Text Technologies, Coursework 2

marko mekjavic : s1813308

December 2021

1 Introduction

This is a report on the second coursework, dealing with IR System evaluation as well as Text Analysis and Text Classification. The following report is split into three parts - IR Evaluation, Text Analysis, and Text Classification.

2 Assignment

2.1 Code Structure

All three tasks of the coursework have been implemented in one file - `code.py`. The first task for **IR Evaluation** was organised into an `Eval class`, whose structure is described in section 3.1. The second task, **Text Analysis**, was divided into two parts, the *Chi Squared (referred to CHI SQ)* and *Mutual Information (referred to MI)* part, and the *LDA* part. The code spreads between lines 405 and 747 for the first part and up to 881 for the second part. The last task, **Text Classification**, is divided into subsections but can mainly be summarised with the section dedicated to the development of the baseline model, and the second section dedicated to improving the baseline model.

2.2 Learning Outcomes

Working on this project has helped me develop many new, as well as sharpen already existing skills. The tasks required development of a large scale code-base, which allowed me to become more confident navigating and working with larger files. Moreover, I also learnt the importance of structuring my code into a cohesive format so that changes can be performed easily. Furthermore, thorough and efficient comments proved to be of utmost importance.
Performing various evaluation tasks I learnt a lot about statistical analysis. From understanding the responsibility of each metric as well as their limitations, I learnt how to use them correctly in order to perform desired evaluations.

2.3 Challenges Faced

One of the biggest challenges I faced was navigating and working with such a large code-base. Mainly I had issues structuring my code into a format in which I could easily perform small changes or replace only specific components. Moreover, working with large files also made it harder to remember what each part of the file is responsible for. Learning how to structure my files into efficient components and thorough commenting helped me overcome these challenges.
The second challenge I would point out was working on the optimisation of the classification model in task 3. Since so many improvements turned out to decrease the model's performance, I realised it is important to dive deeper into the performance of the model, and think about what each potential improvement would bring, instead of just randomly changing parameters. This required more reading and researching as well as analysis of my model and its performance.

3 IR Evaluation

The main objective of the IR Evaluation task was to compare and evaluate the performance of six different IR systems on ten different queries, using various metrics. Each evaluation metric - listed below - is calculated for each system-query pair, while only focusing on the relevant document set for that specific query.
For evaluating the performance of the IR Systems, we used the following metrics; *P@10 - precision at cutoff 10*, *R@50 - recall at cutoff 50*, *r-precision*, *average precision*, *nDCG@10: normalized discount cumulative gain at cutoff 10*, and *nDCG@20: normalized discount cumulative gain at cutoff 20*.

3.1 Code Implementation

The information representation - reading of the input files and translating them into a desired format - as well as calculations of all metrics are hadnled by the `Eval class`. Each metric is an attribute of the class and is calculated by a method, for instance `self.r_50 = self.r_50(self.file_sys_res)` is the calculation of the recall at cut-off 50 for all of the systems.
The input files were translated into two separate dictionaries, `file_qrels` and `file_sys_rels`. The first dictionary represents the results for each query, where a the keys were the query numbers, while the values are the lists of corresponding files for the given query. The second dictionary stores the information about which documents were retrieved by each system for each of the ten queries. This was accomplished with the implementation of a nested dictionary, where the first key represents the system, while the second key represent the query, and the value corresponds to the list of all retrieved documents for that given query by that given system.

3.2 Best Systems - based on scores

The results regarding the performance of each of the six IR systems can be seen in the table below. Note that all the values for the metrics are averaged across all queries for each of the given systems.
The best result for the given metric is coloured with a light green, while the second best is coloured by a light blue. If more than one system has scored the highest value, then all with the same value are coloured with the same colour.

System	P@10	R@50	R-Precision	AP	nDCG@10	nDCG@20
1	0.39	0.834	0.401	0.4	0.363	0.485
2	0.22	0.867	0.253	0.3	0.2	0.246
3	0.41	0.767	0.448	0.451	0.42	0.511
4	0.08	0.189	0.049	0.075	0.069	0.076
5	0.41	0.767	0.358	0.364	0.332	0.424
6	0.41	0.767	0.448	0.445	0.4	0.491

Table 1: The mean scores of systems S1-S6.

In order to determine whether the best system - for each metric separately - is statistically significantly better than the second best, I used the *2-tailed t-test* with the p value of 0.05. This was used to compare the best IR system with the second best system. In case there were more systems, that scored the same highest value, there was no test needed. The results are listed below with the decision.

1. P@10: S3, S5 and S6 - T-Test = 1.0
2. R@50: S2. The *p-value* of S2-S1 paired t-test is $0.703 > 0.05$. S2 is not significantly better than S1.
3. R-Precision: S3. The *p-value* of S3-S1 paired t-test is $0.758 > 0.05$. S3 is not significantly better than S1.

4. Average-Precision: S3. The *p-value* of S3-S6 paired t-test is 0.967 $>$ 0.05. S3 is not significantly better than S6.

5. nDCG@10: S3. The *p-value* of S3-S6 paired t-test is 0.883 $>$ 0.05. S3 is not significantly better than the S6.

6. nDCG@20: S3. The *p-value* of S3-S6 paired t-test is 0.867 $>$ 0.05. S3 is not significantly better than the S6.

As we can see, regardless of the metric, no best-scoring system is significantly better than the second-best system. However, S3 has scored the highest in all categories. S1 and S6 have also performed well as they were obtaining second-highest scores.

4 Text Analysis

4.1 Token Analysis

In the table below we can see the top 10 words with best *Mutual Information - MI* and *Chi Squared - CHI SQ* values for each of the corpora.
An interesting observation, when comparing the top words based on *MI* and *CHI SQ*, is the big overlap between the words scoring the highest. In Old Testament (*OT*) there is only one word difference, while in the New Testament (*NT*) there is a two word difference, and lastly in Quran there is also only a two word difference. However, even though the top words may more or less be the same, the rankings of these words differs between *MI* and *CHI SQ*.
One reason for the overlap in the results can be the fact that both measurements take into account the count of each word, and hence words that occur more frequently should score higher. On the other hand, a justification for the differences within the results can be argued by the difference in each of the measurements. The *CHI SQ* is examining the raw counts of each word in order to obtain the difference between the observed and expected frequencies and hence the sample size matters. On the other hand, *MI* is examining only the marginal and joint probability distributions and does not take into account the size of the sample.

	OT		NT		Quran	
	MI	CHI SQ	MI	CHI SQ	MI	CHI SQ
1	jesu	jesu	jesu	jesu	god	muhammad
2	israel	lord	christ	christ	muhammad	god
3	king	israel	lord	lord	believ	believ
4	lord	king	discipl	discipl	torment	torment
5	christ	believ	israel	paul	messeng	messeng
6	believ	christ	paul	peter	revel	revel
7	muhammad	god	peter	thing	king	unbeliev
8	god	muhammad	king	spirit	disbeliev	disbeliev
9	son	son	peopl	israel	unbeliev	forgiv
10	torment	faith	thing	john	israel	guidanc

Table 2: Mutual Information (*MI*) and Chi Squared (*CHI SQ*) for each of the three corpora.

A very interesting observation from the table is the fact that both words, *jesu* and *muhammad* also appear in the *OT* scoring because they never appear in the corpus itself. Moreover, *jesu* more or less only appears in *NT* while *muhammed* only appears in *Quran*. However, with regards to *MI* this does make sense because from seeing the word *jesu* we know the text probably belongs to the *NT*, and this also means that the word has a lot of mutual information with respect to the class *OT*, because just from seeing the word, we know the text is **not** about the *OT*. On the other hand, it also does not surprise that the two words score highly in the *CHI SQ* rankings for *OT*. Since we are expecting the two words to be equally spread across the three corpora - but in reality they are very much limited to *NT* and *Quran* respectably - this results in higher *CHI SQ* values as there is a higher deviation from the expected distribution.
Having read parts of *OT* and *NT* - and ignoring the words that should not appear - I came to another interesting observation, that the top words provide a fairly good overall recap of the corpus, which amazed me. While the *OT* mainly talks about the pre-Jesus era, the *NT* talks about the role of Jesus and his disciples - amongst which were also Paul and Peter!

4.2 Topic Analysis

For Topic Analysis we used the Latent Dirichlet Allocation *LDA* and limited it to 20 most common topics. For each corpora we then found the ten most related tokens for the given topic. In order to generate same results on all repetitions, I decided to set the `random_seed` parameter to 53.

	OT		NT		Quran	
	token	score	token	score	token	score
1	son	0.106	thing	0.105	god	0.148
2	father	0.079	god	0.085	lord	0.064
3	receiv	0.065	life	0.061	peopl	0.055
4	messeng	0.060	good	0.041	truth	0.053
5	god	0.037	answer	0.038	merci	0.051
6	brother	0.032	creat	0.036	fear	0.048
7	favor	0.031	man	0.036	love	0.042
8	find	0.025	told	0.033	righteous	0.029
9	commit	0.025	reward	0.033	great	0.027
10	dead	0.023	death	0.028	nation	0.024

Table 3: Best tokens and their scores for the best topic for each of the corpora.

The topic I would choose for the *OT* is the *obedience and disobedience* as this explains the relation between the *god/father* and the *son/brother*. The actions the later can take can either obey or disobey god/father and the consequences can therefore either be rewarding - *favor, receive* and *find* - or they can be dreadful, such as *dead*.
The top tokens for the topic in *NT* mostly have very positive connotation, such as *life, good, answer, creat* and *reward*. The other words are harder to group - *god, death* and *thing*. For this corpus I would choose the topic of *salvation and redemption*, which goes very well with the tokens, especially considering that the *NT* is referring to the World after the *OT*.
The topic with which I would label the tokens in *Quran* is *divine judgement*, as this is the relation between the *god/lord* and the *people/nation*. This can be seen in god's characteristic of being *righteous* and *truthful* - *merciful* to those who deserve, while the others should *fear*.

Latent Dirichlet Allocation (LDA) is a popular topic modeling technique to extract topics from a given corpus. In our example, where observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.
Examining the most common topics for each corpus, I realised there is one topic that can be observed in all corpora, which is characterised by the words *son, god, father* etc. While this is something I expected, I was especially amazed that *NT* and *Quran* had more similar topics in common than *OT* and *NT*. The two most common topics are mainly characterised by words *good, life, answer, creat* - which I named *salvation and redemption* above - and *earth, heaven, judgement, deed* and *eye* respectively.
While *CHI SQ* and *MI* provide us with the words that are most related to the given corpus, the LDA provides us with a more structured analysis as it is able to break down the corpus into topics and specifies which words are belonging to which topic.

5 Text Classification

5.1 Baseline Model

In order to develop the baseline classifier model, I used the following steps:

Data Split : The first step in preparing the data for the classification model, was to shuffle the data and split it into two parts, the training and the testing set. I decide to split my data-set dedicating 70% to the training set, and remaining 30% to the testing set, with the aim of preventing over-fitting. In order to generate same splits and minimise the randomisation, I decided to set the *random_state* parameter to 0.

Preprocess : For the preparation of the data-set for the development of the baseline model I used the preprocessing tools from the previous coursework, but have removed the stemming and stop-word removal. This included *case-folding*, *number removal*, and *tokenisation* of texts, by splitting them at all non-alpha numerical characters.

Bag of Words Formatting : The second step in the preparation of the data-set was to convert the verses/documents into a *Bag of Words* formatted matrix, columns representing the documents/verse, while the rows represented different words. Since majority of the inputs in this extraordinary big matrix were 0, I used the *sprase.dok_matrix* from the *scipy* library, allowing the algorithm to run quicker.

Vectorisation : The last step in preparing the data for the classification model was the vectorisation procedure, where each word in the document as well as the corpus the document belonged to, were represented as vectors. For that, I used two dictionaries: `word2id` and `cat2id`, where each word was given a unique ID as well as each corpus.

Model : The `SVC` model was imported from the `sklearn.svm` library, and the parameter C was set to 1000, as suggested.
After fitting the model on the training set, it was then tested on both, the development set as well as the testing set. In order to evaluate its performance, I used the `classification_report` method from the `sklearn.metrics` library. The results for each of the corpus, as well as for the whole corpora, are displayed in the table below.

Split	Precision	Accuracy	F1-Score
train	1.0	1.0	1.0
dev	0.923	0.898	0.909
test	0.919	0.9	0.909

Table 4: Scores for the baseline model.

As we can see from the table above, the model was rather successful on the *dev* as well as *test* set.

Misclassified Verses Even though the model was very successful in classifying the verses, I took a closer look at the ones that were classified wrongly - they can be seen below:

Verse	Original	Predicted
.. what have i to do with you jesus son of the most high god i implore ..	Quran	NT
there is no one equal to him	NT	Quran
.. he is a chosen vessel of mine to bear my name before gentiles kings and the children of israel	Quran	OT

Table 5: Misclassified verses.

The first misclassified sentence is very interesting because it talks about Jesus but actually belongs to the *Quran*. However, it is understandable that the model classified it as belonging to *NT* because it is a very frequent and important word for that corpus. The second sentence is rather hard for the model to predict with high certainty as it is a very short one and does not contain much information, and hence could belong to any of the three corpora. The third verse is also hard example. Israel is a common topic in the *OT*, especially in combination with the children which represent the followers of god.

5.2 Improved Model

The first improvement I touched upon was the rearrangement of my data split. Instead of using only 70% for my training sample, I decided to use 90%, which immediately showed a small increase.
For further improvements I tried various different techniques; from changing the C parameter in the SVC, and changing the *kernel*, to changing the preprocessing methods as well as feature selection. Some of them are listed below:

- A : set data split to 90%
- B : A + set C to 10
- C : A + set C to 20
- D : A + set C to 50
- E : B + normalised BOW
- F : B + stop-word removal
- G : B + stop-word removal + stemming
- H : E + top 5000 MI scoring tokens
- I : E + top 5000 CHI SQ scoring tokens

set	baseline	A	B	C	D	E	F	G	H	I
dev set	0.909	**0.910**	**0.912**	0.897	0.885	**0.914**	0.834	0.792	0.890	0.873
test set	**0.909**	0.907	0.882	0.860	0.845	0.908	0.845	0.784	0.884	0.867

Table 6: Macro-F1 scores for each model.

Surprisingly many of the methods proved to be decreasing the quality of the model. I was especially surprised seeing the decrease in the score when introducing the stop-word removal and stemming, as I thought that would make the verses more efficient for the model to analyse. Moreover, even when using only the top 5000 MI and CHI SQ scoring words, the result did not improve.
Another improvement was the normalised approach to creating the BOW matrix where instead of incriminating the frequency of the words by 1, I decided to normalise it with the length of the document. The last improvement I witnessed was the change in the C parameter. Setting it to 10 was the best achievement. The best performing model was model **E**, which had the C parameter set to 10, training and testing split was set to 90%, and used the normalised approach to generating the BOW matrix.
What is more, model **E** also managed to misclasify only 2 out of the three misclasified verses from *Table 5* - the last verse was not misclassified anymore!