ST-RU-CT--ED

java structured learning package

Tutorial - by Adiyoss

November 24, 2014

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.

1

The db can be found under the db/dummy folder in the project repository.

3 The Code

Here, we present lasses 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, we put the following code in it:

```java
package BL.TaskLoss;

import Constants.Consts;
import java.util.List;

public class TaskLossDummyData implements TaskLoss {

    @Override
    //max{0,|ys-ye - y's-y'e|-epsilon}
    public double computeTaskLoss(String predictClass, String actualClass,
        List<Double> params) {

        double epsilon = params.get(0);
      String predictValues[] = predictClass.split(Consts.CLASSIFICATION_SPLITTER);
      String actualClassValues[] =
          actualClass.split(Consts.CLASSIFICATION_SPLITTER);

      //calculate difference of each classification
      double predictRes = Double.parseDouble(predictValues[0]) -
          Double.parseDouble(predictValues[1]);
      double actualRes = Double.parseDouble(actualClassValues[0]) -
          Double.parseDouble(actualClassValues[1]);

      //subtract the epsilon
      double absRes = Math.abs(predictRes-actualRes) - epsilon;
```

```
        //get the max from the absolute result minus epsilon and 0
        if(absRes > 0)
            return absRes;
        return 0;
    }
}
```

3.2   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. 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:

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

and,

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

For this we create new java class inside the Prediction package, this class should implement the Prediction interface, 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 Helpers.Comperators.MapValueComparatorDescending;
import Helpers.MathHelpers;

public class PredictionDummyData implements Prediction {
```

```java
//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{
        PredictedLabels tree = new PredictedLabels();

        //validation
        if(vector.sizeOfVector<=0)
        {
            System.err.println(ErrorConstants.PHI_VECTOR_DATA);
            return null;
        }

        //loop over all the classifications of this specific example
        for(int i=Consts.MIN_GAP_START_DUMMY-1 ;
            i<vector.sizeOfVector-Consts.MIN_GAP_END_DUMMY ; i++)
        {
            for(int j=i+1 ; j<vector.sizeOfVector- Consts.MIN_GAP_END_DUMMY ;
                j++)
            {
                Example phiData = classifierData.phi.convert(vector,(i+1)+
                    Consts.CLASSIFICATION_SPLITTER+(j+1),classifierData.kernel);
                //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.CLASSIFICATION_SPLITTER+(j+1), realClass,
                        classifierData.arguments);
                }

                //get the max value for the max classification
                tree.put((i+1)+ Consts.CLASSIFICATION_SPLITTER+(j+1),tmp);
            }
        }

        MapValueComparatorDescending vc = new
            MapValueComparatorDescending(tree);
```

```java
            PredictedLabels result = new PredictedLabels(vc);
            result.putAll(tree);

            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);
    }
}
```

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, we put the following code in it:

```java
package Data.FeatureFunctions;
```

```java
import BL.Kernels.Kernel;
import Constants.Consts;
import Constants.ErrorConstants;
import Data.Entities.Example;
import Data.Entities.Vector;
import Data.Factory;

public class PhiDummyConverter implements PhiConverter {

    int sizeOfVector = 6;

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

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

            String labelValues[] = label.split(Consts.CLASSIFICATION_SPLITTER);
            int i = Integer.parseInt(labelValues[0]);
            int j = Integer.parseInt(labelValues[1]);

            //compute the difference/gradients near the start and end points
            double diff_1_Start =
                Math.abs(vector.getFeatures().get(i)-vector.getFeatures().get(i-1));
            double diff_1_End =
                Math.abs(vector.getFeatures().get(j)-vector.getFeatures().get(j+1));

            //compute the difference/gradients near the start and end points
            double diff_2_Start =
                Math.abs(vector.getFeatures().get(i)-vector.getFeatures().get(i-2));
            double diff_2_End =
                Math.abs(vector.getFeatures().get(j)-vector.getFeatures().get(j+2));

            //compute the avg of the signal from start to end
            double avg = 0;
            for(int k=i ; k<j ; k++)
                avg+=vector.getFeatures().get(k);

            if(j-i <= 0){
                System.err.println("Convert single vector: "+
                    ErrorConstants.GENERAL_ERROR+ ErrorConstants.ZERO_DIVIDING);
                return null;
            }
            avg /= (double)(j-i);
```

```java
        //compute the avg MIN_GAP_END_DUMMY after end
        double gapEnd = 0;
        for(int k=j ; k<j+Consts.MIN_GAP_END_DUMMY ; k++)
            gapEnd+=vector.getFeatures().get(k);

        gapEnd/=(double)Consts.MIN_GAP_END_DUMMY;
        //compute the avg MIN_GAP_START_DUMMY before start
        double gapStart = 0;
        for(int k=i-Consts.MIN_GAP_START_DUMMY+1 ; k<=i ; k++)
            gapStart+=vector.getFeatures().get(k);

        gapStart/=(double)Consts.MIN_GAP_START_DUMMY;

        //adding the feature functions
        Vector tmpVector = new Vector();
        tmpVector.put(0, diff_1_Start);
        tmpVector.put(1, diff_1_End);
        tmpVector.put(2, diff_2_Start);
        tmpVector.put(3, diff_2_End);
        tmpVector.put(4, avg-gapStart);
        tmpVector.put(5,avg-gapEnd);

        if(kernel!=null)
            tmpVector = kernel.convertVector(tmpVector, sizeOfVector);

        newVector.setFeatures(tmpVector);

        return newVector;

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

3.4 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:

```java
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 TaskLossRank();
            break;
        case 3:
            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 PredictionRanking();
            break;
        case 3:
            classifier.classifierData.predict = new PredictionDummyData();
            break;
        default:
            return null;
    }
```

```java
        switch (phi) {
            case 0:
                classifier.classifierData.phi = new PhiVowelDurationConverter();
                break;
            case 1:
                classifier.classifierData.phi = new PhiSparseConverter();
                break;
            case 2:
                classifier.classifierData.phi = new PhiRankConverter();
                break;
            case 3:
                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.

4   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 Direct-Loss minimization algorithm):

```
train_path:data/db/dummy/train.txt
w_output:data/weights/dummy.weights
train_path:data/db/dummy/train.txt
w_output:data/weights/dummy.weights
type:2
task:3
epoch:5
task_param:1
```

```
phi:3
prediction:3
reader:0
writer:0
size_of_vector:4
eta:0.5
epsilon:-1.6
```

The config file for the prediction should look like this:

```
test_path:data/db/dummy/test.txt
output_file:res/dummy_scores.txt
w_path:data/weights/dummy.weights
examples_2_display:3
task:3
task_param:1
phi:3
prediction:3
reader:0
writer:0
size_of_vector:4
```

5   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.

  The output after the train should look like this:

```
================================
Processing example number: 7, 2014/11/18 14:28:35
********************************
================================
Processing example number: 8, 2014/11/18 14:28:35
********************************
================================
Processing example number: 9, 2014/11/18 14:28:35
********************************
================================
Processing example number: 10, 2014/11/18 14:28:35
********************************
================================
Processing example number: 11, 2014/11/18 14:28:35
********************************
================================
Processing example number: 12, 2014/11/18 14:28:35
********************************
================================
Processing example number: 13, 2014/11/18 14:28:35
********************************
================================
Processing example number: 14, 2014/11/18 14:28:35
********************************
================================
Processing example number: 1, 2014/11/18 14:28:35
********************************

================================
Epoch: 4, 2014/11/18 14:28:35
================================

2014/11/18 14:28:35
Running time in seconds: 0
```

The output after the prediction should look like this:

```
Loading test data... 2014/11/18 14:29:01
Test file: data/db/dummy/test.txt.  2014/11/18 14:29:01


================================================
Processing example number: 1, Real Label = 6-11, Prediction: 6-11
Processing example number: 2, Real Label = 4-6, Prediction: 4-6
Processing example number: 3, Real Label = 19-22, Prediction: 19-22
Processing example number: 4, Real Label = 3-15, Prediction: 3-15
Processing example number: 5, Real Label = 5-10, Prediction: 5-10
Processing example number: 6, Real Label = 8-16, Prediction: 8-16
================================================
================================================
2014/11/18 14:29:01
Running time in seconds: 0
================================================


================================================
Total files predicted: 6
Cumulative task loss: 0.0
================================================
```

Hope you had fun!
GOOD LUCK!