**Code for Decision Tree**

import numpy as np

import pandas as pd

import warnings

warnings.simplefilter("ignore")

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

plt.style.use('ggplot')

%matplotlib inline

mushrooms = 'C:/Users/Jonathan/Downloads/mushrooms.csv'

dataset=pd.read\_csv(mushrooms)

X=dataset.drop('class',axis=1) #Predictors

y=dataset['class'] #True values for prediction class

X.head()

| **cap-shape** | **cap-surface** | **cap-color** | **bruises** | **odor** | **gill-attachment** | **gill-spacing** | **gill-size** | **gill-color** | **stalk-shape** | **...** | **stalk-surface-below-ring** | **stalk-color-above-ring** | **stalk-color-below-ring** | **veil-type** | **veil-color** | **ring-number** | **ring-type** | **spore-print-color** | **population** | **habitat** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | x | s | n | t | p | f | c | n | k | e | ... | s | w | w | p | w | o | p | k | s | u |
| **1** | x | s | y | t | a | f | c | b | k | e | ... | s | w | w | p | w | o | p | n | n | g |
| **2** | b | s | w | t | l | f | c | b | n | e | ... | s | w | w | p | w | o | p | n | n | m |

5 rows × 22 columns

from sklearn.preprocessing import LabelEncoder

Encoder\_X = LabelEncoder()

for col in X.columns:

X[col] = Encoder\_X.fit\_transform(X[col])

Encoder\_y=LabelEncoder()

y = Encoder\_y.fit\_transform(y)

#Test data 30% of observations, Training set 70%

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

---------------------------------------------For Visualizing Data-----------------------------------------------------------------

def visualization\_train(model):

sns.set\_context(context='notebook',font\_scale=2)

plt.figure(figsize=(16,9))

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.6, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title("%s Training Set" %(model))

plt.xlabel('PC 1')

plt.ylabel('PC 2')

plt.legend()

def visualization\_test(model):

sns.set\_context(context='notebook',font\_scale=2)

plt.figure(figsize=(16,9))

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.6, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title("%s Test Set" %(model))

plt.xlabel('PC 1')

plt.ylabel('PC 2')

plt.legend()

--------------------------------------------------End definition for Visualization------------------------------------------------

from sklearn.model\_selection import cross\_val\_predict, cross\_val\_score

from sklearn.metrics import confusion\_matrix,classification\_report,accuracy\_score

#def for printing scores in an easier to read manner

def print\_score(classifier,X\_train,y\_train,X\_test,y\_test,train=True):

if train == True:

print("Training results:\n")

print('Accuracy Score: {0:.4f}\n'.format(accuracy\_score(y\_train,classifier.predict(X\_train))))

print('Classification Report:\n{}\n'.format(classification\_report(y\_train,classifier.predict(X\_train))))

print('Confusion Matrix:\n{}\n'.format(confusion\_matrix(y\_train,classifier.predict(X\_train))))

res = cross\_val\_score(classifier, X\_train, y\_train, cv=10, n\_jobs=-1, scoring='accuracy')

print('Average Accuracy:\t{0:.4f}\n'.format(res.mean()))

print('Standard Deviation:\t{0:.4f}'.format(res.std()))

elif train == False:

print("Test results:\n")

print('Accuracy Score: {0:.4f}\n'.format(accuracy\_score(y\_test,classifier.predict(X\_test))))

print('Classification Report:\n{}\n'.format(classification\_report(y\_test,classifier.predict(X\_test))))

print('Confusion Matrix:\n{}\n'.format(confusion\_matrix(y\_test,classifier.predict(X\_test))))

#Creation of Decision Tree and Training

from sklearn.tree import DecisionTreeClassifier as DT

from sklearn.cross\_validation import cross\_val\_score

classifier = DT(criterion='entropy',random\_state=42)

classifier.fit(X\_train,y\_train)

---------------------------------------------------------Output------------------------------------------------------------------------

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=42,

splitter='best')

---------------------------------------------------------Output------------------------------------------------------------------------

print\_score(classifier,X\_train,y\_train,X\_test,y\_test,train=True)

---------------------------------------------------------Output------------------------------------------------------------------------

Training results:

Accuracy Score: 1.0000

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2951

1 1.00 1.00 1.00 2735

avg / total 1.00 1.00 1.00 5686

Confusion Matrix:

[[2951 0]

[ 0 2735]]

---------------------------------------------------------Output------------------------------------------------------------------------

print\_score(classifier,X\_train,y\_train,X\_test,y\_test,train=False)

---------------------------------------------------------Output------------------------------------------------------------------------

Test results:

Accuracy Score: 0.8970

Classification Report:

precision recall f1-score support

0 0.90 0.91 0.90 1257

1 0.90 0.89 0.89 1181

avg / total 0.90 0.90 0.90 2438

Confusion Matrix:

[[1139 118]

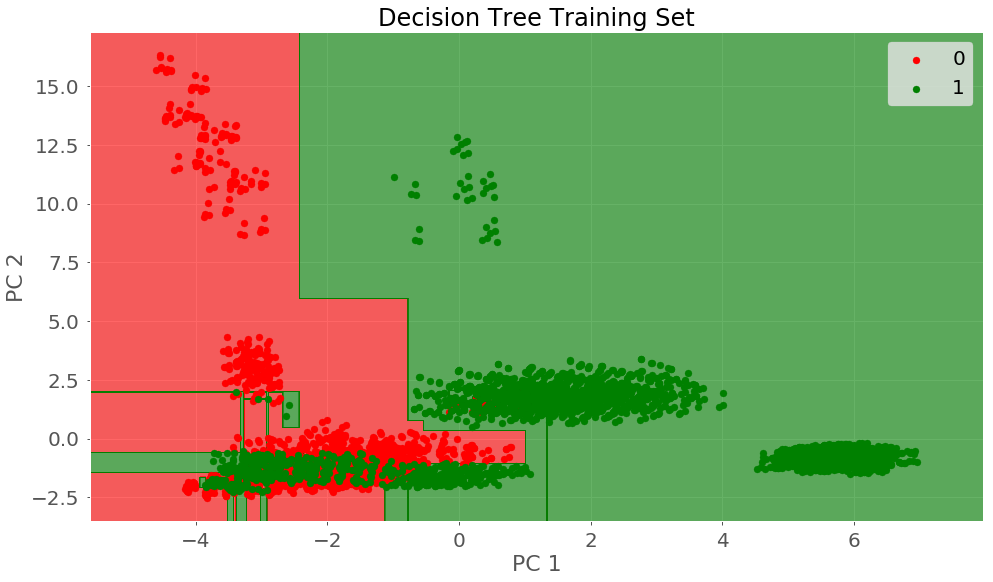
[ 133 1048]]

Average Accuracy: 0.8925

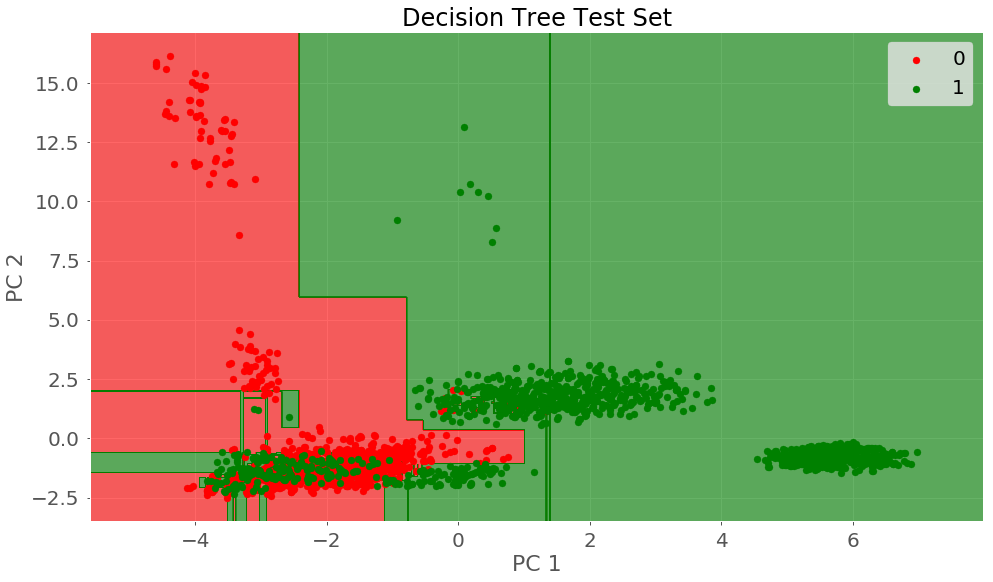
Standard Deviation: 0.0128

---------------------------------------------------------Output------------------------------------------------------------------------

visualization\_train('Decision Tree')



visualization\_test('Decision Tree')



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

classifier2 = DT(criterion='entropy',random\_state=42)

classifier2.fit(X\_train,y\_train)

---------------------------------------------------------Output------------------------------------------------------------------------

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=42,

splitter='best')

---------------------------------------------------------Output------------------------------------------------------------------------

# testing with a 50/50 split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=45)

classifier2 = DT(criterion='entropy',random\_state=42)

classifier2.fit(X\_train,y\_train)

print\_score(classifier2,X\_train,y\_train,X\_test,y\_test,train=True)

---------------------------------------------------------Output------------------------------------------------------------------------

Training results:

Accuracy Score: 1.0000

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2097

1 1.00 1.00 1.00 1965

avg / total 1.00 1.00 1.00 4062

Confusion Matrix:

[[2097 0]

[ 0 1965]]

Average Accuracy: 1.0000

Standard Deviation: 0.0000

---------------------------------------------------------Output------------------------------------------------------------------------

print\_score(classifier2,X\_train,y\_train,X\_test,y\_test,train=False)

---------------------------------------------------------Output------------------------------------------------------------------------

Test results:

Accuracy Score: 0.9990

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2111

1 1.00 1.00 1.00 1951

avg / total 1.00 1.00 1.00 4062

Confusion Matrix:

[[2111 0]

[ 4 1947]]

---------------------------------------------------------Output------------------------------------------------------------------------

**Code for K nearest Neighbor:**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

from matplotlib import pyplot as plt

mushrooms = 'C:/Users/Jonathan/Downloads/mushrooms.csv'

data = pd.read\_csv(mushrooms)

# seperating X vaules from y values

X= data.iloc[:,1:]

y = data.iloc[:,0]

from sklearn.preprocessing import LabelEncoder

from collections import defaultdict

d = defaultdict (LabelEncoder)

Xfit = X.apply(lambda x: d[x.name].fit\_transform(x))

le\_y = LabelEncoder()

yfit = le\_y.fit\_transform(y)

# for x in Xfit.columns:

# print(x)

# print(Xfit[x].value\_counts())

import warnings

warnings.filterwarnings("ignore")

from sklearn.preprocessing import OneHotEncoder

ohc = defaultdict (OneHotEncoder)

# Xfit\_ohc = Xfit.apply(lambda x: ohc[x.name].fit\_transform(x))

final = pd.DataFrame()

for i in range(22):

# transforming the columns using One hot encoder

Xtemp\_i = pd.DataFrame(ohc[Xfit.columns[i]].fit\_transform(Xfit.iloc[:,i:i+1]).toarray())

#Naming the columns as per label encoder

ohc\_obj = ohc[Xfit.columns[i]]

labelEncoder\_i= d[Xfit.columns[i]]

Xtemp\_i.columns= Xfit.columns[i]+"\_"+labelEncoder\_i.inverse\_transform(ohc\_obj.active\_features\_)

# taking care of dummy variable trap

X\_ohc\_i = Xtemp\_i.iloc[:,1:]

#appending the columns to final dataframe

final = pd.concat([final,X\_ohc\_i],axis=1)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(final, yfit, test\_size = 0.7, random\_state = 0)

#testing k nearest neighbors w/ K = 30

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=30,p=2, metric='minkowski')

classifier.fit(X\_train,y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(y\_test,y\_pred)

cm

---------------------------------------------------------Output 1------------------------------------------------------------------------

array([[2963, 5],

[ 6, 2713]], dtype=int64)

---------------------------------------------------------Output 1------------------------------------------------------------------------

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,y\_pred)

---------------------------------------------------------Output 2------------------------------------------------------------------------

0.9980657640232108

---------------------------------------------------------Output 2------------------------------------------------------------------------

classif = KNeighborsClassifier(n\_neighbors=4,p=2, metric='minkowski')

classif.fit(X\_train,y\_train)

y\_pred = classif.predict(X\_test)

accuracy\_score(y\_test,y\_pred)

---------------------------------------------------------Output 3------------------------------------------------------------------------

0.9989449621944786

---------------------------------------------------------Output 3------------------------------------------------------------------------

from sklearn.model\_selection import cross\_val\_score

# creating odd list of K for KNN

myList = list(range(1,4))

# empty list that will hold cv scores

cv\_scores = []

# perform 10-fold cross validation

for k in myList[::2]:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=10, scoring='accuracy')

cv\_scores.append(scores.mean())

# changing to misclassification error

MSE = [1 - x for x in cv\_scores]

# determining best k

optimal\_k = myList[::2][MSE.index(min(MSE))]

print ("The optimal number of neighbors is %d" % optimal\_k)

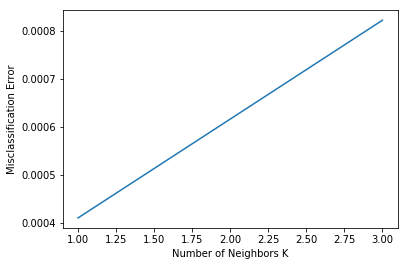
# plot misclassification error vs k

plt.plot(myList[::2], MSE)

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.show()



from sklearn.model\_selection import cross\_val\_score

# creating odd list of K for KNN

myList = list(range(1,4))

# empty list that will hold cv scores

cv\_scores = []

# perform 20-fold cross validation

for k in myList[::2]:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=2, scoring='accuracy')

cv\_scores.append(scores.mean())

# changing to misclassification error

MSE = [1 - x for x in cv\_scores]

# determining best k

optimal\_k = myList[::2][MSE.index(min(MSE))]

print ("The optimal number of neighbors is %d" % optimal\_k)

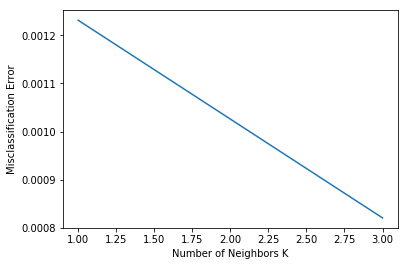
# plot misclassification error vs k

plt.plot(myList[::2], MSE)

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.show()



from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(final, yfit, test\_size = 0.8, random\_state = 0)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=30,p=2, metric='minkowski')

classifier.fit(X\_train,y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(y\_test,y\_pred)

cm

---------------------------------------------------------Output 4------------------------------------------------------------------------

array([[3313, 52],

[ 6, 3129]], dtype=int64)

---------------------------------------------------------Output 4------------------------------------------------------------------------

from sklearn.model\_selection import cross\_val\_score

# creating odd list of K for KNN

myList = list(range(1,4))

# empty list that will hold cv scores

cv\_scores = []

# perform 10-fold cross validation

for k in myList[::2]:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=10, scoring='accuracy')

cv\_scores.append(scores.mean())  
# changing to misclassification error

MSE = [1 - x for x in cv\_scores]

# determining best k

optimal\_k = myList[::2][MSE.index(min(MSE))]

print ("The optimal number of neighbors is %d" % optimal\_k)

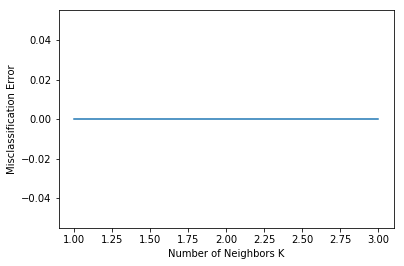
# plot misclassification error vs k

plt.plot(myList[::2], MSE)

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.show()



from sklearn.model\_selection import cross\_val\_score

# creating odd list of K for KNN

myList = list(range(1,4))

# empty list that will hold cv scores

cv\_scores = []

# perform 2-fold cross validation

for k in myList[::2]:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=2, scoring='accuracy')

cv\_scores.append(scores.mean())

# changing to misclassification error

MSE = [1 - x for x in cv\_scores]

# determining best k

optimal\_k = myList[::2][MSE.index(min(MSE))]

print ("The optimal number of neighbors is %d" % optimal\_k)

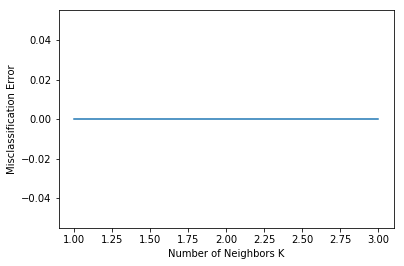
# plot misclassification error vs k

plt.plot(myList[::2], MSE)

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.show()



**#Linear Regression with PCA (Principle Component Analysis)**

pca\_modified=PCA(n\_components=17)

pca\_modified.fit\_transform(X)

---------------------------------------------------------Output 3----------------------------------------------------------------------

array([[-0.5743219 , -0.97578135, -1.22176154, ..., -0.51996599,

-0.78254366, 1.12025933],

[-2.2821023 , 0.27906633, -1.20049669, ..., -0.11307822,

-0.73093408, -0.01817413],

[-1.85803562, -0.27097236, -1.37237069, ..., 0.01652548,

-0.6561675 , 0.10791396],

...,

[-1.62151632, -0.75753671, 2.73357994, ..., -0.51961303,

-0.70768708, 0.22578534],

[ 3.67060561, -1.0327745 , 0.1684595 , ..., -0.08688401,

-0.11464249, -0.14801392],

[-1.57520272, -1.2285814 , 2.44722789, ..., 0.91606764,

-0.77988482, -0.30141893]])

---------------------------------------------------------Output 3----------------------------------------------------------------------

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=4)

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import cross\_val\_score

from sklearn import metrics

#creating Logistical (Linear) Regression model

model\_LR= LogisticRegression()

model\_LR.fit(X\_train,y\_train)

---------------------------------------------------------Output----------------------------------------------------------------------

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,

penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,

verbose=0, warm\_start=False)

---------------------------------------------------------Output----------------------------------------------------------------------

y\_prob = model\_LR.predict\_proba(X\_test)[:,1] # This will give you positive class prediction probabilities

y\_pred = np.where(y\_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.

model\_LR.score(X\_test, y\_pred)

confusion\_matrix=metrics.confusion\_matrix(y\_test,y\_pred)

confusion\_matrix

---------------------------------------------------------Output----------------------------------------------------------------------

array([[814, 31],

[ 37, 743]], dtype=int64)

---------------------------------------------------------Output----------------------------------------------------------------------

auc\_roc=metrics.roc\_auc\_score(y\_test,y\_pred)

auc\_roc

---------------------------------------------------------Output----------------------------------------------------------------------

0.9579388560157791

---------------------------------------------------------Output----------------------------------------------------------------------

from sklearn.metrics import roc\_curve, auc

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

roc\_auc

---------------------------------------------------------Output----------------------------------------------------------------------

0.9902988924290701

---------------------------------------------------------Output----------------------------------------------------------------------

from sklearn.model\_selection import cross\_val\_score

from sklearn import metrics

LR\_model= LogisticRegression()

tuned\_parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] ,

'penalty':['l1','l2']

}

LR.fit(X\_train,y\_train)

y\_prob = LR.predict\_proba(X\_test)[:,1] # This will give you positive class prediction probabilities

y\_pred = np.where(y\_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.

LR.score(X\_test, y\_pred)

confusion\_matrix=metrics.confusion\_matrix(y\_test,y\_pred)

confusion\_matrix

---------------------------------------------------------Output----------------------------------------------------------------------

array([[824, 21],

[ 23, 757]], dtype=int64)

---------------------------------------------------------Output----------------------------------------------------------------------

auc\_roc=metrics.roc\_auc\_score(y\_test,y\_pred)

auc\_roc

---------------------------------------------------------Output----------------------------------------------------------------------

0.9728303747534518

---------------------------------------------------------Output----------------------------------------------------------------------

from sklearn.metrics import roc\_curve, auc

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

roc\_auc

---------------------------------------------------------Output----------------------------------------------------------------------

0.991624943104233

---------------------------------------------------------Output----------------------------------------------------------------------

…..

…..

**#The same methods using only 2 cross validations and a new train test split 60/40, results outlined in report**

**Code for Non Linear Regression**

**Code for K Means**

dataset['class'].unique() # records unique occurrences in the class variable (poisonous vs edible)

dataset['stalk-color-above-ring'].unique() #unique values in this variable

…

X = dataset.iloc[:,1:23] # all rows, all the features and no labels

y = dataset.iloc[:, 0] # all rows, label only

…

dataset.corr() #prints table with the correlation between variables

# Scale the data to be between -1 and 1

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X=scaler.fit\_transform(X)

X

---------------------------------------------------------Output 1----------------------------------------------------------------------

array([[ 1.02971224, 0.14012794, -0.19824983, ..., -0.67019486,

-0.5143892 , 2.03002809],

[ 1.02971224, 0.14012794, 1.76587407, ..., -0.2504706 ,

-1.31310821, -0.29572966],

[-2.08704716, 0.14012794, 1.37304929, ..., -0.2504706 ,

-1.31310821, 0.86714922],

...,

[-0.8403434 , 0.14012794, -0.19824983, ..., -1.50964337,

-2.11182722, 0.28570978],

[-0.21699152, 0.95327039, -0.19824983, ..., 1.42842641,

0.28432981, 0.28570978],

[ 1.02971224, 0.14012794, -0.19824983, ..., 0.16925365,

-2.11182722, 0.28570978]])

---------------------------------------------------------Output 1----------------------------------------------------------------------

from sklearn.decomposition import PCA

pca = PCA()

pca.fit\_transform(X)

---------------------------------------------------------Output 2----------------------------------------------------------------------

array([[-5.74321902e-01, -9.75781349e-01, -1.22176154e+00, ...,

-2.08581362e-01, 8.13996758e-03, -2.98716006e-17],

[-2.28210230e+00, 2.79066333e-01, -1.20049669e+00, ...,

1.52238789e-01, -1.96446132e-01, 8.17047124e-17],

[-1.85803562e+00, -2.70972362e-01, -1.37237069e+00, ...,

2.57581784e-01, -3.62577199e-01, 3.62451933e-17],

...,

[-1.62151632e+00, -7.57536709e-01, 2.73357994e+00, ...,

-1.42532241e+00, 6.36990122e-01, 1.31607921e-18],

[ 3.67060561e+00, -1.03277450e+00, 1.68459501e-01, ...,

9.41440123e-02, -6.43462238e-02, -7.47641463e-20],

[-1.57520272e+00, -1.22858140e+00, 2.44722789e+00, ...,

-8.04626064e-01, 5.90315263e-01, 1.39554149e-19]])

---------------------------------------------------------Output 2----------------------------------------------------------------------

#plotting data on a scatterplot using the numerical transformation of the original data

N=dataset.values

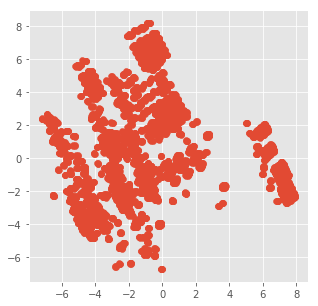
pca = PCA(n\_components=2)

x = pca.fit\_transform(N)

plt.figure(figsize = (5,5))

plt.scatter(x[:,0],x[:,1])

plt.show()



**# K = 2 clusters**

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=2, random\_state=5)

X\_clustered = kmeans.fit\_predict(N)

LABEL\_COLOR\_MAP = {0 : 'g',

1 : 'y',

2 : 'r',

3: 'b'

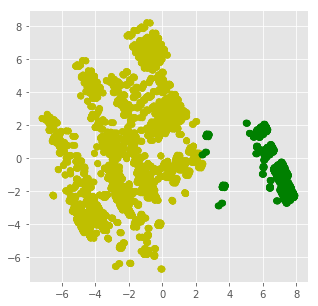
}

label\_color = [LABEL\_COLOR\_MAP[l] for l in X\_clustered]

plt.figure(figsize = (5,5))

plt.scatter(x[:,0],x[:,1], c= label\_color)

plt.show()



**#K = 3 Clusters**

kmeans = KMeans(n\_clusters=3, random\_state=5)

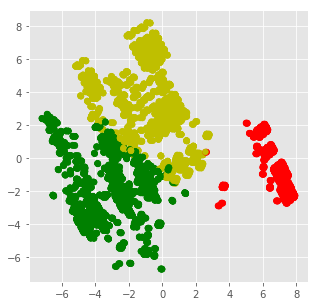
X\_clustered = kmeans.fit\_predict(N)

label\_color = [LABEL\_COLOR\_MAP[l] for l in X\_clustered]

plt.figure(figsize = (5,5))

plt.scatter(x[:,0],x[:,1], c= label\_color)

plt.show()



**#K = 4 clusters**

kmeans = KMeans(n\_clusters=4, random\_state=5)

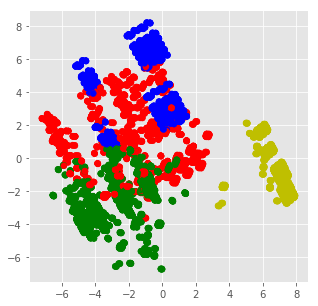
X\_clustered = kmeans.fit\_predict(N)

label\_color = [LABEL\_COLOR\_MAP[l] for l in X\_clustered]

plt.figure(figsize = (5,5))

plt.scatter(x[:,0],x[:,1], c= label\_color)

plt.show()



**Code for Hierarchical Clustering**

import matplotlib.pyplot as plt

import pandas as pd

%matplotlib inline

import numpy as np

mushrooms = 'C:/Users/Jonathan/Downloads/mushrooms.csv'

dataset=pd.read\_csv(mushrooms)

from sklearn.preprocessing import LabelEncoder

labelencoder=LabelEncoder()

for col in dataset.columns:

dataset[col] = labelencoder.fit\_transform(dataset[col])

X = dataset.iloc[:,0:23] # all rows, all the features and labels

---------------------------------------------------------Output 1----------------------------------------------------------------------

array([[ 1.0366127 , 1.02971224, 0.14012794, ..., -0.67019486,

-0.5143892 , 2.03002809],

[-0.96468045, 1.02971224, 0.14012794, ..., -0.2504706 ,

-1.31310821, -0.29572966],

[-0.96468045, -2.08704716, 0.14012794, ..., -0.2504706 ,

-1.31310821, 0.86714922],

...,

[-0.96468045, -0.8403434 , 0.14012794, ..., -1.50964337,

-2.11182722, 0.28570978],

[ 1.0366127 , -0.21699152, 0.95327039, ..., 1.42842641,

0.28432981, 0.28570978],

[-0.96468045, 1.02971224, 0.14012794, ..., 0.16925365,

-2.11182722, 0.28570978]])

---------------------------------------------------------Output 1----------------------------------------------------------------------

from sklearn.decomposition import PCA

pca = PCA()

pca.fit\_transform(X)

---------------------------------------------------------Output 2----------------------------------------------------------------------

array([[-3.21350852e-01, -6.94030533e-01, -2.35421304e-01, ...,

-1.90098038e-01, 5.81245340e-02, 1.20049002e-17],

[-2.47697655e+00, -8.02019582e-02, 8.03020443e-01, ...,

1.23244655e-01, 1.37769748e-01, 9.15787295e-18],

[-2.17990553e+00, -7.70028186e-01, 6.17134078e-01, ...,

2.12242019e-01, 2.95473906e-01, 1.47857572e-17],

...,

[-1.89106512e+00, 1.51835675e-01, -1.59777499e+00, ...,

-1.40391995e+00, -8.22748173e-01, -4.11457227e-19],

[ 3.72914776e+00, -1.12122520e+00, -7.19805204e-01, ...,

9.64043340e-02, 9.85382596e-02, -6.65878887e-20],

[-1.94095417e+00, -3.97285454e-01, -1.73096408e+00, ...,

-7.62980716e-01, -6.46329807e-01, 6.05361613e-20]])

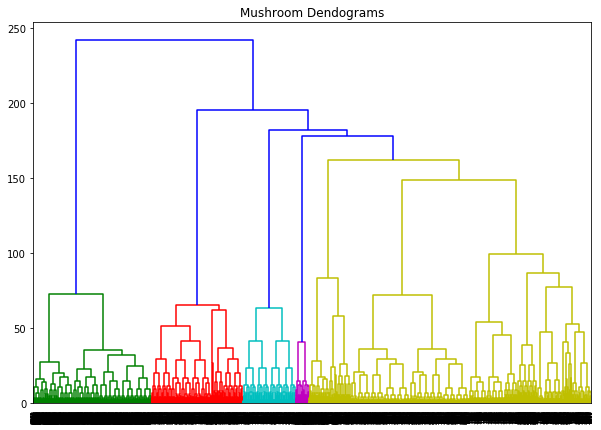
---------------------------------------------------------Output 2----------------------------------------------------------------------

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))

plt.title("Customer Dendograms")

dend = shc.dendrogram(shc.linkage(X, method='ward'))



from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=7, affinity='euclidean', linkage='ward')

cluster.fit\_predict(X)

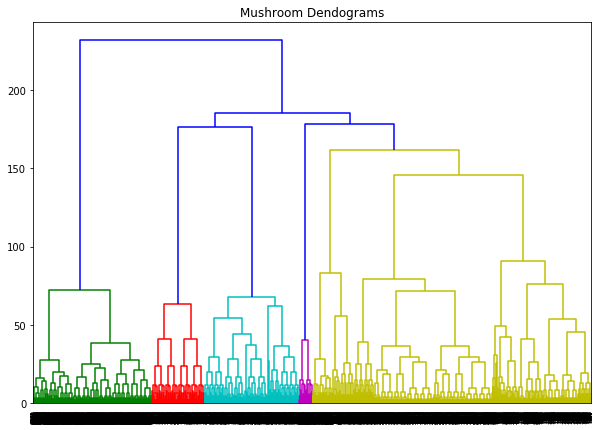
plt.figure(figsize=(10, 7))

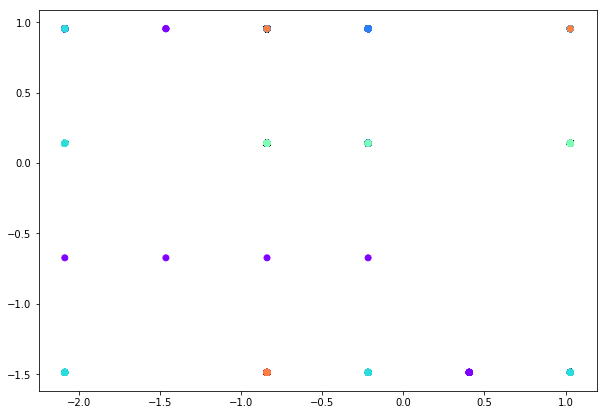
plt.scatter(X[:,0], X[:,1], c=cluster.labels\_, cmap='rainbow')

A scatter plot of 7 classes defined by the dendrogram



…. The same was repeated but excluding the class (poisonous vs edible ) in the data





**Code for Neural Network :**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt # data visualizations

from sklearn.preprocessing import LabelEncoder # label encoder

from sklearn.model\_selection import train\_test\_split # Splitter

mushrooms = 'C:/Users/Jonathan/Downloads/mushrooms.csv'

df = pd.read\_csv(mushrooms)

label\_encoder = LabelEncoder()

df = df.apply(label\_encoder.fit\_transform) # label encoding

#

y = df["class"].values # our labels.. okay to eat or poison.

df.drop(["class"],axis=1,inplace=True) # dropping the lables from the data

df.drop(["veil-color"],axis=1,inplace=True)

df.drop(["veil-type"],axis=1,inplace=True)

x\_data = df # our features..

x = (x\_data - np.min(x\_data))/(np.max(x\_data)-np.min(x\_data)).values # normalization

#

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42) # 20% would be enough

x\_train = x\_train.values.T

x\_test = x\_test.values.T

y\_train = y\_train.reshape(-1,1).T

y\_test = y\_test.reshape(-1,1).T

def initialize\_param(layers\_with\_nodes,in\_data,out\_data):

# we start with initializing the input layer's values.

layer\_amount = len(layers\_with\_nodes)

parameters = {}

parameters["weight1"] = np.random.randn(layers\_with\_nodes[0],in\_data.shape[0]) \* 0.1

parameters["bias1"] = np.zeros((layers\_with\_nodes[0],1))

print("" + str(layer\_amount+2) + " layers. (" + str(layer\_amount) + " hidden layer)")

# then we initialize the hidden layer's values.

for i in range(layer\_amount-1):

#print(i+2)

w = "weight" + str(i+2)

b = "bias" + str(i+2)

parameters[w] = np.random.randn(layers\_with\_nodes[i+1],layers\_with\_nodes[i]) \* 0.1

parameters[b] = np.zeros((layers\_with\_nodes[i+1],1))

# and lastly output layer's values.

lastw = "weight" + str(layer\_amount+1)

lastb = "bias" + str(layer\_amount+1)

parameters[lastw] = np.random.randn(out\_data.shape[0],layers\_with\_nodes[layer\_amount-1]) \* 0.1

parameters[lastb] = np.zeros((out\_data.shape[0],1))

return parameters, (layer\_amount+1)

def sigmoid(x):

y\_head = 1/(1 + np.exp(-x)) # basic sigmoid for output layer

return y\_head;

def compute\_cost(A\_f, y\_train):

logprobs = np.multiply(np.log(A\_f),y\_train)

cost = -np.sum(logprobs)/y\_train.shape[1] # computing the loss value, so we can improve our success more in the future.

return cost

def forw\_prop\_NN(in\_data, parameters, connection\_amount):

cache = {}

a0 = in\_data

# forward propagation until last layer is same because we use tanh for all except output layer.

for i in range(connection\_amount-1):

w = "weight" + str(i+1)

b = "bias" + str(i+1)

z = "Z" + str(i+1)

a = "A" + str(i+1)

cache[z] = np.dot(parameters[w],a0) + parameters[b]

cache[a] = np.tanh(cache[z])

a0 = cache[a]

# then on the output(last) layer we use sigmoid.

z\_fin = "Z" + str(connection\_amount)

a\_fin = "A" + str(connection\_amount)

w = "weight" + str(connection\_amount)

b = "bias" + str(connection\_amount)

a0 = "A" + str(connection\_amount-1)

cache[z\_fin] = np.dot(parameters[w],cache[a0]) + parameters[b]

cache[a\_fin] = sigmoid(cache[z\_fin])

A = cache[a\_fin]

return A, cache

def backw\_prop\_NN(parameters,cache,x\_train,y\_train,connection\_amount):

dz = {}

grads = {}

# since we are going backwards on back propagation, we update values from last to first

# so firstly we will start with output layer.

a\_fin = "A" + str(connection\_amount)

a\_pre\_fin = "A" + str(connection\_amount-1)

w\_fin = "dweight" + str(connection\_amount)

b\_fin = "dbias" + str(connection\_amount)

dz[connection\_amount] = cache[a\_fin] - y\_train

grads[w\_fin] = np.dot(dz[connection\_amount],cache[a\_pre\_fin].T)/x\_train.shape[1]

grads[b\_fin] = np.sum(dz[connection\_amount], axis=1, keepdims=True)/x\_train.shape[1]

# then continue with the rest of layers.

for i in range((connection\_amount-1),1,-1):

a = "A" + str(i)

a\_pre = "A" + str(i-1)

dw = "dweight" + str(i)

db = "dbias" + str(i)

w = "weight" + str(i+1)

dz[i] = np.dot(parameters[w].T,dz[i+1])\*(1 - np.power(cache[a], 2))

grads[dw] = np.dot(dz[i],cache[a\_pre].T)/x\_train.shape[1]

grads[db] = np.sum(dz[i], axis=1,keepdims=True)/x\_train.shape[1]

dz[1] = np.dot(parameters["weight2"].T,dz[2])\*(1 - np.power(cache["A1"], 2))

grads["dweight1"] = np.dot(dz[1],x\_train.T)/x\_train.shape[1]

grads["dbias1"] = np.sum(dz[1],axis =1,keepdims=True)/x\_train.shape[1]

return grads

def update\_param(parameters, grads, connection\_amount, lr=0.01):

for i in range(connection\_amount):

w = "weight" + str(i+1)

b = "bias" + str(i+1)

dw = "dweight" + str(i+1)

db = "dbias" + str(i+1)

parameters[w] = parameters[w] - lr\*grads[dw]

parameters[b] = parameters[b] - lr\*grads[db]

return parameters # basic update for all layers...

def predict(A,parameters,x\_test,connection\_amount):

# x\_test is a input for forward propagation

A, cache = forw\_prop\_NN(x\_test,parameters,connection\_amount)

Y\_prediction = np.zeros((1,x\_test.shape[1]))

for i in range(A.shape[1]):

if A[0,i]<= 0.5:

Y\_prediction[0,i] = 0

else:

Y\_prediction[0,i] = 1

return Y\_prediction # basic binary prediction...

def multi\_layer\_model(layers\_with\_nodes,x\_train,y\_train,x\_test,y\_test, num\_iter, lr = 0.01): # only learning rate is pre-defined

cost\_list = []

index\_list = []

parameters, connection\_amount = initialize\_param(layers\_with\_nodes, x\_train, y\_train) # starting with initializing for only once

# then do the following part for each iteration..

for i in range(0, num\_iter):

# forward propagation

A, cache = forw\_prop\_NN(x\_train,parameters,connection\_amount)

# compute cost

cost = compute\_cost(A, y\_train)

# backward propagation

grads = backw\_prop\_NN(parameters,cache,x\_train,y\_train,connection\_amount)

# update parameters

parameters = update\_param(parameters, grads, connection\_amount, lr)

if i % 100 == 0:

cost\_list.append(cost)

index\_list.append(i)

print ("Cost after iteration %i: %f" %(i, cost))

plt.plot(index\_list,cost\_list)

plt.xticks(index\_list,rotation='vertical')

plt.xlabel("Number of Iterarion")

plt.ylabel("Cost")

plt.show()

# predict

y\_prediction\_train = predict(A, parameters, x\_train, connection\_amount)

y\_prediction\_test = predict(A, parameters, x\_test, connection\_amount)

# Print train/test Errors

print("train accuracy: {} %".format(100 - np.mean(np.abs(y\_prediction\_train - y\_train)) \* 100))

print("test accuracy: {} %".format(100 - np.mean(np.abs(y\_prediction\_test - y\_test)) \* 100))

multi\_layer\_model([20,4],x\_train,y\_train,x\_test,y\_test,num\_iter=3001,lr=0.03)

# the first array is where we desing our net for hidden layers.

# for example layers\_with\_nodes=[4] means one hidden layer with 4 nodes, or layers\_with\_nodes=[10,5] means two hidden layers: first layer with 10 and second with 5 nodes

# lr is learning rate as usual

# num\_iter is number of iterations that we want our model to train.

---------------------------------------------------------Output ----------------------------------------------------------------------

4 layers. (2 hidden layer)

Cost after iteration 0: 0.335326

Cost after iteration 100: 0.345566

Cost after iteration 200: 0.345429

Cost after iteration 300: 0.325646

Cost after iteration 400: 0.271771

Cost after iteration 500: 0.205570

Cost after iteration 600: 0.165133

Cost after iteration 700: 0.143695

Cost after iteration 800: 0.127595

Cost after iteration 900: 0.114244

Cost after iteration 1000: 0.104820

Cost after iteration 1100: 0.098587

Cost after iteration 1200: 0.094336

Cost after iteration 1300: 0.091276

Cost after iteration 1400: 0.088947

Cost after iteration 1500: 0.087074

Cost after iteration 1600: 0.085482

Cost after iteration 1700: 0.084051

Cost after iteration 1800: 0.082702

Cost after iteration 1900: 0.081382

Cost after iteration 2000: 0.080058

Cost after iteration 2100: 0.078722

Cost after iteration 2200: 0.077383

Cost after iteration 2300: 0.076067

Cost after iteration 2400: 0.074805

Cost after iteration 2500: 0.073624

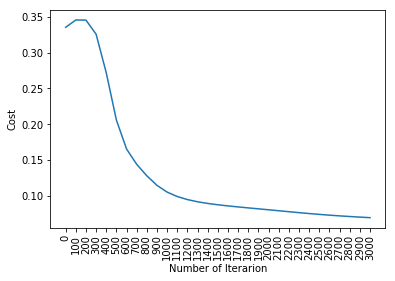
Cost after iteration 2600: 0.072538

Cost after iteration 2700: 0.071547

Cost after iteration 2800: 0.070640

Cost after iteration 2900: 0.069807

Cost after iteration 3000: 0.069032



train accuracy: 96.26096322511155 %

test accuracy: 96.24615384615385 %

---------------------------------------------------------Output ----------------------------------------------------------------------

**Code for Naïve Bayes, Random Forest, and Support Vector Machine**

%matplotlib inline

%config InlineBackend.figure\_format = 'retina'

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import math

mushrooms = 'C:/Users/Jonathan/Downloads/mushrooms.csv'

data = pd.read\_csv(mushrooms)

Y = pd.get\_dummies(data.iloc[:,0], drop\_first=False)

X = pd.DataFrame()

for each in data.iloc[:,1:].columns:

dummies = pd.get\_dummies(data[each], prefix=each, drop\_first=False)

X = pd.concat([X, dummies], axis=1)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=0)

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

models = []

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC(probability=True)))

models.append(('RF', RandomForestClassifier()))

from sklearn.model\_selection import cross\_val\_score, KFold

seed = 321

# evaluating each model in turn

results = []

names = []

for name, model in models:

kfold = KFold(n\_splits=10, random\_state=seed)

cv\_results = cross\_val\_score(model, X\_train, y\_train.iloc[:,1], cv=kfold, scoring='roc\_auc')

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

---------------------------------------------------------Output ----------------------------------------------------------------------

NB: 0.997965 (0.002335)

SVM: 0.999938 (0.000086)

RF: 1.000000 (0.000000)

---------------------------------------------------------Output ----------------------------------------------------------------------

# Compare Algorithms

fig = plt.figure(figsize=(16, 8))

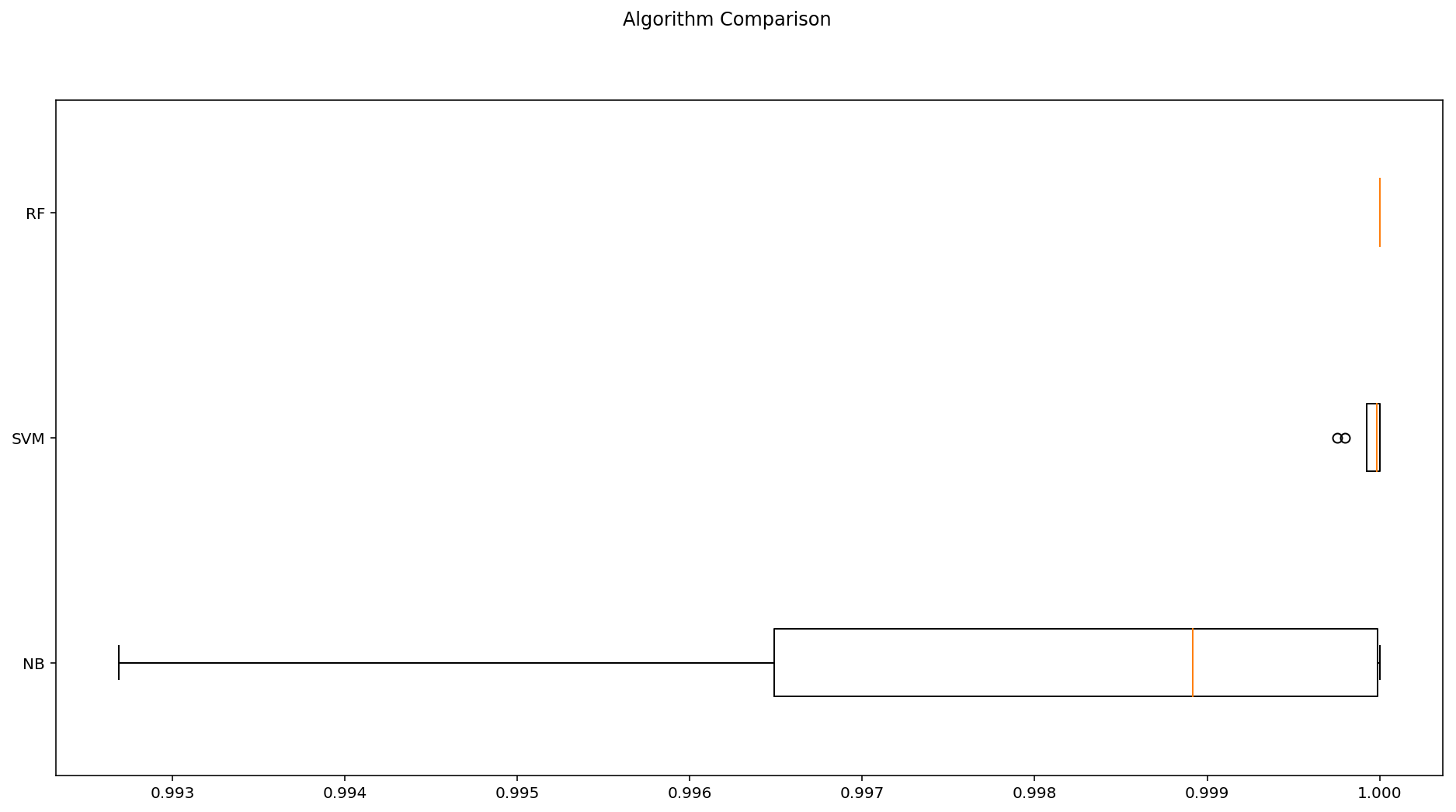
fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results, vert=False)

ax.set\_yticklabels(names)

plt.show()



from collections import defaultdict

from sklearn.metrics import roc\_auc\_score

model\_predictions = defaultdict()

model\_score = defaultdict(np.float)

for name, model in models:

model.fit(X\_train, y\_train.iloc[:,1])

my\_pred = model.predict(X\_test)

model\_predictions[name] = my\_pred

model\_score[name] = roc\_auc\_score(y\_test.iloc[:,1], my\_pred)

msg = "%s: %f" % (name, model\_score[name])

print(msg)

model\_predicions\_df = pd.DataFrame(model\_predictions)

corrmat = model\_predicions\_df.corr()

corrmat

---------------------------------------------------------Output ----------------------------------------------------------------------

NB: 0.989437

SVM: 1.000000

RF: 1.000000

|  | **NB** | **SVM** | **RF** |
| --- | --- | --- | --- |
| **NB** | 1.000000 | 0.978058 | 0.978058 |
| **SVM** | 0.978058 | 1.000000 | 1.000000 |
| **RF** | 0.978058 | 1.000000 | 1.000000 |