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.

Load data

First, we are going to load our same prepared data from week 2 where everything has been converted to numbers. Many autoML packages can handle non-numeric data (they usually convert it to numeric with various methods).

In [98]:
```
import pandas as pd

df = pd.read_csv('prepped_churn_data.csv', index_col='customerID')
df
```

Out[98]:

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

7043 rows × 7 columns

AutoML with pycaret

Our next step is to use pycart for autoML. We will need to install the Python package with conda or pip: `conda install -c conda-forge pycaret -y` . Then we can import the functions we need:so i installed pycaret and imported the functions that i needed.

In [99]:
```
from pycaret.classification import setup, compare_models, predict_model, save
```

There are some instructions for how to use pycaret in their documentation (https://pycaret.org/guide/). Since this is a relatively new package at the moment (it was created in late 2019), there can be some bugs and minor issues with the software. However, it still works well

overall, especially for something so complex. Next, we can setup our autoML:

In [100]:
```
automl = setup(df, target='Churn',preprocess=False)
```

	Description	Value
0	session_id	5167
1	Target	Churn
2	Target Type	Binary
3	Label Encoded	0: 0, 1: 1
4	Original Data	(7043, 7)
5	Missing Values	False
6	Numeric Features	3
7	Categorical Features	3
8	Transformed Train Set	(4930, 6)
9	Transformed Test Set	(2113, 6)
10	Shuffle Train-Test	True
11	Stratify Train-Test	False
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	4d9b
19	Fix Imbalance	False
20	Fix Imbalance Method	SMOTE

This will ask us to check if the datatypes of the input data are correct. In this case, they seem fine. There are a huge number of parameters we can set that we can see in the docs (https://pycaret.org/classification/) or if we run `?setup` in a cell. For now, we are leaving everything else at the default. However, relating it to last week, we can see there is a feature_selection option we could set.

By default, it preprocesses data (converts categorical columns into numeric). We can see what the preprocessed data looks like from one of the elements in the automl object. It seems like the index of the object (6 for unmodified data and 14 for preprocessed here) may change sometimes (possibly a bug or pecularity with pycaret).

In [115]:
```
automl[5]
```

Next up, we simply run the autoML to find the best model:

In [102]:
```
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7905	0.8329	0.8874	0.8361	0.8607	0.4388	0.4434	0.0110
gbc	Gradient Boosting Classifier	0.7880	0.8350	0.8952	0.8286	0.8604	0.4223	0.4288	0.0810
ada	Ada Boost Classifier	0.7874	0.8362	0.8924	0.8296	0.8596	0.4231	0.4291	0.0930
lda	Linear Discriminant Analysis	0.7866	0.8219	0.8910	0.8295	0.8590	0.4220	0.4274	0.0050
ridge	Ridge Classifier	0.7856	0.0000	0.9113	0.8166	0.8612	0.3959	0.4085	0.0050
catboost	CatBoost Classifier	0.7828	0.8319	0.8885	0.8271	0.8565	0.4110	0.4164	0.7140
knn	K Neighbors Classifier	0.7649	0.7503	0.8802	0.8133	0.8453	0.3586	0.3643	0.0120
rf	Random Forest Classifier	0.7590	0.7954	0.8613	0.8183	0.8391	0.3602	0.3628	0.1040
et	Extra Trees Classifier	0.7509	0.7726	0.8418	0.8214	0.8312	0.3552	0.3565	0.0830
qda	Quadratic Discriminant Analysis	0.7491	0.8232	0.7515	0.8876	0.8137	0.4373	0.4520	0.0050
nb	Naive Bayes	0.7193	0.8071	0.7025	0.8896	0.7847	0.3967	0.4204	0.0050
dt	Decision Tree Classifier	0.7164	0.6485	0.7957	0.8120	0.8036	0.2933	0.2938	0.0050
svm	SVM - Linear Kernel	0.7049	0.0000	0.7927	0.8259	0.7665	0.2706	0.3190	0.0080
xgboost	Extreme Gradient Boosting	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0040
lightgbm	Light Gradient Boosting Machine	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0030

From the above table we can see that among various available models,logistic regression have the highest accuracy. We can see the boosting algorithms like catboost take the longest to run.To get xgboost working, we need to allow preprocessing (which converts categorical columns into numeric columns). Our best_model object now holds the highest-scoring model. We can also set an argument `sort` in `compare_models` to choose another metric as our scoring metric. By default, it uses accuracy (and we can see the table above is sorted by accuracy). We could set this to `sort='Precision'` to use precision (TP / (TP + FN)), for example.

In [103]:
```
best_model
```

Out[103]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=5167, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

It looks like our best model is Logistic Regression(LR), closely followed by some others. This changed when i re-ran this - there is some randomness built in that we are not fixing (e.g. for the cross-validation splits possibly), so the top model may be different each time this is run since the accuracy scores are so similar between models.

We can now use the model to make predictions.

In [104]:
```
df.iloc[-2:-1].shape
```

Out[104]:
```
(1, 7)
```

We selected the last row, but using the indexing `[-2:-1]` to make it a 2D array instead of 1D (which throws an error).

However, this only worked after `preprocess=False` in our setup function. Because i found the order of features was different.

A more robust way (in case we are using preprocessing with autoML) is to use pycaret's predict_model function.

In [105]:
```
predict_model(best_model, df.iloc[-2:-1])
```

Out[105]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	
8361-LTMKD	4	0	0	1	74.4	306.6	

We can see this creates a new column, 'Score', with the probability of class 1. It also created a 'Label' column with the predicted label, where it rounded up if score is >= 0.5 (greater than or equal to 0.5).

Saving and loading our model

Next, we want to save our trained model so we can use it in a Python file later. pycaret has a handy function for this, which saves the model as a pickle file:

In [106]:
```
save_model(best_model, 'LR')
```

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

Out[106]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=
[],
                                       id_columns=[],
                                       ml_usecase='classification',
                                       numerical_features=[], target='Chur
n',
                                       time_features=[])),
                 ['trained_model',
                  LogisticRegression(C=1.0, class_weight=None, dual=False,
                                     fit_intercept=True, intercept_scaling=
1,
                                     l1_ratio=None, max_iter=1000,
                                     multi_class='auto', n_jobs=None,
                                     penalty='l2', random_state=5167,
                                     solver='lbfgs', tol=0.0001, verbose=0,
                                     warm_start=False)]],
          verbose=False),
 'LR.pkl')
```

`pickle` is a built-in module in the Python standard library which allows for saving and loading of binary data. It's data that's been encoded (usually using hexidecimal encoding) to a file, and we can store any Python object as-is in a pickle file. Then we can load the data from the file and be right back where we left off. To do this with pickle, we would do:

In [107]:
```
import pickle

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

Here, we use the built-in `open` function to open a file with the name `LR_model.pk`, then open it for writing with `'w'` and in a binary format using `'b'`. We save that file object in the variable `f`. The `with` statement automatically closes the file after we exit the with statement, otherwise, we should call the function `close` from the file object `f`. Then we use pickle to save our data to the file. We could reload it like this:

In [108]:
```
with open('LR_model.pk', 'rb') as f:
    loaded_model = pickle.load(f)
```

In [109]:
```
new_data = df.iloc[-2:-1].copy()
new_data.drop('Churn', axis=1, inplace=True)
loaded_model.predict(new_data)
```

Out[109]:
```
array([0])
```

Loading it is almost the same, except we use `rb` for "read binary" and use pickle's load function.

Under the hood, pycaret is doing something similar, but we can use it with the save_model function as we saw above.

Once we have our saved pycaret model, we can test loading it and making predictions to make sure it works:

In [110]:
```
loaded_lda = load_model('LR')
```

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

In [111]:
```
predict_model(loaded_lda, new_data)
```

Out[111]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	I
8361-LTMKD	4	0	0	1	74.4	306.6	

We can now use this model in a Python file to take in new data and make a prediction. We will first need to compose a Python file. We can do this in many ways:

- Jupyter and Jupyter Lab
- VS Code
- Atom
- Notepad++
- Other text editors or IDEs (integrated development environments)

The benefit of using a code editor or IDE is that it will have lots of bells and whistles, like syntax highlighting, autocomplete, and many other things depending on the code editor or IDE. VS Code is one of the top-most used editors by data scientists and software developers.We can easily install VS Code through Anaconda Navigator or by visiting the VS Code website. VS Code is developed by Microsoft, and there is also an IDE Visual Studio Code.

The file we've created is show below:

In [2]:
```
from IPython.display import Code

Code('predict_churn.py')
```

Out[2]:
```
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 dataf
rame.
    """
    model = load_model('LR')
    predictions = predict_model(model, data=df)
    predictions.rename({'Label': 'Churn_prediction'}, axis=1, inplace=True)
    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)
```

We can test out running the file with the Jupyter "magic" command %run:

In [113]:
```
%run predict_churn.py
```

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

The true values are [1, 0, 0, 1, 0] so our model is working OK but not perfect. We have 1 false positives in the new data. However, this new data was synthesized based on existing data, so it is

a little random.

Saving our code to GitHub ¶

We installed github through github.com and created account and we saved our code to github.Once we have an account and GitHub installed, we can create a new repository, either through the GUI with File -> New repository or through the web interface. It's best to select the option 'Initialize this repository with a README' and 'Python' for the 'Git ignore' option. The Git ignore option creates a file that ignores common files that we don't need that are related to Python (like our Jupyter Notebook checkpoints folders). Lastly, it's not a bad idea to choose a license. The MIT license is a a very open and open-source license, although others are more restrictive like Apache. We can choose MIT here since we aren't worried about protecting intellectual property in this case. Once we've created the repository, we can open the folder by browsing there, with a hotkey through the GUI, or by clicking the button in the GUI to open the folder. We need to copy our work there, then write a short note in the 'summary' area, and hit 'commit to main'. The 'main' label is a branch of the Git repository - we can have several branches in parallel but 'main' is default. Then we can hit the 'push origin' button in the upper right to upload our data to the GitHub's cloud. Last, we simply need to put the link in a text file and turn that in for our week 5 assignment.

Summary

First we loaded our same prepared data from week 2 where everything has been converted to numbers.We used Pycart for autoML.We setup our autoML.After running autoML,we found Logistic Regression as a best model which carries highest accuracy.We saved our trained model so we can use it in a python file.We tested out running the file with the Jupyter"magic"command %run and we compared with the true values for the mew data which are almost similar.At the end,we saved our code to Github.