

ST-RU-CT--ED

java structured learning package

Multi-Class 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 a multi-class problem, the standard benchmark MNIST, you can download the data set from [mnist](http://mnist.cs.toronto.edu/).

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:

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

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. 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/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 $\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y})$.

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

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

and,

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

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. 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. 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

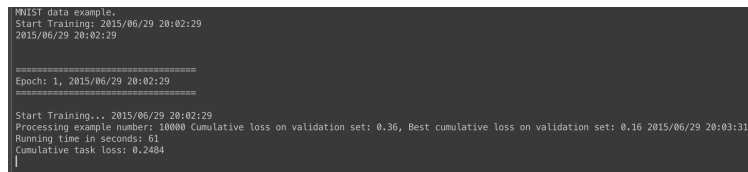
We now can run the StructEDModel with the interfaces we have just implemented, all we have left to do is to choose the model. The code should look like this:

```
// create the model
StructEDModel mnist_model = new StructEDModel(W, new SVM(), new
    TaskLossMultiClass(),
        new InferenceMultiClass(), null, new FeatureFunctionsSparse(),
            arguments);

// train
mnist_model.train(mnistTrainInstances, task_loss_params, mnistDevelopInstances,
    epochNum, isAvg);

// predict
mnist_model.predict(mnistTestInstances, null, numExamples2Display);
```

The output should look like this:



```
MNIST data example.
Start Training: 2015/06/29 20:02:29
2015/06/29 20:02:29

=====
Epoch: 1, 2015/06/29 20:02:29
=====

Start Training... 2015/06/29 20:02:29
Processing example numbers: 10000 cumulative loss on validation set: 0.36, Best cumulative loss on validation set: 0.16 2015/06/29 20:03:31
Running time in seconds: 61
Cumulative task loss: 0.2484
|
```

Hope you had fun!

GOOD LUCK!