Customer Churn Prediction

Predicting Churn/Retention for Banking Customers via Machine Learning

Ryan McReynolds
November 16, 2023
GitHub repo: https://github.com/ryanmcr17/customer-churn-prediction-project

Executive Summary

Project Objective: Predicting Customer Churn

The goal of this analysis is to build a machine learning model that effectively predicts which customers of the bank are likely to churn/'exit' as opposed to be retained as customers, based on several variables/details known about each customer today.

Such a model would help to minimize losses and maximize profits for the bank through targeted churn-mitigation efforts.

Results/Recommendation

The top model/option evaluated so far shows strong predictive accuracy, >86%, while also being relatively simple which should help minimize computing/resource costs.

It is recommended that the bank put this model into use immediately, potentially while continuing to iterate/optimize to improve performance even further.

Analysis Details

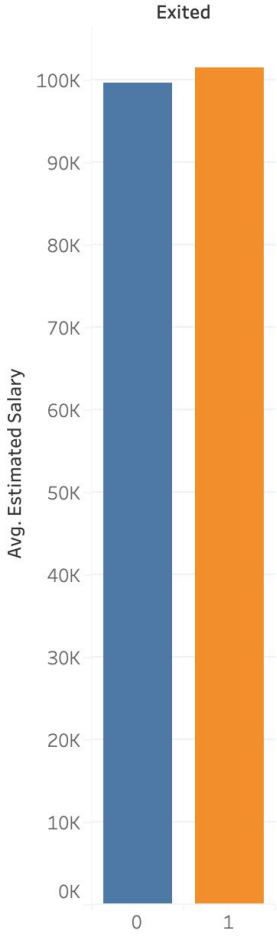
Repo/File Details

Key Files/Folders

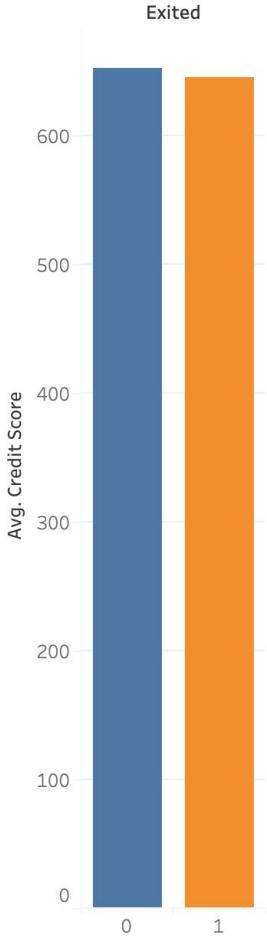
- all files in CustomerChurnPrediction/ folder within the repo
  - customer_churn_prediction.ipynb contains the bulk of the work, including ETL/preprocessing and all ML modeling approaches
  - bank_customer_data.csv is the downloaded file
  - top_customer_churn_prediction_model.h5 is the the recommended model saved as an HDF5 file
  - Model-Performance-Report.md contains the write-up/report and recommendations
  - Customer Churn Prediction_ Model Performance Comparison.xlsx contains the performance comparison of all models/options that were evaluated (also here in Google Sheets)
  - Tableau Public workbook is available here

Tableau Visualizations

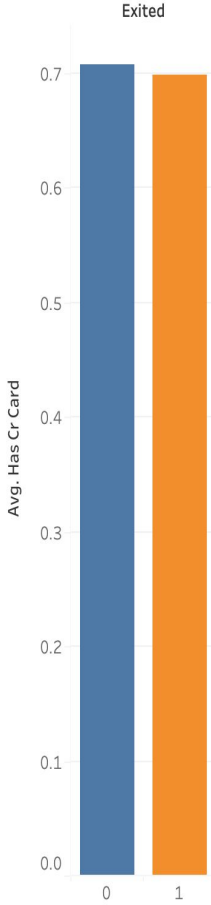
Average Salary by 'Exited' Outcome



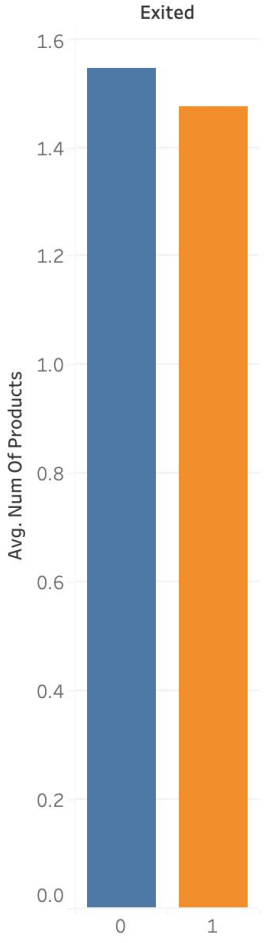
Average Credit Score by 'Exited' Outcome



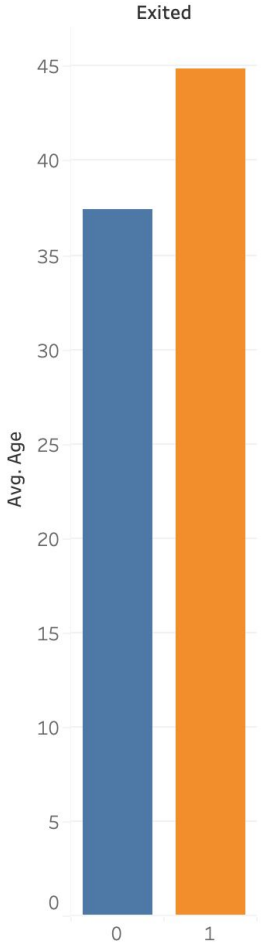
Average Likelihood of Having a Credit Card by 'Exited' Outcome



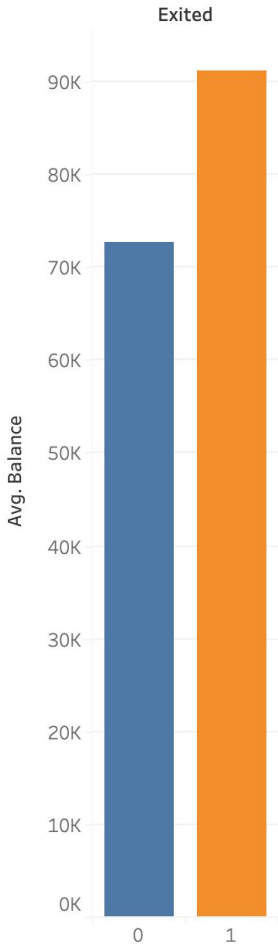
Average Number of Products by 'Exited' Outcome



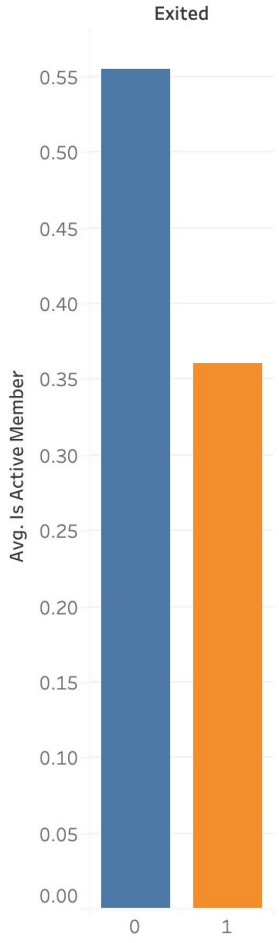
Average Age by 'Exited' Outcome



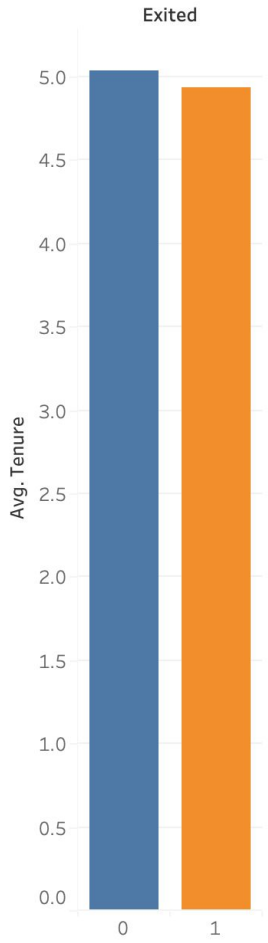
Average Balance by 'Exited' Outcome



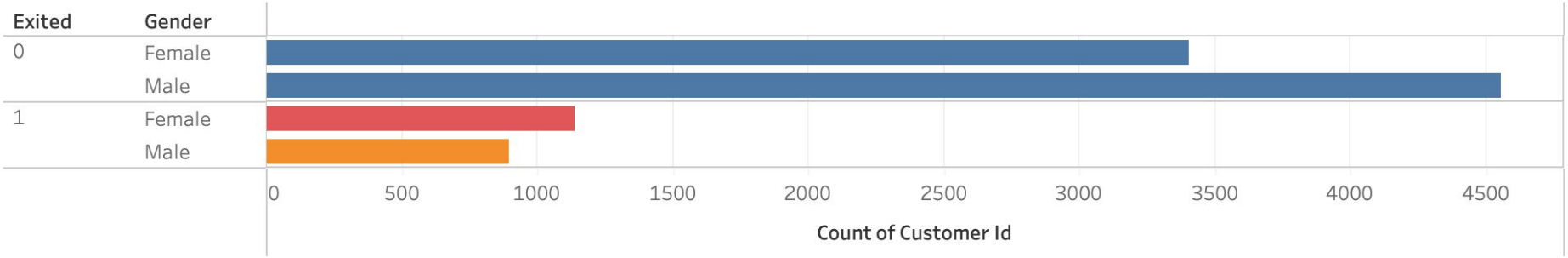
Average Likelihood of Being 'Active' by 'Exited' Outcome



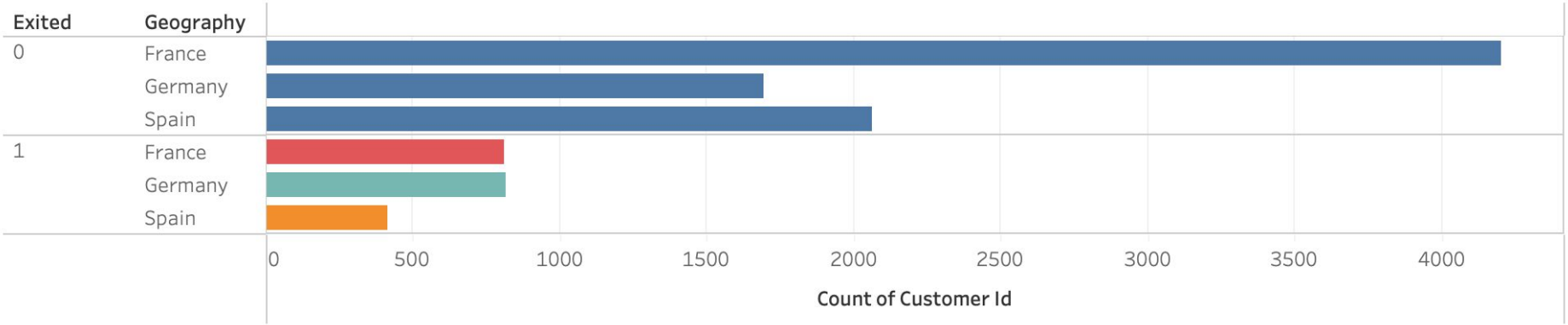
Average Tenure by 'Exited' Outcome



Count of 'Exited' Customers by Gender



Count of 'Exited' Customers by Geography/Country



Data Processing

Feature+Label Data

Dataset obtained from Kaggle including data on 10,000 bank customers

About Dataset

Content

- RowNumber—corresponds to the record (row) number and has no effect on the output.
- CustomerId—contains random values and has no effect on customer leaving the bank.
- Surname—the surname of a customer has no impact on their decision to leave the bank.
- CreditScore—can have an effect on customer churn, since a customer with a higher credit score is less likely to leave the bank.
- Geography—a customer's location can affect their decision to leave the bank.
- Gender—it's interesting to explore whether gender plays a role in a customer leaving the bank.
- Age—this is certainly relevant, since older customers are less likely to leave their bank than younger ones.
- Tenure—refers to the number of years that the customer has been a client of the bank. Normally, older clients are more loyal and less likely to leave a bank.
    - Balance—also a very good indicator of customer churn, as people with a higher balance in their accounts are less likely to leave the bank compared to those with lower balances.
    - NumOfProducts—refers to the number of products that a customer has purchased through the bank.
    - HasCrCard—denotes whether or not a customer has a credit card. This column is also relevant, since people with a credit card are less likely to leave the bank.
    - IsActiveMember—active customers are less likely to leave the bank.

Usability
10.00

License
CC0: Public Domain

Expected update frequency
Never

Tags

Finance | Banking | E-Commerce Services

Data Processing

Read in dataset, separated features/variables from labels/outcomes, dropped non-useful variables, and randomly sampled data for training vs testing

```
[3] # Read the CSV file from the Resources folder into a Pandas DataFrame
    customer_data_df = pd.read_csv("bank_customer_data.csv")

    # Review the DataFrame
    customer_data_df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3														
4														

```
[4] # Separate the y variable, the labels
    y = customer_data_df['Exited']

    # Separate the raw X variable data, the features, and drop the labels/outcomes column as well as additional columns not useful in predicting outcomes
    X_raw_data = customer_data_df.copy()
    X_raw_data.drop(['Exited','Surname','CustomerId','RowNumber'], axis=1, inplace=True)
```

```
[5] # Convert/encode columns with categorical feature values into numerical values, save results as final 'X' feature DataFrame
    X_encoded = pd.get_dummies(X_raw_data, dtype=int)

    X = X_encoded
```

```
[6] # Review the y variable Series
    print(y[:5])
```

ML Modeling

Logistic Regression Models

Logistic Regression model was able to be optimized to reach >71% accuracy, but we want better

```
[48] # Print the balanced_accuracy score of the 4th model
     print(balanced_accuracy_score(y_test4, predictions4))

     0.7118589743589743
```

```
[49] # Generate a confusion matrix for the 4th model
     cm4 = confusion_matrix(y_test4, predictions4)
     cm4_df = pd.DataFrame(
         cm4, index=['actual_retained', 'actual_churned'], columns=['predicted_retained', 'predicted_churned']
     )

     print(cm4_df)

                     predicted_retained  predicted_churned
     actual_retained                1452                528
     actual_churned                  161                359
```

```
[50] # Print the classification report for the 4th model
     target_names = ['retained_customer', 'churned_customer']

     cr4 = classification_report(y_test4, predictions4,
                                 target_names=target_names)

     print(cr4)

                       precision    recall  f1-score   support
```

Neural Network Models

Manually-optimized NN models were able to reach >80% accuracy, which is quite good

```
Epoch 9/17
235/235 [==============================] - 1s 3ms/step - loss: 0.4291 - accuracy: 0.8051
Epoch 10/17
235/235 [==============================] - 1s 3ms/step - loss: 0.4301 - accuracy: 0.8013
Epoch 11/17
235/235 [==============================] - 1s 3ms/step - loss: 0.4291 - accuracy: 0.8012
Epoch 12/17
235/235 [==============================] - 1s 4ms/step - loss: 0.4286 - accuracy: 0.8032
Epoch 13/17
235/235 [==============================] - 1s 4ms/step - loss: 0.4294 - accuracy: 0.8012
Epoch 14/17
235/235 [==============================] - 1s 4ms/step - loss: 0.4290 - accuracy: 0.8003
Epoch 15/17
235/235 [==============================] - 1s 4ms/step - loss: 0.4285 - accuracy: 0.8007
Epoch 16/17
235/235 [==============================] - 1s 4ms/step - loss: 0.4290 - accuracy: 0.7987
Epoch 17/17
235/235 [==============================] - 1s 3ms/step - loss: 0.4288 - accuracy: 0.7975
```

```
[68] # Evaluate the model using the test data
     model_loss, model_accuracy = nn_model5.evaluate(X_test3,y_test3,verbose=2)
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
79/79 - 0s - loss: 0.4242 - accuracy: 0.8068 - 281ms/epoch - 4ms/step
Loss: 0.42423149943351746, Accuracy: 0.8068000078201294
```

Neural Network Models

Using auto-optimization of model parameters via Keras Tuner improved accuracy even further to >86%

```
[56] best_hyper1 = tuner1.get_best_hyperparameters(1)[0]
     best_hyper1.values
```

```
{'activation': 'relu',
 'first_units': 9,
 'num_layers': 1,
 'units_0': 7,
 'units_1': 9,
 'units_2': 5,
 'units_3': 7,
 'units_4': 7,
 'units_5': 7,
 'tuner/epochs': 20,
 'tuner/initial_epoch': 7,
 'tuner/bracket': 2,
 'tuner/round': 2,
 'tuner/trial_id': '0013'}
```

```
[57] # Evaluate best model against full test data
     best_model1 = tuner1.get_best_models(1)[0]
     model_loss, model_accuracy = best_model1.evaluate(X_test3,y_test3,verbose=2)
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
79/79 - 1s - loss: 0.3440 - accuracy: 0.8640 - 517ms/epoch - 7ms/step
Loss: 0.34401363134384155, Accuracy: 0.8640000224113464
```

```
[58] # Export this model to HDF5 file
     best_model1.save('top_customer_churn_prediction_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file forma
  saving_api.save_model(
```

Conclusions and Next Steps

Summary of Options Evaluated

- Initial logistic regression models with manual optimization showed ok performance, with a maximum accuracy of just over 71%.
- Manually-optimized neural network models fit using TensorFlow and manual selection of model parameters were able to improve predictive accuracy significantly, with a few of the options showing close to 80% accuracy.
- Auto-optimization via Keras Tuner was able to improve performance even further, providing 2 models/options with 86+% accuracy.

Model Performance+Efficiency Comparison

Model Type	Feature/Input Data	Model Optimization Method	Layers/Complexity	Model Name in .ipynb File	Accuracy
Logistic Regression	Original/Raw	Manual	-	classifier1	51.35%
Logistic Regression	Oversampled	Manual	-	classifier2	67.39%
Logistic Regression	Scaled	Manual	-	classifier3	59.34%
Logistic Regression	Scaled+Oversampled	Manual	-	classifier4	71.19%
Neural Network	Scaled	Manual	2 layers	nn_model1	79.20%
Neural Network	Scaled+Oversampled	Manual	2 layers	nn_model2	71.36%
Neural Network	Scaled	Manual	4 layers	nn_model3	79.20%
Neural Network	Scaled+Oversampled	Manual	4 layers	nn_model4	79.20%
Neural Network	Scaled	Manual	5 layers	nn_model5	80.68%
Neural Network	**Scaled**	**Auto-Optimized**	**1 layer**	**nn_auto_opt1**	**86.40%**
Neural Network	Oversampled	Auto-Optimized	2 layers	nn_auto_opt2	86.68%
Neural Network	Scaled+Oversampled	Auto-Optimized	6 layers	nn_auto_opt3	81.84%

Recommendation for the Bank

- One of the auto-optimized deep learning models is both highly accurate and also relatively simple, providing >86% accuracy with only a single layer, so that is the recommended model for the bank to put into use.
    - One of the other options produced very slightly higher accuracy (by only 0.28%) but with a more complicated and therefore more resource-intensive model.
- At this level of accuracy it is recommended that the bank start leveraging this model immediately for predicting likely customer outcomes, in order to inform targeting for retention-focused outreach efforts (as well as other customer segmentation and targeting)

Next Steps / Future Analysis

- Given additional time/resources it would be ideal to try additional optimization options to see if accuracy can be improved even further:
  - Digging deeper into the distributions of values for each feature/variable in order to identify and remove potential outliers from the dataset.
  - Bringing additional customer data into the analysis, if available, to see if that can help improve the predictive power of the resulting models.
  - Testing additional deep learning model creation/optimization approaches.
- It would also make sense to go deeper into the predictive power of individual customer features/variables, which could help with model performance optimization as well as other efforts/initiatives the bank may want to consider around customer segmentation/targeting.

Thank You!