Chinmay Raut(93345289), Junlin Wang(12858769), Nikita Patel(34811375)
Team name: easy_dp

CS 178 Final Project Report

1. **Table for all the models**

Model Name	Training (AUC)	2-Fold Cross Valid. (AUC)	Leaderboard (AUC)
KNN	0.8007397692082429	0.6485080640265539	0.44900
Random Forest[RF]	0.9610394363267769	0.7465017174427219	0.67118
Logistic Regression[LR]	0.6377750879233692	0.6375993757565068	0.57977
Ada Boost[AB]	0.6813391358395241	0.6777436738486395	0.60173
Gradient Boost[GB]	0.7182580375671439	0.7055742805048536	Didn't submit
XG Boost[XGB] (stacked using above models as input)	0.97	0.96	0.74041(6 features) 0.74165 (120 features)

2. **Further details on each first level model:**

For all the 1st level models we only gave each one only 6 of the 14 features: x_0, x_2, x_3, x_7, x_{10}, x_{11} (assuming 0 based indexing, so 1st column is x_0, etc...). We decided on these 6 features after looking at the mean and variances of the training data and test data, the pair plots for the original 14 features, and a heat map for the original 14 features. We used the mean and variances of the training data compared to the test data to eliminate features which varied vastly from training to test data as these could be features that could induce bias into our model. We used the heatmap and pair plots to eliminate features that were highly correlated to each other as these were features that would end up becoming redundant in our model. Because if these features are highly correlated to each other, then technically you would only need one of those features to predict the value of the other features. We used the seaborn libraries to generate the pairplot and heatmap and the numpy functions to get the mean and variance of each feature. We also tested the model using all the data and all the second-order combinations (x_0^2, x_0x_1, x_0x_2, etc...) of the features, but we did not see a worthwhile improvement when we used all 120 second order features or just the 6 features I mentioned at the beginning, so we settled on just the simpler model with far fewer features because it was also faster to train and run. All the models output a probability instead of a hard class decision because that lowered the AUC and our models weren't anywhere near perfect.

a. **KNN (from sklearn.neighbors):**

We used KNN because it was a simple algorithm, and it didn't seem like there is any harm in trying it. In KNN you look at the nearest k points according to the features and return the probability that the given point is 1 based on the results of the k nearest points. We tuned the hyperparameters by using the AUC for the ROC curve obtained through cross-validation. We set the # of neighbors to be 8 and the euclidean distance as measure of distance.

b. **Random Forest (from sklearn.ensemble):**

We decided to use RandomForest as it was a good general purpose classification algorithm for datasets with large variance. The RandomForest Algorithm makes a bunch of decision trees on random partitions of the data and average the predictions of each tree. We tuned the hyperparameters by maximizing the AUC for the ROC curve obtained during cross-validation. We set the maxDepth to 23 and the rest of the parameters were left to their defaults (maximum number of nodes to be considered a leaf was 1, and minimum number of nodes to split on was 2).

c. **Logistic Regression (from sklearn):**

We decided to use logistic regression as it was version of linear regression that naturally produced probabilities due to the logit link function present in the model, and we wanted to try a form of linear regression. Logistic Regression constructs a hyperplane using the logit link function(like a logistic line) instead of the standard identity link function(like a straight line) to classify the points according to the features based on whether they are above or

below the hyperplane. Gradient descent is used to find the optimum parameter coefficients. We set the maxiterations to 200 to make sure the model training completed and it didn't get stuck anywhere during gradient descent as this dataset is not perfectly separable. We did play around with the feature selection for the second order parameters a bit more for this model, but we quickly realized that it didn't drastically improve the regression so we decided to include only the features we fed into the other 1st level models for simplicity.

d. Ada Boost (from sklearn.ensemble):

We tried AdaBoost as our first attempt for boosting. ADA Boost builds many "weak" classifiers and ultimately predicts by taking some weighted linear combination of all the "weak" classifiers. The weight for that classifier is calculated as that weak classifier was made and incorporated into the model. We changed the hyperparameters based on the auc produced by the roc curve produced in cross validation. We set the number of estimators to 500 and the learning rate to be 0.75 to ensure we wouldn't miss the optimum for the model.

e. Gradient Boost (from sklearn.ensemble):

We tried Gradient Boosting as to try to reduce bias in areas we were uncertain what to predict. The GradientBoosting Algorithm makes a bunch of decision tree estimators based off of places the model was predicting poorly and averages the results. We didn't change much of the parameters as the defaults performed pretty well in cross validation, but we set the maximum number of estimators to 500 to reduce the training time for this classifier.

3. How we combined all the 1st level models (from XGBoost)

Each model outputs the prediction of the training dataset. We combined 5 of those outputs into another input for our 2nd level model. We decided to use all 5 because we have this assumption that there may be some information that one model catches but the others don't. So we used all 5 and let XGBoost figure out. Each feature is of this newly combined input is going to be one of the models prediction of the training data. Then we used XGBoost as our second level model. XGBoost stands for extreme gradient boosting, and is a more optimized and scalarable version of gradient boosting. Gradient boost tends to overfit, but XGBoost introduces a more regularized model formalization to control overfitting. It also uses optimization methods to push to the limit of boosted tree algorithms. We chose to use this algorithm because it converges fast, and does not overfit (XGBoost has a regularization term). We chose the "binary:logistic" as our input to the parameter "objective". This problem is a binary classification problem, hence the "binary". And we are using logistic regression as our objective function. The reason for that is because logistic regression returns a probability and we knew that probability does better on AUC score. Number of estimators is the number of decision trees we used. We set it to 2000 because it is the best we found using grid search (more would overfit, and less would be more bias). The max depth and min_child_weight are set to 4, and 2 to mainly control overfit. The decision tree is restricted to have a depth of 4, and it should only split if there are at least 2 nodes in each partition.Gamma is set to 0.9 for the same purpose, since gamma controls the minimum loss reduction required to make a further partition on a leaf node of the tree. The minimum loss reduction has to be a least 0.9 to make a further split on the tree. We set the subsample to 0.8, so that XGBoost randomly selects 80% of the data instances to prevent overfitting. We also set the colsample_bytree to 0.8 to prevent overfitting. So as you can see, a lot of parameters are chosen so that we don't overfit the data and produce a better result.

4. Conclusion

For this project we analyzed 6 classifiers in total. The worst classifier just by looking at the leaderboard performance was KNN. We think this was because this classifier is very prone to overfitting and doesn't handle variances in data very well. The best classifier by the leaderboard performance was the XG Boost. We think this was because it was able to combine all the models we had built and combine their predictions together very well using the good parts from one model to overshadow the bad predictions from another model. What truly surprised us through this study was how the cross-validation we were performing was not very indicative of the score for the test data. Our cross-validation results were always much higher than the AUC for the submissions, which made us believe we were overfitting to the variance in the training data. But cross-validation is meant to combat the overfitting problem as one half of the data of the data is not used while training the model. This led us to believe that the variation present in the training data is not completely representative of the test data.