Customer Relationship Prediction

KDD Cup 2009

Wang Sherpa, Janhavi Kashyap, Harshitha Kolukuluru, Rovina Pinto

Univ.AI

Abstract

The KDD cup 2009 is a marketing problem with the goal of identifying data mining techniques. The task was to build upon Customer Relationship Management (CRM) which is a key element of modern marketing strategies. The large marketing databases from the French Telecom company Orange had three main targets: to predict the propensity of customers to switch provider (churn), buy new products or sale services (appetency), or buy upgrades or add-ons proposed to them to make the sales more profitable (up-selling).

The dataset consisted of 50000 values in the small dataset. It consisted of a subset of 230 variables, 40 of which were categorical. The key challenges in this dataset were its heterogeneity, large missing values and a large number of unique categories for each categorical variable. The ROC AUC score was used as the scoring parameter to test all models against.

There were three baseline models constructed: Logistic Regression with Lasso Regularization, Decision Tree with Variance Threshold and Gaussian Naive Bayes.E each of which had its own advantages and disadvantages with respect to the classification problem at hand. LeaveOneOut encoding was used along with standard scaling and SMOTE for over sampling for models which required them.

The models that we then worked on using hyper parameter tuning were largely ensemble models with the exception of neural networks with entity embedding. The model with the highest AUC score of 0.80 for all three parameters was a stochastic gradient boosting method called XGBoost. The model was tuned with frequency encoding for only the top 10 categories- a method similar to the winners of the actual challenge. The model was designed to be cost sensitive using a weighting parameter. No up sampling was done. The effectiveness of the model hence is evident as it is but a simple boost model with powerful capabilities.

Chapter 1: Pre-processing

The dataset posed many challenges:
Heterogeneous data (numerical and categorical variables), noisy data, unbalanced distributions of predictive variables, sparse target values (only 1 to 7 percent of the examples belong to the positive class) and many missing values.

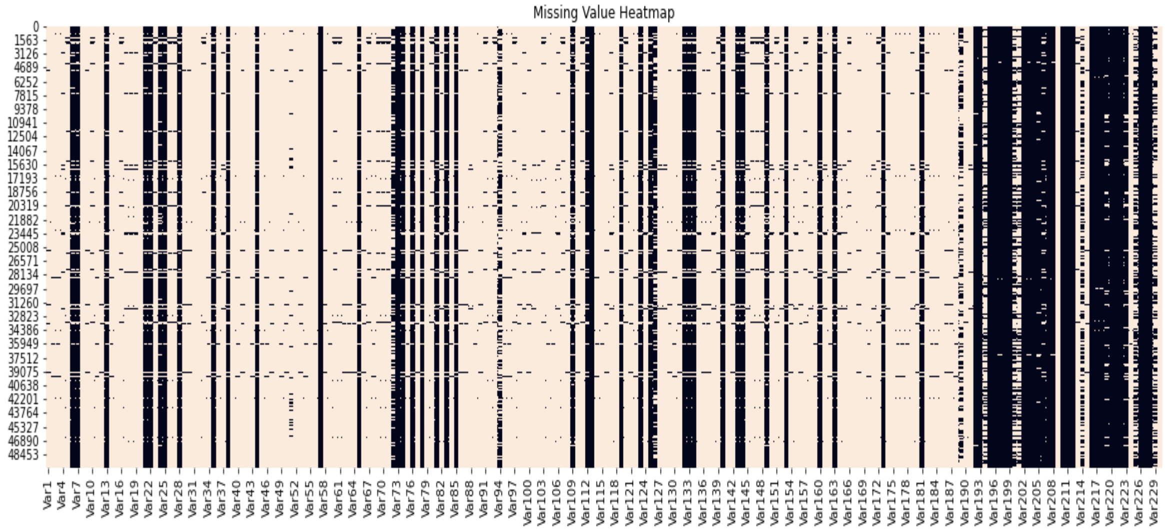


Figure 1: Heatmap showing missing values

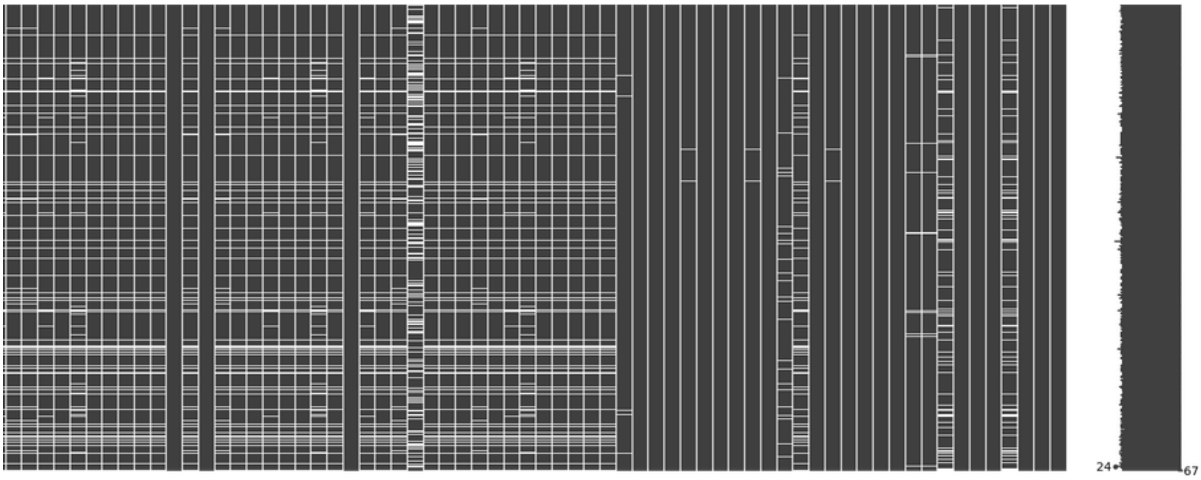


Figure 2: Heatmap after removing features with more than 70 percent missing data

This left us with 39 numerical features and 28 categorical features. We then dropped 6 categorical features which had over 1000 unique categories within it.

We also had high collinearity with some of our features. We applied a threshold to get rid of 0.95 Pearsonsr correlation score and wanted to test how the tree models would handle the others.

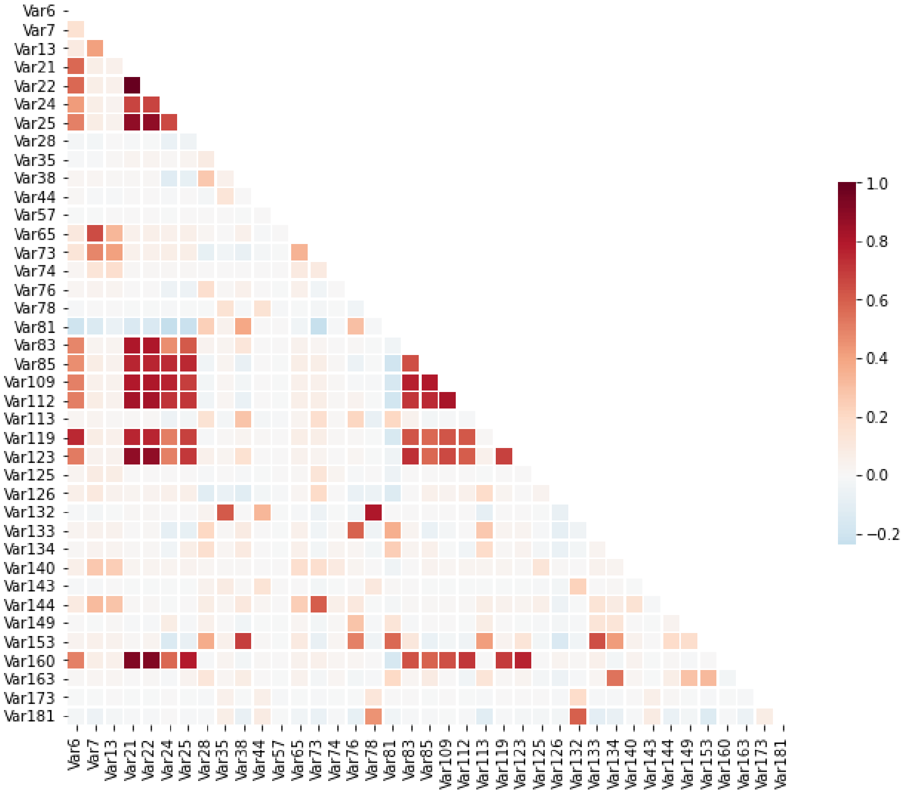


Figure 3: Correlation plot for the features

After this we proceeded to do some feature engineering wherein we encoded all the occurrences of NaN values in ows with 1s or 0s. This was done so that we could pick up on possible relationships with the presence or absence of certain values with respect to other features. This left us with 98 numerical and 22 categorical features
We then used soft imputing: a method of matrix completion by iterative soft thresholding of SVD decompositions for numerical features and imputed a category ‘missing’ for NaN values in categorical features.

We then looked at the target values. The dataset shows highly imbalanced class distribution with the positive class taking the minority role. We used SMOTE from imblearn in the baseline models to address this issue.

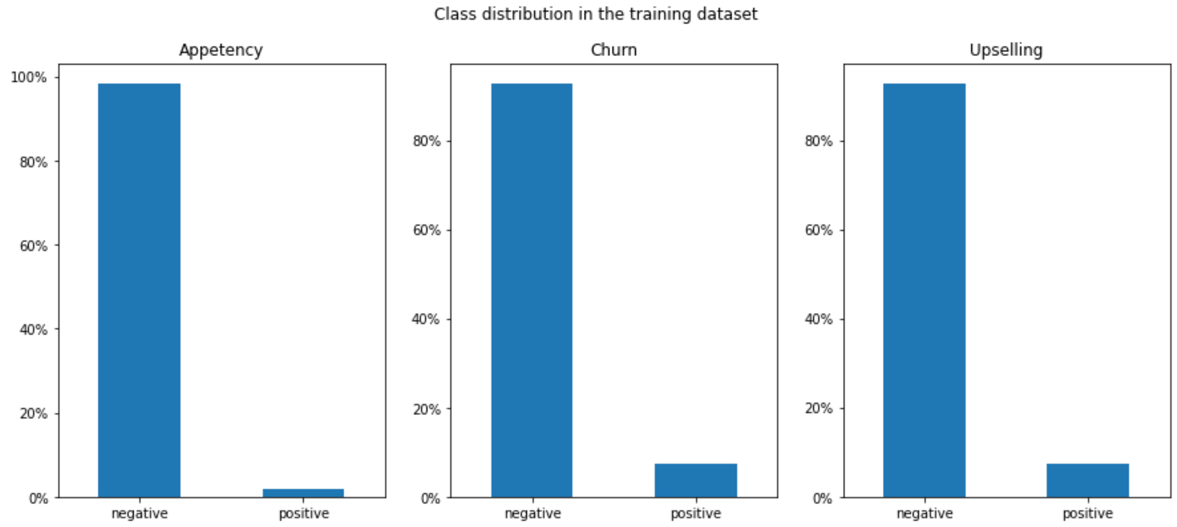


Figure 4: Targets showing high imbalance of the classes

Chapter 2: Baseline Models

Before we built our models, we constructed three baseline models as a benchmark we wanted to beat. The three models were chosen based on their structural differences. We used Logistic regression with Lasso regularization, LogReg, Decision Trees, dtree and Gaussian Naive Bayes, GaussNB.

Standardization was done for LogReg and GaussNB as the data we had was non-normal with extreme values. dtrees are not affected by monotonic transformations and hence we did not standardize our data for it. We used LeaveOneOut encoding on the dataset which is very similar to target encoding but excludes the current row’s target when calculating the mean target for a level to reduce the effect of outliers. SMOTE was used for LogReg and dtree and not for the simple bayes model as we wanted to keep it as heuristic as possible and also wanted to test how the model would handle the imbalance. Undoubtedly dtree performed the best as it is better at handling the classification problem at hand while GaussNB did the worst which is still not so bad as it did not have any hyper parameter tuning. Therefore the baseline model to definitely beat is the GaussNB with dtree being the model to aspire to.

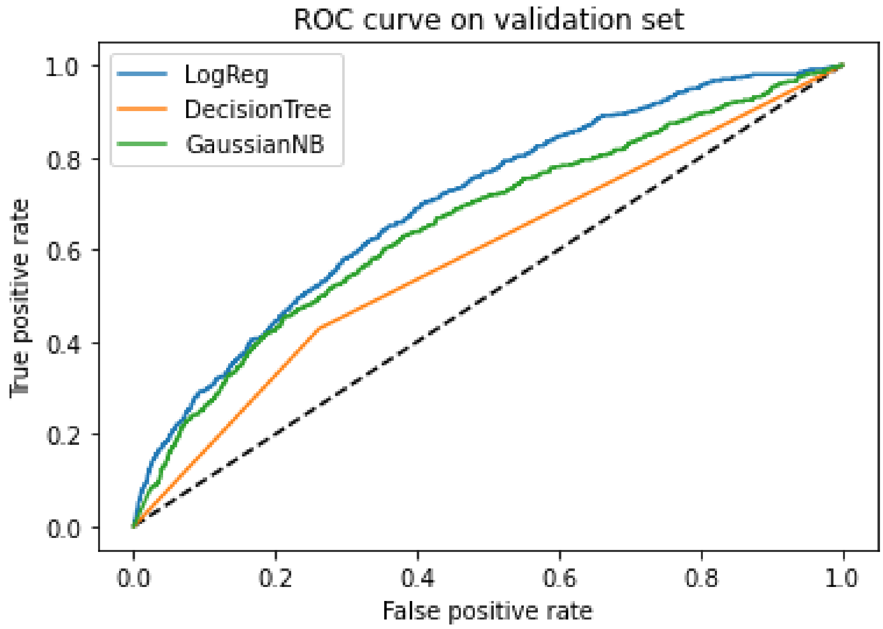
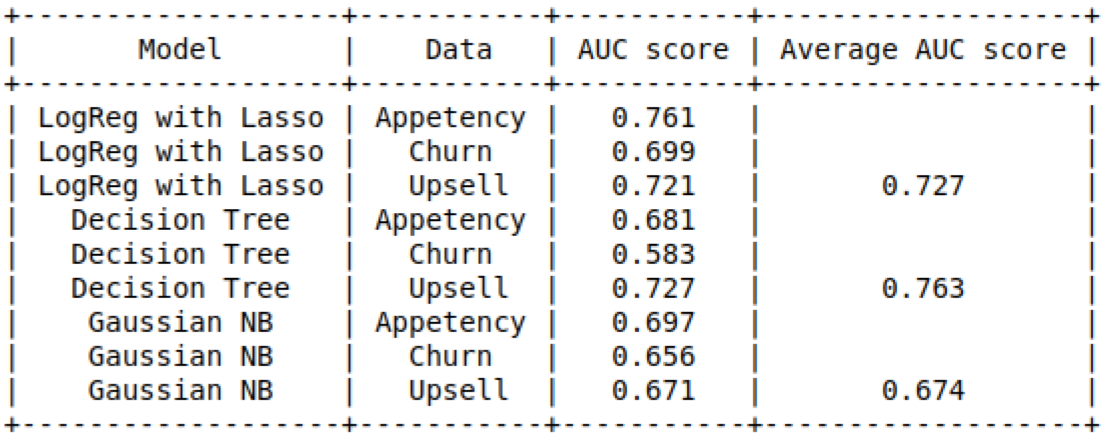


Figure 5: ROC curves for the three baseline models



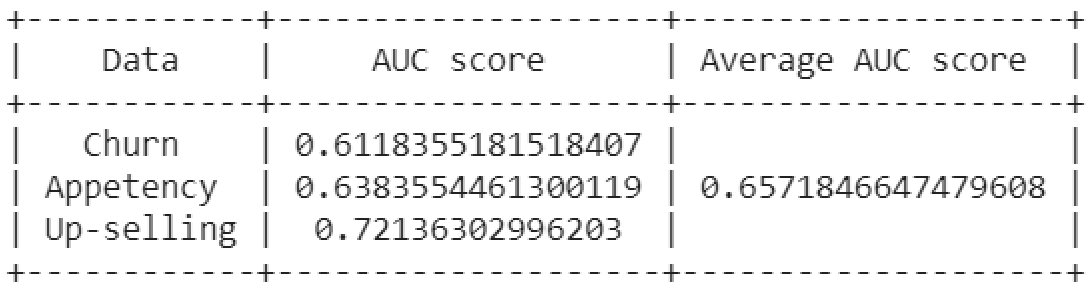
```
+------------------+----------+----------+------------------+
|       Model      |   Data   | AUC score| Average AUC score|
+------------------+----------+----------+------------------+
| LogReg with Lasso| Appetency|   0.761  |                  |
| LogReg with Lasso|   Churn  |   0.699  |                  |
| LogReg with Lasso|  Upsell  |   0.721  |       0.727      |
|   Decision Tree  | Appetency|   0.681  |                  |
|   Decision Tree  |   Churn  |   0.583  |                  |
|   Decision Tree  |  Upsell  |   0.727  |       0.763      |
|    Gaussian NB   | Appetency|   0.697  |                  |
|    Gaussian NB   |   Churn  |   0.656  |                  |
|    Gaussian NB   |  Upsell  |   0.671  |       0.674      |
+------------------+----------+----------+------------------+
```

Chapter 3: Classification Algorithm

There were four classification algorithms that were employed- three of them being ensemble models and the fourth, a neural network with entity embedding. They all used the same pre-processed data with soft imputation while being at the liberty of different encoding methods to apply and the choice of up-down sampling and further feature engineering and selection. The text that follows below gives a brief overview of the hyper-paramter tunings of each of the algorithms along with their results.

3.1 Random Forest

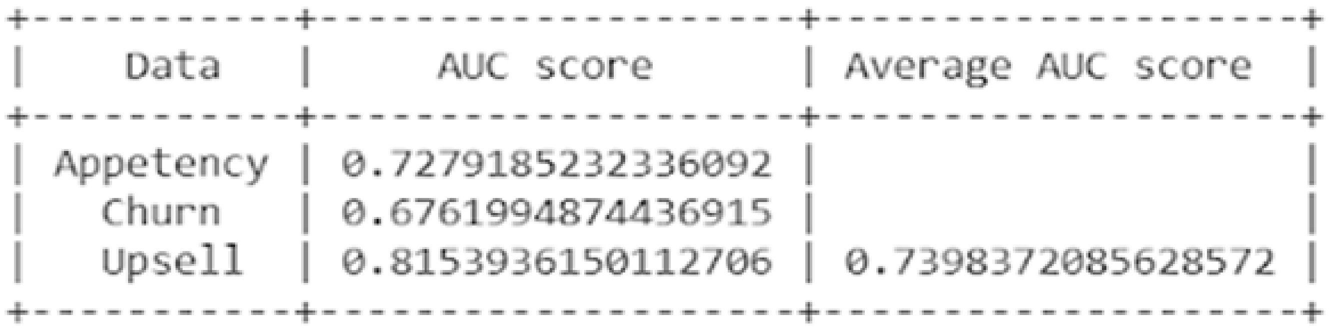
Scikit-learn's RandomForestClassifier() was used with tuned hyperparameters and class_weights set to 'balanced subsample' to deal with class imbalance. In addition, SMOTE was used to oversample data from the minority class. The model was trained with the soft-imputed data and only the top 10 categorical variables were used after binary encoding. We noticed that we got improved accuracy and f1 scores after the data was scaled which should not have happened as the algorithm takes into account only the ranks in the features. A reason why scaling could have changed our results is that we performed feature engineering to see if the occurrence of NaN values across the columns had any patterns and these were binary values. Scaling helped the model as both numerical columns and the indicators were scaled down to the same level. The model was not very effective in dealing with class imbalance and in fact does worse than the baseline logistic regression model while predicting churn.



```
+------------+--------------------+--------------------+
|    Data    |     AUC score      | Average AUC score  |
+------------+--------------------+--------------------+
|   Churn    | 0.6118355181518407 |                    |
| Appetency  | 0.6383554461300119 | 0.6571846647479608 |
| Up-selling |  0.72136302996203  |                    |
+------------+--------------------+--------------------+
```

3.2 Gradient Boosting

As GradientBoostingClassifiers often gave good results, we used it to train our data. First we performed a RandomSearchCV to find the best parameters for the classifier and then trained our data on the best classifier. We used StandardScalar to scale the numerical data as the model seemed to give better results for standardised data. We also used SMOTE for upsampling as there was a huge imbalance between the target classes. After completing the training process, we evaluated the models of the test data and their results were obtained as follows:



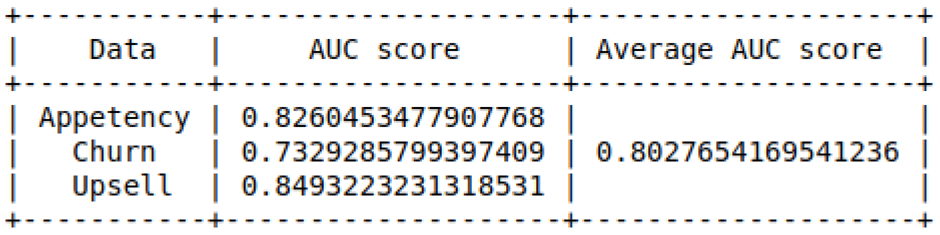
```
+-----------+--------------------+--------------------+
|    Data   |     AUC score      | Average AUC score  |
+-----------+--------------------+--------------------+
| Appetency | 0.7279185232336092 |                    |
|   Churn   | 0.6761994874436915 |                    |
|   Upsell  | 0.8153936150112706 | 0.7398372085628572 |
+-----------+--------------------+--------------------+
```

- Churn: AUC(0.68), F1_score(0.19)
- Appetency: AUC(0.72),F1_score(0.10)
- Upselling: AUC(0.81),F1_score(0.38)

We see that the results were not very different as compared to the RandomForest and the LogisticRegression model. The model performed comparatively well on Upselling as compared to the other target variables like in any other model due to class imbalance. The average AUC score of the model was around 0.74.

3.3 XGBoost

An implementation of the stochastic gradient boosting algorithm, Extreme Gradient Boost or XGBoost allows for tuning parameters specifically dealing with misclassification of the minority class for datasets with a skewed class distribution. Since the algorithm takes care of imbalanced datasets, no up/down sampling was applied and no standardization either.
The first thing done was to encode the categories for which, like the above two algorithms, the winning team's technique to only utilize the top ten categories and then to frequency encode them.
With this some hyper parameter tuning was done for max depth, gamma and n estimators using RandomizedSearchCV in favour of time. What was the defining tool was the use of Cost sensitivity. The XGBoost algorithm was given a scaling parameter to cost effectively fit on the dataset. This is probably what pushed the model to the next step. XGBoost performed the best out of all the models we built including the baseline.



```
+-----------+--------------------+--------------------+
|    Data   |      AUC score     | Average AUC score  |
+-----------+--------------------+--------------------+
| Appetency | 0.8260453477907768 |                    |
|   Churn   | 0.7329285799397409 | 0.8027654169541236 |
|   Upsell  | 0.8493223231318531 |                    |
+-----------+--------------------+--------------------+
```

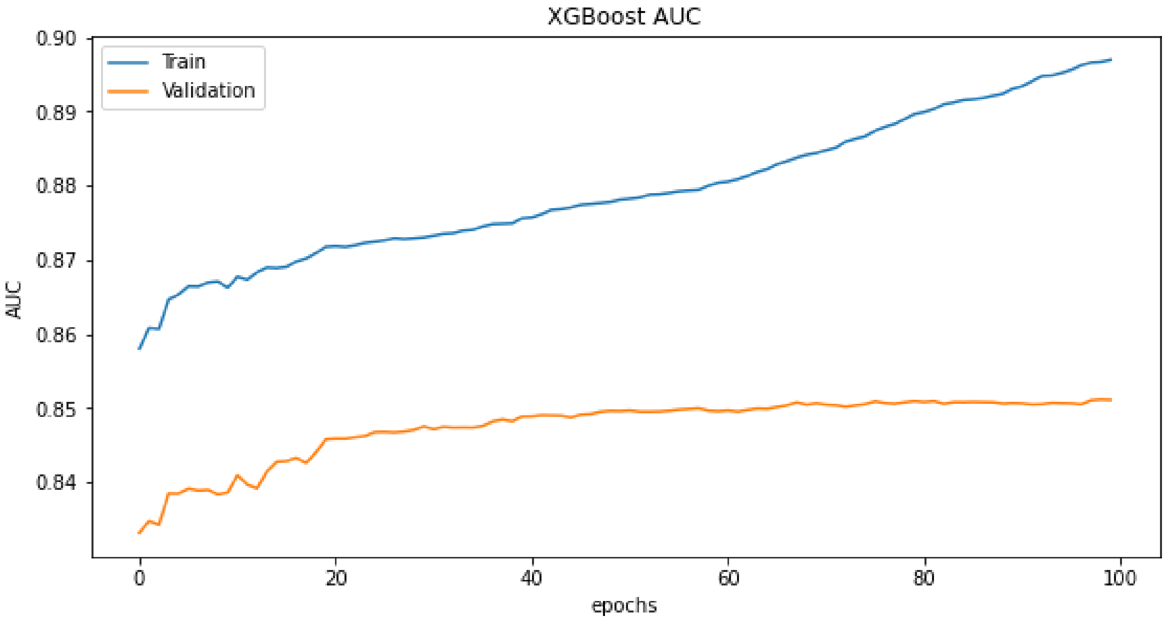


Figure 6: Early stopping being used in the model to prevent the train dataset from overfitting

3.4 Neural Networks with Entity Embedding

As the categorical features in this dataset contain many unique values. We trained a neural network model using entity embeddings. With Entity Embeddings, the input features will benefit from their encoded knowledge, and therefore improve performance. On top of that, assuming compactness of the embeddings, the model itself will require fewer parameters, resulting in faster iteration speed and cost savings in terms of computation during both training and testing.

We trained Entity Embeddings model only using ‘Upselling’ as a target variable and saved the learned embeddings. We later build another three Sequential Neural Networks and trained them using the respective target features(Churn, Appetency & Upselling) and Embedded data (containing both numerical features and categorical as embeddings). After completion of training, we evaluated the models performance on a test set and got the following result:

- Churn: AUC(0.69), F1_score(0.23)

- Appetency: (0.80), F1_score(0.15)
- Upselling: AUC(0.79), F1_score(0.39)

We can see that the results are not excellent, but considering the data we had this AUC score seems fine but the F1 scores are very bad for churn and appetency and a bit better for Upselling, this is due to the severe class imbalance.

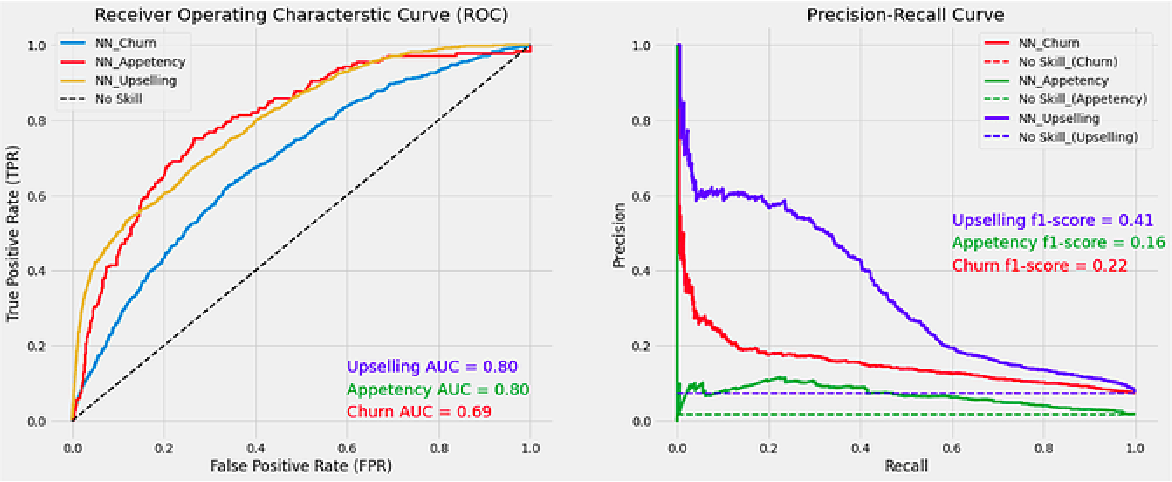
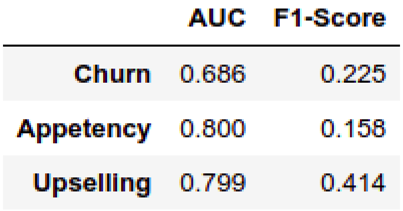


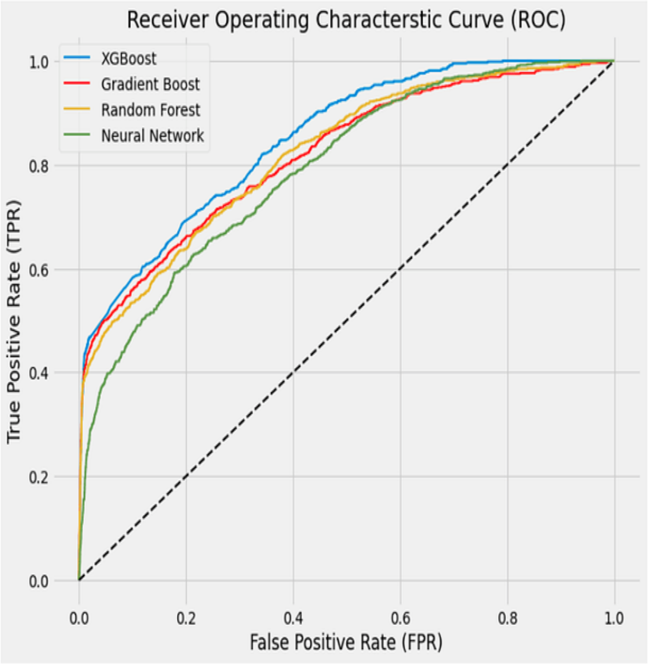
Figure 7: ROC and PR curves for the three different targets in the NN model

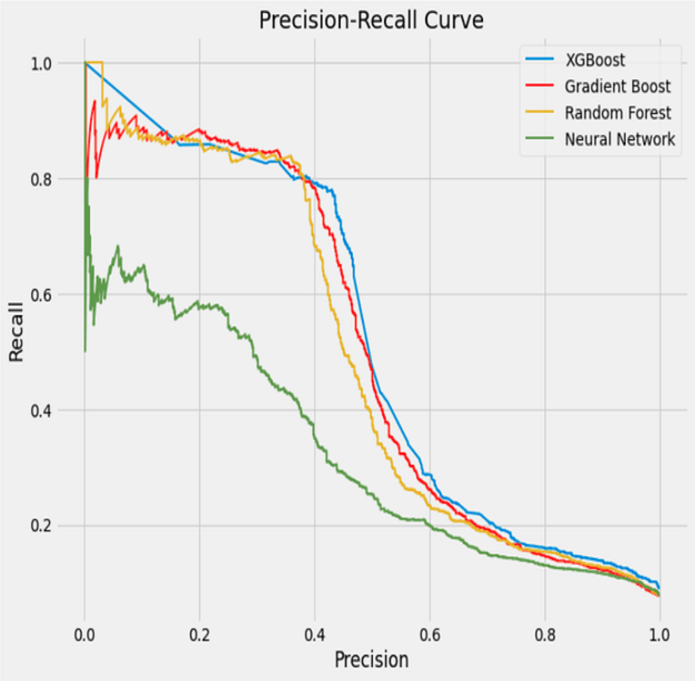


	AUC	F1-Score
Churn	0.686	0.225
Appetency	0.800	0.158
Upselling	0.799	0.414

Chapter 4: Results

From the above classification algorithms, it is evident that Cost weighted XGBoost did the best job on comparing the ROC AUC scores. The score is still lower than the winners' score of 0.84. But it must be noted that they worked on the large dataset which was a lot easier to handle than the smaller one in terms of feature engineering and selection.





Conclusion

We successfully trained three baseline models and 4 classification models on the KDD Cup 2009 challenge dataset. While the dataset was challenging in terms of pre-processing and model hyper parameter tuning, it gave us the opportunity to work on multiple facets of data analysis that one can ask for in just a single dataset. We do believe that several other things could have been done such as discretization of the data as seen in the histogram plots, downsampling of the majority class, rigorous feature selection using backward/forward feature selection and the use of boosting algorithms such as AdaBoost and CatBoost. We also believe that training on the data with just the selected features from feature importance could have given us a better idea about the underlying collinearity and if our models even took care of it. There is much to be done but alas, time is never enough!

References

1. The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009)Challenges in Machine Learning, Volume 3 http://www.mtome.com/Publications/CiML/CiML-v3-book.pdf