Machine Learning End-to-End with Logistic Regression

Today, I will be performing a Logistic Regression end-to-end analysis on the SyriaTel Customer Churn dataset from Kaggle.

First, we import our modules we'll be using for this exercise and then import our dataset.

In [41]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [42]:
```
#open up the file
telecom_df = pd.read_csv('data/bigml_59c28831336c6604c800002a.csv')

telecom_df
```

Out[42]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122
...	...	...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82

3333 rows × 21 columns

In [43]:
```
for column in telecom_df.columns:
    print('---- %s ---' % column)
```

```
    print(telecom_df[column].value_counts())
```

```
---- state ---
WV    106
MN     84
NY     83
AL     80
OH     78
OR     78
WI     78
VA     77
WY     77
CT     74
MI     73
VT     73
ID     73
TX     72
UT     72
IN     71
KS     70
MD     70
MT     68
NJ     68
NC     68
CO     66
WA     66
NV     66
MS     65
RI     65
MA     65
AZ     64
MO     63
FL     63
NM     62
ND     62
ME     62
OK     61
NE     61
DE     61
SD     60
SC     60
KY     59
IL     58
NH     56
AR     55
DC     54
GA     54
HI     53
TN     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64
---- account length ---
105    43
87     42
93     40
101    40
```

```
90     39
       ..
191     1
199     1
215     1
221     1
2       1
Name: account length, Length: 212, dtype: int64
---- area code ---
415    1655
510     840
408     838
Name: area code, dtype: int64
---- phone number ---
354-5764    1
332-9896    1
409-2917    1
417-9455    1
400-8375    1
           ..
337-9303    1
348-5567    1
377-1218    1
409-1244    1
402-5076    1
Name: phone number, Length: 3333, dtype: int64
---- international plan ---
no     3010
yes     323
Name: international plan, dtype: int64
---- voice mail plan ---
no     2411
yes     922
Name: voice mail plan, dtype: int64
---- number vmail messages ---
0     2411
31      60
29      53
28      51
33      46
27      44
30      44
24      42
26      41
32      41
25      37
23      36
36      34
35      32
22      32
39      30
37      29
34      29
21      28
38      25
20      22
19      19
40      16
42      15
17      14
```

```
41      13
16      13
43       9
15       9
18       7
44       7
14       7
45       6
12       6
46       4
13       4
47       3
8        2
48       2
50       2
9        2
11       2
49       1
10       1
4        1
51       1
Name: number vmail messages, dtype: int64
---- total day minutes ---
174.5    8
159.5    8
154.0    8
175.4    7
162.3    7
        ..
199.9    1
105.8    1
125.6    1
179.8    1
270.8    1
Name: total day minutes, Length: 1667, dtype: int64
---- total day calls ---
102    78
105    75
107    69
95     69
104    68
       ..
149     1
157     1
36      1
30      1
165     1
Name: total day calls, Length: 119, dtype: int64
---- total day charge ---
27.12    8
26.18    8
29.67    8
31.18    7
27.59    7
        ..
19.36    1
16.95    1
34.12    1
48.35    1
13.28    1
```

```
Name: total day charge, Length: 1667, dtype: int64
---- total eve minutes ---
169.9    9
230.9    7
209.4    7
201.0    7
220.6    7
        ..
335.0    1
258.9    1
134.7    1
318.8    1
317.2    1
Name: total eve minutes, Length: 1611, dtype: int64
---- total eve calls ---
105    80
94     79
108    71
97     70
102    70
       ..
45      1
49      1
145     1
153     1
0       1
Name: total eve calls, Length: 123, dtype: int64
---- total eve charge ---
14.25    11
16.12    11
15.90    10
18.62     9
14.44     9
         ..
12.64     1
13.83     1
11.39     1
28.03     1
20.53     1
Name: total eve charge, Length: 1440, dtype: int64
---- total night minutes ---
210.0    8
214.6    8
197.4    8
191.4    8
188.2    8
        ..
132.3    1
306.2    1
293.5    1
271.7    1
182.6    1
Name: total night minutes, Length: 1591, dtype: int64
---- total night calls ---
105    84
104    78
91     76
102    72
100    69
       ..
```

```
164     1
166     1
33      1
149     1
36      1
Name: total night calls, Length: 120, dtype: int64
---- total night charge ---
9.66     15
9.45     15
8.88     14
8.47     14
7.69     13
         ..
14.65     1
6.46      1
3.94      1
15.74     1
6.14      1
Name: total night charge, Length: 933, dtype: int64
---- total intl minutes ---
10.0    62
11.3    59
9.8     56
10.9    56
10.1    53
        ..
18.9     1
1.3      1
2.7      1
2.6      1
3.1      1
Name: total intl minutes, Length: 162, dtype: int64
---- total intl calls ---
3     668
4     619
2     489
5     472
6     336
7     218
1     160
8     116
9     109
10     50
11     28
0      18
12     15
13     14
15      7
14      6
18      3
16      2
19      1
17      1
20      1
Name: total intl calls, dtype: int64
---- total intl charge ---
2.70    62
3.05    59
2.65    56
2.94    56
```

```
2.73    53
        ..
0.68     1
4.83     1
0.84     1
0.30     1
5.40     1
Name: total intl charge, Length: 162, dtype: int64
---- customer service calls ---
1    1181
2     759
0     697
3     429
4     166
5      66
6      22
7       9
9       2
8       2
Name: customer service calls, dtype: int64
---- churn ---
False    2850
True      483
Name: churn, dtype: int64
```

Here, we check the values of each of the churn rate values. We are looking to determine the likelihood of someone "churning", or leaving as a customer.

In [44]:

```
print(telecom_df.churn.value_counts())
print()
print(telecom_df.churn.value_counts(normalize=True))
```

```
False    2850
True      483
Name: churn, dtype: int64

False    0.855086
True     0.144914
Name: churn, dtype: float64
```

Here, around 14% of customers end up leaving as customers.

Next, we'll designate our X and y variables and then perform our train-test split. We'll default our split to be a 75-25 split with a random_state = 42

In [45]:

```
# Here, we'll designate what our X and y are

X = telecom_df.drop('churn', axis=1)
y = telecom_df['churn']

# Then, we'll do a train-test split separate the data out between a training dataset an

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=
```

Preprocessing

Now that we have our train-test split. Let's start our preprocessing process.

For this example, we'll be implementing a one hot encoder to convert categorical variables to numeric and standard scaler to make sure all of our data is on the same standard scale.

Let's instantiate our one hot encoder first.

In [46]:
```
from sklearn.preprocessing import OneHotEncoder

# Create the encoder
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(X_train)

# Apply the encoder
X_train = ohe.transform(X_train)
X_test = ohe.transform(X_test)
```

Next, we'll import the StandarScaler module and instantiate it on our X_train and X_test data.

In [47]:
```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler(with_mean=False)
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)
```

And that completes our preprocessing. Now let's get into the modelling itself.

Modelling

Let's create for ourselves a baseline model.

In [51]:
```
# import the needed modules
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# instantiate our logistic regression instance
baseline_model = LogisticRegression(random_state=42)

# perform our cross validation

baseline_model_neg_log_loss_cv = cross_val_score(baseline_model, X_train_scaled, y_trai
baseline_model_neg_log_loss_cv

baseline_log_loss = -(baseline_model_neg_log_loss_cv.mean())
baseline_log_loss
```

Out[51]:
```
0.8387899236355116
```

Here, our log loss is ~0.84. This is the baseline we'll use to compare to the other models to see how much we've improved (or not!)

Modified Logistic Model

Here, we'll use another Logistic Regression model, but with different parameters. Let's try modifying the solver, penalty, and class weights.

In [57]:

```
modified_model = LogisticRegression(solver='saga', penalty='elasticnet', class_weight='


modified_model_neg_log_loss_cv = cross_val_score(modified_model, X_train_scaled, y_trai

modified_model_log_loss = -(modified_model_neg_log_loss_cv.mean())

modified_model_log_loss
```

Out[57]: 0.6156952083584286

Now, let's compare our baseline to our modified model.

In [58]:

```
print("Previous Model")
print("Baseline average:", baseline_log_loss)
print("Current Model")
print("Modified Model average:", modified_model_log_loss)
```

```
Previous Model
Baseline average: 0.8387899236355116
Current Model
Modified Model average: 0.6156952083584286
```

As we're dealing with the log loss function, we are looking for the lowest amount of log loss (lowest error), so we know that we have made some improvement from our baseline model to our modified model.

So, let's try this out on our test data now.

In [61]:

```
from sklearn.metrics import log_loss

modified_model.fit(X_train_scaled, y_train)
log_loss(y_test, modified_model.predict_proba(X_test_scaled))
```

Out[61]: 0.6585999836806087

Our model on the test data performed slightly worst than on the trained data. This is expected as trained data normally perform better than test data does.

Now, let's go and see our evaluation metrics.

In [76]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

print(f" Our accuracy score is {accuracy_score(y_test, modified_model.predict(X_test_sc
print(f" Our precision score is {precision_score(y_test, modified_model.predict(X_test_
print(f" Our recall score is {recall_score(y_test, modified_model.predict(X_test_scaled
```

```
 Our accuracy score is 0.8309352517985612
 Our precision score is 0.2777777777777778
 Our recall score is 0.08
```

And here we have it! We have our evaluation metrics now. What this means is that our model's accuracy is approximately 83%, meaning that, with this model, we can correctly identify the likelihood of someone "churning" around 83% of the time. However, our precision and recall rates are abymissal (~28% and ~8%, respectively). But this exercise was to show how the process unfolds. Next time, we can possibly try out different models and/or parameters to improve our score.