1 Introduction [15 points]

- Group members: Etienne Casanova, Ishaan Mantripragada, Jack Myles
- Kaggle team name: Butters
- Ranking on the private leaderboard: tbd
- AUC score on the private leaderboard: tbd
- Colab link: Project Link
- Piazza link: Piazza Post
- Division of labor: All three of us worked together to preprocess the data. We then split up individually to research and write our own models. After writing our own models, we then combined them into an ensemble for our final submission. Overall, each member of the group was involved in all stages of the project.

2 Overview [15 points]

Models and techniques attempted

This project explored a variety of different models and techniques that we have learned throughout the course. Before implementing any models, we first preprocessed the given dataset by vectorizing the inputs. This involved going through each column manually and including numeric values for each possible output (as well as deleting some columns). After doing this, we could begin exploring various models. At first, we implemented a basic logistic regression model to learn and understand the dataset. We then moved onto more advanced techniques such as Random Forest Classifier, XGBoost, SVM classifier, CatBoost, and AdaBoost. For each model, we fine tuned hyperparameters through grid searches to find the optimal configurations. We tried combining the results of each model through an ensemble, giving the various models weights depending on their individual performance, but this did not result in an increase in performance over our best performing model, which was a CatBoost model.

Work timeline

1. We began our project on Monday where we downloaded the project from Piazza and setup our Kaggle accounts and team. We also started to brainstorm various ideas and techniques for how we can approach this problem.

2. On Tuesday, we started working with our data. We determined how we would vectorize our inputs, so that it could be used by our models later on.

3. On Wednesday, we began exploring our various models. At first, we implemented a basic Logistic Regression model together. After creating the basic framework, we then split off individually to explore our own. Etienne worked on Random Forest and XGBoost, Jack worked on AdaBoost and CatBoost, and Ishaan worked on SVM. We also attempted to use a basic Sequential Neural Network but quickly determined that it was not suitable for this type of project.

4. On Thursday, we reconvened to analyze our results. From there, we tried creating an ensemble of all our models while adding weights to each one depending on their individual performance. When it had lower performance than our CatBoost model, we focused on improving the performance of our CatBoost model.

5. On Friday, we worked on the report and cleaning up our Colab Notebook.

3 Approach [20 points]

Data exploration, processing and manipulation

First, we opened the csv files into a pandas dataframe, to ease the data cleanup process. The first thing we had to do was use feature engineering to make the dataframe only contain numeric values, as strings and bools are not able to be easily processed and learned from machine learning models. So, we went through each data attribute and assigned numeric values to string values, while maintaining logical values for each category. In other words, each value had to be assigned a value which made sense given the other assignments. For instance, the "StreamingTV" was {Yes, No, No internet service} and we turned those values into numbers, ending with: {Yes: 2, No: 1, No internet service: 0}, that way a model could learn what these values mean, since Yes and No internet service are the two extremes. We did this for every category and created a function which would turn any dataset in the given format into a pandas dataframe with only numeric values. This was done using list comprehensions and dictionaries to turn every value in the csv file into the correct format in the correct column. Then, we had to ensure that there were no Nan values in the data, so we just filled all those values with -1 using the pandas .isna() function. We also normalized our data using z-score standard normalization. Z-score normalization transforms the data to have a mean (average) of 0 and a standard deviation of 1. This is helpful because many machine learning algorithms assume that all features are centered around zero and have a similar variance. Also, z-score normalization deals with outliers, so our data is less sensitive to outliers compared to the raw data. After settling on our final model, we calculated the feature importances (covered further in the conclusion) and removed the three lowest, so they did not have any impact on the training of the model, while we also made sure two of these values showed equal probabilities of being discontinued in each category (graphs section shows this). This helped reduce the model's complexity, reducing the risk of overfitting (to irrelevant values) and slightly improving the model's performance. We also removed the customer ID feature because it is irrelevant to the customer's decision to cancel the service.

Details of models and techniques

We tried Logistic Regression, Random Forest Classifier, XGBoost, SVM classifier, CatBoost, and AdaBoost algorithms in order to try and get the best possible validation score before we attempted the submission. At first we used an 80-20 split of the training/valid data, and then moved to 5 fold validation. We used this training-validation split as it would give us a good idea of where our model was at, and of course 5-fold is a much better representation of the data as opposed to a normal train-test split. We tried logistic regression as our baseline, but it did not perform well, so we kept moving to different algorithms/models which we thought might perform well on classifications problems such as ours. The advantages of using XGboost, CatBoost and Adaboost and other boosting models (with weak classifiers) is that the combinations of their classifiers allows for high accuracy without overfitting (as our training data is pretty simple, and not too large). CatBoost likely performed the best overall because it is specifically designed to handle categorical variables directly, which is exactly what our training set has. We also tried using multiple models to create an ensemble, which would in theory help our final accuracy by smoothing out the differences between models, but it ended up giving very similar validation accuracy. As discussed later, we used grid search to fine tune the hyperparameters of each of these models, then compared them to one another before choosing the best model. We also tried using a neural network, but came to the conclusion that it was overkill and would result in overfitting.

4 Model Selection [20 points]

Scoring

For all of our models we created a manual grid search to test out various hyper parameters and find the ones that resulted in optimal performance for each individual model. These parameters were then used for the final models, which we originally intended to ensemble together, but were unable to successfully improve the performance of our best CatBoost model. We determined which models we wanted to use based on their performances on the validation set, as well as their scores when submitting to Kaggle. Models that performed very poorly, such as SVM, were not pursued further. For XGBoost, we tested various different optimization objectives, including hinge loss for binary regression, regression with squared loss, logistic regression for binary classification, and regression with squared log loss. We chose regression with squared log loss because it resulted in the best performance, although we had expected logistic regression for binary classification to perform best since it seemed to best suit the model's function (log loss for binary regression was a close second and probably could have been used). For CatBoost, we chose log loss because the model was built for a binary classification task, and log loss is good for binary classification and penalizes confident but incorrect predictions more heavily. When choosing the features for each model, we used AUC as our evaluation metric and compared the accuracies of each model based on their ROC AUC scores. The model chosen features of the final model had the best ROC AUC score. We chose to use this scoring method because it was the same one used on Kaggle. ROC AUC scoring evaluates the performance of a binary classification model by calculating the area under the Receiver Operating Characteristic (ROC) curve. This curve plots the true positive rate against the false positive rate at various threshold settings. The AUC (Area Under the Curve) represents a measure of how well the model distinguishes between the two classes (positive and negative). CatBoost consistently had the best ROC AUC scores during testing, although XGBoost was a close second.

Validation and test

We split our initial training dataset into a training set (80%) and a validation set (20%), using sklearn's train_test_split. Our validation set was completely hidden from the models until testing which gave us a good understanding of how our model would perform on the final testing dataset. During our grid search we used 2 fold cross validation and 5 fold cross validation for testing. 2 fold was used when there were a large number of parameters we were varying to reduce runtime. Once we had narrowed the number of parameters, we utilized 5 fold cross validation to ensure the selection of the optimal parameters.

5 Conclusion [20 points]

Insights

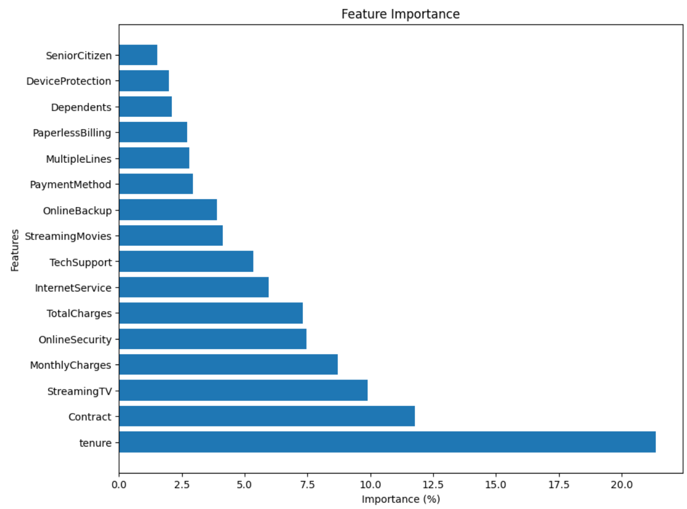
Throughout the course of this project, we gained many valuable insights into how different machine learning algorithims can tackle a basic classification problem. In particular, CatBoost, XGBoost, and AdaBoost performed the best when compared to Random Forests and SVMs. For each algorithm, we explored a variety of hyper-parameters through a large grid search. Though these may be both time and computationally intensive, optimizing these were crucial to improving our base predictions. We explored the feature importance for our CatBoost model because it had the best performance. We created a table ranking the features by their importance using catboost's built-in get_feature_importance method, which shows how much on average the prediction changes if the feature value changes. The top 10 features for our best performing model were: {Tenure, Contract, StreamingTV, MonthlyCharges, OnlineSecurity, TotalCharges, InternetService, TechSupport StreamingMovies, and OnlineBackup}. Other models, such as XGBoost or even CatBoost models with varied parameters, could result in slightly different top features. However, throughout our testing the most important parameters remained relatively consistent.

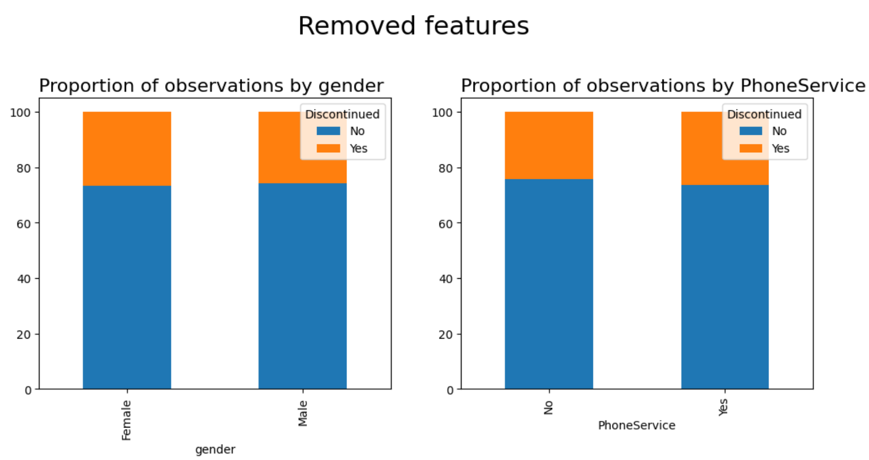
Overall, we learned how to test numerous different models, using various validation and scoring techniques to determine which model was best suited for the task. We saw how data processing plays an important role in training models by filtering out unnecessary entries that could disrupt learning, as well as normalizing data to prevent a huge impact from outliers, thus enabling them to correctly learn the patterns/true distribution. We also found out how grid search can be used for hyper-parameter tuning to optimize performance.

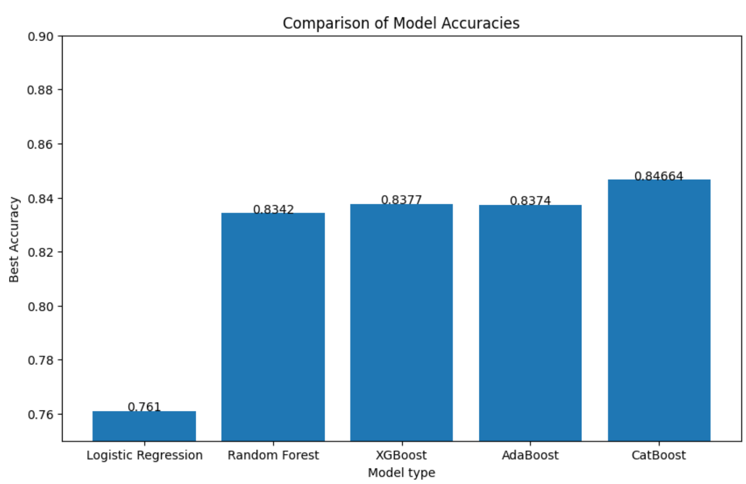
Challenges

Given our 10 submissions per day limit, we had to be strategic with which models we wanted to ensemble and upload. With more submissions, we could have tried different combinations of ensembles to see which performed the best. Our largest struggle of all was certainly finding the best model to use. We started off trying logistic regression and random forests, which were models we were very comfortable with, but these didn't work as well as we hoped. We then tried gradient boosting models which worked much better, but the fine tuning of these models with grid search was very tedious and we did not get the level of results we expected for the amount of hyper-parameter tuning which we did. Also, our ensembles seemed to just average out the testing scores found between models instead of smoothing out the inconsistencies between them, which was slightly disappointing. We also found it slightly difficult to decide which features to remove from the data in the processing, and we are still unsure if our data processing was optimal.

6 Data Visualization







7 Extra Credit [5 points]

• Why do we use AUC as our Kaggle competition metric? Do you think there is a better metric for this project? Why, or why not?

We tested both AUC and LogLoss eval_metric configurations. AUC values range from 0 to 1. A model with an AUC of 0.5 is no better than random guessing, while an AUC of 1.0 indicates perfect prediction. A higher AUC indicates a better performing model. AUC is scale-invariant, meaning it measures how well predictions are ranked rather than their absolute values. It's also threshold-invariant, evaluating the model's ability to distinguish between classes regardless of the specific classification threshold. Log loss values range from 0 to ∞. A perfectly calibrated model has a log loss of 0. Lower values indicate better performance and higher values indicate worse performance, with the log loss increasing as the predicted probability diverges from the actual label. Log loss is sensitive to the predicted probabilities and thus provides a more nuanced view of the model's performance. It directly corresponds to the confidence of the predictions, penalizing false classifications based on how erroneous they are. Since AUC is more focused on the model's ability to rank positive cases higher than negative cases across all possible thresholds, we determined that AUC was more effective.

• Among the machine learning methods that your group used, are there any methods or pipelines that can be parallelized? If so, how? If not, why not? (You are not required to actually parallelize your code)

Some of the boosting algorithms themselves such as XGBoost and CatBoost utilize parallel tree boosting which are useful for large datasets. Another parallelization technique that we could have explored was parallel batch processing during our preprocessing phase. This technique splits the dataset into smaller batches and processing them in parallel which is useful when the dataset is very large. In our case however, the dataset was not big enough to warrant batch processing.

• What additional data would be useful to collect for this problem, and why?

It could be useful to have exact information on the person's age rather than if they are just a senior citizen, as that gives more information from the model to choose from. This could be important as younger people have very different lifestyles. As a result, they could be less financially stable and thus be more likely to discontinue the service, for example. It could also be useful to understand the frequency of which the customer uses the service (which may not be too hard to find), and also how often the user calls customer support to get help or complain about the service. These frequency values may be useful as they give us more of a sense of how much the customer enjoys/likes the product as well as how often they use it. Customers who use the product frequently and like it are less likely to discontinue the service. There is also a lot of other data which could be of use (but may be hard to find), such as the customer's financial stability (through asking about their income).

8 LLM Usage

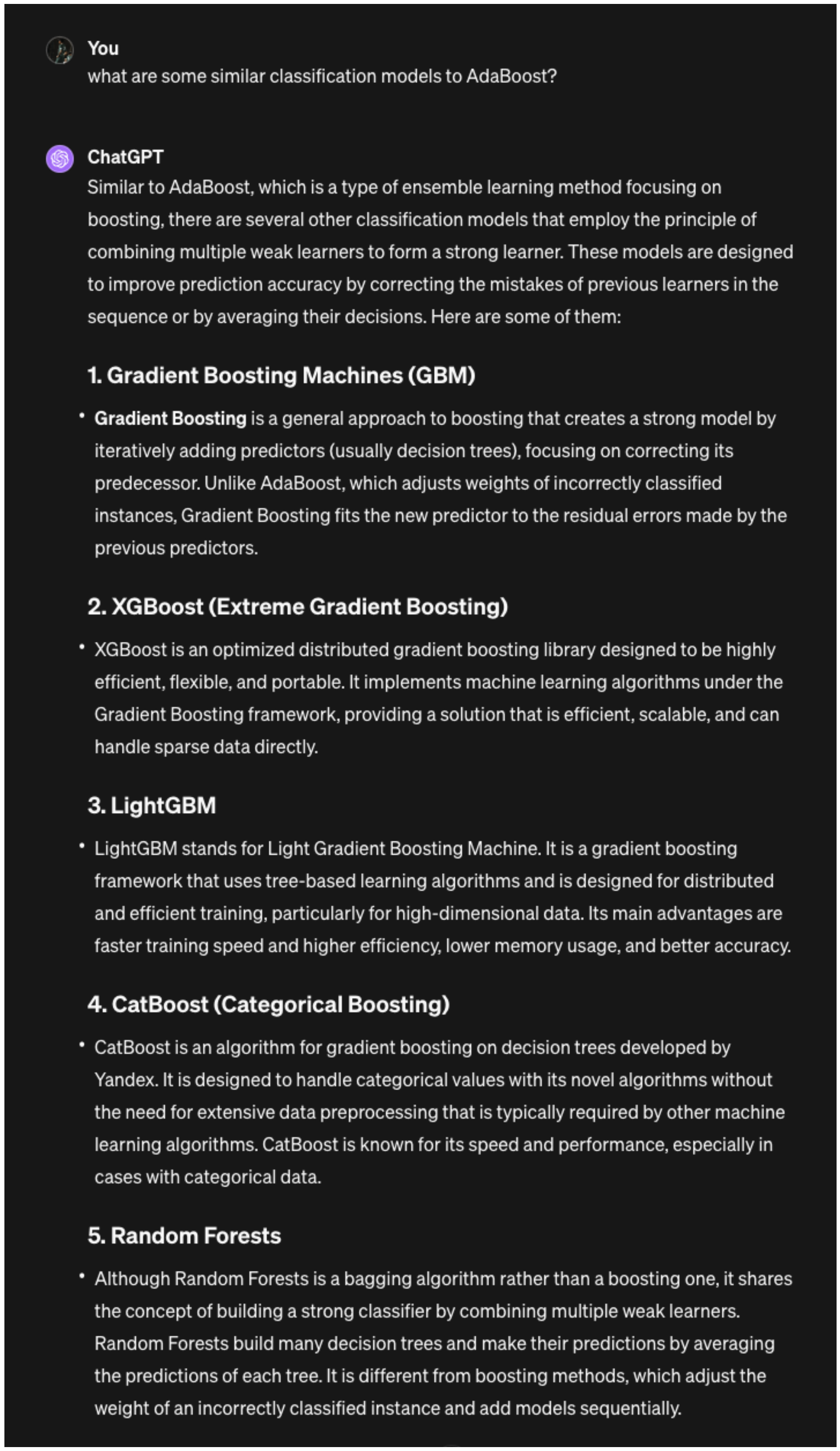
This section has an example of LLM usage reporting below. Please follow this format to report LLM usage. Indicate each usage clearly.

- Name of LLM(s) Used: ChatGPT
- Components of Project involving LLM: Model Brainstorm.

Model Brainstorm

Usage/Integration of Responses

After implementing the AdaBoost model and achieving promising results, we were curious as to what other models might perform similarly. As shown in Figure 1, we asked ChatGPT about similar models to AdaBoost which is where we had the ideas for XGBoost and CatBoost.



You
what are some similar classification models to AdaBoost?

ChatGPT
Similar to AdaBoost, which is a type of ensemble learning method focusing on boosting, there are several other classification models that employ the principle of combining multiple weak learners to form a strong learner. These models are designed to improve prediction accuracy by correcting the mistakes of previous learners in the sequence or by averaging their decisions. Here are some of them:

1. Gradient Boosting Machines (GBM)

- **Gradient Boosting** is a general approach to boosting that creates a strong model by iteratively adding predictors (usually decision trees), focusing on correcting its predecessor. Unlike AdaBoost, which adjusts weights of incorrectly classified instances, Gradient Boosting fits the new predictor to the residual errors made by the previous predictors.

2. XGBoost (Extreme Gradient Boosting)

- XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework, providing a solution that is efficient, scalable, and can handle sparse data directly.

3. LightGBM

- LightGBM stands for Light Gradient Boosting Machine. It is a gradient boosting framework that uses tree-based learning algorithms and is designed for distributed and efficient training, particularly for high-dimensional data. Its main advantages are faster training speed and higher efficiency, lower memory usage, and better accuracy.

4. CatBoost (Categorical Boosting)

- CatBoost is an algorithm for gradient boosting on decision trees developed by Yandex. It is designed to handle categorical values with its novel algorithms without the need for extensive data preprocessing that is typically required by other machine learning algorithms. CatBoost is known for its speed and performance, especially in cases with categorical data.

5. Random Forests

- Although Random Forests is a bagging algorithm rather than a boosting one, it shares the concept of building a strong classifier by combining multiple weak learners. Random Forests build many decision trees and make their predictions by averaging the predictions of each tree. It is different from boosting methods, which adjust the weight of an incorrectly classified instance and add models sequentially.

Figure 1: Interaction with LLM for model brainstorming.