Matt Crnkovich Week 5 Assignment

Brief:

Using our prepared churn data from week 2:

- use pycaret to find an ML algorithm that performs best on the data
  - Choose a metric you think is best to use for finding the best model
- save the model to disk
- create a Python script/file/module with a function that takes a pandas dataframe as an input and returns the probability of churn for each row in the dataframe
  - your Python file/function should print out the predictions for new data (new_churn_data.csv)
- test your Python module and function with the new data, new_churn_data.csv
- write a short summary
- upload Jupyter and Python file to Github repository

Code and process

Foundation work

In [1]:
```
import pandas as pd
import pickle
from pycaret.classification import setup, compare_models, predict_model, save_model
```

In [2]:
```
df_initial = pd.read_csv('prepped_churn_data.csv', index_col='customerID')
#We can exclude making predictions on the calculated field
#as it does not exist in the new churn data
df = df_initial.drop(['monthly_charges_calculated'], axis=1)
df
```

Out[2]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Chu
7590-VHVEG	1	0	0	1	29.85	29.85	
5575-GNVDE	34	1	1	0	56.95	1889.50	
3668-QPYBK	2	1	0	0	53.85	108.15	
7795-CFOCW	45	0	1	3	42.30	1840.75	
9237-HQITU	2	1	0	1	70.70	151.65	
...	...	...	...	...	...	...	
6840-RESVB	24	1	1	0	84.80	1990.50	
2234-XADUH	72	1	1	2	103.20	7362.90	
4801-JZAZL	11	0	0	1	29.60	346.45	
8361-LTMKD	4	1	0	0	74.40	306.60	
3186-AJIEK	66	1	2	3	105.65	6844.50	

7043 rows × 7 columns

In [3]:
```
#autoML to preprocess data
automl = setup(df, target='Churn', preprocess=True,  fold_shuffle=True)
```

	Description	Value
0	session_id	4536
1	Target	Churn
2	Target Type	Binary
3	Label Encoded	None
4	Original Data	(7043, 7)
5	Missing Values	False
6	Numeric Features	3
7	Categorical Features	3
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(4930, 11)
12	Transformed Test Set	(2113, 11)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	93b9
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None

	Description	Value
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	False
37	Combine Rare Levels	False
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Remove Perfect Collinearity	True
45	Clustering	False
46	Clustering Iteration	None
47	Polynomial Features	False
48	Polynomial Degree	None
49	Trignometry Features	False
50	Polynomial Threshold	None
51	Group Features	False
52	Feature Selection	False
53	Feature Selection Method	classic
54	Features Selection Threshold	None
55	Feature Interaction	False
56	Feature Ratio	False
57	Interaction Threshold	None
58	Fix Imbalance	False
59	Fix Imbalance Method	SMOTE

Find an ML algorithm that performs best on the data

In [4]:
```
#run autoML to find the best model
#Choice of Recall is discussed in the summary
best_model = compare_models(sort='Recall')
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
nb	Naive Bayes	0.6858	0.8071	0.8252	0.4495	0.5817	0.3637	0.4070	0.0120
qda	Quadratic Discriminant Analysis	0.5349	0.6166	0.7899	0.3344	0.4618	0.1601	0.2220	0.0150
lr	Logistic Regression	0.7978	0.8309	0.5131	0.6506	0.5730	0.4430	0.4488	0.6920
lda	Linear Discriminant Analysis	0.7901	0.8216	0.5123	0.6270	0.5631	0.4269	0.4312	0.0180
dt	Decision Tree Classifier	0.7410	0.6659	0.4985	0.5116	0.5045	0.3293	0.3297	0.0160
xgboost	Extreme Gradient Boosting	0.7840	0.8121	0.4924	0.6143	0.5459	0.4066	0.4113	0.2980
et	Extra Trees Classifier	0.7578	0.7698	0.4885	0.5477	0.5158	0.3552	0.3566	0.4550
ada	Ada Boost Classifier	0.7947	0.8343	0.4839	0.6516	0.5546	0.4252	0.4335	0.1560
rf	Random Forest Classifier	0.7720	0.7928	0.4808	0.5846	0.5271	0.3789	0.3823	0.5110
lightgbm	Light Gradient Boosting Machine	0.7805	0.8213	0.4786	0.6106	0.5354	0.3947	0.4004	0.0590
gbc	Gradient Boosting Classifier	0.7911	0.8320	0.4778	0.6417	0.5468	0.4151	0.4232	0.3230
svm	SVM - Linear Kernel	0.7146	0.0000	0.4724	0.5468	0.4614	0.2939	0.3089	0.0230
ridge	Ridge Classifier	0.7911	0.0000	0.4471	0.6557	0.5309	0.4027	0.4154	0.0100
knn	K Neighbors Classifier	0.7675	0.7349	0.4425	0.5790	0.5012	0.3533	0.3590	0.1020
dummy	Dummy Classifier	0.7355	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0090

In [5]:
```
#Highest scoring model on Recall
best_model
```

Out[5]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Save the model to disk

In [6]:
```
#Save to a pickle file
save_model(best_model, 'NB')
```

```
Transformation Pipeline and Model Successfully Saved
```

Out[6]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[],
                                       ml_usecase='classification',
                                       numerical_features=[], target='Churn',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='not_available',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None,
                                 numeric_strate...
                 ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
                 ('cluster_all', 'passthrough'),
                 ('dummy', Dummify(target='Churn')),
                 ('fix_perfect', Remove_100(target='Churn')),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  GaussianNB(priors=None, var_smoothing=1e-09)]],
          verbose=False),
 'NB.pkl')
```

In [7]:
```
#Open to test if it works
with open('NB_model.pk', 'rb') as f:
    loaded_model = pickle.load(f)
```

In [8]:
```
new_data = df.iloc[-3:-1].copy()
new_data.drop('Churn', axis=1, inplace=True)
```

In [9]:
```
loaded_NB_model = load_model('NB')
```

```
Transformation Pipeline and Model Successfully Loaded
```

In [10]:
```
#Check to see if it works
predict_model(loaded_NB_model, new_data)
```

Out[10]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Lab
4801-JZAZL	11	0	0	1	29.6	346.45	
8361-LTMKD	4	1	0	0	74.4	306.60	

Create a Python script/file/module with a function that takes a pandas dataframe as an input and returns the probability of churn for each row in the dataframe

- your Python file/function should print out the predictions for new data (new_churn_data.csv)

In [11]:
```
#Import code and display
from IPython.display import Code

Code('predict_churn.py')
```

Out[11]:
```
import pandas as pd
from pycaret.classification import predict_model, load_model

def load_data(filepath):
    """
    Loads churn data into a DataFrame from a string filepath.
    """
    df = pd.read_csv(filepath, index_col='customerID')
    return df


def make_predictions(df):
    """
    Uses the pycaret best model to make predictions on data in the df dat
aframe.
    """
    model = load_model('NB')
    predictions = predict_model(model, data=df)
    predictions.rename({'Label': 'Churn_prediction'}, axis=1, inplace=Tru
e)
    predictions['Churn_prediction'].replace({1: 'Churn', 0: 'No Churn'},
                                            inplace=True)
    return predictions['Churn_prediction']


if __name__ == "__main__":
    df = load_data('new_Churn_data.csv')
    predictions = make_predictions(df)
    print('predictions:')
    print(predictions)
```

In [12]:
```
#Test your Python module and function with the new data
#True values for the new data are [1, 0, 0, 1, 0]
%run predict_churn.py
```

```
Transformation Pipeline and Model Successfully Loaded
predictions:
customerID
9305-CKSKC       Churn
1452-KNGVK    No Churn
6723-OKKJM       Churn
7832-POPKP    No Churn
6348-TACGU       Churn
Name: Churn_prediction, dtype: object
```

Summary

Findings:

I imported the prepped data but chose to drop the monthly_charges_calculated feature (column) because it was both redundant to an existing column, and I did have one case where the prediction of new data gave an error due to predicting on this missing column. As a related note, I had found in week 4 that the models were typically performing better with this removed.
AutoML correctly preprocessed the data, with the label, 3 categorical data types (phone service, contract type, and payment type) and 3 numerical data types (tenure, monthly charges, and total charges)

The autoML best model gave several models with strength, but despite the LDA's Accuracy, I had discussed in the Deployment section of week 3's assigment that our goal should be to avoid the False Negative condition of missing churn that happens, and I stated:
Tweaks to the model should probably emphasize minimizing false negatives even when it moderately increases the false positives
I proposed maximizing the True Positive Rate, which is the same as Recall, and the Naive Bayes model performs far better than the others considered.
Looking back against the past several weeks, week 4 I only emphasized the accuracy, and some tweaks such as feature removal to the random forest model produced a slightly more accurate model than any of the ones this week. Week 3 weighed accuracy versus recall, and my logistic regression model explored threshold adjustment to achieve a balance. In that case, I had a TPR of 80% and accuracy over 74%, while this Naive Bayes model provides a TPR of 84% against an accuracy of 69%, which is below the no information rate. Both of these models seem to perform significantly better than others with respect to our stated goals.

The model was saved and then opened, and made a prediction against 2 sample lines of data.
Lastly, a python script was made to return the churn prediction for each customer. It was tested against a set of new churn data. These results are not ideal; however, this is too small of a sample size to come to a conclusion about the model's usefulness. One positive sign is that the model does seem to be predicting false positive results (churn prediction but did not churn), and this is what we'd expect with emphasis on maximixing the Recall/TPR.

In [ ]: