SpokenLanguages2 - report

Personal Information

Name: Catalin-Stefan Tiseanu
Topcoder handle: CatalinT
Email-address: ctiseanu@gmail.com
Final submission: sub27.csv (output only task), https://www.dropbox.com/s/29onhpuuikz6ajc/sub_new_1.csv?dl=1

Approach Used

SUMMARY

I used a very simple, yet highly effective, approach. Basically, I trained 176 Gaussian Mixture Models with 2048 mixtures and 62 features (one GMM for each language). Finally, I used Logistic Regression to calibrate the individual language scores.

I pretty much followed this: http://www.egr.msu.edu/~deller/icslp-submission-copy.pdf

In terms of steps:
(1) Use sox to convert the mp3's to wav's of 16khz, 16 bit and 1 channel
(2) Use sphinx_fe (from CMU-Sphinx, BSD license) to extract 13 Mel Frequency Cepstral Coefficients (MFCC) features
(3) Add shifted delta coefficients (SDC) using 7-1-3-7 parametrization
(4) At this point I reduced every wav to a 999 x 62 feature array. The 999 comes from the fact that the mfcss are computed every 10 ms and one frame is lost in the conversion process. The 62 comes from 13 mfcc featurex per frame + 7 x 7 sdc features per 10ms frame
(5) Collect every feature vector from every wav by language, therefore obtaining a ((376 * 999) x 62) feature matrix for each language
(6) Train a Gaussian Mixture Model with 2048 mixtures for each language feature matrix, using https://github.com/juandavm/em4gmm (GPL license). The training was done using the standard EM algorithm, with a 0.001 tolerance for the log-likelihood increase at each step
(7) Finally, calibrate the individual language predictions using Logistic Regression (sklearn.linear_model.LogisticRegression(C=1e5, random_state = 0)

Prediction takes 15 seconds on a m4.xlarge machine.
Training (and prediction on training set) takes a lot :), around 80 hours on a c4.8xlarge machine.

APPROACHES CONSIDERED

From the start, I considered a GMM based approach, based on the decent performance in the first iteration of the contest, SpokenLanguages. The huge difference between that contest and this one was in terms of the data provided:

(1) There were far more samples per language for training (376 vs 120)
(2) From what I could tell the samples were from male samples only, which helped **immensely**

From the get go, GMM performed admirably, even with only 13 features and 64 mixtures. From that point, it was a matter of training larger models fast enough (which I would say was the biggest challenge for me in this competition).

STEPS TO APPROACH ULTIMATELY CHOSEN

See the above text for info.

My whole code is in **main.py**

First, there is an init step to be done, detailed in **compute_general_dict**, which compute language→id mappings and so on.

There is a helper method **predict_first_K** which predicts the results for the first K samples from the testing dataset, in terms of alphabetical order. This is useful in order to replicate the results.

The main function is **predict_mp3**

I will detail the workings of each step:

(1) I used the following **sox** command "sox INITIAL_MP3 -R -r 16000 -b 16 -c 1 TMP_WAV"
(2) I used the following **sphinx_fe** command "sphinx_fe -i TMP_WAV -o TMP_MFC -ofmt text"
(3) I used the **compute_sdc** python function to compute a 7-1-3-7 SDC feature configuration, for 7 x 7 additional features, reaching 62 in total
(4) The resulting 999 x 62 feature matrix is converted into a .gz format suitable for the gmmclass binary (from the em4gmm library detailed above)
(5) Each of the 176 models is called on the gz file via "server = subprocess.call(["gmmclass", "-d", feature_62_gz_filename, "-m", "emm/2048_62_emm_{}".format(i), "-t", "4"], stdout = output)"
(6) The predictions are collected into a 176 element vector
(7) Finally, the logistic regression model is called with predict_proba via "probs = logreg.predict_proba(X_inter)"
(8) From there on, it's only a problem of sorting the probabilities in a non-ascending order and writing out the top 3 languages

Finally, in order to train the model, **train_models** is used. It is similar to **predict_mp3**, with the caveat that gmmtrain is used instead of gmmclass, and obviously the logistic regression is fit (instead of predict_proba).

The steps are detailed below:

(1) In the first step, feature are extracted as in the predict stage, and a feature matrix is assembled for each language (by vertically stacking the feature matrices for all samples in the language, i.e the 999 x 62 matrices).
(2) "gmmtrain -d FEATURE_MATRIX.gz -m 2048_62_emm_LANG -s 0.001 -t 4: is called for each LANG
(3) Once all models are trained, prediction is done (exactly as in the predict stage) for the training dataset
(4) Finally, a sklearn.linear_model.LogisticRegression(C=1e5, random_state=0) is fitted over the corresponding language probabilities. What this does is calibrate the predictions of each individual language model (i.e toning down optimistic language models).

OPEN SOURCE RESOURCES AND TOOLS USED

I made heavy use of Python and related python machine learning libraries. Specifically:

- Python
- Numpy, pandas, sklearn for data processing
- multiprocessing
- joblib for model serialization
- sox for converting from mp3 to wav

Additionally, I used two highly important open source libraries:
(1) CMU Sphinx
(2) https://github.com/juandavm/em4gmm for GMM training.

I used (1) for extracting the 13 mfcc features, and (2) for the speed boost over GMM's from sklearn.

ADVANTAGES AND DISADVANTAGES OF THE APPROACH CHOSEN

The main advantage of the above approach is it's simplicity (only one model + one calibration model are involved) and it's applicability to pretty much any good dataset (good means lots of samples and same-sex speakers).

The biggest disadvantage is the time requirement for prediction (which is actually significantly slower than the training time). It took one c4.8xlarge machine 16 hours to predict the results for the test samples (12320), and 4 machines the same amount of time for the training set prediction (which I needed in order to calibrate the prediction using logistic regression).

COMMENTS ON LIBRARIES

I used a fairly standard Python stack for machine learning. The 2 libraries mentioned above were crucial for getting quality features and training them in a quick enough time.

COMMENTS ON OPEN SOURCE RESOURCES USED

I didn't use any external dataset.

SPECIAL GUIDANCE GIVEN TO ALGORITHM BASED ON TRAINING

The only guidance was given to the LogisticRegression C parameter, namely C=1e5. This is actually the first one I tried, and it performed the best. It should be noted that changing this parameter would not meaningfully affect the result.

POTENTIAL IMPROVEMENTS TO YOUR ALGORITHM

It should be said from the get go that the approach I used is definitely not the state-of-the-art. This distinction goes to i-vectors trained from an universal background model GMM (UBM-GMM), with a probabilistic LDA (PLDA) fusion mechanism. However, the main advantages of this method vs the one I chose consist in higher robustness to noise, which is basically non-existant on the dataset

for this competition. I doubt that using the i-vector approach could have improved on the result significantly.