6006CEM Coursework

Chan Khai Shen

Table of Contents

1.0 Introduction

Customer churn is a situation where an existing customer decides not to continue a service or product from a company. Customer churn is an issue because companies will lose their revenue from customer churn. Customer churn analysis is interested in predicting the scenario where the customer shifts to another competitor (Lazarov & Capota, 2007). If a company is able to detect which customer has high possibility of churn, then the company is in an advantage position because actions can be taken to retain those customers.

The variables that can be used for customer churn prediction are behaviour, perception and demographic (Lazarov & Capota, 2007). Behaviour refers to how the customer uses the service or product, such as network usage and tenure. Perception refers to how the customer thinks on the service or product, such as satisfaction. Demographic refers to the customer personal details, such as age and gender.

The dataset[1] used in this project is related to customer churn in a telecommunication company. The variables in the dataset generally includes variables that is related to customer behaviour and demographic.

Predicting customer churn is a binary classification task because the output has only two classes, which is “churn” or “not churn”. So, it is possible customer churn by using machine learning algorithm. Researchers have used various algorithms, such as logistic regression, support vector machine, Naïve Bayes, neural network, decision tree and random forest, to predict customer churn (Lazarov & Capota, 2007) (Lalwani, Mishra, & Chadha, 2021) (Vafeiadis, Diamantaras, Sarigiannidis, & Chatzisavvas, 2015).

In this project, logistic regression and neural network will be compared. Table 1.1 lists out the accuracy achieved by some existing researches that used logistic regression and neural network. Based on Table 1.1, it is observed that neural network has higher accuracy compared to logistic regression.

Table 1.1 Existing research accuracy

Algorithm	Accuracy (%)	Source
Logistic regression	80.45	(Lalwani, Mishra, & Chadha, 2021)
	87.94	(Vafeiadis, Diamantaras, Sarigiannidis, & Chatzisavvas, 2015)
Neural network	94.06	(Vafeiadis, Diamantaras, Sarigiannidis, & Chatzisavvas, 2015)

[1] Dataset source: www.kaggle.com/datasets/dileep070/logisticregression-telecomcustomer-churmprediction

Besides comparing the metrics, this project will compare the accuracy of both algorithm by tuning the regularization techniques (L1, L2 and Elastic Net). This project will also compare the accuracy of neural network model by tuning the batch size.

2.0 Implementation

2.1 Data analysis

The dataset contains 20 variables after excluding the unique identifier, customer ID. Table 2.1 lists out the name and type of all variables, including the target variable, churn and 19 predictor variables.

Table 2.1 Variable name and type

	Variable name	**Type**
1	Gender	Category
2	Senior Citizen	Boolean
3	Partner	Category
4	Dependents	Category
5	Multiple Lines	Category
6	Internet Service	Category
7	Online Security	Category
8	Online Backup	Category
9	Device Protection	Category
10	Tech Support	Category
11	Streaming TV	Category
12	Streaming Movies	Category
13	Tenure	Numerical
14	Phone Service	Category
15	Contract	Category
16	Paperless Billing	Category
17	Payment Method	Category
18	Monthly Charges	Numerical
19	Total Charges	Numerical
20	Churn	Category

2.2 Preprocessing

<u>One-hot Encoding</u>

Machine learning model can only do calculation on numerical value, so all categorical values will be transformed into numerical values using Scikit Learn library's one-hot encoder function. The one-hot encoder will transform categorical feature with n categories into n binary features with value of 0 or 1 (Scikit Learn, n.d.). In this project, one-hot encoding is used instead of ordinal encoding because the most of the categorical fields in this dataset does not have a specific level. Figure 2.1 depicts the list of variables after dropping the original categorical variables. As shown in Figure 2.1, most of the fields is in float type, which is ready to be used.

```
df_combined_transformed.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 7042 entries, 0 to 7041
Data columns (total 47 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   customerID                               7042 non-null   object
 1   SeniorCitizen                            7042 non-null   int64
 2   tenure                                   7042 non-null   int64
 3   MonthlyCharges                           7042 non-null   float64
 4   TotalCharges                             7042 non-null   object
 5   Churn                                    7042 non-null   object
 6   gender_Female                            7042 non-null   float64
 7   gender_Male                              7042 non-null   float64
 8   Partner_Yes                              7042 non-null   float64
 9   Partner_No                               7042 non-null   float64
 10  Dependents_No                            7042 non-null   float64
 11  Dependents_Yes                           7042 non-null   float64
 12  MultipleLines_No phone service           7042 non-null   float64
 13  MultipleLines_No                         7042 non-null   float64
 14  MultipleLines_Yes                        7042 non-null   float64
 15  InternetService_DSL                      7042 non-null   float64
 16  InternetService_Fiber optic              7042 non-null   float64
 17  InternetService_No                       7042 non-null   float64
 18  OnlineSecurity_No                        7042 non-null   float64
 19  OnlineSecurity_Yes                       7042 non-null   float64
 20  OnlineSecurity_No internet service       7042 non-null   float64
 21  OnlineBackup_Yes                         7042 non-null   float64
 22  OnlineBackup_No                          7042 non-null   float64
 23  OnlineBackup_No internet service         7042 non-null   float64
 24  DeviceProtection_No                      7042 non-null   float64
 25  DeviceProtection_Yes                     7042 non-null   float64
 26  DeviceProtection_No internet service     7042 non-null   float64
 27  TechSupport_No                           7042 non-null   float64
 28  TechSupport_Yes                          7042 non-null   float64
 29  TechSupport_No internet service          7042 non-null   float64
 30  StreamingTV_No                           7042 non-null   float64
 31  StreamingTV_Yes                          7042 non-null   float64
 32  StreamingTV_No internet service          7042 non-null   float64
 33  StreamingMovies_No                       7042 non-null   float64
 34  StreamingMovies_Yes                      7042 non-null   float64
 35  StreamingMovies_No internet service      7042 non-null   float64
 36  PhoneService_No                          7042 non-null   float64
 37  PhoneService_Yes                         7042 non-null   float64
 38  Contract_Month-to-month                  7042 non-null   float64
 39  Contract_One year                        7042 non-null   float64
 40  Contract_Two year                        7042 non-null   float64
 41  PaperlessBilling_Yes                     7042 non-null   float64
 42  PaperlessBilling_No                      7042 non-null   float64
 43  PaymentMethod_Electronic check           7042 non-null   float64
 44  PaymentMethod_Mailed check               7042 non-null   float64
 45  PaymentMethod_Bank transfer (automatic)  7042 non-null   float64
 46  PaymentMethod_Credit card (automatic)    7042 non-null   float64
dtypes: float64(42), int64(2), object(3)
memory usage: 2.6+ MB
```

Figure 2.1 List of variables after one-hot encoding

Converting to numerical value and removing missing values

Because of TotalCharges column is string, so it will be converted to numerical value. After conversion, it is found that some NaN value occurred during the conversion. Figure 2.3 is non-null count after conversion. As shown in Figure 2.3, the non-null count of total charges column is 7031 while other colums are all 7042. The null count is very low (11), so the rows with NaN values are removed instead of replacing with mean.

```
In [15]: df_combined_transformed["TotalCharges"] = pd.to_numeric(df_combined_transformed["TotalCharges"], errors = "coerce")

In [16]: df_combined_transformed.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 7042 entries, 0 to 7041
Data columns (total 47 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   customerID                                7042 non-null   object
 1   SeniorCitizen                             7042 non-null   int64
 2   tenure                                    7042 non-null   int64
 3   MonthlyCharges                            7042 non-null   float64
 4   TotalCharges                              7031 non-null   float64
```

Figure 2.2 Column information after conversion

To reduce multicollinearity, for the variables in one same categorical column, one of the one-hot encoded variables has to be removed (Sethi, 2023).. Multicollinearity occurs when two predictor variables are highly correlated to each other (Bhandari, 2023). Figure 2.2 depicts the list of variables after removing the redundant columns.

```
In [37]: df_combined_transformed.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 7031 entries, 0 to 7041
Data columns (total 31 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   customerID                                7031 non-null   object
 1   SeniorCitizen                             7031 non-null   int64
 2   tenure                                    7031 non-null   int64
 3   TotalCharges                              7031 non-null   float64
 4   Churn                                     7031 non-null   object
 5   gender_Female                             7031 non-null   float64
 6   Partner_Yes                               7031 non-null   float64
 7   Dependents_Yes                            7031 non-null   float64
 8   MultipleLines_No phone service            7031 non-null   float64
 9   MultipleLines_No                          7031 non-null   float64
 10  InternetService_DSL                       7031 non-null   float64
 11  InternetService_Fiber optic               7031 non-null   float64
 12  OnlineSecurity_No                         7031 non-null   float64
 13  OnlineSecurity_No internet service        7031 non-null   float64
 14  OnlineBackup_No                           7031 non-null   float64
 15  OnlineBackup_No internet service          7031 non-null   float64
 16  DeviceProtection_No                       7031 non-null   float64
 17  DeviceProtection_No internet service      7031 non-null   float64
 18  TechSupport_No                            7031 non-null   float64
 19  TechSupport_No internet service           7031 non-null   float64
 20  StreamingTV_No                            7031 non-null   float64
 21  StreamingTV_No internet service           7031 non-null   float64
 22  StreamingMovies_No                        7031 non-null   float64
 23  StreamingMovies_No internet service       7031 non-null   float64
 24  PhoneService_Yes                          7031 non-null   float64
 25  Contract_Month-to-month                   7031 non-null   float64
 26  Contract_One year                         7031 non-null   float64
 27  PaperlessBilling_Yes                      7031 non-null   float64
 28  PaymentMethod_Mailed check                7031 non-null   float64
 29  PaymentMethod_Bank transfer (automatic)   7031 non-null   float64
 30  PaymentMethod_Credit card (automatic)     7031 non-null   float64
dtypes: float64(27), int64(2), object(2)
memory usage: 1.7+ MB
```

Figure 2.3 List of variables after removing redundant variables

Feature scaling

The feature scaling is done by using Min Max Scaler function in Scikit Learn library. Min Max Scaler will scale each predictor variable to a value between 0 and 1 (Scikit Learn, n.d.). If the Min max scaling is represented by the formula below where x is the original value and x_{scaled} is the scaled value.

$$x_{scaled} = \frac{x - minimum}{maximum - minimum}$$

Min max scaling is chosen instead of standard scaling because in this dataset there are many columns that has 1 or 0 binary value. Figure 2.4 depicts the result of feature scaling. The value of some columns, such as tenure and total charges are changed, whereas the binary columns does not change in its value.

```
In [39]: x = np.concatenate((data_combined_transformed[:, 1:4], data_combined_transformed[:, 5:]), axis = 1)
         y = data_combined_transformed[:, 4]
         print(x, "\n", x[0], "\n", y)

[[0 1 29.85 ... 0.0 1.0 0.0]
 [0 34 1889.5 ... 0.0 0.0 1.0]
 [0 2 108.15 ... 0.0 0.0 1.0]
 ...
 [0 72 7362.9 ... 1.0 0.0 0.0]
 [0 11 346.45 ... 0.0 1.0 0.0]
 [1 4 306.6 ... 0.0 0.0 1.0]]
 [0 1 29.85 1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 1.0
 0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0]
 ['No' 'No' 'Yes' ... 'No' 'No' 'Yes']

In [40]: x_scaled = MinMaxScaler().fit_transform(x)
         print(x_scaled)

[[0.         0.         0.0012751  ... 0.         1.         0.        ]
 [0.         0.46478873 0.21586661 ... 0.         0.         1.        ]
 [0.         0.01408451 0.01031041 ... 0.         0.         1.        ]
 ...
 [0.         1.         0.84746134 ... 1.         0.         0.        ]
 [0.         0.14084507 0.03780868 ... 0.         1.         0.        ]
 [1.         0.04225352 0.03321025 ... 0.         0.         1.        ]]
```

Figure 2.4 Feature scaling

2.3 Applied machine learning algorithms

Logistic regression and neural network are applied in this project. In this project, 20% of the dataset is split as the test data.

In logistic regression, the target variable ($\hat{y}$) is calculated using the below formula, where e is exponent.

$$\hat{y} = f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

In this project, logistic regression is applied by using Logistic Regression function from Scikit Learn library. The logistic regression function allows the tuning of penalty parameter, solver, max_iter and multi_class (Scikit Learn, n.d.). In this project, multi_class is set to "ovr" for binary classification and max_iter is set as 1000. The solver is set as "saga" to allow the tuning of all penalty parameters (more in 2.4).

Neural network used in this project is specified as three layers. The input layer has 29 neurons, which corresponds to the 29 predictor variables in this dataset. The hidden layer has 14 neurons, which is half of that of the input layer. The output layer has 1 neuron which corresponds with the 1 target variable, which is churn. The first two layers are using ReLU as the activation function, whereas the output layer uses sigmoid function.

In this project, the Sequential model from Keras library is used to implement neural network (Keras, n.d.). The tuning of regularization techniques on each layer (Keras, n.d.). In this project, regularization techniques and batch size will be tuned (more in 2.4). The epoch is fixed as 100.

2.4 Model tuning

For both logistic regression and neural network, model tuning is done by tuning the regularization techniques. The regularization techniques used are L1, L2 and elastic net. Regularization is used to overcome overfit issue. Overfit issue is a situation where the trained model is overly fits to the train data, until the noise of the train data is also captured (Data Quest, 2022).

The formula of L1 regularization is depicted below where m is the number of features, w is the coefficient and λ is the strength of regularization.

$$Cost\ function = Lost\ function + \lambda \sum_{j}^{m} |w_j|$$

Below is the formula of L2 regularization.

$$Cost\ function = Lost\ function + \frac{1}{2}\lambda \sum_{j}^{m} w_j^2$$

Below is the formula for elastic net regularization, which a combination of L1 and L2 where r is L1 ratio.

$$Cost\ function = Lost\ function + r\lambda \sum_{j}^{m} |w_j| + (1-r)\frac{1}{2}\lambda \sum_{j}^{m} w_j^2$$

Figure 2.5 is the result of tuning regularization in logistic regression. As shown in Figure 2.5, the accuracy of each method is nearly the same. The test score of no regularization is 0.8067, which is higher than that of all regularization techniques.

```
In [46]: logisticModels = []
logisticModels.append(LogisticRegression(penalty = None, max_iter = 1000, multi_class = 'ovr', random_state = 0, solver = 'saga')
logisticModels.append(LogisticRegression(penalty = 'l1', max_iter = 1000, multi_class = 'ovr', random_state = 0, solver = 'saga')
logisticModels.append(LogisticRegression(penalty = 'l2', max_iter = 1000, multi_class = 'ovr', random_state = 0, solver = 'saga',
logisticModels.append(LogisticRegression(penalty = 'elasticnet', max_iter = 1000, multi_class = 'ovr', random_state = 0, solver =
```

```
In [47]: df_logistic_regularization = pd.DataFrame(columns = ["Method", "Train score", "Test score"])

df_logistic_regularization["Method"] = ["None", "L1", "L2", "Elastic net"]
df_logistic_regularization["Train score"] = [round(logisticModel.score(x_train, y_train), 4) for logisticModel in logisticModels]
df_logistic_regularization["Test score"] = [round(logisticModel.score(x_test, y_test), 4) for logisticModel in logisticModels]

df_logistic_regularization.head(5)
```

Out[47]:

	Method	Train score	Test score
0	None	0.8057	0.8067
1	L1	0.8058	0.8053
2	L2	0.8049	0.8045
3	Elastic net	0.8053	0.8038

Figure 2.5 Tuning regularization techniques in logistic regression

Based on the result, it is suspected there is no overfit in the model. To prove this, a validation curve is plotted. Validation curve compares the test and train accuracies (Scikit Learn, n.d.). Figure 2.6 is the validation curve. As shown in Figure 2.6, as the iteration increases, the train score line and the test score line is very close to each other. So, it is clear that overfitting does not occur in this model.

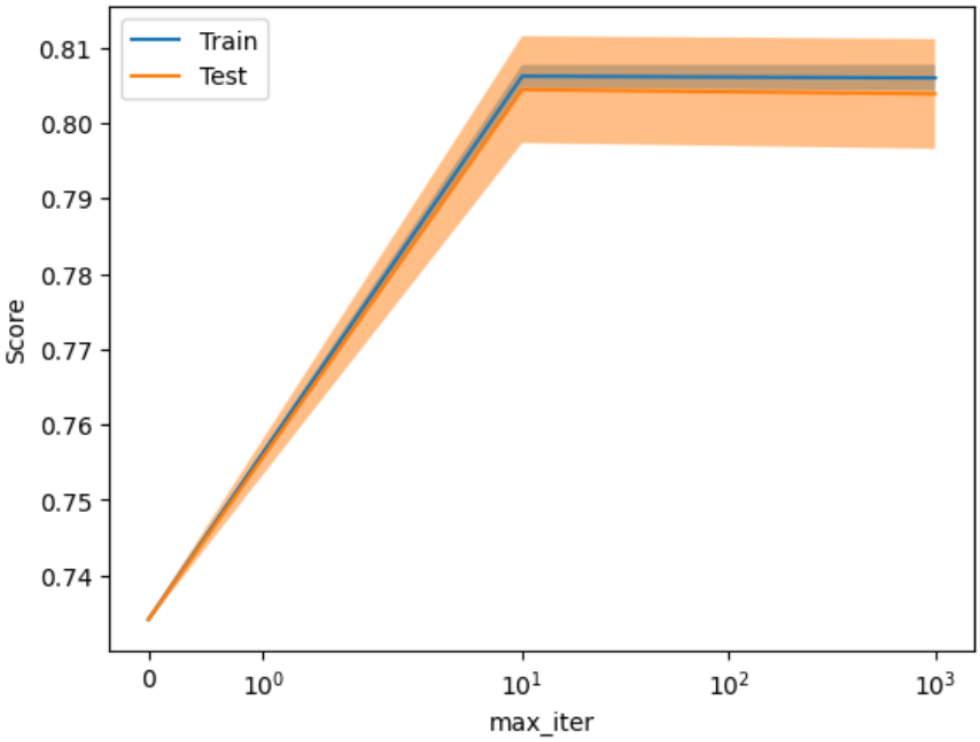

Figure 2.6 Logistic regression validation curve

Neural network

In neural network, the similar tuning is made. The model is trained with different regularization techniques. Figure 2.7 depicts the result. As shown in Figure 2.7, the test score of L2 regularization is the highest, achieving 0.8045. This shows an increase from the test score of no regularization, which is 0.8003. This proves that overfitting occurs in the model and L2 regularization fits this issue well.

```
In [38]: histories = []

         for neuralModel in neuralModels:
             neuralModel.compile(loss = "binary_crossentropy", metrics = ['accuracy'])
             histories.append(neuralModel.fit(x_scaled, y_numeric, validation_split = 0.2, epochs = 100, batch_size = 10))
```

```
563/563 [==============================] - 1s 880us/step - loss: 0.4250 - accuracy: 0.7962 - val_loss: 0.4446 - val_accurac
y: 0.8017
Epoch 3/100
563/563 [==============================] - 1s 887us/step - loss: 0.4209 - accuracy: 0.8012 - val_loss: 0.4398 - val_accurac
y: 0.7939
Epoch 4/100
563/563 [==============================] - 1s 898us/step - loss: 0.4189 - accuracy: 0.8016 - val_loss: 0.4338 - val_accurac
y: 0.7925
Epoch 5/100
563/563 [==============================] - 1s 895us/step - loss: 0.4159 - accuracy: 0.8064 - val_loss: 0.4413 - val_accurac
y: 0.7889
Epoch 6/100
563/563 [==============================] - 1s 907us/step - loss: 0.4148 - accuracy: 0.8046 - val_loss: 0.4348 - val_accurac
y: 0.8067
Epoch 7/100
563/563 [==============================] - 1s 929us/step - loss: 0.4140 - accuracy: 0.8012 - val_loss: 0.4389 - val_accurac
y: 0.7989
Epoch 8/100
563/563 [==============================] - 1s 926us/step - loss: 0.4131 - accuracy: 0.8048 - val_loss: 0.4363 - val_accurac
y: 0.8031
```

```
In [39]: df_neural_regularization = pd.DataFrame(columns = ["Method", "Train score", "Test score"])

         df_neural_regularization["Method"] = ["None", "L1", "L2", "Elastic net"]
         df_neural_regularization["Train score"] = [round(history.history["accuracy"][-1], 4) for history in histories]
         df_neural_regularization["Test score"] = [round(history.history["val_accuracy"][-1], 4) for history in histories]

         df_neural_regularization.head(5)
```

Out[39]:

	Method	Train score	Test score
0	None	0.8311	0.8003
1	L1	0.8042	0.8003
2	L2	0.8019	0.8045
3	Elastic net	0.8016	0.8038

Figure 2.7 Tuning regularization techniques in neural network

In neural network, the batch size is also tuned. The batch size tested are 10, 50 and 100. Figure 2.8 shows the result. As shown in Figure 2.8, among the three tested values, the accuracy score of batch size 10 is clearly lower, which is 0.8006. The accuracy score of batch size 50 and 100 is very close. In fact, in several runs, sometimes 50 is higher and sometimes 100 is higher.

```
In [41]: histories_batch = []

         histories_batch.append(neuralModel.fit(x_scaled, y_numeric, epochs = 100, batch_size = 10))
         histories_batch.append(neuralModel.fit(x_scaled, y_numeric, epochs = 100, batch_size = 50))
         histories_batch.append(neuralModel.fit(x_scaled, y_numeric, epochs = 100, batch_size = 100))
```

```
Epoch 1/100
704/704 [==============================] - 1s 722us/step - loss: 0.6447 - accuracy: 0.7820
Epoch 2/100
704/704 [==============================] - 1s 719us/step - loss: 0.4997 - accuracy: 0.7972
Epoch 3/100
704/704 [==============================] - 1s 718us/step - loss: 0.4728 - accuracy: 0.7993
Epoch 4/100
704/704 [==============================] - 1s 726us/step - loss: 0.4623 - accuracy: 0.7995
Epoch 5/100
704/704 [==============================] - 1s 727us/step - loss: 0.4554 - accuracy: 0.7979
Epoch 6/100
704/704 [==============================] - 1s 718us/step - loss: 0.4519 - accuracy: 0.8003
Epoch 7/100
704/704 [==============================] - 1s 732us/step - loss: 0.4505 - accuracy: 0.7975
Epoch 8/100
704/704 [==============================] - 0s 707us/step - loss: 0.4482 - accuracy: 0.8006
Epoch 9/100
704/704 [==============================] - 1s 714us/step - loss: 0.4469 - accuracy: 0.7996
Epoch 10/100
```

```
In [42]: df_neural_batch = pd.DataFrame(columns = ["Batch size", "Score"])

         df_neural_batch["Batch size"] = ["10", "50", "100"]
         df_neural_batch["Score"] = [round(history.history["accuracy"][-1], 4) for history in histories_batch]

         df_neural_batch.head(5)
```

Out[42]:

	Batch size	Score
0	10	0.8006
1	50	0.8044
2	100	0.8039

Figure 2.8 Tuning batch size in neural network

2.5 Evaluation

The performance of both models is evaluated based on confusion matrix, accuracy, precision, recall, F-1 score and loss. The metrics is computed using Classification Report function in Scikit Learn library. Classification Report function will report the precision, recall and F-score (Scikit Learn, n.d.). Below is the structure of confusion matrix used in this project as a binary classification task.

True negative	False positive
False negative	True positive

Below is the formula for precision.

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

Below is the formula for recall.

$$recall = \frac{true\ positive}{true\ positive + false\ negative}$$

Below is the formula for F-1 score.

$$F - 1\ score = \frac{2 \times precision \times recall}{precision + recall}$$

Below is the formula for loss where L is loss, y is actual value and $\hat{y}$ is predicted value.

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log \hat{y}$$

3.0 Result

3.1 Analysis and evaluation

Figure 3.1 is metrics of logistic regression model.

```
In [46]: print("Logistic Regression:")
         print(classification_report(y_test, y_pred_logistic))

         Logistic Regression:
                       precision    recall  f1-score   support

                   No       0.85      0.89      0.87      1039
                  Yes       0.65      0.56      0.60       368

             accuracy                           0.81      1407
            macro avg       0.75      0.73      0.74      1407
         weighted avg       0.80      0.81      0.80      1407
```

Figure 3.1 Logistic regression metrics

Figure 3.2 is metrics of neural network model.

```
In [57]: print("Neural Network:")
         print(classification_report(y_test_numeric, y_pred_neural))

         Neural Network:
                       precision    recall  f1-score   support

                    0       0.84      0.91      0.87      1039
                    1       0.67      0.52      0.59       368

             accuracy                           0.81      1407
            macro avg       0.76      0.72      0.73      1407
         weighted avg       0.80      0.81      0.80      1407
```

Figure 3.2 Neural network metrics

Based on the metrics, the first observation is the performance of both models are nearly the same. For example, the F-1 score of “not churn” of both models is the same, which is 0.87. Comparing logistic regression with neural network, the precision and recall scores are also very near. For example, the recall for “churn” in logistic regression is 0.56, while that of neural network is 0.59. The precision for “churn” in logistic regression is 0.67, while that of neural network is 0.65. So, based on the data, it can be said that, both models has nearly the same performance.

The second observation is the metrics of predicting “churn” is lower than that of predicting “not churn”. For example, in neural network, the recall of “not churn” is 0.91, while that of “churn” is 0.59. In logistic regression, it is similar, as the precision of “not churn” is 0.89, while that of “churn” is 0.56. So, it can be said that, for both models, the performance of predicting “not churn” is clearly better than predicting “churn”.

Figure 3.3 is confusion matrix of logistic regression.

```
[[929 110]
 [162 206]]
```

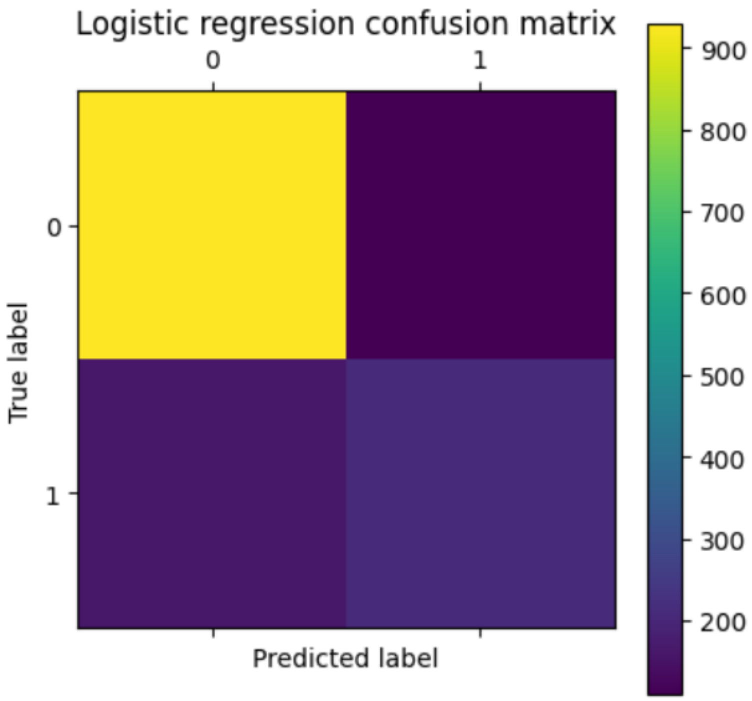

Figure 3.3 Logistic regression confusion matrix

Figure 3.4 is confusion matrix of neural network.

```
[[943  96]
 [175 193]]
```

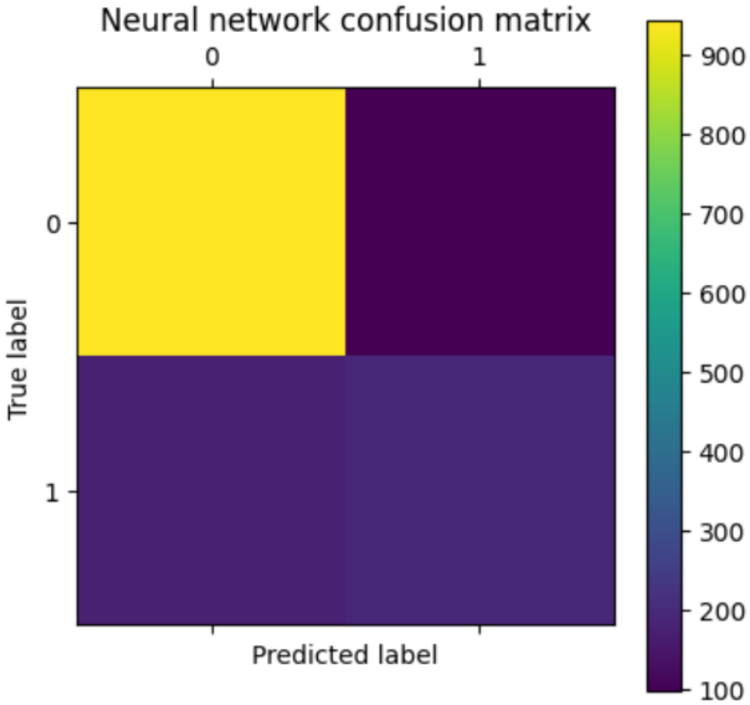

Figure 3.4 Neural network confusion matrix

Based on the confusion matrix, the performance of both models is nearly the same. For example, the true negative count of neural network is 943, while that of logistic regression is 929. This means that neural network is slightly better in determining “not churn” because the count is higher. But oppositely, the true positive count of logistic regression is 206, while that of neural network is 193. This means the opposite, because logistic regression is slightly better in determining “churn”. So, based on the data, it is difficult to differentiate which model has higher performance.

Figure 3.5 depicts the loss of both models. As shown in Figure 3.5, the loss of logistic regression is 0.4219, while that of neural network is 0.4243. This means the loss is actually very near for both models.

```
In [61]: loss_logistic = log_loss(y_test, y_pred_2_logistic)
         print(f'Logistic regression loss: {round(loss_logistic, 4)}')

         Logistic regression loss: 0.4219

In [62]: loss_and_metrics = neuralModel.evaluate(x_test, y_test_numeric)
         print(f"Neural network loss: {round(loss_and_metrics[0], 4)}")

         44/44 [==============================] - 0s 476us/step - loss: 0.4243 - accuracy: 0.8074
         Neural network loss: 0.4243
```

Figure 3.5 Comparing loss for both models

3.2 Conclusion

As conclusion, the results in this project have suggested that logistic regression and neural network has nearly the same performance in predicting customer churn. Although the reason is unknown, but this is actually not in line with other researches which find that neural network has better performance.

(Word count: 2079)

4.0 References

Bhandari, A. (2023, November 9). *Multicollinearity | Causes, Effects and Detection Using VIF*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/

Data Quest. (2022, October 11). *Regularization in Machine Learning*. Retrieved from Data Quest: https://www.dataquest.io/blog/regularization-in-machine-learning/

Keras. (n.d.). *Layer weight regularizers*. Retrieved from Keras: https://keras.io/api/layers/regularizers/

Keras. (n.d.). *The Sequential Model*. Retrieved from Keras: https://keras.io/guides/sequential_model/

Lalwani, P., Mishra, M. K., & Chadha, J. S. (2021). Customer churn prediction system: a machine learning.

Lazarov, V., & Capota, M. (2007). Churn Prediction.

Scikit Learn. (n.d.). *Encoding Categorical Features*. Retrieved from Scikit Learn: https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features

Scikit Learn. (n.d.). *Logistic Regression*. Retrieved from Scikit Learn: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Scikit Learn. (n.d.). *Model evaluation: quantifying the quality of predictions*. Retrieved from Scikit Learn: https://scikit-learn.org/0.15/modules/model_evaluation.html

Scikit Learn. (n.d.). *Standardization, or mean removal and variance scaling*. Retrieved from Scikit Learn: https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling

Scikit Learn. (n.d.). *Validation Curve*. Retrieved from Scikit Learn: https://scikit-learn.org/stable/modules/learning_curve.html#validation-curve

Sethi, A. (2023, June 15). *One Hot Encoding vs. Label Encoding using Scikit-Learn*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

Vafeiadis, T., Diamantaras, K., Sarigiannidis, G., & Chatzisavvas, K. (2015). A comparison of machine learning techniques for customer. *Simulation Modelling Practice and Theory 55*, 1-9.

5.0 Appendix B

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, ValidationCurveDisplay
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,
log_loss
from keras import Sequential
from keras.layers import Dense
import tensorflow as tf


df_churn = pd.read_csv('churn_data.csv')
df_customer = pd.read_csv('customer_data.csv')
df_internet = pd.read_csv('internet_data.csv')


df_customer_internet = pd.merge(df_customer, df_internet, how = 'inner', on = 'customerID')
df_combined = pd.merge(df_customer_internet, df_churn, how = 'inner', on = 'customerID')
df_combined.head(10)


df_combined.info()


# One hot encoding
for column in df_combined:
    print(column)
    print(df_combined[column].value_counts())
    print("\n")


columns = []
```

```
for column in ["gender", "Partner", "Dependents", "MultipleLines", "InternetService",
"OnlineSecurity", "OnlineBackup",
            "DeviceProtection", "TechSupport", "StreamingTV", "StreamingMovies",
"PhoneService", "Contract",
            "PaperlessBilling", "PaymentMethod"]:

   for category in df_combined[column].unique():
      columns.append(column + "_" + category)

print(columns)

data = np.array(df_combined)
print(data, "\n", data[0])

data_to_be_transformed = np.concatenate((data[:, 1:2], data[:, 3:13], data[:, 14:18]), axis =
1)
print(data_to_be_transformed, "\n", data_to_be_transformed[0])

data_transformed = OneHotEncoder().fit_transform(data_to_be_transformed).toarray()
print(data_transformed, "\n", data_transformed[0])

df_transformed = pd.DataFrame(data = data_transformed, columns = columns)
df_transformed.head()

df_combined_transformed = pd.concat([df_combined, df_transformed], axis = 1)
df_combined_transformed.head()

df_combined_transformed = df_combined_transformed.drop(columns = ["gender", "Partner",
"Dependents", "MultipleLines",
                                                   "InternetService", "OnlineSecurity",
"OnlineBackup",
                                                   "DeviceProtection", "TechSupport", "StreamingTV",
                                                   "StreamingMovies", "PhoneService", "Contract",
```

```
                                                  "PaperlessBilling", "PaymentMethod"])
df_combined_transformed.info()


df_combined_transformed["TotalCharges"] =
pd.to_numeric(df_combined_transformed["TotalCharges"], errors = "coerce")


df_combined_transformed.info()


df_combined_transformed.dropna(inplace = True)


df_combined_transformed.info()


df_combined_transformed.head()


df_inputs = df_combined_transformed.drop(columns = ["customerID", "Churn"])
df_vif = pd.DataFrame(columns = ["Column", "VIF"])


df_vif["Column"] = df_inputs.columns
df_vif["VIF"] = [round(variance_inflation_factor(df_inputs.values, i),4) for i in
range(df_inputs.shape[1])]


df_vif.head(50)


df_combined_transformed = df_combined_transformed.drop(columns = ["gender_Male",
"Partner_No", "Dependents_No",
                                                  "MultipleLines_Yes", "InternetService_No",
                                                  "OnlineSecurity_Yes", "OnlineBackup_Yes",
                                                  "DeviceProtection_Yes",
"TechSupport_Yes","StreamingTV_Yes",
                                                  "StreamingMovies_Yes", "PhoneService_No",
"Contract_Two year",
                                                  "PaperlessBilling_No",
"PaymentMethod_Electronic check",
                                                  "MonthlyCharges"
```

```
                                            ])
df_combined_transformed.head()


df_inputs = df_combined_transformed.drop(columns = ["customerID", "Churn"])
df_vif = pd.DataFrame(columns = ["Column", "VIF"])


df_vif["Column"] = df_inputs.columns
df_vif["VIF"] = [round(variance_inflation_factor(df_inputs.values, i), 4) for i in
range(df_inputs.shape[1])]


df_vif.head(50)


df_combined_transformed.info()


# Start of feature scaling
data_combined_transformed = np.array(df_combined_transformed)
print(data_combined_transformed)


x = np.concatenate((data_combined_transformed[:, 1:4], data_combined_transformed[:, 5:]),
axis = 1)
y = data_combined_transformed[:, 4]
print(x, "\n", x[0], "\n", y)


x_scaled = MinMaxScaler().fit_transform(x)
print(x_scaled)


# Start of logistic regression
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size = 0.2, random_state =
0)


logisticModels = []
logisticModels.append(LogisticRegression(penalty = None, max_iter = 1000, multi_class =
'ovr', random_state = 0, solver = 'saga').fit(x_train, y_train))
```

```
logisticModels.append(LogisticRegression(penalty = 'l1', max_iter = 1000, multi_class = 'ovr', random_state = 0, solver = 'saga').fit(x_train, y_train))

logisticModels.append(LogisticRegression(penalty = 'l2', max_iter = 1000, multi_class = 'ovr', random_state = 0, solver = 'saga', ).fit(x_train, y_train))

logisticModels.append(LogisticRegression(penalty = 'elasticnet', max_iter = 1000, multi_class = 'ovr', random_state = 0, solver = 'saga', l1_ratio = 0.5).fit(x_train, y_train))


df_logistic_regularization = pd.DataFrame(columns = ["Method", "Train score", "Test score"])


df_logistic_regularization["Method"] = ["None", "L1", "L2", "Elastic net"]

df_logistic_regularization["Train score"] = [round(logisticModel.score(x_train, y_train), 4) for logisticModel in logisticModels]

df_logistic_regularization["Test score"] = [round(logisticModel.score(x_test, y_test), 4) for logisticModel in logisticModels]


df_logistic_regularization.head(5)


for logisticModel in logisticModels:
    print(logisticModel.coef_, logisticModel.intercept_)
    print("\n")


    print(round(np.average(logisticModel.coef_), 4), round(logisticModel.intercept_[0], 4))
    print("\n")


ValidationCurveDisplay.from_estimator(LogisticRegression(penalty = None, multi_class = 'ovr', solver = 'saga'), x_scaled, y, param_name = "max_iter", param_range = (0, 10, 1000))


LogisticModel = LogisticRegression(penalty = None, max_iter = 1000, multi_class = 'ovr', solver = 'saga').fit(x_train, y_train)


# Start of neural network
df_y_numeric = df_combined_transformed["Churn"].replace(['No', 'Yes'], [0, 1])
y_numeric = np.array(df_y_numeric)
print(y_numeric)
```

```
x_train, x_test, y_train_numeric, y_test_numeric = train_test_split(x_scaled, y_numeric,
test_size = 0.2, random_state = 0)


neuralModels = []


neuralModels.append(Sequential([Dense(units = 29, activation = "relu"),
                                Dense(units = 14, activation = "relu"),
                                Dense(units = 1, activation = "sigmoid")
                               ])
                   )


neuralModels.append(Sequential([Dense(units = 29, activation = "relu", kernel_regularizer =
"l1"),
                                Dense(units = 14, activation = "relu", kernel_regularizer = "l1"),
                                Dense(units = 1, activation = "sigmoid")
                               ])
                   )


neuralModels.append(Sequential([Dense(units = 29, activation = "relu", kernel_regularizer =
"l2"),
                                Dense(units = 14, activation = "relu", kernel_regularizer = "l2"),
                                Dense(units = 1, activation = "sigmoid")
                               ])
                   )


neuralModels.append(Sequential([Dense(units = 29, activation = "relu", kernel_regularizer =
"l1_l2"),
                                Dense(units = 14, activation = "relu", kernel_regularizer = "l1_l2"),
                                Dense(units = 1, activation = "sigmoid")
                               ])
                   )


histories = []
```

```
for neuralModel in neuralModels:
    neuralModel.compile(loss = "binary_crossentropy", metrics = ['accuracy'])
    histories.append(neuralModel.fit(x_scaled, y_numeric, validation_split = 0.2, epochs = 100, batch_size = 10))

df_neural_regularization = pd.DataFrame(columns = ["Method", "Train score", "Test score"])

df_neural_regularization["Method"] = ["None", "L1", "L2", "Elastic net"]
df_neural_regularization["Train score"] = [round(history.history["accuracy"][-1], 4) for history in histories]
df_neural_regularization["Test score"] = [round(history.history["val_accuracy"][-1], 4) for history in histories]

df_neural_regularization.head(5)

neuralModel = Sequential([Dense(units = 29, activation = "relu", kernel_regularizer = "l2"),
                          Dense(units = 14, activation = "relu", kernel_regularizer = "l2"),
                          Dense(units = 1, activation = "sigmoid")
                         ])

neuralModel.compile(loss = "binary_crossentropy", metrics = ['accuracy'])

histories_batch = []

histories_batch.append(neuralModel.fit(x_scaled, y_numeric, epochs = 100, batch_size = 10))
histories_batch.append(neuralModel.fit(x_scaled, y_numeric, epochs = 100, batch_size = 50))
histories_batch.append(neuralModel.fit(x_scaled, y_numeric, epochs = 100, batch_size = 100))

df_neural_batch = pd.DataFrame(columns = ["Batch size", "Score"])

df_neural_batch["Batch size"] = ["10", "50", "100"]
```

```
df_neural_batch["Score"] = [round(history.history["accuracy"][-1], 4) for history in
histories_batch]


df_neural_batch.head(5)


history = neuralModel.fit(x_scaled, y_numeric, validation_split = 0.2, epochs = 100,
batch_size = 50)


print(f'Accuracy: {round(history.history["accuracy"][-1], 4)}')


# Start of evaluation of noth model
y_pred_logistic = LogisticModel.predict(x_test)
print(y_pred_logistic)


print("Logistic Regression:")
print(classification_report(y_test, y_pred_logistic))


y_pred_neural = neuralModel.predict(x_test)
print(y_pred_neural)


y_pred_neural = tf.squeeze(y_pred_neural)
print(y_pred_neural)


print("Neural Network:")
print(classification_report(y_test_numeric, y_pred_neural))


cm_logistic = confusion_matrix(y_test, y_pred_logistic)
print(cm_logistic)


plt.matshow(cm_logistic)
plt.title('Logistic regression confusion matrix')
plt.colorbar()
plt.ylabel('True label')
```

```
plt.xlabel('Predicted label')
plt.show()

cm_neural = confusion_matrix(y_test_numeric, y_pred_neural)
print(cm_neural)

plt.matshow(cm_neural)
plt.title('Neural network confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

y_pred_2_logistic = LogisticModel.predict_proba(x_test)
print(y_pred_2_logistic)

loss_logistic = log_loss(y_test, y_pred_2_logistic)
print(f'Logistic regression loss: {round(loss_logistic, 4)}')

loss_and_metrics = neuralModel.evaluate(x_test, y_test_numeric)
print(f"Neural network loss: {round(loss_and_metrics[0], 4)}")
```

Git Hub Link: https://github.com/ChanKhaiShen/Customer-Churn-Prediction.git

Dataset source: https://www.kaggle.com/datasets/dileep070/logisticregression-telecomcustomer-churmprediction