High Level Machine Learning Classification Project Life Cycle

1.Domain Introduction

We have the customer data for a **telecom** company which offers many services like phone, internet, TV Streaming and Movie Streaming.

2.Problem Statement

"Find the Best model to predict behavior to retain customers. You can analyze all relevant customer data and develop focused customer retention programs."

3. Data Source

Available at : IBM watson analytics page (https://community.watsonanalytics.com/wp-content/uploads/2015/03/WA_Fn-UseC_-Telco-Customer-Churn.csv?cm_mc_uid=14714377267115403444551&cm_mc_sid_50200000=1257819154034445512 7&cm_mc_sid_52640000=36692891540344455130)

4. Data Description

This data set provides info to help you predict behavior to retain customers. You can analyze all relevant customer data and develop focused customer retention programs.

A telecommunications company is concerned about the number of customers leaving their landline business for cable competitors. They need to understand who is leaving. **Imagine that you’re an analyst at this company and you have to find out who is leaving and why.**

> The data set includes information about:
>
> - Customers who left within the last month – the column is called Churn
> - Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
> - Customer account information – how long they’ve been a customer, contract, payment method, paperless billing, monthly charges, and total charges
> - Demographic info about customers – gender, age range, and if they have partners and dependents

5. Identify the target variable

The Goal is to predict whether or not a particular customer is likely to retain services. This is represented by the Churn column in dataset. Churn=Yes means customer leaves the company, whereas Churn=No implies customer is retained by the company.

6. Read the data

In [ ]:

```
# Step1 : importing the Librabries like Numpy, Pandas, matplotlib, seaborn
#          Reading the Data from the CSV File

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings


# to remove the warnings coming in the output
if __name__ == '__main__':
    warnings.filterwarnings(action='ignore', category=UserWarning)
    warnings.filterwarnings(action='ignore', category=DeprecationWarning)

# Laoding The Data From the File
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv',index_col='customerID')

# the shapeof DataSet, number of rows and number of Columns
print(df.shape)
print(df.size)

print(" The Telecom Company Data Set contains %d rows and features %d"%(df.shape[0],df.shape[1]))
```

7. Inspect the data

[https://www.kaggle.com/blastchar/telco-customer-churn# (https://www.kaggle.com/blastchar/telco-customer-churn)](https://www.kaggle.com/blastchar/telco-customer-churn)

In [ ]:

```
# Inspecting the Data of the Customer , seeing the fields of the DataSet
print('-'*80,"\n The Records of Customer of Telecom \n",'-'*80)
print(df.head(3))
```

In [ ]:

```
#Step2: Analysing the customer Data

print('-'*80,"\n Analysing the Data by using functions like info, describe, checking for missing values \n",'-'*
# the datatypes of the fields
print(df.info())

# checking the mean,min, max values for numeric fields
print(df.describe())

# checking if there are any null values
print(df.isnull().sum())

print('-'*80,'''\n We observe that
      1. The attributes are mostly object type(Except -Tenure,monthly charges,Senior Citizen)
      2. Numeric Fields are-Senior Citizen with binary value(0,1), Tenure is maximum 72 months and avg-32 months
         monthly charges maximum is 118 rupees with average 64 Rupees
      3. Senior Citizen should be Categorical field
      4. And No Null Values ''')
```

In [ ]:

```
# Analysing  for non numeric fields

# printing the information on Nonnumerical Fields
print('-'*80,"\n Printing the non numerical fields , their unique values , \n",'-'*80)
print(df.describe(include=object))

print('-'*80,'''\n From Non-numerical Values , We observe that
      1. Gender is mostly balanced , mean both male and female are customers
      2. Maximum Customers have taken phone Service, Internet Service via Fiber optic is
         more preferred
      3. 50% Customers have opted for month-to-month billing , Max opted for paperless
         billing and mostly Electronic cheque
      4. Total Charges should be Float field , instead of Object Type
     ''')

```

In [ ]:

```
print('-'*80)
print(" Identifying the Unique Fields for the Categorical Fields\n",'-'*80)

# printing the Unique values for the Categorical/Object  Fields
# As the Data contains Total charges numerical , instead of Float type , having max-val=10
# so that such columns are not considered.

def print_unique_values(df,max_val = 10):
    for col in df:
        if (len(df[col].unique()) < 10):
            print(df[col].name, ":" ,df[col].unique())

print_unique_values(df)
```

8. Data Manipulation

In [ ]:

```
#Defined A Common Function to remove spaces, special characters, brackets from column Names
# and Fields Values of a DataSet

def filter_df(df):

    import string
#     print(string.punctuation)

    def remove_punctuation(s):
        s = ''.join([i for i in s if i not in frozenset(string.punctuation) and i not in ' '])
        return s

    # To replace the spaces with '_', and removing brackets from Column NAmes
    df.columns = df.columns.str.strip().str.replace(' ', '_').str.replace('(', '').str.replace(')', '')

    # removing punctuation from the object type column values
    df_categorical = df.select_dtypes(include=object)
#    print(len(df_categorical))
    for col in df_categorical.columns:
        df[col] = df[col].apply(remove_punctuation)
    return df

# calling the function to Filter the Column Names and Column Values
df = filter_df(df)


print('-'*80,"\n The DataSet after removing the punctuations \n",'-'*80)
print(df.head())
```

In [ ]:

```
# Creating a Copy of the DataSet
df_org = df
print(df_org.head())
```

Data Manipulation

1. We Observe from the Data , that Total Charges Atrribute should be Float instead of object variable.

2. And Senior Citizen Atrribute should be Categorical , instead of Int

In [ ]:

```
# We need to convert the Total Charges from object type to Numeric
#df = df_org

# replacing the Space if found in Field with null value
df['TotalCharges'] = df['TotalCharges'].replace(r'\s+', np.nan, regex=True)
# converting to numeric
#print(df['TotalCharges'].head())
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'])


print('-'*80,"\n Total Charges converted to numeric Field \n",'-'*80)
print(df.info())
```

In [ ]:

```
# making The SeniorCitizen Column , Categorical

df['SeniorCitizen'] = df['SeniorCitizen'].replace({1:'Yes',0:'No'})

print(" After Making Senior Citizen Field Categorical , checking for Nulls \n",'-'*80)
print(df.isnull().sum())
print(df.head())
```

In [ ]:

```
# We Observe the Conversions There are 11 values null in TotalCharges,
#print(df[df['TotalCharges'].isnull()== True].head())
print('-'*80,'''\n We Observe after converting TotalCharges and Senior Citizen columns
       1. Totalcharges Column has some null values
       2. Tenure for Such customers is 0 , hence we can fill null values by mean value so will not impact much \
df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].mean())
print(df.isnull().sum())
```

We Observe that on converting the Field to Numeric , there are some null values for Total Charges created for customers who has not opted out of Service The Tenure for these customers is 0 ,but there may be additional charges so we can put the total charges as mean value

Imputation

In [ ]:
```
# the Final Data Set
print(df.head())
```

9. Exploratory Data Analysis

In [ ]:
```
# IDentifying the Categorical and Numerical FEatures of the DataSet and creating two Datasets

df_categorical = df.select_dtypes(include=object)

column_categorical = df_categorical.columns
print('-'*80,"\n The Categorical Fields are \n",'-'*80)
print(column_categorical)
```

In [ ]:
```
# The Numerical Fields in the DataSet
df_numerical = df.select_dtypes(include=[np.float,np.int64])

print('-'*80,"\n The Numerical Fields are \n",'-'*80)
column_numerical = df_numerical.columns
print(column_numerical)
```

Univariate Analysis

In [ ]:

```
# Function to display the distribution of each variable for
# Categorical Fields as well as Numerical fields , based on the object_mode value

def display_plot(df, col_to_exclude, object_mode = True):
    """
     This function plots the count or distribution of each column in the dataframe based on specified inputs
     @Args
       df: pandas dataframe
       col_to_exclude: specific column to exclude from the plot, used for excluded key
       object_mode: whether to plot on object data types or not (default: True)

     Return
       No object returned but visualized plot will return based on specified inputs
    """
    n = 0
    this = []

    if object_mode:
        nrows = 4
        ncols = 4
        width = 20
        height = 20

    else:
        nrows = 2
        ncols = 2
        width = 14
        height = 10


    for column in df.columns:
        if object_mode:
            if (df[column].dtypes == 'O') & (column != col_to_exclude):
                this.append(column)


        else:
            if (df[column].dtypes != 'O'):
                this.append(column)

    totcnt = len(this)
```

```
    fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=False, figsize=(width, height))
    for row in range(nrows):
        for col in range(ncols):
            if (totcnt == 0):
                break
            if object_mode:
                g = sns.countplot(df[this[n]], ax=ax[row][col])
            else:
                g = sns.distplot(df[this[n]], ax = ax[row][col])
            totcnt -= 1




            ax[row,col].set_title("Column name: {}".format(this[n]))
            ax[row, col].set_xlabel("")
            ax[row, col].set_ylabel("")
            n += 1
    plt.show();
    return None

```

In [ ]:

```
# Displaying the Plots for each variable for Categorical Fields
# Step3: Data Visualisation of the Customer Data

display_plot(df, 'customerid', object_mode = True)


print('-'*80, "\n The Main observations from the Categorical Data as follows \n",'-'*80)
print('''      1. Almost Equal Percentage of Male and Female Customers
      2. Most of Customers have Phone Service
      3. FiberOptics  is prefered InternetService way
      4. Most of the Customers have taken Monthly Plan
      5. Payment preferred by Customers is paperless and mostly pay through
         Electronic Cheque
      6. Very few Customers have taken online services, like Backup , Streaming ,
         opportunity for company to sell that to customers
      ''')
```

In [ ]:

```
# Observing the Numerical Fields
df.head()

display_plot(df, 'customerid', object_mode = False)

print('-'*80, "\n The Main observations from the Numerical Data as follows \n",'-'*80)
print('''     1. Maximum Customers have taken month wise Tenure and followed by very long tenures,
        Hence Maximum customers pay monthly charges

      ''')
```

Feature Engineering

Based on the value of the services the subscribers subscribed to, there are **yes**, **no**, and **no phone / internet service**. These are somewhat related to primary products. Examples are illustrated through *panda crosstab* function below:

1. **Phone service (Primary) and Multiple lines (Secondary)**

   - If the subscribers have phone service, they may have multiple lines (yes or no).
   - But if the subscribers don't have phone service, the subscribers will never have multiple lines.

In [ ]:

```
pd.crosstab(index = df["PhoneService"], columns = df["MultipleLines"])
```

2. **Internet Service (Primary) and other services, let's say streaming TV (secondary)**

   - If the subscribers have Internet services (either DSL or Fiber optic), the subscribers may opt to have other services related to Internet (i.e. onlineSecurity, OnlineBackup , DeviceProtection,TechSupport,StreamingTV, StreamingMovies,).
   - But if the subscribers don't have the Internet services, this secondary service will not be available for the subscribers.

In [ ]:

```
# Show For StreamingTV Feature, which has yes, no , no Internet option
pd.crosstab(index = df["InternetService"], columns = df["StreamingTV"])
```

So We can transform the Secondary Attributes(MultipleLines, StreamingTV...) to have only two values , Yes and No and Transform **No Phone / Internet service** to be the same **No**

In [ ]:

```python
# Function Defined to identify all the columns/Services that have more than 2 options
# as mentioned above and replace "No Phone Service" &"No Internet Service" to "No"

def convert_no_service (df):
    col_to_transform = []
    for col in df.columns:
        if (df[col].dtype == 'O') & (col != 'customerid'):
            if len(df[df[col].str.contains("No")][col].unique()) > 1:
                col_to_transform.append(col)

    print("Total column(s) that will be transformed: {}".format(col_to_transform))
    for col in col_to_transform:
        df.loc[df[col].str.contains("No"), col] = 'No'

    return df
```

In [ ]:

```python
df = convert_no_service(df)
```

In [ ]:

```python
# Let's see the data after transformation.
print("The Data Can be Visualised again after Transforming the Secondary Columns")
display_plot(df, 'customerid', object_mode = True)
```

The Conclusions that can be made after transforming Columns are

1. Most customers have phone Service and Single line (opportunity 1 for multiline)

2. Customers having Internet Service , have very few Secondary services (StreamingTV, OnlineBackUp...) (Opportunity2 - for cross-sell)

In [ ]:

```python
#We have observed above each Columns Behavior for the customers, now we can see the impact
#of these columns on Target Column Churn


#function to compare each Column with Churn
def compare_Features(cols_cmp,Target = 'Churn'):
    for col in cols_cmp:
        print(col)
        pd.crosstab(index = df[col],columns=df[Target],margins=True).plot(kind='bar',figsize=(5,3))
        for lbl in df[col].unique():
            Fea_cnt = len(df[(df[col] == lbl)&(df['Churn']== 'Yes')])
            print(" Percent of ({} {}) Left Company {}".format(lbl ,col, (Fea_cnt/Churn_cnt)*100))


#compare_Features(df,df.Churn)
cols = column_categorical
Churn_cnt = len(df[(df['Churn'] == 'Yes')])
cols_cmp = cols[0:5]

#Comparing the First 5 Columns /FEatures with Churn
compare_Features(cols_cmp)
print('-'*80)
print(''' The Conclusion that can be made from Comparing with Churn are:
        1. Churning of Customers was independent of Gender (i.e no impact)
        2. More percentage Young Customers, Non Partnered , having no Dependents and have
           PhoneService left Company
        3. Company could make customers partners, and see if there is any issue with Phone Service ''')
```

In [ ]:

```
# Comparing the Next 5 Columns /FEatures with Churn
cols_cmp = cols[5:10]
compare_Features(cols_cmp)
print('-'*80)
print(''' The Conclusion that can be made from Comparing with Churn are:
        1. Churning of Customers was independent of whether they had Multiple Lines
        2. More percentage Customers taking FiberOptic Internet Service, not having Online
           Services ...left Company
        3. Company could check if some issue with FiberOptic Service, and try selling online
           services ''')
```

In [ ]:

```
# Comparing the Next 5 Columns /FEatures with Churn
cols_cmp = cols[10:16]

compare_Features(cols_cmp)
print('-'*80)
print(''' The Conclusion that can be made from Comparing with Churn are:
        1. Percentage of Customers having no TechSupport left Company
        2. Having Streaming Tv, Movies or not had same impact on leaving Company
        3. Maximum percentage of Customers with MonthtoMonth Contract Left
        3. Company could try making customers with longer contracts to retain them  ''')
```

In [ ]:

```
# To check the relation between Tenure and Partner Atrributes
plt.figure(figsize=(8,4))
sns.countplot(x=df['tenure'],hue=df.Partner);
```

We Observer , Most of the Customers that are Partners Stay Longer with The Company. So Being a Partner is a Plus-Point For the Company as they will Stay Longer with Them.

Let's Check for Outliers in Monthly Charges And Total Charges Using Box Plots

In [ ]:

```
df.boxplot('MonthlyCharges');
print(" No Outliers in Monthly Charges")
```

In [ ]:

```
# For Total Charges Attribute , we observe there are outliers

# df.boxplot('TotalCharges')
# print(df['TotalCharges'].describe())
Tot_mean = df['TotalCharges'].mean()
Tot_std  = df['TotalCharges'].std()
std_low =  Tot_mean - 3 * Tot_std
std_upp =  Tot_mean + 3 * Tot_std
print("The number of TotalCharges below (mean -2 std)" + str(std_low) + " are " , len(df[df['TotalCharges'] < st
print("The number of TotalCharges above (mean +2 std)  " + str(std_upp) + " are ", len(df[df['TotalCharges'] > s
print("We Observe 99percent confidence of Total Charges values lie between %0.2f and %0.2f " % (std_low, std_upp
# and %0.3f,std_upp))


```

In [ ]:

```
#df.drop(df[df['TotalCharges'] > std_upp],axis=0, inplace=True)
df = df_org
Out = df[df['TotalCharges'] > std_upp]
print(df.shape)
Tot_new =  df[(df['TotalCharges'] > std_upp)].index
print(Tot_new)
#print(df.drop(Out, inplace=True))
df = df.drop(Tot_new,axis=0)
print(df.shape)

```

Monthly Charges don't have any Outliers so we don't have to Get into Extracting Information from Outliers.

In [ ]:

```
## correlation matrix

# Let's Check the Correaltion Matrix in Seaborn
sns.heatmap(df.corr(),xticklabels=df.corr().columns.values,yticklabels=df.corr().columns.values,annot=True);
```

Here We observe Tenure and Total Charges Atrributes are correlated 50% and also Monthly charges and Total Charges are also correlated 31% with each other. But It is not above 0.90 , so we can retain columns

we can assume from our domain expertise that , Total Charges ~ Monthly Charges * Tenure + Additional Charges(Tax).

Bucketing

In [ ]:
```
# Bucketting the Tenure Feature into slabs
df['tenure'].describe()
```

In [ ]:
```
# Changing the Tenure to categorical column

print(" Min and Max Values of tenure  = %0.2f & %0.2f  "%(min(df['tenure']), max(df['tenure'])))

def tenure_lab(telcom) :

    if telcom["tenure"] <= 12 :
        return "Tenure_0-12"
    elif (telcom["tenure"] > 12) & (telcom["tenure"] <= 24 ):
        return "Tenure_12-24"
    elif (telcom["tenure"] > 24) & (telcom["tenure"] <= 48) :
        return "Tenure_24-48"
    elif (telcom["tenure"] > 48) & (telcom["tenure"] <= 60) :
        return "Tenure_48-60"
    elif telcom["tenure"] > 60 :
        return "Tenure_gt_60"


df["tenure_group"] = df.apply(lambda x:tenure_lab(x),axis = 1)
```

In [ ]:
```
print(" Telecom Data with Tenure in Buckets\n",'-'*80)
print(df.head())
```

10. Data preprocessing

Encoding categorical variable

In [ ]:
```
# Categorical Features and count of unique values
#print(df.nunique())
bin_cols = df.nunique()[df.nunique() < 3].keys().tolist()

multi_cols = df.nunique()[(df.nunique() < 7) & (df.nunique() > 2)].keys().tolist()
num_cols = df.nunique()[(df.nunique() > 7) ].keys().tolist()
print("The Binary Categorical Columns are : \n",'-'*80)
print("Binary Value Categorical Columns \n",'-'*80)
print(bin_cols)
print("Multiple Value Categorical Columns \n",'-'*80)
print(multi_cols)
print("Numerical Columns \n",'-'*80)
print(num_cols)
```

In [ ]:
```
#Label encoding for all the Categorical Values
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

#Encoding the binary columns


le = LabelEncoder()
for i in bin_cols :
    df[i] = le.fit_transform(df[i])

#Duplicating columns for multi value columns
df = pd.get_dummies(data = df,columns = multi_cols )
```

In [ ]:
```
df.head()
```

In [ ]:
```
list(df.columns)
```

Normalizing features

In [ ]:
```
telcom = df

#Scaling Numerical columns as there is large difference in values , to make them comparable
'''
ransforms features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the trainin
'''

std = MinMaxScaler()


scaled = std.fit_transform(telcom[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)
```

In [ ]:
```
print(scaled.shape)
scaled.head(2)
```

In [ ]:
```
#dropping original values and  merging new scaled values for numerical columns
df_telcom_og = telcom.copy()
telcom = telcom.drop(columns = num_cols,axis = 1)

telcom.reset_index(drop=False, inplace=True)

telcom = pd.concat([telcom, scaled], axis=1)

telcom.set_index('customerID', inplace=True)

print("The DataSet with scaled numerical values \n",'-'*80)

print(telcom.shape)
print(telcom.head())
```

Spliting The Data Set into Train/Val/Test data

In [ ]:
```
telcom.info()
```

In [ ]:
```
# importing the metric for performance calculation

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from sklearn.metrics import roc_auc_score,roc_curve,scorer
from sklearn.metrics import f1_score
import statsmodels.api as sm
from sklearn.metrics import precision_score,recall_score
#splitting train and test data

# telcom = df
target_col = telcom["Churn"]
telcom = telcom.drop('Churn',axis=1)
#target_col = target_col.values.reshape(-1,1)

X_train,X_test,y_train,y_test = train_test_split(telcom,target_col,test_size = 0.25 ,random_state = 111)

print(" The Training and Test Data Set size after Splitting \n",'-'*80)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

11. Model Building

In [ ]:
```
# importing the Machine Learning Models
from sklearn.dummy import DummyClassifier

# Machine Learning
from sklearn import tree , linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge, Lasso, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost.sklearn import XGBClassifier
```

In [ ]:
```
# validation
from sklearn import datasets, model_selection, metrics , preprocessing
```

In [ ]:
```
# Grid and Random Search
import scipy.stats as st
from scipy.stats import randint as sp_randint
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [ ]:
```
# Metrics
from sklearn.metrics import precision_recall_fscore_support, roc_curve, auc
```

In [ ]:
```
#utilities
import time
import io, os, sys, types, time, datetime, math, random
```

In [ ]:

```
# calculate the fpr and tpr for all thresholds of the classification
def plot_roc_curve(y_test, preds):
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()



# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, X_test, cv):
    # One Pass
    model = algo.fit(X_train, y_train)
    test_pred = model.predict(X_test)
    if (isinstance(algo, (LogisticRegression,
                          KNeighborsClassifier,
                          GaussianNB,
                          DecisionTreeClassifier,
                          RandomForestClassifier,
                          GradientBoostingClassifier))):
        probs = model.predict_proba(X_test)[:,1]
    else:
        probs = "Not Available"
    acc = round(model.score(X_test, y_test) * 100, 2)
    # CV
    train_pred = model_selection.cross_val_predict(algo,
                                                  X_train,
                                                  y_train,
                                                  cv=cv,
                                                  n_jobs = -1)
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)
    return train_pred, test_pred, acc, acc_cv, probs

# Utility function to report best scores
```

```python
def report(results, n_top=5):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")

```

Baseline model with DummyClassifier

In [ ]:

```python
# Instantiating a Dummy Classifier Model ,that will be the baseline for
# comparisions for all other models

clf = DummyClassifier(strategy='most_frequent',random_state=0)
clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)

```

In [ ]:

```
preds = clf.predict(X_test)


# dummyistic Regression
start_time = time.time()
train_pred_dummy, test_pred_dummy, acc_dummy, acc_cv_dummy, probs_dummy = fit_ml_algo(DummyClassifier(strategy='
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
dummy_time = (time.time() - start_time)
print("Accuracy for Dummy Classfier Model: %s" % acc_dummy)
print("Accuracy CV 10-Fold: %s" % acc_cv_dummy)
print("Running Time: %s" % datetime.timedelta(seconds=dummy_time))

print(" Classification report for Training Data\n",'-'*80 )
print (metrics.classification_report(y_train, train_pred_dummy))

print(" Classification report for Test Data\n",'-'*80 )
print (metrics.classification_report(y_test, test_pred_dummy))

```

Select Candidate Algorithms

1. KNN

2. Logistic Regression

3. Naive Bayes

4. Decision Tree

5. Random Forest

6. Gradient Boosted Trees

In [ ]:

```
# Model 1 - k-Nearest Neighbors

start_time = time.time()
train_pred_knn, test_pred_knn, acc_knn, acc_cv_knn, probs_knn = fit_ml_algo(KNeighborsClassifier(n_neighbors = 3
                                                                                                  n_jobs = -1),
                                                                                                  X_train,
                                                                                                  y_train,
                                                                                                  X_test,
                                                                                                  10)
knn_time = (time.time() - start_time)
print("K Nearest Neighbour Model \n",'-'*80)
print("Accuracy for KNN Model: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time: %s" % datetime.timedelta(seconds=knn_time))

print (metrics.classification_report(y_train, train_pred_knn))

print (metrics.classification_report(y_test, test_pred_knn))
print("ROC_AUC Curve for KNN \n",'-'*80)
plot_roc_curve(y_test, probs_knn)
```

In [ ]:

```
# Specify parameters and distributions to sample from
param_dist = {'penalty': ['l2', 'l1'],
                          'class_weight': [None, 'balanced'],
                          'C': np.logspace(-20, 20, 10000),
                          'intercept_scaling': np.logspace(-20, 20, 10000)}



# Run Randomized Search
n_iter_search = 10
lrc = LogisticRegression()
random_search = RandomizedSearchCV(lrc,
                                   n_jobs=-1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
print(report(random_search.cv_results_))
print('-'*80)
print(random_search.best_estimator_)
print('-'*80)
print(random_search.best_params_)

```

In [ ]:

```
# Modeling the Data with various Models , one by one , since we want to see individual results
# for each model in detail , hence not defined a function for same

# Model 1 - Logistic Regression
start_time = time.time()
train_pred_log, test_pred_log, acc_log, acc_cv_log, probs_log = fit_ml_algo(LogisticRegression(n_jobs = 1,penalt
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
log_time = (time.time() - start_time)
print(" Logistic REgression Model \n",'-'*80)
print("Accuracy for Logistic Regression : %s" % acc_log)
print("Accuracy for CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))

print (metrics.classification_report(y_train, train_pred_log))

print (metrics.classification_report(y_test, test_pred_log))
print(" ROC-AUC Curve for Logistic Regression Model\n",'-'*80)
plot_roc_curve(y_test, probs_log)
```

In [ ]:

```
# Model 3 - Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, test_pred_gaussian, acc_gaussian, acc_cv_gaussian, probs_gau = fit_ml_algo(GaussianNB(),
                                                                                                 X_train,
                                                                                                 y_train,
                                                                                                 X_test,
                                                                                                 10)
gaussian_time = (time.time() - start_time)
print(" Naives Bayes Gausian Model \n",'-'*80)
print("Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
print("Running Time: %s" % datetime.timedelta(seconds=gaussian_time))

print (metrics.classification_report(y_train, train_pred_gaussian))

print (metrics.classification_report(y_test, test_pred_gaussian))
print("ROC_AUC Curve for NB Model\n",'-'*80)
plot_roc_curve(y_test, probs_gau)
```

In [ ]:

```
# Model 4 - Decision Tree Classifier
start_time = time.time()
train_pred_dt, test_pred_dt, acc_dt, acc_cv_dt, probs_dt = fit_ml_algo(DecisionTreeClassifier(),
                                                                        X_train,
                                                                        y_train,
                                                                        X_test,
                                                                        10)
dt_time = (time.time() - start_time)
print(" Decision Tree Model \n",'-'*80)
print("Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))

print (metrics.classification_report(y_train, train_pred_dt))

print (metrics.classification_report(y_test, test_pred_dt))
print("ROC_AUC curve for Decision Tree Model \n",'-'*80)
plot_roc_curve(y_test, probs_dt)
```

In [ ]:

```
# Model 5 - Random Forest Classifier - Random Search for Hyperparameters

# Utility function to report best scores
# def report(results, n_top=5):
#     for i in range(1, n_top + 1):
#         candidates = np.flatnonzero(results['rank_test_score'] == i)
#         for candidate in candidates:
#             print("Model with rank: {0}".format(i))
#             print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
#                   results['mean_test_score'][candidate],
#                   results['std_test_score'][candidate]))
#             print("Parameters: {0}".format(results['params'][candidate]))
#             print("")



# Specify parameters and distributions to sample from
param_dist = {"max_depth": [10, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 20),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}


# Run Randomized Search
n_iter_search = 10
rfc = RandomForestClassifier(n_estimators=10)
random_search = RandomizedSearchCV(rfc,
                                   n_jobs = -1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
print(report(random_search.cv_results_))
print('-'*80)
print(random_search.best_estimator_)
print('-'*80)
```

```
print(random_search.best_params_)
```

In [ ]:

```
#Final Model 5  Random Forest Classifier
start_time = time.time()
rfc = RandomForestClassifier(n_estimators=10,
                             bootstrap=True,
                             max_depth=10,
                             min_samples_leaf=9,
                             min_samples_split=7,
                             criterion='entropy',
                             max_features=2)
train_pred_rf, test_pred_rf, acc_rf, acc_cv_rf, probs_rf = fit_ml_algo(rfc,X_train,y_train,X_test,10)
rf_time = (time.time() - start_time)
print(" Random Forest Model  \n",'-'*80)
print("Accuracy for RF: %s" % acc_rf)
print("Accuracy CV 10-Fold: %s" % acc_cv_rf)
print("Running Time: %s" % datetime.timedelta(seconds=rf_time))

print (metrics.classification_report(y_train, train_pred_rf))

print (metrics.classification_report(y_test, test_pred_rf))
print(" ROC-AUC Curve for RF\n",'-'*80)
plot_roc_curve(y_test, probs_rf)
```

In [ ]:

```
# Model 6 -  Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, test_pred_gbt, acc_gbt, acc_cv_gbt, probs_gbt = fit_ml_algo(GradientBoostingClassifier(),
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
gbt_time = (time.time() - start_time)
print(" Gradient Boost Tree\n",'-'*80)
print("Accuracy: %s" % acc_gbt)
print("Accuracy CV 10-Fold: %s" % acc_cv_gbt)
print("Running Time: %s" % datetime.timedelta(seconds=gbt_time))

print (metrics.classification_report(y_train, train_pred_gbt))

print (metrics.classification_report(y_test, test_pred_gbt))
print(" ROC-AUC Curve for GBT \n",'-'*80)
plot_roc_curve(y_test, probs_gbt)
```

In [ ]:

```
def xgb_f1(y, t):
    #
    # Function to evaluate the prediction based on F1 score, this will be used as evaluation metric when trainin
    # Args:
    #   y: label
    #   t: predicted
    #
    # Return:
    #   f1: F1 score of the actual and predicted
    #
    t = t.get_label()
    y_bin = [1. if y_cont > 0.5 else 0. for y_cont in y]   # change the prob to class output
    return 'f1', f1_score(t, y_bin)

best_xgb = XGBClassifier(objective = 'binary:logistic',
                         colsample_bylevel = 0.7,
                         colsample_bytree = 0.8,
                         gamma = 1,
                         learning_rate = 0.15,
                         max_delta_step = 3,
                         max_depth = 4,
                         min_child_weight = 1,
                         n_estimators = 50,
                         reg_lambda = 10,
                         scale_pos_weight = 1.5,
                         subsample = 0.9,
                         silent = True,
                         n_jobs = 4
                        )

xgbst = best_xgb.fit(X_train, y_train, eval_metric = xgb_f1, eval_set = [(X_train, y_train), (X_test, y_test)],
             early_stopping_rounds = 20)
```

In [ ]:

```
train_pred_xgbst, test_pred_xgbst, acc_xgbst, acc_cv_xgbst, probs_xgbst = fit_ml_algo(xgbst,
                                                                    X_train,
                                                                    y_train,
                                                                    X_test,
                                                                    10)
```

In [ ]:

```
import xgboost as xgb
xgb.plot_importance(best_xgb, max_num_features = 15)
plt.show();
```

Compare all models

In [ ]:

```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes',
              'Decision Tree',
              'Gradient Boosting Trees'],
    'Score': [
        acc_knn,
        acc_log,
        acc_rf,
        acc_gaussian,
        acc_dt,
        acc_gbt,

    ]})

print(" WE observe from the Table , GBT, Logistic Regression,  Random Forest have good accuracy\n",'-'*80)
models.sort_values(by='Score', ascending=False)
```

In [ ]:

```
# printing a ROC-AUC Curve for the All models in same plot to compare
models = [
    'KNN',
    'Logistic Regression',
    'Random Forest',
    'Naive Bayes',
    'Decision Tree',
    'Gradient Boosting Trees',

]
probs = [
    probs_knn,
    probs_log,
    probs_rf,
    probs_gau,
    probs_dt,
    probs_gbt
]
colors = [
    'blue',
    'green',
    'red',
    'cyan',
    'magenta',
    'yellow',
    'black',
]
```

In [ ]:

```
def plot_roc_curves(y_test, prob, model):
    fpr, tpr, threshold = metrics.roc_curve(y_test, prob)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, 'b', label = model + ' AUC = %0.2f' % roc_auc, color=colors[i])
    plt.legend(loc = 'lower right')

for i, model in list(enumerate(models)):
    plot_roc_curves(y_test, probs[i], models[i])

plt.show()
```

Interpretation

1. The Conclusions that can be made from the graph and Probability scores from the test dataset

1. The DummyClassifier , used For Baseline , gave accuracy of 72%

2. We Observe that based on ROC_AUC, the Algoritms that the Model is trained the best accuracy is given by Gradient Boost , Logistic Regression, Random Forest and Naives Bayes Algoritms

3. The Probability score metric also reconfirms the conclusion.

Hence , The Best Model Trained for This test DataSet is Gradient Boost with score 80%

2. The Observations Made From DataSet after Analysis

1. Customer Churning is irrepective of Gender

2. Maximum Customers have Phone Service with single Line , and churn out more in them

3. Maximum Customers having internet, prefer Fiber Optic, But Churn High too in this category

4. Maximum Customers availing month-monthBilling and high churn as compared to long tenure customers

3.Customer Retension Programs Suggested

1. Analyse issues faced by customer in FiberOptics Internet offering and resolve them at earliest, to prevent Churning in this segment

2. Convert Customers from Month to Month billing to long tenure to Churn Less

3. Analyse if Phone Service having any issues and resolve at the earliest, to prevent Churning

4. Less Customers having Multiline and Online Services - Company can focus and sell models

5. Company to offer Partnership to Customers ,they would stay longer duration.