CS 178: Machine Learning and Data Mining

Project Group 62
Kaggle Leaderboard #: 53
Kaggle Team Name: Group 62

Rachel Leung	40066083	leungrw
Suchita Gupta	94391516	suchitag
Tinko Ng	69446046	tinkon

Project Report

1. Models and Their Performance

Model	Training Error	Validation Error	ROC Score
Linear Regression	0.4262619004850009	0.41594827586206895	0.60228
Polynomial Regression (degree=2)	0.4086581641817856	0.44719827586206895	0.56230
Polynomial Regression (degree=3)	0.4190767019938926	0.4682112068965517	0.54661
K-Nearest Neighbors (k=16)	0.38189329980240705	0.44342672413793105	0.61811
Random Forest (n_estimators=700, min_samples_leaf=30)	0.19310220944853607	0.3556034482758621	0.72037
Gradient Boosting (learning_rate=0.1, n_estimators=70, max_depth=3, max_leaf_nodes=8)	0.24753008801868148	0.3464439655172413	0.73745
Ensemble (1. Gradient Boosting 2. Random Forest 3. Logistic Regression 4. K-Nearest Neighbor)	-	-	0.75291

2. Details About Each Model

a. Regression Models

For the first 3 models listed in the table above, we used the sklearn module to create a pipeline containing PolynomialFeatures objects followed by a Logistic Regression object to easily determine the training and validation errors as well as the ROC/AUC score. We tested this pipeline with varying degrees ([1, 2, 3]) in the Polynomial Feature models and split our input using 75% for training and 25% for validation. Our hypothesis was as the polynomial degree increased, the training error would decrease (possibly leading to overfitting the data) and the validation error would decrease and then increase. However, we found both the training and validation error to increase as we added more features.

b. K-Nearest Neighbors

Next, we attempted to fit the data using the KNN classifier using the sklearn.neighbors's KNeighborsClassifier object. Here we first experimented using varying k values from 1 to 20 and split our input using 75% for training and 25% for validation. We aimed to find an optimal k value where the validation error was the lowest for the entire dataset. Over a few iterations, we found KNN best fit the data with k=16, which had the best validation error rate of 0.443 and training error rate of 0.382. Its ROC score however was only slightly better than the linear regression model.

c. Random Forest

After not having much luck with the Regression and KNN models, we tried using the RandomForestClassifier from sklearn.ensemble. Keeping the consistent 75:25 ratio for training and validation data, we used the RandomForestClassifier with different values for n_estimators=[100, 200,

300 … 1000] and ran 200 iterations on each value of n_estimators. Since each time we went through the list of n_estimators, the validation error varied significantly, the best n_estimator was not constant each time we ran the algorithm. Thus, we kept a list of error values and we normalized the error values by running the algorithm 20 times and taking the average of the errors over the amount of iterations it ran. However, this did little to stabilize the results, so we ran this method of normalizing the errors 10 more times and found that n_estimators=700 gave the lowest validation error. As a separate experiment, we tried RandomForest with different values for min_samples_leaf=[10, 20, 30 … 200] while keeping n_estimators constant. Applying the same process with varying n_estimator values to how we calculated min_samples_leaf, we found the optimal min_samples_leaf value was 30.
Using a combination of the optimal values stated above (n_estimators=700 and min_samples_leaf=30), we discovered a significant improvement in the ROC score.

d. Gradient Boosting

Lastly, we tried using sklearn.ensemble module's GradientBoostingClassifier with varying learning_rate values ([1, 0.5, 0.25, 0.1, 0.05, 0.01]) and found 0.1 gave the lowest validation error. Using this learning rate, we tested with the n_estimators parameter with values [10, 20, 30 … 100] and got the best validation score with n_estimator=70. Then, using both optimal values for learning_rate and n_estimators, we experimented with the parameter, max_depth with values 1 to 10 and with the best score at max_depth=3. The final parameter we tested was max_leaf_nodes with values [2, 4, 6 … 30] and found 8 to be the best value. (Note: when determining the optimal values for these parameters, while the values varied per iteration, the results (ie. best n_estimator) was consistent across all iterations.)
Gradient Boosting produced the best score thus far since its algorithm is based on building and improving on previous, weaker learners. For example, after the first iteration, the next tree worked to correct its errors and build another tree with fewer errors.

3. Overall Prediction Ensemble

Given the parameter optimizations and their respective validation error rates written above, we knew the Gradient Boosting algorithm yielded the best output, followed by the Random Forest algorithm, then Logistic Regression, and finally K-Nearest Neighbor. Based on the ROC values, we decided to add more weight to Gradient Boosting when combining the different classifiers, less weight to the random forest, even less weight to K-Nearest Neighbors, and finally the least weight to Logistic Regression. To implement this, we took the weighted average of Yhat values produced by each classifier. For example, Gradient Boosting's Yhat value was weighted 48%, Random Forest's at 42%, Logistic Regression's at 6%, and KNN's at 4%, which all adds up to be a total of 100%.

4. Conclusion

After testing with gradient boosting, random forest, logistic regression, and K-nearest neighbors and finding the best parameter values for each, we think that the gradient boosting and the random forest worked particularly well for this data due to them having higher ROC score when compared to the logistic regression model and the K-nearest neighbors model. Our hypothesis for why that is the case is because both gradient boosting and random forest are ensembles of decision trees, thus they are more complex algorithms that build their models based off of previous weaker learners to explain the given data when compared to the other two models, therefore giving better results. Lastly, we've found that what's even better than gradient boosting and random forest alone is by ensembling them with the logistic regression and K-nearest neighbors with weights to generate even better results as indicated by the higher ROC score. By taking the predictions from multiple models and using the average of them to make the final prediction, we think that it has effectively lowered the chances of an overfitting model, thus creating better results. However, because some models such as gradient boosting and random forests are more accurate when compared to K-nearest neighbor and logistic regression, we have given them more weight to allow for the usually more accurate models to have more of an impact on the predictions.