



**Data Mining Final Term Project**

**On**

**Supervised Data Mining (Classification)**

**Category 5 (Naïve Bayes)**

**Category 8 (Neural Network)**

**By:**

**Sahithi Boinpalli**

**Student ID: 31443821**

**Email: sb2247@njit.edu**

## Table of Contents

<b>1. Introduction</b>	3
a. Final Term Project Option 1	3
b. Tensorflow	3
<b>2. Installing Anaconda</b>	4
<b>3. Installing Tensorflow in Anaconda</b>	5
<b>4. Classification using Naïve Bayes</b>	6
a. Classify using Naïve Bayes	6
b. Analysis of Naïve Bayes	10
c. Area under ROC	11
<b>5. Classification using Neural Network</b>	11
a. Classify using Neural Network	11
b. Analysis of Neural Network	15
c. Area under ROC	16
<b>6. Comparison of results</b>	17
<b>7. Input Data</b>	18
<b>8. References</b>	19
<b>9. Source Code</b>	20
a. Naïve Bayes	20
b. Neural Network	23

## 1. Introduction

### a. Final Term Project Option 1

Supervised Data Mining – Classification

- Category 5: Naïve Bayes
- Category 8: Neural Networks

Dataset: <https://archive.ics.uci.edu/ml/datasets/Iris>

This dataset consists of 4 attributes and 150 instances.

Operating System: Windows 10

System Type: 64 bit operating system

Programming Language: Python

Tool: Tensorflow

Hardware: Intel® Core™ i5 -3210M CPU @ 2.50GHz

Installed Ram: 8GB

**Naive Bayes** is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

**Neural Networks** represent a brain metaphor for information processing. These models are biologically inspired rather than an exact replica of how the brain actually functions. Neural networks have been shown to be very promising systems in many forecasting applications and business classification applications due to their ability to “learn” from the data, their nonparametric nature (i.e., no rigid assumptions), and their ability to generalize. Neural computing refers to a pattern recognition methodology for machine learning. The resulting model from neural computing is often called an artificial neural network (ANN) or a neural network. Neural networks have been used in many business applications for pattern recognition, forecasting, prediction, and classification. Neural network computing is a key component of any data mining tool kit.

### b. Tensorflow

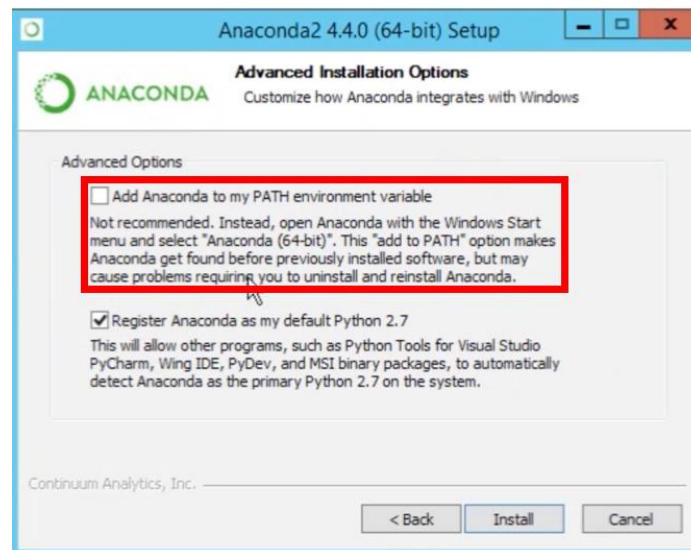
**TensorFlow** is an open source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture enables you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes [TensorBoard](#), a data visualization toolkit.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization for the purposes of conducting machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

## 2. Installing Anaconda

This is split into three sections. The first part is installing Anaconda. The second part is testing your installation (making sure conda works, dealing with path issues etc). Finally, the last part of the tutorial goes over installing packages, and environment management.

- i. Download and install Anaconda (windows version) from:  
<https://www.anaconda.com/download/> and choose python 3.6 to download.



- ii. Select the default options when prompted during the installation of Anaconda.
- iii. After you finished installing, open **Anaconda Prompt**. Type the command below to see that you can use a Jupyter (IPython) Notebook.
- iv. If you didn't check the add Anaconda to path argument during the installation process, you will have to add python and conda to your environment variables. You know you need to do so if you open a **command prompt** (not anaconda prompt) and get the following messages.
- v. This step gives two options for adding python and conda to your path (only choose 1 option).
- vi. 6. Close the current command prompt and open a new one. Try typing python and conda in your **command prompt** to see if the paths are saved. Done!

```
C:\Users\mgalarnyk>python
'python' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\mgalarnyk>jupyter notebook
'jupyter' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\mgalarnyk>conda list
'conda' is not recognized as an internal or external command,
operable program or batch file.
```

### 3. Installing Tensorflow in Anaconda

This guide explains how to install TensorFlow on Windows.

**The Anaconda installation is community supported, not officially supported.**

Take the following steps to install TensorFlow in an Anaconda environment:

1. Follow the instructions on the [Anaconda download site](#) to download and install Anaconda.
2. Create a conda environment named tensorflow by invoking the following command:
  - C:> **conda create -n tensorflow**

Activate the conda environment by issuing the following command:

C:> **activate tensorflow** (tensorflow)C:> # Your prompt should change

Issue the appropriate command to install TensorFlow inside your conda environment. To install the CPU-only version of TensorFlow, enter the following command:

(tensorflow)C:> **pip install—ignore-installed—  
upgrade[https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win\\_amd64.whl](https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl)**

1. To install the GPU version of TensorFlow, enter the following command (on a single line):

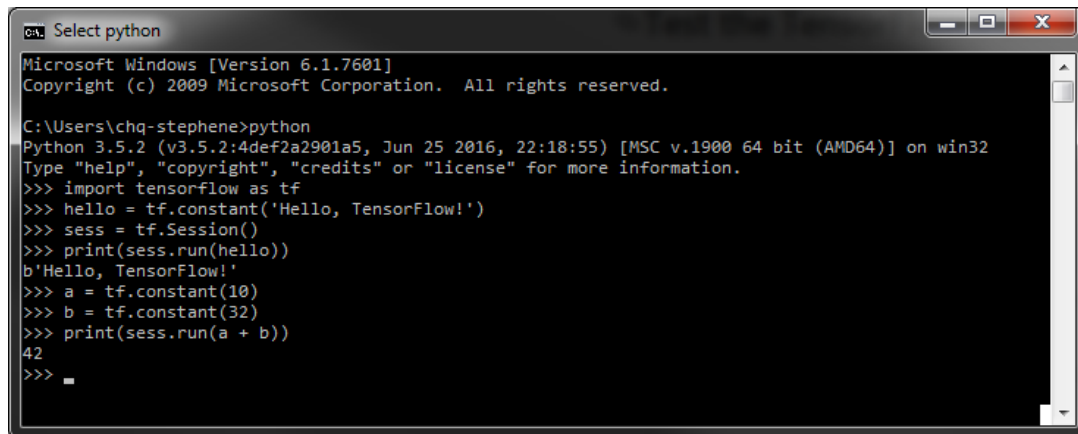
(tensorflow)C:> **pip install—ignore-installed—  
upgrade[https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow\\_gpu-1.0.1-cp35-cp35m-win\\_amd64.whl](https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow_gpu-1.0.1-cp35-cp35m-win_amd64.whl)**

tensorflow\_gpu-1.0.0-cp35-cp35m-win\_x86\_64.whl is not a supported wheel on this platform.

If you are using anaconda distribution and getting the above error, you can do the following to use python 3.5 on the new environment “tensorflow”:

```
conda create --name tensorflow python=3.5
activate tensorflow
conda install jupyter
conda install scipy
pip install tensorflow
```

```
# or  
# pip install tensorflow-gpu
```



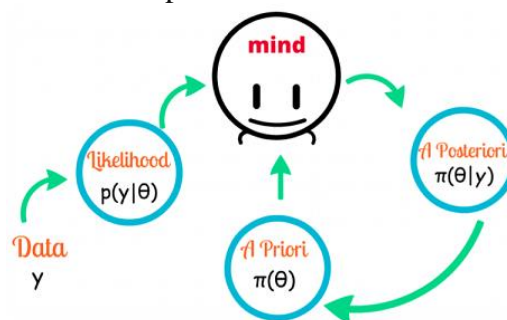
```
Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
C:\Users\chq-stephene>python  
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import tensorflow as tf  
>>> hello = tf.constant('Hello, TensorFlow!')  
>>> sess = tf.Session()  
>>> print(sess.run(hello))  
b'Hello, TensorFlow!'  
>>> a = tf.constant(10)  
>>> b = tf.constant(32)  
>>> print(sess.run(a + b))  
42  
>>> _
```

## 4. Classification using Naïve Bayes

### a. Classify using Naïve Bayes

The Naive Bayes classifier is a frequently encountered term in the blog posts here; it has been used in the previous articles for building an email spam filter and for performing sentiment analysis on movie reviews. Thus a post explaining its working has been long overdue. Despite being a fairly simple classifier with oversimplified assumptions, it works quite well in numerous applications. Let us now try to unravel its working and understand what makes this family of classifiers so popular.

We begin by refreshing our understanding about the fundamental concepts behind the classifier – conditional probability and the Bayes' theorem. This is followed by an elementary example to show the various calculations which are made to arrive at the classification output. Finally, we implement the classifier's algorithm in Python and then validate the code's output with results obtained for the demonstrated example.



### Conditional Probability and Bayes' Theorem

In the simplest terms, conditional probability, denoted as  $P(O|E)$  and read as *probability of outcome O given event E*, is the probability of the occurrence of an outcome O given that some other event E has already occurred. It is calculated as

$$P(O|E) = \frac{P(O \cap E)}{P(E)}$$

$P(O \cap E)$  is the probability that both O and E occur; this is the same as calculating  $P(O)$ , the probability of O occurring multiplied by  $P(E|O)$ , the probability of E occurring given that O has already occurred, or conversely  $P(E)$ , the probability of E occurring multiplied by  $P(O|E)$ , the probability of O occurring given that E has occurred.

$$P(O \cap E) = P(O) * P(E|O) = P(E) * P(O|E)$$

The equality of the two terms on the right leads to the Bayes' Theorem.

$$P(O|E) = \frac{P(E|O) * P(O)}{P(E)}$$

Thus, the Bayes' Theorem is a way to go from  $P(O|E)$  to  $P(E|O)$ . The above equation is often written as

$$\text{posterior probability} = \frac{\text{likelihood} * \text{prior probability}}{\text{evidence}}$$

$P(O)$  is also known as **a priori** probability because it is probability of the class (or outcome or model) which is always known a priori from the training examples, while  $P(O|E)$  is called the **a posteriori** probability because it is the probability of the class (or model) which we calculate after seeing the data (or evidence or events).

### Why is it called “Naive”?

So far, we have only talked about the effect of a single event E on the occurrence of outcome O. In reality however, the probability of outcome O's occurrence is governed by more than one already occurred event. The mathematics behind such a scenario gets very complicated and is made simple by assuming that the various governing events are independent of each other. Mathematically expressing, if an outcome O depends on events  $E_1, E_2, \dots, E_n$ , then assuming  $E_1, E_2, \dots, E_n$  do not depend on one other,

$$P(O|E_1, E_2, \dots, E_n) = P(O)$$

Thus, the model is very naively believing that the effect of the occurrence of any of the events is completely independent of the occurrence of other events. This is obviously not the case with most of the real world problems and the predictor features are known to influence and be influenced by other features. Still, naive Bayes is known to provide results at par and sometimes even better than highly complex and computationally expensive classification models.

### An Illustrative Example

Lets clear our understanding by taking a simple example. Suppose we have the following training data (taken from [this](#) YouTube video) regarding whether a patient had the flu or not, depending on if he had chills, a runny nose, headache and fever.

chills	runny nose	headache	fever	flu?
Y	N	Mild	Y	N
Y	Y	No	N	Y
Y	N	Strong	Y	Y
N	Y	Mild	Y	Y
N	N	No	N	N
N	Y	Strong	Y	Y
N	Y	Strong	N	N
Y	Y	Mild	Y	Y

You might have guessed that chills, runny nose, headache and fever are **features**(analogous to the already occurred events  $E_1, E_2, E_3$  and  $E_4$  explained above), while flu and no flu are the two **classes**  $C_1$  and  $C_2$  in which the above patients were known to be falling (analogous to the two outcomes  $O_1$  and  $O_2$ ). We get a new patient with the following symptoms which are previously unseen by the classifier:

chills	runny nose	headache	fever	flu?
Y	N	Mild	N	?

### How is the classification done?

For the task of classification, we use the training data to determine the prior probabilities of the 2 classes (flu or no flu) and the likelihoods of the 4 events that govern these outcomes. We begin by calculating the likelihoods of each of the event given the class, like  $P(chills = N|flu = Y)$ ,  $P(fever = N|flu = N)$  and so on, just like illustrated in equation 1 above. These values are to be used to determine the posterior probability of the test data belonging to each of the 2 classes, i.e.,  $P(C_1|E_1, E_2, E_3, E_4)$  and  $P(C_2|E_1, E_2, E_3, E_4)$ . The patient is predicted to be belonging to the class for which the value of the calculated posterior probability is **higher**.

The various probabilities that we are going to require as per equation 1 for the case when the patient has flu, ie,  $flu = Y$  are calculated as below..

$$P(flu = Y) = \frac{5}{8} = 0.625$$

$$P(chills = Y|flu = Y) = \frac{3}{5} = 0.6$$



---


$$P(\text{runnynose} = N | \text{flu} = Y) \qquad \frac{1}{5} = 0.2$$


---

$$P(\text{headache} = \text{mild} | \text{flu} = Y) \qquad \frac{2}{5} = 0.4$$


---

$$P(\text{fever} = N | \text{flu} = Y) \qquad \frac{1}{5} = 0.2$$


---

$$P(\text{chills} = Y) \qquad \frac{4}{8} = 0.5$$


---

$$P(\text{runnynose} = N) \qquad \frac{3}{8} = 0.375$$


---

$$P(\text{headache} = \text{mild}) \qquad \frac{3}{8} = 0.375$$


---

$$P(\text{fever} = N) \qquad \frac{3}{8} = 0.375$$


---

Now calculating  $P(\text{flu} = Y | E_1, E_2, E_3, E_4)$  by plugging these probabilities in (1)

$$3. \quad 0.625 = 0.227$$

Similarly, the various probabilities are calculated for the case of  $\text{flu} = N$ . The readers are encouraged to calculate these values and check them with the ones used below. Hence  $P(\text{flu} = N | E_1, E_2, E_3, E_4)$

$$0.375 = 0.674$$

Thus the naive Bayes' classifier will predict the class  $\text{flu} = N$  for the above patient because the value of posterior probability was higher for  $\text{flu} = N$  than for the class  $\text{flu} = Y$ .

If you notice, you can see what the task of prediction eventually reduces to – **just calculating some probabilities and a few multiplications**. All fairly easy and quick. If you pay even more attention, you see that since the denominator is the same while determining the probabilities of the outcome as either flu or no flu, you can even do away without calculating its value.

Thus  $P(\text{flu} = Y | E_1, E_2, E_3, E_4) = 0.6 * 0.4 * 0.2 * 0.2 * 0.625 = \mathbf{0.006}$

$P(\text{flu} = N | E_1, E_2, E_3, E_4) = 0.33 * 0.66 * 0.33 * 0.66 * 0.375 = \mathbf{0.0185}$ .

Again, the calculated probabilities give the same prediction as before, ie, no flu. We will validate these 2 values in the next section via a Python code.

This is what makes naive Bayes' so popular as a classifier, combined with the fact that it has been seen to perform exceptionally well in many applications.

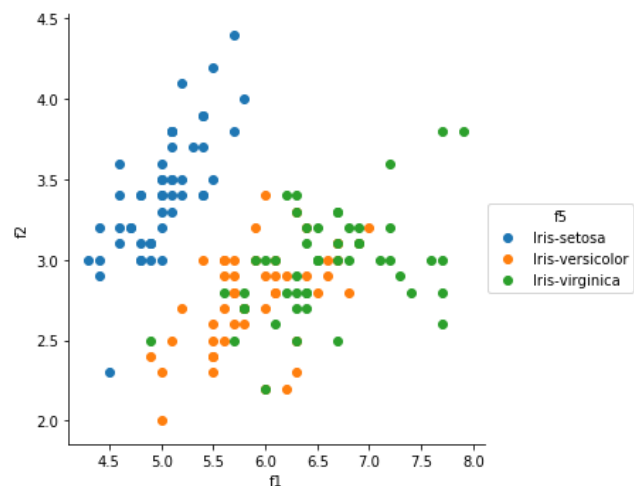
## b. Analysis of Naïve Bayes

```
In [4]: data["f5"].value_counts()
```

```
Out[4]: Iris-virginica    50
       Iris-versicolor    50
       Iris-setosa        50
       Name: f5, dtype: int64
```

```
In [5]: sns.FacetGrid(data, hue="f5", size=5) \
       .map(plt.scatter, "f1", "f2") \
       .add_legend()
```

```
Out[5]: <seaborn.axisgrid.FacetGrid at 0x1bdf90c3ef0>
```



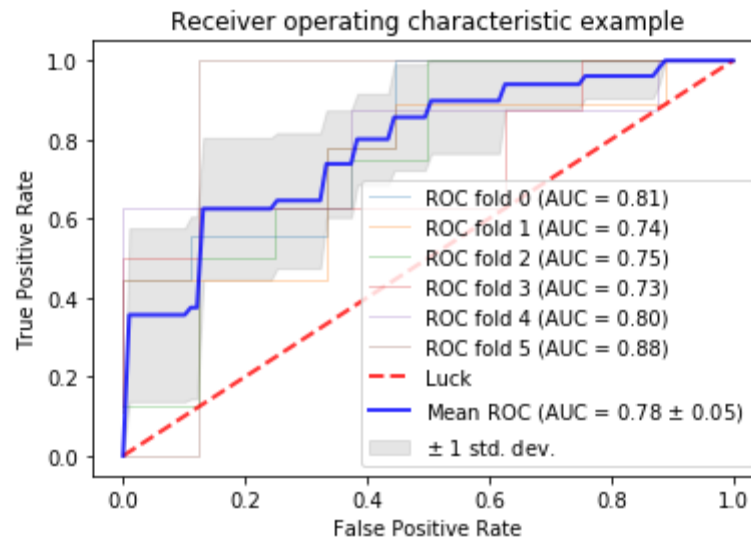
```
In [18]: for step in range(epoch):
         _, c=sess.run([train_step,cross_entropy], feed_dict={x: x_input, y_: [t for t in y_input.as_matrix()]})
         if step%500==0 :
             print(c)
```

```
1.0986122
0.1623094
0.10485032
0.085064806
```

```
In [20]: print(sess.run(accuracy,feed_dict={x: x_test, y_: [t for t in y_test.as_matrix()]})
```

```
1.0
```

### c. Area Under ROC



## 5. Classification Using Neural Network

### a. Classify data into neural networks

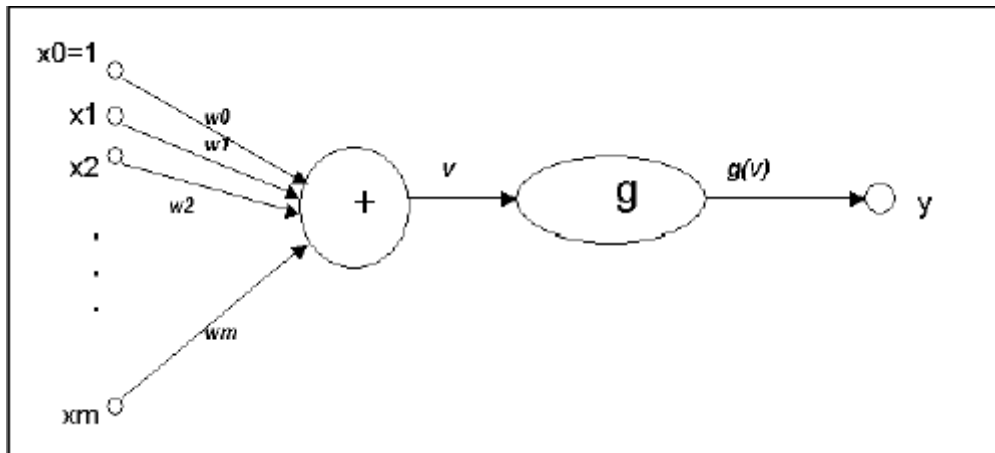
#### Introduction

Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record. The errors from the initial classification of the first record is fed back into the network, and used to modify the networks algorithm for further iterations.

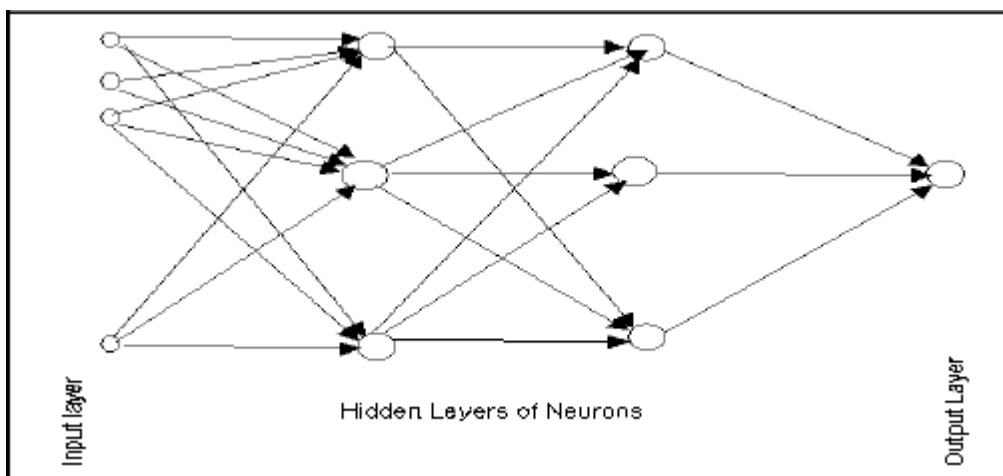
A neuron in an artificial neural network is

A set of input values ( $x_i$ ) and associated weights ( $w_i$ ).

A function ( $g$ ) that sums the weights and maps the results to an output ( $y$ ).



Neurons are organized into layers: input, hidden and output. The input layer is composed not of full neurons, but rather consists simply of the record's values that are inputs to the next layer of neurons. The next layer is the hidden layer. Several hidden layers can exist in one neural network. The final layer is the output layer, where there is one node for each class. A single sweep forward through the network results in the assignment of a value to each output node, and the record is assigned to the class node with the highest value.



### Training an Artificial Neural Network

In the training phase, the correct class for each record is known (termed supervised training), and the output nodes can be assigned correct values -- 1 for the node corresponding to the correct class, and 0 for the others. (In practice, better results have been found using values of 0.9 and 0.1, respectively.) It is thus possible to compare the network's calculated values for the output nodes to these correct values, and calculate an error term for each node (the Delta rule). These error terms are then used to adjust the weights in the hidden layers so that, hopefully, during the next iteration the output values will be closer to the correct values.

## The Iterative Learning Process

A key feature of neural networks is an iterative learning process in which records (rows) are presented to the network one at a time, and the weights associated with the input values are adjusted each time. After all cases are presented, the process is often repeated. During this learning phase, the network trains by adjusting the weights to predict the correct class label of input samples. Advantages of neural networks include their high tolerance to noisy data, as well as their ability to classify patterns on which they have not been trained. The most popular neural network algorithm is the back-propagation algorithm proposed in the 1980s.

Once a network has been structured for a particular application, that network is ready to be trained. To start this process, the initial weights (described in the next section) are chosen randomly. Then the training (learning) begins.

The network processes the records in the Training Set one at a time, using the weights and functions in the hidden layers, then compares the resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record. This process occurs repeatedly as the weights are tweaked. During the training of a network, the same set of data is processed many times as the connection weights are continually refined.

Note that some networks never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also will not converge if there is not enough data to enable complete learning. Ideally, there should be enough data available to create a Validation Set.

### Feedforward, Back-Propagation

The feedforward, back-propagation architecture was developed in the early 1970s by several independent sources (Werbos; Parker; Rumelhart, Hinton, and Williams). This independent co-development was the result of a proliferation of articles and talks at various conferences that stimulated the entire industry. Currently, this synergistically developed back-propagation architecture is the most popular model for complex, multi-layered networks. Its greatest strength is in non-linear solutions to ill-defined problems.

The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there are just one or two. Some studies have shown that the total number of layers needed to solve problems of any complexity is five (one input layer, three hidden layers and an output layer). Each layer is fully connected to the succeeding layer.

The training process normally uses some variant of the Delta Rule, which starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times, which are a scaling factor for global accuracy. This means that the inputs, the output, and the desired output all must be present at the same processing element. The most complex part of this algorithm is determining which input contributed the most to an incorrect output and how must the input be modified to correct the error. (An inactive node would not contribute to the error and would have no need to change its weights.) To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During

the learning process, a forward sweep is made through the network, and the output of each element is computed by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function. The connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached.

### Structuring the Network

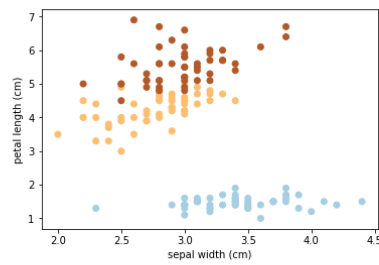
The number of layers and the number of processing elements per layer are important decisions. To a feedforward, back-propagation topology, these parameters are also the most ethereal -- they are the art of the network designer. There is no quantifiable answer to the layout of the network for any particular application. There are only general rules picked up over time and followed by most researchers and engineers applying while this architecture to their problems.

Rule One: As the complexity in the relationship between the input data and the desired output increases, the number of the processing elements in the hidden layer should also increase.

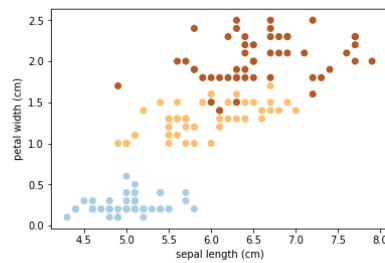
Rule Two: If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization of the training set, and not a true general solution.

Rule Three: The amount of Training Set available sets an upper bound for the number of processing elements in the hidden layer(s). To calculate this upper bound, use the number of cases in the Training Set and divide that number by the sum of the number of nodes in the input and output layers in the network. Then divide that result again by a scaling factor between five and ten. Larger scaling factors are used for relatively less noisy data. If too many artificial neurons are used the Training Set will be memorized, not generalized, and the network will be useless on new data sets.

## b. Analysis of Neural Network



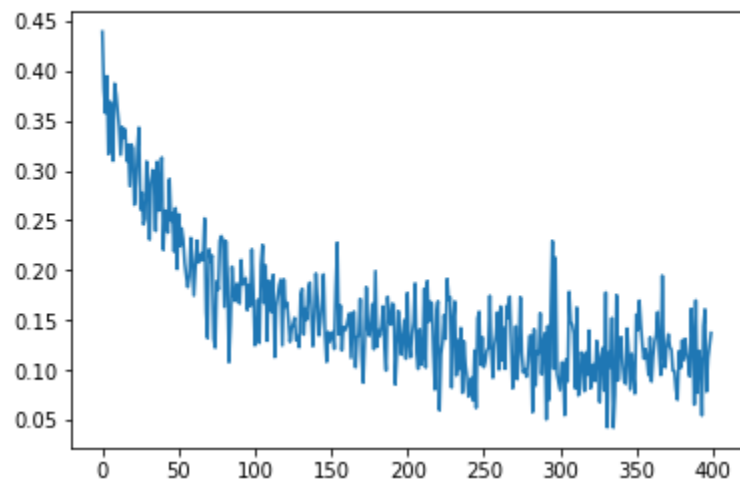
Out[3]: Text(0,0.5,'petal width (cm)')



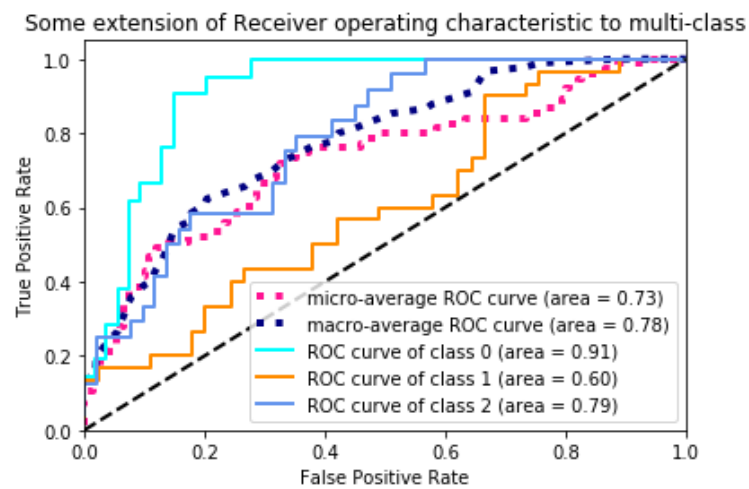
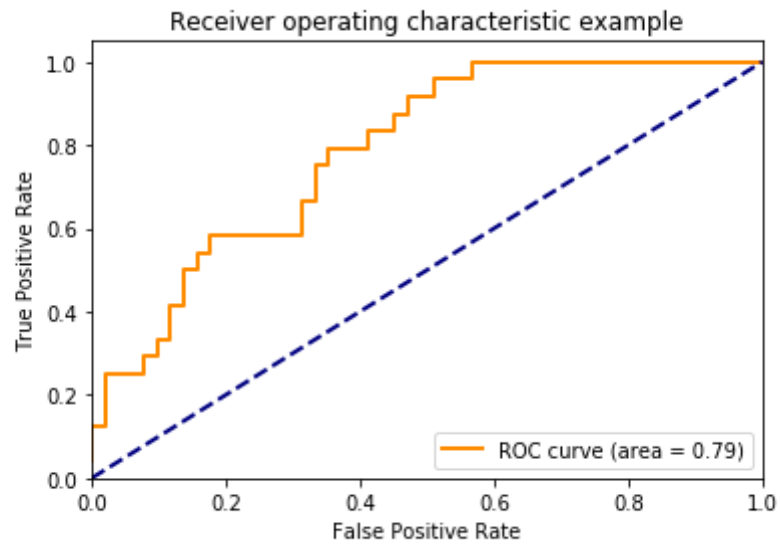
Targets: ['setosa' 'versicolor' 'virginica']

Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Out[12]: [<matplotlib.lines.Line2D at 0x13c3a23a5f8>]

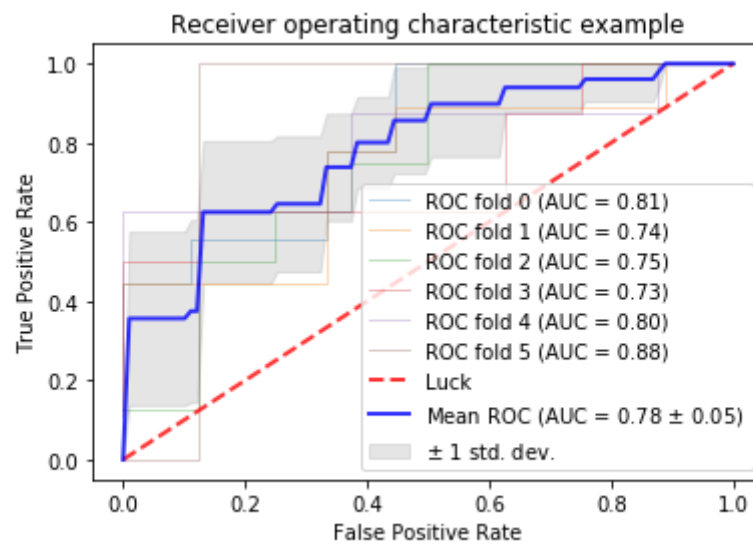
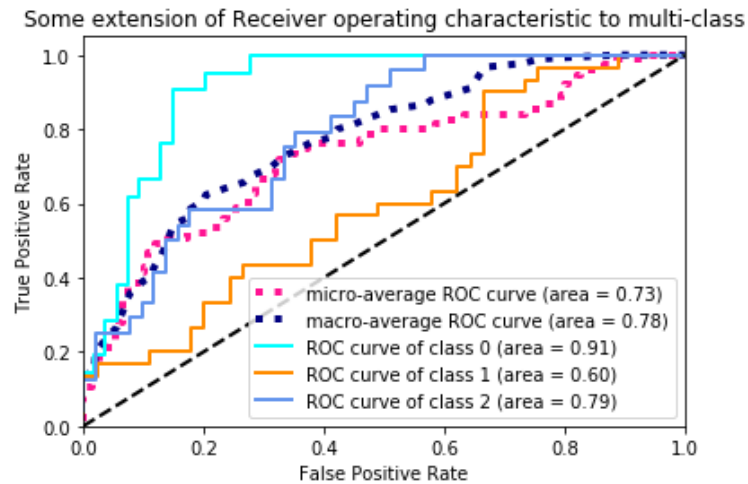


### c. Area Under ROC





## 6. Comparison of Results



## 7. Input Data

In this paper for comparison of all classification we used “iris.arff” dataset the basic information of iris is given below. Relevant Information: This is perhaps the best known database to be found in the pattern recognition literature --- Predicted attribute: class of iris plant. --- This is an exceedingly simple domain. Number of Instances: 150 (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the class Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

Missing Attribute Values: None

## 8. REFERENCES

- <https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/>
- [https://www.tensorflow.org/install/install\\_windows](https://www.tensorflow.org/install/install_windows)
- <https://conda.io/docs/user-guide/install/windows.html>
- <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6942087FD80A3FA308CA0980C115C189?doi=10.1.1.640.136&rep=rep1&type=pdf>
- <http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab4-NaiveBayes.pdf>
- <http://guidetodatamining.com/assets/guideChapters/DataMining-ch6.pdf>
- <https://www.geeksforgeeks.org/naive-bayes-classifiers/>
- <https://www.geeksforgeeks.org/introduction-to-artificial-neural-networks/>
- <https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/>

## 9. SOURCE CODE

### 9.1.Naïve Bayes:

```
# In[1]:
import tensorflow as tf
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from scipy import interp
from itertools import cycle

from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold

# In[2]:
data=pd.read_csv('/Users/my/bezdekIris.csv', names=['f1','f2','f3','f4','f5'])

# In[3]:
data

# In[4]:
data["f5"].value_counts()

# In[5]:
sns.FacetGrid(data, hue="f5", size=5) .map(plt.scatter, "f1", "f2") .add_legend()

# In[6]:
#map data into arrays
s=np.asarray([1,0,0])
ve=np.asarray([0,1,0])
vi=np.asarray([0,0,1])
data['f5'] = data['f5'].map({'Iris-setosa': s, 'Iris-versicolor': ve,'Iris-virginica':vi})

# In[7]:
data

# In[8]:
#shuffle the data
data=data.iloc[np.random.permutation(len(data))]
```

```
# In[9]:
data

# In[10]:
data=data.reset_index(drop=True)

# In[11]:
data

# In[12]:
#training data
x_input=data.ix[0:105,['f1','f2','f3','f4']]
temp=data['f5']
y_input=temp[0:106]
#test data
x_test=data.ix[106:149,['f1','f2','f3','f4']]
y_test=temp[106:150]

# In[13]:
#placeholders and variables. input has 4 features and output has 3 classes
x=tf.placeholder(tf.float32,shape=[None,4])
y_=tf.placeholder(tf.float32,shape=[None, 3])
#weight and bias
W=tf.Variable(tf.zeros([4,3]))
b=tf.Variable(tf.zeros([3]))

# In[14]:
# model
#softmax function for multiclass classification
y = tf.nn.softmax(tf.matmul(x, W) + b)

# In[15]:
#loss function
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))

# In[16]:
#optimiser -
train_step = tf.train.AdamOptimizer(0.01).minimize(cross_entropy)
#calculating accuracy of our model
```

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
# In[17]:
#session parameters
sess = tf.InteractiveSession()
#initialising variables
init = tf.initialize_all_variables()
sess.run(init)
#number of iterations
epoch=2000
```

```
# In[18]:
for step in range(epoch):
    _, c=sess.run([train_step,cross_entropy], feed_dict={x: x_input, y_: [t for t in
y_input.as_matrix()]})
    if step%500==0 :
        print(c)
```

```
# In[19]:
#random testing at Sn.130
a=data.ix[130,['f1','f2','f3','f4']]
b=a.reshape(1,4)
largest = sess.run(tf.argmax(y,1), feed_dict={x: b})[0]
if largest==0:
    print("flower is :Iris-setosa")
elif largest==1:
    print("flower is :Iris-versicolor")
else :
    print("flower is :Iris-virginica")
```

```
# In[20]:
print(sess.run(accuracy,feed_dict={x: x_test, y_: [t for t in y_test.as_matrix()])))
```

```
# In[22]:
# Import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target
X, y = X[y != 2], y[y != 2]
n_samples, n_features = X.shape
```

```

# In[23]:
# Add noisy features
random_state = np.random.RandomState(0)
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# In[24]:
# Run classifier with cross-validation and plot ROC curves
cv = StratifiedKFold(n_splits=6)
classifier = svm.SVC(kernel='linear', probability=True,
                    random_state=random_state)

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

i = 0
for train, test in cv.split(X, y):
    probas_ = classifier.fit(X[train], y[train]).predict_proba(X[test])
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y[test], probas_[ :, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))

    i += 1
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Luck', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,

```

```
label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

## **b. Neural Network:**

```
# In[1]:
get_ipython().run_line_magic('matplotlib', 'inline')
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
from itertools import cycle
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp
iris = datasets.load_iris()
print(iris)

# In[2]:
print(iris.data[0:10,:])

# In[3]:
plt.scatter(iris.data[:,1], iris.data[:,2], c=iris.target, cmap=plt.cm.Paired)
plt.xlabel(iris.feature_names[1])
plt.ylabel(iris.feature_names[2])
plt.show()

plt.scatter(iris.data[:,0], iris.data[:,3], c=iris.target, cmap=plt.cm.Paired)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[3])

# In[4]:
print("Targets: " + str(iris.target_names))
print("Features: " + str(iris.feature_names))
```



```
# In[5]:
import numpy as np
#np.array([[w_1_1, w_1_2], [w_2_1, w_2_2])
mult_matrix = np.array([[-0.2, 0.2], [-1.0, 1.0]])
bias = np.array([0.1, -0.1])
for features in iris.data:
    our_features = [features[2]-3, features[3]-2]
    print(our_features)
    a = np.matmul(our_features, mult_matrix)
    a = a + bias
    print("activations: " + str(a))

# In[6]:
import numpy as np
def sigmoid(activations):
    return 1 / (1 + np.exp(-activations))

#np.array([[w_1_1, w_1_2], [w_2_1, w_2_2])
mult_matrix = np.array([[-0.2, 0.2], [-1.0, 1.0]])
bias = np.array([0.1, -0.1])

for features in iris.data:
    our_features = [features[2]-3, features[3]-2]
    print(our_features)
    a = np.matmul(our_features, mult_matrix)
    a = a + bias
    a = sigmoid(a)
    print("activations: " + str(a))

# In[7]:
import tensorflow as tf
tf.reset_default_graph()

n_input = len(iris.data[0])
n_output = 3 # [0,1,2]... set(iris.target)

input_shape = [None,n_input]
inputplaceholder = tf.placeholder(dtype=tf.float32, shape=input_shape, name="input_placeholder") #
https://www.tensorflow.org/api\_docs/python/tf/placeholder

weights = tf.Variable(tf.random_normal([n_input, n_output]), name="weights")
biases = tf.Variable(tf.zeros([n_output]), name="biases")

print(weights)
print(biases)

layer_1 = tf.matmul(inputplaceholder, weights)
layer_2 = tf.add(layer_1, biases)
outputlayer = tf.nn.sigmoid(layer_2)
```

```
# In[8]:
learning_rate = 0.1
labelsplaceholder = tf.placeholder(dtype=tf.float32, shape=[None,n_output], name="labels_placeholder")
cost = tf.losses.mean_squared_error(labelsplaceholder, outputlayer) #
https://www.tensorflow.org/api\_docs/python/tf/losses/mean\_squared\_error

print(cost)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)
https://www.tensorflow.org/api\_docs/python/tf/train/GradientDescentOptimizer

# In[9]:
init = tf.global_variables_initializer() #
https://www.tensorflow.org/api\_docs/python/tf/global\_variables\_initializer
sess = tf.Session() # https://www.tensorflow.org/api\_docs/python/tf/Session
sess.run(init)

# In[10]:
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(iris.data)
scaled_data = scaler.transform(iris.data)

# In[11]:
import random
mydata = list(zip(scaled_data, iris.target))
# for x in mydata:
#     print(x)
batch_size = 10
iterations = 400
history_loss = list()
for _ in range(iterations):
    inputdata = list()
    output_data = list()
    for _ in range(batch_size):
        input_output_pairs = random.choice(mydata)
        inputdata.append(input_output_pairs[0])
        output_one_hot = [0.0,0.0,0.0]
        output_one_hot[input_output_pairs[1]] = 1.0
        output_data.append(output_one_hot)

    res_optimizer, res_cost = sess.run([optimizer, cost], feed_dict={inputplaceholder: inputdata,
labelsplaceholder: output_data})
    print(res_cost)
    history_loss.append(res_cost)
```

```
# In[12]:
plt.plot(history_loss)

# In[13]:
correct_predictions = 0
for i in range(len(scaled_data)):
    predicted_by_network = sess.run(outputlayer, feed_dict={inputplaceholder: [scaled_data[i]]})
    print("input: %s, expected: %s, predicted: %s " % (str(scaled_data[i]), str(iris.target[i]),
    str(predicted_by_network)))
    if np.argmax(predicted_by_network) == iris.target[i]:
        correct_predictions += 1

print("Correct_predictions: " + str(correct_predictions) + "/" + str(len(scaled_data)) + " Accuracy: " +
str(correct_predictions/len(scaled_data)))

# In[14]:
from sklearn import model_selection
x_train, x_test, y_train, y_test = model_selection.train_test_split(iris.data, iris.target, test_size=0.2,
random_state=42)

## scaling - see next chapter
scaler = preprocessing.StandardScaler().fit(x_train)
scaled_data_train = scaler.transform(x_train)
scaled_data_test = scaler.transform(x_test)

# In[15]:
init = tf.global_variables_initializer() #
https://www.tensorflow.org/api\_docs/python/tf/global\_variables\_initializer
sess = tf.Session() # https://www.tensorflow.org/api\_docs/python/tf/Session
sess.run(init)

mydata = list(zip(scaled_data_train, y_train))

batch_size = 10

history_loss = list()
for _ in range(400):
    inputdata = list()
    outputlogits = list()
    for _ in range(batch_size):
        input_output_pairs = random.choice(mydata)
        inputdata.append(input_output_pairs[0])
        output_expected = [0.0,0.0,0.0]
        output_expected[input_output_pairs[1]] = 1.0
        outputlogits.append(output_expected)

    res_optimizer, res_cost = sess.run([optimizer, cost], feed_dict={inputplaceholder: inputdata,
labelsplaceholder: outputlogits})
```

```
print(res_cost)
history_loss.append(res_cost)

# In[16]:
plt.plot(history_loss)

# In[17]:
logit_y_test = list()
for label in y_test:
    toadd = [0.0,0.0,0.0]
    toadd[label] = 1.0
    logit_y_test.append(toadd)
res_cost, predicted = sess.run([cost, outputlayer], feed_dict={inputplaceholder: scaled_data_test,
labelsplaceholder: logit_y_test})

# In[18]:
print(res_cost)

# In[19]:
correct_predictions = 0
for index in range(len(y_test)):
    print("Label: %d, predicted: %s" % (y_test[index], predicted[index]))
    if y_test[index] == np.argmax(predicted[index]):
        correct_predictions += 1
print(correct_predictions)
print(len(y_test))

# In[20]:
X = iris.data
y = iris.target
# Binarize the output
y = label_binarize(y, classes=[0, 1, 2])
n_classes = y.shape[1]

# In[21]:
# Add noisy features to make the problem harder
random_state = np.random.RandomState(0)
n_samples, n_features = X.shape
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# In[22]:
# shuffle and split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5,
                                                    random_state=0)
```

```
# In[23]:
# Learn to predict each class against the other
classifier = OneVsRestClassifier(svm.SVC(kernel='linear', probability=True,
                                         random_state=random_state))
y_score = classifier.fit(X_train, y_train).decision_function(X_test)
```

```
# In[24]:
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# In[25]:
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

```
# In[26]:
plt.figure()
lw = 2
plt.plot(fpr[2], tpr[2], color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
# In[27]:
# Compute macro-average ROC curve and ROC area
# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
```

```
# In[28]:
# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])
```

```
# In[29]:
# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# In[30]:
# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
         ".format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
         ".format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ".format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```