ST-RU-CT--ED

java structured learning package

Dummy Data Tutorial - by Adiyoss

June 29, 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 small subset from a dummy data that was generated in our lab in order to debug the vowel duration measurement problem, where the input is a speech segment and the output is the onset and offset of the vowel, meaning the start time and end time of the vowel.

2 Dummy Data

The dummy data we use in this tutorial was generated in order to debug the vowel duration measurement problem. Hence, the label for each example is composed from two numbers, the start time and end time ($Y \in \mathbb{R}^2$), and the input data is an arbitrary length vector of numbers for 0,1 domain, ($X \in \{0,1\}^d$ and $d \in \mathbb{N}$).
For example:

8-17 0:0 1:0 2:1 3:0 4:0 5:0 6:0 7:0 8:1 9:1 10:1 11:0 12:1 13:1 14:1 15:1 16:1 17:0 18:0 19:0

Here, the first two numbers are the label which indicates that the signal turns on at the eighth element of the vector and turns off at the seventeenth element of the vector. Each vector data is composed from the index of the feature, we did this to support space features, and the feature data separated by a colon(:). Our goal is to find a function f that gets as input the vector data and outputs the start time where the signal turns on and end time where it turns off. Notice, we add a little bit off noise to the vector data for example at the second index or at the eleventh index, we want to fine f that isn't sensitive to small amount of noise in the data. We know that this is a toy example, but it demonstrate really good the use of the package and the integration with it. We assume that the signal turns on only once.

The db can be found under the db/ folder in this tutorial zip file.

3 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.

3.1 Task Loss

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

$$\ell((t_s, t_e), (\hat{t_s}, \hat{t_e})) = max\{|(\hat{t_s} - \hat{t_e}) - (t_s - t_e)| - \epsilon, 0\} \tag{1}$$

In words, the loss will be the max between zero to the difference between the predicted signals length to its actual length and we minus epsilon. This means that we allow the classifier to be mistaken by at most epsilon.

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.

3.2 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. To make the code go faster we assume that at the beginning and end there's a gap of three frames, which means the beginning of the of the signal can't be in the first 3 frames and the end of the signal can't be at the last three frames, this can be defined other wise if needed. We go over all the possible time sequences for the start time start from minimum gap to maximum gap. Inside an inner loop we go from the start location to minimum gap at the end. The inner loop can also be optimize by start from the minimum length possible of the signal to the maximum length possible of the signal.

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

$$\hat{\boldsymbol{y}}_{\boldsymbol{w}}(\boldsymbol{x}) = \underset{\boldsymbol{y} \in \mathcal{Y}}{\operatorname{argmax}} \ \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \tag{2}$$

and,

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

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.

3.3 Feature Functions

We now add the feature functions. Every feature function class should implement the PhiConvertor interface. Since we want to recognize the pick of the start of the signal and the decrease of the end signal we implemented the following feature functions:

1. Difference between the element at the start index to the element at index start - 1
2. Difference between the element at the start index to the element at index start - 2
3. Difference between the element at the end index to the element at index end + 1
4. Difference between the element at the end index to the element at index end + 2
5. Difference between the mean of the signal from start to end to the mean of the signal from start to start - 3
6. Difference between the mean of the signal from start to end to the mean of the signal from end to end + 3

We expect that the value of the above feature functions will be high when we reach the true start time and end time, and low otherwise.

For this we 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.

4 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 dummy_model = new StructEDModel(W, new PassiveAggressive(), new
    TaskLossDummyData(), new InferenceDummyData(), null, new
    FeatureFunctionsDummy(), arguments);
 // train
dummy_model.train(dummyTrainInstances, task_loss_params, null, epochNum, isAvg);
// predict
ArrayList<PredictedLabels> labels = dummy_model.predict(dummyTestInstances,
    task_loss_params, numExamples2Display);
```

The full code can be found at the tutorial folder or attached to this tutorial. The output should look like this:

```
Dummy data example.
Start Training: 2015/06/29 19:48:34
2015/06/29 19:48:34


==================================
Epoch: 1, 2015/06/29 19:48:34
==================================

Start Training... 2015/06/29 19:48:34
Processing example number: 14
==================================
Epoch: 2, 2015/06/29 19:48:34
==================================

Start Training... 2015/06/29 19:48:34
Processing example number: 14
==================================
Epoch: 3, 2015/06/29 19:48:35
==================================

Start Training... 2015/06/29 19:48:35
Processing example number: 14  2015/06/29 19:48:35
Running time in seconds: 0
Cumulative task loss: 0.0
Y = 6-11, Y_HAT = 6-11
Y = 4-6, Y_HAT = 4-6
Y = 19-22, Y_HAT = 19-22
Y = 3-15, Y_HAT = 3-15
Y = 5-10, Y_HAT = 5-10
Y = 8-16, Y_HAT = 8-16


Process finished with exit code 0
```

GOOD LUCK!