An Investigation of Multiple Language Classifiers COMP 551 - Applied Machine Learning

Thomas Page
Department of Mechanical Engineering
McGill University
Montreal, Canada
thomas.page@mail.mcgill.ca
260771672

David Tamrazov
Computer Science, Arts
McGill University
Montreal, Canada
david.tamrazov@mail.mcgill.ca
260561439

Alexander Wong
Cognitive Science, Arts and Science
McGill University
Montreal, Canada
alexander.wong4@mail.mcgill.ca
260602944

Notorious Language Classifiers
https://github.com/a22wong/notorious-language-classifier

October 23, 2017

I. Introduction

As a result of the first project, corpora in various languages are available to act as training sets for many language related machine learning tasks. The goal of this project is to create classifiers that can determine the language of a sample of text from corpora information developed in the first project. Based on a review of the methods discussed in COMP 551, we construct four classifiers: Naive Bayes, Linear Discriminant Analysis (LDA), Decision Trees, and Extremely Randomized Decision Trees.

II. Related Work

Naive Bayes was chosen as the most promising text classification method for this application based on a previous study in this area. In his master?s thesis, Jason Rennie used a Naive Bayes classifier to classify a given news article into 1 of 20 different categories (e.g. Electronics, Religion, Space, etc.). Training on 1000 documents from each category, the Naive Bayes classifier was able to classify the new articles very well [**?**]. The main difference between that task and this project is that rather than distinguishing between categories, we will be distinguishing between languages, but the same concept applies.

III. Problem Representation

The compiled training dataset contains 276516 labeled language samples from five different languages: Slovak, French, Spanish, German, and Polish. The test set contains 118507 unlabeled samples of zero to ten individual characters. This presented the challenge of classifying seemingly nonsensical data; a common task in practical machine learning. Fig. **??** represents the weight of each language in the training set.

We found that the data required little pre-processing as non-discriminative symbols such as punctuation had already been removed. To process the text, we chose to simply make all characters lowercase as to eliminate the classifier distinguishing between upper- and lower-case symbols. There are many blank lines in the training set, but we found eliminating them to have no effect on our outcome. Nothing is being assigned to a label in these blank lines, and therefore it does not add any weight to the prediction. We also looked into removing numbers from our training set as these have no attachment to language. However, as the corpora are taken from different sources, we decided that some sources may be more likely to include numbers in the dialogue and that we could exploit this distinction.

A. Naive Bayes

As mentioned, the test set is a string of characters. Because of this, we chose to train our classifiers on the probability of characters appearing rather than the probability of words or small groups of words appearing. We lost a lot of the natural language data by making our features individual characters, but this seemed to be the only reasonable solution for this unique problem.

B. Decision Tree

With a 'unexpected' test set of only single characters, we decided to try using a Decision Tree to discriminate test data based on what special characters appeared, and default to a Naive Bayes approach when no special characters appeared. Initial data pre-processing required included identifying all special characters, which we did so by regular expression matching any non-ASCII character. We found there to be 613 non-ASCII characters present in the training set; later this will shown to be too large to use for classification in limited processing time.

Next was to classify the special characters to the language they were unique to. Special characters that mapped to more than one language were not used in the Decision Tree evaluation criteria. However, due to failure to properly implement a program that properly identified special characters, we were unable to implement a Decision Tree algorithm

The initial set of 613 special characters was far too large, reporting over 200 special characters unique to French alone. Attempting to implement the Decision Tree with this set of characters has a time complexity of `O(shcn)` where `s` is the number of special characters unique to the language being

filtered at level `h` of the Decision Tree, for each character `c` in each test case `n`. On a commercial computer, it was making 3 predictions a second, which would have required over a full day of computation.

We tried to streamline the data pre-processing to find a better method without sucess. Specifically, the encountered issue was with unicode encoding, and we suspect specifically with 8 vs 16 bit encoding for special characters that, in the given dataset, actually are two chars in line; for example, the character `à` is given by the unicode sequence `0xc3` and `0xa0`.

In designing the Decision Tree, the goal was to have each node's evaluation criteria make the largest cut in the test set - in other words create the most separation - to effectively filter the test data. So, assuming the training and test sets would have similar distributions of classes, we structured the tree to evaluate most likely languages first: French, Spanish, German, Polish, then Slovak.

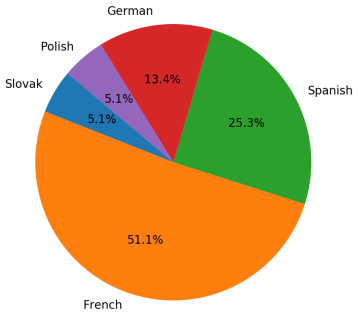

Fig. 1. Training set language distribution.

IV. Testing and Validation

The most effective way to represent the performance metrics of each individual classifier is through their respective confusion matrix. We chose to evaluate our classifiers using a normalized confusion matrix as it highlights important outputs like the true positives and exactly how each class is correctly or incorrectly classified.

A. Naive Bayes

Fig. **??** shows the confusion matrix for the Naive Bayes classifier on the training set. Slovak, French, German, and Polish are all classified 90% correctly. Spanish is the only classifier with a significant difference at 77% correct. It is clear from the matrix that most of the incorrect Spanish classifications are classified at French. This is due in part to both languages sharing many of the same symbols, and because French accounts for over double the training set samples, a Spanish sample is more likely to be classified as French than vice versa. This hypothesis is supported by the confusion matrix, as 16% of Spanish samples are classified as French, but only 7% of French samples are classified as Spanish.

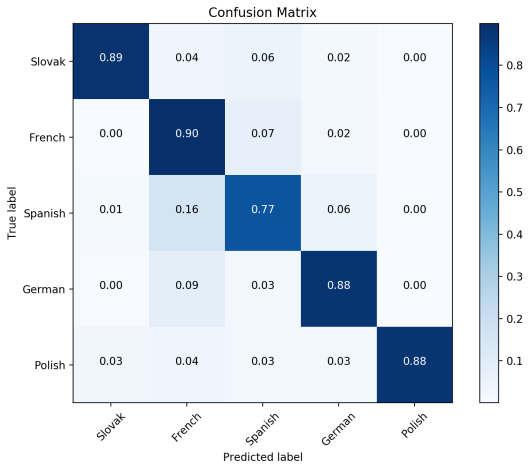

Fig. 2. Naive Bayes confusion matrix.

B. Decision Tree

The expected outcome was that of higher accuracy than Naive Bayes, as it essentially filters out test cases with special characters unique to their respective language. However, due to the special character decoding problem mentioned prior, we were unable to fully test the potential of this method.

For proof of concept, we did implement an incomplete method that lacked sufficient pre-processing of special characters, but out of the 118507 test cases, the Decision Tree only made a decision 148 times - insignificant. Thus, the test accuracy and consequently the confusion matrix for the Decision Tree implementation is the same as that of Naive Bayes in Fig. **??** above.

C. Extremely Randomized Trees

Given that the corpus was uncultured and drew from 5 different languages, we expected variance to be high in the data set. With that in mind we sought to implement a classifier that would trade off variance for a heavier bias, and found Extremely Randomized Trees to be a good fit in that regard. The classifier program made heavy use of the sklearn Python libraries. When read in, the each line of the prediction data is stripped of substrings that pattern match on emojis, pictographs, and other symbols that would be common across all languages, to further reduce variance in the data set. In such a way we read train-set-x.csv, train-set-y.csv, and test-set-x.csv into 3 separate lists, denoted ?training-X?, ?training-labels?, and ?test-X? respectively.

An instance of sklearn?s CountVectorizer() is used to extract the features from the data. The CountVectorizer()?s ?analyzer? option is set to ?char? to extract a vocabulary from individual characters, rather than individual words, to better mirror the test data.

The training data is then passed to CountVectorizer().fit(), where CountVectorizer() extracts a vocabulary from all the documents and stores it. Passing training data and test data (not the training-labels) to the CountVectorizer.transform() will

then return vectorized features of the training data, the training labels, and the test data. Training itself is done through 3-fold cross validation, during which we determine the optimum hyper parameter setting for our learner. We searched for an optimal number of estimators to use for the extremely randomized tree?s classifier. We tested [20, 60, 100, 140, 180] to get a good idea of how accuracy and overfitting scale with the number of estimators.

The confusion matrix for the Extremely Randomized Trees classifier can be seen below. All in all the training-data created a vocabulary of 4103 words.

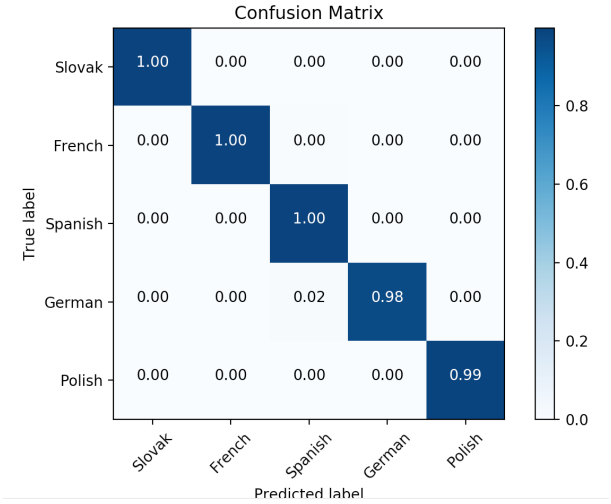

Fig. 3. Extremely Randomized Trees Confusion Matrix.

The data appears to almost uniformly be plit between the five classes. Given that I know this to not be true, one estimate could be that because the vocabulary is built around single-letter words to suit the test data, so there is only so much variance between languages that share the alphabet.

V. Discussion

A. Naive Bayes

Several attempts were made at achieving a higher score using the Naive Bayes classifier. The lower true positive rate of the Spanish classifications were of concern. To remedy this, a corpus of 100,000 Spanish words was added to the training set. This did improve the Spanish rate, but lowered the French rate and overall accuracy of the classifier. In another attempt, a French corpus was added in conjunction with the Spanish corpus, but this led to a very similar result to the initial classifier. The log-likelihood was also calculated in an attempt to improve the classification score, but again there was little difference in the actual output at the expense of a much more complex calculation. Ultimately, the basic Naive Bayes seemed to perform best. Future work in this area will involve testing a Naive Bayes classifier available in a library like sklearn to see how the results compare to the one developed from scratch here.

B. Decision Tree

Dealing with the encoding issues revealed many possible hurdles in pre-processing the data, and for evaluating the Decision Tree algorithm. If we had been able to properly decode the special characters, we might have been able to do better analytics of the special characters to create a more-optimal tree structure, rather than the reality of implementing a pre-Naive Bayes filter.

Regardless of the failed implementation, this filter-by-special-character approach has inherent weaknesses. Because the planned implementation was to immediately filter a test case if it contained a character that mapped uniquely to one class, the Decision Tree would be extremely susceptible to errors in test set. Redundancies could have been included, such as by predicting the `argmax` of classes of special characters appearing in a test case. Also, though special characters that mapped to more than one language were not used in the Decision Tree evaluation criteria, they could be utilized in a more robust, probabilistic method that used the Bayesian probabilities calculated from special character sets alone, instead of just the training set.

Ultimately, even if the Decision Tree method was more effective, it is in practice a data pre-processing step that augments Naive Bayes, or which ever classification algorithm that would be used for the base case.

C. Extremely Randomized Trees

The cross validation returned the optimum number of estimators to be 180 out of what was provided. This could indicate a preference for more indicators to overfit the data with when variance is high as in a situation like this.

An optimizer with 180 estimators was able to gain a training accuracy of 97

```
The best estimator was:
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
           max_depth=None, max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=180, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
With accuracy: 0.966522853929
```

Fig. 4. Report of the best estimator from GridCV crossvalidation

Even though steps were taken to reduce the common vocabulary between languages, like removing all emoticons in pre processing, the classifier still lacked enough bias to make strongly accurate predictions in a generalized setting.

VI. Statement of Contributions

Thomas was responsible for fully implementing the Naive Bayes classifier. He also wrote the Introduction, Related Work, and Problem Representation sections of the report as well as any sections pertaining to Naive Bayes.

Alex was responsible for fully implementing the Decision Trees classifier. For the report, he wrote any sections pertaining to Decision Trees.

David was responsible for the Extremely Randomized Trees classifiers. For the report, he wrote any sections pertaining to LDA or Extremely Randomized Trees.

We hereby state that all the work presented in this report is that of the authors.

REFERENCES

[1] J. D. M. Rennie, "Improving Multi-class Text Classification with Naive Bayes," M.S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2001.