DS Automation Assignment

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; by default, it is accuracy but it could be AUC, precision, recall, etc. The week 3 FTE has some information on these different metrics.
- 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)
  - the true values for the new data are [1, 0, 0, 1, 0] if you're interested
- test your Python module and function with the new data, new_churn_data.csv
- write a short summary of the process and results at the end of this notebook
- upload this Jupyter Notebook and Python file to a Github repository, and turn in a link to the repository in the week 5 assignment dropbox

Optional challenges:

- return the probability of churn for each new prediction, and the percentile where that prediction is in the distribution of probability predictions from the training dataset (e.g. a high probability of churn like 0.78 might be at the 90th percentile)
- use other autoML packages, such as TPOT, H2O, MLBox, etc, and compare performance and features with pycaret
- create a class in your Python module to hold the functions that you created
- accept user input to specify a file using a tool such as Python's `input()` function, the `click` package for command-line arguments, or a GUI
- Use the unmodified churn data (new_unmodified_churn_data.csv) in your Python script. This will require adding the same preprocessing steps from week 2 since this data is like the original unmodified dataset from week 1.

In [1]:

```python
import pandas as pd
df = pd.read_csv('D:/prepared_churn_data.csv', index_col='customerID')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 7590-VHVEG to 3186-AJIEK
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   tenure          7043 non-null   int64  
 1   PhoneService    7043 non-null   int64  
 2   Contract        7043 non-null   int64  
 3   PaymentMethod   7043 non-null   int64  
 4   MonthlyCharges  7043 non-null   float64
 5   TotalCharges    7043 non-null   float64
 6   Churn           7043 non-null   int64  
dtypes: float64(2), int64(5)
memory usage: 440.2+ KB
```

In [2]:

```python
df.head()
```

Out[2]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Churn
7590-VHVEG	1	0	0	2	29.85	29.85	0
5575-GNVDE	34	1	1	3	56.95	1889.50	0
3668-QPYBK	2	1	0	3	53.85	108.15	1
7795-CFOCW	45	0	1	0	42.30	1840.75	0
9237-HQITU	2	1	0	2	70.70	151.65	1

Using PyCaret for Automated Machine Learning (AutoML)

In [3]:

```python
from pycaret.classification import *
```

In [4]:

```python
automl = setup(df, target='Churn')
```

	Description	Value
0	Session id	4142
1	Target	Churn
2	Target type	Binary
3	Original data shape	(7043, 7)
4	Transformed data shape	(7043, 7)
5	Transformed train set shape	(4930, 7)
6	Transformed test set shape	(2113, 7)
7	Numeric features	6
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	b10f

Comparing Different Machine Learning Algorithms and finding Best model

In [5]:

```python
# Compare different ML algorithms
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.7895	0.8383	0.4771	0.6421	0.5461	0.4130	0.4215	0.3080
ada	Ada Boost Classifier	0.7886	0.8354	0.4932	0.6349	0.5527	0.4175	0.4247	0.1400
lr	Logistic Regression	0.7878	0.8312	0.5084	0.6244	0.5592	0.4217	0.4262	0.9840
ridge	Ridge Classifier	0.7866	0.0000	0.4411	0.6454	0.5226	0.3916	0.4041	0.0230
lightgbm	Light Gradient Boosting Machine	0.7846	0.8289	0.4984	0.6173	0.5510	0.4115	0.4159	0.1230
lda	Linear Discriminant Analysis	0.7799	0.8160	0.4809	0.6085	0.5362	0.3948	0.4000	0.0220
xgboost	Extreme Gradient Boosting	0.7775	0.8177	0.5061	0.5953	0.5461	0.4003	0.4031	0.0800
rf	Random Forest Classifier	0.7759	0.8045	0.4770	0.5978	0.5303	0.3856	0.3900	0.3660
knn	K Neighbors Classifier	0.7637	0.7419	0.4296	0.5742	0.4901	0.3408	0.3476	0.0600
et	Extra Trees Classifier	0.7617	0.7843	0.4886	0.5587	0.5205	0.3631	0.3650	0.2780
qda	Quadratic Discriminant Analysis	0.7471	0.8194	0.7378	0.5170	0.6075	0.4295	0.4447	0.0210
svm	SVM - Linear Kernel	0.7373	0.0000	0.3936	0.6159	0.4271	0.2845	0.3241	0.0390
dummy	Dummy Classifier	0.7347	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0170
dt	Decision Tree Classifier	0.7243	0.6514	0.4847	0.4806	0.4826	0.2947	0.2948	0.0370
nb	Naive Bayes	0.7059	0.8016	0.7553	0.4674	0.5772	0.3707	0.3958	0.0210

In [6]:

```python
best_model
```

Out[6]:

```
GradientBoostingClassifier
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='log_loss', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, n_iter_no_change=None,
                           random_state=4142, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [7]:

```python
predict_model(best_model, df)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Gradient Boosting Classifier	0.8116	0.8644	0.4949	0.7072	0.5823	0.4655	0.4779

Out[7]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Churn	prediction_label	prediction_score
7590-VHVEG	1	0	0	2	29.850000	29.850000	0	1	0.5240
5575-GNVDE	34	1	1	3	56.950001	1889.500000	0	0	0.9379
3668-QPYBK	2	1	0	3	53.849998	108.150002	1	0	0.6468
7795-CFOCW	45	0	1	0	42.299999	1840.750000	0	0	0.9241
9237-HQITU	2	1	0	2	70.699997	151.649994	1	1	0.6219
...	...	...	...	...	...	...	...	...	...
6840-RESVB	24	1	1	3	84.800003	1990.500000	0	0	0.9072
2234-XADUH	72	1	1	1	103.199997	7362.899902	0	0	0.9141
4801-JZAZL	11	0	0	2	29.600000	346.450012	0	0	0.6705
8361-LTMKD	4	1	0	3	74.400002	306.600006	1	1	0.5577
3186-AJIEK	66	1	2	0	105.650002	6844.500000	0	0	0.9233

7043 rows × 9 columns

Saving and Loading the Trained Model

In [8]:

```python
# Save the best model
save_model(best_model, 'GradientBoostingClassifier')
```

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

Out[8]:

```
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['tenure', 'PhoneService',
                                              'Contract', 'PaymentMethod',
                                              'MonthlyCharges', 'TotalCharges'],
                                     transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               keep_empty_features=False,
                                                               missing_values=nan,
                                                               strategy='mean',
                                                               verbose='deprecated'))),
                 ('...
                                            criterion='friedman_mse', init=None,
                                            learning_rate=0.1, loss='log_loss',
                                            max_depth=3, max_features=None,
                                            max_leaf_nodes=None,
                                            min_impurity_decrease=0.0,
                                            min_samples_leaf=1,
                                            min_samples_split=2,
                                            min_weight_fraction_leaf=0.0,
                                            n_estimators=100,
                                            n_iter_no_change=None,
                                            random_state=4142, subsample=1.0,
                                            tol=0.0001, validation_fraction=0.1,
                                            verbose=0, warm_start=False))],
          verbose=False),
 'GradientBoostingClassifier.pkl')
```

In [9]:

```python
import pickle

with open('GradientBoostingClassifier.pk', 'wb') as f:
    pickle.dump(best_model, f)
```

In [10]:

```python
with open('GradientBoostingClassifier.pk', 'rb') as f:
    loaded_model = pickle.load(f)
```

In [11]:

```python
new_data = df.copy()
new_data.drop('Churn', axis=1, inplace=True)
loaded_model.predict(new_data)
```

Out[11]:

```
array([1, 0, 0, ..., 0, 1, 0], dtype=int8)
```

In [12]:

```python
loaded_lda = load_model('GradientBoostingClassifier')
```

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

In [13]:

```python
predict_model(loaded_lda, new_data)
```

Out[13]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	prediction_label	prediction_score
7590-VHVEG	1	0	0	2	29.850000	29.850000	1	0.5240
5575-GNVDE	34	1	1	3	56.950001	1889.500000	0	0.9379
3668-QPYBK	2	1	0	3	53.849998	108.150002	0	0.6468
7795-CFOCW	45	0	1	0	42.299999	1840.750000	0	0.9241
9237-HQITU	2	1	0	2	70.699997	151.649994	1	0.6219
...	...	...	...	...	...	...	...	...
6840-RESVB	24	1	1	3	84.800003	1990.500000	0	0.9072
2234-XADUH	72	1	1	1	103.199997	7362.899902	0	0.9141
4801-JZAZL	11	0	0	2	29.600000	346.450012	0	0.6705
8361-LTMKD	4	1	0	3	74.400002	306.600006	1	0.5577
3186-AJIEK	66	1	2	0	105.650002	6844.500000	0	0.9233

7043 rows × 8 columns

Testing the Best Python module and function with new data

In [16]:

```python
# from IPython.display import Code
from IPython.display import Code

Code('D:/predict_churn.py')
```

Out[16]:

```python
import pandas as pd
from pycaret.classification import predict_model, load_model

model = load_model('GradientBoostingClassifier')

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, threshold=0.7):
    """
    Uses the pycaret best model to make predictions on data in the df dataframe.
    Rounds up to 1 if greater than or equal to the threshold.
    """
    predictions = predict_model(model, data=df)
    predictions['Churn_prediction'] = (predictions['prediction_score'] >= threshold)
    predictions['Churn_prediction'].replace({True: '0', False: '1'}, inplace=True)
    drop_cols = predictions.columns.tolist()
    drop_cols.remove('Churn_prediction')
    return predictions.drop(drop_cols, axis=1)


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

In [17]:

```python
%run D:/predict_churn.py
```

```
Transformation Pipeline and Model Successfully Loaded
predictions:
           Churn_prediction
customerID                 
9305-CKSKC                1
1452-KNGVK                0
6723-OKKJM                0
7832-POPKP                1
6348-TACGU                0
```

Summary

A simple process for using PyCaret, an automated machine learning (AutoML) library, to find the best model for predicting customer churn. It starts by loading a dataset about customer churn and setting up PyCaret with this data.

PyCaret compares different machine learning models to see which one does the best job. It looks at things like accuracy, which tells us how often the model is correct, and AUC, which measures how well the model can tell the difference between customers who churn and those who don't. After comparing all the models, PyCaret finds that the Gradient Boosting Classifier is the best one. It's good at predicting churn, with an accuracy of 0.7895 and an AUC of 0.8383.

Once we've found the best model, we save it to our computer so we can use it later. Then, we write a Python script to load the saved model and make predictions on new data. This script is like a set of instructions that tells the computer what to do. It reads new information about customers, uses the saved model to guess whether they'll churn, and gives us the probability of churn for each customer.

To show how well our model works, we test it on some new data. We use the script to predict churn for these new customers, and it tells us the probability that each one will churn. This helps us see if our model is accurate and can be trusted to make predictions in the real world.

Overall, using PyCaret makes it easy to find the best model for predicting customer churn and deploy it in real-world situations. It takes care of a lot of the complicated stuff, so we can focus on understanding our data and making good decisions based on it.