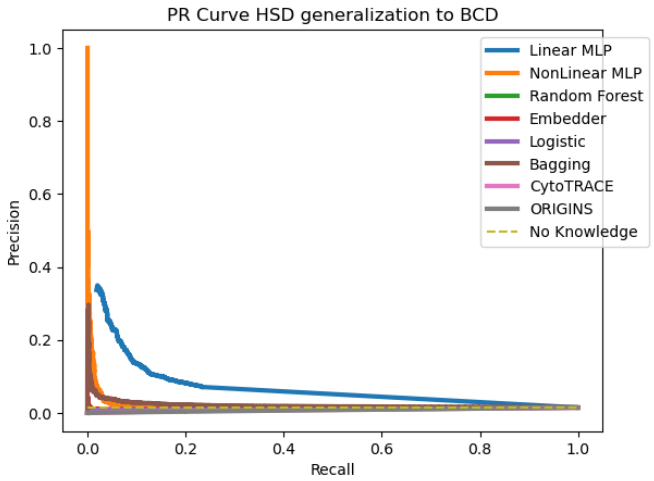


Figure 5.19: Precision-Recall Curve of all models and algorithms (All Genes, HSD to BCD). ORIGINS in grey color.

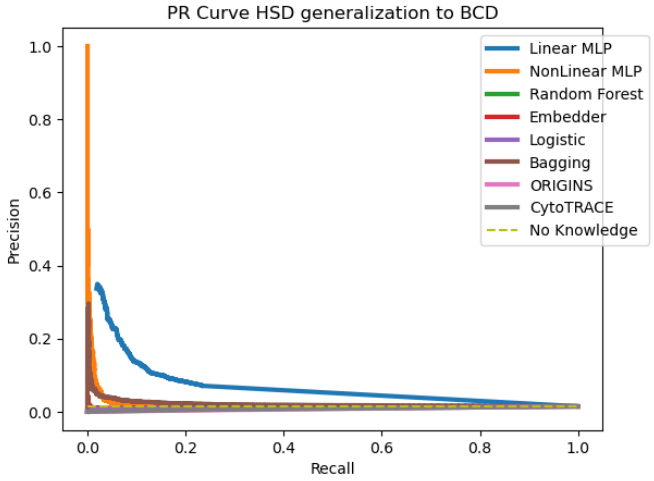


Figure 5.20: Precision-Recall Curve of all models and algorithms (All Genes, HSD to BCD). CytoTRACE in grey color.

Here we can see the precision-recall curve of the models transfered from HSD to BCD, **we show two plots because the curve of CytoTRACE overlaps with the ORIGINS curve, since they are both below the "no knowledge" curve**, so we show both, with their positions swapped, to make sure they can be seen. The rest of the curves also overlap, only the linear MLP, is only one that generalizes to some extent.

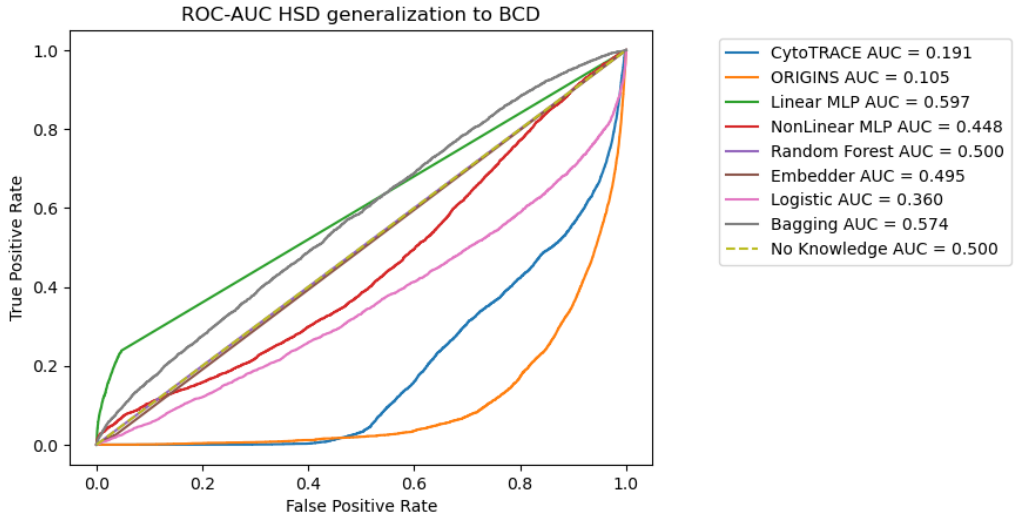


Figure 5.21: ROC-AUC curve of all models and algorithms (HSD to BCD, AllGenes).

Now, for the ROC-AUC curve, we can see that has the highest AUC is obtained with Linear MLPs and CytoTRACE and ORIGINS have the lowest AUC in BCD.

We can conclude that only the linear MLP trained on HSD does generalize to BCD, and somewhat outperforms CytoTRACE and ORIGINS.

5.2.2 General to BSD

	F1
StemMarkers General to BCD (LinearMLP)	0,0106
StemMarkers General to BCD (Non-LinearMLP)	0,0314
StemMarkers General to BCD (Random Forest)	0,0023
StemMarkers General to BCD (Embedder)	0,0017
StemMarkers General to BCD (Bagging)	0,0011
StemMarkers General to BCD (Logistic)	**0,0285**

Figure 5.22: F1 Score resulting when transfering from General to BSD (StemMarkers).

	F1
AllGenes (Linear) General to BCD	**0,0563**
AllGenes (NonLinear) General to BCD	0,0068
AllGenes (RF) General to BCD	0,0034
AllGenes (Embedder) General to BCD	0,0065
AllGenes (Bagging) General to BCD	0,0023
AllGenes (Logistic) General to BCD	0,0272

Figure 5.23: F1 Score resulting when transfering from General to BSD (AllGenes).

	F1
NonStemMarkers (Linear) General to BCD	**0,0320**
NonStemMarkers (NonLinear) General to BCD	0,0231
NonStemMarkers (RF) General to BCD	0,0000
NonStemMarkers (Embedder) General to BCD	0,01439
NonStemMarkers (Bagging) General to BCD	0,00620
NonStemMarkers (Logistic) General to BCD	0,02785

Figure 5.24: F1 Score resulting when transfering from General to BSD (NonStemMarkers).

We can see that, again, the AllGenes format is better when generalizing to other datasets, and the model that best generalizes is a linear MLP.

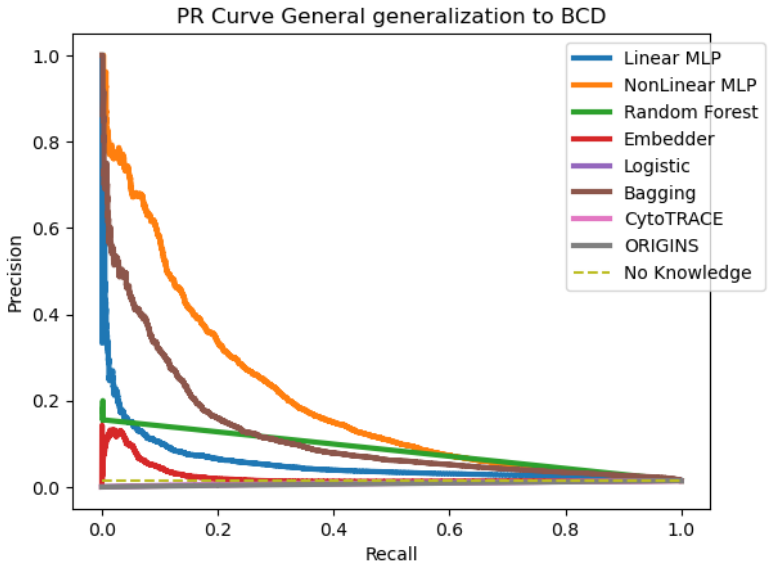


Figure 5.25: Precision-Recall Curve of all models and algorithms (General to BCD, AllGenes). ORIGINS in grey color.

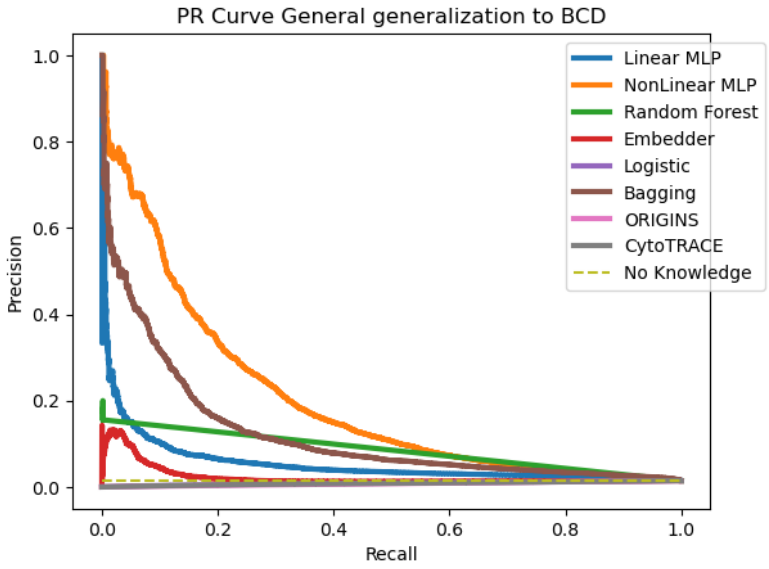


Figure 5.26: Precision-Recall Curve of all models and algorithms (General to BCD, AllGenes). CytoTRACE in grey color.

The precision-recall curve of the models transfered from General to BCD are better overall than when trained on HSD. Again, notice that, we show two plots because the curve of CytoTRACE overlaps with the ORIGINS curve.

Here, even though the linear MLP has better F1-Score the precision-recall curve is better for non-linear MLPs.

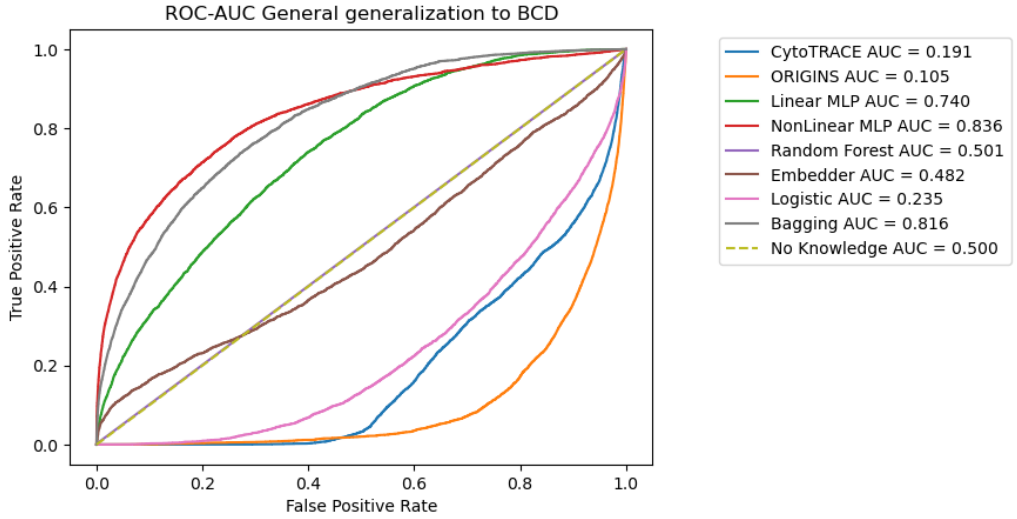


Figure 5.27: ROC-AUC curve of all models and algorithms (General to BCD, AllGenes).

In this case, we can see that has the highest AUC is a non-linear MLP and CytoTRACE and ORIGINS have the lowest AUC.

Also, models from General to BCD do generalize better than models from HSD, but they both outperform CytoTRACE and ORIGINS.

5.2.3 BCD to HSD

	F1
StemMarkers (Linear) BCD to HSD	0,0000
StemMarkers (NonLinear) BCD to HSD	0,0019
StemMarkers (RF) BCD to HSD	0,0000
StemMarkers (Embedder) BCD to HSD	0,0026
StemMarkers (Bagging) BCD to HSD	0,0000
StemMarkers (Logistic) BCD to HSD	**0,0320**

Figure 5.28: F1 Score resulting when transfering from BSD to HSD (StemMarkers).

	F1
AllGenes (Linear) BCD to HSD	0,0000
AllGenes (NonLinear) BCD to HSD	0,0000
AllGenes (RF) BCD to HSD	0,0000
AllGenes (Embedder) BCD to HSD	0,0052
AllGenes (Bagging) BCD to HSD	0,0000
AllGenes (Logistic) BCD to HSD	**0,0320**

Figure 5.29: F1 Score resulting when transfering from BSD to HSD (AllGenes).

	F1
NonStemMarkers (Linear) BCD to HSD	0,0000
NonStemMarkers (NonLinear) BCD to HSD	0,0000
NonStemMarkers (RF) BCD to HSD	0,0000
NonStemMarkers (Embedder) BCD to HSD	0,0005
NonStemMarkers (Bagging) BCD to HSD	0,0000
NonStemMarkers (Logistic) BCD to HSD	**0,0320**

Figure 5.30: F1 Score resulting when transfering from BSD to HSD (NonStemMarkers).

We can already tell that models trained with BCD generalize almost nothing to HSD, with only the logistic regressor having some F1-Score (surprisingly, all the training formats generalize equally).

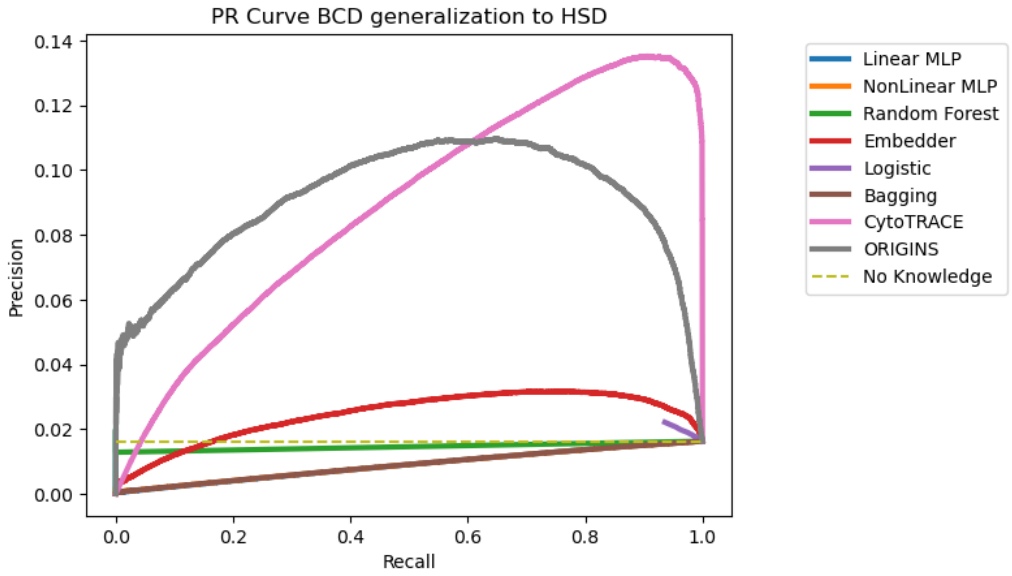


Figure 5.31: PR curve of all models and algorithms (BCD to HSD, and AllGenes format).

Just as expected, some models are below the "no knowledge line". The linear, non-linear, random forest and bagging models all have an F1-Score of 0 which means that either their precision or recall is also 0, thus, all of their curves overlap, however, they way the plots are made, only "bagging" seems to appear since the its curve is drawn over the other 3, but they are in the plot.

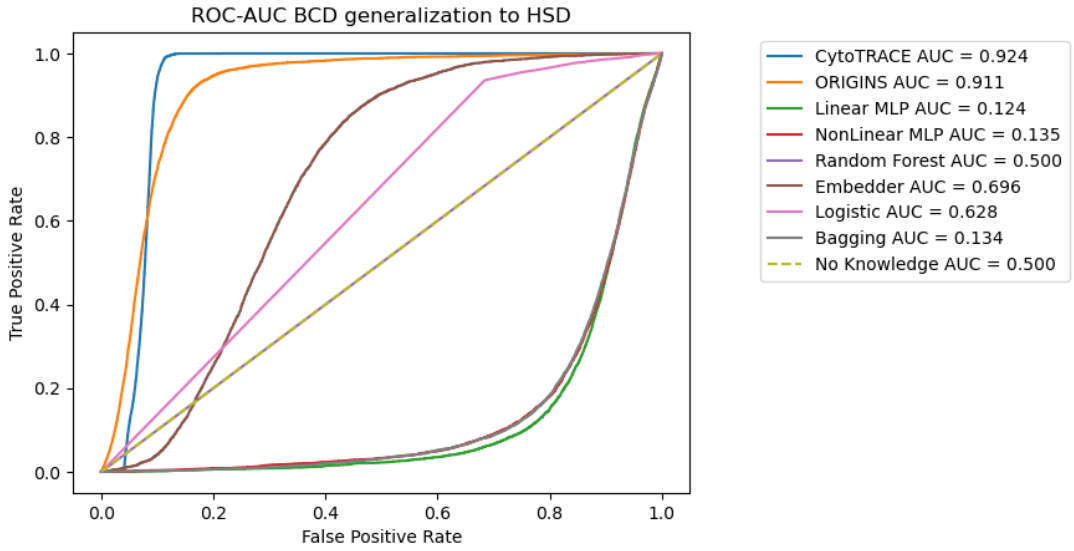


Figure 5.32: ROC-AUC curve of all models and algorithms (BCD to HSD, and AllGenes format).

We can inmediatly notice, how CytoTRACE and ORIGINS have a high AUC, but

a very low F1-Score, and how in both the precision recall curve and the AUC, only the embedder and the logistic regressor seem to generalize from BCD to HSD.

5.2.4 General to HSD

	F1
StemMarkers (Linear) General to HSD	0,0983
StemMarkers (NonLinear) General to HSD	**0,5840**
StemMarkers (RF) General to HSD	0,1169
StemMarkers (Embedder) General to HSD	0,0767
StemMarkers (Bagging) General to HSD	0,4568
StemMarkers (Logisitc) General to HSD	0,0276

Figure 5.33: F1 Score resulting when transfering from General to HSD (StemMarkers).

	F1
AllGenes (Linear) General to HSD	0,0848
AllGenes (NonLinear) General to HSD	0,4201
AllGenes (RF) General to HSD	0,1743
AllGenes (Embedder) General to HSD	0,2036
AllGenes (Bagging) General to HSD	**0,6256**
AllGenes (Logistic) General to HSD	0,0258

Figure 5.34: F1 Score resulting when transfering from General to HSD (AllGenes).

	F1
NonStemMarkers (Linear) General to HSD	0,0130
NonStemMarkers (NonLinear) General to HSD	**0,5302**
NonStemMarkers (RF) General to HSD	0,3993
NonStemMarkers (Embedder) General to HSD	0,0301
NonStemMarkers (Bagging) General to HSD	0,1237
NonStemMarkers (Logistic) General to HSD	0,0314

Figure 5.35: F1 Score resulting when transfering from General to HSD (NonStemMarkers).

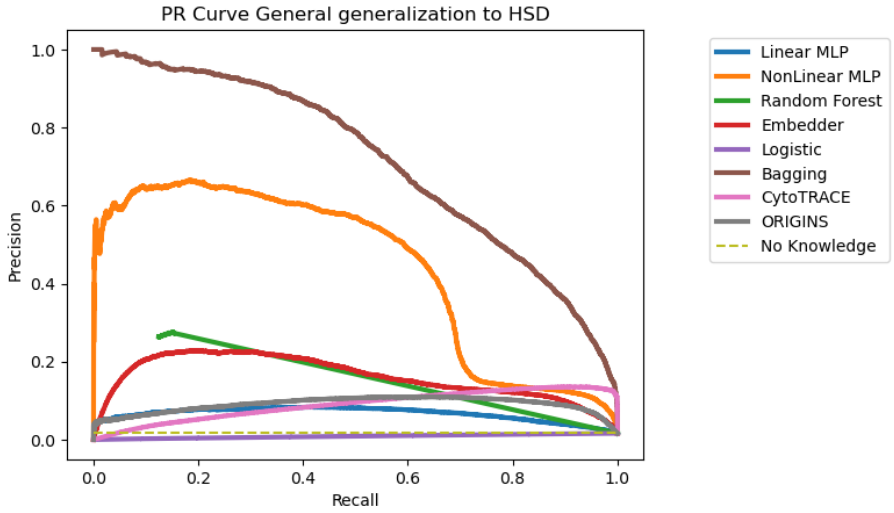


Figure 5.36: PR curve of all models and algorithms (General to HSD, and AllGenes format).

We can clearly see that Bagging and the non-linear MLP have very good generalization from General to HSD, with a very good precision and recall curve. This is expected, since, as we said in chapter 2, the General dataset contains samples of HSCs. With this we can confirm that models trained on General data learn patterns that transfer to HSD, and somewhat transfer to BCD.

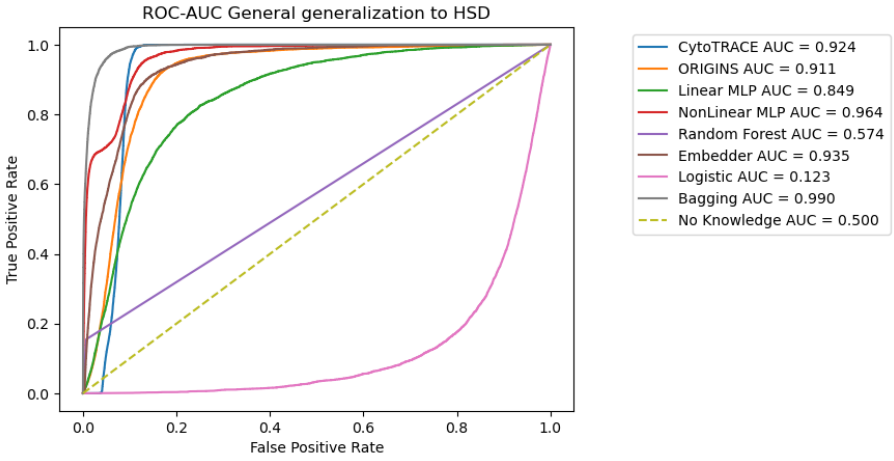


Figure 5.37: ROC-AUC curve of all models and algorithms (General to HSD, and AllGenes format).

Just like the previous one, CytoTRACE and ORIGINS, have high AUC but lower precision and recall, on the contrary, bagging and the non-linear MLP have an AUC of almost 1 and also a good precision recall curve.

5.2.5 BCD to General

	F1
StemMarkers (Linear) BCD to General	0,0422
StemMarkers (NonLinear) BCD to General	0,0933
StemMarkers (RF) BCD to General	0,1445
StemMarkers (Embedder) BCD to General	0,0313
StemMarkers (Bagging) BCD to General	0,0155
StemMarkers (Logistic) BCD to General	**0,1225**

Figure 5.38: F1 Score resulting when transfering from BSD to General (StemMarkers).

	F1
AllGenes (Linear) BCD to General	0,0071
AllGenes (NonLinear) BCD to General	0,0117
AllGenes (RF) BCD to General	0,0006
AllGenes (Embedder) BCD to General	0,0525
AllGenes (Bagging) BCD to General	0,0079
AllGenes (Logistic) BCD to General	**0,1381**

Figure 5.39: F1 Score resulting when transfering from BSD to General (AllGenes).

	F1
NonStemMarkers (Linear) BCD to General	0,0076
NonStemMarkers (NonLinear) BCD to General	0,0891
NonStemMarkers (RF) BCD to General	**0,1803**
NonStemMarkers (Embedder) BCD to General	0,0609
NonStemMarkers (Bagging) BCD to General	0,0070
NonStemMarkers (Logistic) BCD to General	0,1376

Figure 5.40: F1 Score resulting when transfering from BSD to General (NonStemMarkers).

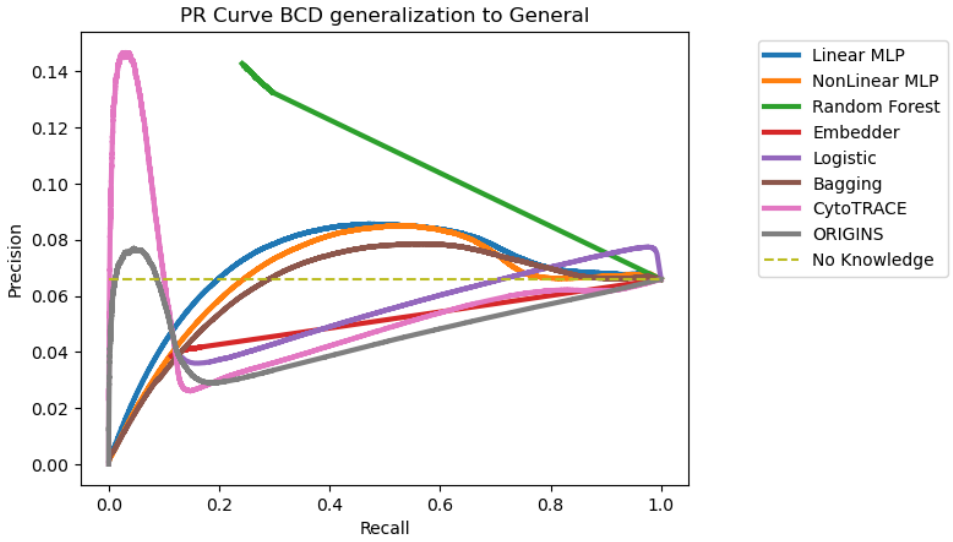


Figure 5.41: PR curve of all models and algorithms (BCD to General, and NonStemMarkers format).

In this case we see the training format is NonStemMarkers, which surprisingly generalizes better from BCD to the General dataset.

The RF has the best curve, and some other barely go over the "no knowledge" line, although the curve for the random forest is almost a straight line.

5.2.6 HSD to General

	F1
StemMarkers (Linear) HSD to General	0,1000
StemMarkers (NonLinear) HSD to General	0,0840
StemMarkers (RF) HSD to General	0,0002
StemMarkers (Embedder) HSD to General	0,0121
StemMarkers (Bagging) HSD to General	0,0829
StemMarkers (Logistic) HSD to General	**0,1196**

Figure 5.42: F1 Score resulting when transfering from HSD to General (StemMarkers).

	F1
AllGenes (Linear) HSD to General	0,0704
AllGenes (NonLinear) HSD to General	0,0354
AllGenes (RF) HSD to General	0,0003
AllGenes (Embedder) HSD to General	0,0903
AllGenes (Bagging) HSD to General	0,0460
AllGenes (Logistic) HSD to General	**0,1211**

Figure 5.43: F1 Score resulting when transfering from HSD to General (AllGenes).

	F1
NonStemMarkers (Linear) HSD to General	0,0480
NonStemMarkers (NonLinear) HSD to General	0,0238
NonStemMarkers (RF) HSD to General	0,0024
NonStemMarkers (Embedder) HSD to General	0,0788
NonStemMarkers (Bagging) HSD to General	0,0169
NonStemMarkers (Logistic) HSD to General	**0,1218**

Figure 5.44: F1 Score resulting when transfering from HSD to General (NonStemMarkers).

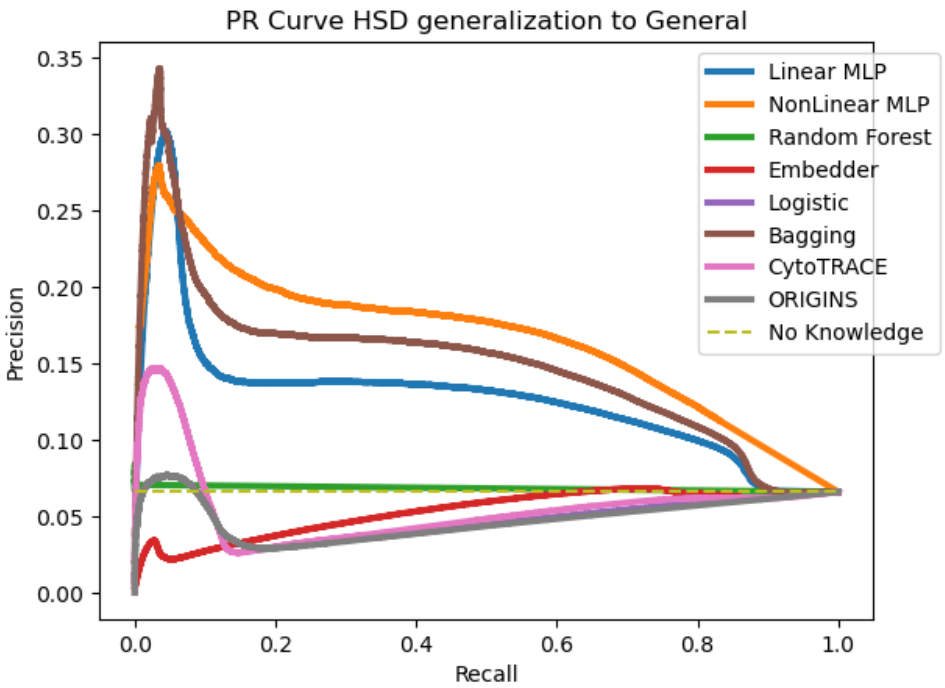


Figure 5.45: PR curve of all models and algorithms (HSD to General, and StemMarkers format).

Lastly, the generalization from HSD to General is good, but not as good as it was from General to HSD.

In this case, the StemMarker format seems make the models generalize better (the have better F1-Score over all), with the non-Linear, bagging and linear MLP being a lot better than the rest.

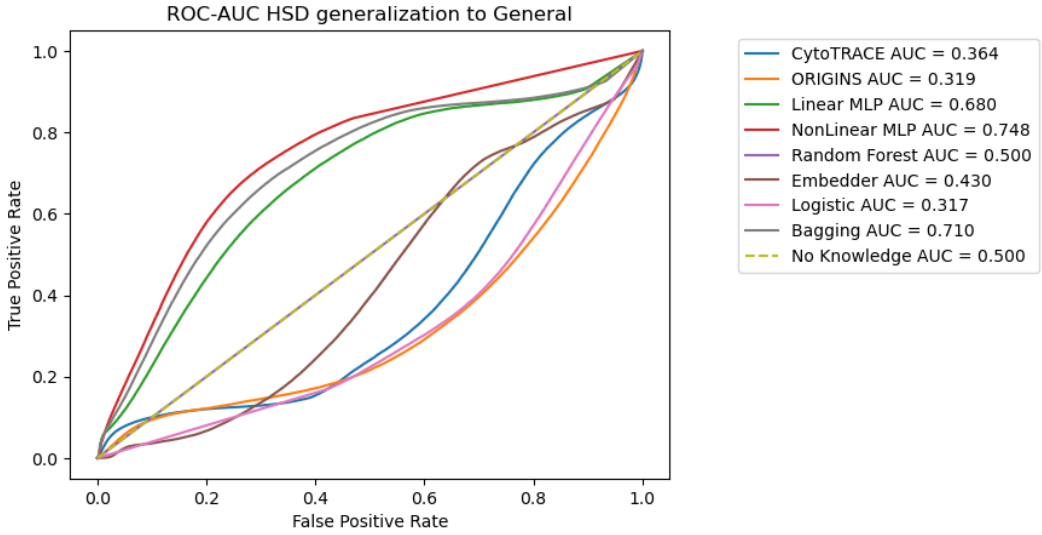


Figure 5.46: ROC-AUC curve of all models and algorithms (HSD to General, and StemMarkers format).

5.3 Signature overlap

One last important result is the poor overlap between gene signatures (a signature is a set of genes which is used to identify and isolate stem cells).

In [21], a table with the most important signatures to identify BCSCs has been put together. However, each time, more and more different signatures are discovered, with each author claiming that their signature identifies cancer stem cells just like the rest, so, the question of which signature is the best arises.

Here, we will show that it turns out that each signature identifies cancer stem cells differently, and they have almost nothing in common.

We take the signatures from [21] and our BCD dataset, and we index the cells which have their gene expression (values) closest to the signature.

Essentially, if a signature is $ABCG2^{+}$ we take all cells that have the gene ABCG2 positively expressed and get the top 100, 250 and 500 cells with the highest expression. Finally, we measure overlap between signatures, by computing the jaccard index of one signature with all the rest, and taking the average.

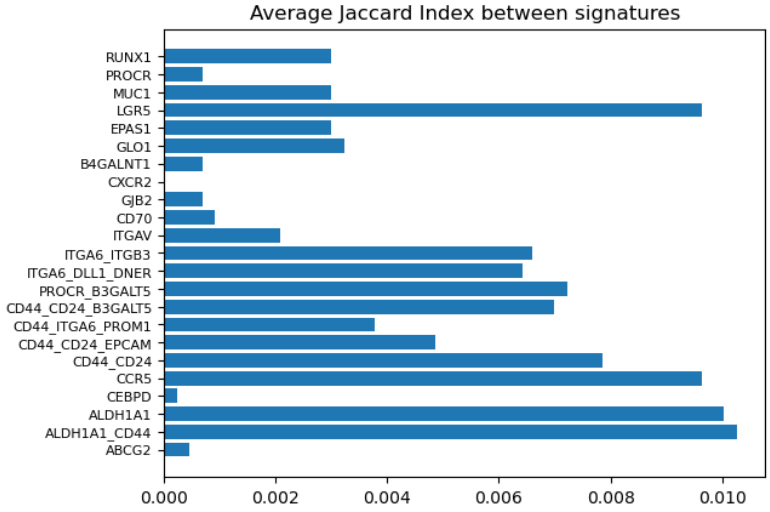


Figure 5.47: Average Jaccard index between signatures taking top 100 cells.

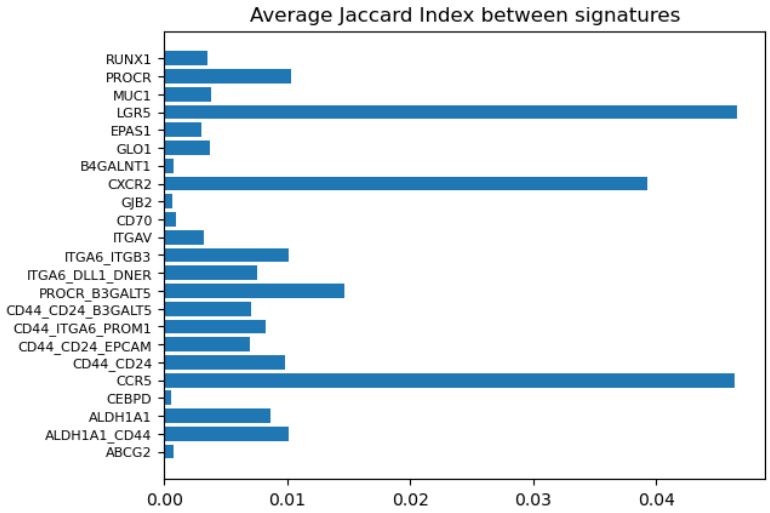


Figure 5.48: Average Jaccard index between signatures taking top 250 cells.

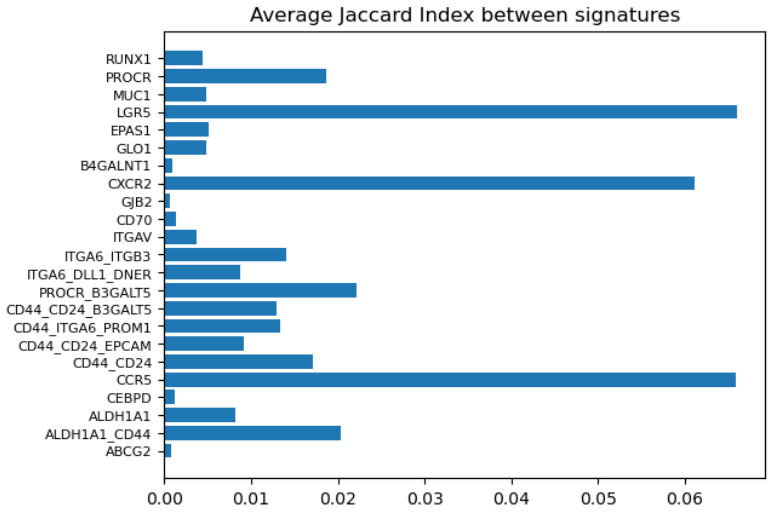


Figure 5.49: Average Jaccard index between signatures taking top 500 cells.

With this, we show that each signature **completely identifies stem cells differently than the rest** (each signature identifies one group of cells and there is no overlap between them). This completes our fourth objective.

This might sound obvious considering the previous sections, but this was done just to prove that the gene signtures by themselves are not enough to solve the problem of identifying BCSCs, since all signatures have completely different results. And that a Machine Learning approach was needed.

5.4 Discussion

We conclude this chapter and the work with a summary and discussion of the results

- Both CytoTRACE and ORIGINS perform very poorly in all three datasets. These two algorithms have the advantage that can be applied to any data given to them without training, however the actual results leave much room to improve. Our models, when generalizing, outperform both algorithms (in five out of six cases).
- All models are able to correctly fit each dataset and perform well in it, in the case of BCD, we can say machine learning can solve the task of predicting BCSC given a sequence of genes. This completes objective one.

- In (almost) all cases, models that learn with the AllGenes format outperform the other training formats, which potentially indicates that the combination of both stem cell markers and non-stem cell markers have more information together than both indepently. This adds to objective two.
- In all models, **non-stem cell markers have demostrated to contain information useful to predict cancer stem cells**. Whether this means that there are some stem cell markers hidden in those genes that have not been discovered yet, or the relation / correlation they have with the stem cell markers is beyond the scope of this work. This could be useful in future work, since this demonstrates that non-stem cell markers should not be ignored.
- The genes the models learn as most important (feature importance) have also been identified as related to cancer in medical literature, which confirms both the literature and the biological validity of what the models learn. This adds to objective two.
- Across datasets, the feature importance is very different, each dataset has different positive and negative most important genes, and they seem to have nothing in common. Also, in some cases, the generalization is very poor, which may point to the fact that **there is no common pattern among cancer stem cells of different cancer types**. This adds to objective two.
- Our models do generalize and transfer between datasets, although they outperform both CytoTRACE and ORIGINS, in some cases like General to HSD they cleary improve the algorithms, when transfering to BSD the improvement in very small. But it is still better, which was our objective. This completes objective three, and adds to objective one.

5.4.1 Sources of error

Here we will briefly point out some possible sources of error.

- Lack of data: Not many datasets exist where stem cells are labeled, so, there is little data to work with and the noise in the datasets could be a potential error source.
- Normalization unknown: When each dataset is created, the authors of the dataset normalize them in the way they see fitting, but they don't specify which method they used. This affects us because all three datasets could potentially have been normalized differently which affects measuring the transferability across datasets and the generalization power.

- Different genes recorded: In each dataset, there is a number of genes recorded that is not the same in all datasets (e.g. BCD has 33,000 genes and HSD 27,000).
  So, there are genes, and thus, information, that is missing in the rest of the datasets, the order of the recorded genes is also different which makes training models difficult.

5.4.2 Future work

In the future, if someone or we were to retake this work, they could try:

- Generate synthetic data using GANs or VAEs.
- Gather more data (although this could be expensive).
- Try more models and hyperparameters (although this will hardly improve the results, since they are already hard to improve).
- Use domain specific knowledge to further solve the task.
- Study in depth the rest of the genes that have been identified as important by the models.

List of Acronyms and Abbreviations

AUC Area Under the Curve.

BCSC Breast Cancer Stem Cell.

DT Decision Tree.

HSC Hematopoietic Stem Cell.

MLP Multilayer Perceptron.

RF Random Forest.

ROC Receiver Operating Characteristics.

Bibliography

[1] Ali A, Shafarin J, Abu Jabal R, Aljabi N, Hamad M, Sualeh Muhammad J, and Hamad M. Unnikannan H. Ferritin heavy chain (fth1) exerts significant antigrowth effects in breast cancer cells by inhibiting the expression of c-myc. 2021.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8564339/.

[2] Peng B, Hu J, and Fu X. Elane: an emerging lane to selective anticancer therapy. 2021.
URL: https://pubmed.ncbi.nlm.nih.gov/34599140/.

[3] Yang B, Zhang M Luo T, Lu Z, Xue X, and Fang G. The novel long noncoding rna rp11-357h14.17 acts as an oncogene by promoting cell proliferation and invasion in diffuse-type gastric cancer. 2017.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5442875/.

[4] Guo C, Liu S, Wang J, Sun MZ, and Greenaway FT. Actb in cancer. 2012.
URL: https://pubmed.ncbi.nlm.nih.gov/23266771/.

[5] Walsh CA, Akrap N, Garre E, Magnusson Y, Harrison H, Andersson D, Jonasson E, Rafnsdottir S, Choudhry H, Buffa F, Ragoussis J, Ståhlberg A, Harris A, and Landberg G. The mevalonate precursor enzyme hmgcs1 is a novel marker and key mediator of cancer stem cell enrichment in luminal and basal models of breast cancer. 2020.
URL: https://pubmed.ncbi.nlm.nih.gov/32692762/.

[6] Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. volume 44, page 5962–5979. Institute of Electrical and Electronics Engineers (IEEE), October 2022. doi: 10.1109/tpami.2021.3087709.
URL: http://dx.doi.org/10.1109/TPAMI.2021.3087709.

[7] Firdous, Shazia, Ghosh, Abhirupa, and Sudipto Saha. BCSCdb: a database of biomarkers of cancer stem cells. volume 2022, page baac082, 09 2022. doi: 10.1093/

database/baac082.
URL: https://doi.org/10.1093/database/baac082.

[8] L. Gao, X. Nie, and W. et al. Zhang. Identification of long noncoding rna rp11-89k21.1 and rp11-357h14.17 as prognostic signature of endometrial carcinoma via integrated bioinformatics analysis. 2020.
URL: https://pubmed.ncbi.nlm.nih.gov/32587476/.

[9] Gunsagar S. Gulati, Shaheen S. Sikandar, Daniel J. Wesche, Anoop Manjunath, Anjan Bharadwaj, Mark J. Berger, Francisco Ilagan, Angera H. Kuo, Robert W. Hsieh, Shang Cai, Maider Zabala, Ferenc A. Scheeren, Neethan A. Lobo, Dalong Qian, Feiqiao B. Yu, Frederick M. Dirbas, Michael F. Clarke, and Aaron M. Newman. Single-cell transcriptional diversity is a hallmark of developmental potential. volume 367, pages 405–411, 2020. doi: 10.1126/science.aax0249.
URL: https://www.science.org/doi/abs/10.1126/science.aax0249.

[10] Zheng HC, Xue H, Zhang CY, Shi KH, and Zhang R. The roles of btg1 mrna expression in cancers. 2022.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9633688/.

[11] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. 2017.

[12] Jackson JT, Mulazzani E, Nutt SL, and Masters SL. The role of plcγ2 in immunological disorders, cancer, and neurodegeneration. 2021.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8318911/.

[13] J. Kim, HL. Piao, and BJ. et al. Kim. Long noncoding rna malat1 suppresses breast cancer metastasis. 2018.
URL: https://doi.org/10.1038/s41588-018-0252-3.

[14] Tapsi Kumar, Kevin Nee, Runmin Wei, Siyuan He, Quy H. Nguyen, Shanshan Bai, Kerrigan Blake, Yanwen Gong, Maren Pein, Emi Sei, Min Hu, Anna Casasent, Aatish Thennavan, Jianzhuo Li, Tuan Tran, Ken Chen, Benedikt Nilges, Nachiket Kashikar, Oliver Braubach, Bassem Ben Cheikh, Nadya Nikulina, Hui Chen, Mediget Teshome, Brian Menegaz, Huma Javaid, Chandandeep Nagi, Jessica Montalvan, Delia F. Tifrea, Robert Edwards, Erin Lin, Ritesh Parajuli, Sebastian Winocour, Alastair Thompson, Bora Lim, Devon A. Lawson, Kai Kessenbrock, and Nicholas Navin. A spatially resolved single cell genomic atlas of the adult human breast. Cold Spring Harbor Laboratory, 2023. doi: 10.1101/2023.04.22.537946.
URL: https://www.biorxiv.org/content/early/2023/04/25/2023.04.22.537946.

[15] T. Lan, Y. Yan, and D. et al. Zheng. Investigating diagnostic potential of long non-coding rnas in head and neck squamous cell carcinoma using tcga database and clinical specimens. 2024.
URL: https://doi.org/10.1038/s41598-024-57987-y.

[16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019.

[17] Montaño-Samaniego M, Bravo-Estupiñan DM, Méndez-Guerrero O, Alarcón-Hernández E, and Ibáñez-Hernández M. Strategies for targeting gene therapy in cancer cells with tumor-specific promoters. 2020.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7768042/.

[18] Prpić M, Franceschi M, Romić M, Jukić T, and Kusić Z. Thyroglobulin as a tumor marker in differentiated thyroid cancer - clinical considerations. 2018.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6536288/.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

[20] Hassan MK, Kumar D, Naik M, and Dixit M. The expression profile and prognostic significance of eukaryotic translation elongation factors in different cancers. 2018.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5771626/.

[21] Shan NL, Shin Y, Yang G, Furmanski P, Suh N. Breast cancer stem cells: A review of their characteristics, and the agents that affect them. Mol Carcinog. Breast cancer stem cells: A review of their characteristics and the agents that affect them. 2021. doi: doi:10.1002/mc.23277.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7855917/.

[22] Singh RK, Saini SK, Prakasam G, Kalairasan P, and Bamezai RNK. Role of ectopically expressed mtdna encoded cytochrome c oxidase subunit i (mt-coi) in tumorigenesis. 2019.
URL: https://pubmed.ncbi.nlm.nih.gov/31299394/.

[23] Orit Rozenblatt-Rosen, Aviv Regev, Bo Li, Monika S Kowalczyk, Michal Slyper, Gaublomme Jellert, Marcin Tabaka, Orr Ashenberg, Julia Waldman, Danielle Dionne, Knecht Abigail, Ma Hui, Yiming Yang, Orit Rozenblatt-Rosen, Mallory Ann Freeberg, Danielle Welter, and Enrique Sapena Ventura. A single cell immune cell atlas of human hematopoietic system. 2022.
URL: https://explore.data.humancellatlas.org/projects/cc95ff89-2e68-4a08-a234-480eca21ce79.

[24] He S, Ding Y, Ji Z, Yuan B, Chen J, and Ren W. Hopx is a tumor-suppressive biomarker that corresponds to t cell infiltration in skin cutaneous melanoma. 2023.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10286411/.

[25] Tang S, Liu W, Yong L, Liu D, Lin X, Huang Y, Wang H, and Cai F. Reduced expression of krt17 predicts poor prognosis in her2high breast cancer. 2022.
URL: https://pubmed.ncbi.nlm.nih.gov/36139022/.

[26] Daniela Senra, Nara Guisoni, and Luis Diambra. Origins: A protein network-based approach to quantify cell pluripotency from scrna-seq data. volume 9, page 101778, 2022. doi: https://doi.org/10.1016/j.mex.2022.101778.
URL: https://www.sciencedirect.com/science/article/pii/S2215016122001583.

[27] Höpner SS, Raykova A, Radpour R, Amrein MA, Koller D, Baerlocher GM, Riether C, and Ochsenbein AF. Light/ltβr signaling regulates self-renewal and differentiation of hematopoietic and leukemia stem cells. 2021.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7887212/.

[28] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. 2017.

[29] Chen X, Zhao L, Yu T, Zeng J, and Chen M. Spink2 is a prognostic biomarker related to immune infiltration in acute myeloid leukemia. 2022.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8829596/.

[30] Liu X, Quan J, Shen Z, Zhang Z, Chen Z, Li L, Li X, Hu G, and Deng X. Metallothionein 2a (mt2a) controls cell proliferation and liver metastasis by controlling the mst1/lats2/yap1 signaling pathway in colorectal cancer. 2022.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9158144/.

[31] Tang Y, Peng X, Huang X, and Li J. Actin gamma 1 is a critical regulator of pancreatic ductal adenocarcinoma. 2021.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9212121/.

[32] H. Zhang, B. Cui, and Y. et al. Zhou. B2m overexpression correlates with malignancy and immune signatures in human gliomas. 2021.
URL: https://doi.org/10.1038/s41598-021-84465-6.

[33] Y. Zhang, H. C. Sun, W. Zhang, T. T. Fu, S. J. Huang, M. J. Mou, J. S. Zhang, J. Q. Gao, Y. C. Ge, Q. X. Yang, and F. Zhu. Cellstar: a comprehensive resource for single-cell transcriptomic annotation. nucleic acids research. 2023.
URL: http://cellstar.idrblab.net/download.

Appendix A

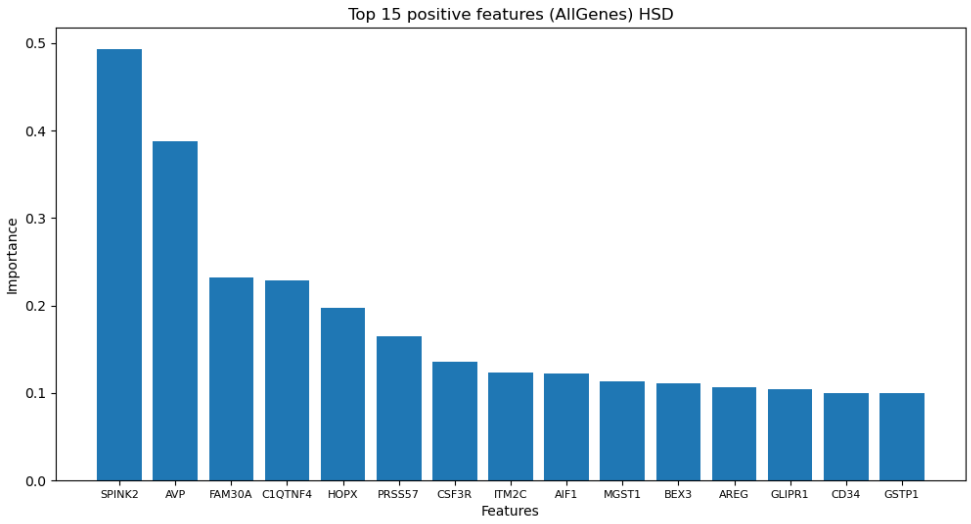
Supplemental figures and discussion

Here we will show and discuss the feature importance of the rest of the data (HSD and general):

A.1 Extra: Feature importance on HSD

As we said earlier, in the results section we only focused on BCD in order to not artificially extend the length of the chapter.

A.1.1 Logistic Regressors



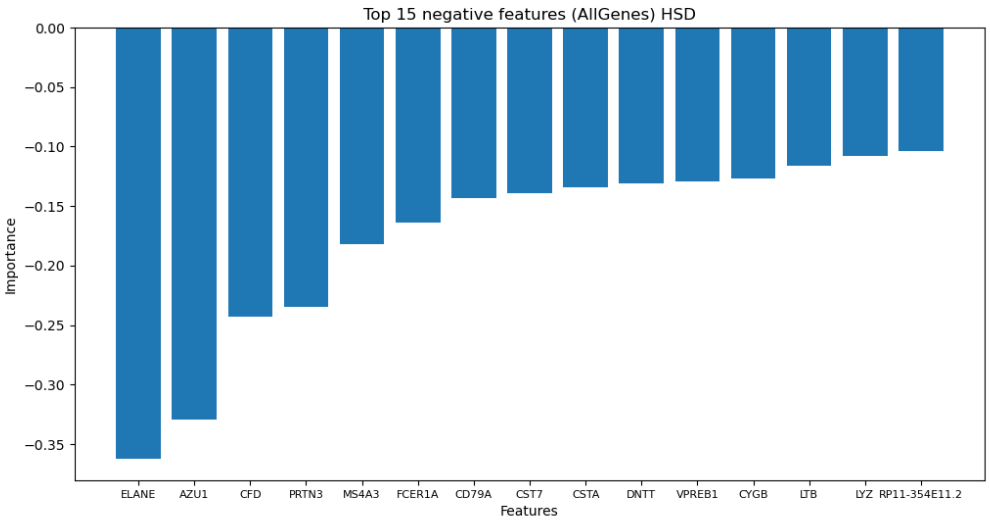


Figure A.1: Feature importance by logistic regressors on HSD (positive importance, on top, negative importance on bottom).

In the haematopoietic data, we can already see that the most important genes are completely different from BCD, in this case, the SPINK2 gene is the most predictive of HSCs. This gene has already been identified as related to leukemia in blood cells [29].

On the flip side, the ELANE gene is the gene with the most negative importance. There is no clear relationship with ELANE and cancer, however, ELANE has already been observed to selectively kill a wide range of cancer cells while sparing proximal non-cancer cells and significantly attenuate tumorigenesis [2]. Which would make sense with the results, since, a high ELANE value means low probability of HSCs, and thus, an HSC with high ELANE expression would die.

A.1.2 MLPs

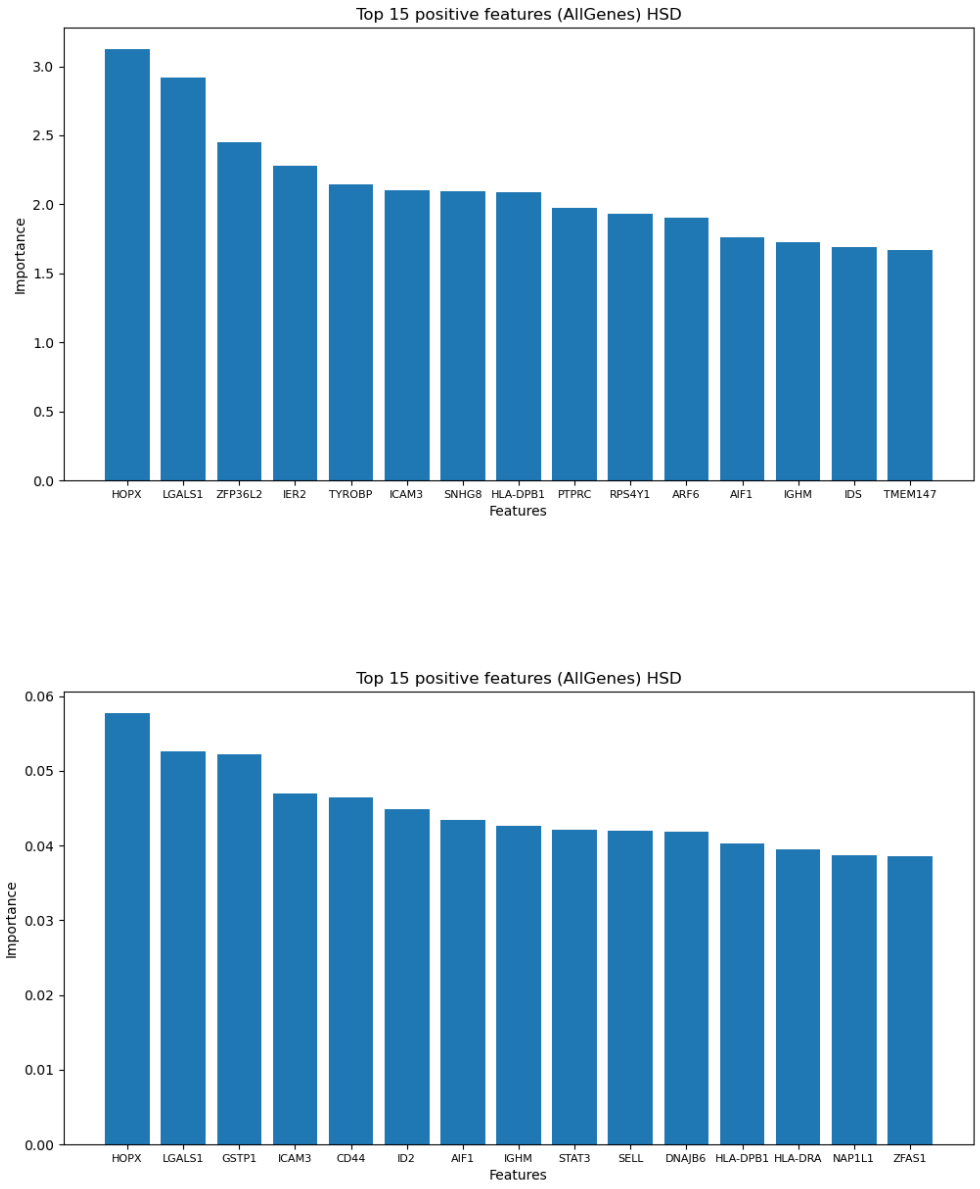


Figure A.2: Positive feature importance by linear MLP (top) and non-linear MLP (bottom).

We can observe that the positive importance greatly varies in HSD with different models, with logistic regressors, the biggest predictor of HSCs was SPINK2, here, for both models is the HOPX gene. It turns out, that HOPX is known to be involved in regulating the homeostasis (any self-regulating process in order to survive) of hematopoietic stem cells

and is closely related to the development of tumors such as breast cancer, nasopharyngeal carcinoma, and head and neck squamous cell carcinoma [24].

We should also note that the non-linear MLP trained on HSD also identifies the gene CD44 as the 5th best prediction of HSCs. CD44 is a know very important BCSC marker [21].

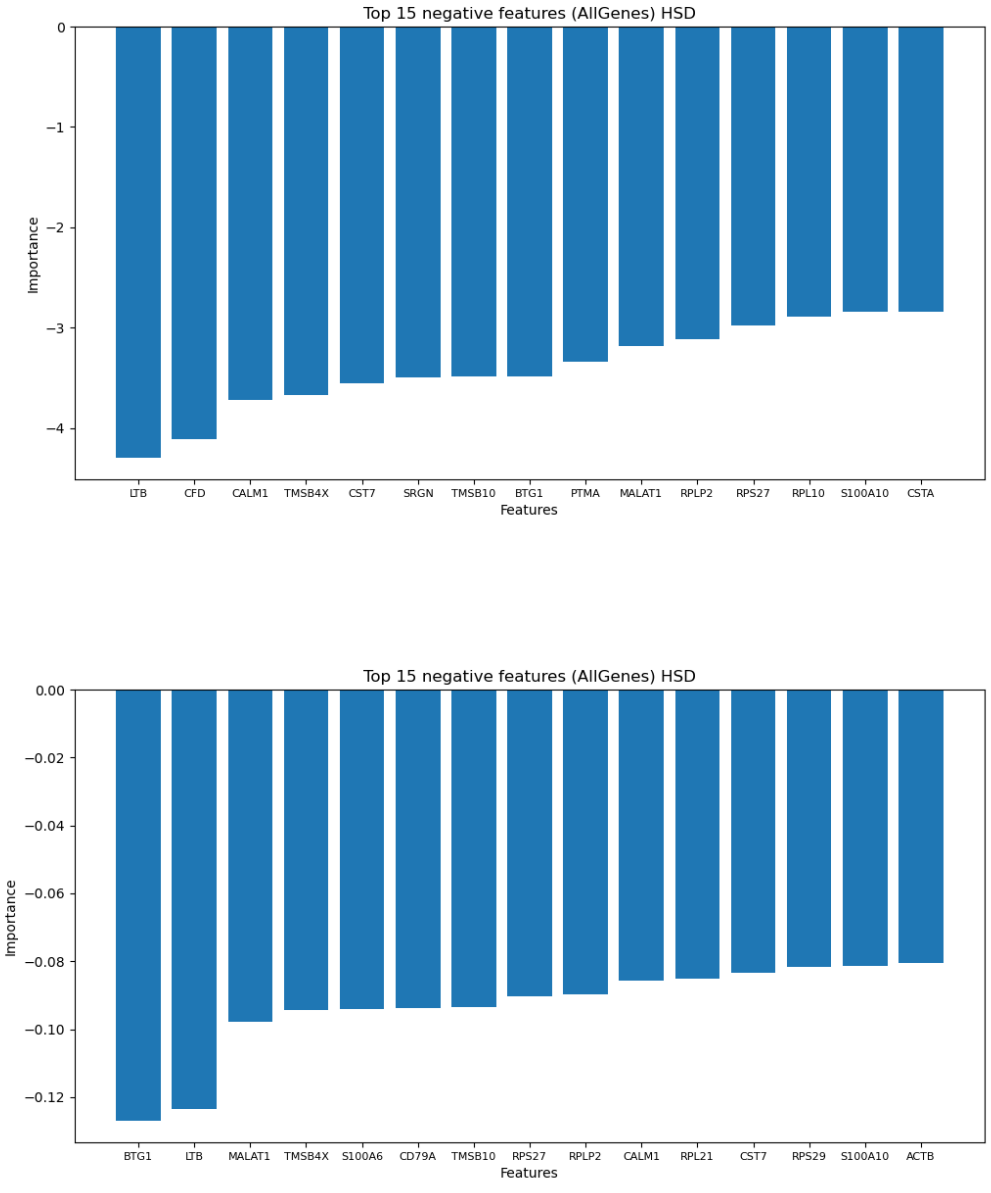


Figure A.3: Negative feature importance by linear MLP (top) and non-linear MLP (bottom).

As for the negative features, the linear MLP identifies the gene LTB as the most

important and the non-linear MLP identifies the BTG1 gene (even though for the linear one, BTG1 is in the 5th position and for the non-linear is 2nd).

The LTB gene has been found to regulate self-renewal and differentiation of hematopoietic and leukemia stem cells [27]. On the other hand, the BTG1 gene has been found to inhibit proliferation and cell cycle progression [10]. Since stem cells proliferate and BTG1 inhibits that process it makes sense that the model thinks that a high BTG1 value correlated with low probability of HSC.

A.1.3 Random Forest

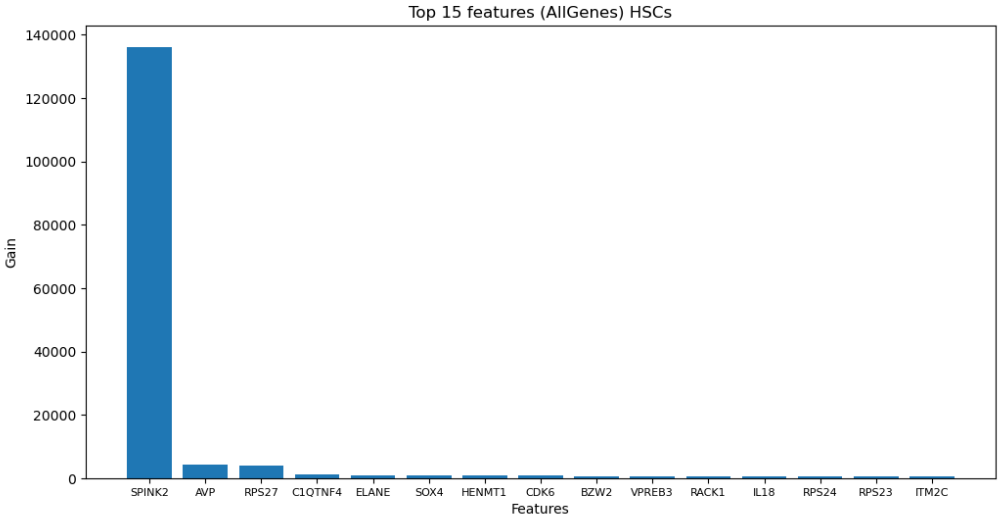


Figure A.4: Average information gain across the random forest.

Just like the logistic regressor, the SPINK2 gene is the most important gene for the RF (and by a lot). Notice that ELANE and SOX4 (a very important BCSC marker [21] appear).

A.1.4 Embedder

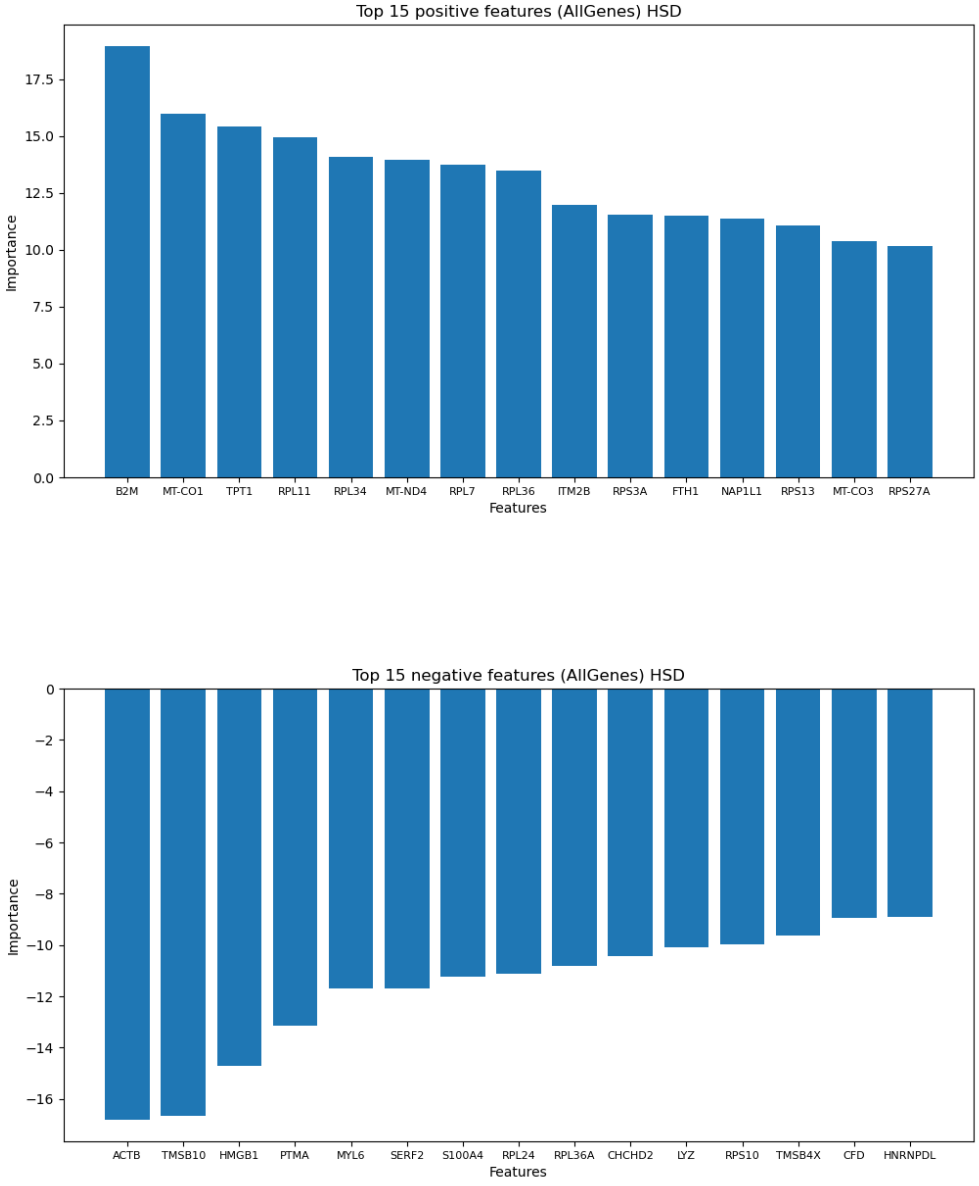


Figure A.5: Positive and negative feature importance by embedder and mlp architecture (top and bottom, respectively).

The embedder, identifies B2M as the most predictive gene of stem cells. It has been found that if B2M is over-expressed (very high gene expression), it correlates with malignancy and immune signatures in human gliomas [32].

On the other hand the ACTB gene has been know for some time that is closely

associated with liver, melanoma, renal, colorectal, gastric, pancreatic, esophageal, lung, breast, prostate, ovarian cancers, leukemia and lymphoma [4]

Now, let's see how the embeddings look like for HSD:

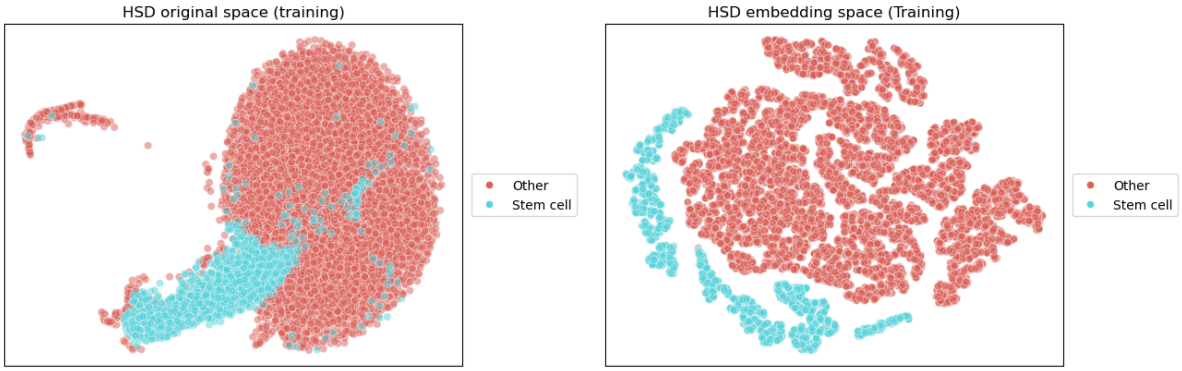


Figure A.6: Original data space (left) versus learned embedding space (right).

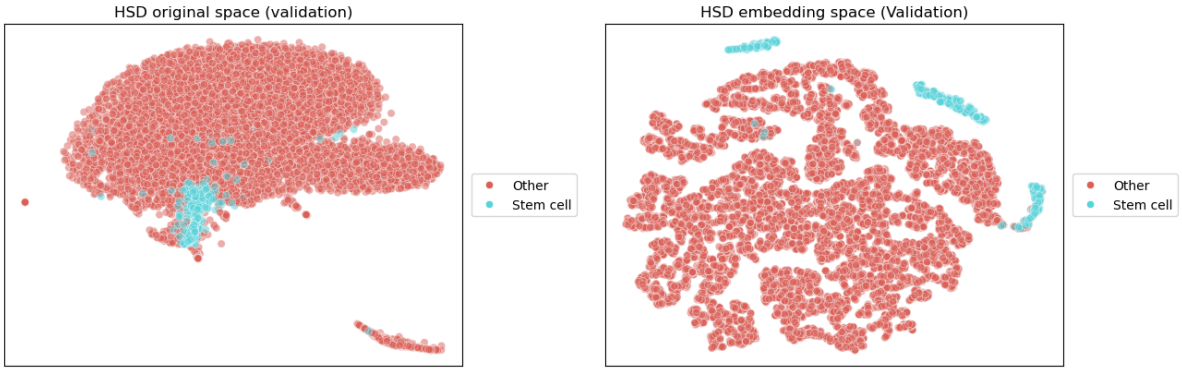


Figure A.7: Original data space (left) versus learned embedding space (right).

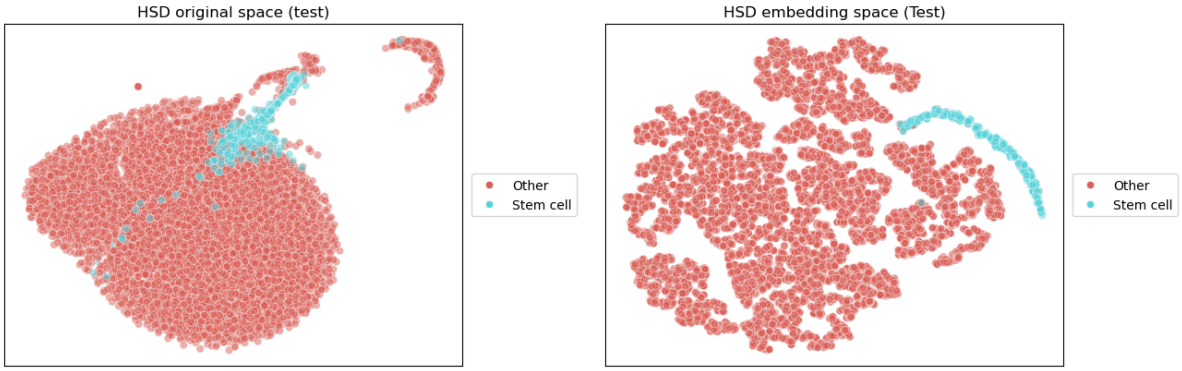


Figure A.8: Original data space (left) versus learned embedding space (right).

A.2 Extra: Results on General data

A.2.1 Logistic Regressor

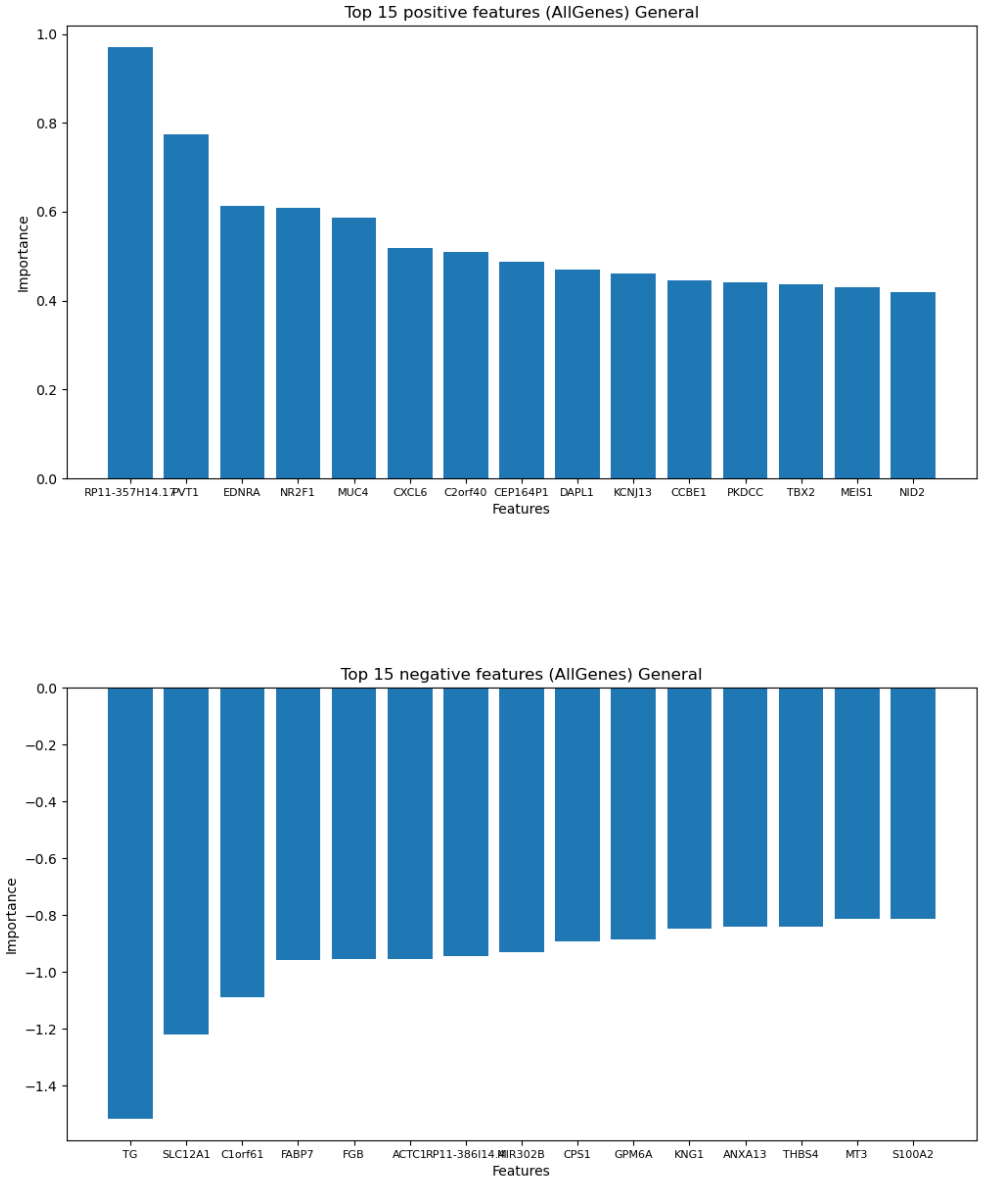


Figure A.9: Feature importance by logistic regressors on General data (positive importance, on top, negative importance on bottom).

In the case of the general data, we find the gene RP11–357H14.17 as the most predictive of cancer stem cells, there are already some articles and work that indicate that this

gene is correlated to gastric cancer [3], endometrial carcinoma [8], and head and neck squamous cell carcinoma [15], which points this gene is in fact correlated with different types of cancer (which makes sense considering this is a general dataset).

TG is the gene identified with the most negative importance, this gene has been identified by some studies as a marker in thyroid cancer [18].

A.2.2 MLPs

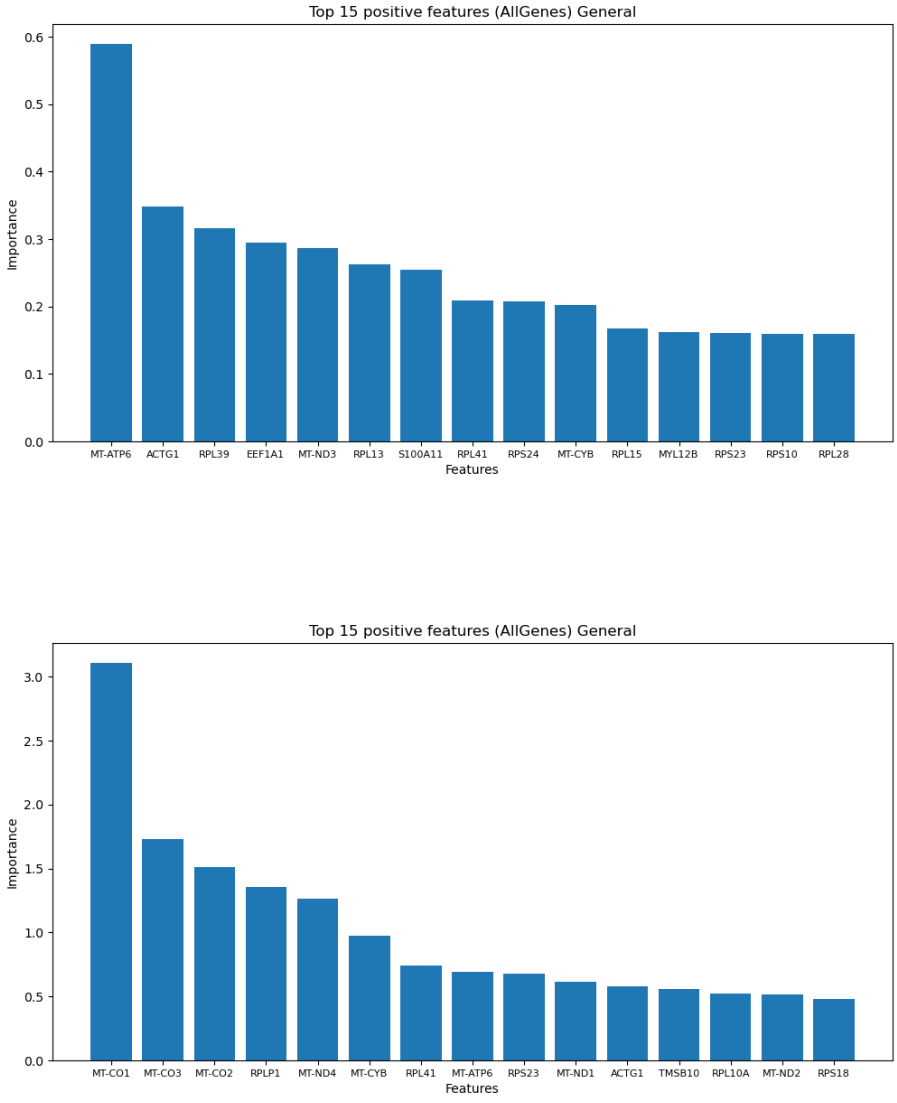


Figure A.10: Positive feature importance by linear MLP (top) and non-linear MLP (bottom).

In the case of the MLPs, the gene MT-ATP6 is the most predictive of stem cells, although there seems to be no papers or studies which relate this gene to cancer. However, the 2nd most important gene, ACTG1 is related to pancreatic ductal adenocarcinoma [31].

For the non-linear MLP, MT-CO1 seems to be the most predictive, turns this gene has been identified as frequent in various cancer types [22].

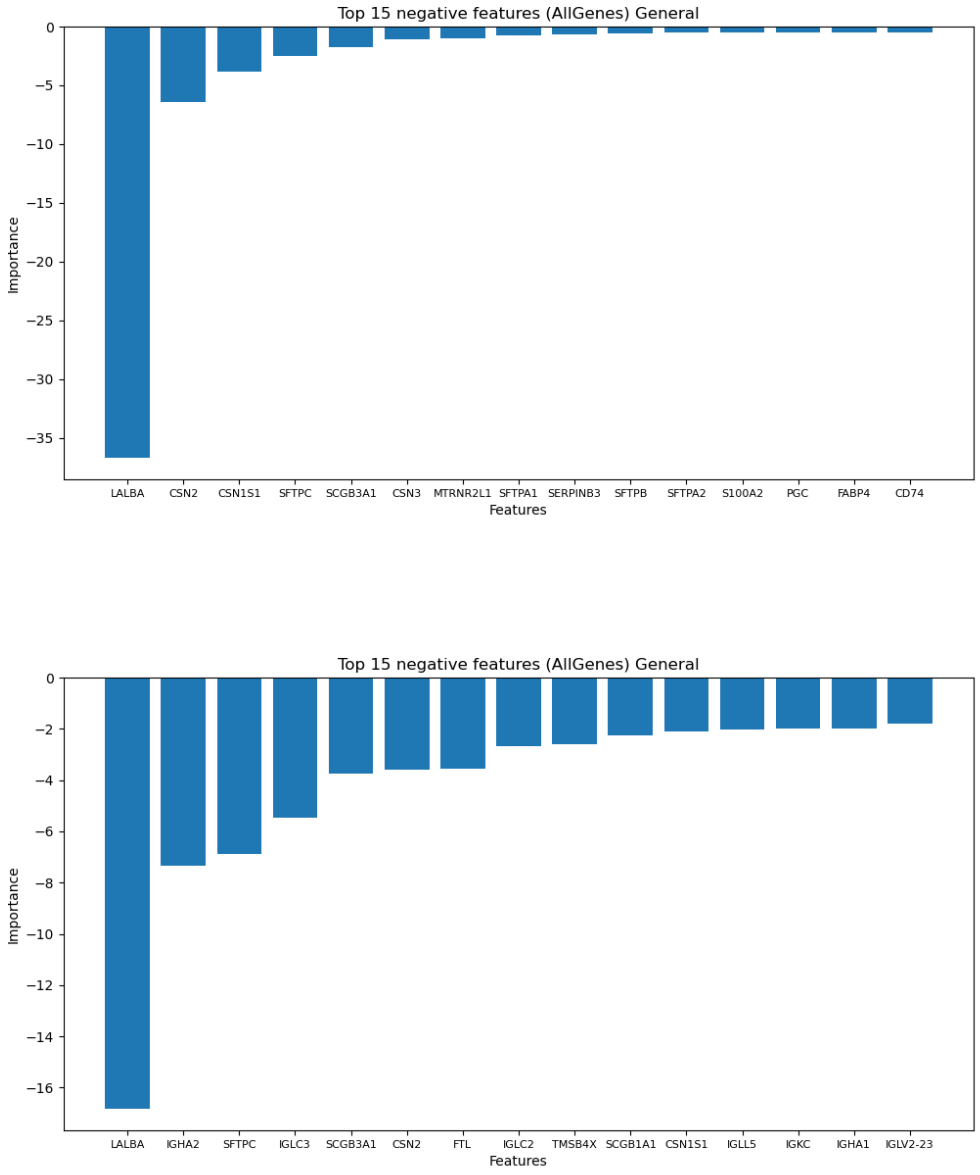


Figure A.11: Negative feature importance by linear MLP (top) and non-linear MLP (bottom).

When looking at the negative importance, both MLPs identify LALBA as the most predictive of non-stem cells, LALBA is known to be a very active gene in breast can-

cer [17], but inactive in all the other types. Since the are no examples of BCSC in the General dataset, this could be the cause of the importance given to this gene, since, for the models, a low value of LALBA means high probability of being a stem-cell.

A.2.3 Random Forest

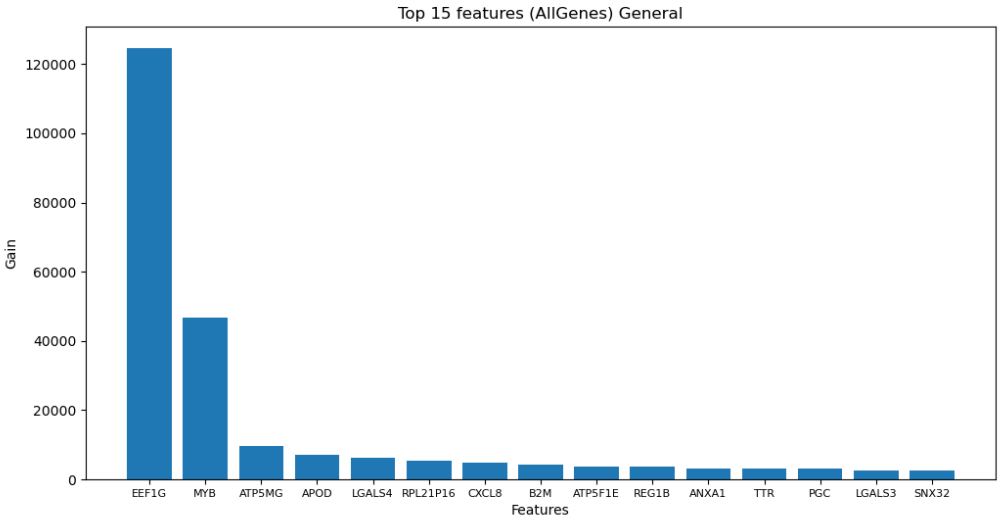


Figure A.12: Average information gain across the random forest.

Here, the gene EEF1G seems to be most important gene for the RF, this gene has been observed to be over-expressed in lung cancer [20].

A.2.4 Embedder

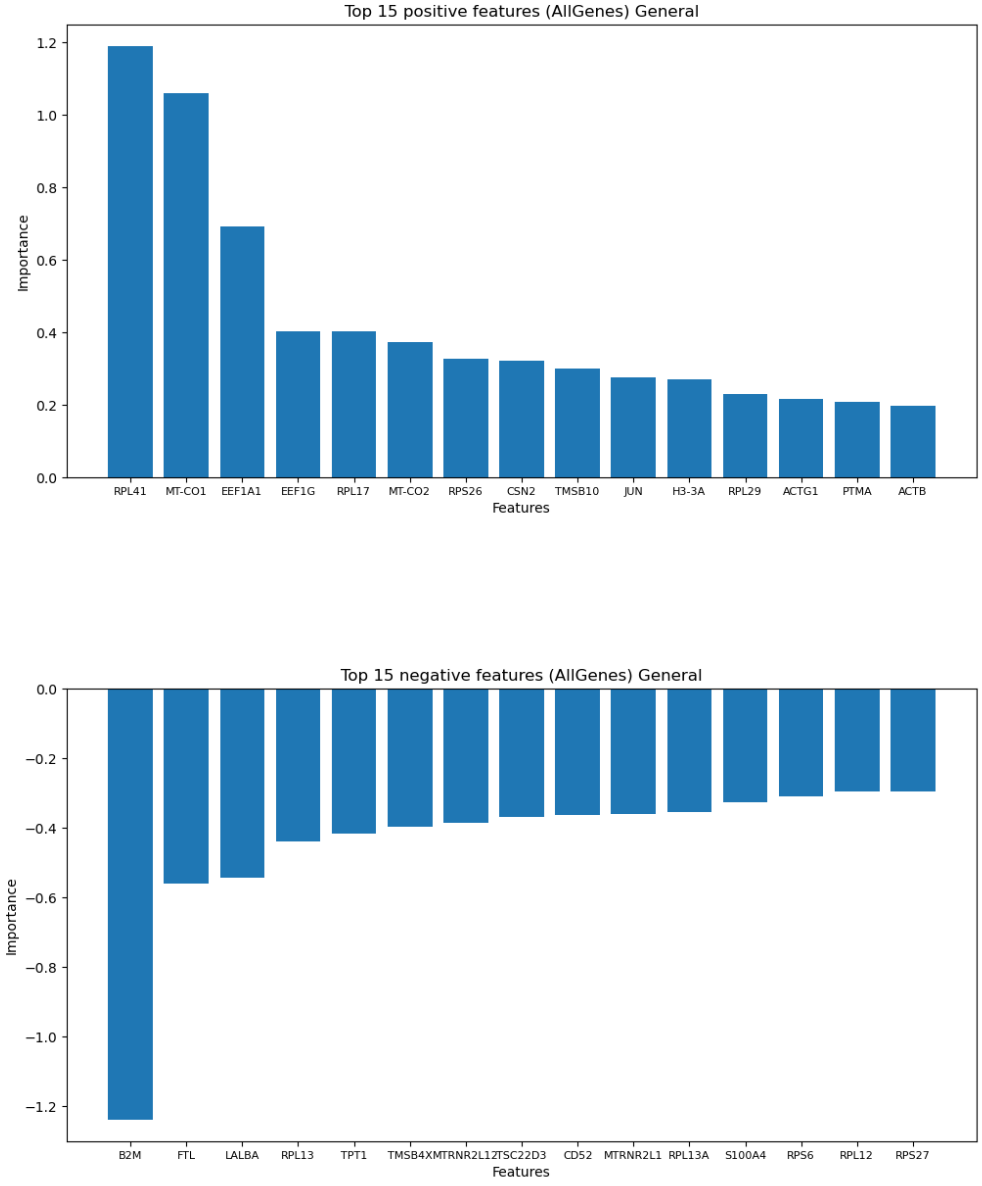


Figure A.13: Positive and negative feature importance by embedder and mlp architecture (top and bottom, respectively).

The embedder, identifies RPL41 as the most predictive gene of stem cells. We should note that RPL41 is a ribosomal gene, ribosomal genes are a type of general called *housekeeping* genes, these genes are always expressed in all cells of an organism, so it's surprising that this would be a prediction of stem-cells.

On the other hand the B2M gene has appeared as the most predictive gene forHSCs by the embedder trained on HSD. So it's very surprising to see that in the General data it's the most predictive of non-stem cells.

Lastly we will show how the learned embeddings are distributed for the General dataset:

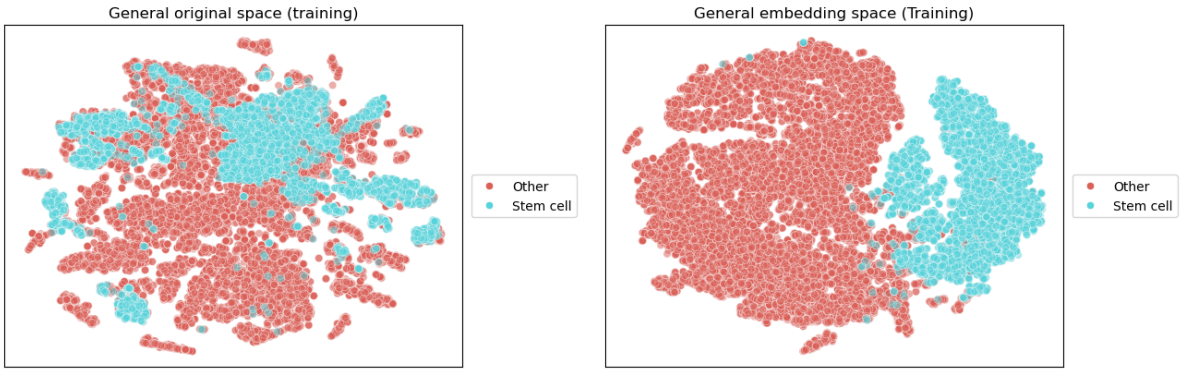


Figure A.14: Original data space (left) versus learned embedding space (right).

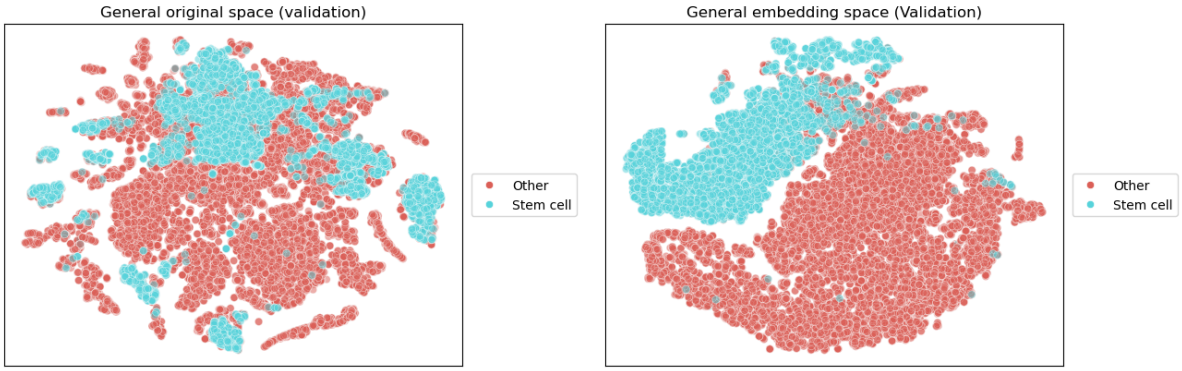


Figure A.15: Original data space (left) versus learned embedding space (right).

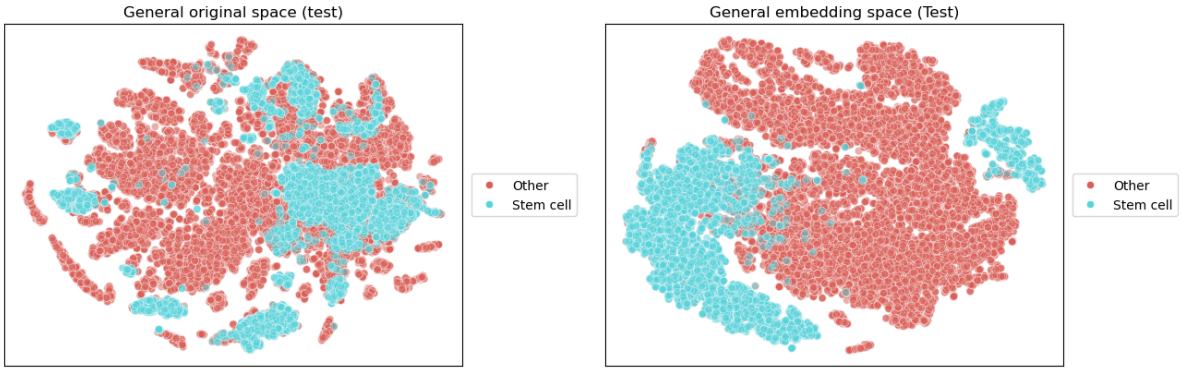


Figure A.16: Original data space (left) versus learned embedding space (right).

A.2.5 Code Availability

All the code used in this work, like the ORIGINS re-implementation, machine learning pipelines, etc is available in: `https://github.com/OverKoder/BCSC_Identification`

The weights of the models will not be uploaded to the repository, but they can be obtained by contacting the owner of the reporsitory.