ST-RU-CT--ED

java structured learning package

Vowel Duration Measurement Tutorial - by Adiyoss

June 30, 2015

1 Introduction

In this tutorial we will demonstrate how to use and add new task to the StructED package. We will give code examples for the Task-Loss, Predict - Inference and Feature Functions interfaces, as well as update for the Factory class. For this tutorial we will use a structured prediction problem called the *Vowel Duration Measurement*. We provide a small subset from the original data we use in our experiments.

2 Vowel Duration Measurement

In the problem of *vowel duration measurement* we are provided with speech signal which includes exactly one vowel, preceded and followed by consonants.

Our goal is to predict the start and end time of the vowel as accurate as possible. We represent each speech signal as a set of acoustic features $\overline{x} = (x_1, x_2,...,x_T)$ where each x_i $(1 \leq \text{i} \leq \text{T})$ is a *d*-dimensional vector. We denote the domain of the feature vectors by $X \subset \mathbb{R}^d$. Since we are dealing with speech, the length of the signals varies from one file to another, thus T is not fixed. We denote by X^* the set of all finite lengths over X. More detailed information about the features and the feature functions is descried later on this paper.

In addition, each speech segment is accompanied with the vowel start and end time, we denote by the pair $t_{onset} \subset \mathcal{T}$ and $t_{offset} \subset \mathcal{T}$ the onset and offset times of the vowel respectively, where $\mathcal{T} = \{1, ..., T\}$, for brevity let us denote (t_{onset}, t_{offset}) as $\overline{t}$.

To sum up, our goal is to learn a function, denoted f, which takes as input a speech signal $\overline{x}$ and returns the sequence $\overline{t}$ which is the start and end time of the vowel. Meaning, f is a function from the domain of all possible CVC speech segments X^* to the domain of all possible onset and offset pairs $\mathcal{T}^2$.

We provide a small subset from the data set we use in our experiment, it can be found under db/ directory.

3 The Data

Before we run into the code, let's look at the data for a second. Each data example contains 22 features that were extracted every 5 ms. The first 5 features refer to a short-time Fourier transform (STFT) taken with a 5 ms Hamming window: the STFT itself; the log of the total spectral energy, E_{total}; the log of the energy between 50 and 1000 Hz, E_{low}; the log of the energy above 3000 Hz, E_{high}; and the Wiener entropy, H_{wiener}, a measure of spectral flatness:

$$H_{wiener}(t) = \log \int |P(f,t)^2| df - \int \log |P(f,t)^2| df, \tag{1}$$

where $P(f,t)$ is the STFT of the signal at frequency f and time t. The sixth feature is Auto Correlation. The seventh feature, P_{max}, is extracted from the signal itself: the maximum of the power spectrum calculated in a region from 6 ms before to 18 ms after the frame center. The eighth feature is the 0/1 output of a voicing detector based on the RAPT pitch tracker[1], smoothed with a 5 ms Hamming window. The ninth feature is the number of zero crossings in a 5 ms window around the frame center. The 10-16 are high level feature that were measured using the predictions of a phoneme classifier. The 10-13 features are an indicator functions, indicating if there is a vowel, nasal, glide or sil phoneme respectively. The 14-16 features are the maximum likelihood of a vowel, nasal or glide phoneme in the current time frame respectively. The 17-20 features are based on the distance between frames of the acoustical signal at two sides of phoneme boundaries as suggested by a phoneme start time t. The distance measure we employ, denoted by d, is the Euclidean distance between feature vectors. Our underlying assumption is that if two frames, x_i and x_{i+1}, are derived from the same phoneme then the distance $d(x_i, x_{i+1})$ should be smaller than if the two frames are derived from different phonemes. Formally,

$$\phi_j(\overline{x}, location) = d(x_{location-j}, x_{location+j}), \qquad j \in \{1,2,3,4\}.$$

The 21-22 features are the F1 and F2 formants. The formant frequencies were measured automatically using the burg algorithm from Praat.

In order to support this we need also to inherit the abstract class Example and to support also matrix data types. We already implemented it in our code, this data structure is called *Example2D*.

To use this data structure we also need to implement an appropriate reader to populate this data. We already implemented this too, this reader is called *VowelDurationReader*.

This too implementations are stored in project source code directory. By exploring those source codes you can see that adjusting StructED to support new data format or different data structure is not much of a work.

4 The Code

Here, we present what classes do we need to add and what interfaces do we need to implement. We provide the source code for each class and interface.

4.1 Task Loss

We now add a task loss class. Every task loss class should implement the TaskLoss interface. In our problem settings we define the task loss as:

$$\gamma(\bar{t}, \bar{t}^{'}) = max\{0, |\bar{t}_{onset} - \bar{t}^{'}_{onset}| \geq \epsilon_{onset}\} + max\{0, |\bar{t}_{offset} - \bar{t}^{'}_{offset}| \geq \epsilon_{offset}\} \quad (2)$$

The above function measures the absolute difference between the predicted and the correct vowel onset and offset. We take into account that the manual annotations are not exact, hence we allow a mistake of ϵ_{onset} and ϵ_{offset} at the vowel onset and offset respectively. This way, only the examples which their prediction is greater then ϵ_{onset} or ϵ_{offset} are penalized.

For this we create new java class inside the TaskLoss package, this class should implement the TaskLoss interface. The code for the implementation of this class is attached to this tutorial or can be found at the tutorial folder.

4.2 Predict - Inference

We now add a predict/IInference class. Every predict class should implement the Prediction interface. In our problem settings the prediction will be brute force, we will run over all the possible onsets and offsets and predict the one that maximizes $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$, but we still make a few assumptions to make our code go faster.

We assume that there is a minimum and maximum vowel length possible, denoted by MIN_VOWEL and MAX_VOWEL const respectively. Moreover we assume that the actual speech signal is never placed at the start of the .wav signal, hence we start searching after MIN_GAP_END const.

In general we need the prediction class to implement the following:

$$\hat{\boldsymbol{y}}_{\boldsymbol{w}}(\boldsymbol{x}) = \underset{\boldsymbol{y} \in \mathcal{Y}}{\operatorname{argmax}} \ \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \quad (3)$$

and,

$$\hat{\boldsymbol{y}}_{\boldsymbol{w}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}} \in \mathcal{Y}}{\operatorname{argmax}} \ \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \hat{\boldsymbol{y}}) + \ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) \quad (4)$$

For this we create new java class inside the Prediction package, this class should implement the Prediction interface. The code for the implementation of this class is attached to this tutorial or can be found at the tutorial folder.

4.3 Feature Functions

We now add the feature functions. We can divide our *feature functions* into three types:

- **Type 1:** $\Delta(location, window - size, feature)$

  While exploring the data we noticed a rapid increase in certain features when the vowel onset occurs and rapid decrease when the vowel offset occurs. As a result we implemented the Δ *feature function* to collect this data. This function computes the difference between the mean of the signal of the given feature, before and after the location parameter in range of window-size parameter.

- **Type 2:** $\mu(location_{start}, location_{end}, window - size, feature)$

  Besides the rapid change in the vowel onset and offset, we've also notice that the mean of signal in certain features between the onset and offset is greater then the mean of signal before and after. Hence, we implemented the μ function. This function computes the difference between the mean of the signal of a given feature from $location_{start}$ to $location_{end}$, to the mean of a signal before $location_{start}$ or after $location_{end}$ in range of window-size parameter.

- **Type 3:** Gamma and Gaussian Distribution over the data

  Since our data is distributed somewhere between Gamma and Gaussian distribution, we competed the probability of a given vowel length to occur as for Gamma and Gaussian distributions.

Now we can create new java class inside the FeatureFunction package, this class should implement the PhiConverter interface. The code for the implementation of this class is attached to this tutorial or can be found at the tutorial folder.

You are more then welcome to play with those feature functions or maybe add new one to reach better results.

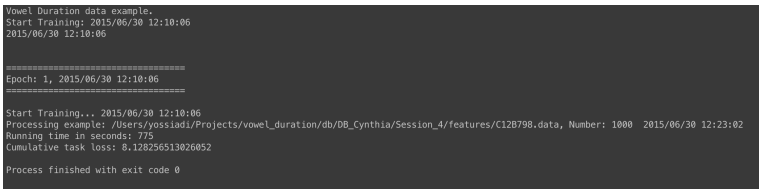
5 Running The Code

We now can run the StructEDModel with the interfaces we have just implemented, all we have left to do is to choose the model. The code should look like this:

```
// create the model
StructEDModel vowel_model = new StructEDModel(W, new DirectLoss(), new
    TaskLossVowelDuration(), new InferenceVowelDurationData(), null, new
    FeatureFunctionsVowelDuration(), arguments);
// train
vowel_model.train(vowelTrainInstances, task_loss_params, null, epochNum, isAvg);
// predict
```

```
vowel_model.predict(vowelTestInstances, null, numExamples2Display);
```

The output should look like this:



Hope you had fun!

GOOD LUCK!

References

[1] Talkin, David, *A robust algorithm for pitch tracking (RAPT)*, Speech coding and synthesis 1995, Vol 495, Page 518.