CS553 Information Retrieval System Homework-1

Mahmut Furkan Gön
21902582
furkan.gon@bilkent.edu.tr

1. Introduction

This homework explores integrating different retrieval techniques, including TF-IDF, BM25, and embedding-based models, to improve information retrieval performance. By assigning varying weights to these methods, the task evaluates their combined effectiveness using the Mean Average Precision (MAP) metric.

The goal is to analyze how weight configurations impact retrieval performance for different embedding models (scratch-trained FastText, pre-trained FastText, and fine-tuned FastText models) and to identify the best and worst-performing configurations. Through systematic experimentation and visualization, the homework provides insights into the strengths and weaknesses of these approaches in handling diverse queries.

2. Coding and Implementation

Python language is used in this homework assignment. I followed the steps explained in the homework assignment. Firstly, I downloaded the CISI dataset. Then, load the dataset and examine the dataset. Investigated the columns in each csv file. Then, I considered the possible preprocessing steps. I applied the following preprocessing steps one by one:

- Decapitalized (lowered) every word.
- Removed the punctuation.
- Tokenized the text with the word_tokenize function from the nltk library.
- Removed stop words using stopwords from the nltk corpus.
- Lemmatized the text with WordNetLemmatizer from the nltk library again.

```python
def preprocess_text(text):
    # Ensure the input is a string, else return an empty string
    if not isinstance(text, str):
        return ""
    # Lowercasing
    text = text.lower()
    # Removing punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenizing
    tokens = word_tokenize(text)
    # Removing stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    # Join tokens back into a single string
    return ' '.join(tokens)
```

Fig 1: Preprocessing function

3. Experiments, Results

In this section, I present the results I obtained from the experiments. At the end of this section, I discussed the results, possible reasons, and insights from the results.

Model	TF-IDF Weight	BM25 Weight	Embedding Weight	MAP
scratch	1	0	0	0.05106627128060904
scratch	0	1	0	0.05172594739367601
scratch	0	0	1	0.051785456414701794
scratch	0.0	0.1	0.9	0.05582977971070579
scratch	0.1	0.1	0.8	0.05934239478655705
scratch	0.2	0.1	0.7	0.05893289510455144
scratch	0.3	0.1	0.6	0.058287938634519046
scratch	0.4	0.1	0.5	0.05680398853649627
scratch	0.5	0.1	0.4	0.05497962806785183
scratch	0.6	0.1	0.3	0.05327287920261702
scratch	0.7	0.1	0.2	0.05286030743177971
scratch	0.8	0.1	0.1	0.052415352862821014
scratch	0.9	0.1	0.0	0.052157772214985486
scratch	0.0	0.2	0.8	0.056263600551108225
scratch	0.1	0.2	0.7	0.058556747766503915
scratch	0.2	0.2	0.6	0.057310545316372266
scratch	0.3	0.2	0.5	0.0571991783565703
scratch	0.4	0.2	0.4	0.05657693061188594
scratch	0.5	0.2	0.3	0.055306716260049835

scratch	0.6	0.2	0.2	0.053507380745870525
scratch	0.7	0.2	0.1	0.05209256263797636
scratch	0.8	0.2	0.0	0.05218205361899426
scratch	0.0	0.3	0.7	0.05503711761417479
scratch	0.1	0.3	0.6	0.056575946076264705
scratch	0.2	0.3	0.5	0.05624713374155509
scratch	0.3	0.3	0.4	0.056446192849699275
scratch	0.4	0.3	0.3	0.05530321448266533
scratch	0.5	0.3	0.2	0.053003568661947366
scratch	0.6	0.3	0.1	0.052402818885380356
scratch	0.7	0.3	0.0	0.051883941480027444
scratch	0.0	0.4	0.6	0.054680079908744135
scratch	0.1	0.4	0.5	0.056458659124057245
scratch	0.2	0.4	0.4	0.05624655985244379
scratch	0.3	0.4	0.3	0.055031833972036766
scratch	0.4	0.4	0.2	0.05444676206339012
scratch	0.5	0.4	0.1	0.05352543488692246
scratch	0.6	0.4	0.0	0.05257476761808997
scratch	0.0	0.5	0.5	0.05406378192530837
scratch	0.1	0.5	0.4	0.05531891338504431
scratch	0.2	0.5	0.3	0.0548069424249132
scratch	0.3	0.5	0.2	0.05520153733086391
scratch	0.4	0.5	0.1	0.054744766928360104
scratch	0.5	0.5	0.0	0.05298100720218981
scratch	0.0	0.6	0.4	0.053409075467556064
scratch	0.1	0.6	0.3	0.054071311101273216
scratch	0.2	0.6	0.2	0.054365806620078484
scratch	0.3	0.6	0.1	0.0553016229099144
scratch	0.4	0.6	0.0	0.05360318099184925
scratch	0.0	0.7	0.3	0.05357445613382243
scratch	0.1	0.7	0.2	0.054660634412638194
scratch	0.2	0.7	0.1	0.054176898186078674
scratch	0.3	0.7	0.0	0.05517497893242858
scratch	0.0	0.8	0.2	0.05274161898953321
scratch	0.1	0.8	0.1	0.05383704209409983
scratch	0.2	0.8	0.0	0.05412741505385292
scratch	0.0	0.9	0.1	0.052304295937540664
scratch	0.1	0.9	0.0	0.053549910243703994
pretrained	1	0	0	0.05106627128060904
pretrained	0	1	0	0.05172594739367601
pretrained	0	0	1	0.018990591306322354
pretrained	0.0	0.1	0.9	0.02815673551574203

pretrained	0.1	0.1	0.8	0.03844822411384682
pretrained	0.2	0.1	0.7	0.04256487254781539
pretrained	0.3	0.1	0.6	0.04745250553139519
pretrained	0.4	0.1	0.5	0.04979396954880709
pretrained	0.5	0.1	0.4	0.051586639474824295
pretrained	0.6	0.1	0.3	0.05240828758922308
pretrained	0.7	0.1	0.2	0.05253439989776238
pretrained	0.8	0.1	0.1	0.05217730272958026
pretrained	0.9	0.1	0.0	0.052157772214985486
pretrained	0.0	0.2	0.8	0.035753536476316995
pretrained	0.1	0.2	0.7	0.04280452191206623
pretrained	0.2	0.2	0.6	0.04645362842936972
pretrained	0.3	0.2	0.5	0.04993015802943292
pretrained	0.4	0.2	0.4	0.0522877872342114
pretrained	0.5	0.2	0.3	0.052439005243169794
pretrained	0.6	0.2	0.2	0.05226174851135952
pretrained	0.7	0.2	0.1	0.052170168612560186
pretrained	0.8	0.2	0.0	0.05218205361899426
pretrained	0.0	0.3	0.7	0.04064936202947387
pretrained	0.1	0.3	0.6	0.04669087095604629
pretrained	0.2	0.3	0.5	0.050652031932013585
pretrained	0.3	0.3	0.4	0.05212665961309989
pretrained	0.4	0.3	0.3	0.052409679105476
pretrained	0.5	0.3	0.2	0.0530910894389454
pretrained	0.6	0.3	0.1	0.05262432726008912
pretrained	0.7	0.3	0.0	0.051883941480027444
pretrained	0.0	0.4	0.6	0.04382403495052723
pretrained	0.1	0.4	0.5	0.04903415388693943
pretrained	0.2	0.4	0.4	0.05225056275526557
pretrained	0.3	0.4	0.3	0.05343515458268431
pretrained	0.4	0.4	0.2	0.053080903273588456
pretrained	0.5	0.4	0.1	0.05223027029136584
pretrained	0.6	0.4	0.0	0.05257476761808997
pretrained	0.0	0.5	0.5	0.04698273039610928
pretrained	0.1	0.5	0.4	0.05078175688998129
pretrained	0.2	0.5	0.3	0.052956478100135033
pretrained	0.3	0.5	0.2	0.05478732489748218
pretrained	0.4	0.5	0.1	0.05347307515525036
pretrained	0.5	0.5	0.0	0.05298100720218981
pretrained	0.0	0.6	0.4	0.04913379843894235
pretrained	0.1	0.6	0.3	0.05280324376217966
pretrained	0.2	0.6	0.2	0.05319885438685883

pretrained	0.3	0.6	0.1	0.054848052657123926
pretrained	0.4	0.6	0.0	0.05360318099184925
pretrained	0.0	0.7	0.3	0.050590365548116724
pretrained	0.1	0.7	0.2	0.052546636430355606
pretrained	0.2	0.7	0.1	0.05400374714969718
pretrained	0.3	0.7	0.0	0.05517497893242858
pretrained	0.0	0.8	0.2	0.051234538801638405
pretrained	0.1	0.8	0.1	0.05287175864422831
pretrained	0.2	0.8	0.0	0.05412741505385292
pretrained	0.0	0.9	0.1	0.05012286193029343
pretrained	0.1	0.9	0.0	0.053549910243703994
finetuned	1	0	0	0.05106627128060904
finetuned	0	1	0	0.05172594739367601
finetuned	0	0	1	0.05346210669539131
finetuned	0.0	0.1	0.9	0.05892331136486038
finetuned	0.1	0.1	0.8	0.060731256770783205
finetuned	0.2	0.1	0.7	0.05916204809861412
finetuned	0.3	0.1	0.6	0.05830069522592451
finetuned	0.4	0.1	0.5	0.05784380772948028
finetuned	0.5	0.1	0.4	0.055800514806221446
finetuned	0.6	0.1	0.3	0.05371812934246286
finetuned	0.7	0.1	0.2	0.052935188362258266
finetuned	0.8	0.1	0.1	0.052492459996522266
finetuned	0.9	0.1	0.0	0.052157772214985486
finetuned	0.0	0.2	0.8	0.057388964171932934
finetuned	0.1	0.2	0.7	0.05906649133534935
finetuned	0.2	0.2	0.6	0.05798993966894849
finetuned	0.3	0.2	0.5	0.05809737116417444
finetuned	0.4	0.2	0.4	0.057267165102312535
finetuned	0.5	0.2	0.3	0.05564148681757513
finetuned	0.6	0.2	0.2	0.05406564732208454
finetuned	0.7	0.2	0.1	0.05223583752971832
finetuned	0.8	0.2	0.0	0.05218205361899426
finetuned	0.0	0.3	0.7	0.05732258963029452
finetuned	0.1	0.3	0.6	0.05745053924423698
finetuned	0.2	0.3	0.5	0.05748307107997188
finetuned	0.3	0.3	0.4	0.05653002724093476
finetuned	0.4	0.3	0.3	0.0558989975415657
finetuned	0.5	0.3	0.2	0.05347202307274018
finetuned	0.6	0.3	0.1	0.05250964459171733
finetuned	0.7	0.3	0.0	0.051883941480027444
finetuned	0.0	0.4	0.6	0.05601317189512648

finetuned	0.1	0.4	0.5	0.057272859466867374
finetuned	0.2	0.4	0.4	0.05672039703940664
finetuned	0.3	0.4	0.3	0.05590114080498122
finetuned	0.4	0.4	0.2	0.05519523997140936
finetuned	0.5	0.4	0.1	0.05359239015113045
finetuned	0.6	0.4	0.0	0.05257476761808997
finetuned	0.0	0.5	0.5	0.05562722998453557
finetuned	0.1	0.5	0.4	0.0561837289839155
finetuned	0.2	0.5	0.3	0.05509778280423603
finetuned	0.3	0.5	0.2	0.055149171824676706
finetuned	0.4	0.5	0.1	0.054734849339614935
finetuned	0.5	0.5	0.0	0.05298100720218981
finetuned	0.0	0.6	0.4	0.054124725464122725
finetuned	0.1	0.6	0.3	0.05496758068518452
finetuned	0.2	0.6	0.2	0.05466606587810567
finetuned	0.3	0.6	0.1	0.05601504534015741
finetuned	0.4	0.6	0.0	0.05360318099184925
finetuned	0.0	0.7	0.3	0.05392068350957547
finetuned	0.1	0.7	0.2	0.05416262780577079
finetuned	0.2	0.7	0.1	0.05416341501861336
finetuned	0.3	0.7	0.0	0.05517497893242858
finetuned	0.0	0.8	0.2	0.05337049167791565
finetuned	0.1	0.8	0.1	0.05431066889933724
finetuned	0.2	0.8	0.0	0.05412741505385292
finetuned	0.0	0.9	0.1	0.05259970845383551
finetuned	0.1	0.9	0.0	0.053549910243703994

Table 1: Results of the extensive experiments

Table 1 depicts the results I obtained from the experiments. For each FastText model, I weighted TF-IDF, BM25, and FastText embeddings so that the weights' sum adds up to 1.0. I tried every possible combination of multiple of 0.1 for every ranking and embedding. The following figures demonstrate the results for different controlled variables. For all plots for each and every controlled and uncontrolled variable, please see the Jupyter Notebook (hw.ipynb.) Here, I provide just some example plots for a reasonable number of pages in this report. There, you will be able to see all results given in Table 1 visually.

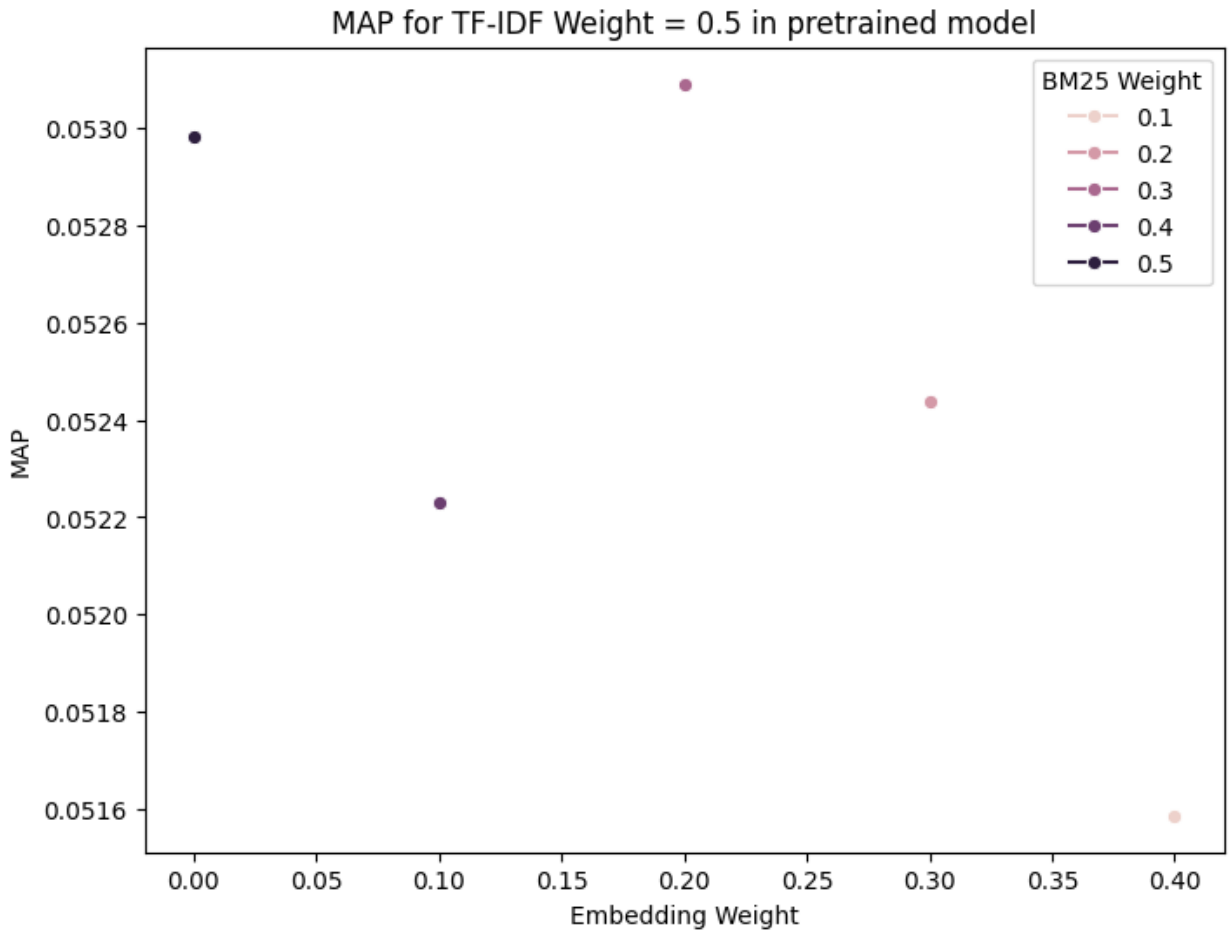

Fig 2: MAP Scores for TF-IDF fixed at 0.5 and varying Embedding and BM25 weights for Pretrained Model

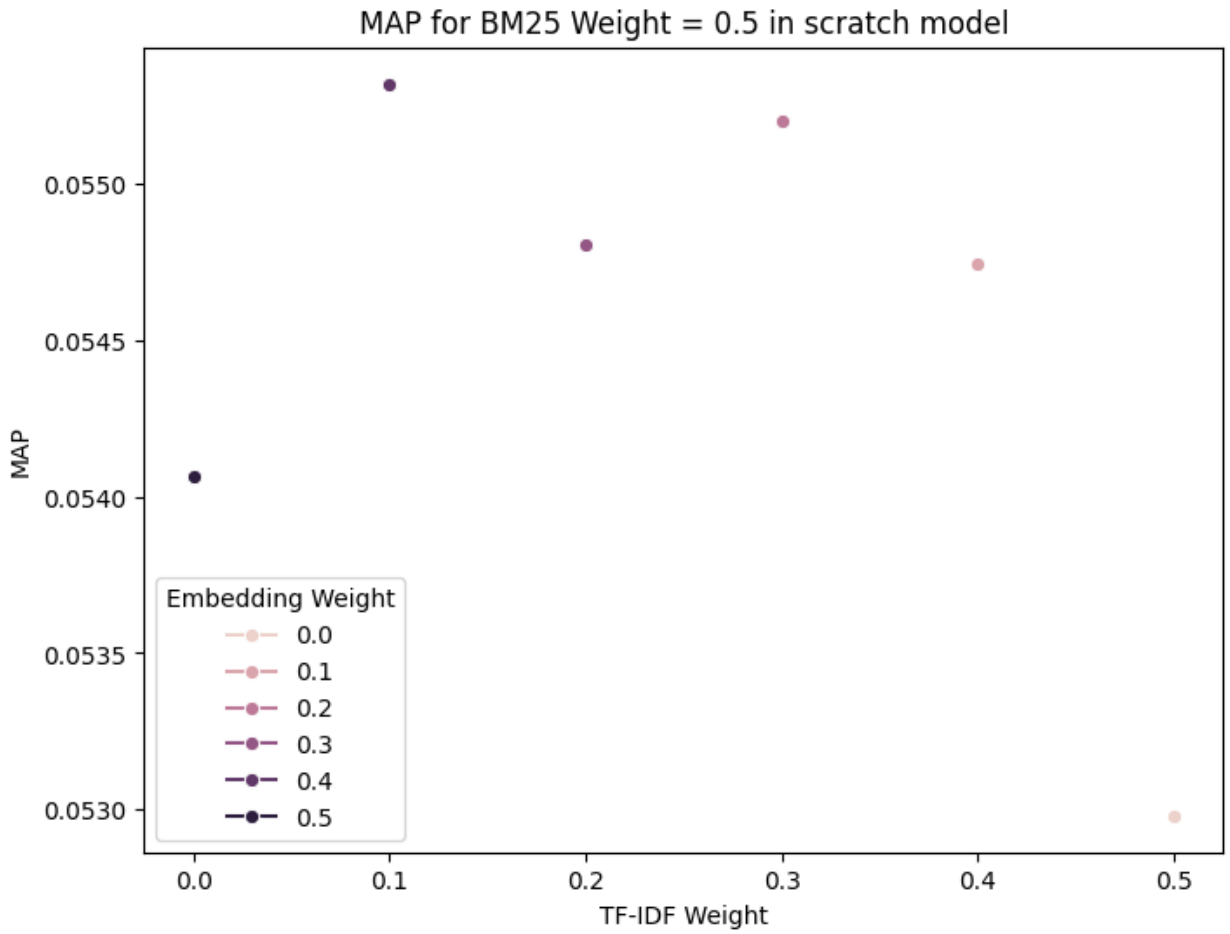

Fig 3: MAP Scores for BM25 fixed at 0.5 and varying Embedding and TF-IDF weights for Scratch Model

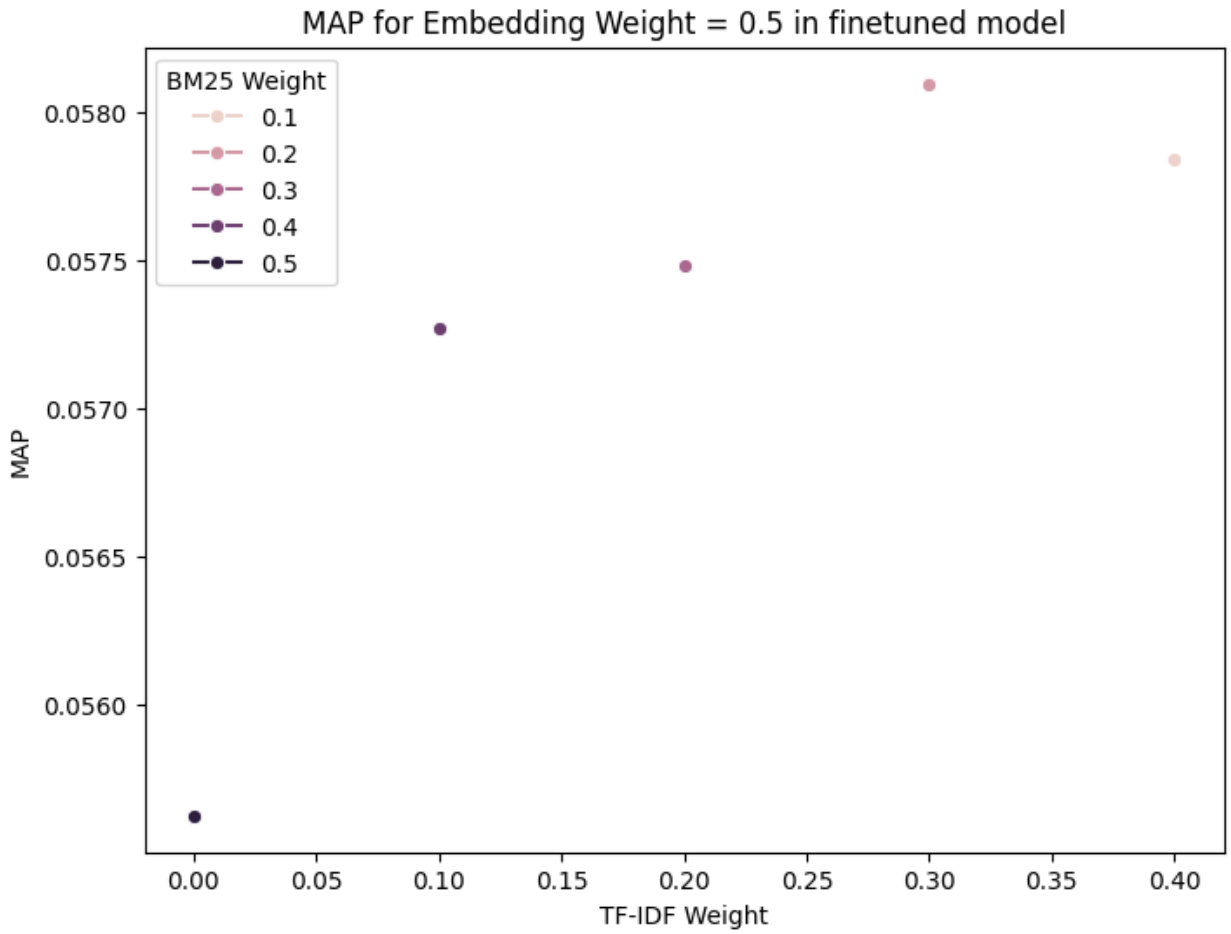

Fig 4: MAP Scores for Embedding fixed at 0.5 and varying BM25 and TF-IDF weights for Finetuned Model

Figure 2,3,4 are just some examples of the results of the experiments I conducted. Please go to the Jupyter Notebook to see all plots in a great detail.

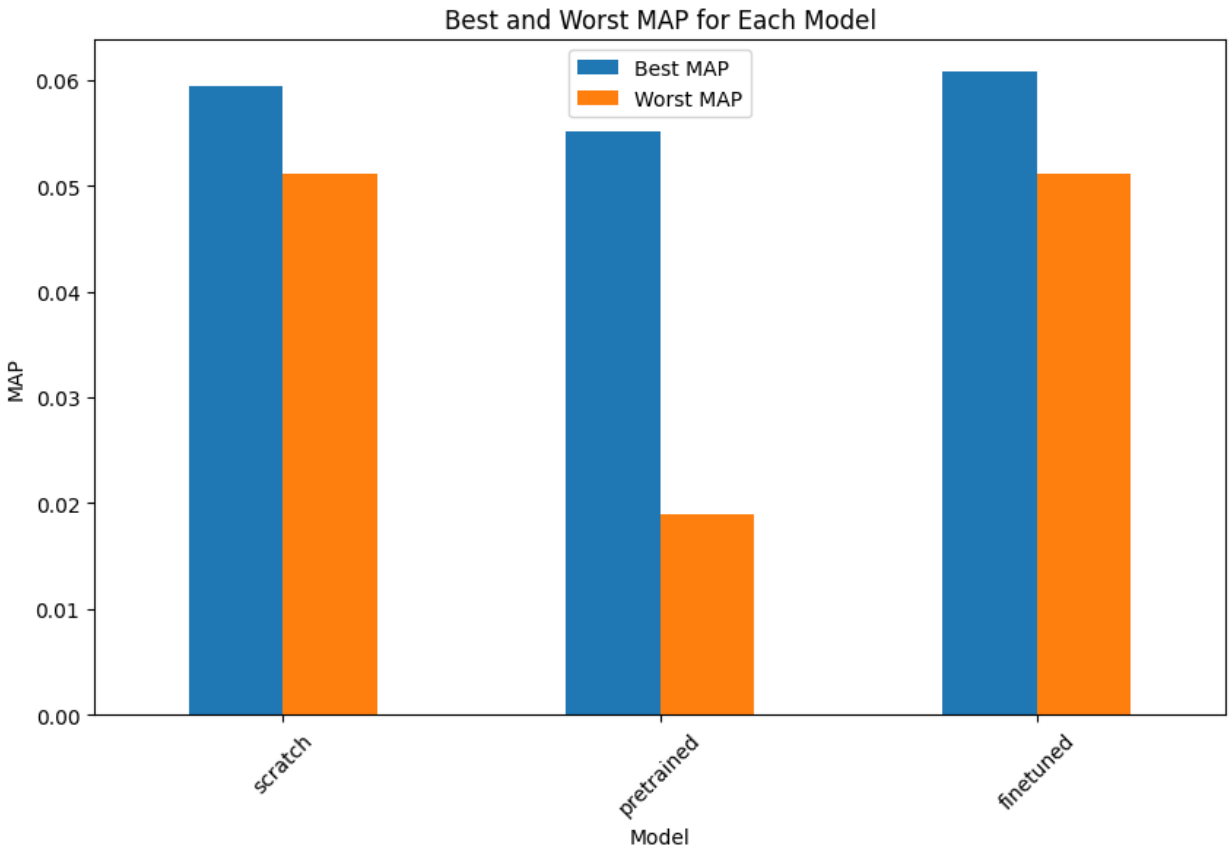

Fig 5: Best and Worst Map Scores for each Model

Figure 5 illustrates each model's best and worst performances based on MAP scores. This graph shows that the finetuned model is the best-performing model based on MAP scores. The model that we trained is the second in the performance. While the pre-trained model is the third in the performance. This is intuitively justifiable, as the pretrained model has no idea about the CISI dataset. Since both the model from scratch and the finetuned model know the dataset, their best and worst performances are similar and better than the poor performance of the pretrained model.

4. Discussion

The experiments conducted yielded several key insights into the performance of the tested models—scratch, pretrained, and finetuned—under varying weight configurations for TF-IDF, BM25, and embeddings. The results demonstrate that the interplay between these weights significantly affects the overall MAP scores, highlighting the importance of balance in multimodal retrieval systems.

Across all experiments, the finetuned model consistently outperformed the scratch and pretrained models in terms of MAP scores. This result aligns with expectations, as fine-tuning enables the model to adapt its representations to the specific nuances of the dataset, leading to

improved relevance in document retrieval. The pretrained model exhibited performance in between the scratch and finetuned models, benefiting from its initial training on large-scale data but lacking the targeted optimization seen in fine-tuning.

Interestingly, the MAP scores showed a notable increase as the embedding weight rose, particularly when paired with higher BM25 weights. This indicates that embeddings play a critical role in capturing semantic relationships, complementing the more surface-level matching provided by BM25. However, excessive reliance on embeddings (e.g., embedding weight = 1.0) caused a slight decline in MAP, suggesting diminishing returns when the contribution of other components like BM25 and TF-IDF is minimized.

while lagging behind the others, the scratch model demonstrated a steady improvement as BM25 and embedding weights were increased. This indicates that even basic models can benefit from robust weighting strategies, albeit to a lesser extent than their pretrained and finetuned counterparts.

The results underscore the value of fine-tuning for domain-specific applications, particularly in scenarios where domain data exhibits characteristics not captured in general-purpose pretraining. The strong performance of embeddings also highlights the growing importance of dense retrieval techniques in modern information retrieval systems.

From a practical perspective, these findings suggest that hybrid approaches combining traditional techniques like BM25 with dense embeddings are highly effective. By carefully tuning the weights for these components, systems can achieve a balance that leverages the strengths of both paradigms—robust keyword matching and nuanced semantic understanding.

These insights have broader implications for developing retrieval systems in domains such as legal, healthcare, or academic research, where the precision of retrieved documents is paramount. Fine-tuning models on domain-specific corpora and optimized weight configurations could substantially improve retrieval quality in these contexts.

5. Conclusion

In conclusion, this homework assignment highlights the importance of leveraging fine-tuned models and optimized weighting strategies for enhancing document retrieval performance. The results demonstrate that combining traditional techniques like BM25 with dense embeddings yields better MAP scores, emphasizing the complementary strengths of these approaches. Fine-tuning, in particular, is crucial for adapting models to domain-specific datasets, significantly outperforming scratch and pretrained models. These findings suggest that hybrid methods can improve retrieval systems in fields requiring high precision, such as healthcare, legal, or academic research. It was a nice experience to do this homework.