**Ensemble voting :**

# coding: utf-8

# ## Introduction

# Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms.

#

# It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data

# In[1]:

import pandas

from sklearn import model\_selection

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

# In[4]:

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

print "Length of dataset is -",dataframe.shape[0]

print "No of columns are - ",dataframe.shape[1]

print "\n\n",dataframe

# In[22]:

array = dataframe.values

print array

print array.shape

X = array[:,0:8]

# First part denotes how many rows we want. Second part denotes how many columns.

Y = array[:,8]

# In[24]:

seed = 7

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

# create the sub models

# ## Using 3 models

# - ### 1. Logistic Regression

#

# In[25]:

estimators = []

model1 = LogisticRegression()

estimators.append(('logistic', model1))

# - ### 2. Decision Tree

# In[26]:

model2 = DecisionTreeClassifier()

estimators.append(('cart', model2))

# - ### 3. SVM

# In[27]:

model3 = SVC()

estimators.append(('svm', model3))

# In[34]:

# create the ensemble model

ensemble = VotingClassifier(estimators)

results = model\_selection.cross\_val\_score(ensemble, X, Y, cv=kfold)

print("Here is the score - ",results.mean()\*100)

**Output error code:**

# coding: utf-8

# In[1]:

from sklearn import datasets

from sklearn.multiclass import OutputCodeClassifier

from sklearn.svm import LinearSVC

import pandas

import numpy as np

# In[4]:

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

print "Length of dataset is -",dataframe.shape[0]

print "No of columns are - ",dataframe.shape[1]

print "\n\n",dataframe

# In[5]:

array = dataframe.values

print array

print array.shape

X = array[:,0:8]

# First part denotes how many rows we want. Second part denotes how many columns.

Y = array[:,8]

# In[7]:

clf = OutputCodeClassifier(LinearSVC(random\_state=0),

code\_size=2, random\_state=0)

clf.fit(X, Y)

pred = clf.predict(X)

print "Here is the score - ",clf.score(X,Y)\*100

**Decision tree :**

# coding: utf-8

# In[104]:

import pandas as pd

from sklearn.cross\_validation import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn import tree

df = pd.read\_csv('dataset.csv')

# In[105]:

df.head()

# # How to handle missing values

# - use dropna as df.dropna(how="any" or "all", inplace=True)

# - use df.fillna(value = -99999,inplace=True)

# - or use df.fillna(method='ffill',inplace=True). So this takes previous values and fill them in in next nan value

# - or use df.fillna(method="bfill',inplace=True). So this takes future values and fill them in in next nan value

# In[106]:

df=df.dropna(axis=0, how='any')

len(df)

# In[107]:

df['Type'] = pd.Categorical(df.Type).codes

df['Reliability'] = pd.Categorical(df.Reliability).codes

df['Country'] = pd.Categorical(df.Country).codes

# \*\*We can also use LabelEncoding or One Hot Encoding for converting categorical values into integers as shown below\*\*

# In[98]:

# from sklearn import preprocessing

# le = preprocessing.LabelEncoder()

# le.fit(df['Type'])

# typeConvertor = le.transform(df['Type'])

# le.fit(df['Country'])

# countryConvertor = le.transform(df['Country'])

# le.fit(df['Reliability'])

# reliabilityConvertor = le.transform(df['Reliability'])

# print reliabilityConvertor

# print countryConvertor

# In[108]:

Y = df['Mileage']

X = df.drop(['Mileage','Name'],axis=1)

# In[113]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, Y, test\_size = 0.15, random\_state = 100)

# In[114]:

clf = tree.DecisionTreeClassifier()

clf = clf.fit(X\_train, y\_train)

# In[116]:

pred = clf.predict(X\_test)

accuracy = accuracy\_score(pred,y\_test)

print("Accuracy is ",accuracy\*100)

# In[ ]:

**GMM :**

import numpy as np

import pandas as pd

import random as rand

import matplotlib.pyplot as plt

from scipy.stats import norm

from sys import maxint

### Setup

# set random seed

rand.seed(42)

# 2 clusters

# not that both covariance matrices are diagonal

mu1 = [0, 5]

sig1 = [ [2, 0], [0, 3] ]

mu2 = [5, 0]

sig2 = [ [4, 0], [0, 1] ]

# generate samples

x1, y1 = np.random.multivariate\_normal(mu1, sig1, 100).T

x2, y2 = np.random.multivariate\_normal(mu2, sig2, 100).T

xs = np.concatenate((x1, x2))

ys = np.concatenate((y1, y2))

labels = ([1] \* 100) + ([2] \* 100)

data = {'x': xs, 'y': ys, 'label': labels}

df = pd.DataFrame(data=data)

# inspect the data

df.head()

df.tail()

fig = plt.figure()

plt.scatter(data['x'], data['y'], 24, c=data['label'])

fig.savefig("true-values.png")

### Expectation-maximization

# initial guesses - intentionally bad

guess = { 'mu1': [1,1],

'sig1': [ [1, 0], [0, 1] ],

'mu2': [4,4],

'sig2': [ [1, 0], [0, 1] ],

'lambda': [0.4, 0.6]

}

# probability that a point came from a Guassian with given parameters

# note that the covariance must be diagonal for this to work

def prob(val, mu, sig, lam):

p = lam

for i in range(len(val)):

p \*= norm.pdf(val[i], mu[i], sig[i][i])

return p

# assign every data point to its most likely cluster

def expectation(dataFrame, parameters):

for i in range(dataFrame.shape[0]):

x = dataFrame['x'][i]

y = dataFrame['y'][i]

p\_cluster1 = prob([x, y], list(parameters['mu1']), list(parameters['sig1']), parameters['lambda'][0] )

p\_cluster2 = prob([x, y], list(parameters['mu2']), list(parameters['sig2']), parameters['lambda'][1] )

if p\_cluster1 > p\_cluster2:

dataFrame['label'][i] = 1

else:

dataFrame['label'][i] = 2

return dataFrame

# update estimates of lambda, mu and sigma

def maximization(dataFrame, parameters):

points\_assigned\_to\_cluster1 = dataFrame[dataFrame['label'] == 1]

points\_assigned\_to\_cluster2 = dataFrame[dataFrame['label'] == 2]

percent\_assigned\_to\_cluster1 = len(points\_assigned\_to\_cluster1) / float(len(dataFrame))

percent\_assigned\_to\_cluster2 = 1 - percent\_assigned\_to\_cluster1

parameters['lambda'] = [percent\_assigned\_to\_cluster1, percent\_assigned\_to\_cluster2 ]

parameters['mu1'] = [points\_assigned\_to\_cluster1['x'].mean(), points\_assigned\_to\_cluster1['y'].mean()]

parameters['mu2'] = [points\_assigned\_to\_cluster2['x'].mean(), points\_assigned\_to\_cluster2['y'].mean()]

parameters['sig1'] = [ [points\_assigned\_to\_cluster1['x'].std(), 0 ], [ 0, points\_assigned\_to\_cluster1['y'].std() ] ]

parameters['sig2'] = [ [points\_assigned\_to\_cluster2['x'].std(), 0 ], [ 0, points\_assigned\_to\_cluster2['y'].std() ] ]

return parameters

# get the distance between points

# used for determining if params have converged

def distance(old\_params, new\_params):

dist = 0

for param in ['mu1', 'mu2']:

for i in range(len(old\_params)):

dist += (old\_params[param][i] - new\_params[param][i]) \*\* 2

return dist \*\* 0.5

# loop until parameters converge

shift = maxint

epsilon = 0.01

iters = 0

df\_copy = df.copy()

# randomly assign points to their initial clusters

df\_copy['label'] = map(lambda x: x+1, np.random.choice(2, len(df)))

params = pd.DataFrame(guess)

while shift > epsilon:

iters += 1

# E-step

updated\_labels = expectation(df\_copy.copy(), params)

# M-step

updated\_parameters = maximization(updated\_labels, params.copy())

# see if our estimates of mu have changed

# could incorporate all params, or overall log-likelihood

shift = distance(params, updated\_parameters)

# logging

print("iteration {}, shift {}".format(iters, shift))

# update labels and params for the next iteration

df\_copy = updated\_labels

params = updated\_parameters

fig = plt.figure()

plt.scatter(df\_copy['x'], df\_copy['y'], 24, c=df\_copy['label'])

fig.savefig(“iteration{}.png".format(iters))

KMEANS and kmode :

# coding: utf-8

# # K-Means

# ### 1. Using make\_blobs dataset :

# In[61]:

import numpy as np

import pandas as pd

import sklearn

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

from sklearn import preprocessing,cross\_validation,neighbors

from sklearn.metrics import accuracy\_score

get\_ipython().magic(u'matplotlib inline')

# In[62]:

plt.rcParams['figure.figsize'] = (16, 9)

# Creating a sample dataset with 4 clusters

X, y = make\_blobs(n\_samples=800, n\_features=3, centers=4)

fig = plt.figure()

ax = Axes3D(fig)

ax.scatter(X[:, 0], X[:, 1], X[:, 2])

# In[63]:

kmeans = KMeans(n\_clusters=4)

kmeans = kmeans.fit(X)

labels = kmeans.predict(X)

C = kmeans.cluster\_centers\_

print C

# In[64]:

fig = plt.figure()

ax = Axes3D(fig)

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)

ax.scatter(C[:, 0], C[:, 1], C[:, 2], marker='\*', c='#050505', s=1000)

# ### 2. Using CSV Dataset :

# In[65]:

import pandas as pd

dataset = pd.read\_csv('knndataset.csv')

dataset.head()

X = dataset.as\_matrix(['Sepal.Length','Sepal.Width','Petal.Length','Petal.Width'])

Y = dataset.as\_matrix(['Species'])

X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(X,Y, test\_size=0.2, random\_state=100)

# In[66]:

k\_means = KMeans(n\_clusters=2)

k\_means.fit(X\_train)

print(k\_means.labels\_[:])

# print(y\_train[:])

# In[67]:

print(k\_means.predict(X\_test))

print("Labels are ",k\_means.labels\_)

print(y\_test[:])

# # K Mode

# In[68]:

import numpy as np

from kmodes.kmodes import KModes

# random categorical data

data = np.random.choice(20, (100, 10))

km = KModes(n\_clusters=4, init='Huang', n\_init=5, verbose=1)

clusters = km.fit\_predict(data)

# Print the cluster centroids

print(km.cluster\_centroids\_)

**KNN**

# coding: utf-8

# In[23]:

import pandas as pd

import numpy as np

from sklearn import neighbors

import sklearn

from sklearn import preprocessing,cross\_validation,neighbors

# In[24]:

dataset = pd.read\_csv('knndataset.csv')

dataset.head()

# In[25]:

X = dataset.as\_matrix(['Sepal.Length','Sepal.Width','Petal.Length','Petal.Width'])

Y = dataset.as\_matrix(['Species'])

X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(X,Y, test\_size=0.4, random\_state=0)

# In[26]:

from sklearn import neighbors

knn = neighbors.KNeighborsClassifier(n\_neighbors=1)

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

print knn.predict(X\_test)

print('Score: ', knn.score(X\_test, y\_test))

**Logistic regression :**

# coding: utf-8

# In[37]:

import numpy as np

import pandas as pd

import scipy.stats as stats

import matplotlib.pyplot as plt

import sklearn

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

import seaborn as sns

from sklearn import preprocessing,cross\_validation,neighbors

from sklearn.metrics import accuracy\_score

# In[38]:

dataset = pd.read\_csv('dataset3.csv')

print dataset

print dataset.dtypes

# # Scatter Plots

# In[39]:

brain = dataset['Brain']

piq = dataset['PIQ']

sns.regplot(x='Brain', y='PIQ', data=dataset, fit\_reg=False)

plt.show()

# In[55]:

x\_train, x\_test, y\_train,y\_test= cross\_validation.train\_test\_split(dataset['Brain'],dataset['PIQ'],test\_size=0.2)

# In[56]:

reg = linear\_model.LinearRegression()

reg.fit(x\_train.values.reshape(-1,1), y\_train.values.reshape(-1,1))

print reg.score(x\_test.values.reshape(-1,1),y\_test.values.reshape(-1,1))

# In[57]:

x\_line = np.arange(80,100).reshape(-1,1)

sns.regplot(x=dataset['Brain'], y=dataset['PIQ'], data=dataset, fit\_reg=False)

plt.plot(x\_line, reg.predict(x\_line))

plt.show()

**multiple logistic regression :**

# coding: utf-8

# # Logistic Regression

# In[26]:

import numpy as np

import pandas as pd

import scipy.stats as stats

import matplotlib.pyplot as plt

import sklearn

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn import preprocessing,cross\_validation,neighbors

from sklearn.metrics import accuracy\_score

# In[27]:

dataset = pd.read\_csv('dataset3.csv')

# In[28]:

print dataset.head()

print dataset.dtypes

# In[53]:

X = dataset.as\_matrix(['Brain','Weight']).astype('float32')

Y = dataset.as\_matrix(['PIQ'])

x\_total = X

y\_total = Y

x\_train, x\_test, y\_train,y\_test= cross\_validation.train\_test\_split(x\_total,y\_total,test\_size=0.1)

# In[54]:

reg = linear\_model.LinearRegression()

reg.fit(x\_train, y\_train)

y\_pred = reg.predict(x\_test)

print y\_pred

print('Score: ', reg.score(x\_test, y\_test))

# In[ ]:

**multiple logistic regression - 3 :**

# coding: utf-8

# In[2]:

import numpy as np

import pandas as pd

import scipy.stats as stats

import matplotlib.pyplot as plt

import sklearn

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn import preprocessing,cross\_validation,neighbors

from sklearn.metrics import accuracy\_score

# In[3]:

dataset = pd.read\_csv('dataset3.csv')

# In[4]:

print dataset.head()

print dataset.dtypes

# In[5]:

brain = dataset['Brain']

weight = dataset['Weight']

height = dataset['Height']

piq = dataset['PIQ']

x\_total = []

# print brain

#print piq

# y\_total = [x for x in piq]

# for i in brain:

# for j in weight:

# x = x.append(i,j)

# break

# x\_total.append(x)

brainx= [y for y in brain]

weightx = [x for x in weight]

heightx = [z for z in height]

a=zip(brainx, weightx, heightx)

for i in range(len(a)):

a[i]=list(a[i])

y\_total = [x for x in piq]

x\_total = a

print x\_total

# In[7]:

x\_train, x\_test, y\_train,y\_test= cross\_validation.train\_test\_split(x\_total,y\_total,test\_size=0.2)

# In[8]:

reg = linear\_model.LinearRegression()

# In[11]:

reg.fit(x\_train, y\_train)

y\_pred = reg.predict(x\_test)

print y\_pred

print('Coefficients: \n', reg.coef\_)

# The mean squared error

# In[12]:

train\_color = "b"

test\_color = "r"

score = reg.score(x\_train,y\_train)

print "Here is the score on training data",score

# In[ ]:

**Perceptron :**

# coding: utf-8

# In[20]:

from sklearn import datasets

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

iris = datasets.load\_iris()

# Assign petal length and petal width to X matrix (150 samples)

X = iris.data[:, [2, 3]]

# Class labels

y = iris.target

# In[21]:

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

# Optimization - Feature scaling

from sklearn.preprocessing import StandardScaler

# Initlalize a new StandardScaler object, sc

sc = StandardScaler()

# Using the fit method, estimate the sample mean and standard deviation for each feature demension.

sc.fit(X\_train)

# Transform both training and test sets using the sample mean and standard deviations

X\_train\_std = sc.transform(X\_train)

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

# In[22]:

# Import the Perceptron implementation

from sklearn.linear\_model import Perceptron

# Initialize a new perceptron object, ppn.

ppn = Perceptron(n\_iter=40, eta0=0.1, random\_state=0)

# Train the model

ppn.fit(X\_train\_std, y\_train)

y\_pred = ppn.predict(X\_test\_std)

print('Misclassified samples: %d /' % (y\_test != y\_pred).sum(), y\_test.size)

# Different performance metrics

from sklearn.metrics import accuracy\_score

# Calculate the classification accuracy of the perceptron on the test

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred))

# In[19]:

from sklearn.neural\_network import MLPRegressor

reg = MLPRegressor()

mlp = MLPRegressor(solver='lbfgs', hidden\_layer\_sizes=50,max\_iter=150, shuffle=True, random\_state=1,activation='relu')

mlp.fit(X\_train\_std, y\_train)

y\_pred2 = mlp.predict(X\_test\_std)

print y\_pred2

from sklearn.metrics import accuracy\_score

# Calculate the classification accuracy of the perceptron on the test

print('Accuracy: %.2f' % mlp.score(X\_test\_std, y\_test))

# In[ ]:

**SVM and SVC :**

# coding: utf-8

# In[3]:

import numpy as np

import pandas as pd

import seaborn as sn

import matplotlib.pyplot as plt

from sklearn.svm import SVC

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

from sklearn.metrics import accuracy\_score

# # Classification

# In[6]:

import numpy as np

import urllib

# url with dataset

# url = "http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"

# # download the file

# raw\_data = urllib.urlopen(url)

# load the CSV file as a numpy matrix

dataset = pd.read\_csv('mtcars.csv')

# separate the data from the target attributes

# X = dataset[:,0:8]

# y = dataset[:,8]

# print X

# print y

print dataset.head()

# In[10]:

y = dataset['am']

X = dataset.drop(['am','model'],axis=1)

print "This is the dataset size - training examples",y.shape

# In[54]:

print "No of testing examples",y.shape

# In[11]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 0)

# # Defining classifier using SVC from SVM classifier

# In[12]:

clf = SVC(kernel = 'linear', C=1000)

# # Training

# In[13]:

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

# In[14]:

pred = clf.predict(X\_test)

# # Output :

# ### Prediction class

# In[15]:

print pred

# # Accuracy :

# In[16]:

accuracy = accuracy\_score(pred, y\_test)

print("Here is the new accuracy",accuracy)

# In[17]:

get\_ipython().magic(u'matplotlib inline')

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');

# # Regression :

# In[76]:

from sklearn.svm import SVR

dataset = pd.read\_csv('mtcars.csv')

print dataset.head

# In[77]:

X = dataset.as\_matrix(['mpg','drat']).astype('float32')

y = dataset.as\_matrix(['wt'])

# # Graph

# In[81]:

get\_ipython().magic(u'matplotlib inline')

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');

# # Training

# In[82]:

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

# # SVM using SVR regressor

# In[83]:

clfr = SVR(C=1.0, epsilon=0.2)

# In[84]:

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

# # Accuracy :

# As seen from the graph above, because of dataset we are getting less accuracy

# In[86]:

score = clfr.score(X\_test,y\_test)

print "Score is ",score