Week 5 Homework: Automated ML

In [1]:
```
#import packages
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")
```

In [14]:
```
#Install TPOT
#!pip install TPOT
```

In [15]:
```
#install XGBoost
```

I chose not to install XGboost because it stalled my computer for hours on first attempt.

In [2]:
```
# import TPOT packages
from tpot import TPOTClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

import timeit
```

In [3]:
```
#Load prepped data
import pandas as pd

df = pd.read_csv('prepped_churn_data.csv')
df.head()
```

Out[3]:

	Cust_number	customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	To
0	1	5375	1	0	0	2	29.85	
1	2	3962	34	1	1	3	56.95	
2	3	2564	2	1	0	3	53.85	
3	4	5535	45	0	1	0	42.30	
4	5	6511	2	1	0	2	70.70	

In [4]:
```
#break our data into features and targets, and train and test sets
features = df.drop('Churn', axis=1)
targets = df['Churn']

X_train, X_test, y_train, y_test = train_test_split(features, targets, stratify=t
```

In [5]:
```
#run TPOT
%time
tpot = TPOTClassifier(generations=5, population_size=50, verbosity=2, n_jobs=-1,
tpot.fit(X_train, y_train)
print(tpot.score(X_test, y_test))
```

```
CPU times: total: 0 ns
Wall time: 0 ns
Imputing missing values in feature set

Optimization Progress:   0%|          | 0/300 [00:00<?, ?pipeline/s]


Generation 1 - Current best internal CV score: 0.793069292738167

Generation 2 - Current best internal CV score: 0.7932588658582037

Generation 3 - Current best internal CV score: 0.7957209870703248

Generation 4 - Current best internal CV score: 0.7957209870703248

Generation 5 - Current best internal CV score: 0.7957209870703248

Best pipeline: ExtraTreesClassifier(input_matrix, bootstrap=False, criterion=gi
ni, max_features=0.6000000000000001, min_samples_leaf=20, min_samples_split=7,
n_estimators=100)
Imputing missing values in feature set
0.8126064735945485
```

According to TPOT, the best model is the Logistic Regression model.

In [30]:
```
#export model
#tpot.export('predict_churn.py')
```

In [9]:
```
#save model to disk
from IPython.display import Code

Code('predict_churn.py')
```

Out[9]:
```
#load packages
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline, make_union
from tpot.builtins import OneHotEncoder, StackingEstimator
from tpot.export_utils import set_param_recursive

# NOTE: Make sure that the outcome column is labeled 'target' in the data file
#load dataset and split out into elements
df = pd.read_csv('prepped_churn_data.csv')
features = df.drop('Target', axis=1)
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, df['Target'], random_state=17)


# Average CV score on the training set was: 0.7936378329176341
exported_pipeline = make_pipeline(
    StackingEstimator(estimator=ExtraTreesClassifier(bootstrap=False, criterion
="gini", max_features=0.7000000000000001, min_samples_leaf=17, min_samples_spli
t=2, n_estimators=100)),
    OneHotEncoder(minimum_fraction=0.05, sparse=False, threshold=10),
    LogisticRegression(C=10.0, dual=False, penalty="l2")
)
# Fix random state for all the steps in exported pipeline
set_param_recursive(exported_pipeline.steps, 'random_state', 17)

#Fit the model
exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)

#Make a prediction
row=[108]
yhat=exported_pipeline.predict([row])
print('Chance of Churn: '%yhat[0])
```

In [8]:
```
%run predict_churn.py
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~\Documents\Regis\Fall_21_8w1\week5\predict_churn.py:29, in <module>
     26 set_param_recursive(exported_pipeline.steps, 'random_state', 17)
     28 #Fit the model
---> 29 exported_pipeline.fit(training_features, training_target)
     30 results = exported_pipeline.predict(testing_features)
     32 #Make a prediction

File ~\anaconda3\lib\site-packages\sklearn\pipeline.py:378, in Pipeline.fit(s
elf, X, y, **fit_params)
    352 """Fit the model.
    353 
    354 Fit all the transformers one after the other and transform the
   (...)
    375     Pipeline with fitted steps.
    376 """
    377 fit_params_steps = self._check_fit_params(**fit_params)
--> 378 Xt = self._fit(X, y, **fit_params_steps)
```

Summary and Analysis

In this exercise, I ran the prepped churn data in TPOT, an autoML tool that helps us find the best model for predicting whether a client will churn. TPOT indicated that the Logistic Regression model we ran a few weeks ago is the best model. I then exported the model and modified the model code in an attempt to create a prediction which would give us the predicted chance of churn for each model. Unfortunately I am struggling with getting this code functional. If I was successful, we would have an autoML tool that would be able to predict the chance of churn for incoming customers with new data that was not included in the dataset we used to build the model. The model would output a score that could be translated into a percentage of likelihood of churn. As some of the data elements are time-related (total charges, tenure), the model should be re-run for existing clients 1x/month in order to keep the scores accurate.

As I have mentioned previously, TPOT indicated that the average accuracy score on the training dataset was ~79% for this model, which in reality is likely an unacceptably low accuracy score. Going forward, the next steps to improve this score would be to ingest more data, tune the model's hyperparameters, ensure the data is cleaned, and ultimately reassess the accuracy and fit of the model once these actions have been taken, possibly by running TPOT again.