DS 5220: Supervised Machine Learning and Learning Theory

TELCO CUSTOMER CHURN PREDICTION

Shuchita Mishra

Dataset: Telco Customer Churn IBM Data

ABSTRACT

Customer churn analysis helps us understand the journey a customer goes through on the service and the key areas where they fall off, which in turn helps us understand where we can improve our strategies and increase brand loyalty. This project proposes to predict the customer churn for a telecom company using predictive modeling.

Lost customers mean a direct loss of revenue which is why it is important to understand the reason behind customers leaving the company. Customer churn is the most important metric to evaluate the performance of a company. From the point of view of the company, it is necessary to gain this information because getting new customers is generally more tedious and costlier than retaining old customers.

I. INTRODUCTION

Customer churn is the rate at which a business's customers leave the company. For a telecom company like "Interconnect", this makes a significant business impact directly affecting the profits. Focusing on long-term relationships and observing customer behavior is more cost-effective than the marketing campaigns to attract new customers. *Customer churn is the percent of customers that stopped using a company's services/product during a particular time.*

> Customer Churn = Number of churned customers / Total customers in the company

It costs 25 times more to acquire a new customer as compared to retaining customers. This alone shows why tracking customer churn is so important. Also, the more we learn about customer behavior, the better understanding we'll gain of the future expected revenue.

When I started this project, I planned to be able to predict the churn of clients for this company. Finally, through this project I analyzed the dataset and trained various ML classifier models to perform "Uplift Modeling" by targeting potential customers with the intention of reducing marketing costs while preserving the profit margins.

II. DATA ANALYSIS AND MODELING

A. Data Analysis

I performed EDA on this dataset to help understand the trends, patterns, and relationships between different features of the dataset that cannot be intuitively understood. One of the things that were observed early on, is that the dataset is only pertinent to California. Another thing to note here is that there is no region with more churning i.e. the number of customers churned is almost uniform throughout even though there is a mild concentration in the center. Hence, I used this as convincing evidence to remove columns such as latitude, longitude, zip code, state, and country. The addition of these columns won't add any information to any of my models. It would just increase the noise.

Then I used graphs to understand the effect that some categorical features had on *Churn Value*, as seen in *Figures 3, 4 and 5*.

Feature Extraction and Preprocessing:

1. Checking for Missing Values:
I first checked whether the data had any missing values, and realized that the dataset did not

contain any null values. However, the feature "*Total Charges*" had 11 empty strings. So I imputed ***Total Charges*** as the product of ***Tenure*** and ***Monthly Charges***.

2. One Hot Encoding:

The dataset had a lot of categorical variables which could not be given as an input to the models as many machine learning algorithms cannot work with categorical data directly. Hence, the categories must be converted into numbers first, for which I have used One Hot Encoding at the risk of increasing the number of columns and hence dimensionality of the dataset.

3. Intuitive feature engineering:

Some of the features in the dataset don't make an impact on the target variable, such as *Customer_Id, Latitude, Longitude, City,* etc. Hence, I simply removed those variables for modeling.

4. Heatmap/Correlation analysis:

A correlation matrix is simply a table that displays the correlation coefficients for different variables and is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data. In *Figure 8* below, we can see clearly from the last line that features like *Phone_Service, Multiple Lines, Streaming TV and Streaming Movies* have a low correlation with our target variable i.e *Churn Value*. *Contract, Total Charges, Online Security, and Tenure Months* seem to negatively affect the *Churn Value.*

5. Backward Stepwise regression (or backward elimination):

It is a recursive feature elimination technique that works as follows:

1. **Begins** with a model that contains all variables under consideration.
2. **Then** starts removing the least significant variables one after the other.

As you can see from *Figure 7*, where we will exclude ***Multiple Lines*** as it had the highest p-value. A value of 0.897 indicates a close to 90% probability that a variable doesn't affect the target variable. Subsequently, we also remove Gender, Streaming TV, and Streaming Movies from the model.[10]

6. Standard Scaling:

Standardization transforms the data to have a zero mean and a variance of 1, that is, they make the data unitless. This operation is performed feature-wise in an independent way. We used the StandardScaler from sci-kit learn's sklearn.preprocessing package.

7. Synthetic Data Augmentation (SMOTE) :

I observed in EDA that the data is an imbalanced set as it has 26.5% of instances belonging to Churned customers while 73.5% instances belong to the customers that did not leave the company resulting in my models being more biased towards the majority class, causing poor classification of the minority class. To avoid this problem, I performed data augmentation using synthetic tabular data from the SMOTE (Synthetic Minority Over-sampling Technique) algorithm. The general idea of SMOTE is the generation of synthetic data between each sample of the minority class and its "k" nearest neighbors.

8. Principal component analysis:

The dataset has 33 features that could give way to noise and difficulties while modeling, hence, to avoid such problems I decided to also use PCA to tackle the curse of dimensionality. Also, if the learning algorithm is very slow due the number of input dimensions being very high, then using PCA can speed up this process. So, I have used PCA from sci-kit-learn's sklearn.decomposition package.

FIGURES:

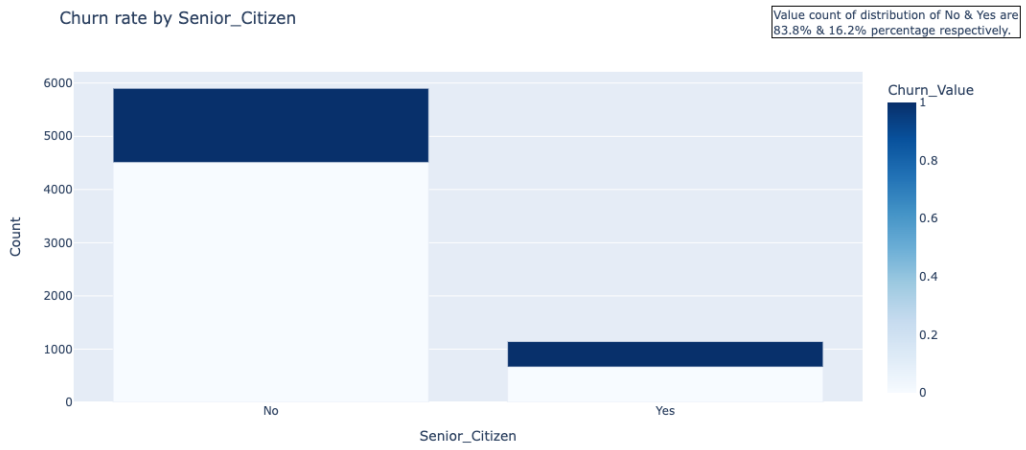


Figure 1: Senior Citizens who are customers of the company are more likely to churn.

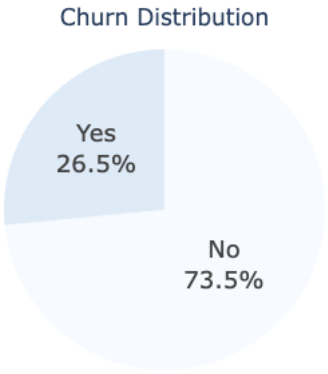


Figure 2: Dataset is imbalanced as it has 26.5% of Churn customer data.

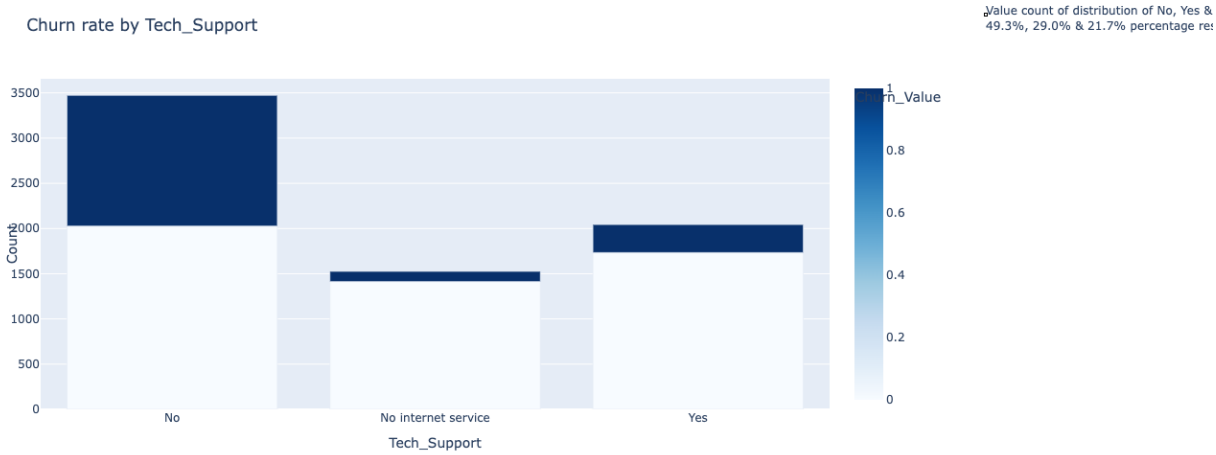


Figure 3: Customers opting for no tech support are more likely to churn.

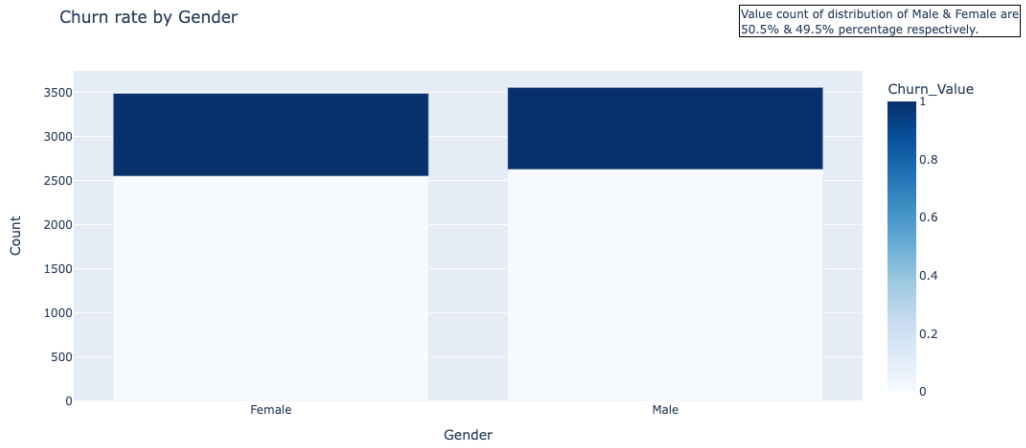


Figure 4: Gender does not affect Churn Value.

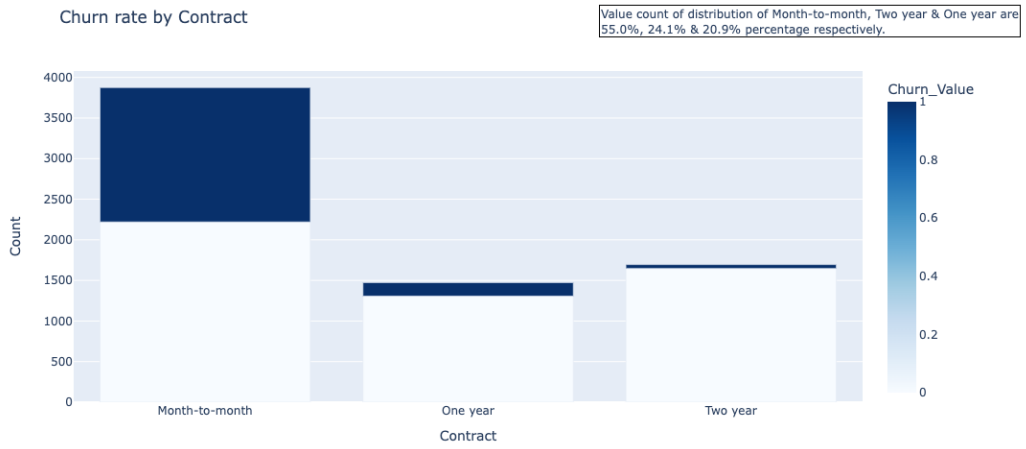


Figure 5: Customers on a monthly contract with the company are more likely to churn.

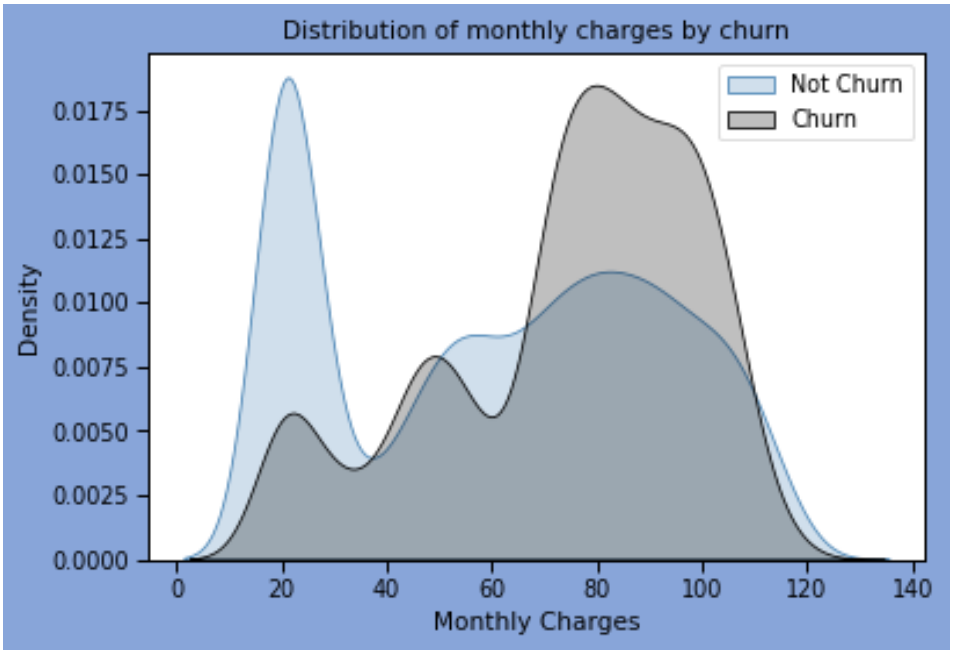


Figure 6: Customers who pay higher monthly charges to the company are more likely to churn.

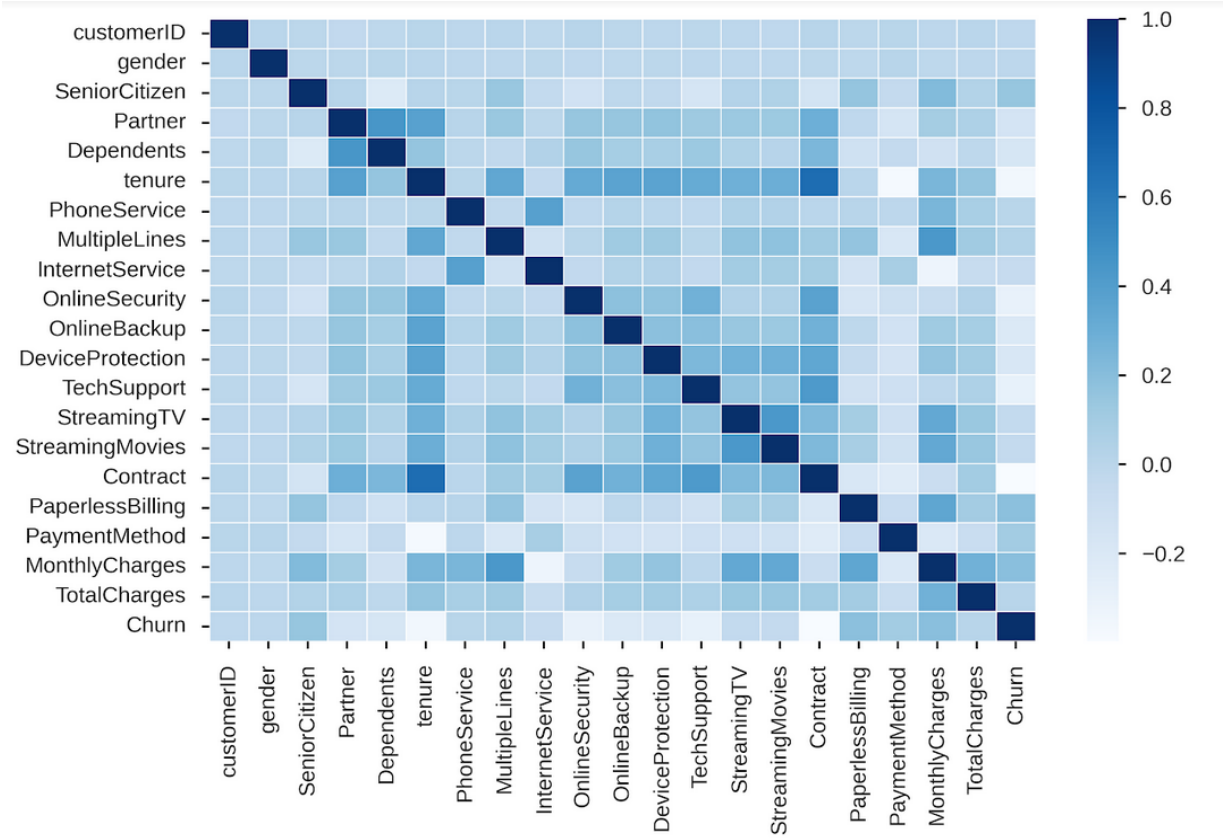


Figure 7: Correlation Matrix for the dataset.

Covariance Type: nonrobust

	coef	std err	t	P>|t|	[0.025	0.975]
Gender	-0.0018	0.009	-0.199	0.842	-0.019	0.016
Senior Citizen	0.0418	0.013	3.285	0.001	0.017	0.067
Partner	0.0353	0.010	3.431	0.001	0.015	0.055
Dependents	-0.1599	0.012	-13.685	0.000	-0.183	-0.137
Tenure Months	-0.0022	0.000	-4.778	0.000	-0.003	-0.001
Phone Service	-0.1962	0.035	-5.540	0.000	-0.266	-0.127
Multiple Lines	0.0016	0.012	0.129	0.897	-0.022	0.025
Internet Service	-0.0396	0.012	-3.198	0.001	-0.064	-0.015
Online Security	0.0631	0.011	5.560	0.000	0.041	0.085
Online Backup	0.0256	0.011	2.394	0.017	0.005	0.047
Device Protection	0.0243	0.011	2.138	0.033	0.002	0.047
Tech Support	0.0788	0.012	6.695	0.000	0.056	0.102
Streaming TV	0.0018	0.012	0.148	0.883	-0.022	0.026
Streaming Movies	0.0020	0.012	0.163	0.870	-0.022	0.026
Contract	-0.0686	0.007	-10.425	0.000	-0.082	-0.056
Paperless Billing	0.0502	0.010	5.085	0.000	0.031	0.070
Payment Method	-0.0102	0.005	-2.259	0.024	-0.019	-0.001
Monthly Charges	0.0083	0.000	19.720	0.000	0.007	0.009
Total Charges	-5.328e-05	6.32e-06	-8.427	0.000	-6.57e-05	-4.09e-05

Figure 8: Snippet of the First Step of Stepwise Regression.

B. Model

1. Logistic Regression:

It is an algorithm that can be used for regression as well as classification tasks, but it is widely used for classification tasks. It is a more specific case of linear regression where we have a binary target variable. I use the sigmoid function as it maps the predictor variable between 0 and 1. If the prediction is over 0.5, it is categorized as class 1, the first, or else it's class 0. The mathematical function I use for Logistic Regression classification is explained below.

In this case, I will use the Logistic regression algorithm on the dataset features (x_1, $x_2 \ldots x_d$) to classify them as Churn (Yes) or Not Churn (No). The equation for Logistic regression cost function using Lasso regularization for feature selection is as follows:

$$\Sigma_{i = 1 \text{ to } n} (y_i - \beta_0 - \Sigma_{j = 1 \text{ to } d} \beta_{ij} x_{ij})^2 + \lambda \Sigma_{j = 1 \text{ to } n} |\beta j|$$

2. K-nearest Neighbor:

KNN is a non-parametric and lazy learning algorithm. The non-parametric algorithm means that it makes no assumptions about the underlying data structure. Lazy Learner means that it doesn't train the model until a test query is made. It is useful in applications where the dataset is ever increasing. K stands for the nearest neighbor, and it makes the predictions based on feature similarity. KNN is an instance learner which requires all the data to be available at the same time which makes it prone to overfitting. In this project, I selected the value of K using cross validation.

The pseudo-code for the same is as follows:

```
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
        distances = list()
        for train_row in train:
                dist = euclidean_distance(test_row, tra
                distances.append((train_row, dist))
        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(num_neighbors):
                neighbors.append(distances[i][0])
        return neighbors
```

Figure 9: kNN algorithm

3. Support Vector Machines:

In the SVM algorithm, we plot each data item as a point in n-dimensional space, n being the number of features in the dataset. Then, we perform classification by finding the hyper-plane that differentiates the two classes the best. SVM algorithms also make use of Kernels, to take data as an input and transform it to the required form. I have used the RBF kernel in our model.

The kernel of the RBF model is as follows:

$$K(x_i,x_j)=\exp(-\gamma||x_i-x_j||^2)$$

where; K is th ebernel function which is used in my SVM classifier, x_i and x_j are the features and γ is the kernel parameter. A small value of γ will make the model behave like a linear SVM. A large value of γ will make the model heavily impacted by the support vectors examples.

4. Random Forest:

This algorithm creates decision trees on sub-samples of the dataset, and then predicts each of the decision trees, and finally, it uses voting to get the best solution out of those decision trees. In this project, I have used different numbers of n estimators (number of decision trees in the forest) such as 50, 150, 250,350,450.

The pseudo-code for the Random Forest hyper parameter tuning using GridSearchCV() is as follows:

```
def randomForestModelWithParameterTuning(X_train, y_train, X_test, y_test):
    t1= time.time()
    params = { 'n_estimators': [50, 150, 250,350,450],
      'max_features': ['sqrt', 0.25, 0.5, 0.75, 1.0],
      'min_samples_split': [2, 4, 6]
      }

    rfc = RandomForestClassifier()
    rfc_cv = GridSearchCV(rfc, params, scoring = "accuracy", n_jobs = n_cpus , verbose = 0, cv = 5)
    rfc_cv.fit(X_train, y_train)
    best_params = rfc_cv.best_params_
    print(f"Best parameters: {best_params}")
    #best_params = {'max_features': 0.5, 'min_samples_split': 6, 'n_estimators': 250}
    rfc = RandomForestClassifier(**best_params)
    rfc.fit(X_train, y_train)
    rf_f1=evaluation(rfc, X_train, y_train, X_test, y_test)
    t2= time.time()
    print("RandomForest is done with F1 score " + str(rf_f1)  + " Time is " + str(t2-t1))
```

Figure 10: Random Forest algorithm with hyper parameter tuning

5. XGBoost:

XGBoost, which stands for eXtreme Gradient Boosting, is an implementation of Gradient Boosted Decision Trees. It attempts to accurately predict a target variable by combining ensemble estimates from a set of simpler and weaker models. For this project, I have used the gbtree booster as it uses a version of regression tree as a weak learner. XGBoost uses Taylor series to approximate the value of the loss function for a base learner $f_t(x_i)$, thus, reducing the load to calculate the exact loss for different possible base learners. Taylor's series expansion can be described as:

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots$$

Figure 11: Taylor's series expansion

6. Ensemble Model:

This involves using multiple different models to predict the outcome, which can be achieved by using many different modeling algorithms or using different training data sets. The ensemble model will then aggregate the prediction of each base model and generate one final prediction on the unseen data. I have implemented an ensemble model which uses different modeling algorithms such as Logistic Regression, Support Vector Machines, Random Forest Classifier, and XGBoost to make the Churn prediction.

7. LightBoost:

LightGBM is a gradient boosting framework based on the decision trees algorithm. The major difference between LightGBM and other boosting algorithms is that it splits the tree leaf-wise with the best fit whereas other boosting algorithms split the tree depth-wise or level-wise. The leaf-wise algorithm can reduce more loss than the level-wise algorithm.

8. Artificial Neural Networks:

ANN tries to mimic the way the human brain works to understand the relationship between a set of data. It comprises node layers, containing an input layer, one or more hidden layers, and an output layer. Activation functions are used to decide whether a neuron should be activated or not. The purpose of the activation function is to introduce non-linearity in the output of the neuron. Here, I have used the ReLu activation function in intermediate layers and the Sigmoid activation function for the output layer in my project. The following image [1] shows the mathematical model behind the ANN concept I have used:

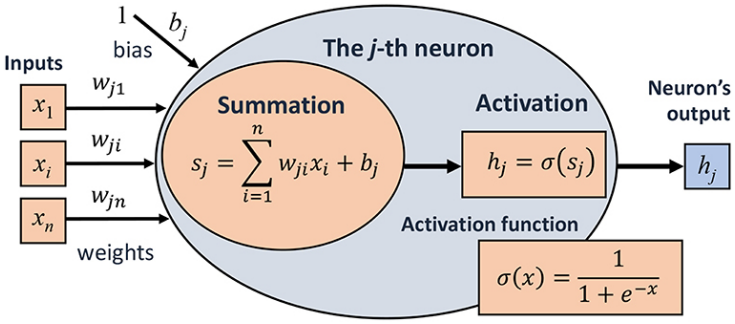


Figure 12: Artificial Neural Network basis

Hyperparameter Tuning:

For hyperparameter tuning we used the following two approaches:

(i) GridSearchCV: In grid search, we try all the combinations of the preset list of hyperparameters and then evaluate the model for all the combinations. One of the major drawbacks of grid search is that when it comes to dimensionality, it suffers when the number of hyperparameters grows exponentially.

(ii) RandomSearchCV: Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. It tries random combinations of a range of values. To optimize with random search, the function is evaluated at some number of random configurations in the parameter space.
Out of the two, I found random search to be a better option as it is faster and along with that it has a better chance of converging to the optimal set of parameters. The random search was about 16% faster and gave better results as well.

Additionally, I also used MLFlow to track the parameter tuning over the ML Lifecycle since there were a lot of different combinations of input such as Scaling, SMOTE, SMOTE + Scaling, SMOTE + PCA, etc.

Evaluation metrics:

In my project, as highlighted above in EDA, the Churn classes are highly imbalanced. Also, we

know that the F1 score is balancing precision and recall on the positive
class while accuracy looks at correctly classified observations both positive and negative. That makes a big difference especially for
the imbalanced problem in my dataset where by default our model will be good at predicting true negatives and hence accuracy will be high. Additionally, one big difference between F1 score and ROC AUC is that the first one takes predicted classes and the second takes predicted scores as input. Because of that, with F1 score you need to choose a threshold that assigns your observations to those classes. Often, you can improve your model performance by a lot if you choose it well.

Since, my dataset is heavily imbalanced and I mostly care about the positive class since we are more interested in the customers who will churn (we can try retaining them), I've considered using F1 score, or Precision-Recall curve and PR AUC. The additional reason to go with F1 (or Fbeta) is that these metrics are easier to interpret and communicate to
business stakeholders.

Comparative analysis of multiple ML models trained:

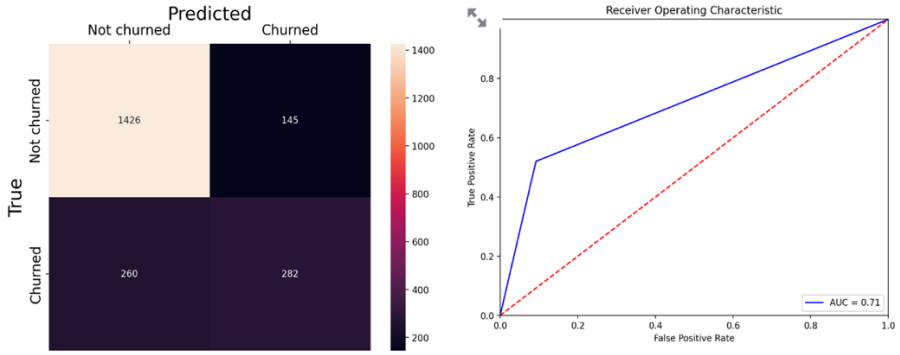
Results, in a nutshell, look good as on day 1 I had a baseline with an F1 score of 0.61. In *Table 1*, we can see the best result for each model corresponding to each of the different preprocessing steps I had tried.
The confusion matrix and ROC curve for the top 3 models indicate that the ensemble model (LR + SVM + Random Forest + KNN + XGBoost) was the best model.

As seen in most of my results, the Random Forest classifier performed much better as compared to SVM classifier, KNN and Logistic Regression. This is due to randomness of Random Forest. Most of the randomness-based algorithms yield better results most of the time in most cases, since, due to randomness, the error rate becomes low and improved results are obtained. Now, Random Forest is a bagging algorithm which uses multiple base learners (in this case decision trees) to improve and select the best classifying hyper-parameters and hereby increase the performance of the model.

For my scaled, balanced data, I observed that XGBoost performed the best with an F1 score of 0.794. Again, being an ensemble learning algorithm, more precisely, a boosting algorithm, XGBoost improves the accuracy of the model at each base learner output. Boosting happens to be iterative learning which means the model will predict something initially and self-analyses its mistakes as a predictive toiler and give more weightage to the data points in which it made a wrong prediction in the next iteration. After the second iteration, it again self-analyses its wrong predictions and gives more weightage to the data points which are predicted as wrong in the next iteration. This process continues as a cycle. Hence technically, if a prediction has been done, there is an at most surety that it did not happen as a random chance but with a thorough understanding and patterns in the data. Such a model that prevents the occurrences of predictions with a random chance is trustable most of the time.

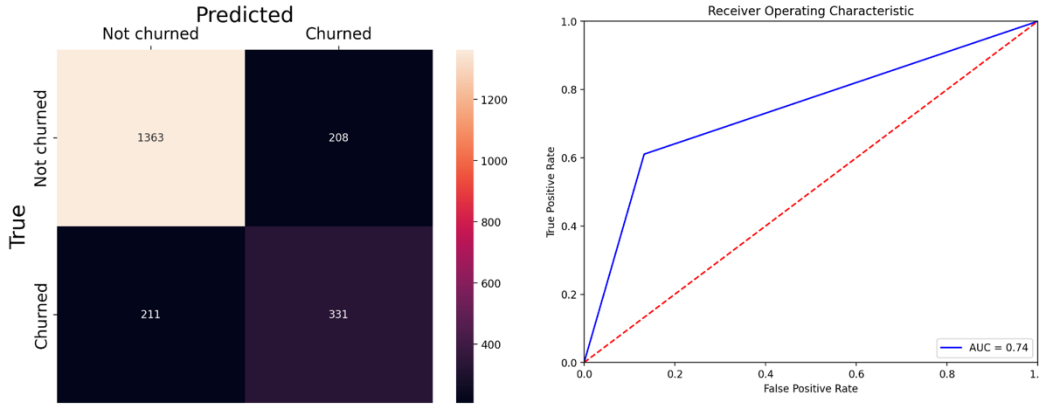
XGBoost is a good option for unbalanced datasets, but we cannot trust random forest in

FIGURES:



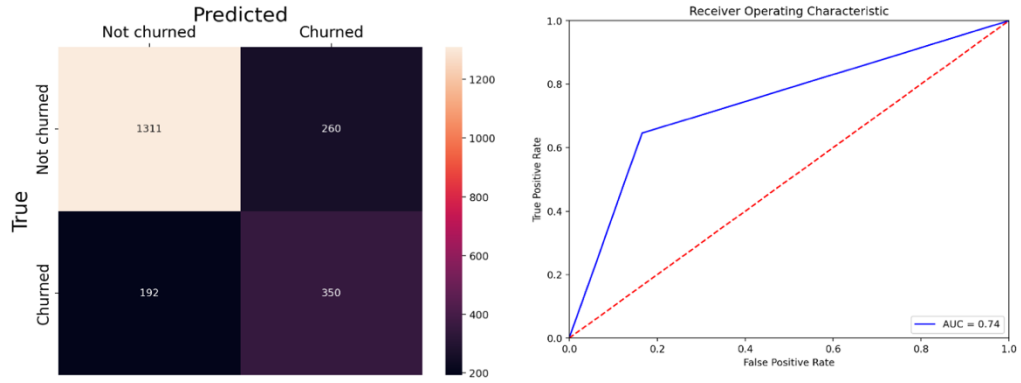
Model trained with an F1 score of `0.80034`

Figure 13: XGBoost + Feature Engineering + Scaling



Model trained with an F1 score of `0.80152`

Figure 14: Ensemble + Feature Engineering + SMOTE



Model trained with an F1 score of `0.79003`

Figure 15: ANN + Feature Engineering + SMOTE + Scaling

Model	Feature Engineering	Feature Engineering + Scaling	Feature Engineering + SMOTE	Feature Engineering + SMOTE + Scaling	Feature Engineering + SMOTE + Scaling + PCA
LR	0.766	0.7645	0.77	0.77	0.7784
SVM	0.77	0.7977	0.753	0.78	0.7833
Random Forest	0.795	0.789	0.78	0.7934	0.769
KNN	0.77	0.783	0.72	0.71381	0.744
XGBoost	**0.8**	**0.8033**	**0.799**	0.79489	0.773
LightBoost	0.763	0.76016	0.758	0.7562	0.69
Ensemble	0.8011	0.8012	0.7855	**0.7993**	0.787
Neural Network	0.79878	0.7928	0.786	0.798	**0.792**

Table 1: Comparative analysis of ML models

these types of cases. In my dataset, the classes are almost highly imbalanced where the number of authentic churns will be huge when compared with unauthentic churns. In XGBoost, when the model fails to predict the anomaly for the first time, it gives more preferences and weightage to it in the upcoming iterations thereby increasing its ability to predict the class with low participation, but we cannot assure that random forest will treat the class imbalance with a proper process and hence, XGBoost performed better than Random Forest in this case.

One of the things I could work on is parameter tuning - fine-tuning the parameters more-specifically, neural networks might give an F1 score of around 0.85.

Based on my analysis, I had presumed that ensemble learning methods would out-perform the traditional models. As expected, the ensemble algorithms, Random Forest and XGBoost performed much better than the singular ML models such as Logistic regression which was my baseline model and the SVM classifier.

III. CONCLUSIONS

It has become known that predicting churn is one of the most important sources of income for telecom companies. Hence, this project aimed to build a system that predicts the churn of customers in a telecom company.

Throughout the project, I tried various permutations and combinations for the input preprocessing steps to the model as can be seen in *Table 1*. Then I have included the best 5 processing steps above. Adding SMOTE as a data preprocessing step did reduce the F1 score for a few models. However, this happens sometimes as adding synthetic data reduces the accuracy of the majority class but increases for the minority class which is the end goal in almost any imbalanced dataset.

I started with logistic regression as my baseline model. Then I proceeded to train SVM classifer, KNN classifer to check the improvement. I also included ensemble learning methods such as Bagging (Random Forest classifier) and Boosting (XGBoost and LightGBM classifiers) to further improve my classification performace. I also trained a Artificial neural network which gave me the best results as highlighted above.

- One thing we observe for LR, SVM, and KNN is that their F1 score changes drastically sometimes upon small changes in the hyperparameters and input preprocessing steps. This made it very difficult to obtain the ideal hyperparameter for these models.
- The heavier models such as XGBoost, Neural Networks, RandomForest, and Ensemble model did take a lot of time during the hyperparameter tuning. However, they gave consistently good results.
- LightBoost didn’t give such good results as compared to other ensemble learning models, however, the training speed was about twice as quickly. Also, it was fast during the prediction. One application for this model might be when we want some real-time prediction.

- Neural Networks also gave quite good results. I got the best F1 score with a network of merely 2 hidden layers with 128 neurons each and activation function as 'sigmoid'. The model was severely overfitting and was a more complex model.
- The ensemble model was the star of the lot - this is primarily because it helps alleviate mispredictions in a particular model. Upon experimentation, I took it as a combination of LR + SVM + Random Forest + KNN + XGBoost.
- However, its predictions weren't much better than XGBoost and RandomForest perhaps because XGBoost and RandomForest are themselves ensemble models. However, the exclusion of these models from our combined ensemble models gave worse results.

In order to increase the success of our results, what we need is sufficiently large data that is error-prone, along with that the explanation of independent variables should be sufficient. One of the things we can work on is parameter tuning - fine-tuning the parameters more- specifically, neural networks might give us an F1 score of around 0.85. The use of Social Network Analysis features might be really helpful in enhancing results of predicting the churn in telecom. Deploying it on AWS and training on a higher compute environment on SageMaker would allow for more resources that can be used for a more exhaustive parameter tuning.

REFERENCES:

1. https://www.frontiersin.org/articles/10.3389/fmech.2019.00030/full
2. Çelik, Ö., & Osmanoğlu, U.Ö. (2019). Comparing to Techniques Used in Customer Churn Analysis.
3. Ahmad, A.K., Jafar, A., & Aljoumaa, K. (2019). Customer churn prediction in telecom using machine learning in big data platform. *Journal of Big Data, 6*, 1-24.
4. https://mlflow.org/docs/latest/tutorials-and-examples/index.html
5. https://docs.streamlit.io/library/api-reference
6. https://medium.com/@peterworcester_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85
7. https://towardsdatascience.com/an-overview-of-the-pep-8-style-guide-5672459c7682
8. https://www.tensorflow.org/tutorials/structured_data/feature_columns
9. https://scikit-learn.org/stable/modules/ensemble.html
10. https://scikit-learn.org/stable/modules/model_evaluation.html
11. https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
12. https://towardsdatascience.com/getting-started-with-xgboost-in-scikit-learn-f69f5f470a97
13. https://towardsdatascience.com/feature-selection-techniques-1bfab5fe0784
14. https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc#:~:text=If%20your%20dataset%20is%20heavily,and%20communicate%20to%20business%20stakeholders.
15. https://github.com/shuchita28/ML-models/blob/main/kNNFromScratch.py