|  |
| --- |
| **LLMs for Transaction-Merchant Matching** |

**Rishabh Kaushick**

College of Engineering

Northeastern University

Toronto, ON

*kaushick.r@northeastern.edu*

**Abstract**

In this report, I explored the performance of traditional supervised learning algorithms and large language models to classify messy financial transaction descriptors to their corresponding merchants. The supervised learning algorithm explored was XG Boost, and the LLMs used were LLaMA 3.2 & Gemma 3 both running locally on Ollama. Various prompt-engineering techniques like zero-shot prompting, 10-shot prompting method have been explored. Additionally, techniques such as Retrieval Augmented Generation (RAG) & Facebook AI Similarity Seach (FAISS) was implemented to add relevant examples to the prompt. Finally, meta-search engine ‘SearXNG’ was used to see if adding merchant information to rare merchants helped improve the performance. The results show that combining local LLMs with RAG-based prompting and external web augmentation significantly improves classification accuracy and generalization compared to the traditional machine learning approach.

**1 Problem Statement**

There are millions of credit card transactions that take place every hour, and billions of transactions every year [4]. Each time we swipe our card a messy *merchant descriptor* gets allocated for the corresponding transaction. Large financial organizations like MasterCard or Visa have a difficulty correctly matching these transaction merchant descriptors to merchant names. Consider the examples in the following table:

Table 1:

|  |  |  |
| --- | --- | --- |
| Messy Merchant Descriptors | Merchant Names | Label |
| AMZN Mktp CA\*UC33G8423 | Amazon Marketplace Canada | Match |
| NETFLX#999-USA | NetFlex Gym USA | Mismatch |

Based on the current process (as shown in the Figure 1 below), sometimes the transaction merchant descriptors are mapped to the correct merchant names like in the first Amazon example. This is known as a ‘Match’. However, for the second transaction, which belongs to Netflix USA, has been incorrectly matched to ‘NetFlex Gym USA’, therefore being classified as a ‘Mismatch’.

Apart from traditional machine learning approaches, this project seeks to explore whether Large Language Models (LLMs), with their emergent abilities, can solve this classification problem better.

**A black rectangular sign with white text

AI-generated content may be incorrect.**Figure 1: Flow chart of the current process.

Sometimes the very large rule-based algorithm could correctly guess the merchant name from the merchant descriptor - like the Amazon example above. However, sometimes it could make mistakes such as the second example.

Therefore, given a pair of merchant descriptor and merchant name, can we classify whether the current process correctly identified the merchant? (MATCH condition) Or is it possible that it guessed the merchant wrongly? (MISMATCH)

Apart from traditional machine learning approaches, this project seeks to explore whether LLMs, with their emergent abilities, can solve this classification problem better.

**2 Datasets**

Since the dataset for this project is confidential, other ways to generate a synthetic dataset, like using LLMs, have been explored. The objective of this project is to use LLMs to classify whether the descriptors match the merchants. If we use the same LLMs to generate the data and perform classification, it could potentially have a lot of bias. To solve this, creative approaches had to be followed to create synthetic data which was like real-world data:

1. The LLMs which generate the data will be different from the LLMs which perform the classification task.
2. Industry standard, cloud-based LLMs are used to generate the data. Smaller open source, open weights local models are used to perform the classification. Such LLMs are used to perform classification as no financial sensitive data will be needed to be sent to the servers of large organizations hosting their LLMs on the cloud.
3. The data generation & classification LLMs are as follows:
   1. Data Generation LLMs: OpenAI ChatGPT-4o, ChatGPT-4.1, Gemini 1.5 Pro
   2. Data Classification LLMs: Google Gemma 3, Meta Llama 3.2

The figure shows a flowchart of the data generation pipeline for ‘Matched’ transactions:

A diagram of a generation flow

AI-generated content may be incorrect.  
Figure 2: Data generation flow for matched transactions.

A screenshot of a computer program

AI-generated content may be incorrect.  
Figure 3: Prompt used for generating transaction descriptors for ‘Match’ transactions.

To generate data for ‘Mismatched’ transactions, a different approach has been followed as LLMs were not able to generate satisfactory incorrect data.

A diagram of a diagram

AI-generated content may be incorrect.  
Figure 4: Data generation flow for ‘Mismatch’ transactions using Python.

First OpenAI ChatGPT-4o has been used to generate over 300 unique merchant names. Next, for each merchant 5 merchant descriptors are generated using the OpenAI and Gemini APIs.

Table 2: The basic feature of both datasets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Data Set Characteristics | Attribute Characteristics | Associated Tasks | Number of Instances | Number of Attributes |
| LLM Generated Transactions | Multivariate | Real | Classification | 5350 | 3 |

**1.1 Data characteristics**

Once the dataset was generated, it was divided into the following sets:

1. Training Set: 4280 rows, 80%
2. Validation Set: 535 rows, 10%
3. Testing Set: 535 rows, 10%

The histograms of the data distribution of the labels are shown in Figure 3. We can see that we have the similar number of records for ‘Match’ & ‘Mismatch’ records.

A graph of a distribution of labels

AI-generated content may be incorrect.  
Figure 5: The class frequency of the labels.

Digging deeper, the following graph shows the distribution of how each of the data was generated.

A graph of a number of different colored bars

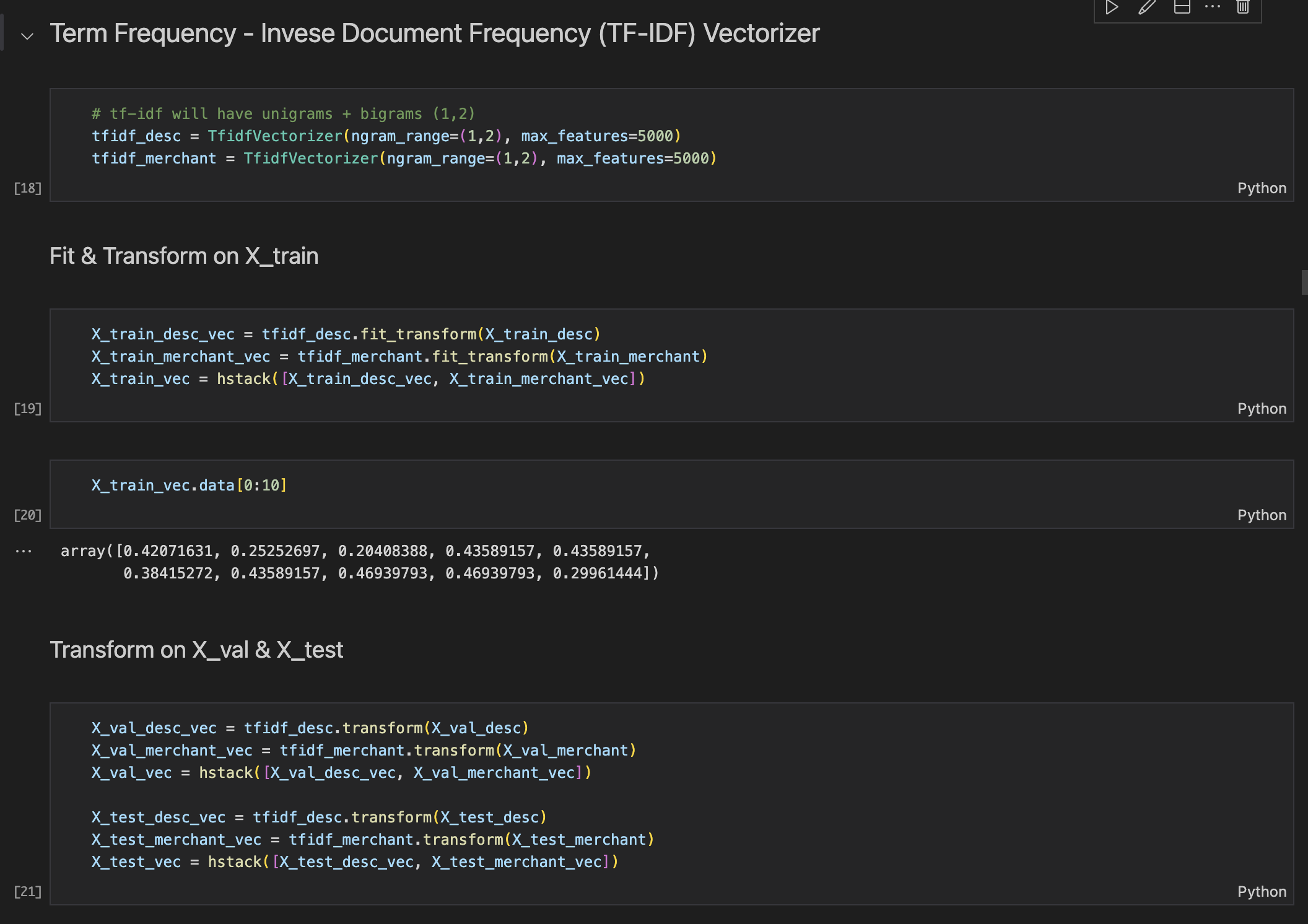
AI-generated content may be incorrect.  
Figure 6: Distribution of how each data point was generated.

**3 Supervised Learning: XG Boost Approach**

I chose to begin this project with a strong baseline traditional machine learning model. This way once we try different LLMs, we can benchmark them with the baseline to understand if it realty is better or perhaps worse. Extreme Gradient Boost model (XG Boost) was selected as it is an ensemble learning method known as ‘boosting’ which uses multiple weaker decision trees to collectively form a strong model. This approach is not very simple yet not too complex and provides a good starting point as a baseline. If simpler model is required, we can go back to decision tree or if a more complex model is required, we can experiment with LLMs.

**3.1 Term Frequency – Inverse Document Frequency**

As most machine learning models, XG Boost cannot understand text data or perform operations on strings. Therefore, the first step before the model is to use an algorithm to convert string into numerical data like vectors. First, we fit and transform the data in the training set and following this we just transform the validation and test set based on the TF-IDF model.

  
Figure 7: TF-IDF

**3.2 XG Boost Model #1**

A baseline XG Boost model was initially developed with the following parameters as shown in the figure below:

A screenshot of a computer

AI-generated content may be incorrect.  
Figure 8: XG Boost model #1

However, this model was overfitting on the training dataset with 86% accuracy but failed to generalize on validation set with 65% accuracy and test set with 66% accuracy. Therefore, hyper-parameter tuning approach was explored to improve the XG Boost baseline.

**3.2 Hyperparameter Tuned XG Boost Model**

The figures below show the different hyper-parameters which were used to re-train the XG Boost model and the final set of parameters which worked the best.

A screenshot of a computer program

AI-generated content may be incorrect.Figure 9: XG Boost hyper-parameters

A screenshot of a computer

AI-generated content may be incorrect.  
Figure 10: XG Boost Model #2

**3.4 Evaluating XG Boost Model**

The figure below shows how the loss gradually in the training and validation sets gradually reduce. After around 200 boosting rounds, we still see the training loss reducing drastically, but the validation loss seems to plateau.

A graph of a line

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**4 Pretrained LLM Approach**

**4.1 Zero-Shot Prompting**

As described in the

A screenshot of a computer

AI-generated content may be incorrect.

The results are as follows:

A screenshot of a computer

AI-generated content may be incorrect.  
Figure: Gemma 3 results on training set.

A screenshot of a computer

AI-generated content may be incorrect.  
Figure: Gemma 3 results on validation set.

A screenshot of a computer

AI-generated content may be incorrect.  
Figure: Gemma 3 results on test set.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**4.2 Few-Shot Prompting: 20-Shot**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Since the performance of the Llama3.2 LLM was sub-optimal even on the 10-shot prompting exercise, it was not evaluated on the validation and test sets.

**5 Pretrained LLM Approach with RAG & FAISS**

The above 10-shot prompting technique could be potentially better than zero-shot prompting in many cases. However, the examples that we provide to the prompt are very important. Sometimes these examples are irrelevant to the current record being classified. This could confuse the model or lead to poor performance. To tackle this problem, I have decided to update each prompt by using the most relevant examples based on the record being classified.

Here is how we can do this using Retrieval Augmented Generation style:

1. store the labeled examples from the training set in a vector store
2. save the vectors in a FAISS index
3. retrieve the K-most similar examples (based on the provided merchant descriptor & merchant)
4. update the prompt with only relevant examples
5. get the classification result from gemma/ llama

**5.1 Embeddings Using Sentence Transformer**

A screen shot of a computer program

AI-generated content may be incorrect.

**5.2 Few-Shot Prompting: Relevant 5-Shot Prompts**

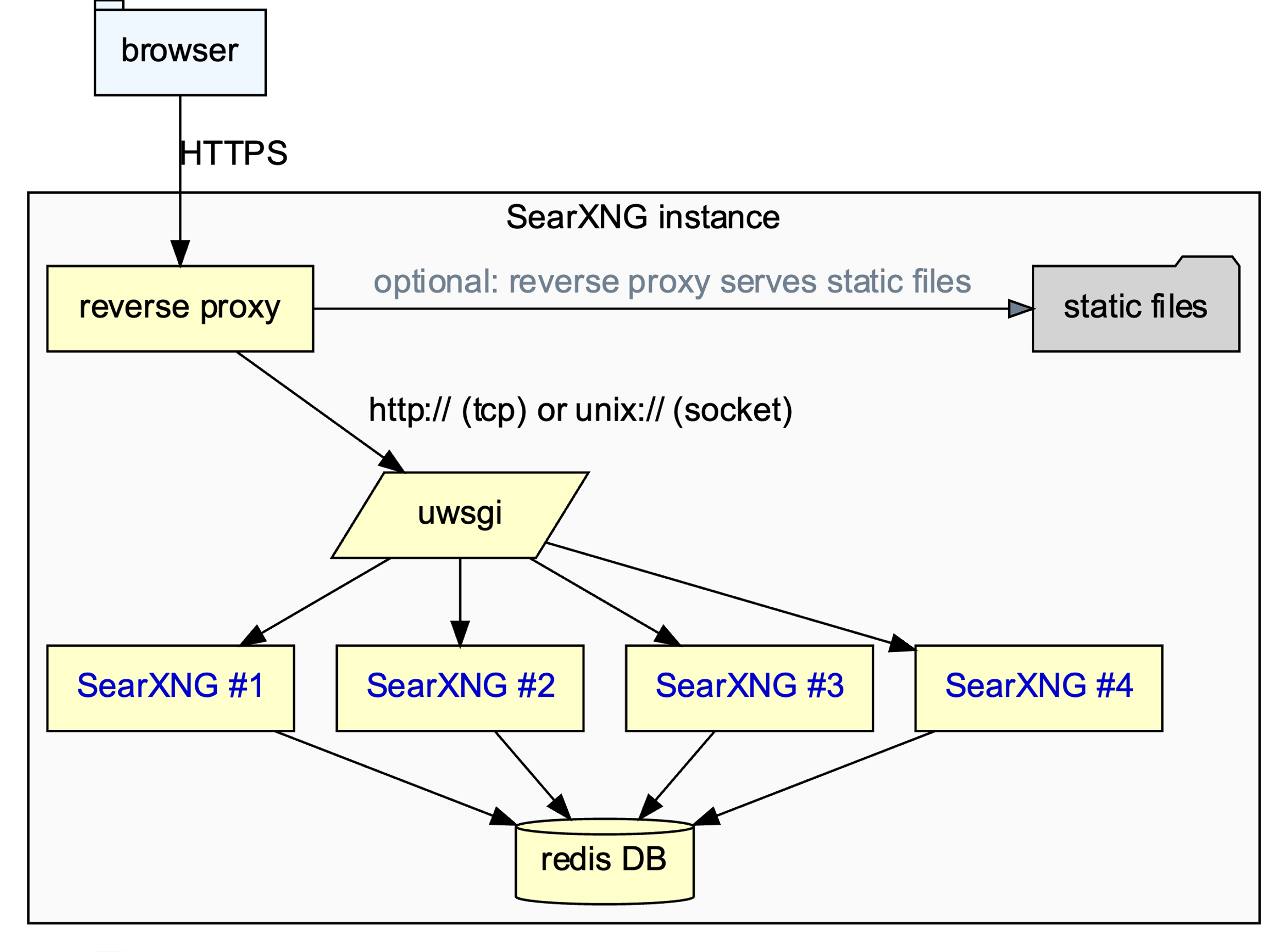
Once the dataset was generated, it was divided into the following set

**5.3 Few-Shot Prompting: Relevant 10-Shot Prompts**

Once the dataset was generated, it was divided into the following set

**5 Pretrained LLM Approach with RAG, FAISS & SearXNG**

This is an experiment to use Search functionality using meta-search engines - like Google, Bing, DuckDuckGo and others to privately browse the web and get results. The architecture of SearXNG is as follows:

****Figure: Architecture of SearXNG [7]

The goal is to utilize this search capabilities to improve the prompts to local models. This allows them to give more context into what each merchant is doing. Below is the proposed flow:

1. LLM has received the merchant descriptor & merchant pair for a new or rare merchant that it does not have much information about.
2. In such a case, the LLM utilizes SearXNG to Query about the Merchant. For example, it may ask:

*Who is [Merchant Name]?* or just *[Merchant Name]*

1. Receive clean and condensed results from Langchain.
2. Add the extra information back into the RAG prompt:  
    *# remaining code  
    prompt += f"External Info: {snippet} \n\n"  
    prompt += "Now classify this pair:\nDescriptor: ..."  
    # remaining code*
3. Then LLM does the classification with this extra information.

**5.1 RAG, FAISS & SearXNG Results**

The results of augmenting web search into the pipeline for rare merchants provided the following results:

**5.2 Visualizing Results**

First looking into the result of the LLaMA model on validation and test sets in terms of the confusion matrix elements such as True Positive & True Negative (Correct), False Positive and False Negative.

A graph with text and numbers

AI-generated content may be incorrect.

A graph of different colored bars

AI-generated content may be incorrect. A graph with different colored bars

AI-generated content may be incorrect.

**A graph with text and numbers

AI-generated content may be incorrect.**

**A graph with blue and orange bars

AI-generated content may be incorrect.A graph with different colored bars

AI-generated content may be incorrect.**

Below are the same results for the test set for LLaMA & Gemma:

**A graph with text and numbers

AI-generated content may be incorrect.**

**A graph of different colored bars

AI-generated content may be incorrect.** **A graph with text and numbers

AI-generated content may be incorrect.**

**A graph with a bar and a number of error

AI-generated content may be incorrect.  
A graph of a bar graph

AI-generated content may be incorrect. A graph with different colored bars

AI-generated content may be incorrect.**

From the above figures, we can see that the Gemma models were able to consistently provide ‘Correct’ classification (true positives & true negatives). Gemma 3 correctly classified around 459 rows out of 535 rows on the test and validation sets.

**6 Evaluating All Models**

**6.1 XG Boost vs LLMs with Zero-Shot Prompting**

The overall results based on accuracy are as follows:

Table: Comparing Accuracy Metric:

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Train Accuracy | Val Accuracy | Test Accuracy |
| XG Boost | 0.69 | 0.63 | 0.65 |
| Llama 3.2 | 0.47 | 0.46 | 0.47 |
| Gemma 3 | 0.78 | 0.79 | 0.76 |

Here, we can see that Gemma3 outperformed XG Boost and Llama 3.2 across all sets — especially in validation and test, indicating better generalization with zero-shot prompts.

The overall results based on F1-Score are as follows:

Table: Comparing Weighted Average F1-Score Metric:

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Train F1-Score | Val F1-Score | Test F1-Score |
| XG Boost | 0.66 | 0.60 | 0.63 |
| Llama 3.2 | 0.32 | 0.31 | 0.33 |
| Gemma 3 | 0.76 | 0.77 | 0.74 |

Again, Gemma3 offers a better harmonic balance between precision and recall compared to XG Boost, especially for real-world generalization.

**6.2 LLMs with Zero-Shot Prompting vs 10-Shot Prompting**

Comparing Llama3.2 results on Train Set:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prompting | Accuracy | ‘Mismatch’  F1-Score | ‘Match’  F1-Score | Macro  F1-Score | Weighted F1-Score |
| Zero-Shot | 0.47 | 0.63 | 0.05 | 0.34 | 0.32 |
| 10-Shot | 0.50 | 0.64 | 0.19 | 0.41 | 0.40 |

Here we can see some improvement on each of the metrics for Llama 3.2 with 10-shot prompting, however, it is still much worse than the baseline XG Boost model. Therefore the evaluation based on validation and test sets were not performed.

Comparing Gemma3 results:

Table: Gemma3 comparing b/w Zero-Shot & 10-Shot

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | Prompting | Accuracy | ‘Mismatch’  F1-Score | ‘Match’  F1-Score | Macro  F1-Score |
| Train | Zero-Shot | 0.78 | 0.68 | 0.83 | 0.75 |
| **10-Shot** | **0.81** | **0.77** | **0.84** | **0.80** |
| Validation | Zero-Shot | 0.79 | 0.70 | **0.83** | 0.77 |
| **10-Shot** | **0.80** | **0.76** | **0.83** | **0.80** |
| Test | Zero-Shot | 0.76 | 0.65 | **0.82** | 0.73 |
| **10-Shot** | **0.78** | **0.73** | **0.82** | **0.77** |

A clear takeaway from the above table shows that 10-shot prompting consistently improves results across all metrics, especially f1-score for ‘MISMATCH’ label.

**6.3 10-Shot Prompting: Fixed vs RAG Based**

Comparing Llama3.2 results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | Prompting | Accuracy | ‘Match’  F1-Score | ‘Mismatch’  F1-Score | Macro  F1-Score |
| Train | Fixed | 0.50 | 0.19 | **0.64** | 0.41 |
| **RAG** | **0.68** | **0.72** | 0.63 | **0.68** |
| Validation | Fixed | - | - | **-** | - |
| RAG | 0.62 | 0.58 | 0.66 | 0.62 |
| Test | Fixed | - | - | **-** | - |
| RAG | 0.73 | 0.78 | 0.66 | 0.72 |

The table below shows the details for Gemma3 model:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | Prompting | Accuracy | Match  F1- Score | Mismatch F1-Score | Macro  F1-Score |
| Train | **Fixed** | **0.81** | **0.84** | **0.77** | **0.80** |
|  | **RAG** | 0.79 | 0.83 | 0.72 | 0.78 |
| Validation | **Fixed** | **0.80** | **0.83** | **0.76** | **0.80** |
|  | **RAG** | 0.78 | 0.81 | **0.76** | 0.78 |
| Test | Fixed | 0.78 | 0.82 | 0.73 | 0.77 |
|  | **RAG** | **0.86** | **0.89** | **0.83** | **0.86** |

Interestingly, these results show that the RAG based approach is slightly worse on all accounts in the training and validation sets. However, test performance is significantly improved with RAG based approach performing with over 8% accuracy and 9% macro F1-Score.

**6.4 Comparing Test Performance on All Three Models**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | Technique | Accuracy | Match  F1- Score | Mismatch  F1-Score | Macro  F1-Score |
| ****Gemma3**** | **RAG 10-shot** | **0.86** | **0.89** | **0.83** | **0.86** |
| Fixed 10-shot | 0.78 | 0.82 | 0.73 | 0.77 |
| Zero-shot | 0.76 | 0.82 | 0.65 | 0.73 |
| **LLaMA3.2** | RAG 10-shot | 0.73 | 0.78 | 0.66 | 0.72 |
| Fixed 10-shot | - | - | - | - |
| Zero-shot | 0.47 | 0.06 | 0.64 | 0.35 |
| **XG Boost** | Baseline | 0.65 | 0.73 | 0.51 | 0.62 |

Gemma3 with RAG 10-shot based prompting is the top performer across every metric with 21% accuracy over XG Boost, and more than 24 points in macro F1-score. We can see that using RAG 10-shot prompting, even the Llama3.2 model performs with 12% improvement on accuracy over the XG Boost baseline and 16 points in macro F1-score.

**6.5 Comparing Test Performance on RAG & SearXNG**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | Technique | Accuracy | Match  F1- Score | Mismatch  F1-Score | Macro  F1-Score |
| **Gemma3** | **RAG & SearXNG** | **0.86** | **0.88** | **0.82** | **0.85** |
| **RAG Only** | **0.86** | **0.89** | **0.83** | **0.86** |
| Fixed 10-shot | 0.78 | 0.82 | 0.73 | 0.77 |
| Zero-shot | 0.76 | 0.82 | 0.65 | 0.73 |
| **LLaMA3.2** | **RAG & SearXNG** | **0.75** | **0.80** | **0.69** | **0.74** |
| RAG Only | 0.73 | 0.78 | 0.66 | 0.72 |
| Zero-shot | 0.47 | 0.06 | 0.64 | 0.35 |
| **XG Boost** | Baseline | 0.65 | 0.73 | 0.51 | 0.62 |

**7 Challenges & Insights**

**7.1 Unpredictable LLM Response**

It is sometimes difficult to force LLMs to respond with just ‘MATCH’ or ‘MISMATCH’ and often requires prompt engineering to try out a few alternatives and perform extensive testing. For instance, the following prompt worked fine with ‘Gemma3’ models, but the same prompt gave lengthy responses from the ‘Llama 3.2’ model as shown in the figures below:

A screenshot of a computer

AI-generated content may be incorrect.  
Figure: Few-shot learning prompt provided to both Gemma & Llama models.

A screenshot of a computer program

AI-generated content may be incorrect.  
Figure: Green lines show correct response from Gemma3 & red lines show incorrect response from Llama3.2

Even after adding an extra line at the end of the prompt asking the model to ‘Only return the label: MATCH or MISMATCH.’, still the Llama model was responding differently as seen in the figure below.

A screenshot of a computer

AI-generated content may be incorrect.  
Figure: Even after updating the prompt, Llama 3.2 model still returns lengthy responses.

Therefore, another clever way was required to make sure we can correctly classify the words, a second cleaning method is used to reclassify the records:

A black rectangular object with white text

AI-generated content may be incorrect.  
Figure: Cleaner method

**8 Conclusion**

In this report, I have analyzed the performance of a traditional XG Boost Model along with Gemma 3 & LLaMA 3.2 for a text classification task. The Gemma 3 model along with RAG performed the best on this task with a macro F1-score of 0.86 on the test set. The extra step of using SearXNG to get more details of the merchant did not necessarily improve the performance of the Gemma 3 model, however it made a notable improvement for the LLaMA 3.2 model. This project shows how certain LLMs like Gemma 3 can have significant improvement to traditional machine learning models which are trained specifically for one task.

**References**

1. <https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety>
2. <https://ai.google.dev/gemini-api/docs/api-key#macos---zsh>
3. <https://ai.google.dev/gemini-api/docs/rate-limits#free-tier>
4. <https://capitaloneshopping.com/research/number-of-credit-card-transactions/>
5. <https://www.datacamp.com/tutorial/few-shot-prompting>
6. <https://www.nvidia.com/en-us/glossary/xgboost/>
7. <https://docs.searxng.org/admin/architecture.html#arch-public>
8. <https://github.com/searxng/searxng-docker/>