Comp-551: Applied Machine Learning
Project 2: Language Classification
Team: NAS

Alan An
McGill University
Montreal, Canada
Email: chengyun.an@mail.mcgill.ca
Student number: 260683828

Salman Memon
McGill University
Montreal, Canada
Email: salman.memon@mail.mcgill.ca
Student number: 260726294

Niklas Brake
McGill University
Montreal, Canada
Email: niklas.brake@mail.mcgill.ca
Student number: 260602499

***Abstract*—This project entailed language classification for text using Machine learning. Preliminary steps included data pre-processing and feature extraction, where Bag of Words, TF-IDF and N-grams were features considered to represent the data. This was followed by coding classification algorithms for Naive Bayes and Random Forests and using Python's Scikit-learn [1] library to implement Logistic Regression and a feed-forward neural network. The results were quite promising with the neural nework producing the best results, predicting 80.1% of the test data label accurately.**

I. Introduction

Language classification is an important task given the multilingual nature of the internet. In this project, we attempted to identify the language of docuemnts (or utterances) in a corpus derived from Project 1 using 4 different machine learning algorithms.

The project was broken into multiple steps. For the initial pre-processing and feature extraction, we used Logistic Regression from the Scikit-learn [1] to benchmark the best dataset representation. This was followed by coding Naive Bayes as our linear classifier and Random Forests as our non-linear classifier. Finally, a feed forward neural network was implemented to maximize the accuracy on the test set.

II. Related Work

A variety of work highlights the different approaches to feature selection and design as well as the use of different classifiers for text language classification.

[6] shows the use of trigrams and small words for language classification. The triagram apprach uses sequences of three characters as features to distinguish between languages. The author's approach was to calculate the probaility of a trigram in a given language. Test set sentences were then tokenized, converted to trigrams and the probabilities obtained in used to predict their language. A very similar approach was used for small words which are also highly representative of a language. This work used the ECI Language corpus which consists of millions of words in English, French, Spanish and German amongst others.

[7] is a more relevant approach to ours, and uses the DSL Corpus Collection data for training and evaluation. The data uses a more diverse set of languages. Moreover, the authors also tested their classifiers on excerpts from newspaper websites in different languages. Information gain was used to idenitify the most relevant features from word n-grams and character n-grams. More importantly, their classifiers included Logistic Regression, Naive Bayes and SVM, two of which we have used in this project and the last covered extensively in class.

Lastly, [9] showed the use of TF-IDF for the task of Native Language Identification. It utilized the TOEFL11 Dataset that consists of essays of non-native English speakers. Moreover, this work also utilized a variety of features in parallel including TF-IDF, word n-grams, character n-grams and spelling errors. The choice of classifiers included SVM, logistic regression and perceptrons.

These works alongside the guidelines given in class gave us the base for our feature design and classifier choice.

III. Problem Representation

The foremost part of the project was pre-processing the data and extracting features. This proved to be the most important influencer on the results obtained from the classification algorithms.

A. Pre-Processing

The following steps were then executed to clean up the data:

- Consolidating the documents and their labels into a single Pandas DataFrame.
- Removing all the blank documents from the DataFrame. Since the same blanks were associated with multiple labels, they had the potential to mislead a classifier.
- Encoding all the documents in Utf - 8 (Unicode Transformation Object, 8 bit) to account for special characters
- Removing punctuation marks from the documents
- Removing all spaces to create a string of just characters
- Converting all the uppercase characters to lowercase.

In addition to cleaning up the data, we also chose to create random documents from existing documents to better match the test set. We chose new string to be 20 characters (which is the average length of a test set document) and extracted multiple random strings from longer strings. This not only reflected the test data better but also increased the size of our training data. As shown in the results, this apprach yielded the highest test accuracy and helped our classifier generalize better.
Lastly, for the Naive Bayes classifier, we used a Gaussian feature mapping for one of the trials.

B. Feature selection and design

Having cleaned the data, the next step was to represent it using suitable a feature set. The following options were considered:

- Bag of characters
  This popular method of feature representation uses the frequency of each character in a document as a feature. The order of characters is ignored as well as their frequency of occurrence in the entire corpus.

- N-grams
  N-grams use a continuous sequence of N items (characters for our case) and uses their frequency of occurance as a feature. This feature makes the important distinction of taking into the account the sequence in which characters are being used in words and would have proved an invaluable feature for our classifier.
  However, since the test set consisted of randomly picked characters from documents, order was no longer a discerning feature for the dataset rendering n-grams uselss for our.

- TF-IDF
  TF-IDF builds on the bag-of-characters representation by taking the frequency on occurrence in the whole corpus into account as well. The Term Frequency term is given by:

$$\text{TF} = \frac{\text{Number of times the character appears in the document}}{\text{Total number of characters in the document}}$$

  The Inverse Document Frequency Term (IDF) introduced the following term:

$$\text{IDF(t)} = \log \frac{\text{Total number of documents}}{\text{Total number of documents containing character 't'}}$$

  The final feature is given by:

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

  This representation takes into account the frequency of the character within the document while also giving larger weights to the rarer tokens in the corpus. This was the primary feature representation used throughout the project

The final matrix created by our feature extraction procedure had 549 columns, corresponding to 549 individual characters used throughout the training dataset.
An important consideration was how many of these characters in the training process. An analysis showed that accuracy of our training and test sets for the logistic regression classifier levelled off after restricting our matrix to the top 120 characters. After this insight, we restricted training and test sets to 200 columns to reduce computation time and memory usage.

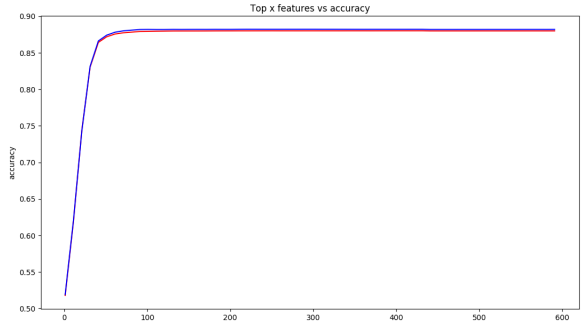

Fig. 1. Training error (blue) and validation error (red) vs the number of top characters used for training the logistic regression classifier

IV. Classifiers

We used four different classifiers through the course of this project. Logistic regression from the Scikit-learn library was used extensively to benchmark design decisions for the feature design phase. Next, we implemented Naive Bayes and Random Forests from scratch as per the project requirements. Finally, we used a feedforward Neural network to maximize the accuracy on the test set.

To benchmark all the classifiers, the data was divided into a 75/25 training validation split.

A. Naive Bayes

Naive Bayes is a popular choice for text classification. This is mainly due to its ease of implementaiton and training and high accuracy in text classification applications [2].

Our implementation of Naive Bayes was run over all the combination of feature sets and preprocessing methods due to the Bayesian assumption setting it apart from Logistic regression.

To train the Naive Bayes classifier, the conditional probability of a character given the language is calculated over all the documents for that language. As a smoothing step, we add 1 to the count of character in the numerator and count of total individual characters (alphabet for the corpus) in the denomintor. We ensured that the sum of the conditional probability equaled 1 to keep the methodology consistent. The log odds for calculated to account for very low values of probailities. For a given language 1 (French), we calculated:

$$\log \Pr(y{=}1 |\ x_j) \propto \log \Pr(y = 1) + \log \prod_{j=1}^{m} \Pr(x_j | y = 1)$$

To predict the language of a test document, we split it into a character array and predict the probability of the combination of utterances given each language. As an extra step, we calculated the log of each probability to avoid underflow and summed them in to calculate the score. The language with the highest score (or probability) was the prediction.

An interesting observation during analysis of misclassified documents was that nearly half the misclassifications were between French and Spanish. To account for this, we tried to add some additional functionality whereby the probabilites for a document were recalculated if the predicted probability for French and Spanish was within a certain margin. This margin was chosen using cross validation. Probabilites for both languages were then recalculated using an equal prior probability.

Lastly, we tried implementing a Gaussian feature mapping on our feature matrix for TF-IDF with the assumption of a non-linear relationship between features and labels. However, this failed to yield good results proving the assumption false.

1) Training and Test results: The results for the different feature set variants using the Naive Bayes classifier are displayed in Figure 2.

The Term Frequency feature with randomly generated documents representation gave the best results. The extra examples generated from larger documents as well the training set now closely matching the test set seemed to help the classifier generalize better.

Some overfitting was observed from the results of the training and validation set. The unique construction of the test set also yielded different test set results from the validation set results.

Method	Bag of Characters	TF	TF-IDF	Gaussian feature mapping on TF-IDF	Bag of Characters with random documents
Entire training set	81.5%	86.7%	81.4%	73.47%	86.7%
Test set	71.90%	78.23%	74.40%	65.79%	78.50%
Validation set	77.90%	80.9%	75.3%	70.6%	83.4%

Fig. 2. Training, validation and test accuracies for the Naive Bayes Classifier

B. Random Forests

Random forests are a popular method for non-linear classification. This method essentially implements bagging over decision trees, using random sample of data to create a forest of decision trees. This reduces the high variance nature of decision trees without the need for the bottom-up pruning. Random forests generally perform quite well for classification tasks which was our primary motivation for using them [5]. However, training them is computationally expensive especially with a large number of trees.

Taking an n by m matrix containing m feature values for each of n samples, the decision tree construction function creates a node. Upon creating a node, the first k features are observed and the midpoints of all feature values are calculated. We justified taking the first k features instead of random k features because of our analysis of our logistic regression classifier as explained above. The information gain from splitting the data along each feature at each midpoint is calculated and the feature and threshold with the highest information gain are taken to define that nodes test function. However, for computational speed, not all midpoints are taken. Instead the number of feature thresholds is chosen as a function of the deviation of that parameter. That is, if the feature values have a large spread, fewer thresholds are chosen to split the data. If the feature values are very tightly distributed, more thresholds are taken in an attempt to split the data. This new node, with its associated test function, is added to a queue of nodes left to be expanded. While the queue is non-empty, the first node is dequeued and the current data at that node is split left and right by the test function. The current data is stored as a Boolean vector, where if the element is in the node's current data, it is true, and false otherwise. If the data is not split, that is, if either the left or right child has inherited no data from the parent, the parent node is turned into a leaf and is labelled with the majority class of its data. If all the samples in either child are of the same class, a leaf node is created, labelled with that class, and that node is not queued. If there is heterogeneity in the childs data classes, the child node is created in a similar manner as the first node and added to the back of the queue. In this way, we iteratively construct a decision tree breadth first. The prototypical recursive implementation was abandoned due to pythons inefficient tail recursion.

To further increase computation speed, the condition for defining whether the data was split or not was relaxed. Instead of replacing the parent node with a leaf node if a child inherits no data from it, even if the data was split n 1 to one child and 1 to the other, the parent node is turned into a leaf node. This heavily biased the tree towards a certain class because the classes in the training data are not equally likely. Over 50% of the data is French, so the majority class has a disproportionately high chance of being French. Although the number of data points in the current data is monotonically decreasing, without this relaxed conditions, the tree construction would tend to slow down for long stretches while the current data decreased by one data point each node. We postulate that this is due to homogeneity in the feature values. Since we are only looking at a subset of the features the first k features looking at more features may alleviate this problem. Alternatively, the feature set could be reduced to a lower dimension. However, this would still have to be balanced with computation speed. Either way this is an optimization left to be done. Due to the computation time and memory limitations intrinsic to decision tree and random forest construction, debugging and testing this algorithm was quite difficult, especially considering our large dataset. As a result, full optimization was not achieved and the algorithm only produced 54% accuracy.

Linear Regression	TF-IDF	TF	TF-IDF with Gaussian feature mapping	TF-IDF with randomly generated documents from original documents
Training	88.00	79.99	52.60	80.43
Validation	88.12	79.94	51.36	80.47
Test	77.4	-	-	78.29

Fig. 3. Training, validation and test accuracies for the Logistic Regression Classifier

C. Logistic Regression

Multinomail logistic regression was used from the inception of the project as a benchmark for feature selection and design. The Scikit-learn implementation in python helped quickly benchmarking any design decisions. Moreover, inbuilt support for sparse matrices (as outputted by our TF-IDF implementation) is supported which proved to be more efficient in memory usage. Lastly, logistic regression is an intuitive and quick to train and run, making it ideal for prototyping new features.

L2 regularization was considered but given the low different between training and validation errors observed, we choose not to implement it.

1) Training and Test Results: Resuts yielded from Logistic regression were quite good, showing very little overfitting. Moreover, the performance on the test set was also good with our randomly generated representation of documents giving us 78.29% accuracy. As with Naive Bayes, introducing random documents derived from the training set that mirrored the test set increased the bias of the classifer but helped it generalize better. The complete results are shown in Figure 3.

D. Neural Network

Finally, to maximize our accuracy score, we tried a feed-forward neural network using Keras [3]. The neural network structure consisted of an input layer containing 200 neurons corresponding to the top 200 characters from TF-IDF. We implemented just one hidden layer containing 100 neurons. The output layer contained 5 neurons corresponding to the five languages.

All the neurons used a sigmoid function and used the built in Adam optimizer [4]. Cross entropy was used as the loss function based on cross validation.

1) Training and Test results: As initially expected, our neural network gave excellent results on the test. We made extensive use of randomizing function to match the test set as closely as possible. This yielded the highest accuracy of all our classifiers on the test data with an 80.1% accuracy. The detailed results are shown in Figure 4.

V. Conclusion

All in all, our feature set and algorithms performed well on the task at hand. The high variation between the training data and test data presented a challenge, limiting the methods we could use to create useful features for our data. It also

Feedforward Neural Network	TF-IDF with randomly generated documents from original documents
Training	85.63
Validation	84.22
Test	80.1

Fig. 4. Training, validation and test accuracies for the Logistic Regression Classifier

motivated us to use more unconventional approaches such as the randomly generated documents which helped our classifier generalize better. Our neural network performed gave the highest accuracy on the test set, followed by Naive Bayes and Logistic Regression. The implementation of the Random Forest proved to be a challenge given the size of the data, memory limitations and very high training times.

Statement of Contributions

Alan An: Naive Bayes implementation

Salman Memon: Feature design and extraction, pre-processing, Logistic Regression and Neural Network

Niklas Brake: Random Forest implementation

References

[1] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of Machine Learning Research 12.Oct (2011): 2825-2830.

[2] Kim, Sang-Bum, et al. "Some effective techniques for naive bayes text classification." IEEE transactions on knowledge and data engineering 18.11 (2006): 1457-1466.

[3] Chollet, Franois, et al. "Keras". https://github.com/fchollet/keras. 2015

[4] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[5] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[6] Grefenstette, Gregory. "COMPARING TWO LANGUAGE IDENTIFICATION SCHEM ES." (1995).

[7] King, Ben, Dragomir Radev, and Steven Abney. "Experiments in sentence language identification with groups of similar languages." Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects. 2014.

[8] Liling Tan, Marcos Zampieri, Nikola Ljubei, Jrg Tiedemann (2014) Merging Comparable Data Sources for the Discrimination of Similar Languages: The DSL Corpus Collection. Proceedings of the 7th Workshop on Building and Using Comparable Corpora (BUCC). pp. 6-10. Reykjavik, Iceland

[9] Gebre, B. G., Zampieri, M., Wittenburg, P.,& Heskes, T. (2013). Improving Native Language Identification with TF-IDF weighting. In Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications (pp. 216-223).