**Abstract**

Native Language Identification (NLI) is the idea of determining a native language of an individual based on an essay written in a second language. The problem of NLI is usually treated as a classification problem. Our main focus in this project was to find a feature extraction and classification algorithm that could improve the accuracy of language classification from a base of around 9% which is randomly choosing languages.

**Introduction**

As mentioned in class, Natural Language Identification is a new and emerging research topic in the field of machine learning and natural language processing. In a general sense, the task is to be able to identify the native language of an individual based on their writing done in a second language. In many cases, research shows that individuals writing in their second language demonstrate language use patterns that are common amongst others of the same background. In that sense and many others, researching NLI helps to provide feedback to second language learners. It may enable them to better adapt their educational goals while also influencing language teaching techniques (Koppel et al., 2005; Blanchard et al., 2013). After extensive research and development in NLI, the most common framework that is used as a basis for expansion was introduced by Koppel, Schler, and Zigdon (2005). They used the idea of essay-based text that was pulled from the International Corpus of Learner English with an SVM classifier (Granger et al. 2009). Using previous attempts, they were able to broaden their approach to include syntactic, lexical, and stylistic features like function words, character level n-grams, and part of speech bigram tagging. From then, there have been many attempts to improve the accuracy of natural language identification problems.

**Data**

For the purposes of this project, our class was tasked with using the TOEFL-11 as our corpora. TOEFL-11 is a data set for NLI from 2013 (Blanchard et al., 2013). In this, there are 11 native languages including Arabic, Chinese, French, German, Hindi, Italian, Japanese, Korean, Spanish, Telugu, and Turkish. The data that is included is responses to 8 different prompts written in English. From each language, there are 1,100 essays. Totaling 12,100 essays, there are 9,900 from the training set, 1,100 in the development set, and 1,100 in the testing set.

**Basic System Design**

When we first began approaching this problem, we thought about doing a part of speech n-gram feature set going up to at least 5-gram models. Knowing that this was just the most basic level, we were happy to see an accuracy level of 21.9% which we knew could be improved. Because the accuracy of randomly choosing a language is less than 10%, we knew that even though this was a low accuracy, we were headed in the right direction. At this point, our focus with the n-grams was frequency. After playing around and trying out new ideas, we thought about implementing a binary based n-gram model. Just from this change, we were able to increase the accuracy to 43.3%. Because the accuracy went up by so much, we decided that a binary based approach would be much more successful than a frequency based approach.

Our next idea to improve the accuracy was to include part of speech tagging using the UPenn Tagset as opposed to the universal tagset we were previously using. At the bigram level, as we trained, although it did take a little longer than we thought it would, we were able to bump up our accuracy to 47%. Hoping to increase the accuracy even more, we implemented this same algorithm at the trigram level. Unfortunately, because the size of the feature vector using trigrams was 97, 336 we ran into an out of memory error. After some optimization, we tried to train the model again which after 3 hours we then terminated.

After doing more research, Katherine suggested splitting up each essay into shorter paragraphs (say 4 on average) in order to give us more labeled data to train on. From there, we would use the most frequent predictions for the separate paragraphs and combine them to be the prediction for the whole essay. Unfortunately, that did not improve the accuracy as we had hoped and actually cut our accuracy down to ~20%.

Since we had access to the English proficiency levels of each essay, we were curious to see if including those in our calculation would improve our accuracy. Our focus with this approach was to read in the proficiency levels with each essay and turn it into a weight. So, high accuracy essays were given a weight of 1, medium accuracy essays were given a weight of 2, and the low accuracy essays were given a weight of 3. Our initial weight addition raised our accuracy by .3% to a total of 47.6%. Though the training data was weight, we didn’t have any way of including weights into our predictor once we approached the dev set. However, we hoped that the weight given to certain essays in the training data would help our model more accurately classify responses in the dev set. Additionally, we tried varying the weight sizes but found a reduction in accuracy of roughly 7%.

After we tried adding weights, we added a concatenation of the bigram and unigram feature vectors which was just an addition of one line of code. After training and running the evaluate.py file on this model, we were able to increase our accuracy to 47.9%. Although it was not the improvement we had estimated, it still increased by a bit.

Just for experimental purposes, we briefly tried another approach using a linear SVC model. A linear model uses a one-vs-all multiclass reduction and liblinear while the regular SVC uses a one-vs-one multiclass reduction and libsvm. The liblinear estimators are optimized for a linear case and for that reason converge faster on big amounts of data. We tried this both on the binary based model we created and the frequency model. Unfortunately, this also decreased our accuracy down to 40%.

**Features**

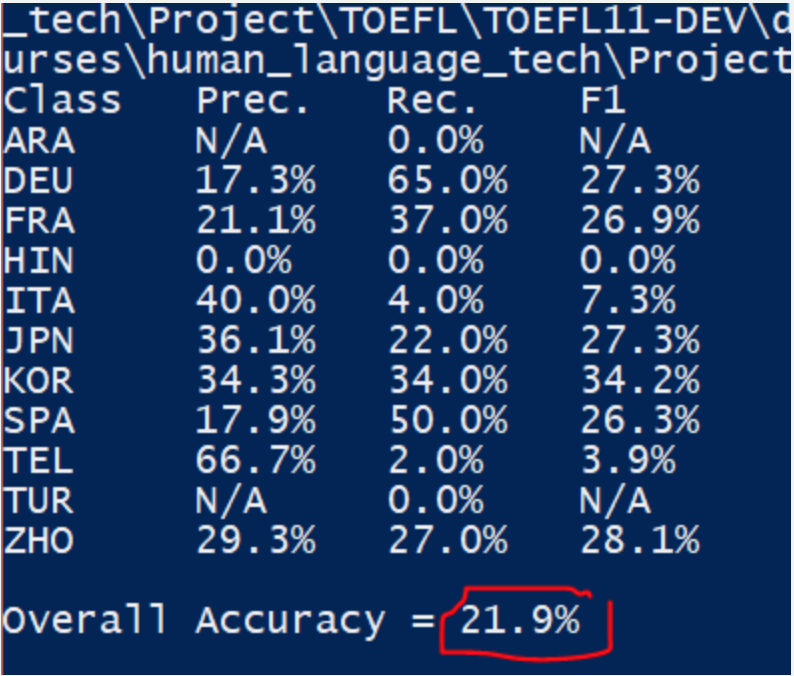
As we began, our main idea was deciding whether we wanted to use n-grams, at which level, and if we wanted to also include part of speech tagging. As mentioned earlier, we first focused on frequency based as that was what we were most used to dealing with in n-gram models. However, after researching other ways to represent n-grams, we came across a binary based representation. Instead of the essay being a vector whose elements are the frequency of different word n-gram represented in this essay, we used a binary representation where a 0 would mean that a specific n-gram is not represented in the essay and 1 would mean that it is.

**Classifier**

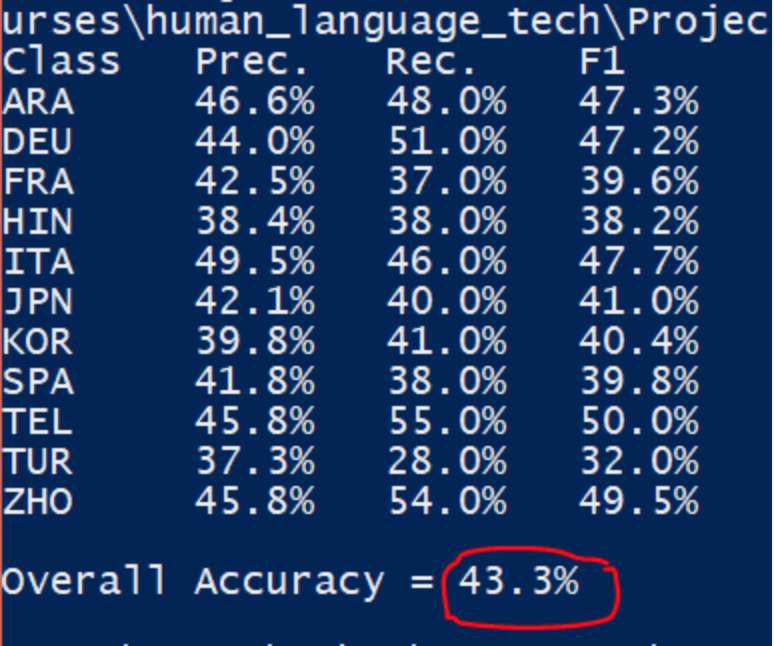
As mentioned earlier, there were many approaches we could’ve taken for the classifier. While researching previous studies and results of this NLI problem, we were able to see that SVM (support vector machine) was the most commonly used classification algorithm. Along with that, we also knew that SVM is known to be an adaptive model which is beneficial when the dimensions are high. To see if the accuracy would improve at all, we tried researching other classification methods. For the sake of experimentation, we tried using a Naive Bayes classifier, k-nearest neighbors, and a Neural Net classifier. In the case of Naive Bayes, we chose this classifier due to the fact that during our research we saw that many online sources suggested using this for classifying text. However, using this classification decreased our accuracy to around 21%. Another thought we had was that there might be multiple small “clusters” of the same class in the feature space. For that assumption, K-nearest neighbors was a good classification method to try. Unfortunately, this also decreased our accuracy to 25%. Using the Neural Net approach, the accuracy still decreased but not to the same degree as the Naive Bayes. Due to the fact that the svm model was producing the greatest accuracy, we decided to continue with that.

**Results**

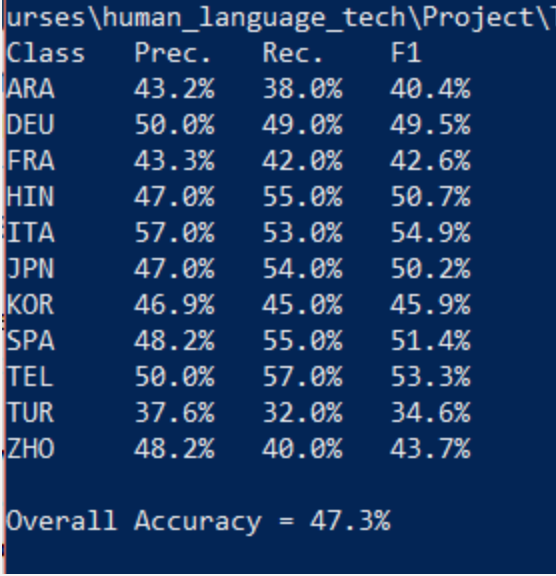
Figure 1 shown below is the details and final accuracy of the first model we tried. This was the Frequency-based word bigram model. Most of the percentages displayed are lower than 40%. Figure 2 below is the same model using a binary based approach. As you can see, the overall accuracy doubled just by making this small change. In this case, almost all of the percentages are over 40%. Figure 3 is a representation of the model in which we used the binary based word bigram with the default tagset. Although a smaller bump, you can see most of the values are  now above 50%. The last Figure 4 shown is a representation of our final model. This one included concatenating the feature vectors for unigrams and bigrams while also adding the proficiency weights.



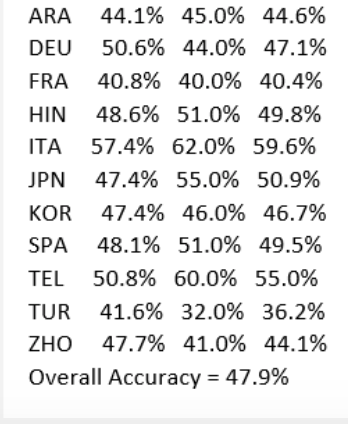
**Figure 1- Frequency Based Model**



**Figure 2- Binary Based Model**



**Figure 3- Binary Based model w/ Default Tagset**

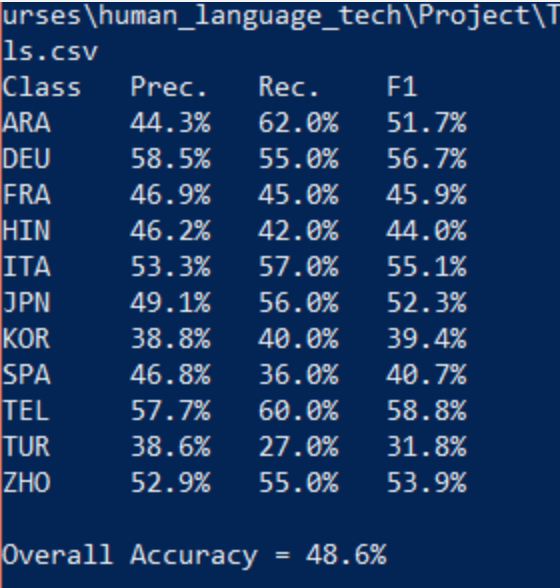
****

**Figure 4- Final Result**

**Evaluation**

Using our model applied on the test data, we were able to receive an accuracy of 48.6%. Shown below, this resulted in a higher accuracy than we received with the training data. With the last model we trained, we wanted to try it out using trigrams but unfortunately did not have enough time. We felt that if we had a couple more days we would’ve been able to make some more improvements on the overall accuracy.

If we could approach this task again, here are some other methods we might try: 1.) use four grams with the universal tagset 2.) take advantage of sklearn’s feature extraction capabilities to create more distinguishing and sophisticated feature vectors and 3.) find ways to more thoroughly investigate the nature of the feature vectors - what are the distinguishing features of each vectorized response?



**Figure 5 - Test set**

**References**

Blanchard, D., Tetreault, J., Higgins, D., Cahill, A., and Chodorow, M. 2013. *TOEFL11: A Corpus of Non-Native English.* Educational Testing Service.

Koppel, M., Schler, J., and Zigdon, K. 2005. *Automati- cally determining an anonymous author’s native lan- guage*. In ISI, 209–217.