The Language Classification Problem

James Leslie
Cape Town, South Africa
jlesl09@gmail.com

This report provides an overview of the methods used to create a Naive Bayes classifier capable of discriminating between English, Afrikaans, and Dutch phrases. The classifier was trained and tested using a labelled dataset containing 2761 phrases. Training was done on a subset of 2000 phrases, while the remaining 761 phrases were kept separate for evaluation of the classifier's accuracy. The final accuracy achieved by the model was 97.5%.

1 External Libraries

The code for this solution was written in Python 3.6.1. The libraries used were pandas, numpy and sklearn. These libraries are all included with Python when installed using Anaconda.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import pandas as pd
import numpy as np
```

The pandas library allows for easy data analysis on .csv files. Numpy provides many options for efficient matrix and vector manipulation. The sklearn library provides many easy-to-use machine learning methods; in the case of this problem, the Count Vectoriser and Multinomial Naive Bayes modules were used to tokenize words and train a classifier.

2 Data Cleaning and Analysis

The data provided for this problem consisted of 2839 phrases labelled either 'English', 'Afrikaans' or 'Nederlands'. However, some of these phrases were actually blank lines and needed to be removed from the data, .

```
# read input file
phrases = pd.read_csv('lang_data.csv')
# remove empty cells
phrases = phrases.dropna()
```

The `lang_data.csv` file is read in as a pandas dataframe, which allows for easy indexing and manipulation. The `phrases.dropna()` command removes all rows with empty entries from the dataframe. In total, 78 blank phrases were removed from the dataframe, leaving a total of 2761 phrases.

Submitted to:
DataProphet 2017

The data could be further cleaned by removing punctuation and capital letters, since these are not useful in distinguishing one language from another.

```
for phrase in phrases:
    # make lowercase
    phrase[0] = phrase[0].lower()
    # replace punctuation with whitespace
    phrase[0] = phrase[0].translate(str.maketrans('.,:;!?-/%', ' '*len('.,:;!?-/%')))
    # remove extra whitespace
    phrase[0] = " ".join(phrase[0].split())
```

At this stage, the data has been cleaned and prepared for feature extraction and training of the classifier.

3 Feature Extraction

However before extracting features, it was first necessary to separate the data into two separate sets for training and testing. The training set was created from the first 2000 phrases, while the remaining 761 were kept separate for testing at a later stage.

```
# create training set from first 2000 phrases
train = phrases[:2000, 0]
targets = phrases[:2000, 1]
```

The process of extracting features from the corpus of phrases was done using the Bag of Words representation. This is done using the CountVectorizer() function from the scikit-learn text feature extraction module.

```
# create the bag of words using the count vectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(train)
```

The Count Vectorizer tokenizes each phrase into individual words and assigns a unique integer ID to each word in the corpus. In this way, a single vector containing all of the unique words is created. From this vector, a matrix is constructed with the rows as each individual phrase and the columns as the unique words in the corpus.

The variable `vectorizer`, above is a vector containing all of the words in the feature space. `X` is the Bag of Words matrix. This matrix is stored as a sparse matrix, since most of the values are zeroes, thus greatly saving space.

```
>> vectorizer.get_feature_names()
['10', '1500s', '22', '90', 'aalwyn', ... , 'zoekt', 'zonk', 'zwart', 'ugur', 'n']

>> len(vectorizer.get_feature_names())
3533
```

The table below shows an example matrix for a set of 2000 phrases, with 3533 words in the feature space. This matrix is not representative of the one which was used to solve this problem; the values have been fabricated for ease of understanding.

	w1	**w2**	...	**w3532**	**w3533**
p1	1	1	...	0	0
p2	0	2	...	1	0
...	...	...	...	...	...
p1999	0	1	...	2	0
p2000	0	0	...	1	1

In this simplified example case, phrase 1 contains both word 1 and word 2 once. Phrase 2 contains word 2 twice and word 3532 once, etc.

4 Training and Testing

The classifier used in this solution was the Naive Bayes classifier. This model is based on Bayes' Theorem, which is shown below:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{1}$$

That is, we can calculate the probability of event A occurring, given that event B has already occurred if we know the probability of B given A, the probability of A, and the probability of B.

The training of the Naive Bayes classifier can be done in only one line of code as shown below:

```
clf = MultinomialNB().fit(X, targets)
```

This takes the Bag of Words matrix which has already been created and attaches the labels ('English', 'Afrikaans', 'Nederlands') to each row of the matrix. Then, Bayes' Theorem is used to create a look up table of probabilities for every word.

In this problem, there are three classes into which a phrase can be classified. For each word (each column in the matrix), the model calculates three conditional probabilities:

1. $P(word|English)$
2. $P(word|Afrikaans)$
3. $P(word|Dutch)$

What this is doing is essentially calculating the relative frequencies of words within each language. So, words which appear very often in one language will have a high score in the look up table and words which do not appear in that language will get a score of 0.

This look up table is used to classify new phrases based on the presence of already-seen words in the training set.

For example, if the following: *"This is an example"* were given to the model, then its probability of being English would be calculated, using Bayes' Theorem in the following way:

$$P(English|sentence) = [P(sentence|English) \times P(English)]/P(sentence)$$

where:

$$P(sentence|English) = P(this|English) \times P(is|English) \times P(an|English) \times P(example|English)$$

$$P(English) = \Sigma\texttt{English phrases} \div \Sigma\texttt{all phrases}$$

$$P(sentence) = P(this) \times P(is) \times P(an) \times P(example)$$

Similarly, the probability of this sentence being Afrikaans and Dutch would both be calculated in the same way and the sentence would then be classified based on the three scores it receives.

The classifier was tested using the following code:

```
test = phrases[2000:, 0]
test_targets = phrases[2000:, 1]
X_new = vectorizer.transform(test)

predicted = clf.predict(X_new)
```

The test set was created from the remaining 761 phrases. Instead of using the `vectorizer.fit_transform()` function to create a new Bag of Words, the `vectorizer.transform()` function is used to fit the test set to the existing bag of words model. Thus any words in the test set which are not in the training set will be ignored and will play no role in the classification of their parent phrases.

The accuracy of the classification can be seen as follows:

```
>> np.mean(predicted == test_targets)
0.975032
```

A more detailed summary of testing results is shown in the table below:

Set	# phrases in training set	# phrases in testing set	Testing Accuracy (%)
English Phrases	1488	567	99.118
Afrikaans Phrases	463	176	99.432
Dutch Phrases	49	18	27.778
All Phrases	2000	761	97.503

As seen above, the overall testing accuracy was 97.503%. In particular, classification accuracy of English and Afrikaans phrases was over 99%, while accuracy of classification of Dutch phrases was below 28%. This is due to the fact that the number of Dutch phrases in the dataset was very low, comprising less than 2.6% of the total number of phrases. There were simply not enough unique Dutch words which could represent the Dutch language.

5 Bonus Questions

5.1 Other Approaches to Language Classification

The approach used in the solution above was to train a Naive Bayes classifier using the Bag of Words model for feature extraction. The merits of this approach are that it is simple and not computationally very expensive. However, an issue with this method is that the trained classifier has no ability to deal with new words which are not contained in the training set. So for the purpose of this task, it is essential that the training data is varied enough so as to provide the classifier with a large enough vocabulary of words. In the case of the Dutch phrases, there were very phrases on which to train the model and this resulted in a low testing accuracy.

One approach would be to use a neural network. The same bag of words could be used for this method. But going beyond classifying phrases based on their presence of certain words, the neural network could be trained on individual words represented as a vector of letters (represented as ASCII numbers). In this way, the neural network would find patterns unique to each language in the structure of words - in the same way a human with a very limited Afrikaans vocabulary would be able to make a guess if a new word is Afrikaans vs English, the neural network would be able to classify words which are not seen in the training set. Since neural networks require inputs of equal dimensions, all words would need to be padded with zeroes such that all words are equal in length to the longest word in the training set.

A second approach would be to use logistic regression to solve this problem. Unlike linear regression which seeks to predict a continuous variable, logistic regression seeks to predict a discrete variable (in this case, the language of a phrase). In this approach the logistic regression classifier is trained in three parts, each one as a one-vs-all model. That is, three classifiers are made: English vs *not* English, Afrikaans vs *not* Afrikaans, Dutch vs *not* Dutch. For a test case, each of the three classifiers' scores are compared to each other and the language with the highest score is chosen as the class for the new test case.

5.2 Supervised vs Unsupervised learning

The key difference between supervised and unsupervised learning is that the former deals with labelled data, while the latter deals with unlabelled data.

Supervised learning algorithms are commonly used for classification and regression problems. An example of a classification problem would be to train a convolutional neural network to classify images into a discrete number of classes. The training data in this case would be a set of images, each with a label indicating to which class they belong. The classification of new images is done based on patterns which are found in the training data.

Unsupervised learning algorithms are used to perform clustering on unlabelled data. These algorithms attempt to group data by examining the similarity among that data. An example of a common clustering method is the k-means method. In the k-means method, the user selects a certain number (k) of means and selects the feature space of the dataset. Then, through an iterative approach, data are grouped into clusters based on the shortest euclidean distance to the mean of a cluster.

5.3 Classification vs Regression

The difference between classification and regression models is in their outputs.

A classification model is used to predict the class of an input. The number of classes is discrete and finite. For example, a classifier may be trained to label an input image of a face as either being child or adult. In this example, there are two classes and the classifier must choose between one of these two.

A regression model is used to predict an output value for a given input. The possible range of output values exists on a continuous scale. A simple example of linear regression would be to predict a person's height using their weight. In this example we would have a training set of ordered height and weight pairs. The regression model would use the linear trend in this data to make predictions for new inputs, e.g. a person of 85kg will be 1.87m tall.

5.4 Class Imbalance

A class imbalance is found in supervised classification when there is a majority of data belonging to one class. For example, if we trained a model to classify voice recordings as either 'male' or 'female' using a training set consisting of 9 900 male voice recordings and only 100 female voice recordings. Even if the classifier simply classified all recordings as male, it would still achieve 99% accuracy.

In these cases where there is a class imbalance, the classification rules that predict the small classes tend to be fewer and weaker than those that predict the prevalent classes. In many cases, the cost of misclassifying a minority class can be more costly than misclassifying a majority class. For example, when detecting fraudulent transactions or cancer diagnoses, the cost of not detecting one of these two serious events can be severe.

5.5 Mitigating the Effects of Class Imbalance

Overall testing accuracy is not a sufficient metric on which to examine the performance of a classifier. Using the previous example, if a model misclassifies 80/100 fraudulent transactions and 20/9 900 regular transactions the error rate would be 1%. However, if another misclassified 10/100 fraudulent transactions, but 190/9 900 regular transactions, its error rate would be 2%, even though it is much better at detecting fraudulent transactions.

One possible solution to this problem is to provide a higher weighting to false negatives in their influence on the model's accuracy (e.g. missing a fraudulent transaction should be 100 times more costly than incorrectly classifying a regular transaction as fraudulent). This **cost-based** approach provides a new metric for evaluating the effectiveness of a classifier.

In addition to cost-based approaches to dealing with class imbalance, there are some **sampling-based** approaches which are used to manipulate the training data. **Undersampling** involves removing some of the majority class from the training set so that it has less effect on the classification model. This approach risks removing instances of the majority class which are representative, thus discarding useful information. **Oversampling** involves adding more of the minority class to the training set so that it has more of an effect on the classification model. However, this just leads to overfitting and reduces the model's ability to generalise.

An algorithm called the Synthetic Minority Over-Sampling Technique (SMOTE) has been created to combine oversampling and undersampling. In SMOTE, new instances of the minority class are not simply duplicated. Instead, an algorithm is used to construct new, similar instances of the minority class. New instances are created by using an existing instance's feature vector in addition to the feature vectors of the k-nearest neighbours to the existing instance.

5.6 Deep Learning vs Non-deep Learning

Machine learning uses data to make decisions. Data can be labelled or unlabelled, but ultimately data is classified or clustered based on its features. In non-deep learning models, the features on which classification or clustering is based are human-selected. Deep learning models do not require feature extraction, they perform this step autonomously.

A convolutional neural network used for image classification of cats vs dogs does not need to be told that a cat has whiskers, eyes etc. Instead, the model is fed thousands of labelled images of cats and dogs and it *learns* the features which describe each class of image.

Deep learning models benefit from advanced computing power and the availability of large quantities of training data. These two requirements were previously the main stumbling blocks in the use of deep learning models, but recent advances in GPUs and the increase in availability of data for training has greatly increased the popularity of deep learning in the field of machine learning.