1 Introduction

There are two parts, **IR Evaluation** and **Text Classification**, in this coursework. In the first part, I implemented a compact IR evaluation system and ran statistical hypothesis testing on the IR systems. In the second part, I implemented a vectorizer which transforms input tweets into bag-of-words (BOW) representation, then ran a Support Vector Machine (SVM) algorithm to classify the tweets and finally, examined the performance of the classifier based on several scores. Further improvement has been made by manipulating the feature space.

2 Pipeline & Learning Outcomes

2.1 IR Evaluation

The components of the IR Evaluation system (encapsulated in a single class `Eval`) are:

- Parsers to read system results and relevant documents of queries.
- Several kinds of IR evaluation measures that evaluate a system S against a query Q:
  1. **P@N**: Precision at cutoff N, i.e. only consider the top N results (ranked) of S against Q.
  2. **R@N**: Recall at cutoff N.
  3. **r-Precision**: Precision at rank r, where r is the **total number of relevant documents** Q has.
  4. **AP**: Average Precision, defined as: $\textbf{AP} = \frac{1}{r}\sum_{k=1}^{n} \textbf{P@N}(k) \times \delta(k)$, where r is the rank defined before, n is the number of retrieved documents of S against Q, $\delta(k)$ is an indicator function which has value 1 if the document @ k is a relevant to Q, otherwise 0.
  5. **DCG@N**: Discounted Cumulative Gain at cutoff N, defined as: $\textbf{DCG@N} = rel(1) + \sum_{i=2}^{N} \frac{rel(i)}{\log_2(i)}$, where $rel(i)$ gives the relevance grade of the ith document in the ranking list of S.
  6. **iDCG@N**: the ideal **DCG@N**, i.e. the **DCG** value for the perfect ranking.
  7. **nDCG@N**: the normalized **DCG** value, i.e. $\frac{\textbf{DCG@N}}{\textbf{iDCG@N}}$.

  For the reusability of the code, the method `r_precision` invokes `r_at_N` with the input cutoff N replaced by the rank r of Q; the method `nDCG_at_k` invokes `DCG_at_k` and `iDCG_at_k` for obvious reason. However, the method `average_precision` does not invoke `p_at_N` because that will increase the time complexity. Moreover, to take care of the term, $rel(1)$, in the definition of **DCG@N**, my code checks whether or not the index of the first relevant document in the ranking list of S is 1. If so, the corresponding relevance score is added separately, otherwise all operands are summed up together.
- IR evaluation measures that evaluate a system S in general (against all queries), i.e. the mean scores. For each individual measure, the best system is the one that attains the highest mean score.
- Using `pandas.DataFrame`[1] to store the scores (which makes the later `t-test` much easier to conduct) and `Decimal.quantize` [2] to keep the output numbers in three decimal places.
- Statistical hypothesis testing (`t-test`) between the first and the second best systems for each measure.

[Learning Outcomes]: In this task, I learned how to evaluate a system from various angles. Depending on the purpose of the system, we might consider one measure over the other (e.g. for a search task, precision is more important than recall). Finally, running the `t-test` between systems gives me an idea of how to determine whether or not a system is **significantly** better than the other.

[1] Documentation: `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html`
[2] Documentation: `https://docs.python.org/3.6/library/decimal.html`

2.2 Text Classification

The pipeline for Text Classficication task on the tweets dataset is that:

- Preprocess the input tweets, obtain the information of tweet ID, tweet content (cleaned) and the category.
- Vectorize the preprocessed tweet by transforming the content into various features, e.g. the presence of each token.
- Feed the vectorized data into the `SVM Multiclass Classifier`[3] and examine the results.

I built two versions of the vectorizers while left the SVM classifier unchanged, seeking for better performance from manipulating the feature space. Please refer to section 4 for details.

[Learning Outcomes]: In this task, I learned the importance of Feature Selection and Feature Engineering, i.e. to reasonably select and transform features into predictive information. Some intuitive and ad hoc ideas related to the task are proven useful and even with some minor modification, the resulting performance can be improved prominently. Lastly, an overall improvement does not necessarily imply individual improvements., e.g. my final improved system attains a worse score for the 4th class `Entertainment` than the baseline model.

3 Analysis of IR Evaluation

The table below presents the content in the required output file `All.eval`.

	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	0.448	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	0.332	0.424
S6	0.410	0.767	0.448	0.445	0.400	0.491

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

According to the results, the best and the second best systems for each individual score are:

	P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
1st Best	S3, S5, S6	S2	S3, S6	S3	S3	S3
2nd Best	S1	S1	S1	S6	S6	S6

Table 2: The first and second best systems for each score.

Overall, System 3 is likely to be the best as it appears the most frequently in the 1st Best row of Table 2. However, we need to run statistical hypothesis testing, the `t-test`, to study that whether or not the best systems are **significantly** better than the second best.

[T-test]: Assume that the differences between the corresponding scores of the first and the second best systems are sampled from Normal Distribution. Let A be the random variable of the scores of the best system and let B be the random variable of the scores of the second best system, we conduct a two-tailed `t-test` with the following hypotheses:

$$H_0 : \overline{A} = \overline{B} \qquad \text{(Null Hypothesis)}$$
$$H_1 : \overline{A} \neq \overline{B} \qquad \text{(Alternative Hypothesis)}$$

The test statistic is then: $t = \frac{\overline{A-B}}{\sigma_{(A-B)}} \cdot \sqrt{N}$ where $\sigma_{(A-B)}$ is the standard deviation of $A - B$ and N is the number of samples. Without further ado, I report the resulting `p-values` directly in Table 3.

Note that for **P@10** and **r-Precision**, there are multiple best systems, but I investigated the standard deviations of each best system and found that they are identical, meaning that there is no difference to pick any of the

[3] Available at: `https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html`

	P@10	R@50	r-Precision	AP	nDCG@10	nDCG@20
p-values	0.888	0.703	0.759	0.967	0.882	0.869

Table 3: p-values after running t-test for the best and the 2nd best systems of each individual score.

best systems to conduct the t-test.

As illustrated in Table 3, the `p-values` are all much larger than the significance level $\alpha = 0.05$, meaning that there is no evidence to reject the H_0, i.e. we **cannot** determine that the best system are statistically significantly better than the second best system for all different scores. This is not a surprising result because after all, only System 4 seems to yield low scores consistently.

4 Text Classification in Details

Since the classifier is given, I will only go through the details of how I preprocess the tweets, select and engineer the features.

4.1 Baseline Model

[File Loading:] The course organiser mentioned in the Piazza that the tweets data file should be UTF-8 encoded. However, some special characters hinder me from loading the file, so I simply set the parameter `errors='ignore'` such that these characters are omitted. Admittedly, this will lead to data loss, but after I examined the data, such erroneous characters are **rare** and therefore, I believe that removing them will not cause a disruptive bias. The example code is shown below:

```
with open('tweets/Tweets.14cat.train', 'r', encoding='utf-8-sig', errors='ignore') as f:
    lines = f.readlines()
```

[Preprocess]: After loading the data, I conducted preprocessing on the raw data as follows:

1. Split each line of tweet on **tab character** to get the ID, content and category of that tweet.
2. Remove the URL links using the code:

   ```
   def remove_links(text):
       return re.sub(r'http\S+', '', text)
   ```

   One disadvantage of using this simple method to remove links is that, although all links started with "http:" or "https:" will be removed, phrases like "abhttp" will be changed "ab" and thus a loss of information. But again, such kinds of phrases are rare and often meaningless.
3. Tokenize the cleaned tweet using the code:

   ```
   def tokenize(text):
       return re.findall(r'\w+-?\w+', text)
   ```

   "`\w+`" in the above code matches **one of more** Unicode word characters; this includes most characters that can be part of a word in any language, as well as numbers and the underscore. The overall meaning of the regular expression is that only sequences of **two or more** Unicode word characters with a potential hyphen in between are kept as tokens. If we simply use "`\w+`", hyphenated words like "anti-nuclear" will be split into "anti" and "nuclear" such that the negation is lost. Also, since our tweets are mostly English-written, single character is not likely to contribute to the classification, so words of length one are omitted as well.
4. Lowercase all the tokens (which reduces the dimensionality of the feature space).

[Vectorization]:

1. Assign an unique index to each token, obtaining two Python `dict`: `token_to_index` and `index_to_token`.
2. For each tweet, encode the preprocessed tokens into bag-of-words representation using `token_to_index`. The feature space is simply all the tokens in the dictionary.

   Each feature takes a binary value, with 1 indicating the presence and 0 indicating the absence. In fact, we can replace the binary values with the counts of the tokens, but it actually jeopardizes the performance of the classifier a lot.

For unknown tokens, I chose to ignore them because adding a "UNK" token to the vectorizer actually results in worse performance.

The above choices are made based on several experiments I have run.

The size of the resulting feature space is 10249.

[Training]: With the vectorizer built above, we can transform the tweets into vectors and output the transformed training data as required in the coursework specification, so as to feed the training data into the multiclass SVM classifier.

[Evaluation]: Use the trained model to predict the class labels of the test data. For each category, compute the **precision, recall** and **F-1 score**. For the whole system, report the overall **accuracy** and **Macro-F1 score** which is defined as the average of the individual F-1 scores for each class.

	Accuracy	Macro-F1
baseline	0.666	0.652

Table 4: Scores for the baseline model

4.2 Enhanced Model

[File Loading:] Same routine as in 4.1.

[Preprocess]: The general steps still remain the same as in the Baseline Model. However, the tokenization method is modified and besides casefolding, **stopping** and **stemming** are applied as well.

The modified tokenization method is:

```
def tokenize(text):
    # add hashtag content as well
    return re.findall(r'#?\w+', text)
```

Compared with the previous version, the most important change is that the hashtags are preserved because normally in a tweet, the hashtags provide vital information of the content's topic. Keeping the hashtags intact will yields extra features for the classifier to learn. However, this time the hyphenated words are not preserved, and words of length one are not removed. I chose to do so because it actually gives better performance with stopping and stemming applied, but I have no intuitive explanation for that.

The reason to do the stopping is that, words in the stopwords list are functional words that appear very frequently but give trivial information. As for the stemming, it is a way of reducing the dimensionality of the feature space by treating various forms of a word the same. But the potential trade-off is that we may lose the word's contextual meaning.

[Vectorization]: The modifications I made here are the following:

1. Besides the preprocessed tokens, I add the **counts** of the hashtag sign "#" as an additional feature because tweets with more hashtags are more inclined to clarify their topics. The resulting size of the feature space is 9570 (around 7% smaller than in the Baseline Model).
2. Most of the remaining features still take the binary values except for the hashtag tokens, which take values of their **counts plus one**. The additional 1 is an empirical value which attaches higher importance to the hashtag tokens during the training.

```
    ...
    for feat, count in Counter(sorted(feats)).items():
        if '#' in token_to_index[feat]:
            # For hashtag tokens, the value is (count + 1)
            result += str(feat) + ':' + str(count + 1.0) + ' '
        else:
            # For other tokens, the value is binary
            result += str(feat) + ':' + str(1) + ' '
    ...

```

[Training]: Same routine as in 4.1.

[Evaluation]: Same routine as in 4.1.

Finally, I report the overall scores of the above two systems in Table 5.

	Accuracy	Macro-F1
baseline	0.666	0.652
improved	0.720	0.709

Table 5: Scores for the baseline & the enhanced model

Accuracy is increased by 8.1% and **Macro-F1 score** is increased by 8.7%.

5 Conclusion

In this report, I first describe the pipeline and learning outcomes for each of the two tasks. Then, I write an analysis of the IR Evaluation system with the help of t-test. Finally, I specify the details of the text classification models and the key differences between the baseline and the improved one.