Language Identification - COMP472 Project 2 Report

Duy-Khoi Le

40026393 alvyn279@gmail.com

1 Introduction

This report describes the implementation of models for language identification, and analyzes their performances against a testing dataset. This report assumes that the reader is familiar with the problem statement.

1.1 Implemented solutions

In this project, four concrete models were designed in order to identify the language of a tweet. A character-based n-gram approach to the Naive Bayes Classifier (with unigrams, bigrams, and trigrams) was implemented as part of the mandatory models. My own model was a *TF-IDF with stop-words* approach. This algorithm will be discussed further in details in this report.

1.2 Technical Details

The solutions were implemented in *Python3*. For my own model, I make use of the popular *nltk* Python library [1]. The script must therefore be ran inside a virtual environment containing the dependencies listed in the project's defined requirements. Please refer to the repository's README.md for additional information.

2 Datasets Analysis

During the demo, a flaw in my implementation was revealed. The testing dataset that was given to me at that time was not formed as expected. My implementation had made the assumption that all the five languages would be present at least once in that set. Namely, the Basque (eu) language had no tweets within the given test dataset, as seen in Figure 1. This actually broke my code at the time of computing probabilities and statistics, and required me to apply a hot fix.

Apart from this, I was handling all vocabulary-related incertitude in all my models. For the n-gram-based models, I was checking each scanned character from the test input against the built-in *isalpha* function. In my own model, I made sure that the scanned word contained only alphanumerical characters

Language	Training Test	Given Test	Demo Test
eu	374	380	0
ca	1493	75	1391
gl	456	1	506
es	12855	3973	4589
en	971	516	483
pt	2169	2055	96

Table 1: Tweet language frequency in input files

before adding it to model's corpus. These procedures made my implementation robust for any well-formed testing dataset.

One major issue was that the Naive-Bayes models were trained against data with uneven tweet language frequency, as seen in Figure 1. For example, one could expect the n-gram-based model to have a substantially larger Spanish corpus, as opposed to the Galician corpus. In other words, the Spanish corpus would have an increased probability to catch relevant character n-grams. There is therefore a *normalization* issue, a problem that is addressed in my own model.

3 Build Your Own Model

I call my model the ***TFIDF with stop words*** model. During the training phase, the model's corpus inherently replicates the bag of words model, but with a twist. The corpus is an occurrence dictionary of words, but with additional occurrence counts added when the parser encounters a stop word from that particular language.

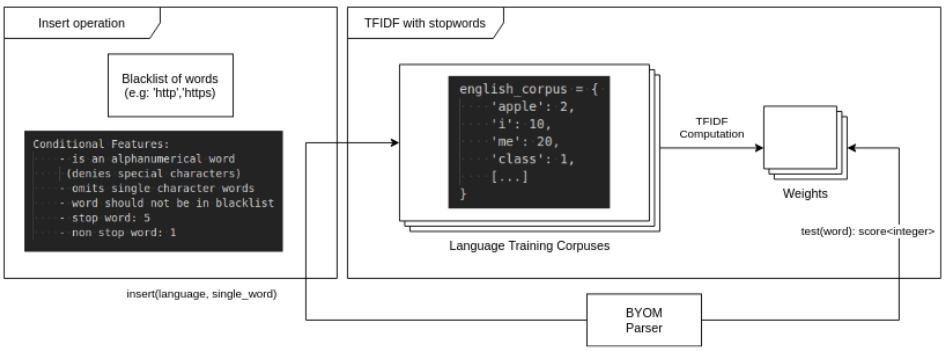

Fig. 1: Summary of operations for ***TFIDF with stop words*** model

The stop words of a language are defined as words of that language that are commonly used [3]. In the scenario of identifying the language of a tweet, finding a stop word inside the tweet gives a strong indication of its potential language. While training the model, we therefore add five occurrences (arbitrarily chosen) of a word to the training corpus if that word is a stop word, and only one occurrence of any non-stop word. This can be observed in Figure 1. In the context of this model, the occurrence values of stop/non-stop words are the only hyper-parameters that are configurable. Giving more than 5 as an occurrence value for a stop word had no more impact than it already did on the resulting metrics.

Ultimately, each corpus will end up being a bag of words with enhanced occurrence count. The motivation behind doing so is to obtain a corpus that is representative of the vocabulary and jargon used by people to write tweets (i.e: the training dataset) in each respective language.

However, even with these stop words, there still remains the problem of uneven number of tweets per language in the training dataset. This is where term frequency-inverse document frequency (TF-IDF) statistical computation comes into play. The latter computes a weight by giving more value to a frequent word in a document (TF) while downscaling the value if there are many occurrences of that word across the rest of the documents [4]. The document in this context is the trained language. The formula used to calculate this weight value [4] was the following:

$$weight = (1 + log(f)) \times log(N/n) \tag{1}$$

where f is the word frequency in the language, N the number of languages studied, and n the number of occurrences for the word across all the rest of the languages. As we can see, there is normalization applied for both for the word's frequency and the inverse language frequency through a logarithmic scale. At test time, this weight is summed for every word within a tweet, hence giving the final score for the language analyzed. The results of my model will be be further analyzed in the next section.

4 Results Analysis

From the disparity in tweet language frequency seen in datasets of Table 1, it was expected that the lack of normalization would lead to performance issues for the Naive-Bayes model with n-grams feature, as explained in Section 2. Ideally, the *TF-IDF model with stop words* should perform better since it tackles this normalization issue. In this section, we explore more in details the performance metrics. **For space and readability concerns, the analysis constantly refers to evaluation outputs gathered in Appendices A & B**.

During analysis, one might be tempted to give more importance to a more popular and understandable metric, such as accuracy. However, in the case of uneven

class distribution (appearing in our Naive-Bayes models), it might be more helpful to give more value to F1 scores, as they take into consideration false positives and false negatives [5].

It is to note that for outputs with the initial given test set, the Galician language has many statistical metrics of value 0. This is because there is only one Galician tweet in the test set, as seen in Table 1.

4.1 Overall Vocabulary-Based Performance

When looking at the performance of Naive-Bayes based on the type of vocabulary in Appendix A & B, we can hypothesize that having a restricted vocabulary (V=1 instead of V=2, for example) can lead to better overall performance. An average macro-F1 measure of 0.56 for V=1 beats average macro-F1 of 0.36 for three runs with V=2 in the initial test set. This maybe explained by the common use of alphabet letters (A-Z, a-z) in all the test tweets, or even the omission of the probability when a non-existing character is scanned. In other words, we are restraining the computation of probabilities to the existing corpus, which can only be characters of the English alphabet.

4.2 Overall Smoothing-Value-Based Performance

The smoothing value is a very small hyper-parameter used to avoid the probability of zero for non-occurring n-grams in the training dataset [2]. We do not expect it to yield massive differences in performance. This is in fact the behaviour observed in the evaluation outputs. For V=2, $\delta 1$=0.1, $\delta 2$=0.5, and $\delta 3$=0.8, the difference in accuracy and weighed-average-F1 goes almost unnoticeable in the case of unigrams and bigrams. However, the same cannot be said about trigrams, where a lower value of delta leads to higher accuracy and macro-F1. For example, for the demo test set, the macro-F1 measure starts at 0.5242 ($\delta 1$=0.1) before falling to 0.4927 ($\delta 2$=0.5) and 0.4738 ($\delta 3$=0.8). The same linear descent is observable for the accuracy. This behaviour is expected, as the corpus of a trigram-based model is large and contains many non-occurring trigrams. Hence, having a larger smoothing value would slightly increase the probability of those non-occurring trigrams. At test time, rarer trigrams will be using this larger probability, ultimately leading to unwanted larger scores for some models.

4.3 Overall n-gram Size-Based Performance

From Table 2, out of the 8 combinations of vocabulary and delta values seen in Appendices A & B, 7 combinations see the bigram-based model having the highest accuracy. However in the case of missing normalization in the training dataset, it might be wise to use trigrams, as they yield the largest macro F1-measures (which abstracts the size of the language).

Large values for the metrics listed in Table 2 are all labelled as desirable; they are all linearly derived from ratios having as numerator the number of correctly

Desirable Characteristic	Unigrams	Bigrams	Trigrams
Highest accuracy	0	7	1
Highest macro-F1	0	0	8
Highest weighed-average-F1	0	6	2

Table 2: Metrics performance by n-gram for the eight Naive-Bayes setups in Appendix A & B

predicted positive predictions [5]. All in all, one could definitely suggest the use of bigrams or trigrams over the use of unigrams.

4.4 Lowest Identification Success for a Language

From all the evaluation outputs, we can pinpoint the language that was the hardest to identify. From what is seen, there is one language that consistently carries the lowest per-class precision, recall, and F1-measure when using the Naive-Bayes model: Catalan (ca). This is easily observable for V=2 and δ=0.5 in both test sets, as the maximum precision for this language is 0.055 while that of the F1-measure is 0.105. From the trace files, it seems as though Catalan tweets are mistaken a lot for Spanish tweets by the model. This makes sense, as both those languages are used in Spain. However, the F1-measures for the Catalan language using the *TF-IDF with stop words model* are 0.8 and 0.23 for both test sets. This is an improvement from the Naive-Bayes model. I suspect the unique Catalan stop words combined with inverse frequency being a huge factor in improving the Catalan score for a tweet.

4.5 Classification of Test Results

The metrics gathered in the evaluation files are based on the confusion matrices generated from testing data. In reality, there would be an associated confusion matrix for each language of each model. To summarize the results, the data was summed up into sample cumulative tables (Table 3 & Table 4).

	Predicted Positive	Predicted Negative
Actual Positive	4770	2230
Actual Negative	2230	32770

Table 3: Cumulative confusion matrix for Naive-Bayes with bigrams, with hyper-parameters V=2, δ=0.5 on initial test dataset

	Predicted Positive	Predicted Negative
Actual Positive	6582	418
Actual Negative	418	29650

Table 4: Cumulative confusion matrix for *TF-IDF with stop words* for initial test dataset

Table 3, depicting Naive Bayes with bigrams, shows that there was a non-negligible amount of false positives and false negatives. These imply that were many cases where the top score was not that of the actual language. On the other hand, this scenario seems to be more limited for the *TF-IDF with stop words model* (Table 4). As a matter of fact, the majority of cases were found to be true positives (6582) and true negatives (29650).

4.6 Accuracy and Macro-F1

With both test datasets, the Naive-Bayes model does never seem to achieve a consistent accuracy in language identification. For 24 executions, the accuracy oscillates between 57% and 81%. My own model, on the other hand, seems to outperform Naive-Bayes by a noticeable amount. For the best hyper-parameters described in the previous section, my model achieves 94.0% and 88.3% accuracy for initial and demo test sets respectively. The difference in accuracy between models can easily be observed by comparing Tables 3 & 4.
For the macro-F1 measure, my model yields maximums for both test sets: 0.7344 and 0.7317 for initial and demo respectively. The closest macro-F1 measure by a Naives-Bayes model seen is 0.6371 (with trigrams). This goes to show the importance of normalization of number of tweets per language in the training data.

4.7 Conclusions

Based on the metrics gathered for each model, the *TF-IDF with stop words* model objectively outperforms most if not all the Naive-Bayes models because it handles the much needed normalization of the training data.

5 Work Distribution

As I worked on this project alone, I own all the design, implementation, and research involved in the completion of this project. However, additional stop words (other than those from nltk) were used from an existing reference [6] (please refer to the repository's README).

6 Appendices

6.1 Appendix A - Evaluation outputs for initial test set

Evaluation output for hyper-parameters V=1, δ=0.5

```
# unigram
0.6978571428571428
0.7241  0.2170  0.0  0.7207  0.4618  0.7856
0.6631  0.3733  0.0  0.8401  0.5872  0.4690
0.6923  0.2745  0.0  0.7759  0.5170  0.5874
0.47454119984863635 0.6914852091056984

# bigram
0.8135714285714286
0.6322  0.2313  0.0  0.9022  0.7572  0.9169
0.8868  0.7866  0.0  0.8243  0.8643  0.7678
0.7382  0.3575  0.0  0.8615  0.8072  0.8358
0.6000580549644812 0.8377438166471004

# trigram
0.7268571428571429
0.5110  0.3971  0.0  0.9722  0.902 0.9548
0.9763  0.7466  0.0  0.6342  0.8740  0.8223
0.6708  0.5185  0.0  0.7677  0.8877  0.8836
0.6214279234603598 0.8025645121344722
```

Evaluation output for hyper-parameters V=2, δ=0.1

```
# unigram
0.5754285714285714
0.0  0.0360  0.0  0.5820  0.3043  0.8308
0.0  0.0933  0.0  0.9546  0.0135  0.1075
0.0  0.0520  0.0  0.7231  0.0259  0.1904
0.1652697853116172 0.46882634490942676

# bigram
0.6945714285714286
1.0  0.0579  0.0  0.7873  0.8638  0.8489
0.03421 0.84  0.0  0.7213  0.6395  0.7737
0.0661  0.1083  0.0  0.7529  0.7349  0.8095
0.41199329761952996 0.7239344208456544

# trigram
0.7104285714285714
0.8897  0.0676  0.0  0.9057  0.8010  0.7759
0.5736  0.96  0.0  0.5924  0.8817  0.9119
0.6975  0.1264  0.0  0.7163  0.8394  0.8384
0.5363935535775495 0.7538521655965988
```

Evaluation output for hyper-parameters V=2, δ=0.5

```
# unigram
0.5748571428571428
0.0   0.0360   0.0   0.5816   0.3043   0.8282
0.0   0.0933   0.0   0.9546   0.0135   0.1055
0.0   0.0520   0.0   0.7228   0.0259   0.1873
0.1647031806577115 0.4677527834816184

# bigram
0.6814285714285714
1.0   0.0537   0.0   0.7892   0.8285   0.8371
0.0368   0.84  0.0   0.6899   0.6744   0.7805
0.0710   0.1009   0.0   0.7362   0.7435   0.8078
0.4099514090993163 0.7147820161535855

# trigram
0.6447142857142857
0.8415   0.0557   0.0   0.9134   0.7757   0.7084
0.6289   0.9466   0.0   0.4651   0.8914   0.9221
0.7198   0.1053   0.0   0.6164   0.8295   0.8012
0.5120793997838593 0.6864462227542479
```

Evaluation output for hyper-parameters V=2, δ=0.8

```
# unigram
0.5747142857142857
0.0   0.0360   0.0   0.5815   0.3043   0.8275
0.0   0.0933   0.0   0.9546   0.0135   0.1051
0.0   0.0520   0.0   0.7228   0.0259   0.1865
0.16456125395183333 0.46748391677209583

# bigram
0.676
1.0   0.0521   0.0   0.7902   0.8228   0.8307
0.0394   0.84  0.0   0.6760   0.6841   0.7858
0.0759   0.0981   0.0   0.7287   0.7470   0.8077
0.4095958803472741 0.7109548944163072

# trigram
0.6175714285714285
0.8145   0.0539   0.0   0.9158   0.7714   0.6767
0.6473   0.9466   0.0   0.4137   0.8895   0.9260
0.7214   0.1020   0.0   0.5700   0.8262   0.7820
0.5003029876036477 0.6542772773326113
```

Evaluation output for *TF-IDF with stop words*

```
0.9402857142857143
0.6988   0.8333   0.0   0.9413   0.9824   0.9932
```

```
0.8184  0.8  0.0  0.9697  0.8662  0.9299
0.7539  0.8163  0.0  0.9553  0.9207  0.9605
0.7344795492871204 0.9605428499623019
```

6.2 Appendix B - Evaluation outputs for demo test set

Evaluation output for hyper-parameters V=1, δ=0.5

```
# unigram
0.6615711252653927
0.8177  0.5714  0.7492  0.4260  0.0674
0.3580  0.0395  0.8293  0.6438  0.4062
0.498 0.0739  0.7872  0.5127  0.1157
0.3975  0.65132908420527

# bigram
0.7787685774946922
0.8387  0.3828  0.8999  0.6688  0.1908
0.7742  0.4584  0.8097  0.8405  0.7395
0.8052  0.4172  0.8524  0.7449  0.3034
0.6246723733549239 0.7972017315645574


# trigram
0.6736022646850672
0.9277  0.1896  0.9583  0.8062  0.3459
0.7943  0.7608  0.6121  0.8012  0.7604
0.8559  0.3036  0.7470  0.8037  0.4755
0.6371872435414552 0.7369303998921263
```

Evaluation output for hyper-parameters V=2, δ=0.1

```
# unigram
0.6260438782731776
0.2892  0.0  0.6522  0.25  0.2191
0.0833  0.0  0.9339  0.0103  0.1666
0.1294  0.0  0.7681  0.0198  0.1893
0.22135889445380513 0.5283336541698829

# bigram
0.6767162066525124
0.5072  0.4  0.8442  0.7752  0.1425
0.8526  0.0039  0.7064  0.5714  0.7812
0.6360  0.0078  0.7692  0.6579  0.2411
0.4624511873225732 0.6737130712447871

# trigram
0.6520877565463553
```

```
0.5494  0.3523  0.9063  0.7461  0.1025
0.9503  0.1956  0.5905  0.8033  0.9166
0.6963  0.2515  0.7151  0.7736  0.1844
0.5242453568330739 0.6750260064218634
```

Evaluation output for hyper-parameters V=2, δ=0.5

```
# unigram
0.6259023354564756
0.2878  0.0  0.6524  0.2380  0.2191
0.0833  0.0  0.9337  0.0103  0.1666
0.1293  0.0  0.7681  0.0198  0.1893
0.22132759411008768 0.5283202566689633

# bigram
0.6571832979476292
0.4879  0.25  0.8481  0.7603  0.1291
0.8612  0.0019  0.6694  0.6107  0.8020
0.6229  0.0039  0.7482  0.6773  0.2225
0.4550193365998664 0.6582989462964379

# trigram
0.5845718329794763
0.5164  0.3074  0.9196  0.7040  0.0785
0.9583  0.2272  0.4787  0.8178  0.9375
0.6711  0.2613  0.6296  0.7567  0.1449
0.4927  0.613581197734178
```

Evaluation output for hyper-parameters V=2, δ=0.8

```
# unigram
0.6259023354564756
0.2878  0.0  0.6524  0.2380  0.2191
0.0833  0.0  0.9337  0.0103  0.1666
0.1293  0.0  0.7681  0.0198  0.1893
0.22132759411008768 0.5283202566689633

# bigram
0.6461429582448691
0.4768  0.4  0.8496  0.7361  0.1243
0.8655  0.0039  0.6513  0.6066  0.8020
0.6149  0.0078  0.7373  0.6651  0.2153
0.44813325173121205 0.6489910829616988

# trigram
0.5515923566878981
0.5020  0.2687  0.9179  0.6928  0.0713
0.9568  0.2193  0.4288  0.8219  0.9375
0.6585  0.2415  0.5845  0.7518  0.1325
```

```
0.4738353957329374 0.5798820904489562
```

Evaluation output for *TF-IDF with stop words*

```
0.883934890304317
0.9860   0.7701   0.8825   0.8957   0.5845
0.9151   0.1324   0.9627   0.8364   0.8645
0.9492   0.2259   0.9208   0.8650   0.6974
0.7317465741634247 0.6974789915966387
```

References

1. Natural Language Toolkit, https://www.nltk.org/. Last accessed 13 Apr 2020
2. Leila Kosseim, Russell S., Norvig P.: Annotated class slides - 472-5-NLP-Winter2020-annotated, [Class Moodle]. Last accessed 13 Apr 2020
3. Stop words, https://en.wikipedia.org/wiki/Stop_words. Last accessed 13 Apr 2020
4. Gebre, B.G., Zampieri, Wittenburg, P. Heskes, T.: Improving Native Language Identification with TF-IDF Weighting. In: Max Planck Institute for Psycholinguistics, University of Cologne, Rahboud University, https://pdfs.semanticscholar.org/c685/35ad8bc233c6a923c3997cfd338bba55561a.pdf
5. Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures, https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. Last accessed 17 Apr 2020
6. Xangis/extra-stopwords, Extra stopword lists for use with NLTK. https://github.com/Xangis/extra-stopwords. Last accessed 17 Apr 2020