ST-RU-CT--ED

java structured learning package

Vowel Duration Measurement Tutorial - by Adiyoss

February 24, 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 $\leq$ i $\leq$ 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 $\bar{t}$.

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 $\bar{t}$ 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, we put the following code in it:

```
package BL.TaskLoss;

import Constants.Consts;

import java.util.List;

public class TaskLossVowelDuration implements TaskLoss {

   @Override
    //max{0, |ys - ys'| - epsilon} + max{0, |ye - ye'| - epsilon}
   public double computeTaskLoss(String predictClass, String actualClass,
       List<Double> params) {
        try {
           Double epsilon_onset = params.get(0);
           Double epsilon_offset = params.get(1);

           String predictValues[] =
               predictClass.split(Consts.CLASSIFICATION_SPLITTER);
           String actualClassValues[] =
               actualClass.split(Consts.CLASSIFICATION_SPLITTER);

           double predictResStart = Double.parseDouble(predictValues[0]);
           double actualResStart = Double.parseDouble(actualClassValues[0]);
```

```
            double predictResEnd = Double.parseDouble(predictValues[1]);
            double actualResEnd = Double.parseDouble(actualClassValues[1]);

            double diffStart = Math.abs(predictResStart - actualResStart);
            double diffEnd = Math.abs(predictResEnd - actualResEnd);

            //subtract the epsilon
            double absRes = 0;
            if(diffStart >= epsilon_onset)
               absRes += diffStart;
            if(diffEnd >= epsilon_offset)
               absRes += diffEnd;

            //get the max from the absolute result minus epsilon and 0
            return absRes;

         } catch (Exception e){
            e.printStackTrace();
            return 0;
         }
   }
}
```

4.2 Predict - Inference

We now add a predict/inference 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:

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

and,

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

For this we create new java class inside the Prediction package, this class should implement the Prediction interface, we put the following code in it:

```
package BL.Prediction;

import BL.ClassifierData;
import Constants.Consts;
import Constants.ErrorConstants;
import Data.Entities.Example;
import Data.Entities.PredictedLabels;
import Data.Entities.Vector;
import Data.Logger;
import Helpers.Comperators.MapValueComparatorDescending_IntKey;
import Helpers.MathHelpers;
import Helpers.ModelHandler;

import java.util.TreeMap;

public class PredictionVowelDurationData implements Prediction{

    //predict function
    //argmax(yS,yE) (W*Phi(Xi,yS,yE)) + Task Loss
    //this function assumes that the argument vector has already been converted
        to phi vector
    //return null on error
    public PredictedLabels predictForTrain(Example vector, Vector W, String
        realClass, ClassifierData classifierData, double epsilonArgMax)
    {
        try{
            double maxVal = 0;
            String maxLabel = "";
            boolean isFirst = true;

            //validation
            if(vector.sizeOfVector<=0)
            {
                Logger.error(ErrorConstants.PHI_VECTOR_DATA);
                return null;
            }

            //loop over all the classifications of this specific example
            for(int i=Consts.MIN_GAP_START ;
                i<vector.sizeOfVector-(Consts.MIN_GAP_END) ; i++)
            {
                for(int j=i+Consts.MIN_VOWEL ; j<i+Consts.MAX_VOWEL ; j++)
                {
                    if(j>vector.sizeOfVector-(Consts.MIN_GAP_END))
                        break;
                    Example phiData =
                        classifierData.phi.convert(vector,(i+1)+Consts.CLASSIFICATION_SPLITTER+(j+1),c
                    //multiple the vectors
                    double tmp = MathHelpers.multipleVectors(W,
                        phiData.getFeatures());

                    if(epsilonArgMax != 0){
                        //add the task loss
                        tmp +=
                            epsilonArgMax*classifierData.taskLoss.computeTaskLoss((i+1)+Consts.CLASSIF
                            realClass, classifierData.arguments);
                    }

                    if(isFirst) {
                        maxLabel = (i + 1) + Consts.CLASSIFICATION_SPLITTER + (j +
                            1);
                        maxVal = tmp;
                        isFirst = false;
                    }
                    else if(tmp > maxVal) {
                        maxLabel = (i + 1) + Consts.CLASSIFICATION_SPLITTER + (j +
                            1);
                        maxVal = tmp;
                    }
                }
            }

            PredictedLabels result = new PredictedLabels();
            result.put(maxLabel, maxVal);

            return result;

        } catch (Exception e){
            e.printStackTrace();
            return null;
        }
    }

    public PredictedLabels predictForTest(Example vector, Vector W, String
        realClass, ClassifierData classifierData, int returnAll)
    {
        return predictForTrain(vector, W, realClass, classifierData ,0);
    }
}
```

If you wish to use it from a different multi-class problem all you need to do is just change the numOfClass parameter at the top from 10 to the right number of classes.

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, we put the following code in it:

```java
package Data.FeatureFunctions;

import BL.Kernels.Kernel;
import Constants.Consts;
import Data.CacheVowelData;
import Data.Entities.Example;
import Data.Factory;
import Data.Entities.Vector;
import Data.Logger;
import Helpers.ConverterHelplers;
import Helpers.MathHelpers;
import jsc.distributions.Gamma;
```

```
public class PhiVowelDurationConverter implements PhiConverter {

    int sizeOfVector = 42;
    final int win_size_1 = 1;
    final int small_offset = 1;
    final int offset_20 = 4;
    final int offset_30 = 6;
    final int offset_40 = 8;
    final int offset_50 = 10;
    final int offset_200 = 40;
    final int offset_180 = 36;
    final int offset_160 = 32;
    final int win_size_5 = 1;
    final int win_size_10 = 2;
    final int win_size_15 = 3;
    final int win_size_20 = 4;
    final int win_size_30 = 6;
    final int win_size_40 = 8;
    final int win_size_50 = 10;
    final double NORMALIZE = 0.01;
    final double NORMALIZE_BIG = 0.001;

    final int SHORT_TERM_ENERGY = 0;
    final int TOTAL_ENERGY = 1;
    final int LOW_ENERGY = 2;
    final int HIGH_ENERGY = 3;
    final int WIENER_ENTROPY = 4;
    final int AUTO_CORRELATION = 5;
    final int PITCH = 6;
    final int VOICING = 7;
    final int ZERO_CROSSING = 8;
    final int IS_VOWEL = 9;
    final int IS_NAZAL = 10;
    final int IS_GLIDE = 11;
    final int IS_SIL = 12;
    final int SUM_VOWELS = 13;
    final int SUM_NAZALS = 14;
    final int SUM_GLIDES = 15;
    final int MFCC_1 = 16;
    final int MFCC_2 = 17;
    final int MFCC_3 = 18;
    final int MFCC_4 = 19;
    final int F_1 = 20;
    final int F_2 = 21;
```

```
@Override
//return null on error
public Example convert(Example example, String label, Kernel kernel) {

    try{
        Example newExample = Factory.getExample(0);

        String labelValues[] = label.split(Consts.CLASSIFICATION_SPLITTER);
        Vector phiFeatures = new Vector();

        //convert full vector
        if(ConverterHelplers.tryParseInt(labelValues[0])) {

            int start = Integer.parseInt(labelValues[0]);
            int end = Integer.parseInt(labelValues[1]);

            //=================calculate the features=================//
            //=========Difference 5 and 10 frames from location=======//
            int loc=0;

            //====Short Term Energy====//
            phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
                SHORT_TERM_ENERGY));//short term energy, end location - 30
                window
            loc++;
            phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
                SHORT_TERM_ENERGY));//short term energy, end location - 30
                window
            loc++;

            //====Total Energy====//
            phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
                TOTAL_ENERGY));//total energy, start location - 20 window
            loc++;
            phiFeatures.put(loc, calculateDiff(example, win_size_15, start,
                TOTAL_ENERGY));//total energy, start location - 30 window
            loc++;
            phiFeatures.put(loc, Math.abs(calculateDiff(example, win_size_10,
                end, TOTAL_ENERGY)));//total energy, start location - 20 window
            loc++;
            phiFeatures.put(loc, Math.abs(calculateDiff(example, win_size_20,
                end, TOTAL_ENERGY)));//total energy, start location - 30 window
            loc++;
            phiFeatures.put(loc, calculateDiff(example, win_size_10, end -
                offset_50, TOTAL_ENERGY));//total energy, start location - 30
                window
```

```
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end -
    offset_50, TOTAL_ENERGY));//total energy, start location - 30
    window
loc++;

//====Low Energy====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    LOW_ENERGY));//low energy, start location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    LOW_ENERGY));//low energy, start location - 30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    LOW_ENERGY));//low energy, start location - 40 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    LOW_ENERGY));//low energy, end location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    LOW_ENERGY));//low energy, end location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    LOW_ENERGY));//low energy, end location - 30 window
loc++;

//====High Energy====//
phiFeatures.put(loc, Math.abs(calculateDiff(example, win_size_10,
    end, HIGH_ENERGY)));//high energy, end location - 20 window
loc++;
phiFeatures.put(loc, Math.abs(calculateDiff(example, win_size_20,
    end, HIGH_ENERGY)));//high energy, end location - 30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end -
    offset_50, HIGH_ENERGY));//total energy, start location - 30
    window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end -
    offset_50, HIGH_ENERGY));//total energy, start location - 30
    window
loc++;

//====Wiener Entropy====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    WIENER_ENTROPY));//wiener entropy, start location - 20 window
loc++;
```

```
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    WIENER_ENTROPY));//wiener entropy, start location - 30 window
loc++;
phiFeatures.put(loc, Math.abs(calculateDiff(example, win_size_10,
    end, WIENER_ENTROPY)));//wiener entropy, start location - 20
    window
loc++;
phiFeatures.put(loc, Math.abs(calculateDiff(example, win_size_20,
    end, WIENER_ENTROPY)));//wiener entropy, start location - 30
    window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end -
    offset_50, WIENER_ENTROPY));//total energy, start location -
    30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end -
    offset_50, WIENER_ENTROPY));//total energy, start location -
    30 window
loc++;

//===========Auto Correlation==========//
phiFeatures.put(loc, calculateDiff(example, win_size_5, start,
    AUTO_CORRELATION));//auto correlation, start location - 5
    window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    AUTO_CORRELATION));//auto correlation, start location - 10
    window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_15, start,
    AUTO_CORRELATION));//auto correlation, start location - 15
    window
loc++;

//==========Difference 5 and 15 frames from start=========//
//====Pitch====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    PITCH));//pitch, start location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    PITCH));//pitch, start location - 30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    PITCH));//pitch, start location - 20 window
loc++;
```

```
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    PITCH));//pitch, start location - 30 window
loc++;

//====Voicing====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    VOICING));//voicing, start location - 15 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    VOICING));//voicing, start location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    VOICING));//voicing, start location - 30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    VOICING));//voicing, end location - 30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    VOICING));//voicing, end location - 30 window
loc++;

//====Zero-Crossing====//
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    ZERO_CROSSING));//zero-crossing, start location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    ZERO_CROSSING));//zero-crossing, start location - 30 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    ZERO_CROSSING));//zero-crossing, end location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    ZERO_CROSSING));//zero-crossing, end location - 30 window
loc++;

//=================Phoneme-Classifier==================//
//====VOWELS - INDICATOR====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    IS_VOWEL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    IS_VOWEL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    IS_VOWEL));
loc++;
```

```
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    IS_VOWEL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    IS_VOWEL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    IS_VOWEL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_40, end,
    IS_VOWEL));
loc++;

//====NASAL - INDICATOR====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    IS_NAZAL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    IS_NAZAL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    IS_NAZAL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    IS_NAZAL));
loc++;

//====GLIDE - INDICATOR====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    IS_GLIDE));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    IS_GLIDE));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    IS_GLIDE));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    IS_GLIDE));
loc++;

//====SIL - INDICATOR====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    IS_SIL));
loc++;
```

```
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    IS_SIL));
loc++;

phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    IS_SIL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    IS_SIL));
loc++;

phiFeatures.put(loc, calculateDiff(example, win_size_10, end -
    offset_50, IS_SIL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end -
    offset_50, IS_SIL));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end -
    offset_50, IS_SIL));
loc++;

//====VOWELS - SUM DIVIDE BY SUM ALL====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    SUM_VOWELS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    SUM_VOWELS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    SUM_VOWELS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    SUM_VOWELS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    SUM_VOWELS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    SUM_VOWELS));
loc++;

//====NAZAL - SUM DIVIDE BY SUM ALL====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    SUM_NAZALS));
loc++;
```

```
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    SUM_NAZALS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    SUM_NAZALS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    SUM_NAZALS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    SUM_NAZALS));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    SUM_NAZALS));
loc++;

//====GLIDE - SUM DIVIDE BY SUM ALL====//
phiFeatures.put(loc, calculateDiff(example, win_size_10, start,
    SUM_GLIDES));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    SUM_GLIDES));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    SUM_GLIDES));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_10, end,
    SUM_GLIDES));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    SUM_GLIDES));
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    SUM_GLIDES));
loc++;

//======F1======//
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    F_1));//F1, end location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    F_1));//F1, end location - 20 window
loc++;

phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    F_1));//F1, end location - 20 window
```

```
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    F_1));//F1, end location - 20 window
loc++;

//======F2======//
phiFeatures.put(loc, calculateDiff(example, win_size_20, start,
    F_2));//F2, end location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, start,
    F_2));//F2, end location - 20 window
loc++;

phiFeatures.put(loc, calculateDiff(example, win_size_20, end,
    F_2));//F2, end location - 20 window
loc++;
phiFeatures.put(loc, calculateDiff(example, win_size_30, end,
    F_2));//F2, end location - 20 window
loc++;

//==============Mean value from start to end==============//
//true means prev the start point, false means after the end point
phiFeatures.put(loc, calculateMean(example, start, end,
    SHORT_TERM_ENERGY, win_size_30, true));//Mean of short-term
    energy
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    SHORT_TERM_ENERGY, win_size_30, false));//Mean of short-term
    energy
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    LOW_ENERGY, win_size_30, true));//Mean of low energy
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    LOW_ENERGY, win_size_30, false));//Mean of low energy
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    WIENER_ENTROPY, win_size_30, true));//Mean of wiener entropy
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    WIENER_ENTROPY, win_size_30, false));//Mean of wiener entropy
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, VOICING,
    win_size_30, true));//Mean of voicing
loc++;
```

```
phiFeatures.put(loc, calculateMean(example, start, end, VOICING,
    win_size_30, false));//Mean of voicing
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    ZERO_CROSSING, win_size_30, true));//Mean of zero-crossing
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    ZERO_CROSSING, win_size_30, false));//Mean of zero-crossing
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, IS_VOWEL,
    win_size_50, true));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, IS_VOWEL,
    win_size_30, false));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, IS_NAZAL,
    win_size_20, true));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, IS_NAZAL,
    win_size_20, false));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, IS_GLIDE,
    win_size_20, true));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end, IS_GLIDE,
    win_size_20, false));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    SUM_VOWELS, win_size_50, true));
loc++;
phiFeatures.put(loc, calculateMean(example, start, end,
    SUM_VOWELS, win_size_30, false));
loc++;

//==============MFCC Feature Function==============//
//=====================START=======================//
phiFeatures.put(loc, NORMALIZE
    *example.getFeatures2D().get(start).get(MFCC_1));
loc++;
phiFeatures.put(loc, NORMALIZE
    *example.getFeatures2D().get(start).get(MFCC_2));
loc++;
phiFeatures.put(loc, NORMALIZE
    *example.getFeatures2D().get(start).get(MFCC_3));
loc++;
```

```
        phiFeatures.put(loc, NORMALIZE
            *example.getFeatures2D().get(start).get(MFCC_4));
        loc++;

        //=======================END=======================//
        phiFeatures.put(loc, NORMALIZE
            *example.getFeatures2D().get(end).get(MFCC_1));
        loc++;
        phiFeatures.put(loc, NORMALIZE
            *example.getFeatures2D().get(end).get(MFCC_2));
        loc++;
        phiFeatures.put(loc, NORMALIZE
            *example.getFeatures2D().get(end).get(MFCC_3));
        loc++;
        phiFeatures.put(loc, NORMALIZE
            *example.getFeatures2D().get(end).get(MFCC_4));
        loc++;

        //===============Gamma Distribution Over The Vowel
            Length==============//
        //shape = mean^2/var
        //scale = var/mean
        double variance = Math.pow(Consts.STD_VOWEL_LENGTH,2);
        double shape = Math.pow(Consts.MEAN_VOWEL_LENGTH,2)/variance;
        double scale = variance/Consts.MEAN_VOWEL_LENGTH;
        Gamma gamma = new Gamma(shape, scale);
        double vowelLength = end - start;
        phiFeatures.put(loc,gamma.pdf(vowelLength)/gamma.pdf(Consts.MAX_VOWEL_LENGTH));
        loc++;

        //===============Gaussian Distribution Over The Vowel
            Length==============//
        double numerator = -Math.pow((vowelLength -
            Consts.MEAN_VOWEL_LENGTH),2);
        double denominator = 2*Math.pow(Consts.STD_VOWEL_LENGTH,2);
        phiFeatures.put(loc,Math.exp(numerator/denominator));
        loc++;

        newExample.setFeatures(phiFeatures);
        return newExample;
    }
    return null;

} catch (Exception e) {
    e.printStackTrace();
    return null;
    }
}


//*********************************FEATURE
    FUNCTIONS**************************************//
//*****************************************************************************************//
//calculate the average difference of featureNumber win_size before and after
    location
public double calculateDiff(Example example, int win_size, int location, int
    featureNumber)
{
    try {
        double preVal;
        //get the cumulative values of featureNumber
        double startVal = CacheVowelData.getCumulativeValue(example, location
            - win_size, featureNumber);
        double endVal = CacheVowelData.getCumulativeValue(example, location,
            featureNumber);

        preVal = endVal - startVal;
        preVal /= win_size;


        double afterVal;
        //get the cumulative values of featureNumber
        startVal = endVal;
        endVal = CacheVowelData.getCumulativeValue(example, location +
            win_size, featureNumber);

        afterVal = endVal - startVal;
        afterVal /= win_size;

        Double value = afterVal - preVal;
        if(value.isNaN())
            value = 0.0;

        //return the diff
        return value;

    } catch (Exception e){
        Logger.error("Error in function: calculateDiff, feature number: " +
            featureNumber + ", example: " + example.path);
        return 0;
    }
}
```

```
//calculate the avg value from start till end of featureNumber
//if isPrev equals true then create mean difference from start
//else create mean difference from end
public double calculateMean(Example example, int start, int end, int
    featureNumber, int win_size, boolean isPrev)
{
    try {
        double avg;
        int counter = end - start;

        //get the cumulative values of featureNumber
        double startVal =
            CacheVowelData.getCumulativeValue(example,start,featureNumber);
        double endVal =
            CacheVowelData.getCumulativeValue(example,end,featureNumber);

        //computer the mean
        avg = endVal-startVal;

        avg /= counter;

        double val;
        if(isPrev) {
            //get the cumulative values of featureNumber
            startVal = CacheVowelData.getCumulativeValue(example, start -
                win_size, featureNumber);
            endVal = CacheVowelData.getCumulativeValue(example, start,
                featureNumber);

            val = endVal - startVal;
            val /= win_size;
        } else {
            //get the cumulative values of featureNumber
            startVal = CacheVowelData.getCumulativeValue(example, end,
                featureNumber);
            endVal = CacheVowelData.getCumulativeValue(example, end +
                win_size, featureNumber);

            val = startVal - endVal;
            val /= win_size;
        }

        Double value = MathHelpers.sigmoid(avg - val);
        if(value.isNaN())
            value = 0.0;
            return value;

        } catch (Exception e){
            Logger.error("Error in function: calculateMean, feature number:
                "+featureNumber+", example: "+example.path);
            return 0;
        }
    }
    //*****************************************************************************************//
}
```

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

4.4 Factory Update

The last code segment we need to add is the update of the Factory class. The factory class is responsible on the creation of all the objects in our package. It gives us the flexibility to add new classes to a new problem that we wish to learn. We need to add the option to create those new classes that we just implemented.

We need to update the getClassifier function to support the creation of those classes, which means just add new case to the relevant switch case statement, it should look like this:

```
   public static Classifier getClassifier(int taskLossType, int updateType, int
       predictType, int kernelType, int phi, ArrayList<Double> arguments){

      Classifier classifier = new Classifier();
        classifier.classifierData = new ClassifierData();
        classifier.classifierData.arguments = new ArrayList<Double>();

      switch (taskLossType) {
         case 0:
               classifier.classifierData.taskLoss = new TaskLossVowelDuration();
            break;
         case 1:
               classifier.classifierData.taskLoss = new TaskLossMultiClass();
            break;
            case 2:
               classifier.classifierData.taskLoss = new TaskLossDummyData();
               break;
         default:
```

```
        return null;
    }

      switch (predictType) {
          case 0:
              classifier.classifierData.predict = new
                  PredictionVowelDurationData();
              break;
          case 1:
              classifier.classifierData.predict = new PredictionMultiClass();
              break;
          case 2:
              classifier.classifierData.predict = new PredictionDummyData();
              break;
          default:
              return null;
      }

      switch (phi) {
          case 0:
              classifier.classifierData.phi = new PhiVowelDurationConverter();
              break;
          case 1:
              classifier.classifierData.phi = new PhiSparseConverter();
              break;
          case 2:
              classifier.classifierData.phi = new PhiDummyConverter();
              break;
          default:
              return null;
      }

      switch (kernelType) {
          case 0:
              classifier.classifierData.kernel = new Poly2Kernel();
              break;
          case 1:
              classifier.classifierData.kernel = new RBF2Kernel();
              break;
          case 2:
              classifier.classifierData.kernel = new RBF3Kernel();
              break;
          default:
              classifier.classifierData.kernel = null;
      }
    switch (updateType) {
      case 0:
              classifier.classifierData.algorithmUpdateRule =
                  PassiveAggressive.getInstance(arguments);
          break;
      case 1:
              classifier.classifierData.algorithmUpdateRule =
                  SVM_Pegasos.getInstance(arguments);
          break;
      case 2:
              classifier.classifierData.algorithmUpdateRule =
                  DirectLoss.getInstance(arguments);
          break;
      case 3:
              classifier.classifierData.algorithmUpdateRule =
                  CRF.getInstance(arguments);
          break;
      case 4:
              classifier.classifierData.algorithmUpdateRule =
                  RampLoss.getInstance(arguments);
          break;
      case 5:
              classifier.classifierData.algorithmUpdateRule =
                  ProbitLoss.getInstance(arguments);
          break;
          case 6:
              classifier.classifierData.algorithmUpdateRule =
                  RankSVM.getInstance(arguments);
              break;
      default:
          classifier.classifierData.algorithmUpdateRule = null;
    }

    return classifier;
  }
```

We can see here that if we wish to add new Kernel function or even new algorithm, it can be done in the same way.

5 The Config File

The last thing we have left to do is update the config file to indicate that we wish to create this classes. Very detailed information about the Config file can be found at: http://adiyoss.github.io/StructED/ or at the config_details.txt file which is placed inside

the docs folder as well.
The config file for the train should look like this(Here we use the Passive-Aggressive algorithm):

```
train_path:data/db/train.txt
w_output:data/weights/vowel.weights.PA
type:0
task:0
epoch:1
task_param:1;2
phi:0
prediction:0
reader:2
writer:0
isAvg:1
size_of_vector:116
c:3
```

The config file for the prediction should look like this:

```
test_path:data/db/test.txt
output_file:res/PA.txt
w_path:data/weights/vowel.weights.PA
examples_2_display:-1
task:0
task_param:2;3
phi:0
prediction:0
reader:2
writer:0
size_of_vector:116
```

6 Running The Code

To run the code after adding this new classes we need to compile the code for the train and also for the prediction. We can do this either with the IDE that we are working with or from the command line using javac. Notice, both the train and predict executables requirers the path to the config file as parameter.

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.