Abstract

Native Language Identification is the task of predicting the native language of a speaker, given audio of their speech. We employ traditional (e.g., SVMs) and neural (e.g., Feed Forward Nets, RNNs, GRUs) methods of Machine Learning on speech data procured from the TOEFL exam administered by the ETS. Through utilizing both i-vector representations of speech audio, as well as textual speech transcripts, we obtain results that rival cutting-edge accuracy with our best performing one, a linear SVM at 83% on a dataset using both speech transcripts and i-vectors.

1 Introduction

In this paper, we are very interested in identifying the native language of a writer based on a sample of their writing or speech, which is the task identified for the 2017 Shared Task on Native Language Identification by the Association for Computational Linguistics. The task has been traditionally framed as a classification problem where a set of native languages is known a priori, focused on identifying the native language of a non-native speaker of English-based inputs. Given a dataset provided by the ACLI, this task is composed of three sub-tasks: a) 11-way classification using all available data sources, b) 11-way classification solely using the essays, c) 11-way classification using solely the transcripts and/or i-vectors. This is a very interesting problem that entails exploring the nuances of the difference in pitches, accents, emphasis, and other facets of speakers using different languages.

With that in mind, we seek to explore approaches utilizing different statistical methods to tackle the above problem or different abstractions of the problem through the use of traditional statistical learning methods such as Support Vector Machines to the utilization of neural networks.

2 Background

2.1 NLI Shared Task

The original task of Native Language Identification, proposed in 2013, involved predicting the native language of a writer based solely on examples of their writing. In 2016, a subtask was devised that involved predicting native language based on speech samples only. In this paper we work on the speech subtask, operating on i-vector representations of the speech audio as well as transcriptions of the speech recordings.

There are eleven different languages to classify from: Arabic, Chinese, French, German, Hindi, Italian, Japanese, Korean, Spanish, Telugu, and Turkish.

2.2 Past Work

The earliest models for speaker verification relied on directly representing speakers using the super-vector M, a method called joint factor analysis, proposed by Kenny et al. (2008) [3]. However, the new low-dimensional i-vector representation proposed by Dehak et al. (2011) led to results that outperformed the traditional joint-factor analysis [2]. In the literature, these representations have been used in classification methods such as LDA and SVMs. Bahari et al. (2013) then proceeded to employ these i-vectors in accent recognition and achieved consistently higher accuracy than other methods of the time [1].

Below, we provide a more comprehensive review of recent literature relevant to our task.
2.3 Literature Review

The task of native language identification has been explored quite a bit in the past. We seek to explore the different approaches that previous attempts have performed to improve on them.

In "An I-Vector Extractor Suitable for Speaker Recognition with both Microphone and Telephone Speech," Senoussaoui et al. [8] extends the work of Dehak et al. on extracting i-vectors from low-dimensional space by focusing on speaker recognition given sparse data. A small amount of microphone data was supplemented with ten times the amount of telephone data. After extracting the i-vectors, a total variability matrix was created by appending the telephone data to the microphone data. The new feature extractor combined with a fused joint factor analysis (JFA), support vector machine (SVM), and cosine distance scoring (CDS) classifier was able to produce a 13% relative improvement on error rate and decrease the minimum value of detection cost function.

Although our approach will not involve supplemental data, the paper provides helpful approaches for evaluating and combining different classifiers. Some of the approaches taken in the paper include: the use of gender-dependent classifiers, normalizing the cosine kernel, using z-normalized SVM scores, using zt-normalized CDS scores, use transcripts of speech data as a proxy for voice activity detection, and combine JFA with SVM and CDS. We don't yet know if any of these approaches will produce useful results, but the success found by Senoussaoui et al. is promising.

In "Accent Recognition Using I-Vector, Gaussian Mean Supervector and Gaussian posterior Probability Supervector for Spontaneous Telephone Speech”, Bahari et al. [1] discuss three utterance modelling approaches (Gaussian Mean Supervector, i-Vector and Gaussian Posterior Probability Supervector) and three classifiers (Support Vector Machine, Naive Bayes Classifier, and Sparse Representation Classifier) in an attempt to find suitable matches between utterance models and classifiers for the accent recognition problem. The authors employed a dataset from the National Institute of Standards and Technology 2008 Speaker Recognition Evaluation (SRE) database, resulting in an evaluation database consisting of English utterances of Russian, Hindi, English, Thai, Vietnamese, and Cantonese speakers. It was found that the Gaussian Posterior Probability Supervector and the i-vector models perform better in the accent recognition task than the Gaussian Mean Supervector model, and among the classifiers, Support Vector Machine works best with i-vector and Sparse Representation Classifier works best with the Gaussian Posterior Probability Supervector.

This paper provides some key insights in what utterance modelling approaches and also what classifiers should be employed for optimal results in the accent recognition task, and this is beneficial since this is a different abstraction of the problem we proposed. It is most helpful because it shows the different permutations of the utterance models and classifiers and what results they produce. This paper is especially relevant because we are interested in exploring the usage of i-vectors in the task of native language recognition.

In "Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval," Liu et al. [5] describes a multi-task neural framework for jointly training two separate, but complementary NLP tasks: Query Classification and Web Search Relevance Ranking (given a query). The data and weights are shared between the two tasks, and each task has its own softmax output from the penultimate layer of weights. This type of joint training appears to have a synergistic effect on the performance of the two tasks, where the weights are updated to store information relevant to both tasks. There is also an alternative neural structure in which only a subset of the training data (and consequently, weights too) are shared between tasks.

Drawing from this inspiration, it would be interesting to consider a multitask model, involving Native Language Detection and Dependency Parsing. The idea is that when speakers of different native languages try to compose the same idea into speech/writing, the general semantics will be the same. Thus, more interesting results will come from features capturing syntactic information. A primary issue, though, is that the NLI Shared Task data does not provide gold-standard dependency parses for the speech transcriptions.

2.4 I-vectors

I-vectors are fixed-length and low-dimension representations of audio files. Initially used for speaker recognition, they are able to convey a wide variety of speaker characteristics in a com-
pact form and have since been used in a variety of speaker classification tasks. Features such as transmission channel, acoustic environment, or phonetic content can be represented by in i-vector. Previous studies have shown i-vectors yield impressive performance for language recognition [6]. They outperform JFA by considering only the single total variability subspace and not trying to separate inter-class and channel variability [2]. And, i-vectors can be trained in an unsupervised manner.

2.5 Total Variability Model

I-vectors are extracted using the Total Variability Model which states that the qualities of an utterance can be summarized into a supervector $M$. This model can be decomposed as:

$$M = m + Tw + e$$

Where $m$ is the Universal Background Model supervector, $T$ is a low-rank matrix that captures the variability of $M$, and $e$ is a noise term which captures variability not modeled by $T$.

The i-vector is $w$, which represents a speaker in the variability space defined by $T$.

For each utterance, our i-vector is a maximum a-posteriori estimate of the latent variable $w$.

3 Data

3.1 Speech Data

The data was provided by the Educational Testing Service (ETS), as the speech data is from spoken responses of the TOEFL iBT exam. The dataset contains speech data from 11,000 test takers, each producing a 45-second English language response. There are typically around 100 words per spoken response. And there are 1,000 speakers from each mother language in the dataset. An additional 1,100 transcriptions and i-vectors were also provided for testing purposes.

Orthographic transcriptions of each spoken response were also provided, both tokenized and un-tokenized.

3.2 I-vectors

Along with the transcriptions, we also obtained i-vectors that represent each speech audio sample, each vector being a low-dimensional (800 dimensions) representation of the supervector capturing the utterance – which were formed from a Gaussian mixture of 1024 components. These components were tuned on the development set by using the Kaldi ASR toolkit. I-vectors have traditionally been used in the task of speaker verification, though their application to accent-recognition in recent years has shown significant success due to the fact that they capture the variability of a speaker’s speech patterns – at a high level, this could include quirks in pronunciation of different phonemes in the English language.

3.3 Glove vectors

As input to our neural network models, we employ GLoVe vector embeddings of the words in the speech transcriptions. GLoVe vectors are vector representations of words that capture the contextual information of a word, in relation to the words it co-occurs with in a corpus. These vectors are formed by minimizing a log-bilinear training loss on the word co-occurrence matrix of the data [7]. The vectors we utilize in this paper are trained using data from Wikipedia and the Gigaword corpus.

4 Approach

4.1 N-grams and dimension reduction

To incorporate speech transcript features into our SVM models, we used a simple bag-of-words representation for the terms in the speech transcriptions. This easily led to an explosion in the dimensions of our feature vectors, so for dimensionality reduction we utilized a truncated SVD representation for the data.

The truncated SVD representation is obtained by first finding the SVD of the training set matrix. Let $M$ be the training set matrix, of dimension $n \times d$ where $n$ is the number of training examples and $d$ is the dimensionality of the feature vectors. We obtain the decomposition:

$$M = U \Sigma V^*$$

where the columns of $U$ form an orthogonal basis, $\Sigma$ is a diagonal matrix of the "singular values" of matrix $M$, and the rows of $V^*$ also form an orthogonal basis.

As a rule of thumb, Leskovec et al. (2014) suggest that the sum of the squares of the retained singular values should be at least 90% of the sum of squares of all the singular values [4]. To reduce the dimensionality of $M$ to, say, dimension $t$, we
simply zero-out all element of matrix $\Sigma$, except for the $t$ largest principal components. By this, we form a new, $t \times t$ matrix, denoted $\Sigma_t$. Analogously, we then discard all but the first $t$ columns of $U$ to form the matrix $U_t$ of dimension $n \times t$, and discard all but the first $t \times t$ rows of $V^*$, to form the matrix $V_t^*$. The lower-dimension representation of the data matrix $M$, is then:

$$M_t = U_t \Sigma_t$$

$M_t$ is then of dimension $n \times t$. Note that $U_t \Sigma_t V_t^*$ returns a matrix of the same dimensionality as the original training-data matrix $M$, however a few components of the variability of the rows (observations) have been discarded.

Now, when it comes time to transform the test data, special caution must be taken. One cannot simply repeat the above process on the test data, i.e., re-running the truncated SVD procedure. This would cause the training and test data to be represented in two different vector spaces. This renders the data useless, and we then cannot assess our model on the test set.

But we also cannot merge the training and test sets solely for the sake of the truncated-SVD procedure, then separate them after dimensionality-reduction has been performed. This is a common pitfall. Though the training and test sets are now represented in the same vector space, we have introduced significant bias in the data, as we have allowed the test set to influence how the training set is represented. This makes our model assessment unreliable often leads to optimistic values.

What we did, then, was retain the matrix $V_t^*$, whose rows are often called the principal components of the dataset. For the test set matrix, denoted $T$, we reduce the dimensionality of $T$ by right-multiplying it with the matrix $V_t$, the conjugate transpose of $V_t^*$. This transforms the test set matrix to the same vector space as the dimension-reduced training set matrix.

### 4.2 Linear SVM

We started by looking at SVMs since they are effective in high dimensions and when there are more features than samples.

SVMs are not scale invariant and it is necessary to scale testing and training data in the same manner. Thus, we attempted to standardize the i-vectors and speech transcriptions before and after merging the features. We experimented with standardizing the mean and variance of the vectors, and with normalizing the L2-norm of the vectors. We then ran and compared our linear SVM trained on: transcription data alone, i-vectors alone, and with the two datasets combined. We focused on combined transcriptions and i-vectors since all the features combined saw the best results.

### 4.3 Other SVM Approaches

After tuning our linear SVM, we began exploring other SVM classifiers and kernels (RBF, Polynomial, C-SVM, and Nu-SVM). All yielded notably worse performance (by a few percentage points) than all of our linear SVM trials. We also retried all our methods of data standardization and were unable to achieve better performance than using L2 normalized i-vectors with a linear SVM.

### 4.4 Neural Networks

We then explored the usage of neural nets as a means of classification, with primarily two kinds, a two layer feed forward neural net and then a GRU (Gated Recurrent Unit) recurrent neural net. The feed forward neural net is implemented as a simple baseline for deep learning classification, where there exists no cycles between the units. The GRU recurrent neural net is then employed in an attempt to capture more context, as it is a variant of the LSTM (Long Short-Term Memory) RNN (recurrent neural net). LSTMs were designed to combat vanishing gradients (which typically prevent RNNs from learning long-term dependencies) through a gating mechanism. The idea behind a GRU layer is similar to that of a LSTM layer with a reset gate and an update gate.

These two models seemed to perform well, contingent on tweaked hyperparameters, with the GRU performing better than the feed forward.

### 4.5 Cross validation

At the time of writing, we have only been provided with the official training and development sets for the NLI Shared Task. Thus, we tune all our models by performing 5-fold cross validation on the merged training and development data sets. Representatives from the NLI Shared Task will provide the complete test set in late June.

The method of K-fold cross-validation has been widely used in machine learning and statistics, especially when the dataset is small [cite]. It involves splitting the data into K disjoint subsets ("folds"), and iteratively choosing one of the folds
to hold out as the model is trained on the other K-1 folds –finally testing the model on the held-out fold.

The benefits of cross-validation are that it allows us to assess our models using all of the data available to us, using different sets of “folds” to examine the robustness of the model as we vary the training set. Estimating error using K-fold cross validation also has the benefit of reducing the bias of the estimator as we increase K. However, the variance of the estimator also increases. This is because at each iteration of cross-validation, you are training with a data set that greatly overlaps with the training sets used in the other iterations. Thus the error estimates produced by each iteration will be dependent. In the end, the final error estimate, which is just the average across iterations, will have a variance that increases as the co-variance (a measure of dependence) of the individual terms increases.

5 Results

5.1 Baseline

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CV Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transcriptions</td>
<td>52%</td>
</tr>
<tr>
<td>I-vectors</td>
<td>75%</td>
</tr>
<tr>
<td>Transcriptions and I-vectors</td>
<td>76%</td>
</tr>
</tbody>
</table>

Table 1: Baseline Cross Validation Accuracies with a Linear SVM

With all the provided data combined, we established a baseline for speech transcriptions, i-vectors, and speech transcriptions combined with i-vectors. The respective baseline accuracies can be seen in Table 1 in the Appendix.

5.2 Best Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CV Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combined</td>
<td>82%</td>
</tr>
<tr>
<td>I-Vectors</td>
<td>79%</td>
</tr>
<tr>
<td>Transcriptions</td>
<td>52%</td>
</tr>
</tbody>
</table>

Table 2: SVM Accuracies Post-Normalization on Transcriptions, I-Vectors, and Combined

Table: SVM Accuracies Post-Normalization on Transcriptions and I-Vectors

<table>
<thead>
<tr>
<th>Kernel</th>
<th>CV Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>83%</td>
</tr>
<tr>
<td>Nu</td>
<td>75%</td>
</tr>
<tr>
<td>RBF</td>
<td>45%</td>
</tr>
<tr>
<td>Polynomial</td>
<td>44%</td>
</tr>
</tbody>
</table>

Table 3: SVM Accuracies Post-Normalization on Transcriptions and I-Vectors

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>CV Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSProp</td>
<td>79%</td>
</tr>
<tr>
<td>Adam</td>
<td>78%</td>
</tr>
<tr>
<td>SGD</td>
<td>9%</td>
</tr>
</tbody>
</table>

Table 4: 2 Layer Feed Forward Neural Net Accuracies Post-Normalization on Transcriptions and I-Vectors

For the data that we fed into our neural nets, we used the combined transcription and I-vector dataset because the combined approach was optimal in all previous cases, so we were seeking to improve that optimal case. We tried improving our base neural net with various optimization and initialization (e.g., Adam optimizer, Adagrad optimization, Xavier initialization) routines. Adam and RMSProp were the two optimizers that performed best and others performed very poorly. Granted, these models were only trained for 10 epochs (due to time constraints), so perhaps allowing the models more time to train would show us more promising results. For the accuracy plot, see Figure 1.

5.3 2-Layer Feed-Forward Network

5.4 GRU-RNN

Similar results were obtained with the GRU-RNN as with the 2-Layer Feed Forward Network. These recurrent models took significantly longer to train per epoch (taking GLoVe data into account) so more time is needed to tell what the contingencies of good performance are. With that said, the GRU-RNN was trained for only 3 epochs and performed really well, so better performance might be able to be reached with more time given. For the accuracy plot, see Figure 2.
<table>
<thead>
<tr>
<th>Optimizer</th>
<th>CV Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSProp</td>
<td>81%</td>
</tr>
<tr>
<td>Adam</td>
<td>78%</td>
</tr>
<tr>
<td>SGD</td>
<td>14%</td>
</tr>
</tbody>
</table>

Table 5: GRU-RNN Accuracies Post-Normalization on Transcriptions and I-Vectors

6 Discussion

Our results confirm that i-vectors are fantastic for native language identification, due to their ability to capture the speech variations of a speaker. And upon L2 Regularization of the i-vectors, we saw improvements across all of our models. An explanation for this may be that the absolute magnitude of the components of the i-vectors aren’t as vital as the relative differences within a given vector.

Keeping in mind that an i-vector represents the coordinates of a speaker’s utterance in the dimensions of the vector space defined by the total variability matrix $T$, one can envision each dimension encoding a certain aspect of speech that can vary across individuals/utterances. Thus, for an individual speaker, it appears to be more informative to quantify the relative differences of these speech “aspects,” rather than their magnitudes.

The confusion matrix included in the appendix provides some interesting insights. The matrices looked similar for all our classifiers. struggled the most. ...

Incorporating N-grams features in our models improved accuracy slightly but not significantly (an increase of roughly 1 percent). Since we employed a truncated SVD representation of the dataset to reduce dimensionality as well as runtime of the model-training, a bit of the information from the N-grams features may have been lost.

Classifiers that are designed for unsupervised learning appear to have poor performance in the context of L1 identification.

The RBF and Polynomial SVCs performed extremely poorly compared to the Linear and Nu SVCs. Perhaps this is because the data follows a certain structure that is well-modeled with a linear separator, but that is not compatible with the RBF or polynomial kernels. Performance could be improved via parameter tuning but is unlikely to surpass the results of the linear SVM without more data.

GRU-RNN performs better than the feed forward neural net because the GRU-RNN is able to capture more context in terms of the speech transcriptions, especially with the usage of GLoVe vectors to learn word representations. An important thing to note in the context of speech transcriptions is that GRU is actually a variation of LSTM RNNs, which were designed to combat vanishing gradients through a gating mechanism. The vanishing gradient problem is dangerous because it prevents standard RNNs from learning long-term dependencies between words that are several steps apart. This is problematic because the sentiment of a given utterance is often determined by tokens that are not very close, that is, context is needed in order to get the big picture. This is a consequence of the sigmoid function having a derivative of 0 at both ends approaching negative infinity and infinity, and gradient contributions from “far way” steps become zero and the state at those steps does not contribute to what the model is learning. GRU-RNNs are designed to address vanishing gradients and learn long-range dependencies, which is a great thing in the context of learning from speech transcriptions.

7 Conclusion

Surprisingly, our best performing model thus far was the simplest one. Given only 11,000 training examples, it was likely our more complex models were starved for data and overfit the data. Another possibility to explore is whether there is some structure inherent in the data that allows it to be well-modeled with the simple SVM, but ill-suited for the more complex models.

We will spend more time training our models further to see if the results above are maintained. In the future, we would like to see what insights are gained from the combination of statistical classification techniques to obtain better performance for L1 identification.

Another point of potential improvement is to experiment with GloVe vectors that are trained on larger corpora, and that have dimensionality – thus containing more information that can be leveraged in making classification decisions. Additionally, it would be interesting to experiment with the GloVe vectors we use in our models here. For instance, let’s say that we train a model for neural dependency parsing, which takes GloVe embeddings as an input. If we allow the GloVe vectors to be updated just like any other parameter while this de-
dependency parsing model is being trained, at the end of the process, we would have a set of GloVe vectors that were updated to capture some information useful to dependency parsing—including, but not limited to, syntactic information. It would be interesting to see how using these enhanced GloVe vectors would affect our results, especially since none of our models incorporate syntactic features.

References


A Figures

B Code