PREDICTING PERSONAL LOAN APPROVAL

# # This project about buiding machine learning models for a data set of customers applying for loan in a bank. The aim is to predict if the bank should approve the loan for a partcular customer or not.

# # Preparing train data

# In[1]:

# importing packages that we will need through out this project

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# In[2]:

# loading test data into pandas data frame and droping some unnecessary coloumns

loan=pd.read\_excel("Project - 4 - Train Data.xlsx")

loan=loan.drop(['Loanapp\_ID','first\_name','last\_name','email','address','INT\_ID','Prev\_ID','AGT\_ID'],axis=1)

loan.head()

# In[3]:

# Lets see the info. of the data frame

loan.info()

# In[4]:

# chacking if there is any missing values

loan.isnull().values.sum()

# In[5]:

loan['Sex'].value\_counts()

# In[6]:

loan['Marital\_Status'].value\_counts()

# In[7]:

loan['Dependents'].value\_counts()

# In[8]:

loan['Qual\_var'].value\_counts()

# In[9]:

loan['SE'].value\_counts()

# In[10]:

loan['Prop\_Area'].value\_counts()

# In[11]:

loan['CPL\_Status'].value\_counts()

# In[12]:

loan.isnull().sum()

# Here we can see that there are still some missing values in the data set

# In[13]:

loan.isnull().sum().sum()

# In[14]:

# filling the missing values by mean values for continuous data

loan.CPL\_Amount.fillna(loan['CPL\_Amount'].mean(),inplace=True)

loan.CPL\_Term.fillna(loan['CPL\_Term'].mean(),inplace=True)

loan.Credit\_His.fillna(loan['Credit\_His'].mean(),inplace=True)

loan.isnull().sum()

# In[15]:

# check the info of data set

loan.info()

# In[16]:

# mapping catagorical values to numerical values

loan['Sex']=loan.Sex.map({'M':1,'F':0})

loan['Marital\_Status']=loan.Marital\_Status.map({'Y':1,'N':0})

loan['Dependents']=loan.Dependents.map({0:0,1:1,2:2,'3+':3})

loan['SE']=loan.SE.map({'Y':1,'N':0})

loan['Qual\_var']=loan.Qual\_var.map({'Grad':1,'Non Grad':0})

loan['CPL\_Status']=loan.CPL\_Status.map({'Y':1,'N':0})

prop\_area=pd.get\_dummies(loan['Prop\_Area'],prefix='Prop\_Area',drop\_first=True)

loan=pd.concat([loan,prop\_area],axis=1)

loan=loan.drop('Prop\_Area',axis=1)

# In[17]:

loan.head(20)

# In[18]:

# filling the missing values for catagorical values by mode

mode=loan.mode(axis=0)

print(mode)

loan['Sex'].fillna(mode.iloc[0,0],inplace=True)

loan['Marital\_Status'].fillna(mode.iloc[0,1],inplace=True)

loan['Dependents'].fillna(mode.iloc[0,2],inplace=True)

loan['SE'].fillna(mode.iloc[0,4],inplace=True)

# In[19]:

loan.isnull().sum()

# In[20]:

# plot the heat map to check if there is any high correlation among attributes

plt.figure(figsize=(20,10))

sns.heatmap(loan.corr(),annot=True)

# In[21]:

# dividing the train data into X and Y

X\_train=loan.drop('CPL\_Status',axis=1)

Y\_train=loan['CPL\_Status']

# # Test data preparation

# In[22]:

# load test data set and follow all the above steps to process this data

loan1=pd.read\_excel("Project - 4 - Test Data.xlsx")

loan1=loan1.drop(['Loanapp\_ID','first\_name','last\_name','email','address','INT\_ID','Prev\_ID','AGT\_ID'],axis=1)

loan1.head()

# In[23]:

loan1.info()

# In[24]:

loan1.isnull().values.sum()

# In[25]:

loan1['Sex'].value\_counts()

# In[26]:

loan1['Marital\_Status'].value\_counts()

# In[27]:

loan1['Dependents'].value\_counts()

# In[28]:

loan1['Qual\_var'].value\_counts()

# In[29]:

loan1['SE'].value\_counts()

# In[30]:

loan1['Prop\_Area'].value\_counts()

# In[31]:

loan1.isnull().sum()

# In[32]:

loan1.isnull().sum().sum()

# In[33]:

loan1.CPL\_Term.fillna(loan1['CPL\_Term'].mean(),inplace=True)

loan1.Credit\_His.fillna(loan1['Credit\_His'].mean(),inplace=True)

loan1.isnull().sum()

# In[34]:

loan1['Sex']=loan1.Sex.map({'M':1,'F':0})

loan1['Marital\_Status']=loan1.Marital\_Status.map({'Y':1,'N':0})

loan1['Dependents']=loan1.Dependents.map({0:0,1:1,2:2,'3+':3})

loan1['SE']=loan1.SE.map({'Y':1,'N':0})

loan1['Qual\_var']=loan1.Qual\_var.map({'Grad':1,'Non Grad':0})

prop\_area1=pd.get\_dummies(loan1['Prop\_Area'],prefix='Prop\_Area',drop\_first=True)

loan1=pd.concat([loan1,prop\_area1],axis=1)

loan1=loan1.drop('Prop\_Area',axis=1)

# In[35]:

loan1.head()

# In[36]:

mode1=loan1.mode(axis=0)

print(mode1)

loan1['Sex'].fillna(mode1.iloc[0,0],inplace=True)

loan1['Dependents'].fillna(mode1.iloc[0,2],inplace=True)

loan1['SE'].fillna(mode1.iloc[0,4],inplace=True)

# In[37]:

loan.isnull().sum().sum()

# In[38]:

X\_test=loan1

# # Model Building

# Lets now build models using different ml algorithms and choose the best model among them

# # Logistic Regression

# In[39]:

# import all the required libraries

from sklearn.model\_selection import KFold

from sklearn.linear\_model import LogisticRegression

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

from sklearn.svm import SVC

from sklearn.preprocessing import scale

# In[40]:

# scale train and test data using scale funtion

X\_train=scale(X\_train)

X\_test=scale(X\_test)

# In[41]:

# declare logisticregression object

LR=LogisticRegression(max\_iter=200)

# declare a kfold object for k fold cross validation

kf=KFold(n\_splits=5,shuffle=True,random\_state=10)

# apply cross validation on the train set to get the accuracy

accuracy=cross\_val\_score(LR,X\_train,Y\_train,cv=kf,scoring='accuracy')

print('Accuracy=',accuracy)

print('Average accuracy =',accuracy.mean())

# In[42]:

# store the accuracy scores in a data frame

Score\_Table=pd.DataFrame(columns=['Model name','Accuracy(%)'])

Score\_Table.loc[0]=['Logistic Reg',accuracy.mean()\*100]

# # Support Vector Classifier

# In[43]:

# declare a svc object

svc=SVC(C=1)

# calculate accuracy score using cross validation

svc\_accuracy=cross\_val\_score(svc,X\_train,Y\_train,cv=kf,scoring='accuracy')

# print accuracy scores

print('Accuracy=',svc\_accuracy)

print('Average accuracy with C=1 is=',svc\_accuracy.mean())

# In[44]:

from sklearn.model\_selection import GridSearchCV

# now lets find the best parameter by grid search method

params={"C":[0.1,1,10,100,1000]}

Gridsearch=GridSearchCV(estimator=svc,param\_grid=params,scoring='accuracy',cv=kf,verbose=1,return\_train\_score=True)

# In[45]:

# fit the data set

Gridsearch.fit(X\_train,Y\_train)

# store the results to results variable

results=pd.DataFrame(Gridsearch.cv\_results\_)

results

# In[46]:

# plot the accuracy vs C by using matplot library

plt.figure(figsize=(6,6))

plt.plot(results['param\_C'],results['mean\_test\_score'])

plt.plot(results['param\_C'],results['mean\_train\_score'])

plt.xlabel('C')

plt.ylabel('score')

plt.legend(['test score','train score'], loc='upper right')

plt.xscale('log')

plt.show()

# In[47]:

# now lets tune the parameter more and search for best parameter

params={"C":[0.5,0.6,0.7,0.8,0.9,1,1.1,1.2,1.3,1.4,1.5,1.6,1.7,2]}

Gridsearch=GridSearchCV(estimator=svc,param\_grid=params,scoring='accuracy',cv=kf,verbose=1,return\_train\_score=True)

Gridsearch.fit(X\_train,Y\_train)

results=pd.DataFrame(Gridsearch.cv\_results\_)

results

# In[48]:

plt.figure(figsize=(6,6))

plt.plot(results['param\_C'],results['mean\_test\_score'])

plt.plot(results['param\_C'],results['mean\_train\_score'])

plt.xlabel('C')

plt.ylabel('score')

plt.legend(['test score','train score'], loc='upper right')

plt.show()

# In[49]:

# see which parameter value gives best result

results[results.mean\_test\_score==results.mean\_test\_score.max()]

# In[50]:

print('best score for SVC=',Gridsearch.best\_score\_)

print('best parameter for SVC=',Gridsearch.best\_params\_)

# In[51]:

# store and print the best parameter

best\_c=Gridsearch.best\_params\_

model=SVC(best\_c['C'])

svc\_accuracy1=cross\_val\_score(model,X\_train,Y\_train,cv=kf,scoring='accuracy')

print('Accuracy for SVC=',svc\_accuracy1)

print('Mean Accuracy for SVC=',svc\_accuracy1.mean())

# In[52]:

# store the accuracy score to the dataframe

Score\_Table.loc[1]=['SVC for C=0.5',svc\_accuracy1.mean()\*100]

# # K-Nearest Neighbor

# In[53]:

# now lets build the model using knn

from sklearn import neighbors

# apply grid search cv to find the best value of number of neigbors

N={"n\_neighbors":[1,2,5,10,20,30,40,50,60]}

clf=neighbors.KNeighborsClassifier()

Gridsearch=GridSearchCV(estimator=clf,param\_grid=N,scoring='accuracy',cv=kf,verbose=1,return\_train\_score=True)

# In[54]:

Gridsearch.fit(X\_train,Y\_train)

clf\_results=pd.DataFrame(Gridsearch.cv\_results\_)

clf\_results

# In[55]:

best\_score=Gridsearch.best\_score\_

best\_n=Gridsearch.best\_params\_

print('best test score=',best\_score)

print('best number of neighbours=',best\_n)

# In[56]:

N={"n\_neighbors":[14,15,16,17,18,19,20,21,22,23,24,25]}

clf=neighbors.KNeighborsClassifier()

Gridsearch=GridSearchCV(estimator=clf,param\_grid=N,scoring='accuracy',cv=kf,verbose=1,return\_train\_score=True)

Gridsearch.fit(X\_train,Y\_train)

clf\_results=pd.DataFrame(Gridsearch.cv\_results\_)

best\_score=Gridsearch.best\_score\_

best\_n=Gridsearch.best\_params\_

# In[57]:

print('best accuracy score=',best\_score)

print('best number of neighbours=',best\_n)

# In[58]:

Score\_Table.loc[2]=['KNN for neighbors =22',best\_score\*100]

# # Decision Tree

# lets again prepare the data that to be applied for the decision tree algorithm. Here i have applied label encoder to encode the catagorical variables

# In[59]:

loan=pd.read\_excel("Project - 4 - Train Data.xlsx")

loan=loan.drop(['Loanapp\_ID','first\_name','last\_name','email','address','INT\_ID','Prev\_ID','AGT\_ID'],axis=1)

loan.head()

# In[60]:

# lets fill the missing values for continuos values

loan.CPL\_Amount.fillna(loan['CPL\_Amount'].mean(),inplace=True)

loan.CPL\_Term.fillna(loan['CPL\_Term'].mean(),inplace=True)

loan.Credit\_His.fillna(loan['Credit\_His'].mean(),inplace=True)

loan.isnull().sum()

# In[61]:

# temporarily map the catagorical values to numerical values

loan['Sex']=loan.Sex.map({'M':1,'F':0})

loan['Marital\_Status']=loan.Marital\_Status.map({'Y':1,'N':0})

loan['Dependents']=loan.Dependents.map({0:0,1:1,2:2,'3+':3})

loan['SE']=loan.SE.map({'Y':1,'N':0})

loan['Qual\_var']=loan.Qual\_var.map({'Grad':1,'Non Grad':0})

loan['Prop\_Area']=loan.Prop\_Area.map({'Urban':1,'Semi U':2,'Rural':3})

loan['CPL\_Status']=loan.CPL\_Status.map({'Y':1,'N':0})

# In[62]:

# fill the missing values for catagorical values

mode=loan.mode(axis=0)

print(mode)

loan['Sex'].fillna(mode.iloc[0,0],inplace=True)

loan['Marital\_Status'].fillna(mode.iloc[0,1],inplace=True)

loan['Dependents'].fillna(mode.iloc[0,2],inplace=True)

loan['SE'].fillna(mode.iloc[0,4],inplace=True)

# In[63]:

# again map the catagorical values back to previous values

loan['Sex']=loan.Sex.map({1:'M',0:'F'})

loan['Marital\_Status']=loan.Marital\_Status.map({1:'Y',0:'N'})

loan['Dependents']=loan.Dependents.map({0:'0',1:'1',2:'2',3:'3+'})

loan['SE']=loan.SE.map({1:'Y',0:'N'})

loan['Qual\_var']=loan.Qual\_var.map({1:'Grad',0:'Non Grad'})

loan['Prop\_Area']=loan.Prop\_Area.map({1:'Urban',2:'Semi U',3:'Rural'})

loan['CPL\_Status']=loan.CPL\_Status.map({1:'Y',0:'N'})

# In[64]:

# now lets encode the catagorical values into a separate dataframe

from sklearn import preprocessing

df\_categorical = loan.select\_dtypes(include=['object'])

df\_categorical = df\_categorical.apply(le.fit\_transform)

df\_categorical.head()

# In[65]:

# concatinate the two dataframe into a single data frame

loan = loan.drop(df\_categorical.columns, axis=1)

loan = pd.concat([loan, df\_categorical], axis=1)

loan.head()

# Model building for DT

# In[66]:

# devide the data set in X and Y

X\_train=loan.drop('CPL\_Status',axis=1)

Y\_train=loan['CPL\_Status']

# In[67]:

from sklearn.tree import DecisionTreeClassifier

# tuning maximum depth by grid search method

parameters = {'max\_depth': range(1, 10)}

DT = DecisionTreeClassifier(criterion = "gini",

random\_state = 100)

DT\_clf = GridSearchCV(DT, parameters,

cv=kf,

scoring="accuracy")

DT\_clf.fit(X\_train, Y\_train)

# In[68]:

results=pd.DataFrame(DT\_clf.cv\_results\_)

results.head()

# In[69]:

# print best values and best scores

print('best maximum depth=',DT\_clf.best\_params\_)

print('best accuracy=',DT\_clf.best\_score\_)

# In[70]:

# tuning minimum samples leaf

parameters = {'min\_samples\_leaf': range(1, 100, 2)}

DT = DecisionTreeClassifier(criterion = "gini",

random\_state = 100)

DT\_clf1 = GridSearchCV(DT, parameters,

cv=kf,

scoring="accuracy")

DT\_clf1.fit(X\_train, Y\_train)

print('best min\_samples\_leaf=',DT\_clf1.best\_params\_)

print('best accuracy=',DT\_clf1.best\_score\_)

# In[71]:

# tuning minimum samples split

parameters = {'min\_samples\_split': range(2, 100, 2)}

DT = DecisionTreeClassifier(criterion = "gini",

random\_state = 100)

DT\_clf2 = GridSearchCV(DT, parameters,

cv=kf,

scoring="accuracy")

DT\_clf2.fit(X\_train, Y\_train)

print('best min\_samples\_split=',DT\_clf2.best\_params\_)

print('best accuracy=',DT\_clf2.best\_score\_)

# In[72]:

# now lets tune all the parameters

param = {

'max\_depth': range(1, 10),

'min\_samples\_leaf': range(1, 100, 10),

'min\_samples\_split': range(2, 102, 10),

'criterion': ["entropy", "gini"]

}

DT = DecisionTreeClassifier()

DT\_clf3 = GridSearchCV(estimator=DT, param\_grid=param,

cv=kf,

scoring="accuracy",verbose = 1)

DT\_clf3.fit(X\_train, Y\_train)

# In[73]:

# print the best scores and best values

print('best parameters=',DT\_clf3.best\_params\_)

print('best accuracy=',DT\_clf3.best\_score\_)

# In[74]:

# store the accuracy into the data frame

Score\_Table.loc[3]=['DT (max depth=1, min\_samples\_leaf=1, min\_samples\_split=2)',(DT\_clf3.best\_score\_)\*100]

# In[75]:

# now lets annalyze the models

Score\_Table

# So the best model for this data is logistic regression with gives an accuracy of 81%

#

# Now lets build the final model using logistic regression and predict for test data

# In[76]:

# Do all the data processing steps that were done before

loan=pd.read\_excel("Project - 4 - Train Data.xlsx")

loan=loan.drop(['Loanapp\_ID','first\_name','last\_name','email','address','INT\_ID','Prev\_ID','AGT\_ID'],axis=1)

loan.CPL\_Amount.fillna(loan['CPL\_Amount'].mean(),inplace=True)

loan.CPL\_Term.fillna(loan['CPL\_Term'].mean(),inplace=True)

loan.Credit\_His.fillna(loan['Credit\_His'].mean(),inplace=True)

loan.isnull().sum()

loan['Sex']=loan.Sex.map({'M':1,'F':0})

loan['Marital\_Status']=loan.Marital\_Status.map({'Y':1,'N':0})

loan['Dependents']=loan.Dependents.map({0:0,1:1,2:2,'3+':3})

loan['SE']=loan.SE.map({'Y':1,'N':0})

loan['Qual\_var']=loan.Qual\_var.map({'Grad':1,'Non Grad':0})

loan['CPL\_Status']=loan.CPL\_Status.map({'Y':1,'N':0})

prop\_area=pd.get\_dummies(loan['Prop\_Area'],prefix='Prop\_Area',drop\_first=True)

loan=pd.concat([loan,prop\_area],axis=1)

loan=loan.drop('Prop\_Area',axis=1)

mode=loan.mode(axis=0)

loan['Sex'].fillna(mode.iloc[0,0],inplace=True)

loan['Marital\_Status'].fillna(mode.iloc[0,1],inplace=True)

loan['Dependents'].fillna(mode.iloc[0,2],inplace=True)

loan['SE'].fillna(mode.iloc[0,4],inplace=True)

# In[77]:

# divide the data into X and Y

X\_train=loan.drop('CPL\_Status',axis=1)

Y\_train=loan['CPL\_Status']

# In[78]:

# scale data by using scale function

X\_train=scale(X\_train)

X\_test=scale(X\_test)

# In[79]:

# declare the logistic regression object

LR=LogisticRegression(max\_iter=200)

# fit the train data

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

# In[80]:

# predict the CPL amount for the test data

Y\_pred=LR.predict(X\_test)

# In[81]:

# Convert the predicted data into dataframe

Y\_pred=pd.DataFrame(Y\_pred)

# In[82]:

Y=pd.DataFrame(columns=['Predicted CPL Status'])

# map 1 to Y and 0 to N

Y=Y\_pred.loc[:,0].map(lambda x:'Y' if x==1 else 'N')

# In[83]:

# export the predicted data into an excel sheet

Y.to\_excel("Predicted CPL Status.xlsx")

le = preprocessing.LabelEncoder()