ST-RU-CT--ED

java structured learning package

Multi-Class 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 multi-class problem, the standard benchmark MNIST, you can download the data set from mnist.

This tutorial is also suitable for any other multi-class problem.

2 MNIST

MNIST is a dataset of handwritten digits labeled from 0 to 9, and contains 60,000 training examples and 10,000 test examples. Each example has been size-normalized and centered in a fixed-size image of 28×28.

Notice that when downloading MNIST db from the web we get a compressed files, so we need to make a little bit of preparations before using StructED . StructED should get as input the db from the following format,

Each example should be in a different line (meaning each example should end with \n). Each feature/value pair should be separated by a space character and a : between the feature number and its value. Features with value zero can be skipped. The label(target value) should be the first value in each example.

For example, the line: 3 1:0.55 8:0.07 2293:0.11

specifies an example of class 3 for which feature number 1 has the value 0.55, feature number 8 has the value 0.07, feature number 2293 has the value 0.11, all the other features have value 0.

We provide a sample from the MNIST data set with this tutorial, it can be found under db/ directory.

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 0-1 loss:

$$\ell(y, \hat{y}) = \mathbb{1}[y \neq \hat{y}] \tag{1}$$

In other words, the loss will be one if y not equal to $\hat{y}$, otherwise it will be zero.

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 java.util.List;

//Task loss multi class
public class TaskLossMultiClass implements TaskLoss {

   @Override
   public double computeTaskLoss(String predictClass, String actualClass,
       List<Double> params) {
      if(predictClass.equalsIgnoreCase(actualClass))
        return 0;
      return 1;
   }
}
```

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. We will just run over all the possible classes and predict the one that maximizes $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$.

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, 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;
import Helpers.MathHelpers;

public class PredictionMultiClass implements Prediction{

    int numOfClass = 10;
    //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) {
                Logger.error(ErrorConstants.PHI_VECTOR_DATA);
                return null;
            }

            for(int i=0 ; i<numOfClass ; i++){

                Example phiData =
                    classifierData.phi.convert(vector,String.valueOf(i),classifierData.kernel);
                //multiple the vectors
                double tmp = MathHelpers.multipleVectors(W, phiData.getFeatures());
```

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

            //get the max value for the max classification
            tree.put(String.valueOf(i),tmp);
        }

        MapValueComparatorDescending vc = new
            MapValueComparatorDescending(tree);
        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){

    if(returnAll != Consts.ERROR_NUMBER) {
        try {
            PredictedLabels tree = new PredictedLabels();

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

            for (int i = 0; i < numOfClass; i++) {
                Example phiData = classifierData.phi.convert(vector,
                    String.valueOf(i), classifierData.kernel);
                //multiple the vectors
                double tmp = MathHelpers.multipleVectors(W,
                    phiData.getFeatures());

                //get the max value for the max classification
                tree.put(String.valueOf(i), tmp);
```

```
                }

                MapValueComparatorDescending vc = new
                    MapValueComparatorDescending(tree);
                PredictedLabels result = new PredictedLabels(vc);
                result.putAll(tree);

                return result;

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

3.3 Feature Functions

We now add the feature functions. In multi-class problems there is no real need for feature functions, but we need to store as many weight vectors as the number of classes as defined in the task settings. To solve that we just concatenate all the vectors one after the other into a single weight vector.

Thus, the feature functions is just putting the right vector in his place according to its class number.

To do this we create new java class inside the FeatureFunction package, this class should implement the PhiConverter interface, we put the following code in it:

```
package Data.FeatureFunctions;

import BL.Kernels.Kernel;
import Constants.ConfigParameters;
import Data.Entities.Example;
import Data.Factory;
import Data.Entities.Vector;

public class PhiSparseConverter implements PhiConverter {

    int maxFeatures = 784;

   @Override
```

```
    public Example convert(Example vector, String label, Kernel kernel) {
        try{
            //parse the label
            int intLabel = Integer.parseInt(label);
            Example newVector = Factory.getExample(0);
            Vector tmpVector = new Vector();

            //run the phi function
            for(Integer feature : vector.getFeatures().keySet())
                tmpVector.put(feature+intLabel*maxFeatures,vector.getFeatures().get(feature));

            if(kernel!=null)
                tmpVector = kernel.convertVector(tmpVector,
                    ConfigParameters.getInstance().VECTOR_SIZE);

            newVector.setFeatures(tmpVector);

            return newVector;

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

     //setter for the max feature parameter
     public void setMaxFeatures(int maxFeatures){
         this.maxFeatures = maxFeatures;
     }
}
```

If you wish to use it from a different multi-class problem all you need to do is just change the maxFeatures parameter at the top from 784 ($28\times$ 28) to the right vector length of each example in the new task.

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:

```
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.

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 Passive-Aggressive algorithm):

```
train_path:data/db/train.txt
w_output:data/weights/mnist.weights.PA
type:0
task:1
epoch:4
phi:1
prediction:1
reader:0
writer:0
isAvg:1
size_of_vector:7840
c:0.01
```

The config file for the prediction should look like this:

```
test_path:data/db/test.txt
output_file:res/mnist_scores_PA.txt
w_path:data/weights/mnist.weights.PA
```

```
examples_2_display:3
task:1
phi:1
prediction:1
reader:0
writer:0
size_of_vector:7840
```

---

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.

Hope you had fun!

GOOD LUCK!