6.867 Final Project

Applying Machine Learning Methods to Classify Spoken Language

Qian Long, Wenting Zheng, Laphonchai Jirachuphun

December 6, 2013

1 Introduction

Classification of spoken language based on audio signals is an interesting problem that is currently not completely solved. This problem is practical because spoken language classification can be applied in automated telephone systems and translation systems. In addition, the different approaches taken to model languages can reveal patterns among languages that could be useful for studying their underlying linguistics.

The exact formulation of our project is as follows. We pre-select a set of recordings from every language and train different models on them. For testing, we randomly select an equal number of recordings from each language, we attempt to correctly classify its language.

In this paper, we explore a few different features for representing the audio data (Section 2), explore several different machine learning methods for classification (Section 3), analyze our results (Section 4), and consider future improvements (Section 5).

2 Data and Feature Extraction

2.1 Data

The recordings we use come from Voxforge, a website dedicated to collecting transcribed audio samples for use in open source speech recognition research and software[1]. Voxforge has a speech corpus of thousands of audio files submitted by people all over the world in many languages. The audio files we use have a sampling rate of 48kHz and uses 16 bits per sample. We selected ten languages with the most audio recordings for our classification problem: German, Dutch, English, Greek, Spanish, French, Italian, Portuguese, and Hebrew.

We downloaded all of the audio files for these languages and take different subsets to be training and testing data. We divided the files such that the training data and testing data comes from different speakers in order reduce confounding variables and improve the robustness of our system.

2.2 Feature Extraction

The main features we use are Mel-frequency Cepstral Coefficients(MFCC). MFCCs are well studied feature representations used in speech recognition systems[2]. MFCCs are calculated as follows[2].

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the Discrete Cosine Transform(DCT) of the log filterbank energies.
6. The DCT coefficients are the MFCCs

In our classification methods, we divide each recording into 25ms window frames with 5ms overlaps between frames. We use the lower 12 MFCCs as the main feature vectors since that is within the main human speech range[2]. We also calculate the delta and delta-deltas of the main MFCCs and explore adding them as part of the feature vector in the different classification methods.

We use MFCCs to model phonemes of a language because it is extremely difficult to separate actual phonemes from a sound recording. One possible way to do so is to use the air gaps in a sound recording. However, this method is not always reliable, and is prone to errors if someone speaks too quickly. MFCCs are better ways of modeling phonemes. By quantifying phonemes according to how humans actually hear and process the sounds, it is possible to then cluster these feature vectors into their corresponding phoneme groups. If we plot MFCCs on a hyperplane, data from the different phoneme groups should be spatially far from each other.

We use an open source implementation for computing MFCCs and Bark frequencies in our code[3].

3 Classification Methods

3.1 K Nearest Neighbor

The first weak classification method we use is K nearest neighbor. We try a variety different Ks for classification using MATLAB's Nearest Neighbor package[8].

In our neighbor model, we treat each 25ms window frame in a recording as a sample; an average recording from our dataset has around 300 samples. We use a 12 dimensional feature vector consisting of the first 12 MFCCs for each frame. We plot all of the samples from every language, using a different label for each language. For classifying a new audio recording, we classify every frame based on the majority of its K nearest neighbors and take the majority vote of the frames as the final prediction of the recording.

We use 200 audio recordings per language for training, which gives us around 600k total samples in our model.

3.2 Support Vector Machines

For SVMs, we use different feature vectors for computational feasibility and noise reduction. We want to reduce the number of samples to one per recording but also retain the important, distinguishing acoustic features from each recording. We collapse the features of the frames of each recording into one higher dimension "representative" feature vector. The representative feature vector is constructed as follows. First, we compute the 12 MFCCs of each frame in a recording. Then we cluster the MFCCs of all the frames in one recording using Gaussian Mixture Models initialized with K-means into 10 components. We take the top five means (highest mixing proportion) and concatenate them into a 60 dimension feature vector for that record. This construction of the feature vector is meant to contain the most common phonemes of a recording while removing noise through clustering. This representation is reasonable because the most common phonemes for a language tends to be vowels, and each language tends to have a distinct subset of vowels.

The second feature vector we use is constructed similarly. We use 15 clusters for each recording and concatenate the 3 means with the highest and the 3 means with the lowest mixing proportions into a 72 dimension feature vector. This feature vector models taking the most popular and least popular phonemes of a recording. The least popular phonemes of languages are likely to be different, so they should also be useful in distinguishing languages.

Single support vector machines can only be used for binary classification, so we must use combinations of SVMs for multiclass classification. We employ three different methods for multiclass SVM classifications: *one-against-all*, *one-against-one*, and *directed acyclic graph SVM(DACSVM)*.[6,7]

For *one-against-all* multiclass SVM classification, we train 10 SVMs, where each SVM is trained with data from one language(label 1) against data from all the other languages(label 2). For classifying a new sample, we compute its distance(using the kernel function, support vectors, alphas, and bias from the SVM in the equation below) to the margin for each SVM and take pick the prediction with the largest distance.

Distance:

$$\text{abs}\left(\sum_{t \in SV} (\alpha_t y_t K(x_t, x)) + \theta_0\right)$$

For *one-against-one* multiclass SVM classification, we train 45 pairwise SVMs one for every language pair combination. We classify a new sample by taking the majority of the predictions on all of the pairwise SVMs.

For *DACSVM* multiclass SVM classification, we use the same 45 pairwise SVMs as above. Instead of taking the majority for classification, we instead build a decision tree(DAG) of depth 9. A new sample is classified at each node, which indicates where to move down the decision tree. The final classification of the new node is the prediction at the leaf.

We use MATLAB's implementation of SVM in their Statistics Toolbox for training models, which uses the primal/dual forms for solving the quadratic optimization problem to maximize margin[9]. For all of the SVMs, we use the lowest order polynomial kernel that converged for better generalization. We also experiment with using models trained with different slack penalty constants for testing; we tried 1, 0.1, and 0.01. In general, reducing the slack penalty constant lead to SVMs that converged on lower order polynomials, which is expected because we allow for more misclassifications during training.

3.3 Gaussian Mixture Model

The motivation behind using Gaussian mixtures comes from the notion that a language can be defined in terms of discrete phonemes. Each language consists of a set of specific phonemes and a certain distribution of it. Languages may have overlapping phonemes, but the distribution should be different among different languages. Under this assumption, Gaussian mixture model would be a good model for classification. The generative model that we are assuming is a regular Gaussian mixture. Each mixture component represents a single phoneme, and its mixing proportion corresponds to the proportion of

occurrence of that phoneme in the language. We train one Gaussian mixture model for each language. During classification, we take each recording's set of MFCCs (along with its deltas and delta-deltas) and find its likelihood for these models. The recording is classified to the language that has the maximum likelihood.

The EM algorithm is used to classify the data into Gaussian mixtures. The following are the equations for the expectation step and the maximization step [11]

Expectation:

$$\text{Evaluate posterior probabilities: } P^{(l)}(j|t) = P(j|x_t, \theta^{(l)})$$

Maximization:

$$P^{(l+1)}(j) = \frac{\hat{n}(j)}{n}, \text{ where } \hat{n}(j) = \sum_{t=1}^{n} p^{(l)}(j|t)$$

$$\underline{\mu}_j^{(l+1)} = \frac{1}{\hat{n}(j)} \sum_{t=1}^{n} p^{(l)}(j|t)\underline{x}_t$$

$$\Sigma_j^{(l+1)} = \frac{1}{\hat{n}(j)} \sum_{t=1}^{n} p^{(l)}(j|t)(\underline{x}_t - \underline{\mu}_j^{(l+1)})(\underline{x}_t - \underline{\mu}_j^{(l+1)})^T$$

Since the EM algorithm is classifying data points with multiple dimensions, the Gaussian distribution is actually the multivariate Gaussian distribution. It uses means and the covariance matrix as parameters. Hence the probability density for the multivariate Gaussian is a bit different than that of the single variable Gaussian distribution. There are two equations specifying the probability density [10]:

Non-degenerate case:

$$f_x(x_1, \ldots, x_k) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} exp(-\tfrac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

where Σ is the covariance matrix, μ is the mean, Σ^{-1} is the inverse of covariance, and $|\Sigma|$ is the determinant of the covariance

Degenerate case:

$$f(x) = (det^*(2\pi\Sigma))^{-0.5} exp(-0.5 \cdot (x-\mu)' \Sigma^{+}(x-\mu))$$

where det^* gives the pseudo-determinant of the covariance matrix (calculated from the non-zero eigenvalues of the matrix), and Σ^{+} gives the pseudo-inverse of the covariance matrix (Moore-Penrose pseudo-inverse is used in the code).

Using these different probability equations, we can update the EM algorithm accordingly.

The initial implementation of the Gaussian mixture involves using K-means (and K-means++ initialization) for EM's initialization. The K-means algorithm clusters the data into an specified *K* number of

components, and returns *K* number of means, or centroid, for the clusters. These means are the inputs used for initializing the EM algorithm. Variance is then calculated for each cluster by using the points that are the closest to the cluster. This implementation proved to be not useful. One possible reason is that Gaussian mixture is an advanced version of K-means clustering. Since it is similar to K-means, the results that we get from K-means will force the EM algorithm to converge locally, and not at an optimal solution. The classification error rates for this initial implementation were well in the 90% range.

A second implementation takes away the entire K-means algorithm, and instead initializes the EM algorithm with K-means++ method. The K-means++ initialization introduces "smart" initialization by picking the means from the training data using probabilities. The further away a point is from the current means, the more likely it is to be picked. Using this initialization and training using the EM algorithm produces much better results than before. Since the EM algorithm also needs to initialize variances, it uses the current means and, for each mean, calculates the variance using the entire training data set.

There is an obvious problem with this model. It is unclear how many phonemes are present in each language. Although we can find information online regarding the total number of phonemes in a certain language, fewer phonemes are used in these speech samples we acquired. We solved this problem by experimenting with a different number of phonemes, and found 25 or 30 to be the optimal number of phoneme clusters.

3.4 Hidden Markov Model

For Hidden Markov Model we use both normal HMM and continuous density hmm (CDHMM). For both HMMs, we model the states to be phoneme of languages and the emissions/outputs to be the acoustic feature vectors: Mel-frequency Cepstral Coefficients(MFCC) that we extract from the audio recordings. We do not use delta and delta-delta of the MFCCs in the feature vector to reduce the runtime complexity. Moreover, deltas capture the transitions between phonemes which is already modeled by state transitions in HMM. We train an HMM model for each language, and for prediction, we pick the language model that gives the best likelihood.

In the normal HMM, we use K-means with K++ Initializer to cluster the feature vectors (MFCC). Each mean represents each phoneme in the languages, so after applying the K-Mean, we get the observed states. Thus, we explicitly find the initial probability and transitional probability that achieve the maximum likelihood of all the training data. The explicit form for initial probability is the proportion of each state appearing at the beginning of all the observed states for each training input (each audio recording). Similarly, transitional probability is the proportion of times that one state changes to another state in all training data. If the probability is zero (due to lack of appearance), we will set it to 0.0001 so that the log likelihood does not go to negative infinity.

In continuous density HMM, we model it in the similar way as in normal HMM. However, each state will have its own emission probability modeled by a Multivariate Gaussian (parameterized by one mean vector and one covariance matrix), using the 12 dimensional MFCCs that we extract. To reduce the number of parameters, we set all off-diagonal elements of the covariance matrix to zero. Since there is no explicit form for the parameters of the model (initial probability, transitional probability, means and covariance

matrix for each state), we will use expectation-maximization algorithm (EM) to find the best set of parameters that gives the highest likelihood to the training data. For each iteration/update in E and M step, see in [4]. In the EM algorithm, we also use scaled forward-backward algorithm to calculate the posterior probability. We decide to scale the forward and backward probability to prevent both forward and backward probabilities (alpha and beta) from diminishing to zero as the computer cannot handle that many of decimals [5]. For initialization of the parameter, we use randomization to prevent all the states having the same distribution. For initializing initial and transitional probabilities, we randomize them using uniform distribution. For mean and covariance, we randomize around zero and one respectively.

For EM algorithm see [4] for parameter reference
E-Step calculate posterior probability

$$b_i(o_t) = \frac{1}{(2\pi)^{P/2}\prod_{n=1}^{P}\sigma_{in}} e^{-\frac{1}{2}\sum_{n=1}^{P}(\frac{o_{tn}-\mu_{in}}{\sigma_{in}})^2}$$

$$P_\lambda(q_t = i \mid o^l) = \frac{\hat{\alpha}^l_t(i)\hat{\beta}^l_t(i)}{\sum_{i=1}^{N}\hat{\alpha}^l_t(i)\hat{\beta}^l_t(i)}$$

$$P_\lambda(q_t = i,\ q_{t+1} = j \mid o^l) = \hat{\alpha}^l_t(i)a_{ij}\hat{\beta}^l_{t+1}(i)b_j(o^l_{t+1})$$

M-Step update parameters

$$\bar{\pi}_i = \frac{\sum_{l=1}^{K} P_\lambda(q_1 = i \mid o^l)}{K}$$

$$\bar{a}_{ij} = \frac{\sum_{l=1}^{K}\sum_{t=1}^{T_l-1} P_\lambda(q_t = i,\ q_{t+1} = j \mid o^l)}{\sum_{l=1}^{K}\sum_{t=1}^{T_l-1} P_\lambda(q = i \mid o^l)}$$

$$\bar{\mu}_{in} = \frac{\sum_{l=1}^{K}\sum_{t=1}^{T_l} P_\lambda(q_t = i \mid o^l) o^l_{tn}}{\sum_{l=1}^{K}\sum_{t=1}^{T_l} P_\lambda(q_t = i \mid o^l)}$$

$$\bar{\sigma}^2_{in} = \frac{\sum_{l=1}^{K}\sum_{t=1}^{T_l} P_\lambda(q_t = i \mid o^l)\,(o^l_{tn} - \bar{\mu}_{in})^2}{\sum_{l=1}^{K}\sum_{t=1}^{T_l} P_\lambda(q_t = i \mid o^l)}$$

For Scaled Forward-Backward algorithm see [5]

Analysis for CDHMM:

K = number of training input (records)
T = expected number of feature vectors(mfcc) per record
N = number of states
P = dimension of feature vector(mfcc)

Number of parameter to specify the model = Initial probability (N-1) + Transitional probability (N^2-1) + means (NxP) + covariance matrix (NxP) = N^2 + (2P+1)N - 2
Running time: O(KT(N^2)P) from calculating posterior probability for transitional probability

4 Results and Analysis

The following table shows the best results achieved by the different types of classifiers(the table shows error rates):

	1 Nearest Neighbor	SVM	Gaussian Mixture	CDHMM
German	0.56	0.86	0.76	0.87
Dutch	1	0.28	0.95	0.56
Greek	0.48	0.61	0.82	0.3
English	0.98	0.85	0.95	0.79
Spanish	0.92	0.77	1	0.72
French	0.84	0.57	0.822	0.69
Hebrew	0.89552	0.77612	0.75	N/A
Italian	0.64	0.59	0.4	0.76
Portuguese	0.92	0.76	0.31	0.99
Russian	0.8	0.93	0.4	1

Please refer to the charts in Appendix 1 for full experimental testing results.
The overall results obtained from all of these classifiers is lower than what we had hoped. Some of this can be attributed to the fact that nine out of our ten languages come from the Indo-European language family, while Hebrew is in a different family. There are already many linguistic similarities among these languages, which makes it difficult to find very distinctive features. Refer to Appendix 3 for the language tree.

4.1 K Nearest Neighbor
Testing error for K Nearest Neighbor is around 80%, which is only slightly better than random chance(90% since we are classifying into 10 languages). This is expected because languages share many overlapping features. Increasing K has almost no effect on the results.

4.2 Support Vector Machines

Test results for SVMs are about 10% better than nearest neighbor, though it still yields fairly low overall accuracy. Even though the overall testing error is high, the testing error for each language varied drastically, but the variation was, surprisingly, consistent among all of the SVM methods. For example, all of the SVM models perform well for Dutch, meaning that Dutch is more linearly separable than the rest of the languages.

The majority vote *one-against-one* multiclass strategy seems to perform the best for most of the SVMs. This indicates that the data is more separable between 2 languages rather than between one language against the rest of the languages, which makes sense because using data from multiple languages as one label increases the chance of overlapping features with the target language. Even though both majority vote *one-against-one* and *DAGSVM* use pairwise SVMS, one-against-one perhaps performs slightly better because *DAGSVM* penalizes a wrong classification more. In *DAGSVM*, if a recording of language *t* is misclassified on a pairwise SVM that included language *t*, then that recording will just be misclassified, but in *one-against-one*, that recording will still be classified on all the other pairwise SVMs that uses language *t*.

The 2 different feature vectors (60 dimension and 72 dimension) produce similar results for each language. Dutch performs uniformly relatively well while Hebrew performs uniformly badly across all the different variations.

A slack penalty constant of 0.1 seems to perform better than 1 and 0.01. This could be due to the fact that allowing more slack tends to reduce overfitting of the training data, making the classifier more generalizable. A low penalty that allows for more slack might have led to a non optimal margin.

In conclusion, SVM generally does not perform well for this problem because the features we use for these languages are probably not linearly separable due to linguistic similarities among the languages.

4.3 Gaussian Mixture Model

The Gaussian mixture model works fairly well on Portuguese but not on Greek. There are a few possible reasons for this discrepancy. It is possible that the feature vectors used do not separate the phonemes well enough into Gaussian like clusters. For the Gaussian mixture model training, the MFCCs, its deltas, and the delta-deltas are all used to identify phonemes. The delta and delta-deltas are used to indicate relationships of the phonemes that come before and after the phoneme in question, thereby encoding some information about the phoneme transition. Therefore, it is not purely identifying a phoneme independent of its dependencies. This was thought to be better because, unlike the Markov model, the Gaussian mixture does not take into account the existing relationship among phonemes. By combining information from the deltas and delta-deltas, it is possible to classify based on phonemes not only by itself, but also by its neighbors. Therefore, K-means clustering would not purely cluster phonemes. However, it is entirely possible that, for a Gaussian mixture, it is sufficient to use the MFCCs only for classification. If the deltas and delta-deltas do not provide sufficiently different information, it may actually falsely classify different MFCCs as coming from the same phoneme cluster.

It is even possible that the underlying clusters are not Gaussian at all. It is also possible that the underlying mixture distribution is not Gaussian at all. Depending on a language's speech features, the speech variations among speakers are most likely not distributed the same way. Even though multivariate Gaussian distribution is one of the most widely seen distributions, it might not necessarily be the best distribution to use for speech classification.

4.4 Hidden Markov Model

For normal HMM, the best testing error is about 75%. One of the reason that makes this model have a low accuracy is that we might not be predicting the observed state well enough. In other words, using K-Means with the current window size of MFCC does not capture the acoustic feature of each different phonemes. However, it is as expected that using too low or too high for the number of clusters (K) will reduce the accuracy. The ideal K for all these languages is around 40 to 50. The other reason might be that all these languages are all Indo-European so their phoneme will have similar acoustic feature vector. Hence, the K-means will cluster these little different phonemes as the same.

For continuous density HMM, the best testing error is around 74% which is relatively low compare to random guess (90%). One major problem with this model is the huge number of parameters to specify the model and the long runn time (Section 3.4 - analysis). We do not have sufficient time to train the model in order to compensate for the big number of parameters. Even though we use up to 500 records, we train only one fourth of each record (from 3/8 to 5/8) and we set the number of iteration to 5 or 10 of the EM step instead of waiting until it converges. Moreover, we cannot increase number of states to 40 as stated in the normal HMM. As we can see from the testing result that increasing the number of states yields better accuracy. Last but not least, the acoustic feature vectors of these language are similar as they come from the same family language.

*We do not train Hebrew language because it does not have enough training data compare to other language.

5 Future Work

There are many different directions we can take to improve classification results. Since all of our methods yielded low accuracy, one method we can try is bossting. We would use weak classifiers in combination with an n-ary boosting algorithm to boost our results, though it will be more involved for classifying multiple classes. Another classifier often used for speech recognition, that we could explore in the future, is neural network. Neural network could potentially be used to pre-process the MFCCs instead of K-means.

The other direction to explore is using different acoustic features to representing the data. We only used MFCC and its derivatives in our classifications, but there are other acoustic features such as PLP, Bark, and RASTA that are worth trying [3].

References

[1] *Free speech.. recognition (linux, window, and mac)*. (n.d.). Retrieved from

http://www.voxforge.org/
[2]Lyons, J. [Web log message]. Retrieved from http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
[3] Ellis, D. (n.d.). *Plp and rasta (and mfcc, and inversion) in matlab using melfcc.m and invmelfcc.m*. Retrieved from http://labrosa.ee.columbia.edu/matlab/rastamat/
[4]*Tutorial on hidden markov models*. (n.d.). Retrieved from http://mplab.ucsd.edu/tutorials/hmm.pdf
[5]*Hmms in a nutshell*. (n.d.). Retrieved from http://www.cs.tut.fi/courses/SGN-4106/nutshell.pdf
[6]Keerthi, S. (n.d.). *Which is the best multiclass svm method?*. Retrieved from http://research.yahoo.com/files/multiclass_mcs_kaibo_05.pdf
[7]*Large margin DAGs for multiclass classification*. (n.d.). Retrieved from http://www.wisdom.weizmann.ac.il/~bagon/CVspring07/files/DAGSVM.pdf
[8]*Nearest neighbors*. (n.d.). Retrieved from http://www.mathworks.com/help/stats/nearest-neighbors.html
[9]*Support vector machines (svm)*. (n.d.). Retrieved from http://www.mathworks.com/help/stats/support-vector-machines-svm.html
[10] Do, C. (n.d.). *The multivariate gaussian distribution*. Retrieved from http://cs229.stanford.edu/section/gaussians.pdf
[11] Jaakkola, Tommi. (n.d.) *Mixture of Gaussians*. Retrieved from http://courses.csail.mit.edu/6.867/fall_2009/lectures/lecture14_notes.pdf

Appendix 1 [Code]

Please reference https://github.com/hungryha/6867FinalProject for our code.

Appendix 2 [Detailed Results]

Testing Error	1 nearest neighbor	5 nearest neighbor	10 nearest neighbor	SVM 13	Gaussian Mixture Model	CDHMM	normal HMM (best)
de(German)	0.56	0.55	0.55	0.86	0.76	0.87	0.63
dutch	1	1	1	0.28	0.95	0.56	0.6
el(Greek)	0.48	0.5	0.39	0.61	0.82	0.3	0.97
english	0.98	0.94	0.96	0.85	0.95	0.79	1
es(Spanish)	0.92	0.88	0.86	0.77	1	0.72	1
french	0.84	0.88	0.87	0.57	0.822	0.69	0.32
he(Hebrew)	0.89552	0.89552	0.89552	0.77612	0.75		
italian	0.64	0.88	0.87	0.59	0.4	0.76	0.31
portuguese	0.92	0.97	0.96	0.76	0.31	0.99	0.93
russian	0.8	0.82	0.8	0.93	0.4	1	1
overall	0.8004	0.8294	0.8128	0.697	0.7162	0.7422	0.7511
Feature Vector	MFCC per frame	MFCC per frame	MFCC per frame	MFCC concatenation	MFCC, delta,	MFCC per frame, 25	MFCC per frame,

	(each frame is one sample)			per recording (see section 3.2)	delta-deltas per frame	number of states,	K-mean, K =40
Feature Vector Dimension	12	12	12	60	36	12	12
Training Size (per language)	200 recordings per language	200 recordings per language	200 recordings per language	2000 recordings per language, 658 for el, 200 for he, 700 for russian	300 recordings per language, Hebrew is 200 recordings	100 recordings per language	200 recordings per language
Test Size (per language)	100 recordings per language, 67 for he	100 recordings language, 67 for he	100 recordings language, 67 for he	100 recordings language, 67 for he	100 recordings per language	100 recordings per language	100 recordings per language

	SVM 1	SVM 2	SVM 3	SVM 4	SVM 5	SVM 6	SVM 7	SVM 8	SVM 9
de(German)	0.95	0.94	0.97	0.88	0.97	0.99	0.96	0.99	0.97
dutch	0.525	0.49	0.82	0.14	0.27	0.36	0.18	0.23	0.41
el(Greek)	0.615	0.67	0.76	0.79	0.85	0.69	0.71	0.98	0.77
english	0.82	0.68	0.86	0.86	0.64	0.86	0.89	0.99	0.85
es(Spanish)	0.755	0.88	0.77	0.7	0.96	0.87	0.78	0.6	0.8
french	0.695	0.96	0.63	0.72	0.67	0.69	0.71	0.88	0.73
he(Hebrew)	0.91045	0.89552	0.79104	0.8209	0.95522	0.71642	0.77612	0.97015	0.64179
itallian	0.76	0.95	0.76	0.64	0.91	0.52	0.53	0.44	0.6
portuguese	0.75	0.44	0.72	0.77	0.56	0.71	0.67	0.38	0.73
russian	0.775	0.94	0.82	0.85	0.98	0.88	0.82	0.97	0.89
overall	0.7445	0.7808	0.7901	0.7135	0.7704	0.7291	0.7001	0.7353	0.7425
slack penalty	1	1	1	0.1	0.1	0.1	0.01	0.01	0.01
multiclass strategy	one against one	one against all	DAGSVM	one against one	one against all	DAGSVM	one against one	one against all	DAGSVM
Feature Vector Dimension	72	72	72	72	72	72	72	72	72

	SVM 10	SVM 11	SVM 12	SVM 13	SVM 14	SVM 15	SVM 16	SVM 17	SVM 18
de(German)	0.87	0.99	0.96	0.86	0.99	0.98	0.94	0.97	0.95
dutch	0.27	0.21	0.51	0.28	0.23	0.64	0.14	0.14	0.48
el(Greek)	0.64	0.68	0.71	0.61	0.98	0.71	0.75	0.98	0.79
english	0.86	0.74	0.78	0.85	0.99	0.82	0.81	0.8	0.77
es(Spanish)	0.81	0.93	0.81	0.77	0.6	0.86	0.79	0.72	0.78
french	0.65	0.84	0.58	0.57	0.88	0.64	0.67	0.62	0.67
he(Hebrew)	0.92537	0.86567	0.79104	0.77612	0.97015	0.85075	0.76119	1	0.77612
itallian	0.68	0.94	0.64	0.59	0.44	0.46	0.57	0.44	0.57
portuguese	0.78	0.56	0.73	0.76	0.38	0.81	0.8	0.52	0.83
russian	0.87	0.85	0.86	0.93	0.97	0.94	0.89	1	0.88
overall	0.7291	0.757	0.7353	0.697	0.7353	0.7684	0.7104	0.7094	0.7487
Feature dim	60	60	60	60	60	60	60	60	60
slack penalty	1	1	1	0.1	0.1	0.1	0.01	0.01	0.01
multiclass strategy	one against one	one against all	DAGSVM	one against one	one against all	DAGSVM	one against one	one against all	DAGSVM

Appendix 3 [Language Tree]