The Jensen-Shannon Divergence and Machine Learning Toolbox; Quick Start Guide

Nicholas Carrara, Jesse Ernst

August 11, 2017

1 Introduction

For those who do not want to wade through the full documentation, this quick start guide will get the user up and running with some simple examples for the JSDML Toolbox which are included with the main code. The file 'quick_start_examples.py' contains the examples for this quick start guide. Each example runs in several chunks;

- Import the signal and background distributions from a file and specify a training and testing set for the neural network
- Specify the network topology and compile the network
- Run the training data on the network to train it and then run the testing data and evaluate the output of the network
- Evaluate the CDF's of the network outputs and generate accept/reject curves and compute the area under curve
- Finally, compute the mutual information before and after and print the results to the screen
- We may also generate various plots for the different examples

2 Example One - 5D Gaussians

The first example imports a set of signal and background events from two pre-made files that contain five-dimensional Gaussians with means $\mu_s = 1.0$ and $\mu_b = -1.0$ for signal and background respectively. The variances are all equal to $\sigma = 1$ for both signal and background. The signal and background files are .csv files where each column is one of the five variables $(x_1, x_2, x_3, x_4, x_5)$. The first line in the first example imports these variables into a testing and training set;

```
11 train_x,train_y,test_x,test_y=pn.create_feature_sets_and_labels("QuickStartGaussSignal.csv",
12                                                                "QuickStartGaussBackground.csv",
13                                                                num_of_vars=5,
14                                                                test_size=0.3,
15                                                                file_size=0.999,
16                                                                var_set=[0,1,2,3,4] )
```

Now we have our training data and training answer in both **train_x** and **train_y** respectively. We also have a testing set made from 30% of the data (specified by **test_size=0.3**) which is contained in **test_x** and **test_y** respectively. Next we specify the network topology and the network object;

```
17 layer_vector = [5, 11, 4, 1]
18 network = pn.nnet( layer_vector, learning_rate=0.05, decay_value=1e-5 )
```

The network is a fully connected network with topology $5 \rightarrow 11 \rightarrow 4 \rightarrow 1$; i.e. five input nodes, two hidden layers of 11 and 4 nodes each and one output node. To compile the network object we run line 18, and the learning rate and decay rate are set to **learning_rate=0.05** and **decay_value=1e-5** respectively. The next step is to train and evaluate the network which is done with the next couple of lines;

```
20 network.train_network( train_x, train_y, num_epochs=25, batch=100 )
21 results = network.evaluate_network( test_x, test_y )
```

Here we've sent the training data to the network to train it for 25 epochs (**num_epochs=100**) and with a batch size of 100 (**batch_size=100**). Then once the network is trained we send the testing data through and record the outputs for each event which are saved in the **results** array. The next block of code evaluates the mutual information (or JSD) for both the input and output of the neural network;

```
23 train_y = [[train_y[i]] for i in range(len(train_y))]
24 mutual = pn.mi(network.normalize_data(train_x),train_y,k=1)
25 signal, background = network.split_binary_results( results )
26 new_mutual = network.network_output_jsd(signal, background, neighbors=1)
27 print "MI before: ", mutual
28 print "MI after:  ", new_mutual
```

The first line prepares the class labels as a list of lists; i.e. $[1.0, -1.0, -1.0, 1.0, \ldots] \rightarrow [[1.0], [-1.0], [-1.0], [1.0], \ldots]$. Line 24 calculates the mutual information on the input data. The training data is normalized however before passing it to the mutual information calculation (**network.normalize_data(train_x)**). The reason for this is explained in the corresponding paper[1] and the full documentation[2]. Line 25 separates the results array into individual signal and background arrays to be passed to the mutual information calculation again. This calculates the mutual information on the output of the network. The next block of code generates CDF's for the accept/reject curves and plots them along with a histogram of the network output;

```
30 network.plot_network_output(results,symmetric=False)
```

The **symmetric=False** argument tells the plotting software to not make the cuts in equal percentages of signal. More information on this is in the documentation. Running this part of the code we get the following output after epoch 25;

```
1 Epoch 25/25
2 139860/139860 [==============================] - 1s - loss: 0.0407 - acc: 0.9761
3 59800/59940 [==============================>.] - ETA: 0sScore: [0.040476502045382647, 0.97620955204(
4 MI before:  0.949721835535
5 MI after:  0.947794012923
6 AUC: Background Rej. vs. Signal Acc.;
7 Keras/Tensorflow: 0.999363913884
```

With the corresponding plots;

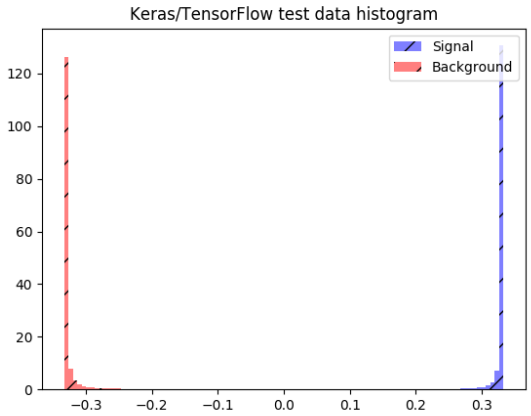

Figure 1: Histogram of the neural network output for example one.

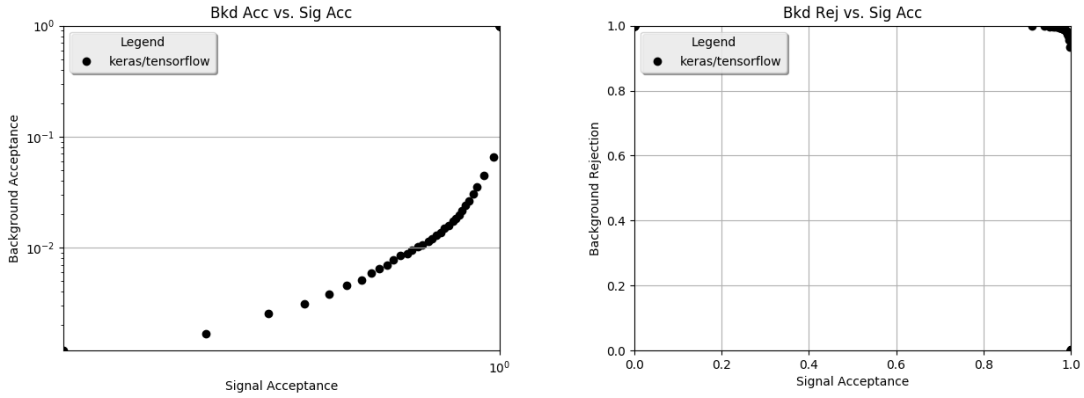

Figure 2: ROC curves for example one. The left plot is a background acceptance vs. signal acceptance curve, and the right plot is a background rejection vs. signal acceptance curve.

This particular example has signal and background distributions which are very distinguishable, hence the area under curve almost equal to one.

3 Example Two - MAGIC Gamma Telescope

The MAGIC Gamma Telescope[4] is one of the examples from UC Irvine's repository of machine learning data sets[3]. The code to run this example is exactly the same as before, the only difference being the file names specified and the number of variables. The topology of the network and the number of epochs are changed accordingly;

```
train_x,train_y,test_x,test_y=pn.create_feature_sets_and_labels("QuickStartMagicSignal.csv",
                                                               "QuickStartMagicBackground.csv",
                                                               num_of_vars=5,
                                                               test_size=0.3,
                                                               file_size=0.999,
                                                               var_set=[0,1,2,3,4,5,6,7,8,9] )
layer_vector = [10, 16, 9, 1]
network = pn.nnet( layer_vector, learning_rate=0.05, decay_value=1e-5 )
#   Now we train the network on the training data and evaluate the testing data
network.train_network( train_x, train_y, num_epochs=100, batch=100 )
results = network.evaluate_network( test_x, test_y )
#   Now we evaluate the mutual information on the input and output
train_y = [[train_y[i]] for i in range(len(train_y))]
mutual = pn.mi(network.normalize_data(train_x),train_y,k=1)
signal, background = network.split_binary_results( results )
new_mutual = network.network_output_jsd(signal, background, neighbors=3)
print "MI before: ", mutual
print "MI after:  ", new_mutual
#   This section generates the ROC curves and calculates the area under curve
network.plot_network_output(results, symmetric=False)
```

After running this example we get the following output after 100 epochs;

```
Epoch 100/100
9354/9354 [==============================] - 0s - loss: 0.4297 - acc: 0.7025
3200/4008 [=======================>.....] - ETA: 0sScore: [0.44687112401464502, 0.65219560789015
MI before:   0.508839446834
MI after:    0.485335164462
AUC: Background Rej. vs. Signal Acc.;
Keras/Tensorflow: 0.924574048694
```

with the corresponding plots;

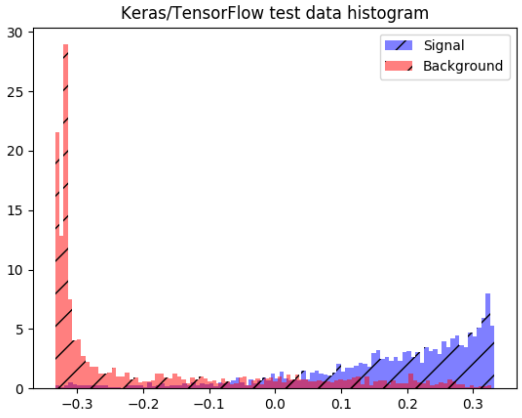

Figure 3: Histogram of the neural network output for example two.

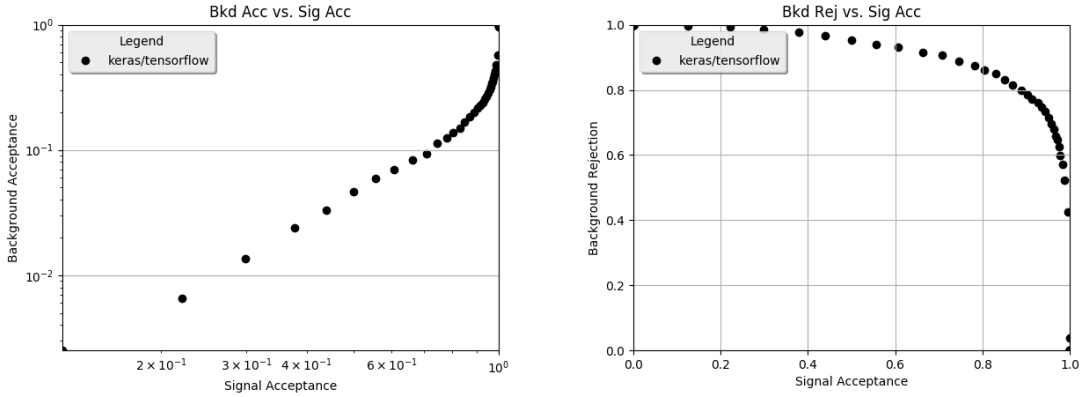

Figure 4: ROC curves for example two. The left plot is a background acceptance vs. signal acceptance curve, and the right plot is a background rejection vs. signal acceptance curve.

As you can see, this data is not as distinguishable as the Gaussian example, as given by the JSD (mutual information) of 0.5088

4 Example Three - Breast Cancer Wisconsin (Diagnostic)

This example looks at part of an older study[?] that only has a few hundred examples. The number of features is high and so this is a good example of how a sparse data set can still have the mutual information be calculated quite accurately. The code only needs to be changed slightly like we did for the previous example;

```
60 #   Example 3 - Breast Cancer Data
61 train_x,train_y,test_x,test_y=pn.create_feature_sets_and_labels("QuickStartBCSignal.csv",
62                                                                "QuickStartBCBackground.csv",
63                                                                num_of_vars=30,
64                                                                test_size=0.3,
65                                                                file_size=0.999 )
66 layer_vector = [30, 50, 15, 1]
```

```
network = pn.nnet( layer_vector, learning_rate=0.05, decay_value=1e-5 )
#   Now we train the network on the training data and evaluate the testing data
network.train_network( train_x, train_y, num_epochs=1000, batch=100 )
results = network.evaluate_network( test_x, test_y )
#   Now we evaluate the mutual information on the input and output
train_y = [[train_y[i]] for i in range(len(train_y))]
mutual = pn.mi(network.normalize_data(train_x),train_y,k=1)
signal, background = network.split_binary_results( results )
new_mutual = network.network_output_jsd(signal, background, neighbors=3)
print "MI before: ", mutual
print "MI after:  ", new_mutual
#   This section generates the ROC curves and calculates the area under curve
network.plot_network_output(results, symmetric=False)
```

The number of discriminating variables is 30 and the output from running after 100 epochs is the following;

```
Epoch 100/100
381/381 [==============================] - 0s - loss: 0.0177 - acc: 0.9869
42/42 [==============================] - 0s
Score: [0.10017212480306625, 0.92857140302658081]
MI before:  0.888210349168
MI after:  0.882675769278
AUC: Background Rej. vs. Signal Acc.;
Keras/Tensorflow: 0.995454545455
```

with the corresponding histogram of the output;

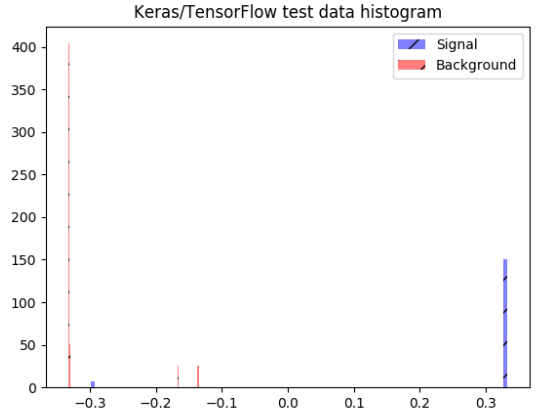

Figure 5: Histogram of the neural network output for example two.

The amount of testing samples for this example was reduced to 10% since the number of events were so few. To get a handle on the value of the MI estimation, we've included an additional block of code that will randomly sample a group of points from the Breast Cancer data set and calculate an average mutual information from them. The code is here;

```
#   Example 3 - Breast Cancer Data Average MI
train_x, train_y, test_x, test_y = pn.create_feature_sets_and_labels( "QuickStartBCSignal.csv",
"QuickStartBCBackground.csv",
num_of_vars=30,
test_size=0.01,
file_size=0.999 )
JSD_list = []
layer_vector = [30, 50, 15, 1]
num_samples = len(train_x)-1
```

```
num_random_samples = int(len(train_x)/2.0)
network = pn.nnet( layer_vector, learning_rate=0.05, decay_value=1e-5 )
network.find_normalization_parameters(train_x)
train_x = network.normalize_data(train_x)
for i in range(100):
#   Now randomly sample the train_x data set and calculate MI
samples = np.random.randint(num_samples, size=num_random_samples)
temp_train_x = [train_x[j] for j in samples]
temp_train_y = [[train_y[j]] for j in samples]
JSD_list.append(pn.mi(temp_train_x,temp_train_y,k=1))
print "MI for sample set ",i,": ",JSD_list[i]
avg_JSD = sum(JSD_list)/100
print "MI average: ", avg_JSD
```

and the output of running this test gives an average MI of

```
MI average:  0.90423707641
```

5 User Data

The end of the quick start file contains a block section of code similar to the other examples except with the specific details removed. To use it, one simply has to enter the signal and background file names, specify the numbers of variables (or the variable set) and the network topology. Once this is completed the user can test his example again the code block here.

References

[1] N. Carrara and J. Ernst, *On the Upper Limit of Separability*

[2] N. Carrara and J. Ernst, *The Jensen-Shannon Divergence and Machine Learning Toolbox*

[3] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[4] Bock, R.K., Chilingarian, A., Gaug, M., Hakl, F., Hengstebeck, T., Jirina, M., Klaschka, J., Kotrc, E., Savicky, P., Towers, S., Vaicilius, A., Wittek W. (2004). Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope. Nucl.Instr.Meth. A, 516, pp. 511-528.

[5] W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.