Differentiating Between Languages Using Machine Learning Algorithms

Brian Campbell, James Raynor, Andrew Stephenson, and Colin Thompson
Department of Computer Science
Brigham Young University
b.v.campbell3@gmail.com, jamessraynor@gmail.com,
andrew.stephenson123@gmail.com, cdt1_2@msn.com

Abstract

In an increasingly connected word, many transactions occur between people and organizations with different primary languages. Therefore tools for helping us manage inter-language interactions will become increasingly necessary. To aid in the, we will need programmatic methods for determining the language of a text. We have endeavored to use machine learning techniques to classify text into one of a wide number of languages.

1 Introduction

1.1 The Language Detection Problem

The first step to understanding a piece of text is to know what language it is written in. To this end, we decided to use machine learning techniques to attempt to differentiate between 27 languages: Afrikaans, Arabic, Bosnian, Czech, Danish, Dutch, English, Esperanto, Finnish, French, Gaelic, German, Greek, Italian, Kurdish, Norwegian, Polish, Portuguese, Russian, Serbian, Spanish, Swahili, Swedish, Tagalog, Ukrainian, Vietnamese, and Welsh.

1.2 Feature Extraction

Our final data set contained 23 features. There were four features for the percentage of the text in various scripts (Latin, Cyrillic, Greek, and Arabic), 14 features for the relative percentages of diacritical marks (acute, breve, grave, tilde, circumflex, macron, ring above, dot below, horn, hook above, cedilla, caron, diaeresis, and ogonek), average number of diacritics per word, average vowel and consonant cluster sizes, average word length, and average sentence length.

1.3 Learning and Results

We experimented with several different learning models, including multi-layer perceptron, k-nearest neighbors, and several different types of decision trees. In the end we got the best results with a random forest of 101 trees with 4 attributes each, for a classification accuracy of 64.11%, compared with 3.70% for a random classifier.

2 Methods

2.1 Data Gathering

In order to learn to recognize the different languages, we need significant amounts of data about each language. The data, in this case, is in the form of a large corpus of text in each language. We decided to extract these corpora from Wikipedia articles as it gives us access to a large amount of text in a single place. It has distinct advantages over the other two types of locations where large amount of text is freely available: places that give free access to books and places were people come to talk online. Compared to websites that provide free access to books, Wikipedia provides texts written in modern styles, while free book websites tend to distribute older works with lapsed copyrights. Wikipedia also provides a wider variety of subjects by a number of authors in the same amount of text as a book. Compared to chat and discussion websites, Wikipedia provides some guarantee that the grammar and spelling in the text will be correct and the text will be free from non-standard usage. However, there are some disadvantages to using Wikipedia as well. Text from Wikipedia is written entirely in an encyclopedic style. The features we chose to gather should not be affected by this, but it is worth considering. Wikipedia articles also frequently feature foreign words when dealing with foreign subjects, which could cause confusions for the learned but the frequency of foreign words was low and did not appear to mislead the learners

Creating a large corpus of text from Wikipedia was a multi-step process. We first created a program that, by using the "random page" function of the Wikipedia API, downloaded sixteen different pages from each language's Wikipedia. The Wikipedia API returns the articles in wiki markup format, which includes information about links, images and templates used in the article. For our data corpus, we want only the text of the article, so we created another program that extracted the plain text of the article from the wiki markup version of the article. This gave us 16 articles in each language consisting of three to five paragraphs of text in that language.

2.2 Data Format & Feature Extraction

Our next major barrier was finding some way to translate a block of text into a set of features that machine learning models could use to predict the language that it's written in. One of the major goals was to use features that didn't require knowing anything about the language the text sample was written in, since using features that required knowledge of the language the sample is in would prevent us from detecting the language of novel instances. To this end, we wrote a Python script that made use of simple text processing techniques to extract a relevant set of features.

The earliest features and the easiest to extract had to do with word and sentence length. We hypothesized that the simple variations in the average number of characters per word and number of words per sentence between language would provide us with a useful way of distinguishing between different languages. Shortly after these features, we added the capability to auto-detect all of the different alphabets that a text sample was written in using a Python alphabet detection library[1]. Using this capability, we first detected all of the alphabets that a given sample used, then used that information to calculate the percentage of the sample that was written in each alphabet.

The next features to be added were vowel and consonant cluster sizes. Some languages, such as French, tend to include words with larger clusters of vowels, while other languages tend to favor clusters of consonants. Still others seem to have little clustering at all, alternating relatively evenly back and forth between vowels and consonants. We hypothesized that the information about these clusterings would help us to be able to distinguish between different languages. A subroutine was added to the feature extraction script which scanned through and calculated the size of each vowel and consonant cluster, then averaged these cluster sizes. A minor inconsistency was noted here—since the characters used to denote vowel sounds are not the same between different languages, even languages that use the same alphabet, it was difficult to determine when some characters should be considered vowels and when they should be considered consonants. The letter 'y' in the Latin alphabet is a prime example of this. To resolve this problem, we investigated which characters are most frequently used as vowels in a number of languages in each alphabet, then selected a set of those characters that appeared most commonly. The instances obtained using this method are therefore an approximation rather than an exact value. An exact value could have been obtained if we had used our foreknowledge of the target language to use the correct vowel character set for each language, but as using this method would not be available for novel instances, it could result in obtaining results that do not accurately reflect the ability of the model to predict novel instances, and we therefore opted not to use it. An additional inconsistency was noted in that diacritical markings can interfere with the correct detection of vowel characters. To help resolve this problem, each text instance was run through a preprocessor which performs a unicode normal form compatibility decomposition (NFKD), converting all diacritical markings to combining diacritics, which are interpreted as their own individual character. We later realized that this may have caused some problems in these features, however, as these diacritical markings were counted as if they were consonants, which could have caused some vowel clusters to be seen as two separate smaller clusters by our feature extraction script, and which also may have affected the average consonant cluster size. Due to time constraints, these errors were left uncorrected. Further experimentation is necessary to determine if they make a significant difference in the results.

The final set of features to be extracted was the information about the diacritical markings in the text. This was accomplished by using the Python unicodedata module to determine the unicode name of each character. Since we had previously converted all diacritical markings to combining diacritics, we detected their presence by simply checking to see if the character name contained the word "combining." Individual types of diacritics could then be detected using the same technique, searching for the diacritical marking's name instead of "combining." This allowed us to determine both the average number of diacritical markings per word in the text, as well as the percentage of all diacritical markings that were of each type. Originally, some diacritical markings for the Arabic alphabet were also included in the feature set to help distinguish between Arabic and Kurdish. However, some experimentation proved that the text preprocessor that was converting all of our diacritical markings to combining diacritics was not correctly detecting Arabic diacritical markings, and therefore these features were being set to 0. These features were eliminated from the dataset as a result.

The final step in feature extraction was converting the features that we had extracted into a format usable for learning. This was made more difficult by the fact that not all of the instances that we had created had the same features. Each instance contained data about the percentage of each alphabet that it contained, but if, for example, a text were written in the Greek alphabet, it its corresponding feature had no data about the percentage of the text that was written in the Latin alphabet. This made a preliminary pass through all of the instances necessary, so that we could determine which features were actually needed. After that, a second pass through the data allowed us to convert it into the Attribute-Relation File Format (ARFF). Any time that an instance didn't contain data for a specific feature, its value for that feature was set to zero, since the only features whose values were unknown were percentages of alphabets that the instance had not contained, which, by definition, should be zero.

We then processed our entire corpus of text data by loading each file in sequence, splitting it into paragraphs, and treating each paragraph as an instance. The initial dataset produced by this method contained some irregularities, mostly caused by instances that were very short, so we imposed a restriction that an instance would only be included if it was 25 characters or more. This made for a much

[1]https://pypi.python.org/pypi/alphabet-detector

cleaner dataset, but does place some limitations on the ability of the model to detect novel instances.

2.3 Machine Learning Techniques

Our attempts to learn the data set took place in two steps. We started with a few features to start understanding our data set and to see if we got worthwhile results without more difficult to extract features. We then added in the more difficult features and saw a significant increase in accuracy. We then went on to attempt refinements of our features and methods, but saw few positive results.

Our initial attempts at learning did not use the diacritic mark based features, as it took us some time to figure out how to extract that information from the text corpus. We then attempted a variety of classifiers on the data as provided by Weka 3.7[2]. From those classifiers, we found that IBK (k-Nearest Neighbors) with 21 neighbors and inverse distance weighting and a random forest with 100 trees both performed comparably at 44.01% accuracy. Running both models together using Weka voting classifier wrapper increased that accuracy to 45.03%. We were fairly pleased with these results, since random classification would result in 3.70% accuracy. However, we were still missing more languages than we were getting correct, and we thought we could do better. Any attempts to prune attributes or change dimensions decreased our accuracy, which was to be expected, since we were using a bare minimum of features.

We then added the diacritic-based features. With the added attributes, the various models tried previously all saw increases in accuracy. IBK, now using 9 neighbors to maximize accuracy, has an accuracy of 61.19%, and the random forest using the same parameters as above has an accuracy of 64.00%. The accuracy of the voting method used with the old features is 63.27%. Since this is now lower than the accuracy of the Random Forest, we did not continue examining the voting method.

Once we had found a baseline model for the data set, we attempted to improve our model through attribute filtering. The first method we tried was PCA, which did surprisingly well, as we did not expect linear variance in our data. When retaining 95% of the variance, PCA reduced the number of attributes to 20 features and had an accuracy of 62.73% on the baseline random forest. While not improving accuracy, it didn't dramatically reduce accuracy, which indicated that changing other parameters could help us. We re-ran the PCA filter, instructing it to retain 99% of the variance. This gave us 22 attributes and had an accuracy of 63.39%. This indicates that the variance we have in our attributes contains usable information.

This inspired us to explore how valuable our attributes actually are. We calculated the information gain ratio for each of the attributes with the results presented in the following table:

Attribute Name	Gain Ratio
percent_arabic	0.8949
percent_greek	0.8817
percent_diacritics_dot_below	0.7735
percent_cyrillic	0.7585
percent_latin	0.7538
percent_diacritics_horn	0.7314
percent_diacritics_hook_above	0.7074
percent_diacritics_ogonek	0.6480
percent_diacritics_ring_above	0.5796
percent_diacritics_breve	0.5574
percent_diacritics_cedilla	0.5487
percent_diacritics_caron	0.5402
percent_diacritics_circumflex	0.5298
percent_diacritics_grave	0.4805
percent_diacritics_acute	0.4437
percent_diacritics_diaeresis	0.4420
percent_diacritics_tilde	0.4336
consonant_cluster_size	0.2833
avg_diacritics_per_word	0.2591
vowel_cluster_size	0.2096
avg_chars_per_word	0.1080
avg_words_per_sentence	0.0748
percent_diacritics_macron	0.0000

Table 1: Info Gain Ratio for Each Attribute

As can been seen from Table 1 above, only one attribute provides little useful information, "percent _diacritics_ macron." Only two attributes have a ration less than 0.1. With all but one of the features providing information and with few enough attributes to be handled reasonably by the random forest algorithm. For that reason, we opted to continue without pruning attributes.

Once we had finalized our model and attributes, we needed to fine-tune the parameters for our model. The defaults given by Weka were a forest with 100 trees generated with 5 random attributes. We explored using a large number of trees, 400, and a few trees, 10. In the end, 101 trees provided the best results. We also tried few random attributes in each tree, 1, and many, 20. The best results were provided by using 4 attributes in each tree. This provided us with a final accuracy of 64.11%.

[2]http://www.cs.waikato.ac.nz/ml/weka/

```
   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z  aa   <-- classified as
 218   1   2   0   0   0   1   1   0   1   0   0   0   3   1   0   2   0   2   0   1   1   0   0   0   0   2 |   a = GREEK
   1   8   0   0   0   0   0  13   0   1   0   2   0   0   3   0   0   3   2   3   1   1   9   0   0   0   4 |   b = DUTCH
   1   0 199   3   0   3   2   1   0   5   0  10   1   0   3   1   1   1   2   5   4   2   2   0   0   1   2 |   c = BOSNIAN
   0   0   1 258   1   0   0   1  27   0   0   0   0   1   1   0   0   1   0   0   3   1   1   0  25   0   0 |   d = UKRAINIAN
   0   0   0   0  98   0   6   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0 |   e = VIETNAMESE
   0   0   4   0   0  85   1   1   0   2   0   2   0  11   0   2   1   4   6   1   4  18   4   0   0   2  10 |   f = NORWEGIAN
   0   0   9   0   5   2 155   1   0   3   6   1   0   4   1   0   0   1   0   8   4   1   1   0   0   0   5 |   g = CZECH
   0   4   0   1   0   0   0  59   0   5   2   7   0   3   6   0   2   7   3   3   2   4   8   0   1   0   7 |   h = AFRIKAANS
   0   0   0  35   0   0   0   0 158   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   8   0   1 |   i = RUSSIAN
   3   2   5   0   0   8   4   7   1  22   2   8   0   5   3   3   3   2   3   6   4   2   5   0   0   0  14 |   j = WELSH
   0   1   1   0   0   1   4   0   0   1  91   4   1   6   1   4   3   3   0  63   2   4   3   0   0   1   6 |   k = GAELIC
   0   1  11   0   1   5   3   3   0   2   2 179   1   6   2   1   5   5   3   6   3   8   6   0   0   1   6 |   l = ESPERANTO
   0   0   4   0   0   0   0   1   0   0   0   2 228   1   0   0   0   0   0   1   1   1   0  10   0   0   0 |   m = ARABIC
   1   0   0   0   0   7   2   1   0   2  13   9   1 169   4   1   6   2   3  24   3   0   4   0   0   0   0 |   n = FRENCH
   0   5   7   1   0   3   1   6   0   2   3   4   0   1  35   1   1   4   7   2   2   2   2   0   0   0  15 |   o = SWAHILI
   0   0   1   0   0   1   0   2   0   1   2   4   0   1   5   7   1   2   8   2   3   1   3   0   0   0   5 |   p = TAGALOG
   2   0   5   0   0   0   1   1   0   4   9   4   0  13   1   0  79   0   4  16   3   0   0   0   0   0   6 |   q = PORTUGUESE
   1   2   5   0   1   2   3   6   0   2   1   5   0   6   4   0   2 208   3   2   5   3  17   0   0   1   8 |   r = FINNISH
   1   3   5   0   0   6   0   4   0   1   2   5   0   3   4   2   0   6  98   6   2   3   6   0   0   0  21 |   s = ITALIAN
   1   2   5   0   0   2   3   2   0   1  57   5   0  17   6   1   6   0   6 128   6   2   1   0   0   0   7 |   t = SPANISH
   0   0   5   1   2   3   4   1   0   4   5   7   0   6   3   1   0   4   2   2 126   3   7   0   1   0  12 |   u = POLISH
   1   1   6   1   0  24   5   2   0   1   1   7   0   3   3   1   0   4   3   4   4  30   6   0   0   6  15 |   v = DANISH
   0   4   4   1   0   6   4   4   0   2   0   8   1   9   4   1   1  23  10   0   4   5 174   0   0   5  11 |   w = GERMAN
   0   0   0   0   0   0   0   0   0   0   0   0  22   0   0   0   0   0   0   0   0   0   0  80   0   0   0 |   x = KURDISH
   0   0   8  39   0   1   0   1   3   0   1   0   0   0   0   0   0   0   1   0   0   0   1   0  55   0   0 |   y = SERBIAN
   0   0   1   0   0   3   0   0   0   0   0   1   1   1   0   0   0   0   0   0   3   5   7   0   0  60   1 |   z = SWEDISH
   2   1   4   0   0   8   3   3   1   5   4   7   0   0   6   1   4   4  14   7   8  10  10   0   0   0 117 |  aa = ENGLISH
```

Table 2: Confusion Matrix

Language	Classification Accuracy	Language	Classification Accuracy
Afrikaans	0.476	Kurdish	0.784
Arabic	0.916	Norwegian	0.538
Bosnian	0.799	Polish	0.633
Czech	0.749	Portuguese	0.725
Danish	0.234	Russian	0.778
Dutch	0.157	Serbian	0.500
English	0.534	Spanish	0.496
Esperanto	0.688	Swahili	0.337
Finnish	0.721	Swedish	0.723
French	0.671	Tagalog	0.143
Gaelic	0.455	Ukrainian	0.804
German	0.618	Vietnamese	0.933
Greek	0.924	Welsh	0.196
Italian	0.551	Average (not weighted)	0.611

Table 3: Classification Accuracy by Language

3 Results

As mentioned above, our final classification accuracy was 64.11%. This, though, is across all languages and instances. However, there are some interesting insights into our methods when we consider some of the languages.

3.1 Average Accuracy

The accuracy discussed above is the accuracy across all instances. Table 3 above considers the classification accuracy for each language individually. The average classification accuracy for each language is 0.611, slightly lower than the accuracy by language. This tells us that our model was more accurate than average on some languages that had more instances than some of the other languages. However, for the average language, we were able to accurately classify it more often than not.

3.2 Misclassified Languages

There are a number of languages that we classified incorrectly more often than correctly (had a classification accuracy less that 0.5), namely Afrikaans, Danish, Dutch, Gaelic, Spanish, Swahili, Tagalog, and Welsh. We would like to consider some of the specific results for these languages in order to discuss correcting specific problems later.

Dutch was frequently misclassified as Afrikaans and German. This is a reasonable mistake to make, as Dutch is closely related to German and Afrikaans is derived from Dutch. Therefore, it is not hard to imagine why it was frequently confused with the other two.

Spanish and Gaelic were often misclassified as one another. This is an unexpected result, as Gaelic and Spanish are not closely related languages. However, from the perspective of the attributes that we extracted from the text, they are surprisingly similar. They both have similar vowel and consonant distribution, and both use the same two diacritical marks, the acute accent and the tilde, with similar frequency.

Welsh and Tagalog are interesting cases to consider because both had very poor accuracies (less that 0.2) yet they didn't have specific languages that they were frequently misclassified as, like the languages above. Welsh was classified as Welsh almost twice as often as any other language. Tagalog was classified more often as Italian than Tagalog, but only by a margin of one instance.

3.3 Correctly Classified Languages

There were also a handful of languages that were accurately classified at a rate that was much higher than average (greater than 0.8). Those languages are Arabic, Greek, Ukrainian, and Vietnamese.

Arabic, Greek, and Ukrainian and connected in that they use non-Latin alphabets. Greek is the only language we considered that uses the Greek alphabet. Arabic is one of two languages that use the Arabic alphabet, the other being Kurdish. Ukrainian is one of four languages we considered that uses the Cyrillic alphabet, with the others being Russian, Serbian, and in some cases, Bosnian. Since we considered so few languages with non-Latin alphabets, once you knew which alphabet the text used, the possible classes was reduced dramatically.

Vietnamese, however, is written with the Latin alphabet. The reason we were able to classify Vietnamese with such accuracy, though, is because of it's frequent use of a wide variety of diacritic marks. Czech was the only language besides Vietnamese where the majority of instances had a rate of diacritic per word greater than one. However, we were able to differentiate between the two because of the many types of diacritics both languages use, they only share two, the grave and acute accent. Unfortunately, Czech does share its other diacritics with other languages, leading to more misclassifications of Czech.

4 Discussion

The results obtained are unsurprising, for the most part. Most of the languages that were frequently misclassified have at least one other language in the set of possible outputs that is very similar, making it very difficult for the machine learning models to correctly separate them. In order to accurately classify these instances, the addition of new features may be necessary. We hypothesize that the addition of some features based on letter frequency may help to distinguish between these languages.

One problem with using letter frequency as a feature, however, is that it requires determining before execution which letters should be used. We considered adding one new feature for each letter, but that would add several dozen new features, most of which would probably not be useful. The best results would be had if we determined beforehand which letters were the most important. This results in a problem in that should we decide to add any new languages to the dataset, we must then determine if we should modify the set of letters whose frequencies we detect to accommodate this new language. This problem is already present in our current feature set, since we manually define which characters are considered vowels. Any language that does not use the alphabets that we have detected thus far would have a vowel cluster size of 0 and a consonant cluster size equal to the average word length of that language. This same problem occurs with all of the diacritic-based features. In order to make the model more extensible, it would be desirable to find features which do not require the manual specification of specific characters to detect, or to determine some method of programmatically selecting which characters should be detected, so that the addition of new languages into the corpus does not result in the need to entirely redesign the model used to learn them.

An additional possibility for further experimentation would be to fix the errors already mentioned in the detection of vowel and consonant cluster sizes. It is possible that these features could give much more meaningful data than they currently do if extracted correctly, which would in turn result in higher accuracy.

In the same vein is the possibility of fixing the detection of diacritical markings in the Arabic alphabet. This could help to detect the difference between Arabic and Kurdish. However, these languages tended to be classified relatively accurately even without the diacritical features, so this experimentation would not be as important as other possibilities unless more languages that used the Arabic alphabet were introduced into the list of possible languages to consider.

5 Conclusion

The problems presented by language barriers are widespread and varied. As has been shown, however, the ability to close these gaps is not far away. While the strategies used here provide good methods for detecting languages, and future efforts could achieve even better results, the problem of language detection won't have been completely solved until it is possible to add languages in new alphabets to the data set without having to manually adjust the features that are required to effectively learn it.

Once languages can be reliably auto-detected, the first step in automated machine translation will have been achieved. Further areas for investigation are auto-detecting the language of an audio sample and speech recognition, which would allow spoken communication to be auto-translated correctly. The barriers to world communication are coming down.

Our source code repository can be found at
https://github.com/jamessr2/Language-Detection