Churn Prediction Part 2

April 25, 2020

0.0.1 Telco Churn Prediction (Part 2 of 2)

In Part 2, we train several candidate models and choose the most suitable model to predict customer churn for Telco.

```
In [18]: import pandas as pd
         import numpy as np
         import seaborn as sns
         import matplotlib.pyplot as plt

         from sklearn.model_selection import train_test_split, GridSearchCV
         from sklearn.feature_selection import SelectKBest, chi2
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import confusion_matrix, classification_report
```

```
In [2]: data = pd.read_csv('Telco-Customer-Churn.csv')
```

The data is relatively clean. In the `TotalCharges` column, 11 records are an empty string. All of these records have a valid `MonthlyCharges` and `tenure` of 0. We can impute these values as 0.

```
In [3]: data['TotalCharges'] = data['TotalCharges'].apply(lambda x: 0 if x == ' ' else x)
```

0.0.2 Prepare the Data for Modeling

We start by one hot encoding categorical variables and randomly partitioning the data between training and testing. After partitioning the dataset, **we do not touch the test set except for evaluation purposes**.

In [4]:
```
categorical_cols = [col for col in data.columns if col not in ['customerID',
                                                               'MonthlyCharges',
                                                               'TotalCharges',
                                                               'tenure',
                                                               'Churn']]

# write a simple function for one hot encoding a categorical column
def onehot_column(df, categorical_col):
    dummy_columns = pd.get_dummies(df[categorical_col], prefix=categorical_col)
    return pd.concat([df.drop(columns=categorical_col), dummy_columns], axis=1)

for col in categorical_cols:
    data = onehot_column(data, col)
data.columns
```

Out[4]:
```
Index(['customerID', 'tenure', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'gender_Female', 'gender_Male', 'SeniorCitizen_0', 'SeniorCitizen_1',
       'Partner_No', 'Partner_Yes', 'Dependents_No', 'Dependents_Yes',
       'PhoneService_No', 'PhoneService_Yes', 'MultipleLines_No',
       'MultipleLines_No phone service', 'MultipleLines_Yes',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'OnlineSecurity_No',
       'OnlineSecurity_No internet service', 'OnlineSecurity_Yes',
       'OnlineBackup_No', 'OnlineBackup_No internet service',
       'OnlineBackup_Yes', 'DeviceProtection_No',
       'DeviceProtection_No internet service', 'DeviceProtection_Yes',
       'TechSupport_No', 'TechSupport_No internet service', 'TechSupport_Yes',
       'StreamingTV_No', 'StreamingTV_No internet service', 'StreamingTV_Yes',
       'StreamingMovies_No', 'StreamingMovies_No internet service',
       'StreamingMovies_Yes', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaperlessBilling_No', 'PaperlessBilling_Yes',
       'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

In [5]:
```
data.drop(columns='customerID', inplace=True)
y = data['Churn'].apply(lambda x: 0 if x == 'No' else 1)
X = data.drop(columns='Churn')

# partition training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=1)
```

0.0.3 Feature Selection

The visualizations we created in Part 1 told us a lot about which features are important. We can also use the Chi squared test to make a more informed decision.

Recall that the values can be interpreted as follows:

The P-value indicates the % of more "extreme" observations if the null hypothesis is true and if we repeat the same experiment many times. The null hypothesis here is that the feature has no effect on the target (i.e. we should not use it as a predictor). Low p-values suggest we should reject our null hypothesis - in other words, reject the claim that the feature has no effect.

```
In [6]: feature_selector = SelectKBest(chi2, k='all')
        feature_selector.fit(X_train, y_train)
        p_values = feature_selector.pvalues_
```

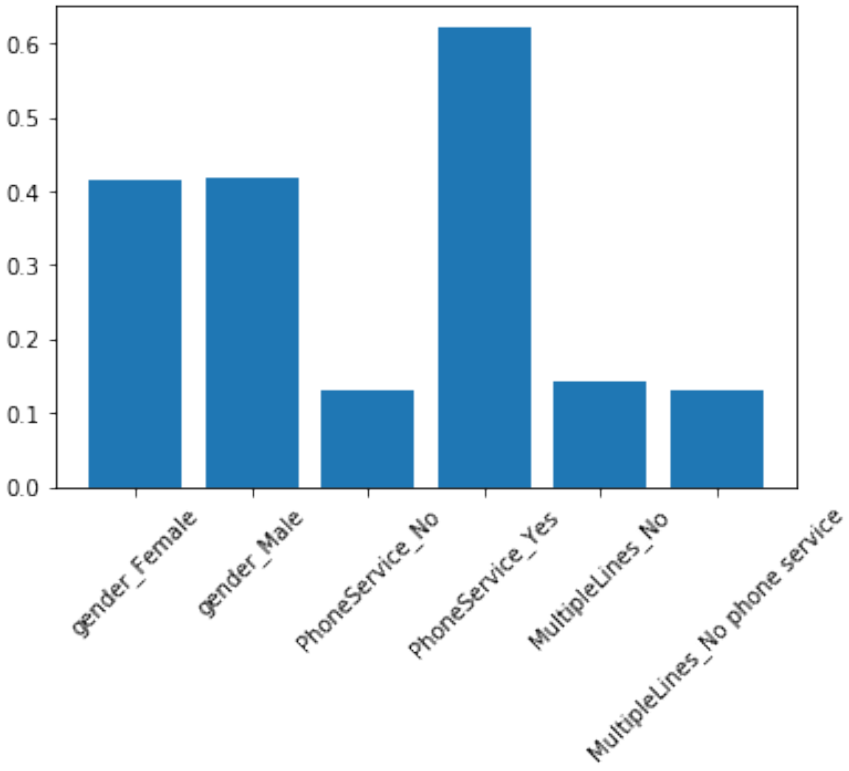
The p-values for most of the features are very low. Only `gender`, `PhoneService` and `MultipleLines` have p-values > 0.05, which means that we do not have enough evidence to reject the claim that these features have no effect.

The visualizations we created in Part 1 tell a similar a story. Based on this, we will be removing these 3 features from the model.

```
In [7]: p_above_thresh = [(i, p) for i, p in enumerate(p_values) if p > 0.05]

        plt.bar(height=[i[1] for i in p_above_thresh],
                x=X_train.columns[[i[0] for i in p_above_thresh]])
        plt.xticks(rotation=45)
```

```
Out[7]: ([0, 1, 2, 3, 4, 5], <a list of 6 Text xticklabel objects>)
```



We will also drop the `TotalCharges` since that can be approximated by `MonthlyCharges` and `tenure` to further reduce the dimensionality (the other two columns appear to be more reliable than this one, which had data issues).

```
In [8]: X_train = X_train.drop(columns=['gender_Female', 'gender_Male',
                                        'PhoneService_No', 'PhoneService_Yes',
                                        'MultipleLines_No', 'MultipleLines_Yes',
                                        'MultipleLines_No phone service',
                                        'TotalCharges'])
        X_test = X_test.drop(columns=['gender_Female', 'gender_Male',
                                      'PhoneService_No', 'PhoneService_Yes',
                                      'MultipleLines_No', 'MultipleLines_Yes',
                                      'MultipleLines_No phone service',
                                      'TotalCharges'])
```

0.0.4 Model Training

The sample solution will assume that Logistic Regression is the best model to use. In your solution, it is recommended to try several different classifiers and compare performance, model interpretability and other criteria that may be important prior to selecting the candidate algorithm.

We will also use `GridSearchCV` for hyperparameter tuning. Since we do not want to touch our test set until the very end, we will further partition our training set into training and validation.

```
In [9]: X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                         test_size=0.1, random_state=1)
```

We will use grid search to determine the optimal values for these hyperparameters: - whether we should use a L1 or L2 penalty - the C value - whether we should fit an intercept - class weight (whether we should put more emphasis on the Churn class as opposed to the non-Churn class)

Under the hood, `GridSearchCV` will be training and evaluting numerous models against the validation set to determine which hyperparameters yield the best performance.

```
In [10]: %%capture

         param_grid = {
             'penalty': ['l1', 'l2'],
             'C': [0.01, 0.1, 1, 10, 100],
             'fit_intercept': [True, False],
         }
         grid = GridSearchCV(estimator=LogisticRegression(),
                             param_grid=param_grid,
                             scoring='roc_auc',
                             verbose=1,
                             n_jobs=-1)
         result = grid.fit(X_train, y_train)
```

This gives us the best hyperparameter values.

```
In [11]: grid.best_params_

Out[11]: {'C': 1, 'fit_intercept': False, 'penalty': 'l1'}
```

With this information, we can formally train our model. We will be training the model on the training and validation set combined.

```
In [15]: # rest are default values
         logreg = LogisticRegression(penalty='l1', fit_intercept=False)
         logreg.fit(X_train.append(X_val), y_train.append(y_val))

Out[15]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=False,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l1', random_state=None, solver='warn',
                   tol=0.0001, verbose=0, warm_start=False)
```

0.0.5 Evaluating the Model

This is where we bring back our test set.

```
In [16]: y_pred = logreg.predict(X_test)
         print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      1061
           1       0.63      0.58      0.60       348

   micro avg       0.81      0.81      0.81      1409
   macro avg       0.75      0.73      0.74      1409
weighted avg       0.81      0.81      0.81      1409
```

```
In [19]: ax= plt.subplot()
         cm = confusion_matrix(y_test, y_pred)

         sns.heatmap(cm / cm.sum(axis=1)[:, np.newaxis], ax=ax, cmap="YlGnBu")
         ax.set_xlabel('Predicted labels')
         ax.set_ylabel('True labels')
         ax.set_title('Normalized Confusion Matrix')
```

```
Out[19]: Text(0.5, 1.0, 'Normalized Confusion Matrix')
```