**COURSE#9 WEEK6 – FINAL PROJECT**

**import** seaborn **as** sns

bins **=** np**.**linspace(df**.**Principal**.**min(), df**.**Principal**.**max(), 10)

g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)

g**.**map(plt**.**hist, 'Principal', bins**=**bins, ec**=**"k")

g**.**axes[**-**1]**.**legend()

plt**.**show()

bins **=** np**.**linspace(df**.**age**.**min(), df**.**age**.**max(), 10)

g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)

g**.**map(plt**.**hist, 'age', bins**=**bins, ec**=**"k")

g**.**axes[**-**1]**.**legend()

plt**.**show()

Day of the Week for Loan Approval

df['dayofweek'] **=** df['effective\_date']**.**dt**.**dayofweek

bins **=** np**.**linspace(df**.**dayofweek**.**min(), df**.**dayofweek**.**max(), 10)

g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)

g**.**map(plt**.**hist, 'dayofweek', bins**=**bins, ec**=**"k")

g**.**axes[**-**1]**.**legend()

plt**.**show()

df['weekend'] **=** df['dayofweek']**.**apply(**lambda** x: 1 **if** (x**>**3) **else** 0)

df**.**head()

Converting Categorical Variable – Gender

df**.**groupby(['Gender'])['loan\_status']**.**value\_counts(normalize**=True**)

df['Gender']**.**replace(to\_replace**=**['male','female'], value**=**[0,1],inplace**=True**)

df**.**head()

One Hot Encoding (Education)

df**.**groupby(['education'])['loan\_status']**.**value\_counts(normalize**=True**)

Feature Prior to One Hot Encoding

df[['Principal','terms','age','Gender','education']]**.**head()

Use of One Hot Encoding to Convert Categorical Variables to Binary Variables

Feature **=** df[['Principal','terms','age','Gender','weekend']]

Feature **=** pd**.**concat([Feature,pd**.**get\_dummies(df['education'])], axis**=**1)

Feature**.**drop(['Master or Above'], axis **=** 1,inplace**=True**)

Feature**.**head()

Feature Selection

X **=** Feature

X[0:5]

Ascertaining Labels

y **=** df['loan\_status']**.**values

Normalization of Data

X**=** preprocessing**.**StandardScaler()**.**fit(X)**.**transform(X)

X[0:5]

Classification

K Nearest Neighbour (KNN)

**from** sklearn.neighbors **import** KNeighborsClassifier

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

**from** sklearn **import** metrics

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

print ('Train set:', X\_train**.**shape, y\_train**.**shape)

print ('Test set:', X\_test**.**shape, y\_test**.**shape)

Train set: (276, 8) (276,)

Test set: (70, 8) (70,)

In [22]:

k **=** 7

check **=** KNeighborsClassifier(n\_neighbors **=** k)**.**fit(X\_train,y\_train)

check

yhat **=** check**.**predict(X\_test)

yhat

print("Train set Accuracy: ", metrics**.**accuracy\_score(y\_train, check**.**predict(X\_train)))

print("Test set Accuracy: ", metrics**.**accuracy\_score(y\_test, yhat))

Decision Tree

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn **import** preprocessing

DTree **=** DecisionTreeClassifier(criterion**=**"entropy", max\_depth **=** 5)

DTree

DTree**.**fit(X\_train,y\_train)

predTree **=** DTree**.**predict(X\_test)

DTree**.**fit(X,y)

Out[25]:

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=5,

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')

In [27]:

print("Train set Accuracy: ", metrics**.**accuracy\_score(y\_train, DTree**.**predict(X\_train)))

print("Test set Accuracy: ", metrics**.**accuracy\_score(y\_test, yhat))

Support Vector Machine

**from** sklearn **import** svm

clf **=** svm**.**SVC(kernel**=**'rbf')

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

yhat **=** clf**.**predict(X\_test)

print("Train set Accuracy: ", metrics**.**accuracy\_score(y\_train, clf**.**predict(X\_train)))

print("Test set Accuracy: ", metrics**.**accuracy\_score(y\_test, yhat))

Logistic Regression

**from** sklearn.linear\_model **import** LogisticRegression

LR **=** LogisticRegression(C**=**0.01, solver**=**'liblinear')**.**fit(X\_train,y\_train)

yhat **=** LR**.**predict(X\_test)

yhat\_prob **=** LR**.**predict\_proba(X\_test)

print("Train set Accuracy: ", metrics**.**accuracy\_score(y\_train, LR**.**predict(X\_train)))

print("Test set Accuracy: ", metrics**.**accuracy\_score(y\_test, yhat))

Model Evaluation Using Test Set

**from** sklearn.metrics **import** jaccard\_similarity\_score

**from** sklearn.metrics **import** f1\_score

**from** sklearn.metrics **import** log\_loss

Downloading and Loading Test Set

**!**wget -O loan\_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan\_test.csv

Loading Test Set for Evaluation

test **=** pd**.**read\_csv('loan\_test.csv')

test**.**head()

test['due\_date'] **=** pd**.**to\_datetime(test['due\_date'])

test['effective\_date'] **=** pd**.**to\_datetime(test['effective\_date'])

test['dayofweek'] **=** test['effective\_date']**.**dt**.**dayofweek

test['Weekend'] **=** test['dayofweek']**.**apply(**lambda** x: 1 **if** (x**>**3) **else** 0)

test['Gender']**.**replace(to\_replace**=**['male','female'], value**=**[0,1],inplace**=True**)

test['loan\_status']**.**replace(to\_replace**=**['PAIDOFF','COLLECTION'],value**=**[1,0],inplace**=True**)

x **=** test[['Principal','terms','age','Gender','Weekend']]

x **=** pd**.**concat([x,pd**.**get\_dummies(test['education'])], axis**=**1)

x**.**drop(['Master or Above'], axis **=** 1,inplace**=True**)

x1 **=** preprocessing**.**StandardScaler()**.**fit(x)**.**transform(x)

Y **=** test['loan\_status']**.**values

print("KNN Jaccard index: %.2f" **%** jaccard\_similarity\_score(y, check**.**predict(X)))

print("KNN F1-score: %.2f" **%** f1\_score(y\_test, yhat, average**=**'weighted'))

KNN Jaccard index: 0.80

KNN F1-score: 0.67

print("DTree Jaccard index: %.2f" **%jaccard\_similarity\_score**(y, DTree.predict(X)))

print("DTree F1-score: %.2f" **%** f1\_score(y\_test, yhat, average**=**'weighted'))

print("SVM F1-score: %.2f" **%** f1\_score(y\_test, yhat, average**=**'weighted'))

print("SVM Jaccard index: %.2f" **%** jaccard\_similarity\_score(y, clf**.**predict(X)))

print("LR Jaccard index: %.2f" **%** jaccard\_similarity\_score(y, LR**.**predict(X)))

print("LR LogLoss: %.2f" **%** log\_loss(y\_test, yhat\_prob))

print("LR F1-score: %.2f" **%** f1\_score(y\_test, yhat, average**=**'weighted'))