Text Technologies for Data Science

Teodora Georgescu - s1530344

Coursework II

1 Introduction

Coursework II is split into two parts - IR Evaluation and Text Classification. The main task of the IR Evaluation part was to compare and evaluate IR systems using 6 different retrieval scores.

2 Overall work

The IR evaluation system was built incrementally, with helper functions extracting the information from the input files and passing them on to the actual evaluation functions.

The module takes as input a folder path, for example `./system`. In this folder, the module expects to find the following files:

- `qrels.txt` - a list of query numbers, the relevant documents for each of them and the relevance score for each document
- `S[1-6].results` - results of 6 different IR systems, each of which contains a list of queries and the relevant documents retrieved for each, in decreasing order of relevance

The module reads all the relevant documents for each query and their relevance score, then it parses through all the `S[1-6].results` files and builds a dictionary of the retrieved documents for each query, for each system. The module then computes ass the retrieval measures.

The retrieval measures used were:

- Precision at 10, with the formula:

$$P@k = \frac{\#\,of\,relevant\,documents\,retrieved\,@\,k}{\#\,of\,retrieved\,documents\,@\,k}$$

- Recall at 50, with the formula:

$$R@k = \frac{\#\,of\,retrieved\,documents\,that\,are\,relevant\,@\,k}{\#\,of\,relevant\,documents}$$

- R-Precision, with the formula:

$$R\_Precision = Precision\,@\,rank\,R$$

  for a query with known R relevant documents

- Average Precison, with the formula:

$$AP = \frac{1}{r}\sum_{k=1}^{n} P(k) * rel(k)$$

  r = `number of relevant documents for a given query`
  n = `number of documents retrieved for a query`
  P(k) = `precision @ k`
  rel(k) = `1 if retrieved doc @ k is relevant, 0 otherwise`

- Normalised Discount Cumulative Gain at 10 and 20, with the formula:

$$nDCG@k = \frac{DCG@k}{iDCG@k}$$

  where

$$DCG@k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{log_2(i)}$$

  and

$$iDCG@k = \texttt{ideal discounted cumulative gain @ k}$$

First, I computed each of these measures for each query in each system, and the results for each system are summarised in files `S[1-6].eval`. The last line in each of these files represents the average of the query scores for each measure. The averages are then collected into a file called `All.eval`, where it is easier to compare the six systems with respect to each retrieval measure.

For the second part of the coursework, I built a simple parsing module that expects three files as input:

- A training tweet collection
- A testing tweet collection
- A file mapping the tweet categories to class numbers

The module reads through the training tweet file, and for each tweet, it applies the selected methods - tokenising, hashtag decoding, link decoding, stemming, stopping and emoji decoding. This build the `feats.bow` file, which is a bag-of-words list and their associated ID. Then, the feature file for each of the training and test set is built using the bag of words model and the ID for each word encountered in the training set.

The classification model used in this coursework is a multi-class Support Vector Machine [1], developed at Cornell University. Given a feature file for the training set, it learns parameters for the SVM, then it produces a predictions file for the test set. The accuracy, macro-f1 score for the overall system and the precision, recall and f-score for each class are calculated in the python module as well, by reading through the prediction file and comparing the predicted class with the true class of each test tweet.

One of the challenging parts of this coursework was properly indexing arrays for the computation of retrieval scores, since most of them assume indices starting from 1. This lead to errors in the beginning of development, but they were quickly overcome by truly understanding the formulas. Furthermore, highly modular code was very helpful for retrieving all the different counts needed in calculating the formulas.

The most challenging part in building the classifier with improvements was retrieving links from tweets. Twitter shortens any referred links by default, and when a user clicks on a link, they are taken through a series of redirects until they land on the originally linked page. This had to be resolved by sending header packages to the redirect links, extracting the next link from the response header, and re-sending packets to the next link, until there were no more redirects.

3 IR Evaluation

The table below presents a summary of the average measures for each IR system.

	P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
S1	0.390	0.834	0.401	0.400	0.363	0.485
S2	0.220	0.867	0.253	0.300	0.200	0.246
S3	0.410	0.767	0.449	0.451	0.420	0.511
S4	0.080	0.189	0.049	0.075	0.069	0.076
S5	0.410	0.767	0.358	0.364	0.333	0.424
S6	0.410	0.767	0.449	0.445	0.400	0.490

Table 1. Average Evaluation Scores for all IR Systems

For `P@10`, systems 3, 5 and 6 were the best performing ones, each scoring an average of 0.410 across all 10 queries. Precision at K is an important measure for search engines in particular, because users tend to look only at the topmost retrieved documents. The performance of a system can be analysed with different K's as well, depending, for example, on how many results are displayed to the user on the first page. This, however, does not take into account the positions of the relevant documents in the top 10 retrieved documents (for example, a system that shows 6 irrelevant documents first, then 4 relevant documents, is scored the same as a system showing the 4 relevant documents first). It would also be a bad measure if the system gave less than K results for a query - in this case, even a perfect system would

have a P@K score less than 1. For our systems, a 0.410 P@10 means that within the first 10 retrieved documents, 41% were relevant. The three documents have the same score, so they are not even trivially different.

Looking at `R@50`, system 2 performs best, with a score of 0.867. The second-best performing system is S1, with a score of 0.834. To see if these systems have statistically significant different results, I did the t-test using the SciPy package [2] and the resulting p-value was 0.70, so the difference is **not** statistically significant. This means that the difference could have happened by chance.

Next, the best `r-Precision` score is equal for systems S3 and S6 - 0.449. This means that about 45% of all relevant documents are fetched in the top k results of each query, where k is the number of relevant documents for that query. The scores are not trivially different, so they are not statistically significantly different either.

For `Average Precision (AP)`, the best performing systems are again S3 and S6. S3 has the top score of 0.451 and S6 comes second, with a score of 0.445. This measure takes into account both the number of relevant documents retrieved, but also whether the top retrieved documents are relevant - a system that returns most of the relevant documents among the first positions will have a higher AP score than a system that returns more irrelevant documents as the top results. The difference between the scores of S3 and S6 is **not** statistically significant, with a p-value of 0.97.

Looking at `nDCG@10`, the best performing system is S3, with a score of 0.420, and S6, with a score of 0.400. There results are **not** statistically different, because their associated p-value is 0.88.

Finally, for the `nDCG@20`, S3 performs best, with a score of 0.511, and S6 comes in second, with a score of 0.490. There results are **not** statistically different either - their p-value is 0.87.

Although it could be seen as the 'fairest' evaluation measure, one drawback of the nDCG metric is that it is reliant on an ideal ordering of results. In real life, this might be hard to come across when only partial and subjective relevance feedback is available from users.

4 Text Classification

The baseline classification model built a bag-of-words model out of the tweets in the training set. The only pre-processing applied on the tweets were simple tokenisation and hyperlink removal.

After running the classification model with a set `-c` parameter of 1000, the following global scores were obtained for the baseline model:

- Accuracy - 0.672
- Macro-F1 - 0.660

The first improvement of the baseline classifier was to use a special Tweet tokeniser from the nltk package, that would keep punctuation marks, mentions, hashtags and text emojis intact. For punctuation marks and in-word characters, a maximum of three repetitions were kept - for example, `I am aweeesooooooommmeeee???????` would be tokenised to `[I, am, aweeesooommmeee, ?, ?, ?]`. This way, as little information as possible is lost during tokenisation and I could later on extract more information from the punctuation marks and emojis remaining in the text.

Inspecting some of the misclassified tweets, it seemed that a good way to improve the classifier was to extract the links, follow the trail of redirects and add the title of the landing page to the text of the tweet. This would help give more context to the tweets, since some of them only contain emojis and a simple link, so having an idea of what the link represents would improve the classification accuracy.

The shortcoming of this strategy was the fact that some landing pages were either non-existent anymore (perhaps the link had been taken down since the publishing of the tweet), or would redirect me to a sign in page, so everything that the HTML title would contain would be words like 'Sign Up Today', which are obviously not useful for classification. However, most links were redirected to relevant pages whose titles made sense in the context of the tweet.

Another strategy for classification improvement was hashtag decoding. Using a list of common English words, I tried to break up a hashtag into the composing words. For example, `#iamawesome` would be decomposed into `i am awesome`, and the resulting words would be

added to the text of the tweet. Some hashtags really gave more context to the tweet itself, such as `#womensrights`, while other hashtags contain social media abbreviations that were not matched in the common English words list, such as `#lol`. In these cases, the entire word in the hashtag was added to the text of the tweet.

To improve the classification even more, I applied stemming and stopping to the tokenised tweet. Stemming was performed using the nltk package, and stopping was also performed using the nltk Enligh stopwords list. This would help match different forms of suggestive words, such as `pets` and `pet`.

Another step in the classification improvement was a simple text emoji decoder. I gathered a list of commonly used text emojis, such as `:-)`, `:-o`, `:-(` and replaced every appearance of these emojis with a suggestive word, for example 'happy', 'surprised', 'sad'. This gave more context to the text written in emojis, and, although done in a minimal way, could potentially improve classification accuracies much more, if a larger set of emojis was used.

Finally, I tried to optimise the `-c` parameter of the SVM classifier experimentally, and found that a value of 6000 is ideal for this dataset.

The final accuracy for the improved system is 0.726, which is a 0.054 increase from the baseline system (5% improvement), and a Macro-f1 score of 0.720, which is 0.060 better than the baseline one.

References

1. Multi-class support vector machine. https://www.cs.cornell.edu/people/tj/svm$_light/svm_multiclass.html, accessed : 2019 − 11 − 10
2. Scipy t-test. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest$_ind.html, accessed : 2019 − 11 − 10