Explainability in Decision Tree

Dr. Sabu M K
Department of Computer Applications
CUSAT

- # Importing pandas
- import numpy as np, pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns
- import warnings
- warnings.filterwarnings('ignore')

- # Loading dataset
- df = pd.read_csv('/home/sabu/Desktop/WA_Fn-UseC_-Telco-Customer-Churn.csv',header=0)
-
- # Inspecting data
- df.head()
-
- # Inspecting basic information out of columns
- df.info()

- # Displaying summary statistics
- df.describe()
-
-
- # Assigning 0 and 1 to Yes and No
- df['SeniorCitizen'] = df['SeniorCitizen'].map({0:'No',1:'Yes'})

- #Binning the tenure column
- cut_labels = ['0-12', '13-24', '25-36', '37-48','49-60','61-72']
- cut_bins = [0, 12,24,36,48,60,72]
- df['Tenure Period'] = pd.cut(df['tenure'], bins=cut_bins, labels=cut_labels)
- df['Tenure Period'].value_counts()

- #Binning the MonthlyCharges column
- cut_labels = ['0-20', '21-40', '41-60', '61-80','81-100','101-120']
- cut_bins = [0, 20,40,60,80,100,120]
- df['MonthlyCharges_Range'] = pd.cut(df['MonthlyCharges'], bins=cut_bins, labels=cut_labels)
- df['MonthlyCharges_Range'].value_counts()

- df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],errors='coerce')
- df['TotalCharges'].describe()

- #Binning the total charges column
- cut_labels = ['0-1000', '1001-2000','2001-4000','4001-6000','6001-8000','8001-10000']
- cut_bins = [0, 1000,2000,4000,6000,8000,10000]
- df['TotalCharges_Range'] = pd.cut(df['TotalCharges'], bins=cut_bins, labels=cut_labels)
- df['TotalCharges_Range'].value_counts()

- # Dropping colummns that are not required
- cols_to_drop = ['customerID','MonthlyCharges','tenure','TotalCharges']
- df.drop(labels=cols_to_drop,axis=1,inplace=True)
-
- # Sanity checks
- df.head(4)

- # Checking count of null values by the columns
- df.isna().sum()

- # Missing values imputation
- df['TotalCharges_Range'].fillna(df['TotalCharges_Range'].mode()[0], inplace=True)
- df['Tenure Period'].fillna(df['Tenure Period'].mode()[0], inplace=True)

- #Label Encoding
- # Importing LabelEncoder
- from sklearn.preprocessing import LabelEncoder
-
- # Instantiating LabelEncoder
- le=LabelEncoder()
-
- # Iterating over all the values of each column and extract their dtypes
- for col in df.columns.to_numpy():
-     # Comparing if the dtype is object
-     if df[col].dtypes in ('object','category'):
-     # Using LabelEncoder to do the numeric transformation
-         df[col]=le.fit_transform(df[col].astype(str))

- # Sanity Check
- df.head()

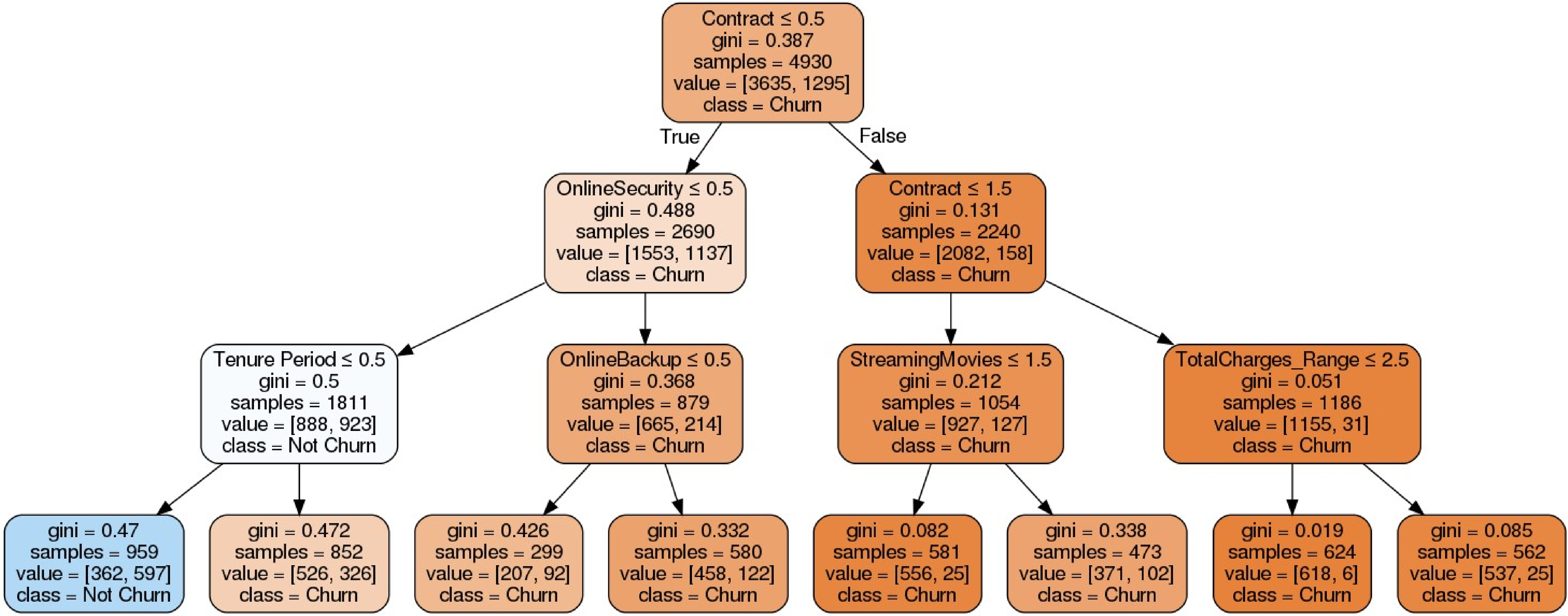
- # Putting feature variable to X
- X = df.drop('Churn',axis=1)
-
- # Putting response variable to y
- y = df['Churn']

- from sklearn.model_selection import train_test_split
- X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42)
- X_train.shape, X_test.shape

- #Model Building
- from sklearn.tree import DecisionTreeClassifier
- dt = DecisionTreeClassifier(max_depth=3,random_state=43)
- dt.fit(X_train, y_train)

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(dt, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = X.columns,class_names=['Churn', "Not Churn"])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Churn.png')
Image(graph.create_png())
```



Contract ≤ 0.5
gini = 0.387
samples = 4930
value = [3635, 1295]
class = Churn
True
False
OnlineSecurity ≤ 0.5
gini = 0.488
samples = 2690
value = [1553, 1137]
class = Churn
Contract ≤ 1.5
gini = 0.131
samples = 2240
value = [2082, 158]
class = Churn
Tenure Period ≤ 0.5
gini = 0.5
samples = 1811
value = [888, 923]
class = Not Churn
OnlineBackup ≤ 0.5
gini = 0.368
samples = 879
value = [665, 214]
class = Churn
StreamingMovies ≤ 1.5
gini = 0.212
samples = 1054
value = [927, 127]
class = Churn
TotalCharges_Range ≤ 2.5
gini = 0.051
samples = 1186
value = [1155, 31]
class = Churn
gini = 0.47
samples = 959
value = [362, 597]
class = Not Churn
gini = 0.472
samples = 852
value = [526, 326]
class = Churn
gini = 0.426
samples = 299
value = [207, 92]
class = Churn
gini = 0.332
samples = 580
value = [458, 122]
class = Churn
gini = 0.082
samples = 581
value = [556, 25]
class = Churn
gini = 0.338
samples = 473
value = [371, 102]
class = Churn
gini = 0.019
samples = 624
value = [618, 6]
class = Churn
gini = 0.085
samples = 562
value = [537, 25]
class = Churn

- #Model Evaluation
- from sklearn.metrics import confusion_matrix, accuracy_score

- #Let's build a tree to it's full depth
- dt = DecisionTreeClassifier(random_state=43)
- dt.fit(X_train,y_train)

- y_train_pred = dt.predict(X_train)
- y_test_pred = dt.predict(X_test)

- print(accuracy_score(y_train, y_train_pred))
- confusion_matrix(y_train, y_train_pred)

- print(accuracy_score(y_test, y_test_pred))
- confusion_matrix(y_test, y_test_pred)

- # Let's check the overall accuracy.
- trainaccuracy= accuracy_score(y_train, y_train_pred)
- testaccuracy= accuracy_score(y_test, y_test_pred)
-
- confusion_TRN = confusion_matrix(y_train, y_train_pred)
- confusion_TST = confusion_matrix(y_test, y_test_pred)

- TP = confusion_TRN[1,1] # true positive
- TN = confusion_TRN[0,0] # true negatives
- FP = confusion_TRN[0,1] # false positives
- FN = confusion_TRN[1,0] # false negatives

- TP_TST = confusion_TST[1,1] # true positive
- TN_TST = confusion_TST[0,0] # true negatives
- FP_TST = confusion_TST[0,1] # false positives
- FN_TST = confusion_TST[1,0] # false negatives

- trainsensitivity= TP / float(TP+FN)
- trainspecificity= TN / float(TN+FP)
-
- testsensitivity= TP_TST / float(TP_TST+FN_TST)
- testspecificity= TN_TST / float(TN_TST+FP_TST)

- # Let us compare the values obtained for Train & Test:
- print('-'*30)
- print('On Train Data')
- print('-'*30)
- print("Accuracy   : {} %".format(round((trainaccuracy*100),2)))
- print("Sensitivity : {} %".format(round((trainsensitivity*100),2)))
- print("Specificity : {} %".format(round((trainspecificity*100),2)))
- print('-'*30)
- print('On Test Data')
- print('-'*30)
- print("Accuracy   : {} %".format(round((testaccuracy*100),2)))
- print("Sensitivity : {} %".format(round((testsensitivity*100),2)))
- print("Specificity : {} %".format(round((testspecificity*100),2)))
- print('-'*30)

- #Hyper Parameter Tuning
- from sklearn.model_selection import GridSearchCV
-
- dt_hp = DecisionTreeClassifier(random_state=43)
-

- params = {'max_depth':[3,5,7,10],
- 'min_samples_leaf':[5,10,15,20],
- 'min_samples_split':[10,12,18,20],
- 'criterion':['gini','entropy']}

- GS = GridSearchCV(estimator=dt_hp,param_grid=params,cv=5,n_jobs=-1, verbose=True, scoring='accuracy')

- print('Best Parameters:',GS.best_params_,end='\n\n')
- print('Best Score:',GS.best_score_)

- dt_hp = DecisionTreeClassifier(max_depth= 9,min_samples_leaf= 25, min_samples_split=5 ,random_state=43)

- dt_hp.fit(X_train, y_train)
-
- y_train_pred = dt_hp.predict(X_train)
- y_test_pred = dt_hp.predict(X_test)

- # Let's check the overall accuracy.
- trainaccuracy= accuracy_score(y_train, y_train_pred)
- testaccuracy= accuracy_score(y_test, y_test_pred)

- confusion_TRN = confusion_matrix(y_train, y_train_pred)
- confusion_TST = confusion_matrix(y_test, y_test_pred)

- TP = confusion_TRN[1,1] # true positive
- TN = confusion_TRN[0,0] # true negatives
- FP = confusion_TRN[0,1] # false positives
- FN = confusion_TRN[1,0] # false negatives

- TP_TST = confusion_TST[1,1] # true positive
- TN_TST = confusion_TST[0,0] # true negatives
- FP_TST = confusion_TST[0,1] # false positives
- FN_TST = confusion_TST[1,0] # false negatives

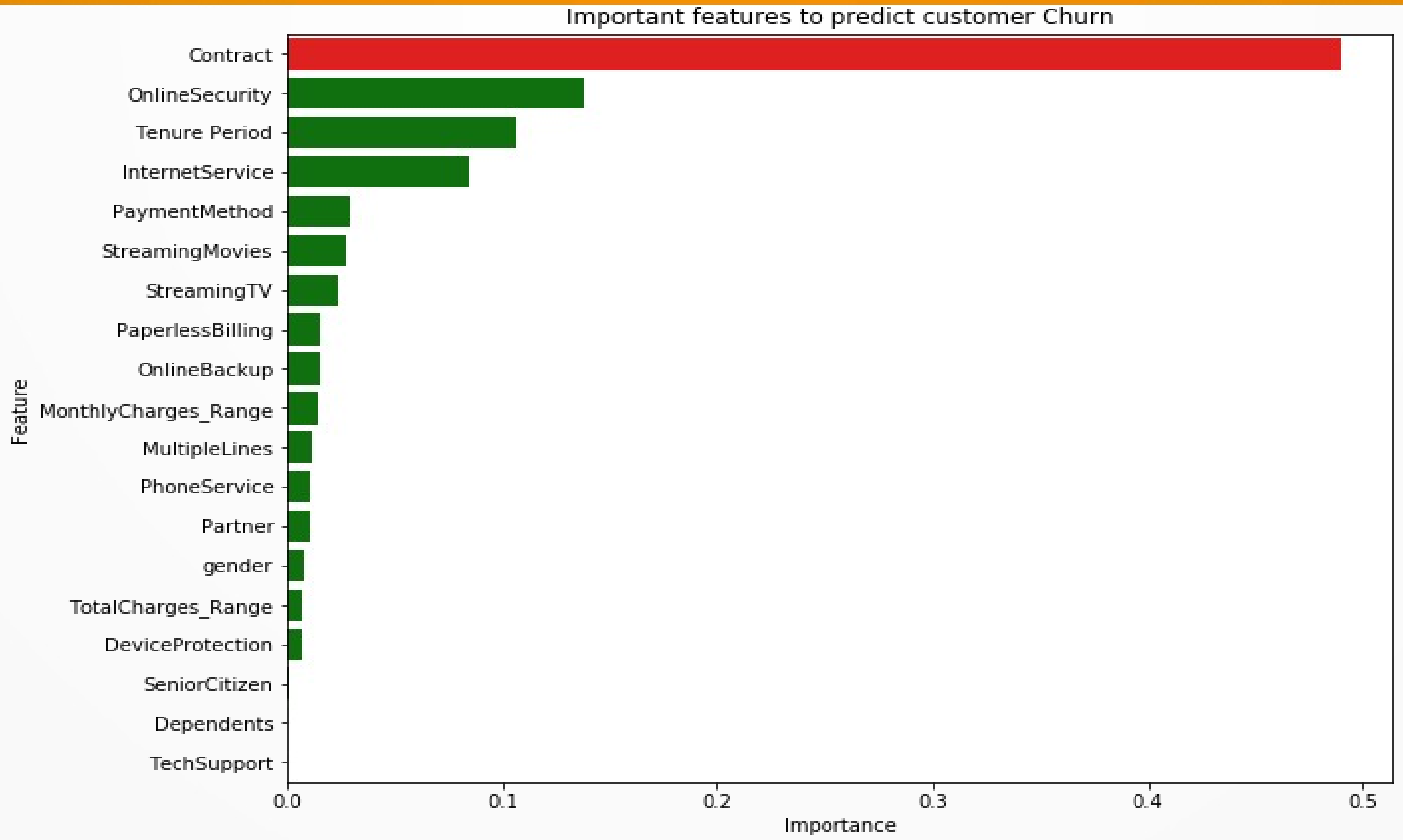
- trainsensitivity= TP / float(TP+FN)
- trainspecificity= TN / float(TN+FP)
-
- testsensitivity= TP_TST / float(TP_TST+FN_TST)
- testspecificity= TN_TST / float(TN_TST+FP_TST)
-

- # Let us compare the values obtained for Train & Test:
- print('-'*30)
- print('On Train Data')
- print('-'*30)
- print("Accuracy   : {} %".format(round((trainaccuracy*100),2)))
- print("Sensitivity : {} %".format(round((trainsensitivity*100),2)))
- print("Specificity : {} %".format(round((trainspecificity*100),2)))
- print('-'*30)
- print('On Test Data')
- print('-'*30)
- print("Accuracy   : {} %".format(round((testaccuracy*100),2)))
- print("Sensitivity : {} %".format(round((testsensitivity*100),2)))
- print("Specificity : {} %".format(round((testspecificity*100),2)))
- print('-'*30)

- #Feature Importance
- # let's create a dictionary of features and their importance values
- feat_dict= {}
- for col, val in sorted(zip(X_train.columns, dt_hp.feature_importances_),key=lambda x:x[1],reverse=True):
-   feat_dict[col]=val

- feat_df = pd.DataFrame({'Feature':feat_dict.keys(),'Importance':feat_dict.values()})
-
- feat_df

- #visualize the relative importance using Seaborn
- values = feat_df.Importance
- idx = feat_df.Feature
- plt.figure(figsize=(10,8))
- clrs = ['green' if (x < max(values)) else 'red' for x in values]
- sns.barplot(y=idx,x=values,palette=clrs).set(title='Important features to predict customer Churn')
- plt.show()



Important features to predict customer Churn
Contract
OnlineSecurity
Tenure Period
InternetService
PaymentMethod
StreamingMovies
StreamingTV
PaperlessBilling
OnlineBackup
MonthlyCharges_Range
MultipleLines
PhoneService
Partner
gender
TotalCharges_Range
DeviceProtection
SeniorCitizen
Dependents
TechSupport
Feature
0.0
0.1
0.2
0.3
0.4
0.5
Importance