IMPLEMENTATION

**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** sklearn  
**import** seaborn **as** sns  
*#to print multiple statements***from** IPython.core.interactiveshell **import** InteractiveShell  
InteractiveShell.ast\_node\_interactivity = **"all"***#to ignore warning***import** warnings  
warnings.filterwarnings(**"ignore"**)  
print(**"Student admission prediction"**)  
*# %matplotlib inline*plt.show()  
*#dataset saved it in local as 'data.csv'  
# Define the headers since the data does not have any  
# headers = ["Admit","GRE","GPA","Rank"]  
# Read in the CSV file*df = pd.read\_csv(**"data.csv"**)  
*#First step is to visualize our dataset  
#Check first 5 rows*print (**"Head values:"**)  
print(df.head())  
*#Check types of data*print (**"Types:"**)  
print(df.dtypes)  
print(type(df))  
*#Check shape of dataset*print(**"Dataset Shape values:"**)  
print(df.shape)  
*#Check some information*df.info()  
*#Check if there are any null values in data*print (**"Any Null values:"**)  
print(df.isnull().values.any())  
*##Second step is to check for some statistics  
  
#Sum of each column value*print (**"Sum of each column:"**)  
print(df.sum())  
*##Here sum of admit and rank column is not meaningful. Even the sum of GRE and GPA column seems meaningless.  
##But from these value, We can say that we have to normalize the each feature for further calculation.  
  
#Some more statistics*print (**"Mean values: (Sum/Total examples)"**)  
print(df.mean())  
  
print(df.describe())  
*#Third step is to visualize these data in graphical form  
#I have used Seaborn library for data visualization*sns.set(color\_codes=**True**)  
*#Set multiple axis*f, axes = plt.subplots(1, 2, figsize=(20, 10))  
*#GRE data*sns.distplot(df.ix[:,1],color=**"r"**,ax = axes[0]) *#All Row data and first column  
#GPA data*sns.distplot(df.ix[:,2],color=**"b"**,ax = axes[1]) *#All Row data and second column  
#We can see that the data forms good bell shaped curve*sns.factorplot(x=**"rank"**, y=**"gpa"**, hue=**"admit"**, data=df,  
 size=6, kind=**"bar"**, palette=**"muted"**)  
sns.factorplot(x=**"rank"**, y=**"gre"**, hue=**"admit"**, data=df,  
 size=6, kind=**"bar"**, palette=**"muted"**)  
*#Draw graphs for gre --> gpa for rank1,2,3,4*g = sns.FacetGrid(df,col=**'rank'**,hue=**"admit"**,size=6)  
g.map(plt.scatter, **"gpa"**, **"gre"**, alpha=.7)  
g.add\_legend();  
  
*#From below graphs, we can see that for each rank, data are scattered randomly and we can not draw a line to seperate admit/rejected*sns.pairplot(df, hue=**"admit"**,palette=**"muted"**,x\_vars=[**'gre'**,**'gpa'**,**'rank'**],y\_vars=[**'gre'**,**'gpa'**,**'rank'**],size=6)  
*#From following graphs we can the data are scattered randomly and we can not draw a simple boundary line for accepted/rejected  
#Forth part is to train our own model using sigmoid function and find accuracy of it.  
#First of all, we have to convert 'rank' column into four seperate columns according to its value.  
#I have used get\_dummies method for it.*df\_with\_dummies = pd.get\_dummies(df[**'rank'**])  
  
*#This will create four columns and fill values according to original column value. SO if in original 'rank' column,  
#if the value is 3 for some example then in new columns, column 3 will have value 1 and all other (1,2,4) will have  
#value 0*print(df\_with\_dummies.head())  
df\_new = pd.concat([df, df\_with\_dummies], axis=1)  
df\_new = df\_new.drop(**'rank'**,axis=1)  
print(df\_new.head())  
*#Now we need to normalize GRE and GPA columns value as they have large values.  
#I am using Sigmoid function. This function gives zero slope for very large and small values.  
#So gradient descent will be zero if we use large values.  
#So, I have normalize both columns with mean value as 0 and standard deviation as 1*gre\_mean,gre\_std = df\_new[**'gre'**].mean(),df\_new[**'gre'**].std()  
gpa\_mean,gpa\_std = df\_new[**'gpa'**].mean(),df\_new[**'gpa'**].std()  
  
df\_new.loc[:,**'gre'**] = (df\_new[**'gre'**] - gre\_mean) / gre\_std  
df\_new.loc[:,**'gpa'**] = (df\_new[**'gpa'**] - gpa\_mean) / gpa\_std  
  
print(df\_new.head())  
msk = np.random.rand(len(df\_new)) < 0.9  
  
train = df\_new[msk]  
  
test = df\_new[~msk]  
print(len(train))  
print(len(test))  
  
print(train.head())  
*# Split into features and targets*features, targets = train.drop(**'admit'**, axis=1), train[**'admit'**]  
features\_test, targets\_test = test.drop(**'admit'**, axis=1), test[**'admit'**]  
  
  
*# Sigmoid function definition***def** sigmoid(x):  
 **return** 1 / (1 + np.exp(-x))  
  
**def** sigmoid\_prime(x):  
 **return** sigmoid(x) \* (1 - sigmoid(x))  
  
*# n\_records will have no. of rows and n\_features will have no. of column value*n\_records, n\_features = features.shape  
last\_loss = **None***# Initialize weights*weights = np.random.normal(scale=1 / n\_features \*\* .5, size=n\_features)  
print(weights)  
*# Neural Network hyperparameters*epochs = 1000  
learnrate = 0.5  
  
**for** e **in** range(epochs):  
 del\_w = np.zeros(weights.shape)  
  
 **for** x, y **in** zip(features.values, targets):  
 *# Loop through all records, x is the input, y is the target  
  
 # del\_w, W and X are 1 x 6.  
 # Following np.dot will do element wise element multiplication.* output = sigmoid(np.dot(x, weights))  
  
 error = (y - output) \* sigmoid\_prime(np.dot(x, weights))  
  
 del\_w += error \* x  
  
 weights += (learnrate \* del\_w) / n\_records  
  
 *# Printing out the mean square error on the training set* **if** e % (epochs / 10) == 0:  
 out = sigmoid(np.dot(features, weights))  
 loss = np.mean((out - targets) \*\* 2)  
 **if** last\_loss **and** last\_loss < loss:  
 print(**"Train loss: "**, loss, **" WARNING - Loss Increasing"**)  
 **else**:  
 print(**"Train loss: "**, loss)  
 last\_loss = loss  
  
*# Calculate accuracy on test data*tes\_out = sigmoid(np.dot(features\_test, weights))  
predictions = tes\_out > 0.5  
accuracy = np.mean(predictions == targets\_test)  
print(**"Prediction accuracy: {:.3f}"**.format(accuracy))  
  
  
*#Sixt parth is about understanding of Machine learning models from sklearn module.  
#Above we have already created train and test data.***from** sklearn.metrics **import** classification\_report  
**from** sklearn.metrics **import** confusion\_matrix  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.model\_selection **import** KFold  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
**from** sklearn.naive\_bayes **import** GaussianNB  
**from** sklearn.svm **import** SVC  
  
  
*#Model Preparation*num\_trees = 200  
max\_features = 3  
models=[]  
models.append((**'Decision Tree Classifier'**,DecisionTreeClassifier()))  
print(models)  
print(**'--------------'**)  
msk1 = np.random.rand(len(df)) < 0.9  
train\_on\_original = df[msk1]  
test\_on\_original = df[~msk1]  
print (len(train\_on\_original))  
print (len(test\_on\_original))  
print (train\_on\_original.head())  
  
*# Split into features and targets*features\_train\_original, targets\_train\_original = train\_on\_original.drop(**'admit'**, axis=1), train\_on\_original[**'admit'**]  
features\_test\_original, targets\_test\_original = test\_on\_original.drop(**'admit'**, axis=1), test\_on\_original[**'admit'**]  
  
*#Fit model***from** sklearn.model\_selection **import** KFold  
results = []  
names = []  
scoring = **'accuracy'  
  
for** name,model **in** models:  
 kfold = KFold(n\_splits=10,random\_state=7)  
 cv\_results = cross\_val\_score(model,features\_train\_original,targets\_train\_original,cv=kfold,scoring=scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = **"%s: %f (%f)"** % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)  
  
print (results)  
*#Here LDA is achieving ~71% of training accuracy  
#sns.boxplot(results)*ax = sns.boxplot(data=results, palette=**"Set3"**)  
ax.set\_xticklabels(names)  
*#Now the prediction of a new data on any model. I have worked with LDA.*model = DecisionTreeClassifier()  
model.fit(features\_train\_original,targets\_train\_original)  
predictions = model.predict(features\_test\_original)  
print(predictions)  
print(**"Decision tree Model"**)  
print(accuracy\_score(targets\_test\_original, predictions)\*100)  
print(classification\_report(targets\_test\_original, predictions))  
*#LDA is trained on the training data also we have made predictions on test data.  
#So the overall accuracy is 73.33% with LDA.*cm = confusion\_matrix(targets\_test\_original,predictions)  
sns.heatmap(cm,annot=**True**,xticklabels=[**'reject'**, **'admit'**],yticklabels=[**'reject'**, **'admit'**])  
*#Prediction on new data  
  
#data\_x = [gre,gpa,rank]  
#model.predict method expects 2D array. First I tried to define data\_1 as [750,4,1] which was 1D array so  
#it was throwing an error. So I converted data\_1 from (3,) to (1,3).*labels = [**"Rejected"**,**"Accepted"**]  
data\_1 = np.array([750,4,1],ndmin=2)  
data\_2 = np.array([700,2,2],ndmin=2)  
data\_3 = np.array([200,2,4],ndmin=2)  
  
prediction\_1 = model.predict(data\_1)  
prediction\_2 = model.predict(data\_2)  
prediction\_3 = model.predict(data\_3)  
  
print (**"Status for data\_1 is: "**, labels[int(prediction\_1)])  
print (**"Status for data\_2 is: "**, labels[int(prediction\_2)])  
print (**"Status for data\_3 is: "**, labels[int(prediction\_3)])

Output

Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32

runfile('C:/Users/Priya/PycharmProjects/Admission/student.py', wdir='C:/Users/Priya/PycharmProjects/Admission')

Student admission prediction

Head values:

admit gre gpa rank

0 0 380 3.61 3

1 1 660 3.67 3

2 1 800 4.00 1

3 1 640 3.19 4

4 0 520 2.93 4

Types:

admit int64

gre int64

gpa float64

rank int64

dtype: object

<class 'pandas.core.frame.DataFrame'>

Dataset Shape values:

(400, 4)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 400 entries, 0 to 399

Data columns (total 4 columns):

admit 400 non-null int64

gre 400 non-null int64

gpa 400 non-null float64

rank 400 non-null int64

dtypes: float64(1), int64(3)

memory usage: 12.6 KB

Any Null values:

False

Sum of each column:

admit 127.00

gre 235080.00

gpa 1355.96

rank 994.00

dtype: float64

Mean values: (Sum/Total examples)

admit 0.3175

gre 587.7000

gpa 3.3899

rank 2.4850

dtype: float64

admit gre gpa rank

count 400.000000 400.000000 400.000000 400.00000

mean 0.317500 587.700000 3.389900 2.48500

std 0.466087 115.516536 0.380567 0.94446

min 0.000000 220.000000 2.260000 1.00000

25% 0.000000 520.000000 3.130000 2.00000

50% 0.000000 580.000000 3.395000 2.00000

75% 1.000000 660.000000 3.670000 3.00000

max 1.000000 800.000000 4.000000 4.00000

1 2 3 4

0 0 0 1 0

1 0 0 1 0

2 1 0 0 0

3 0 0 0 1

4 0 0 0 1

admit gre gpa 1 2 3 4

0 0 380 3.61 0 0 1 0

1 1 660 3.67 0 0 1 0

2 1 800 4.00 1 0 0 0

3 1 640 3.19 0 0 0 1

4 0 520 2.93 0 0 0 1

admit gre gpa 1 2 3 4

0 0 -1.798011 0.578348 0 0 1 0

1 1 0.625884 0.736008 0 0 1 0

2 1 1.837832 1.603135 1 0 0 0

3 1 0.452749 -0.525269 0 0 0 1

4 0 -0.586063 -1.208461 0 0 0 1

355

45

admit gre gpa 1 2 3 4

0 0 -1.798011 0.578348 0 0 1 0

3 1 0.452749 -0.525269 0 0 0 1

4 0 -0.586063 -1.208461 0 0 0 1

5 1 1.491561 -1.024525 0 1 0 0

6 1 -0.239793 -1.077078 1 0 0 0

[ 0.20993373 -0.10351051 0.5387559 -0.78619029 0.06384284 -0.02969082]

Train loss: 0.2410885956585201

Train loss: 0.2064429826552695

Train loss: 0.19957587823830028

Train loss: 0.19738938703056033

Train loss: 0.19653236683956055

Train loss: 0.19615038099623258

Train loss: 0.19596475840513297

Train loss: 0.1958687084843852

Train loss: 0.19581655805367604

Train loss: 0.19578715051762505

Prediction accuracy: 0.689

[('Decision Tree Classifier', DecisionTreeClassifier(class\_weight=None, criterion='gini', 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=None,

splitter='best'))]

--------------

364

36

admit gre gpa rank

0 0 380 3.61 3

2 1 800 4.00 1

3 1 640 3.19 4

5 1 760 3.00 2

6 1 560 2.98 1

Decision Tree Classifier: 0.612312 (0.081764)

[array([0.64864865, 0.62162162, 0.72972973, 0.56756757, 0.52777778,

0.72222222, 0.47222222, 0.63888889, 0.66666667, 0.52777778])]

[1 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0]

Decision tree Model

69.44444444444444

precision recall f1-score support

0 0.86 0.69 0.77 26

1 0.47 0.70 0.56 10

micro avg 0.69 0.69 0.69 36

macro avg 0.66 0.70 0.66 36

weighted avg 0.75 0.69 0.71 36

Status for data\_1 is: Accepted

Status for data\_2 is: Accepted

Status for data\_3 is: Rejected









