Name of The Project: Customer Churn Rate Prediction

Problem Description:

Churn rate means the rate of customers that stopped doing business with a company after using the product for some time. The terminology is used in many fields from telecommunication operators or banks to mobile apps or even in in-app purchases. And the word has different meanings in every field:

- For banks, telecommunication companies and/or other service providers; churned users stop using the company. For example, an Akbank client switched to Yapı Kredi is a churned customer; a user not using the bank anymore is a churned user; a Superonline customer moving away to the countryside without internet is a churned customer.
- For applications people deleting their account, uninstalling the app or not using the app even though they have it (not -logging in) are churned customers.
- For applications with in-app purchases or subscriptions. People canceling the purchase, refunding them or not doing it anymore are all churn examples.

It can also be explained with a visual as below:

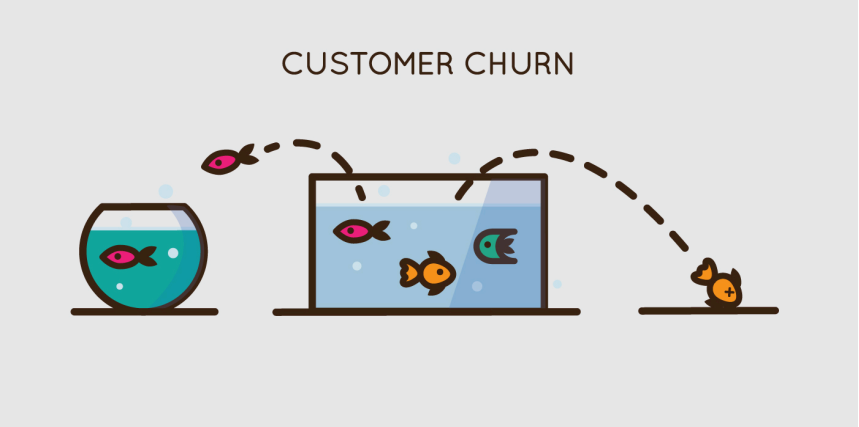


Figure 1: Churn Rate Visualized [5]

Fish that transfer to another aquarium symbolize the users using a competitor's product or the one that is outside the aquarium (dead-user) that symbolizes the user that stops using the product are all churn users.

Companies try to reduce the user churn ratio because it is directly proportional to their profit with the legitimate assumption that companies don't sell products that they don't make profit off of. Therefore, the churn rate prediction data is very crucial in decision-making for a company, because if the company predicts a user will go churn they may offer special discounts to keep them in contact and still profit off of them. Furthermore, telecommunication companies like Turkcell use this information in their "Salla Kazan" algorithm to keep their users satisfied and prevent high churn rates. Contrarily, if the customer is very unlikely to churn, the company may try to rip more from that user by making him/her buy more products or even scam them before they leave, making the user that decides to churn, leave with an unsatisfactory experience but increase the gain margin. A company that is considered "successful" in the user permanency field has a monthly churn rate around 0.5%. (the good churn rate – translates to 0.42 – 0.58% monthly churn) [1].

The churn rate is calculated by dividing the "left customers" by the sum of "available and new" customers. That can also be seen below:

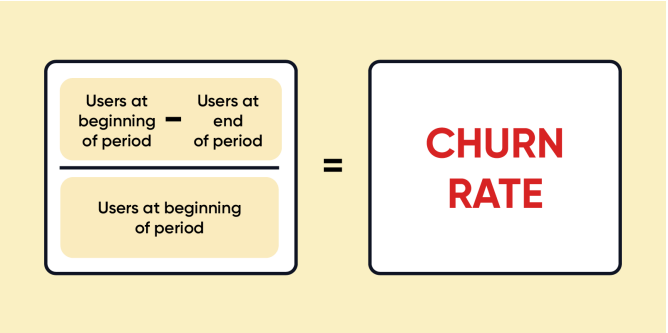


Figure 2: Churn Rate Calculation [6]

Our project will determine if a user will churn in an arbitrary timeframe and also calculate the churn rate of the given dataset that is explained in the next section of the report.

Dataset Description:

In the project, we plan on using a dataset obtained from users of telecommunication companies. The dataset has 4250 lines of training data and 750 lines of test data, training data has 20 columns that have 19 features and a class. The current dataset has the below layout (also see ref. 2):

- feature 1: "state", string. Residency state of the user
- feature 2: "account_length", integer. User's past subscription period length
- feature 3: "area_code", string. In the form "area_code_AAA" where AAA is 3 digit area code.
- feature 4: "international_plan", bool. The customer has an international plan.
- feature 5: "voice_mail_plan", bool. The customer has a voicemail plan.
- feature 6: "number_vmail_messages", integer. Number of voice-mail messages.
- feature 7: "total_day_minutes", integer. Total minutes of the day spent in calls.
- feature 8: "total_day_calls", integer. Total number of calls during the day.
- feature 9: "total_day_charge", integer. Total charge due to calls during the day.
- feature 10: "total_eve_minutes", integer. Total minutes of evening spent in calls.
- feature 11: "total_eve_calls", integer. Total number of calls during the evenings.
- feature 12: "total_eve_charge", integer. Total charge due to calls during the evening.
- feature 13: "total_night_minutes", integer. Total minutes of night spent in calls.
- feature 14: "total_night_calls", integer. Total number of calls during the night.
- feature 15: "total_night_charge", integer. Total charge due to calls during the night.
- feature 16: "total_intl_minutes", integer. Total minutes of international calls.
- feature 17: "total_intl_calls", integer. Total number of international calls.
- feature 18: "total_intl_charge", integer. Total charge due to international calls
- feature 19: "number_customer_service_calls", integer. Number of calls to customer service.
- class: "churn", (yes/no). Customer churn - target variable.

Generally, the statistics are all about calls in the dataset spreaded out to different times of the day or occasions like international calls. The dataset assumes the calls are directly related to the telecommunication usage of the user and hence can be used for predicting the future behavior.
Also you can see a sample from the dataset below visually:

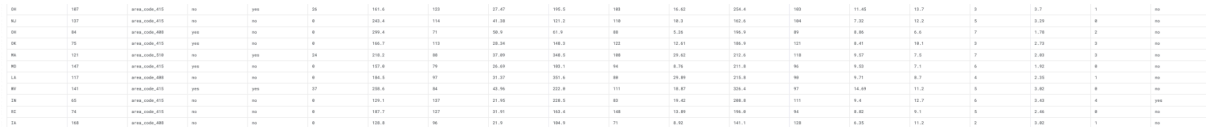


Figure 3: Sample From Dataset [2]

The test data has no "class" column thus we cannot test on it. Therefore, the training set will be splitted to 90% training + validation, 10% test. Then the 90% will be splitted to 85% training and 15% validation in itself. The dataset has 86% class = no and 14% class = yes data. All the data are from real-world users thus it is crucial for achieving high accuracy in real life scenarios.

Machine Learning Models Used:

Logistic Regression:

Logistic regression is a supervised learning model commonly used for classification problems. In this project, it is used for the binary classification of "churn" or "no churn" given 19 features (described in the previous section). The sigmoid function is used to make the predictions where above a certain threshold of probability, the data points are classified as "churn" and below them are "no churn". The sigmoid function can be written as $\frac{1}{1+e^{-z}}$ where $z = \sum_{i=1}^{n} \omega_i x_i + b$ where ω's are weights, x's are features, and b is the bias term. To obtain the weights and bias terms, cost functions are calculated and batch gradient descent is applied next. For logistic regression, the cost function is log loss because, during gradient descent, the chance of getting to a local minimum is higher than when using the least squared error, which is typically used in linear regression. Resulting log loss cost function is as follows:

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} [y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) * \log(1-h_\theta(x^{(i)})]$$

Eqn.1: The log loss cost function for logistic regression [7]

In eqn.1, h_θ is the sigmoid function. Using eqn. 1, the gradient descent algorithm is applied by taking its derivative with respect to the weights and the bias to find the weight that minimizes the cost. This derivation is as follows:

$$d\omega = \frac{1}{n} * (h_\theta(x) - y) * x)$$

Eqn.2: Derivative of the cost function with respect to weights [7]

To update the weights, eqn. 2, multiplied with a α hyper-parameter that is decided on later during validation, is subtracted from the original weights. This process is repeated for m iterations where m is also another hyper-parameter that is decided on in the validation process.
For prediction, the test data set is put into the sigmoid function, and at each data point, if the resulting probability is higher than a predetermined threshold (hyper-parameter that is decided on in the validation process), the point is classified as 1 (churn) and 0 (no churn) otherwise. Before putting the training, validation, and test data sets into these functions, they are standardized to avoid the vanishing gradient problem during the training phase [8].

Ridge Classifier:

Ridge classifier is similar to logistic regression but includes a regularization term in the loss function, which is helpful to prevent overfitting. The added term is a penalty term to reduce the weights close to zero, resulting in a model with less variance. This is called L2 regularization [10]. Ridge classifier is particularly useful when dealing with data where overfitting can be a problem. The added cost formula is as below:

$$J(\theta) = (-\frac{1}{n} \sum_{i=1}^{n} [y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) * log(1 - h_\theta(x^{(i)})]) - \lambda \sum_{j=0}^{n} \omega_j^2$$

Eqn.3: Loss function for ridge classifier [11]

where:

λ is the regularization parameter and everything else is the same as eqn. 2. To decide on λ, 10-fold cross validation is applied and its steps are mentioned in the results section. The hyperparameter λ is responsible for the balance between fitting the training data well and having a simpler model.
For the ridge classifier, the prediction function is also different from the logistic regression's sigmoid function. For this model, hyperbolic tangent is used to fit better with the class values used. Here, the class labels are -1 and 1 instead of 0 and 1. This is because the decision function in ridge classifier is typically defined as a linear combination of feature values, where positive and negative coefficients contribute to the positive and negative class predictions, respectively. This formulation naturally lends itself to the use of -1 and 1 as class labels, as the signs of the coefficients match the class labels.

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Eqn.4: The hyperbolic tangent function [11]

The Ridge classifier uses the same weight-update algorithm as logistic regression in the training phase. In prediction, the hyperbolic tangent function in eqn 4. is used where $z = \sum_{i=1}^{n} \omega_i x_i + b$ and ω's are weights, x's are features, and b is the bias term. If the result of eqn. 4 is less than 0, the data point is classified as -1 and if it is more than 0, it is classified as 1.

Naive Bayes:

Naive Bayes Algorithm is mostly used for classification tasks. The algorithm is based on Bayes' theorem, which means the probability of a hypothesis (in our project, the churn class label) given some features (such as phone usage) is proportional to the likelihood of the evidence given the hypothesis, multiplied by the prior probability of the hypothesis [9]. Naive Bayes requires all the features to be independent thus the preprocessing is extra essential in this step. The formula is:

$$P(y|x1,\ x2,\ \ldots,\ xn) = P(x1,\ x2,\ \ldots,\ xn|y) * P(y) \ / \ P(x1,\ x2,\ \ldots,\ xn)$$

Eqn.5: Naive Bayes prediction

where:
P(y|x1, x2, ..., xn) is the posterior probability for class label y given the features
P(x1, x2, ..., xn|y) is the likelihood for features given the class label y.
P(y) is the prior probability for y.
P(x1, x2, ..., xn) is the probability for features x1, x2, ..., xn occurring.
After the calculation is done the class (y) with the highest probability is selected as prediction.

Neural Network:

Neural Network (NN) is a supervised machine learning algorithm inspired by the structure and function of biological neural networks, such as the human brain. Neural networks learn by adjusting the weights and biases associated with the connections between neurons [13]. While training the network, labeled training data is fed to the algorithm and the algorithm iteratively updates the weights to minimize a loss function that measures the error between the predicted output and the true output.
For the project, a basic multi-layer perceptron (MLP) model, which is suitable for binary classification tasks and is capable of learning non-linear correlations between the input and output, was used [13]. The implemented NN consists of an input layer with a size of 13 (number of features), a hidden layer with 10 neurons and one neuron in the output layer which was designed for binary classification. The model updated its weights to minimize the MSE loss function, which measures the squared difference between the predicted and the true class labels. The function can be seen below:

$$MSE = (1/n) * \Sigma(\hat{y} - y)^2$$

Eqn.6: Mean Squared Error (MSE) as the Loss Function for the Neural Network

where;
n is the number of features in the dataset which in our case is 13
ŷ is the predicted value and y is the true class label

The model learned to make more accurate predictions in each iteration by backpropagation and gradient descent algorithm. The used gradient descent algorithm is online gradient descent. In the implemented model no bias terms were included. As for the activation functions the sigmoid function was used to squash the data between [0,1] and the function was preferred as it always has a derivative. The function can be seen below with its derivative which is essential for back propagation algorithm:

$$sigmoid(x) = 1 / (1 + e^{(-x)}) \text{ and } sigmoid\_prime(x) = sigmoid(x) * (1 - sigmoid(x))$$

Eqn.7: Sigmoid Activation Function and its Derivative

The derivative for the activation function is crucial because during backpropagation the gradient of loss wrt. weights are calculated.
In short, NN is trained over a predefined number of iterations called epochs and in each iteration the input is given to the network to obtain a predicted output. The loss between the predicted and the actual output is calculated. During backpropagation the derivative of this loss is obtained with respect to each weight. Then the weights are updated with gradient descent to minimize the loss. At the end of the iteration process weights were fixed and ready to be tried in the test dataset [12].

Machine Learning Tasks and Challenges:

For this project, we plan on evaluating the problem using various machine learning models to get more accurate results and compare these models in the end. Since this is a classification task, Logistic Regression, Ridge Classifier, Naive Bayes, and Neural Network models will be used. For binary classification, where we have two possibilities, churn = yes or churn = no, logistic regression uses the sigmoid function, and the coefficients are estimated using the log-likelihood function. A challenge with this model could be overfitting, i.e. the classifier can not provide good predictions for features that did not occur in training, and to overcome this a regularization technique like ridge regression could be applied to the model to produce another model: Ridge Classifier. The Ridge Classifier will penalize the higher coefficients in the model to reduce its complexity by adding a regularization term involving λ, which will be chosen using cross-validation, to the loss function of the original model [3]. The Naive Bayes model is based on Bayes' theorem but assuming conditional independence between every pair of features given the value of the class variable. The model also uses Maximum A Posteriori (MAP) estimation [4]. One challenge with this model is the "Zero Frequency" problem when a feature belongs to a certain category in the test dataset, which was not found in the training dataset, then the model will assign a zero probability leading to no prediction. To tackle this challenge, smoothing techniques can be applied. Lastly, the neural network models are prone to overfitting and if it happens, it will be handled with L2 regularization once again. Also hyperparameter tuning is another challenge with neural networks since they have many hyperparameters such as learning rate, batch size, number of layers, and activation functions. Finding the optimal combination of hyperparameters is essential for achieving good performance. To find the best parameters, tracking the loss of the validation set will be essential. Another challenge which is related to the nature of the data set is that the data set contains much more negative classes than positive classes. This could lead to all of the models to have low recall because they will most likely predict "no" then "yes" and the true positive rate might be low. To tackle this issue, data augmentation will be applied to the data set to have more balanced data to get more reliable models in the end.

Results:

Pre-processing:

As asked by our TA, we started the implementation stage with the pre-processing of the data.

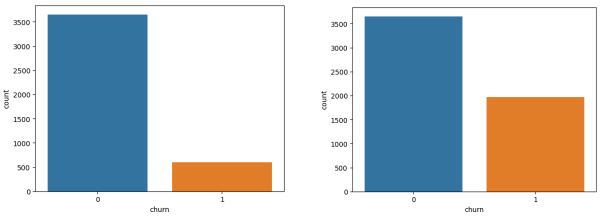


Figure 4: Dataset Class Distribution Before and After Data Augmentation

Our dataset as explained before contained around 86% "no" and 14% "yes". Since the dataset was biased it was hard to work on. Thus, as suggested by out TA we expanded the "yes" class rows using data augmentation by noise injection which included adding noise to features of "yes" labeled rows and appending them as new rows so that in the end our data had 5618 rows and 65% to 35% "no" "yes" class distribution. The graphs of our data before and after data augmentation can be seen in the figure 4.

Then the data was explored with plots. In order to do that, the numeric and categorical columns were identified and the class distribution in these columns were observed.

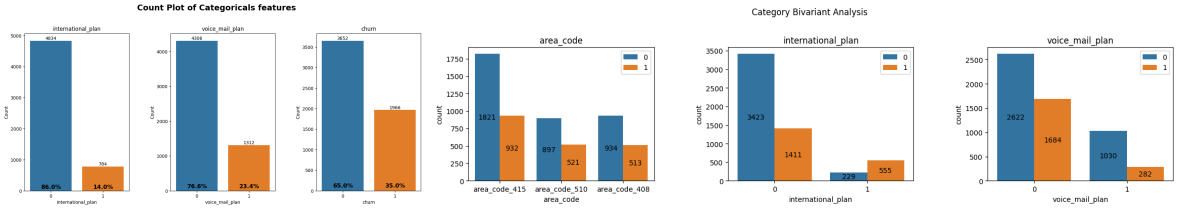


Figure 5: Categorical Columns (Left) and Class Distribution in These Columns (Right)

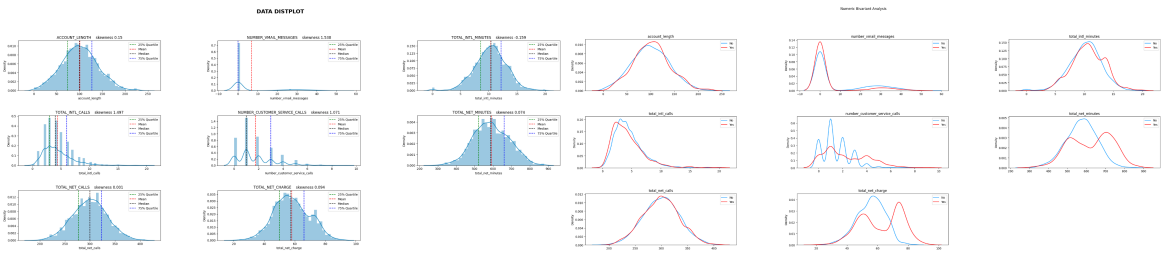


Figure 6: Numeric Columns Data Distribution (Left) and Class Distribution in These Columns (Right)

In the figures the categorical and numeric columns can be seen after the first manipulation of features which was combining plans besides international plans in the same column, dropping the state column. Also, categorical columns besides area code were mapped to binary values. For area code one-hot encoding was used as requested by our TA. Finally as observed from our previous correlation matrix, the international charge and international call minutes had 100% correlation so one of the features got dropped as we decided we don't need to investigate both, the chosen column to drop was international charge. After all the pre-processing final columns are as below:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_intl_minutes	total_intl_calls	number_customer_service_calls	total_net_minutes	total_net_calls	total_net_charge	churn	area_code_area_code_408	area_code_area_code_415	area_code_area_code_510
4113	115.0	0	0	0	4.8	2.0	3.0	828.6	245.0	80.76	1	0	1	0
3891	104.0	0	0	0	14.1	8.0	2.0	660.0	335.0	68.72	0	0	1	0
2170	190.0	0	0	0	9.3	4.0	2.0	627.5	319.0	59.51	0	1	0	0
3480	80.0	0	0	0	14.7	5.0	3.0	467.2	269.0	45.32	0	0	1	0
3394	118.0	0	0	0	7.7	4.0	1.0	636.4	298.0	65.33	0	1	0	0

Figure 7: Data Columns

The correlation between these columns can be seen below:

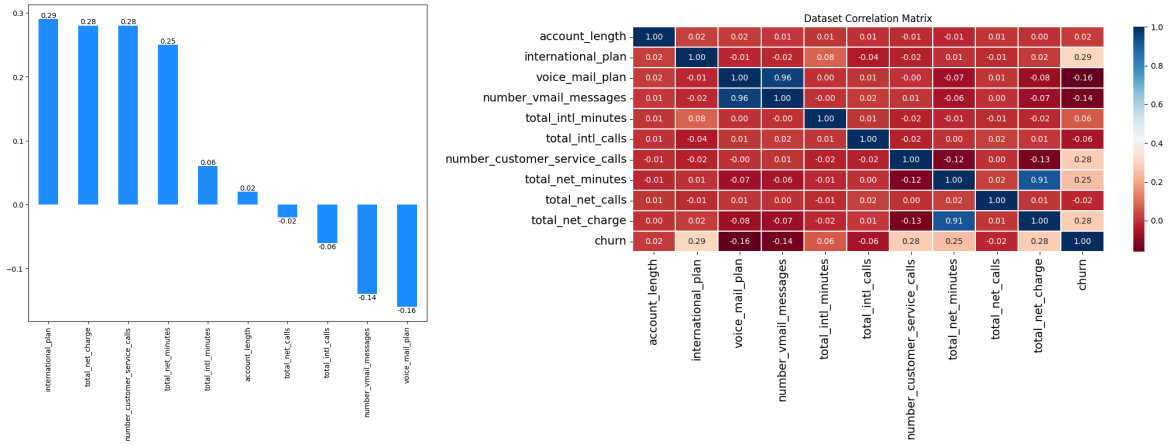


Figure 8: Correlation Graph and Matrix in Final Columns

The end dataset columns can be seen in the above figures alongside their correlation to the churn rate. With these results the modeling started.

Logistic Regression:

During training, validation and training costs are calculated simultaneously to produce a cost graph of each for every iteration and compare them to see if there is overfitting or not. For $\alpha = 1 * 10^{-4}$ and iteration number = 1000, the below loss graph was obtained and it showed no overfitting (validation error is less than training error and they are both decreasing so the model is learning):

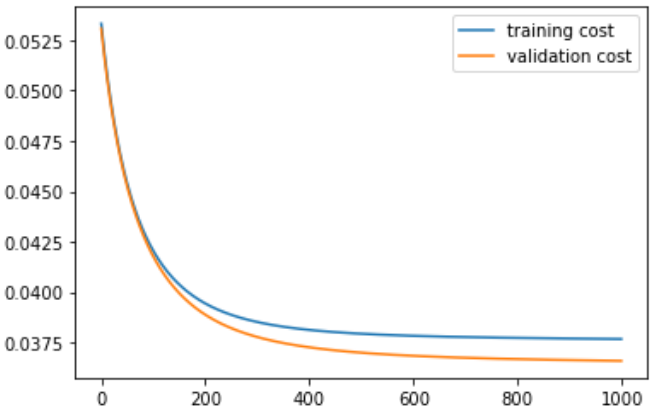


Figure 9: Training and Validation loss curves for Logistic Regression

Next, test data set was put into the prediction function with the probability threshold of 0.44. There was a skewed distribution of positive and negative classes in the data set that resulted in low recall because it is the true positive rate and since the data set had much more negatives than positives, the model often predicted the actual positive classes as negative. So, as mentioned in the pre-processing section, after the data augmentation the number of negative classes went from 86% to 65% and positive classes went from 14% to 35% to have more accurate results about the model performance. This is also why the results used to show a high accuracy and specificity (true negative rate) but low recall (true positive rate). To avoid this large gap between these performance metrics, some hyperparameters were altered, such as the probability threshold in prediction. Below graphs show the tradeoff between F1 score and accuracy and how they change when the probability threshold is altered. The first plot is after the data augmentation and the second plot is before.

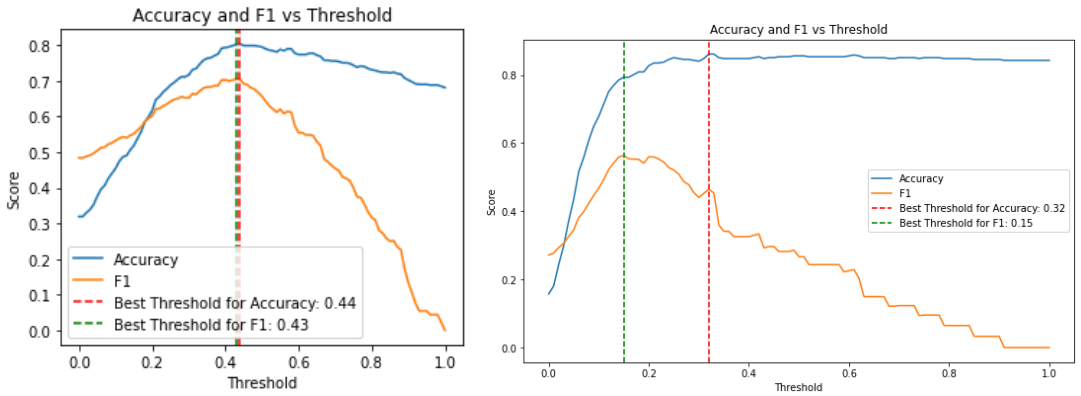


Figure 10: Best threshold values for F1 score and accuracy for logistic regression (after and before the data augmentation respectively)

From fig. 10, it can be seen that there is no tradeoff left between the F1 score and the accıracy after data augmentation. This is because the data set is more balanced and the model stops predicting only one of the classes, the negative one, and is more reliable. Using the values in figure 10, the threshold became 0.44, another prediction was made which resulted in improved metrics given as follows:

```
Accuracy =  0.803921568627451
F1 score =  0.7043010752688171
Precision =  0.6787564766839378
Recall =  0.7318435754189944
Error =  0.19607843137254902
Confusion matrix =  [[131  62]
 [ 48 320]]
```

Figure 11: Improved performance metrics for logistic regression

From figure 11 it can be seen that F1 score, precision and recall improved without decreasing the accuracy. The accuracy is high, 80% and the model is no longer predicting just one of the classes because if it did, the accuracy would be either 65% or 35%. The training time for the algorithm is estimated to be 7 seconds, which is low.

Ridge Classifier:

During the implementation phase of the ridge classifier the same constants were used. Which are $\alpha = 1 * 10^{-4}$ and iteration number = 1000, below loss graph was obtained. Since the hyperbolic tangent function is used for prediction, the class labels, i.e. y values, are transformed into -1 and 1 where -1 represents the 0 class in logistic regression (no churn) and 1 represents the 1 class. Cost is the same as logistic regression, binary cross entropy with L2 regularization added, as mentioned in the machine learning models used section. To obtain the L2 regularization parameter, 10-fold cross-validation was implemented. The data set was split into training and test first (in the pre-processing phase of the project). Then, for cross-validation, the training data set was split into 10 again. One of these 10 pieces was assigned the validation data set and the rest were used as training. Then, the algorithm was run as usual for the ridge classifier, and at the end of the last iteration, the validation cost was put into a "holdout" list. This procedure was repeated 10 times, each time the validation set was changed and in the end, the mean of the 10 holdout costs was found. Various values of λ were used and they were as follows: [0.001, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 10, 100, 150, 1000]. To find the λ which causes the lowest cost on the training data set below graph is used:

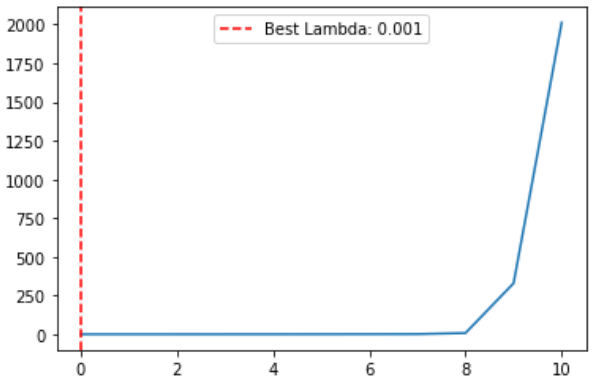


Figure 12: λ values and their corresponding mean costs after 10 folds

From fig. 12, it can be seen that λ = 0.001 was chosen in the end to use in the ridge classifier. After this, the ridge algorithm was run and below loss curves for validation and training were obtained where the validation cost is lower than the training cost and they are both decreasing so the algorithm learns and it doesn't overfit. To see the difference the value of λ makes, for λ = 100, the training is done once again and curves in fig. 13 are produced.

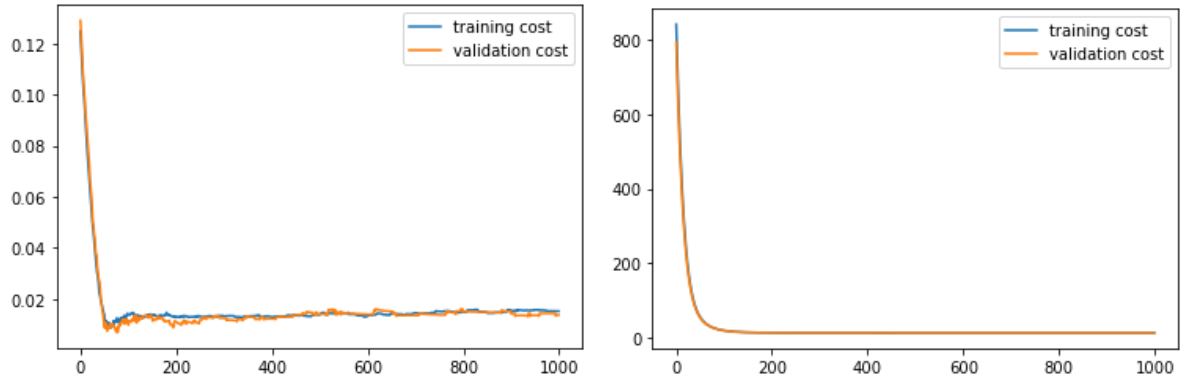


Figure 13: Loss Curves for Ridge Classifier λ = 0.001 and λ = 100 side by side

In fig. 13, in the left plot, the lowest loss is below 0.02 but in the right plot it is below 200 so the loss is 10000 times more when λ = 100 compared to λ = 0.001. Then, predictions were made. The only difference between logistic regression and ridge classifier in this step was that since hyperbolic tangent was used for prediction, instead of a threshold that determines the class label in sigmoid, if the value found from hyperbolic tangent was below 0, the data instance is predicted as -1, else it was predicted as 1. Then the F1 score, accuracy, precision, recall, confusion matrix and error were calculated. According to the feedback from our TA, RMSE was no longer calculated for ridge classifier. The results were as below:

```
Accuracy =  0.7967914438502673
F1 score =  0.6627218934911242
Precision =  0.7044025157232704
Recall =  0.6256983240223464
Error =  0.20320855614973263
Confusion matrix =  [[112  47]
 [ 67 335]]
```

```
Accuracy =  0.7754010695187166
F1 score =  0.5499999999999999
Precision =  0.7623762376237624
Recall =  0.4301675977653631
Error =  0.22459893048128343
Confusion matrix =  [[ 77  24]
 [102 358]]
```

Figure 14: Performance Metrics for Ridge Classifier at λ = 0.001 and λ = 100

In fig. 14, when λ becomes 100 instead of 0.001, accuracy drops by 2% whereas the F1 score drops by 11%. The outcome of the ridge classifier was similar to the logistic regression. The accuracy is

only 0.8% lower whereas precision is 0.2% higher. The training time for this algorithm is estimated to be 6 seconds, which is lower than logistic regression. This was an expected result since with regularization, the complexity of the calculations decreases as the weights are lowered. However, the 10-fold cross-validation took about 2 minutes which makes the overall runtime of the ridge classifier the highest among all 4 models.

Naive Bayes Classification:

As for the third model of our project, Naive Bayes was chosen as it is proven to be successful at classification tasks. It is also a simpler model compared to the other three models in the project, so its runtime is expected to be the least. The only performance metrics used in this model are accuracy, F1 score, precision, recall, confusion matrix and error. Below are the results:

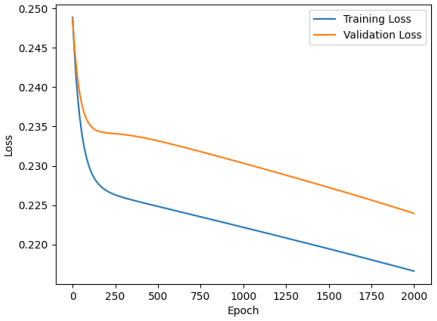
```
Accuracy 0.8306595365418895
F1 0.7665847665847665
Precision 0.8041237113402062
Recall 0.7323943661971831
Confusion Matrix
[[156  38]
 [ 57 310]]
Error 0.16934046345811052
```

Figure 15: Predictions of Naive Bayes

From fig. 15, it can be seen that the model has high accuracy and F1 scores and a low error. Its runtime also turned out to be 2 seconds, as expected. The "Zero Frequency" problem mentioned in the machine learning and tasks section wasn't encountered because the data set had an instance for each class, churn and no churn. Therefore, smoothing techniques were not necessary and not applied.

Neural Network:

Our final model was Neural Network. We used MLP as our model and the details of the algorithm was explained in the previous section. The algorithm had one hidden layer with 10 neurons. It used sigmoid activation function, online gradient descent algorithm and MSE loss function. While training the model we encountered some problems. For example when the learning rate was too low (e.g. = 0.0001) the model stuck in a local minimum while trying to find the lowest error. This resulted in:




```
Accuracy 0.6773618538324421
F1 0
Precision 0
Recall 0
Confusion Matrix
[[  0   0]
 [181 380]]
Error 0
Specificity 0
```

Figure 16: Neural Network not Working as Intended

Here the model only predicted "no" as the label and did not learn. In order to fix this we tried changing the learning rate.
Thus in the performance analysis of the model the optimal learning rate for the model was decided to be 0.005 and we saw the best loss curve in 2000 epochs. The MSE loss and accuracy curve for every epoch for the training and the validation dataset can be seen below:

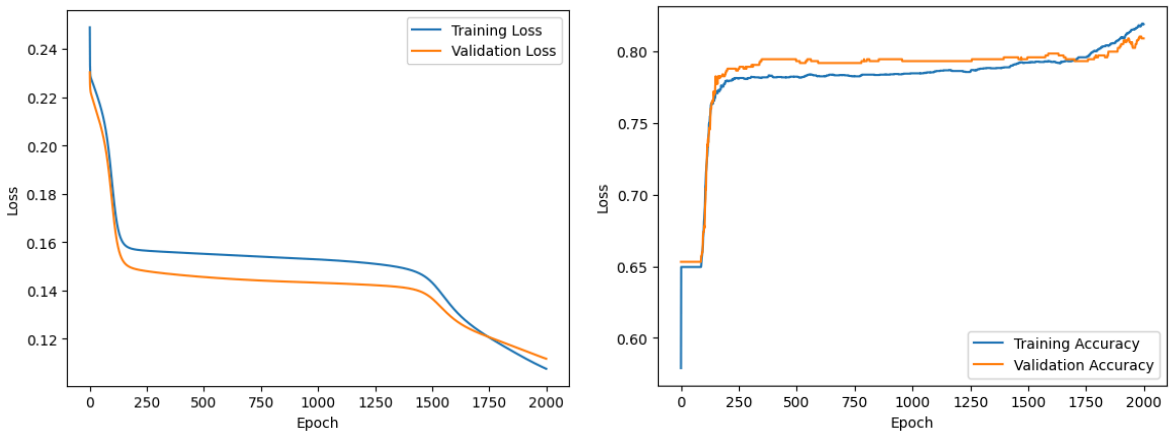


Figure 17: Neural Network Loss (Left) and Accuracy (Right) Graphs for Training and Validation

In the figure 17 it can be seen that the loss is iteratively decreasing so the model is learning. This can also be seen in the accuracy curve of the model. Also from the curves it can be said that there is no significant over-fitting in the model as both training and validation accuracy increased over time and the training loss is higher than the validation loss (this changes slightly after the epoch 1750 but can generally be claimed that there is no over-fitting). This model was found to be successful and later tried on the test set. The results are as below:

```
Accuracy 0.8627450980392157
F1 0.8079800498753117
Precision 0.8617021276595744
Recall 0.7605633802816901
Confusion Matrix
[[162  26]
 [ 51 322]]
Error 0.13725490196078433
Specificity 0.9252873563218391
```

Figure 18: Neural Network Performance on Test Set

The Neural Network got the highest accuracy in our models alongside the highest F1 score but it required way more computing power in comparison to our models with an estimated runtime around 1 minute. After changing the learning rate, the confusion matrix also looked promising.

Conclusion:

To evaluate the results discussed in the previous section, below table will be used:

Table 2: Model Evaluation

	Runtime (s)	Accuracy (%)	F1 Score (%)	Error (%)
Naive Bayes	2	83.06	76.65	16.93
Logistic Regression	7	80.39	70.43	19.61
Ridge Classifier	6 (+120 for cross-validation)	79.68	67.61	20.32
Neural Network	60	86.27	80.79	13.72

From table 2, we can see that the quickest model in terms of training time is Naive Bayes, as expected, since it is the simplest among all four models. In terms of accuracy, F1 score and error

metrics, the best model is the Neural Network because it is the most detailed one, thus the most reliable among all models. Ridge classifier is an improvement to logistic regression in terms of preventing overfitting. In this project, however, there was no overfitting in the logistic regression model. Overall, all models have high accuracy (almost all of them are 80+) and low error rates with the correct hyperparameters chosen for each model.

Even though all the models showed high accuracy, to improve this classification task further, the gradient descent algorithm can be changed in the future. Logistic and ridge models used batch gradient descent, to achieve more local minimums mini-batch gradient descent could be applied. For the neural network, different optimizers such as adam by getting the help of various Python libraries, could be used to get better and neater results.

References:

[1] L. Murphy, "SaaS Churn Rate: Benchmarks, Industry Reports, and Analysis," [Online]. Available: https://sixteenventures.com/saas-churn-rate. Accessed on: Apr. 26, 2023.
[2] Kaggle Inc., "Customer Churn Prediction 2020," [Online]. Available: https://www.kaggle.com/competitions/customer-churn-prediction-2020/data. Accessed on: Apr. 26, 2023.
[3] M. Gersmann and J. Stefanowski, "Comparison of Logistic Regression and Ridge Classifier for Binary Classification Tasks," [Online]. Available: https://www.cis.uni-muenchen.de/~stef/seminare/klassifikation_2021/referate/LogisticRegressionRidgeClassifier.pdf. Accessed on: Apr. 26, 2023.
[4] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, Oct. 2011. [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed on: Apr. 26, 2023.
[5] Nextommerce, "Churn Rate Benchmarks for 2021: How Does Your Industry Stack Up?," [Online]. Available: https://nextommerce.com/churn-rate-benchmarks/. Accessed on: Apr. 26, 2023.
[6] CleverTap, "Churn Rate: Definition, Calculation, and Industry Benchmarks," [Online]. Available: https://clevertap.com/blog/churn-rate/. Accessed on: Apr. 26, 2023.
[7] J. Brownlee, "Understanding Logistic Regression step by step," [Online]. Available: https://towardsdatascience.com/understanding-logistic-regression-step-by-step-704a78be7e0a. Accessed on: Apr. 26, 2023.
[8] J. Brownlee, "Which Machine Learning Models Require Normalized Data?," [Online]. Available: https://towardsdatascience.com/which-models-require-normalized-data-d85ca3c85388. Accessed on: Apr. 26, 2023.
[9] Javatpoint, "Naive Bayes Classifier in Machine Learning," [Online]. Available: https://www.javatpoint.com/machine-learning-naive-bayes-classifier#:~:text=Na%C3%AFve%20Bayes%20algorithm%20is%20a,a%20high%2Ddimensional%20training%20dataset. Accessed on: Apr. 26, 2023.
[10] J. Brownlee, "The Basics of Logistic Regression and Regularization," [Online]. Available: https://towardsdatascience.com/the-basics-logistic-regression-and-regularization-828b0d2d206c. Accessed on: Apr. 26, 2023.
[11] Wolfram MathWorld. (n.d.). “Hyperbolic Tangent. Wolfram MathWorld.” [Online] Available: https://mathworld.wolfram.com/HyperbolicTangent.html. Accessed: May 15, 2023.
[12] R. M. Johnson and S. M. Davis, "A Comparative Study of Neural Network Algorithms for Binary Classification," in 2020 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 145-150, 2020, doi: 10.1109/DSAA49011.2020.00027.
[13] H. Chen and T. Liu, "A Comprehensive Study of Multi-Layer Perceptron for Pattern Recognition," in IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 10, pp. 2943-2956, Oct. 2019, doi: 10.1109/TNNLS.2018.2876003.