Large language models for document classification

Trygve Hope Thorland

Department of Information Science and Media
Studies at the University of Bergen, Norway
2024

Year: 2024
Title: Large language models and document classification
Author: Trygve Hope Thorland

Acknowledgements

I would like to thank Andreas L.O for being my supervisor on this thesis project, who was always available no matter when. I would also like to thank the student choir BLAK, for always being there when i needed friends and somethign to do when school was not in the forefront. Lastly i would like to thank my family whose encouragement throughout the years of studying has never stopped. I could never have finished this without them.

Abstract

This thesis is a research endeavor to explore the efficacy of large language models in conjunction with document classification. This thesis started out as a project started together with the ”Byggebot”- project, started by the PolarisMedia-group, after my thesis advisor Professor Andreas Lothe Opdahl was contacted by the project lead to see if he was interested in the idea. The original goal of this project was to see whether or not one could utilize large language models as a means of classifying ”news-worthy” stories, with a focus on municipal building permit documents. The municipality i decided on working in was Tromsø, as one of the local newspapers were working on a similar project, and were originally interested in collaborating. I eventually has issues with bad data, as there was the need to acquire them manually, and the municipal database of Tromsø had less than desirable document markings and classifications. After some trial and error, the decision on shifting the thesis to document classification itself, as this was where most of the time was spent on this project, and it seemed like an interesting angle to research. After rigorous attempts at fine-tuning models and semi-manually classified verification documents, it was unclear whether or not the models were being verified well enough because of lacking data. In the end the decision ended up being to use multiple different state-of-the-art models and compared them against each other to test the agreement between them when being tested against multiple different datasets, trying to classify each one with a similar prefixed-prompt. In the end the resulting scores seem to indicate that certain models tend to agree with each other. The argument is then that the probability of the classification being wrong when multiple models have agreed to this extent, is moderately low. The models are by no means perfect, but it seems to be a trend for certain ones to be more effective than others in some of the domains tested.

Contents

Chapter 1

Introduction

1.1 Introduction

In the current digital age, the volume of data generated by individuals and organizations is growing at an unprecedented rate. This massive amount of information presents us with challenges, but also opportunities. Central to managing this vast sea of data is the process of document classification, which involves categorizing content into predefined groups for easier retrieval and analysis. Effective document classification underpins various applications, from enhancing search engine algorithms to streamlining workflows in various sectors.

Despite advancements in technology, many organizations face the challenging task of managing and utilizing older historical data that was accumulated before modern classification systems were in place, or data that was collected, but not planned for other uses. This data, often poorly labeled or unorganized, represents a significant untapped resource. Traditional methods of data classification often struggle with the sheer amount of data, each dataset often containing hundreds of thousands of data points to classify.

Recent advancements in AI and language processing technology, mainly large language models, offer a new and promising angle at attempting to find a solution to these issues. Large language models(LLM's), such as the most famous chatGPT, or GPT-4o, GPT-4 and GPT-3.5 turbo as the models are called, are capable of understanding and parsing human language on a scale we've never seen before with remarkable accuracy. These models have been trained on billions of data points from all over the internet, making them able to understand complex linguistic patterns and understand context in a nuanced

way.

This thesis is an exploration of how we might use these models in today's technological landscape, and investigate how effective these models have become at completing complex multi-variable classifications tasks when facing different domains, such as news articles, or documents with more esoteric language such as municipal building permit documents, and does language play a role?

1.1.1 Research question

1. **RQ1 -** How can large language models be used in the context of document classification in the domain of building case documents and news-stories?

Chapter 2

Background

In this section i will be going over theories and the overarching technologies used in this thesis. In Chapter 3 i will be going over any new technologies used in each iteration when they were first utilized as they appear.

2.1 Technology

In this section i will be going over the various technologies that i have used in this thesis, a brief description and their use-cases in the technological landscape.

2.1.1 Programming language - Python

*** Python is a high-level programming language known for its simplicity and readability, which makes it an good choice for both beginners and veteran developers. Created by Guido van Rossum in the late 1980s and released in 1991van Rossum, Python emphasizes code readability with its use of a clean syntax. Python's extensive standard library and the vast ecosystem of third-party packages enable developers to perform a wide range of tasks, from web development and data analysis to artificial intelligence and scientific computing. Its versatility and ease of use have contributed to its widespread adoption in various fields.
***

Web scrapers

Selenium is a powerful open-source tool for automating web browsers. It allows developers to programmatically control and interact with web pages, making it ideal for tasks like automated testing, web scraping, and repetitive web

interactions. Selenium supports multiple programming languages, including Python, and can work with all major browsers. BeautifulSoup is a similar package, but works directly from the command line instead of simulating using web drivers

2.1.2 Large language models

Large language models are a variant of artificial intelligence designed to comprehend, analyze and generate human language by leveraging massive amounts of training text data. These models are trained on extensive corpora, to then be fine tuned to enhance their efficacy in specific tasks, increasing their accuracy by leaps and bounds. Contrary to traditional machine learning models that require large amounts of training data on very task specified datasets, large language models, such as the GPT models, that boast billions of parameters, can perform extremely well in a varied selection of language tasks, despite only having been fed a few examples of the task. This process of feeding examples to a large language model is called ”Few shot learning”. This low need for task-specific training data is what sets large language models apart from the old generation of machine learning models. It allows for cheap scalability with extreme efficacy. Brown et al. (2020)

Tokens

Tokens are the fundamental units of text processing in LLM's. A token can be a word, a segment of a word, or a character, depending on the tokenization strategy. The model converts input text into tokens, which are then embedded into vectors that capture semantic meaning. These token embeddings are processed through multiple layers of a transformer architecture, where each token calculates the probability for what token might come next.Vaswani & Polosukhin. (2017)

Prompt engineering

Prompt engineering is a strategy of crafting a prompt in such a way to effectively communicate with large language models. The desire is to create prompts that elicit the wanted response, making the generated output as accurate and relevant as possible. The methods of doing this involves using words, phrasing questions in a particular manner, and iterate upon the prompt based on the feedback you get from the output.

Chapter 3

Research method

3.0.1 Design science research

This section contains a description of the research method used, and will help to better develop an artifact to be used to answer the research question.

Design science

Design science research is an important method of development of information systems that aim to solve a defined problem. The emphasis of this method of development lies in its practical solutions and grounding in scientific method. According to Hevner et al. (2004) the method relies on these seven tenets of development:

Design as an Artifact Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiating.

Problem Relevance The objective of design-science research is to develop technology-based solutions to important and relevant business problems.

Design Evaluation The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.

Research Contributions Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact,design foundations, and/or design methodologies

Research Rigor Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.

Design as a Search Process The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment

Communication of Research Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

3.0.2 Extreme Programming

Extreme programming(XP) is a form of agile software development methodology that emphasises requirement satisfaction from the customer. It puts a large emphasis on satisfaction, flexibility and efficiently. It was developed by Kent Beck in the 90s. XP promotes quick releases in small development cycles, which allows for quick integration of new customer requirements. The practices of XP include test driven development, where each piece of code is tested before moving on to the next, pair programming, where one person always reviews the code the other has written, pair programming, where one programs, while another person reviews the code of the other. Lastly it emphasises collective code ownership, where all members of the development team should be familiar with the project code. XP mainly ensures software quality by collaboration between customer and developer by valuing simplicity, feedback and rapid communication. XP is about being adaptable on a short time frame. Yasvi (2019)

Chapter 4

Development process

The original goal of this thesis project was to create an application that was capable of giving a precise score on to a document whether it was news-worthy or not. As many of the technologies and the field of using large language models in conjunction with building permit documents was rather new, the exact approach was never a known path. As such, the development process was a process of having a requirement goal, reviewing the results, and re-evaluating the approach of the next set of requirements. This resulted in a shift from classifying news-worthiness to document classification.

Søkeresultat byggsaker

Totalt antall resultater: 79

Saksnr	Dato	Gnr/Bnr	Sakstype	Beskrivelse	Avsender/Mottaker	Saksbehandler		Saksdokument	Vedlegg	Mer info
24/9334-1	14.05.2024	57/10	Sak	57/10/0/0 Leirstrandvegen 245 - henvendelse om dialog vedrørende søppelbu	TROMS DØVEFORENING	Guro Rustad Grebstad		Last ned	Vedlegg (3)	Mer

Figure 4.1: Example of a row in the building case database

4.1 Iteration 1

Goals

The goal of the first iteration was to gather and store data. As one of the collaboration partners early on was a group from PolarisMedia, who had a project group working on something similar at the iTromsø newspaper editorial office, the decision to work with the same set of data as them in the Tromsø municipal building case database was made. To do this, we first had to identify which documents were relevant, then extract the meta-data as surrounding each main document and the document itself in the database, and save them to an easily workable format.

1. Extract data
2. Save to a workable format

Tools and technologies

Visual Studio Code The IDE being used for this project. Visual Studio Code was chosen as this is a widely used IDE, at the same time it is an open source program with a multitude of plugins that can be utilized should the need arise. For this project only a CSV visualiser was used for easy reading of data.

Python The programming language used in this thesis. As mentioned in 1.2.1, python is a programming language that is highly versatile, as well as being very suitable for text analysis jobs.

Scrapers There was two web-scraping python-libraries utilized in the development, one was BeautifulSoup, the other was Selenium.

PyMuPDF PyMuPDF is a lightweight binding to muPDF, which is a pdf, and various other format viewer, and allows for easy working of pdf documents in python code PyMuPDF (2024).

iTromsø While not a workable tool, we were given a website from iTromsø that contained all news-stories that were tagged with ”Byggesak” and had used a building-permit document while being research, making them already deemed news-worthy. These are the documents we wanted to collect.

1. BeautifulSoup is a web scraper with a python integration that lets the user extract html and css, but cannot load any javascript.
2. Selenium is another scraper with a python integration that uses the chrome web-driver in order to simulate a user accessing a website, and as such is able to load a complete website.

Development process

When starting this iteration, the first objective was to identify the correct building cases. To do this, a manual process had to be done, identifying which address was the one being discussed in the news stories on the iTromsø page. Each case has a case number, seen in the first column in figure 4.1, called ”Saksnr”. Theese numbers have a structure of xx/xxxxx-xx where the two

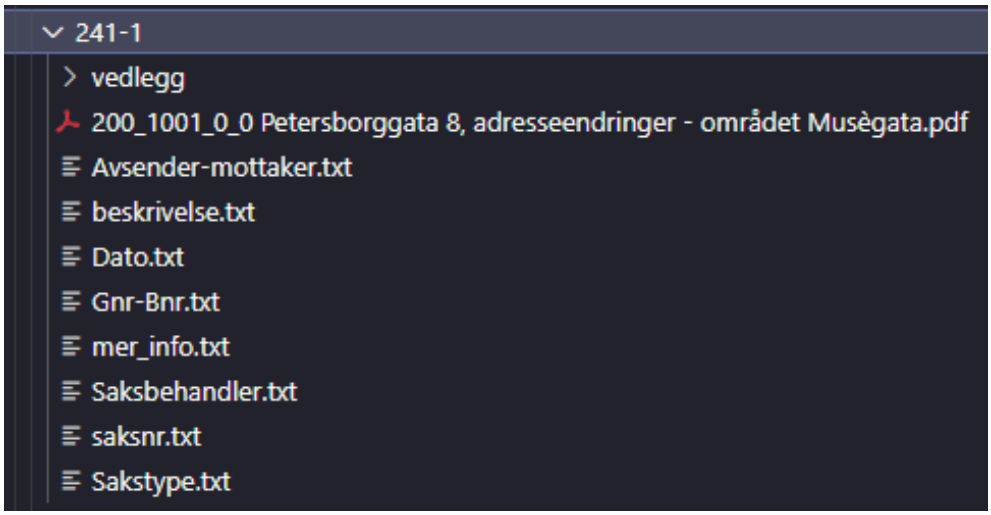

Figure 4.2: Example of file structure after initial scrape

numbers before the slash represent the year the case was started, the following numbers being the main identifier of the case, and the numbers after the dash being the progression number of the document in the case. Here the most relevant to the current goal is the year and main identifier-numbers, as these are searchable. At first the web-scraping of the documents was attempted using BeautifulSoup, but as the website‘s database tables are loaded dynamically using javascript, it proved fruitless to use this approach. The next option discovered was Selenium, which simulates a user accessing the page in real time, simulating navigation and clicks, using a chrome driver. This allowed the program to load the javascript and automatically select each item on the specified rows of data, and save them to various formats. The first gathering of data was just the raw data, so images and .pdf documents were downloaded as-is, while the html strings in the columns were saved in .txt files and all attachments saved in a nested folder. See fig 4.2. The PyMuPDF library was then used to convert the main document in each case into text. After all data was in very a very basic .txt format, the goal of getting the separate files into a single, easy and workable format remained. At first the format that was desired was csv, as this was a familiar format. This proved to be more hassle than what was worth to convert. The large amounts of text in some documents, and the amount of commas in them, proved to be uncooperative. As such, JSON was chosen instead, as large amounts of text would not interfere with the format of the file.

validation

To validate this process, once the files were saved, a manual review of the file structure was done, which pointed out certain errors, where a small amount of cases had not had their main document scraped, so these were manually downloaded later and added to the file structure. This was possible as it was a small amount. After this was fixes, no further errors in the data was found.

Results

The results of this iteration is that the data was scraped with selenium, saved to basic formats, and converted to a easily workable JSON-format. Going forward, it became clear that more meta-data was needed for the project, and so the need for document classifications arose.

4.2 Iteration 2

Goals

For the second iteration, the goal was to obtain a means of document classifying, fine tuning a GPT-model was decided upon. Goals:

1. Fine tune GPT-model for classifying
2. Classify the documents with the fine tuned model

Tools and technologies

OpenAI-API OpenAI offers a paid-access to their API solution. This API allows for a myriad of functionalities tied to their large language models such as fine tuning, which is of most interest to the current goal. Most features on the API are also available on their chat.GPT web-platform, but the API allows for programmatic access, which is useful for working large amounts of data.

Models Used

- Babbage and Davinci Babbage and Davinci are, according to OpenAI‘s documentation GPT-3 base models, they are not trained to heed instructions, but can generate code and text. The current versions listed in OpenAI‘s documentation, suffixed by ”-002” are an update from a previous version now deprecated, now being based on GPT-3.

- GPT3.5-Turbo GPT-3.5 Turbo Is probably the most used version out of all of the publicly available models, as it has been the standard for the free version of the web-solution version of chatGPT for a number of months prior to the writing of this thesis. GPT-3.5 Turbo is a version of GPT-3.5 which again is a version of a series of fine tuned versions of the GPT-3 base model Davinci.

N-gram analysis An N-gram analyis is a text analysis method of looking for re-occuring collections of words or phrases in a text. The N stands for the number of words in the collection. A 1-gram would just be how often a single word occurs, while a 2-gram is how often every combination of 2 subsequent words occur.

Development process

The development phase for iteration 2 started by researching which models would be best for the task at hand. Multiple factors went into this consideration, as the API is not free, there were not enough available funds to prompt of train GPT-4. Another facet was the difference in the objects returned when using the api. The models babbage and davinci, based on the base GPT-3 model use the ”completions”-object endpoint, gpt3.5 can be used with the ”completions”-object. It is the first model to use the ”chat-completions”-object, and all proceeding models use this format. The difference between these two objects can be summarised by saying a ”completions”-object is only a text value being sent as a prompt, but a ”chat-completions”-object includes potential extra instruction as to how the model should act in addition to the prompt itself, in addition to add a response from a user, to simulate a pre-existing conversation. The choice was made to use babbage, davinci and gpt3.5-turbo, as they are all affordable enough, as well as babbage and davinci being reccommended for jobs where fine tuning is intended for them. Gpt3.5-turbo is a model already generally good at most use cases. The way you fine tune these models is to prepare a jsonl file, where each line is a Key value pair of ”prompt”:”input text”, and a key value pair of ”completion”:”expected output” as an example of an expected input and expected output for the use case of the trained model. The reccomended amount of training examples for davinci and babbage is not stated in the documentation, but for gpt3.5-turbo it is stated that at about 50 examples clear improvement is seen. OpenAI (2024a) For gpt3.5-turbo a ”role” needs to be specified as well for both input and output.
To prepare the fine-tuning data, two operations were done. The first was to create the training jsonl file, where the input would be the ”beskrivelse” field

```
Top 7 2-grams:
('anmodning', 'om'): 32
('søknad', 'om'): 32
('inga', 'sparboes'): 23
('sparboes', 'veg'): 23
('fjellheisen', 'øvre'): 21
('øvre', 'stasjon'): 21
('kommuneplanens', 'arealdel'): 20
```

Figure 4.3: List of 2-grams found through analysis

from the Tromsø database, see example fig: 4.1
The ”completion” field was obtained by doing an n-gram search of the same ”beskrivelse”, translated to description, field to see which words were the most common among all the 300 descriptions. This process was believed to reveal the most common ”types” of documents. If any of the words in the 2-grams were found in a given file, that 2-gram was added as the ”completion” in the jsonl-training file. Stopwords were excluded, as well as some address names that were very common. As the n-gram lists were generated 1-gram through 5-grams were generated, however 2-grams seemed to give the most insight. In figure 4.3 the list is displayed, however this is before the addresses were added to the stopwords, and does not contain full context. Later on a manual review of the 2-grams were done, removing any words not deemed to add value to the classification of the document such as miscellaneous words or less common adresses. This technically makes some of the results no longer 2-gram, this does not matter as the intention of the analysis was just to find the most common, and most likely to be most classifying words.
What remained was to upload the jsonl file, which was split into a test and a train file at a 80/20 split. The OpenAI-API finetuning interface allows for a test file upload to monitor test metrics automatically while training.
A 3.5-turbo model was also trained alongside the davinci and babbage models. The results of the fine-tuned models will be discussed in the Results portion of this iteration

Example request gpt-3.5-turbo python

```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful
    {"role": "user", "content": "Hello!"}
  ]
)

print(completion.choices[0].message)
```

Figure 4.4: Chat completions object example from OpenAI documentation

Validation

The validation of this iteration was at the time challenging, as the expertise of what constitutes as a valid ”document category” for building case documents is not something that is widely known, or available information. This caused the classifications to mainly be educated guesses based on some discussions with the supervisor of this thesis. The model verification is also rather inconclusive, as the built-in test validator was used, but did not calculate the accuracy of the models.

Results

As these models were ran on the main documents in the chosen dataset, the results were less than desirable.During a discussion with the supervisor of the thesis, it was discovered that it might have been an issue that the models were trained on the descriptions, but ran on the full document, and so the results of this iteration was mostly un-usable, and so the next goal was to train models using the whole document, instead of just the description. The idea to use the pseudo-2-gram classifications as a velidation stayed. The prefix used in the prompt before the document itself was:

```
Under kommer teksten fra et byggesaks-dokument.
    Jeg vill at du skal klassifisere dokumentet
    med opp til tre av disse merkelappene:
S knad.
Tilsyn.
Anmodning.
Uttalelse.
Plan.
Dokumentasjon.
Merknad.
Detaljregulering.
Tilbakemelding.
Varsel.
Kommuneplanens arealdel.
Oppf ring.
P legg.
Detaljreguleringsplan.
Arealdel for perioden.
Igangsettingstillatelse.
Anmodning om m te.
Uttalelse til varsel.
Planstart.
Om du ikke syns noen av disse passer,
gi dokumentet merkelappen 'no classification.'
```

The results of the differing models were stark contrasts of one another. The Davinci and Babbage fine tuned models had very poor results, sometimes classifying everything as ”no classification”, or nothing at all, as opposed to GPT3.5-turbos fine tuned, which seemed like it only wanted to give too much information, never classifying anything, but beginning a long explanation of its reasoning, but never finishing, as the token limit was already set very low in an attempt to limit this result.

4.3 Iteration 3

Goals

The goals of iteration 3 were mainly an alternate approach to the goals of iteration 2, but with more insight into the task at hand.

1. Fine tune a model using the whole document as input, and the categories

gleaned from the description of the document as output.

2. Classify test-documents and

3. Calculate an accuracy score for the model

Tools and technology

Iteration 2 Most of the technologies from iteraion 2 was also used in iteration 3, with the exeption of n-gram analysis.

Building term dictionary Two dictionaries containing specialised building case vocabulary was used. This was needed as this is a field where some terms are rather esoteric, and an explanation was needed for those words.

Development process

When starting the development on iteration 3, the main goal was to shift the fine-tuning data from the descriptions to the main documents. This was a rather simple task, being that the only thing needed to be changed was the document variable in the prompting code, and to retrieve the text value from the converted pdf-file from iteration 1. Subsequently the idea to include descriptions to some of the words that would be used for classifying was introduced. The notion had been introduced earlier in a meeting with the project lead of the ”byggebot”-project, but not much else had been done with the idea. Sources of documentation on this vocabulary is sparse, but two sources of building-document related vocabulary was found, namely byggesaksordboka (2024) and samform (2024) which individually did not have all the wanted words, but did complete one another. It should be noted that not all the words being used as classifications were from these dictionaries. Words such as ”Varsen”-warning, or ”Søknad”-application are common enough, that a general purpose dictionary was good enough to provide the extra context. As such, the new prefix became the following:

```
Her kommer en liste med etiketter (eller
      labels    ) som skal brukes i
   klassifiseringen.
Hver etikett beskrives pa en egen linje.
Selve etiketten kommer    f rst pa hver linje,
    avsluttet med et punktum.
Deretter f lger en forklaring pa etiketten med
    tegnene *to hashtags* f r og etter.
```

S knad. En s knad til kommunen om tillatelse til a oppf re bygg og anlegg samt til a utf re visse andre typer av tiltak.

Tilsyn. Tilsyn er en viktig del av bygge- og eiendomsbransjen, og det refererer til overvakingen eller inspeksjonen som utf res for a sikre at arbeidet pa et byggprosjekt f lger regler, forskrifter og standarder.

Anmodning. A oppfordre eller be om (p a en h flig m ate).

Uttalelse. muntlig eller skriftlig ytring, meddelelse som gir uttrykk for en mening (og rettes til offentligheten, til en myndighet e.l.)

Plan. Plan er en ordnet fortegnelse over gj remal, rutiner eller lignende som skal utf res; en gjennomtenkt, metodisk fremgangsmate; et grunnlag for den praktiske gjennomf ringen av noe.

Dokumentasjon. Dokumentasjon kan enkelt forstas som presentasjon av informasjon, der dokumentasjonen viser pa en eller annen m ate en viss mengde av informasjon.

Merknad. synonym for kommentar Detaljregulering. –

Detaljregulering er en planleggingsprosess som beskriver hvordan et omrade eller eiendom skal utvikles og brukes. Dette omfatter blant annet bestemmelser for byggeh yde, antall boenheter og parkeringsplasser, samt krav til gr ntarealer og fellesanlegg.

Tilbakemelding. Svar pa et tidligere saksdokument.

Varsel. Et varsel er en pastand om ett eller flere kritikkverdige forhold.

Kommuneplanens arealdel. Kartlegging av eksisterende arealbruk av kommunen samt fremtidige behov, handlingsdelen med konkrete tiltak som skal iverksettes i

perioden planen omfatter.
Oppf ring. det a f re eller skrive opp noe (f . eks. navn p a liste eller lignin– ende) ; noe som er oppf rt pa liste, ark eller lignende.
Palegg. synonym for ordre eller p abud.
Detaljreguleringsplan. Detaljregulering er en planleggingsprosess som beskriver hvordan et omrade eller eiendom skal utvikles og brukes. Dette omfatter blant annet bestemmelser for byggeh yde, antall boenheter og parkeringsplasser, samt krav til gr ntarealer og fellesanlegg.
Arealdel for perioden. Arealet det er snakk om i saksperioden.
Igangsettingstillatelse. En igangsettingstillatelse er en tillatelse som gis av kommunen til en utbygger eller eiendomsutvikler f r bygging av et nytt byggverk eller en st rre ombygging kan starte.
Anmodning om m te. Oppfordring til a holde et m te.
Uttalelse til varsel. – At noen gir til utrykk for at det burde bli varslet om noe
Planstart. – A begynne arbeidet pa en lagt plan.
Om du ikke syns noen av disse passer, gi dokumentet merkelappen no classification.

At this point, the decision to drop the fine-tuned gpt3.5-turbo was made, as it was a major drain source of funds compared to Babbage and Davinci, and had produced nothing that could resemble any validation-able results in the last iteration.

validation

1. View the accuracy to see if its good enough for use.

Results

The results of iteration 3 was done by calculating the accuracy with some python code that looked at all the completions returned from the prompts, and the verification data from earlier iterations, if any completions from a prompt existed in the validation data for that document, give a +1 score. At the end divide by the amount of documents to get the accuracy.

```
with open(file_path, 'r', encoding='utf-8',
   errors='ignore') as file:
    completion_values_list = []
    replies_values_list = []
    for line in file:
        data = json.loads(line)

        # Split the 'completion' and 'replies'
           into lists of values, converting
           them to lowercase
        completion_values = [x.strip().lower()
           for x in data['completion'].replace
           ('.', ',').split(',') if x.strip()
           != '']
        replies_values = [x.strip().lower() for
            x in data['replies'].replace('.',
           ',').split(',') if x.strip() != '']
        completion_values_list.append(
           completion_values)
        replies_values_list.append(
           replies_values)

        for reply in replies_values:
            if reply in completion_values:
                total_score += 1

        total_replies += len(replies_values)


accuracy = total_score / total_replies
```

```
    return accuracy, completion_values_list,
        replies_values_list
```

The best result of this iteration was an accuracy of 0.2217741935483871, which came from the fine-tuned babbage model, but alas, is not a good enough score for the models to be performing well enough for use.

4.4 Iteration 4

Goals

The goal of iteration 4 was to create a confusion matrix in order to check for any patterns in the labeling of documents, weather they be correct or incorrect. The intent is to see weather there are any suspected malformations in the data.

1. Create a confusion matrix for pattern-search of labeling

Tools and technology

Matplotlib Matplotlib is a comprehensive and fine-grain library intended for statistical visualization, allowing for heavy customization of plots. Matplotlib (2024)

Seaborn Like Matplotlib, Seaborn is a library used for statistical visualization, but is built on top of Matplotlib, and is developed to be heavily integrated with pandas data-structures. The way it iterates upon the field of visualizations of Matplotlib is that it takes care of a lot of the drawing of the plots internally, and lets you focus more on what the plots mean, rather than having to differentiate between them manually. Seaborn (2024)

Numpy Numpy is a python library for fundamental computing, which provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Development process

The development for iteration 4 is rather simple, as the technology used, as well as iteration 3‘s saved format was easily converted into a numpy array, and visualized with Matplotlib and Seaborn. The resulting confusion matrix can be seen in fig 4.5 Labels used for this confusion matrix are as seen in fig 4.7 It should be noted the implementation of this code also took into account all

the labels that the babbage model had generated, but were not part of the set of labels that were instructed to be given to the documents. The total dataset for this confusion matrix used classifications from 66 documents, but each document could have 1 to 3 classifications given to it by the model. One such given classification was for example *"Eiendom (gnr",* or *"Ad, sak. Deres ref. TILSYN-22/01778-3,"* which are completley nonsensical. The davinci-fine-tuned model was dropped from use from this point onwards, as it was performing worse that the babbage model in every classification accuracy calculation, the exact scores were not saved, but were lower than the 0.22 that the previous iteration managed.

validation

1. Look for any patterns

Results

The results of this iteration came down to the dispersion and accuracy of labels. From the confusion matrix in fig 4.5 it is clear that the fine-tuned Babbage model classifies a lot of junk, which is other labels that are not exactly what was defined, such as the examples given in the development section above. The classification of "plan" seems to be doing a marginally better job than most other labels, with 14 out of 28 actual "plan"-labeled documents being labeled correctly. "No classification" is also one that has been classified correctly a fair amount, however, it is only classified correctly 19 out of 62 times. As a lot of documents had been errously been labeled labels in the junk category, the aim for the next iteration was to sift the junk from the classification dataset, in order to see if the things it classified as specified in the prompt were correct. The conclusion that the amount of labels might have an impact on the results, so they should also be reduced.

4.5 Iteration 5

Goals

The goals for iteration 5 is to:

1. Reduce the amount of labels used, as the complexity of a lot of the labels might make them more difficult to classify.

2. Only implement documents where the model has actually classified a label that was specified

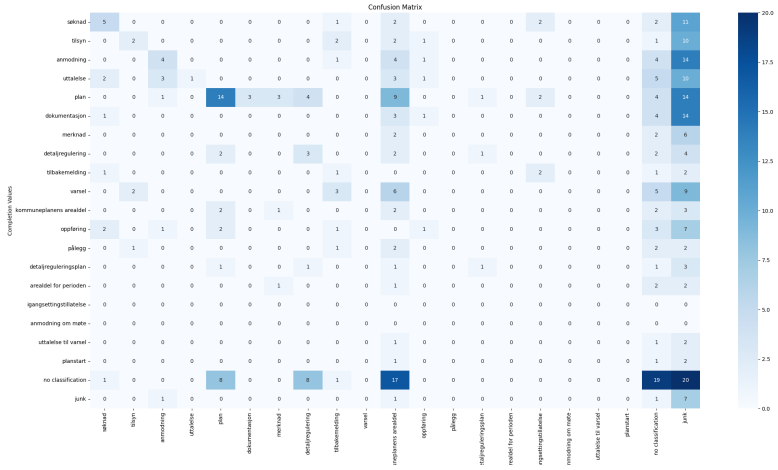

Figure 4.5: Confusion Matrix for babbage model, 1st iteration

```
labels = [
    "søknad", "tilsyn", "anmodning", "uttalelse", "plan", "dokumentasjon",
    "merknad", "detaljregulering", "tilbakemelding", "varsel", "kommuneplanens arealdel",
    "oppføring", "pålegg", "detaljreguleringsplan", "arealdel for perioden",
    "igangsettingstillatelse", "anmodning om møte", "uttalelse til varsel",
    "planstart", "no classification", "junk"
]
```

Figure 4.6: Confusion Matrix labels used in 1st iteration of confusion matrix

3. Look for patterns in the results

Tools and technology

Pandas Pandas is a widely used, powerful, open-source data analysis and manipulation library for python. It allows for various data-structures to be used, but in this section, only the dataframe structure. Pandas (2024)

Development process

For iteration 5 a new prompt-prefix was introduced, with the aim of reducing the complexity of the labels, removing the extended explanations, as well as the amount of labels. The verification data was reduced to only documents which had any of the prompt-prefix classification words explicitly appear in the ”description” section of their saved json file from iteration 1. The new prompt-prefix is as follows:

> Her kommer en liste med etiketter (eller ”labels”) som skal brukes i klassifiseringen.

```
    Hver etikett beskrives p  en egen linje.
   Selve etikettenkommer f rst p  hver linje,
    avsluttet med et punktum. Verdsett starten
   av teksten mest.
S knad.
Plan.
Igangsettingstillatelse.
Varsel.
Dokumentasjon.
Anmodning.
Om du ikke syns noen av disse passer, gi
   dokumentet merkelappen 'no classification.'
Jeg vil at du skal returnere et JSON-objekt med
    en "label" key som inneholder den
   kategorien fra listen over du tror
   dokumentet burde klassifiseres som, og en "
   forklaring" key hvor du gir en kort
   forklaring p  hvorfor du ga den labelen.
Her kommer dokumentet jeg  nsker  at du skal
   klassifisere:
```

For this iteration, a new chunk of code was used, where Pandas was utilized instead of numpy, as it has the data structures that are intended to be used with Seaborn. See the appendix for this code, fig A.2. This time around the prompting strategy was to ask for a json-object as the returned format. The model used for this iteration was just the plain babbage-002 base model, which in hindsight, was definitely a mistake, as the base models are not trained to follow instructions. The intention of this was to see if changing variables in the process would yield any positive outcomes.

validation

Validation is the same as before, to look at the confusion matrix and glean any patterns that might help identify the issue of a low accuracy.

Results

The results for iteration 5 were unable to be used in a confusion matrix, as the responses were so vastly different to what was asked, that a "label" was not possible to extracted reliably from nearly any return prompts. It should be noted that in the period of this iteration these prompt were being conducted,

reports of GPT models spouting garbage seemingly out of nowhere were being reported OpenAI (2024c)

4.6 Iteration 6

Goals

Iteration 6 is another attempt at trying to identify any patterns in a created confusion matrix like the two previous iterations.

1. Create a confusion matrix on the new prompt-prefix that is functional
2. Identify any patterns or issues in the data displayed in the confusion matrix
3. Calculate accuracy

Tools and technology

Sklearn The sklearn library, also known as scikit-learn, is a powerful and versatile Python library designed for machine learning and data science.One key subsection of scikit-learn is the .metrics module, which offers a comprehensive suite of functions for evaluating the performance of machine learning models.

Development process

During the development of this iteration a discovery was made in the documentation of gpt3.5-turbo, in that when calling the chat.completions object for a response, it is possible to specify a response format, json being one of them. OpenAI (2024b)

```
    response = client.chat.completions.create(
  model="gpt-3.5-turbo-0125",
  response_format={ "type": "json_object" },
  messages=[
    {"role": "system", "content": "Message"},
  ]
)
```

This allowed for a more consistent string return from each prompt. With this mode enabled, the chat completions looked like this:

{\n "label": "S knad",\n "forklaring": "
Dokumentet ser ut til v re en s knad om
ferdigattest i henhold til plan- og
bygningsloven. Det inneholder informasjon om
tiltaket, gjenv rende arbeider, oppdatert
dokumentasjon og ansvarlige parter.",\n "
sender": "Til kommunen"\n}

To specify, this response was returned as a string, and not as an actual json object, so some reformatting into acual json was needed, but this was easily done.

validation

To validate we look at the confusion matrix.

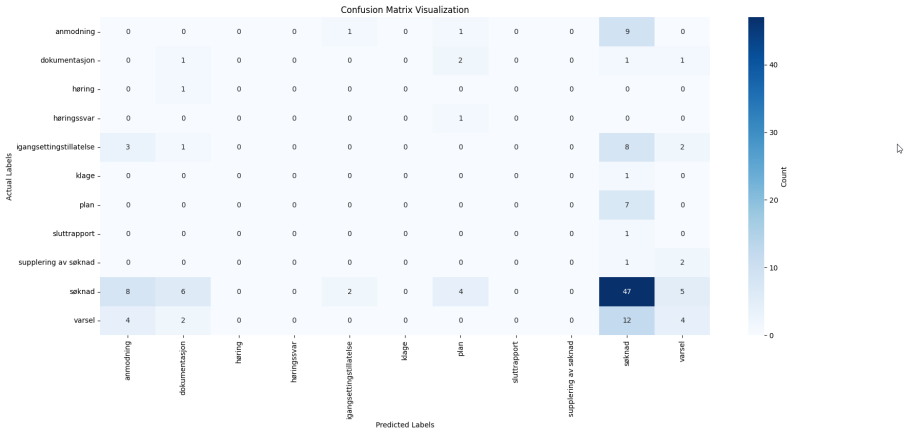

Figure 4.7: 2nd iteration of confusion matrix

Results

The results of this iteration show by way of the confusion matrix that there is a gross over-representation of "søknad"-labels being present in the validation set, but also of the labels being predicted, the "anmodning", "igansettingstillatelse", "plan" and "varsel" labels in particular have gotten a larger amount of "søknad"-labels assigned to them. While this result overall has an accuracy of 41.27 percent, this is the macro accuracy, which is calculated by averaging all classifications, no matter the category, while the micro accuracy, where we take each category, calculate the accuracy of each individual label, and average

out the scores, we get a slightly worse, but not drastically lower 37.68 percent. What we want, however is to see how the classification works overall, with no class being weighted more than another, and so using the macro-F1 accuracy calculation method is what should give the best insight according to this article Leung (2024). The macro-F1 accuracy is no more than 8.53 percent. For this an easy import of the sklearn.metrics library was used, see fig A.3 in the appendix.

A reflection was made at the end of this iteration, which was that the validation data felt too sparse, and unreliable. This reasoning came from the sense that a lot of the building documents contained, for a lack of a better phrasing, a lot of ”legal speak”. And so without proper labeling for the original documents, or without the expert knowledge needed to manually do so at a later time while being sure that they were correct. The decision to move away from concrete validation metrics was made.

4.7 Iteration 7

Goals

Going forward it was clear the results of the previous analysis method were producing rather inconsistent results, and as th issue was thought to be with the validation data, the method of verifying using a validation dataset was scrapped all together. Instead of using only one large language model, the plan became to use multiple ones, and gauge their agreement when it came to labeling documents. The research argument was that if there arose a pattern of agreeing models that classified documents similarly independent of one another, then the results they were producing must be closer to the accurate truth. To do this more models were required. Amazon web services(AWS), offers a service called ”Bedrock”, which houses a multitude of what they call ”foundation models”, which in essence are large language models of 3rd party organizations that are integrated into the ”Bedrock”-service in a way that allows for a streamlined user interface both in the web-playground, and programmatically in their API.

1. Create a Bedrock account and get familiar with the API

2. Choose which models to use

3. Create a pipeline for calculating similarities between the models

Tools and technology

Blackrock Amazon web services(AWS), offers a service called ”Bedrock”, which houses a multitude of what they call ”foundation models”, which in essence are large language models of 3rd party organizations that are integrated into the Bedrock-service in a way that allows for a streamlined user interface both in the web-playground, and programmatically in their API.

Jurassic-2 Ultra Jurassic-2 Ultra is a large language model provided by the organization ai21. They provide three versions of the Jurassic-2 model, Ultra, which is the version used here, which is the most powerful and best suited for the most complex language tasks. The other two are called mid and light. Mid is a cost-effective balanced version of ultra, while light is designed with speed in mind. In their documentation, they describe the jurassic-2 models capability as highly versatile general purpose text-generators capable of composing human-like text and solving complex tasks like question answering, text classification and other objectives. Ai21 (2024)

Llama3-70b-instruct Llama-3 is the most recent in a line of large language models developed by Meta, an open source model that Meta claims excels at tasks like language nuances, contextual understanding, and complex tasks like translation and dialogue generation. Meta (2024) The instruct part of the model name used is to indicate it was trained with following prompt instructions in mind.

Mistral-large Mistral is developed by Mistal AI. The Mistral-large model is among what Mistral calls their ”optimized” suite of models. They also provide a selection of open-source models. AI (2024)

Cohere-Command The cohere-command model is provided by the cohere organization, which advertises itself by providing business-oriented uses for their models. On their web-playground cohere offers classification as a specific GUI for users. Cohere (2024)

Cohen’s Kappa The method of calculating the similarity of the models was by using *Cohens Kappa* which is a method of performing an inter-rater agreement analysis. The way this analysis was performed was by looking at every pair of models compared against one another, as this method is designed for only comparing two raters against each other. A method for calculating the agreement between all models called *Fleiss Kappa* exists, but the interesting

metric is which models are performing well, so just *Cohens Kappa* will be used for this analysis. The library used for the calculations are once again sklearn.metrics. When reading the scores of *Coehns Kappa* analysis, L. (2012) determines that a score above 0.60 seems to indicate a moderate agreement where between raters. They provide a range of percentage of the data that is viable of 35-63 percent, but what determines where in this range the data lands is unclear.

Development process

When developing this iteration 7 of the project, the first thing that was necessary was to set up a Bedrock account on the AWS cloud service. This was rather straight forward. The way Bedrock words is that they have an almost standardized model calling framework, with small alterations between the parameters needed for model calls, and some differences between which items to retrieve from returned objects. See fig: 4.8 for example for how the framework differs between models.
Once familiarization with the bedrock environment was complete, it was time to start prompting. Bedrock allowed for the same functionality of specifying which data format the returned text should be, so JSON was chosen for all of the Bedrock models as well. The prompt-prefix was changed in an attempt to obtain more consistently standardized returns, as sometimes the models would not complete the JSON-object format in the return text, and this new prompt including an example of exactly how the json-format looks like seemed to work well.

```
Jeg vil at du skal returnere et JSON-
    objekt med en "label" key som
    inneholder den kategorien fra listen
     over du tror dokumentet burde
    klassifiseres som, og en "forklaring
    " key hvor du gir en KORT forklaring
     p  maks 2 setninger om hvorfor du
    ga den labelen. Verdsett starten av
    teksten mest.
Her kommer en liste med etiketter (
    eller "labels") som skal brukes i
    klassifiseringen. Hver etikett
    beskrives p  en egen linje. Selve
    etiketten kommer f rst p  hver
    linje, avsluttet med et punktum.
```

```
    V r oppmerksom p  at dokumenter
    kan snakke om dokumenter som
    omhandler andre labels , for eksempel
     at et varsel om en s knad ikke er
    en s knad , men et varsel. En mail
    som diskuterer en anmodning on
    holde et m te for  diskutere en
    klage er heller ikke en klage.
S knad.
Plan.
Igangsettingstillatelse.
Varsel.
Dokumentasjon.
Anmodning.
Om du ikke syns noen av disse passer,
    gi dokumentet merkelappen 'no
    classification.'
Pass p  at du fullf rer JSON formatet
    , eksempel: *curly bracket*"key1": "
    value1", "key2": "value2"*curly
    bracket*
Her kommer dokumentet jeg  nsker  at du
     skal klassifisere:
```

The settings for the differing models also got tuned to be as similar as possible, as some models did not have all settings some other models had, such as the *top k* variable in the mistral model call in fig 4.8 is missing from the Llama model call. During the development of the extraction of the JSON elements of the returned model texts, a regular expression was used, which later was discovered to be faulty

```
json_pattern = re.compile(r'\{(?:[^{}]|(?R))*\}')
```

There were some issues with this expression which led to some data loss for this iteration, and led to worse scores in the results than necessary.

validation

- Look at the Kappa score calculated by comparing the models

Results

The resulting scores were as follows:

```python
def classify_document_mistral(doc, prompt):
    prompt_complete = prompt + doc

    mistral_body = json.dumps({
            "prompt": prompt_complete,
            "max_tokens": 200,
            "temperature": 0.0,
            "top_p": 0.1,
            "top_k": 10
    })

    mrt_mistral = boto3.client(service_name='bedrock-runtime')
    modelId = 'mistral.mistral-large-2402-v1:0'
    accept = 'application/json'
    contentType = 'application/json'

    response_mistral = mrt_mistral.invoke_model(
        body=mistral_body,
        modelId=modelId,
        accept=accept,
        contentType=contentType
    )

    response_body = json.loads(response_mistral.get('body').read())

    return response_body
```

```python
def classify_document_llama(doc, prompt):
    prompt_complete = prompt + doc

    max_gen_len = 200
    temperature = 0.0
    top_p = 0.1


    llama_body = json.dumps({
            "prompt": prompt_complete,
            "max_gen_len": max_gen_len,
            "temperature": temperature,
            "top_p": top_p
    })

    mrt_llama = boto3.client(service_name='bedrock-runtime')
    modelId = 'meta.llama3-70b-instruct-v1:0'
    accept = 'application/json'
    contentType = 'application/json'

    response_mistral = mrt_llama.invoke_model(
        body=llama_body,
        modelId=modelId,
        accept=accept,
        contentType=contentType
    )

    response_body = json.loads(response_mistral.get('body').read())

    return response_body
```

Figure 4.8: Comparison between model call code of Mistral and Llama models

```
Kappa for jurassic and llama: 0.16985138004246292
Kappa for jurassic and mistral: 0.21470759504200976
Kappa for jurassic and cohere: 0.11556603773584906
Kappa for jurassic and GPT: 0.1981202446665672
Kappa for llama and mistral: 0.3157894736842105
Kappa for llama and cohere: 0.3246753246753247
Kappa for llama and GPT: 0.44654088050314467
Kappa for mistral and cohere: 0.1806926551781829
Kappa for mistral and GPT: 0.4625052720371151
Kappa for cohere and GPT: 0.23608805485384332
```

Here we can see some rather low scores for most pairs of models, a cause of this was thought to be the domain of the documents being analyzed, as municipal building case documents were probably quite a small amount of documents that the models were trained on. The next iteration will try to compare different domains of documents, to see if its the obscure domain that is the source of the low scores. As for the pipeline, a series of steps in the form of python scripts were created for a faster processing of the files. These steps handle:

1. Script 1: The prompting and returned results themselves.

2. Script 2: Data cleaning, as some of the models had a tendency to include unwanted symbols such as differing amounts of backslashes or breaklines in the raw strings returned.

3. Script 3: Extraction of the JSON object, this is where the regex is used.

4. Script 4: Creates a dictionary for each dataset
5. Script 5: The Coehn's Kappa calculation is done here.

4.8 Iteration 8

Goals

The main goal of iteration 8 is to incorporate a different dataset on news stories into the analysis to see if domain is the issue on documentation.

1. Find a dataset on news stories
2. Create a prompt-prefix for the new documents
3. Calculate the kappa score for the new dataset and compare with the analysis from the previous iteration

Tools and technology

The NCC-corpus The NCC-corpus, or the "Norwegian Colossal Corpus" is a collection of documents meant to be machine learning training data for training on the Norwegian language. The dataset was curated by the National Library of Norway. This dataset encompasses many domains such as almost 10 million news-documents, museum records and official department documents. It is meant to be a training set that represents the Norwegian language generally. Thankfully these domains are separated, and lets allows for selection of the desired domain category to be used.library of Norway (2024)

Development process

The search for a dataset on Norwegian new stories took a little bit of time. The first dataset that was considered was a dataset from NTNU called the "SmartMedia Adressa News Dataset" which was originally used for research in conjunction with reccomender systems. The issue here is that to get access to the full dataset they required an application, and by the time it got aprooved, the next dataset, NCC, had already been used in the analysis, so it was dropped. On that note, the datased that ended up being used is the NCC dataset, more specifically the *newspaper ocr* subsection of the dataset. This subsection contains a total of 9.8 million documents averaging 199 words per document. The news documents in the set ranges from the 1940's up to contemporary time. The NCC dataset was easy to work with, as the whole dataset was saved

to a single JSON file, where all that needed to be done was filter our the JSON objects that had the *newspaper ocr* doc type. The format of the NCC JSON objects are:

```
{
"id": "
    firdafolkeblad_null_null_19680801_63_56_1_MODSMD_ARTICLE8
    ",
"doc_type": "newspaper_ocr",
"publish_year": 1968,
"lang_fasttext": "no",
"lang_fasttext_conf": "0.749",
"text": "Sample text"
}
```

The only element that is necessary is the "text" value. 126 random JSON objects were chosen from the whole dataset, and saved to a separate JSONL file to be used for prompting.
Now that there is a new domain, the need for a new prompt-prefix was needed. An attempt to keep the structure similar to the prompt prefix for the building case documents. Adding short descriptions to each label was re-introduced after testing, as it seemed to improve results somewhat. Here are the two updated prompt-prefixes for this iteration.
Building case prompt prefix:

```
Jeg vil at du skal returnere et JSON-
    objekt med en "label" key som
    inneholder den kategorien fra listen
     over du tror dokumentet burde
    klassifiseres som, denne dele SKAL
    komme f rst i svaret ditt. En "
    forklaring" key hvor du gir en KORT
    forklaring p  maks 1-2 setninger om
     hvorfor du ga den labelen. Verdsett
     starten av teksten mest.
Her kommer en liste med etiketter (
    eller "labels") som skal brukes i
    klassifiseringen. Hver etikett
    beskrives p  en egen linje. Selve
    etiketten kommer f rst p  hver
    linje, avsluttet med et punktum.
    Deretter f lger en forklaring p
```

etiketten. V r oppmerksom p at dokumenter kan snakke om dokumenter som omhandler andre labels, for eksempel at et varsel om en s knad ikke er en s knad, men et varsel. En mail som diskuterer en anmodning on holde et m te for diskutere en klage er heller ikke en klage.
S knad. – S knad om tillatelse om noe bygg–relatert
Plan. – En utarbeidet plan om et prosjekt
Igangsettingstillatelse. – Tillatelse om sette i gang med et arbeid
Varsel. – Varsel om noe som strider med lover eller rettningslinjer
Dokumentasjon. – Relevant info til en sak
Anmodning. – R dning om noe.
Om du ikke syns noen av disse passer, gi dokumentet merkelappen 'no classification.' Det er godt mulig at noen ikke passer merkelappene nevnt over.
Pass p at du fullf rer JSON formatet, eksempel: ∗curly bracket∗"key1": "value1", "key2": "value2"∗curly bracket∗
Her kommer dokumentet jeg nsker at du skal klassifisere:

News-story prompt-prefix:

Jeg vil at du skal returnere et JSON–objekt med en "label" key som inneholder den kategorien fra listen under du tror dokumentet burde klassifiseres som, denne dele SKAL komme f rst i svaret ditt. En "forklaring" key hvor du gir en KORT

```
    forklaring p  maks 1-2 setninger om
     hvorfor du ga den labelen. Verdsett
     starten av teksten mest.
Her kommer en liste med etiketter (
    eller "labels") som skal brukes i
    klassifiseringen. Hver etikett
    beskrives p  en egen linje. Selve
    etiketten kommer f rst p  hver
    linje, avsluttet med et punktum,
    etterfulgt av en kort utdyping av
    hva labelen betyr.
Konflikt. - Krangler mellom parter,
    uenigheter eller lignende
 konomi . - Budsjett, bevilgelser, bruk
     av penger
Kjendis-relatert. - Ting som ang r
    kjendte personer
Politikk. - politiske saker, kan v re
    p  kommunalt eller nasjonalt niv
Kriminalitet. - politisaker, rettsaker,
     osv...
Kultur. - Ting som musikk, kunst,
    fritidsaktiviteter eller lignende
Historie. - Ting omhandlende historikk
Om du ikke syns noen av disse passer,
    gi dokumentet merkelappen 'no
    classification.' Det er godt mulig
    at noen ikke passer merkelappene
    nevnt over.
Pass p  at du fullf rer JSON formatet
    , eksempel: *curly bracket*"key1": "
    value1", "key2": "value2"*curly
    bracket*
Her kommer dokumentet jeg  nsker  at du
     skal klassifisere:
```

This amount was just to keep the dataset as large as the other one, but it is believed to not really impact any scores, and also to keep the prompting time down, as just running these two datasets through all the prompters took roughly 50 minutes. An attempt to improve the regex used to retrieve the JSON pattern was made, but ultimately unsuccessful for this iteration.

validation

- Look at the Kappa score calculated by comparing the models

Results

The results for the NCC documents were as follows:

```
Kappa for jurassic and llama: 0.11820431851156442
Kappa for jurassic and mistral: 0.07383548067393465
Kappa for jurassic and cohere: 0.021756793355601572
Kappa for jurassic and GPT: 0.0350453675185216 36
Kappa for llama and mistral: 0.335696754840469
Kappa for llama and cohere: 0.11005058700009573
Kappa for llama and GPT: 0.15859766277128562
Kappa for mistral and cohere: 0.1401696790852084
Kappa for mistral and GPT: 0.3856869232108654
Kappa for cohere and GPT: 0.16958489081251232
```

This set of results does seem to indicate a higher success, however there still seems to be too low scores for any actually useful classification with the current method. For the next iteration the plan was to increase the domains, and include English versions of the Norwegian domains, building case documents and news stories.

4.9 Iteration 9

Goals

The goal for this last iteration was to implement the English versions of the domains, as well as check if there are any emerging patterns in the results of the four domains.

1. Implement English building case documents
2. Implement English news-story documents
3. Identify any patterns in the calculated data

Tools and technology

English building cases The English building case documents that were used are from the New York City Departement of City Planning's database of building permits. of City Planning (2024)

Claude v2.1 The Claude series of models are a set of large language models developed by the organization Anthropic in collaboration with Amazon, and made available on Amazon's Bedrock platform. This set of models is advertised as being well suited to perform tasks like being a customer chatbot, extracting knowledge from legal, insurance og buisness documents, as well as being suited for coding assistance for developers. Amazon (2024)

English news-stories The Selection of English news-stories that was opted for are from a Kaggle dataset taken from *thenews.com* and includes news-stories from 2015 to 2017 ASADMAHMOOD (2024)

chat.lmsys.org Chat.lmsys.org is a research website created by the organization LMSYS in cooperation with researchers from UC Berkley Skylabs. It aims to "rank" the different models available on the market. It does this by letting participants write a prompt, where then two models will answer that prompt, and the user will pick which one answered better. LMSYS (2024)

Development process

Looking for matching english datasets for the building cases in implementation for this iteration proved challenging, as the structure and scope and transparency of access of documents included in the Norwegian municipal databases are quite unique. The conclusion that was reached was that the aim is not so much the specific type of document, but rather the type of language used, and such the dataset from NYCDoCP proved sufficient. For the news-story dataset it was quite the opposite, there are a myriad of datasets containing news-stories openly available on the internet. The kaggle dataset was chosen for no particular reason other than it contained multiple different categories of news-stories.
Next was to come up with English prompt-prefixes for the new documents. The strategy was to keep the prefixes the exact same, just translated to English
English building case prompt-prefix:

```
I want you to return a JSON object with
    a "label" key that contains the
   category from the list of which you
   think the document should be
   classified as, this part MUST come
   first in your answer. An "
   explanation" key where you give a
  SHORT explanation of a maximum of
```

```
    1–2 sentences about why you gave
    that label. Value the beginning of
    the text the most.
Here comes a list of tags (or "labels")
     to be used in the classification.
    Each label is described on a
    separate line. The label itself
    comes first on each line, ending
    with a period. Then follows an
    explanation of the label with the
    characters. Please note that
    documents may refer to documents
    that deal with other labels, for
    example that a notice of an
    application is not an application,
    but a notice. An email discussing a
    request to hold a meeting to discuss
     a complaint is also not a complaint
    .
Application. – Application for
    permission for something
    construction–related
Plan. – A prepared plan for a project
Commissioning permit. – Permission to
    start work
Notice. – Notification of something
    that contravenes laws or guidelines
Documentation. – Relevant information
    for a case
Request. – Advice about something.
If you don't think any of these fit,
    give the document the label 'no
    classification.' It is quite
    possible that some do not fit the
    labels mentioned above.
Make sure you complete the JSON format,
     example: *curly bracket*"key1": "
    value1", "key2": "value2"*curly
    bracket*
```

```
Here is the document I want you to
    classify:
```

English news-story prompt-prefix:

```
I want you to return a JSON object with
    a "label" key that contains the
    category from the list below you
    think the document should be
    classified as, this part MUST come
    first in your answer. An "
    explanation" key where you give a
    SHORT explanation of a maximum of
    1-2 sentences about why you gave
    that label. Value the beginning of
    the text the most.
Here comes a list of tags (or "labels")
    to be used in the classification.
    Each label is described on a
    separate line. The label itself
    comes first on each line, ending
    with a period, followed by a brief
    explanation of what the label means.
Conflict. - Arguments between parties,
    disagreements or the like
Economy. - Budget, grants, use of money
Celebrity-related. - Things that
    concern famous people
Policy. - political matters, may be at
    municipal or national level
Crime. - police cases, court cases, etc
    ...
Culture. - Things like music, art,
    leisure activities or the like
History. - Things about history
If you don't think any of these fit,
    give the document the label 'no
    classification.' It is quite
    possible that some do not fit the
    labels mentioned above.
Make sure you complete the JSON format,
```

```
        example: *curly bracket*"key1": "
        value1", "key2": "value2"*curly
        bracket*
    Here is the document I want you to
        classify:
```

The regular expression issue was fixed in this iteration, making all the JSON data returned from the models available in the analysis, vastly improving the results.
New regex:

```
pattern = re.compile(r"'label'\s*:\s*'([^']+)'")
```

Additionally, another model was added to the analysis, amazons Claude v2.1 model. This model was added in an attempt to incorporate more models that were highly regarded as some of the best on the current market, according to the website chat.lmsys.org. An attempt to obtain API access to google's Gemini model was attempted as well, but this is not available in Norway at the current moment. The Vicuna model was explored as an option as well, but at the time seemed to only be available in some sort of beta access program. The implementation of the Claude model went smoothly thanks to the standardized format in Bedrock. An issue in the previous iterations when calculating the Kappa score, was that a substantial amount of some models, but mostly llama managed the JSON-format very poorly, so the data could not be extracted from the text promps. This lead to many empty lines in the JSONL file that was used to calculate the Kappa score. To help with this metric, a choice was made to ignore the instances where any one of the two data points being compared were empty. When calculating the results, the scores were plotted into a seaborn heat map for better readability. A 6th script was added to the pipeline that took care of this plotting.

validation

1. Look for patterns in the heat map

Results

The resulting heatmaps are figures 4.9, 4.10, 4.11 and 4.12. Each of the heatmaps represents the results of each of the 4 data domains. Across all the plots it seems like the GPT, Llama and Mistral models have some sort of connection. The scores themselves are not always great enough to be reliable, but they are definitely agreeing on a larger amount of data than the majority of the other models.

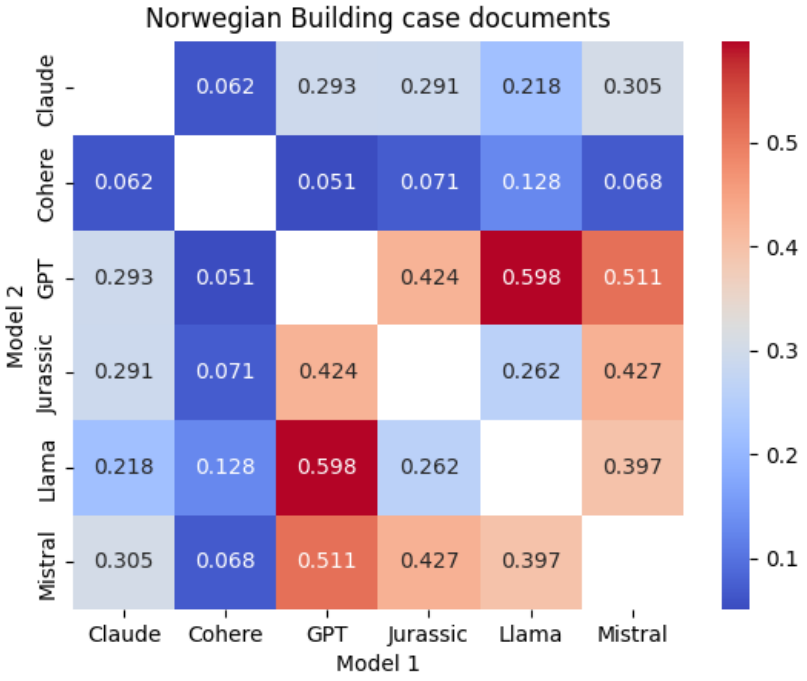

Figure 4.9: Norwegian building case documents

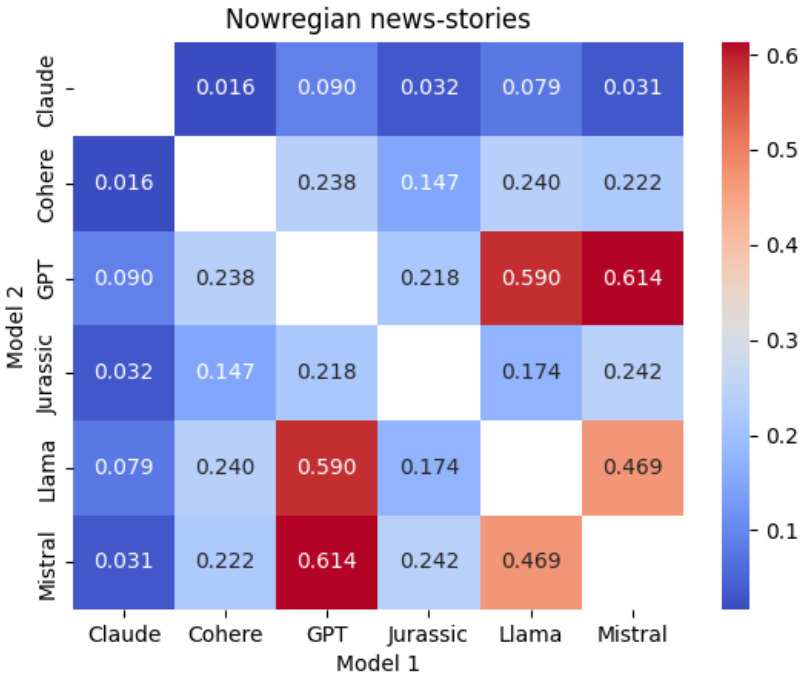

Figure 4.10: Norwegian news-story documents

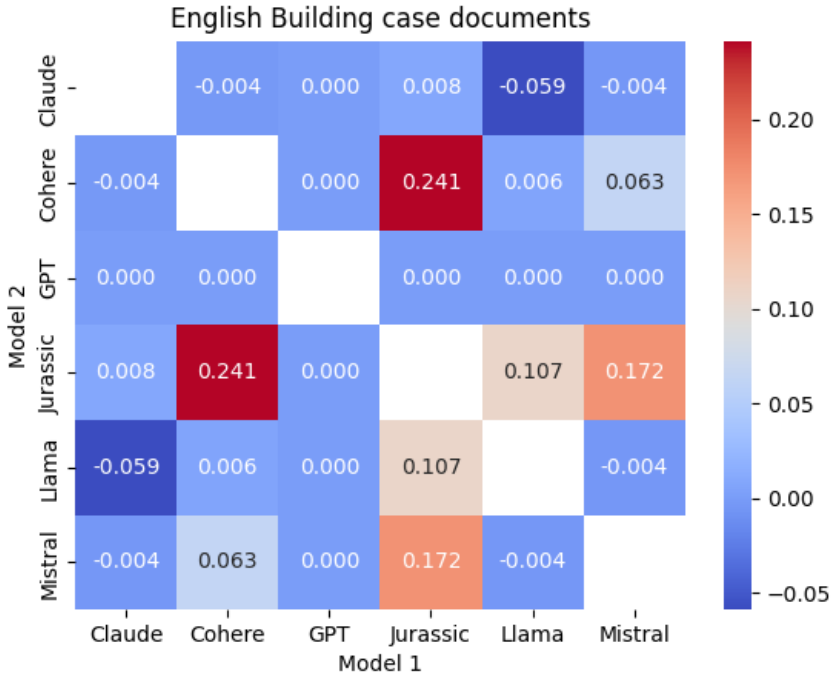

Figure 4.11: English building case documents

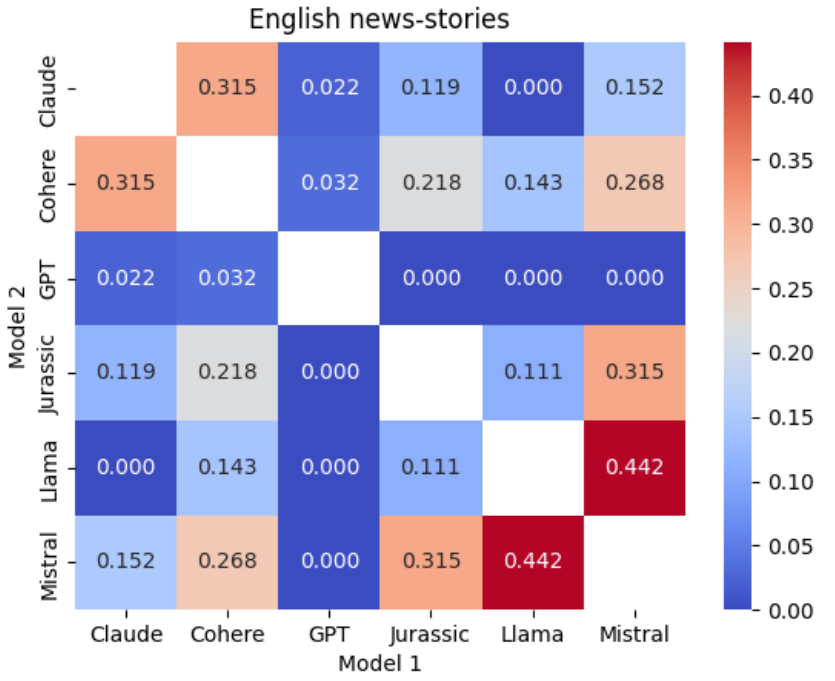

Figure 4.12: English news-story documents

Chapter 5

Discussion

5.1 Research question 1

RQ1- How can large language models be used in the context of document classification in the domain of building case documents and news-stories?
Throughout the development of this thesis the application of the large language models GPT3.5-turbo, Mistral-Large and Llama3-70b-command in the domain of document classification has shown some promising potential. The patterns in the resulting heatmaps show that these three models shows potential.
The journey to this conclusion however, was not straight forward. The initial attempts at doing classification with verifiable data was a method that eventually was dropped as the original data barely had any classification data at all, separate from an arbitrary non-standard description of each document. Thus a neccesary shift towards evaluating using a method that did not need verification data, but verified based on the metric of similar output between models, an inter-rater agreement analysis. This discussion will delve a little deeper into the efficacy, the challenges and perhaps future implications of using LLM's as document classificators in the domains of Building case documents mainly as these documents tend to be less labeled compared to the news-story documents that usually have fairly good labeling.

Inter-Rater Agreement Analysis The research pivoted from a direct evaluation of classification accuracy to an analysis of inter-rater agreement due to the subpar quality of the labeled data. This change in methodology was crucial in deriving meaningful insights. The moderate scores observed across GPT, Mistral, and LLaMA indicate that these models exhibit a fair degree of consistency in their classifications. The patterns in their agreement suggest that while the models may not always converge on the same classification, their

outputs are often aligned in a way that reflects similar underlying patterns in the data.

Challenges faced

1. **Quality of the labeled data:** The original dataset had poor labeling leading to significant issues not only when trying to verify the output of the models in earlier iterations, but also probably caused the fine tuning of the Davinci and Babbage models to be sup-optimal as well. High quality data is imperative when training and validating models effectivley.

2. **Model Agreement:** While GPT, Mistral and Llama showed promise in the inter-rater agreement analysis, the best scores obtained still are far from ideal, showing som clear divergenses in their classifications. This still varied output underlines the need for further refinement of the strategies used.

3. **Context specific Adaptations:** Building case permit documents are a quite esoteric domain, and is probably not a large amount of the myriad of data that the models are trained of fine tuned on. Further fine-tuning with good data that is labeled well enough will hopefully improve accuracy and consistency.

4. **Datasets:** Furthermore when looking at the results for the English versions of building case documents and news stories, they seem to do worse than the Norwegian counterparts in nearly every regard. This is probably due to a lack of implementation of a sound prompting strategy as the strategy was to just copy the structure of the Norweguan versions.

Chapter 6

Conclusions

In this thesis the use of large language models have been proven to have some merit in the domain of building case permit documents and news-stories. The initial phase of the project was hindeerd by poor data quality, initiating a shift in verification methods to an inter-rater analysis. This adjustment allowed for a more meaningful insight into the performance of the models.
The use of six models were used: GPT3.5-turbo, Mistral-Large, Llama3-70-command, Cohere-Command, Jurassic-2 Ultra and Claude v2.1. An assesement of their consistency in classification tasks were performed. Most of the models produced quite poor results.GPT, Mistral and Llama produced analysis scores high enough to suggest that they might have potential for multi-variable classification, even though their outputs were not entirely consistent.
Future endeavors should focus on several key areas. Using better labeled data in order to fine tune models for a better contextual domain understanding.
The study highlights the promise of LLMs in document classification, yet also underscores the necessity for further refinement and development. By addressing the challenges identified, we can enhance the reliability and accuracy of these models, ultimately streamlining the organization and accessibility of critical information in building case documents and news stories. Future efforts should aim to build on the findings of this research, focusing on areas that can maximize the value and efficiency of LLMs in document classification tasks.

Appendix A

Code screenshots

```python
import json
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define all possible tags including a "junk" tag for unexpected predictions
labels = [
    "søknad", "tilsyn", "anmodning", "uttalelse", "plan", "dokumentasjon",
    "merknad", "detaljregulering", "tilbakemelding", "varsel", "kommuneplanens arealdel",
    "oppføring", "pålegg", "detaljreguleringsplan", "arealdel for perioden",
    "igangsettingstillatelse", "anmodning om møte", "uttalelse til varsel",
    "planstart", "no classification", "junk"
]

# Initialize counters for both completions and replies
label_count = {label: 0 for label in labels}

# Path to your input file
input_file_path = 'fine-tuning-redundant\conf_matrix_data.jsonl'  # Replace with your actual file path

combined_dict = {}
with open(input_file_path, 'r', encoding='utf-8') as file:
    for index, line in enumerate(file):
        data = json.loads(line)
        completion_values = [x.strip().lower() if x.strip().lower() in labels else 'junk'
                             for x in data['completion'].replace('.', ',').split(',') if x.strip() != '']
        replies_values = [x.strip().lower() if x.strip().lower() in labels else 'junk'
                          for x in data['replies'].replace('.', ',').split(',') if x.strip() != '']
        combined_dict[index] = [completion_values, replies_values]

        # Update label counts for both completions and replies
        for label in completion_values + replies_values:
            label_count[label] += 1

# Initialize the confusion matrix
confusion_matrix = np.zeros((len(labels), len(labels)), dtype=int)
label_index = {label: i for i, label in enumerate(labels)}

# Populate the confusion matrix
for completion_values, replies_values in combined_dict.values():
    for completion_item in completion_values:
        for reply_item in replies_values:
            i = label_index[completion_item]
            j = label_index[reply_item]
            confusion_matrix[i][j] += 1

# Plotting the confusion matrix
plt.figure(figsize=(12, 10))
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Replies Values')
plt.ylabel('Completion Values')
plt.title('Confusion Matrix')
plt.show()

# Print the count of each label
for label, count in label_count.items():
    print(f"{label}: {count}")
```

Figure A.1: Code from 1st iteration of confusion matrix

```python
import json
from collections import defaultdict
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

file_path = 'acc_compare_18_4.jsonl'

def create_confusion_matrix(file_path):
    class_set = set()
    confusion_matrix = defaultdict(lambda: defaultdict(int))

    with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
        for line in file:
            data = json.loads(line)

            # Handle classification as a list
            classifications = data['classification'] if isinstance(data['classification'], list) else [data['classification']]
            predicted_label = data['predicted_label'].strip().lower()

            # Normalize all classifications to lowercase
            classifications = [c.strip().lower() for c in classifications]

            # Add classes to the set
            class_set.update(classifications)
            class_set.add(predicted_label)

            # Increment the corresponding count in the matrix for each classification
            for classification in classifications:
                confusion_matrix[classification][predicted_label] += 1

    # Convert the confusion matrix to a pandas DataFrame for better visualization
    matrix_df = pd.DataFrame(confusion_matrix).fillna(0).reindex(index=sorted(class_set), columns=sorted(class_set), fill_value=0)
    return matrix_df

def visualize_confusion_matrix(matrix_df):
    plt.figure(figsize=(10, 8))
    sns.heatmap(matrix_df, annot=True, fmt='g', cmap='Blues', cbar_kws={'label': 'Count'})
    plt.xlabel('Predicted Labels')
    plt.ylabel('Actual Labels')
    plt.title('Confusion Matrix Visualization')
    plt.show()

def calculate_accuracy(file_path):
    total_correct = 0
    total_labels = 0

    with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
        for line in file:
            data = json.loads(line)

            classifications = data['classification'] if isinstance(data['classification'], list) else [data['classification']]
            predicted_label = data['predicted_label'].strip().lower()

            # Normalize and check match
            classifications = [c.strip().lower() for c in classifications]
            total_labels += 1

            # Check if predicted label matches any classification
            if predicted_label in classifications:
                total_correct += 1

    accuracy = total_correct / total_labels if total_labels > 0 else 0
    return accuracy

# Create the confusion matrix
conf_matrix = create_confusion_matrix(file_path)

# Display and visualize the confusion matrix
print(conf_matrix)
visualize_confusion_matrix(conf_matrix)

# Calculate and print accuracy
accuracy = calculate_accuracy(file_path)
print(f'Accuracy of classification vs predicted label: {accuracy:.2%}')
```

Figure A.2: Code from 2nd iteration of confusion matrix

```python
import pandas as pd
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import json

# Function to load data from a JSONL file
def load_jsonl(file_path):
    data = []
    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            data.append(json.loads(line.strip()))
    return data

# Function to calculate metrics
def calculate_metrics(data):
    # Transform the data into a DataFrame
    df = pd.DataFrame(data)

    # Flatten the 'classification' column
    df = df.explode('classification')

    # Calculate micro-accuracy
    y_true_micro = df['classification']
    y_pred_micro = df['predicted_label']
    micro_accuracy = accuracy_score(y_true_micro, y_pred_micro)

    # Calculate macro-accuracy
    macro_accuracy = accuracy_score(df.groupby(df.index)['classification'].first(), df.groupby(df.index)['predicted_label'].first())

    # Calculate precision, recall, and F1 for micro and macro
    micro_precision, micro_recall, micro_f1, _ = precision_recall_fscore_support(y_true_micro, y_pred_micro, average='micro')
    macro_precision, macro_recall, macro_f1, _ = precision_recall_fscore_support(y_true_micro, y_pred_micro, average='macro')

    metrics = {
        "micro_accuracy": micro_accuracy,
        "macro_accuracy": macro_accuracy,
        "micro_precision": micro_precision,
        "micro_recall": micro_recall,
        "micro_f1": micro_f1,
        "macro_precision": macro_precision,
        "macro_recall": macro_recall,
        "macro_f1": macro_f1
    }

    return metrics

# Load data from JSONL file
file_path = 'general_workspace/acc_compare_30_5.jsonl'
data = load_jsonl(file_path)

# Calculate metrics
metrics = calculate_metrics(data)

# Print metrics
print(json.dumps(metrics, indent=4, ensure_ascii=False))
```

Figure A.3: Code from micro and macro accuracy calculation

Bibliography

AI, M. (2024). Mistral technology. URL: https://mistral.ai/technology/#models last accessed 31 May 2024.

Ai21 (2024). Jurassic-2 models. URL: https://docs.ai21.com/docs/jurassic-2-models last accessed 31 May 2024.

Amazon (2024). Anthropic's claude in amazon bedrock. URL: https://aws.amazon.com/bedrock/claude/ last accessed 2 June 2024.

ASADMAHMOOD (2024). News story dataset. URL: https://www.kaggle.com/datasets/asad1m9a9h6mood/news-articles?resource=download last accessed 2 June 2024.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language models are few-shot learners. arXiv:2005.14165.

byggesaksordboka (2024). Wordlist for building-case language. URL: https://www.byggordboka.no/artikkel/utskrift-alle/ last accessed 20 May 2024.

of City Planning, N. D. (2024). Housing database. URL: https://www.nyc.gov/site/planning/data-maps/open-data/dwn-housing-database.page#housingdevelopmentproject last accessed 2 June 2024.

Cohere (2024). We're building the future of language ai. URL: https://cohere.com/about last accessed 31 May 2024.

Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram, & Sudha (2004). Design science in information systems research. *Management Information Systems Quarterly*, *28*, 75–.

L., M. M. (2012). Interrater reliability: the kappa statistic. *Biochem Med (Zagreb)*, .

Leung, K. (2024). Micro, macro weighted averages of f1 score, clearly explained. URL: https://www.kdnuggets.com/2023/01/micro-macro-weighted-averages-f1-score-clearly-explained.html last accessed 30 May 2024.

LMSYS (2024). chat.lmsys website. URL: https://chat.lmsys.org last accessed 2 June 2024.

Matplotlib (2024). Matplotlib documentation. URL: https://matplotlib.org/stable/index.html last accessed 29 May 2024.

Meta (2024). Introducing meta llama 3: The most capable openly available llm to date. URL: https://ai.meta.com/blog/meta-llama-3/ last accessed 31 May 2024.

library of Norway, N. (2024). Datasets. URL: https://ai.nb.no/datasets/ last accessed 31 May 2024.

OpenAI (2024a). Fine tuning. URL: https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset last accessed 25 May 2024.

OpenAI (2024b). Openai $_{j}son - modedocumentation. URL :$ last accessed 29 May 2024.

OpenAI (2024c). Unexpected responses from chatgpt. https://status.openai.com/incidents/ssg8fh7sfyz3 last accessed 29 May 2024.

Pandas (2024). Pandas documentation. https://pandas.pydata.org/docs/ last accessed 29 May 2024.

PyMuPDF (2024). Pymupdf turorial. https://pymupdf.readthedocs.io/en/latest/tutorial.html last accessed 23 May 2024.

van Rossum, G. (). A brief timeline of python. https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html last accessed 21 May 2024.

samform (2024). Wordlist for building-case language. https://www.samform.no/produkt/ordbok last accessed 20 May 2024.

Seaborn (2024). Seaborn documentation. https://seaborn.pydata.org/tutorial/introduction.html last accessed 29 May 2024.

Vaswani, N. S. N. P. J. U. L. J. A. N. G. u. K., Ashish, & Polosukhin., I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, .

Yasvi, M. (2019). Review on extreme programming-xp.