Machine Learning Airplane Delay Prediction: Comparison of Single, Bagging, and Boosting Algorithms

Vikas Thoti Reddy*

Abstract

The abstract is a brief (usually one paragraph) summary of the whole paper, including the problem, the method for solving it (when not obvious), the results, and the conclusions suggested or drawn. Do not write the abstract as a hasty afterthought. Look at it as a real exercise in cramming the most information in one paragraph. The reader should not have to read any of the rest of the paper in order to understand the abstract fully. Many readers will read only the abstract. Other readers will use it to decide what to look for in the paper, or to decide whether to read the whole thing.

Keywords: journal, template, latex

1 Introduction

Domestic airplane delays have a significant impact on the US economy and cost billions of dollars each year. In 2007, 31.2 billion dollars was wasted as a result of the domestic flight delays in the US. It has been shown that machine learning algorithms can predict arrival delays in advance; this would allow airport management to adjust schedules, minimize delays, and develop flight delay management procedures. However, previous work in this regard has focused on the utilization of classical classifier machine learning algorithms, more recently a new class of methodology focused on boosting techniques has become available. The purpose of this work is to compare newer, more complex models like random forests and AdaBoost to more classical models like decision trees. Moreover, previous works have used classifier models that predict whether a flight will be delayed or not. In this specific case, simply determining whether an incoming flight will be delayed or not through a binary output will not considerably help airport traffic management. Rather, in this study, regression models were used to predict how delayed a flight will be as a continuous output. A model that predicts accurate arrival delay times as a continuous value will more substantially help airport traffic management than a classifier model. In turn, models focused on continuous outputs will help decrease the amount of money wasted from airline delays by bolstering airport traffic management.

*Email: vikas.thoti@gmail.com.

2 Related Works

There are several previous related works on machine learning models to predict airplane delays. In Kuhn and Jamadagni (2017), the most recent related work, decision tree, simple neural network, and logistic regression classification models were used to predict whether an airplane would have a delayed arrival time. The authors used the definition that, in the US, a delayed arrival time is when an airplane that arrives 15 min after the scheduled arrival time for their classification models. Similarly, the authors used the US Bureau of Transportation Statistics for the domestic airplane data. Kuhn and Jamadagni (2017) subsetted the 5 million samples to 50 thousand flights without arrival delay and 50 thousand flights with arrival delay. Moreover, from the 13 features known in advance of the flight, the authors used a decision tree classifier model to determine the three most important features that affect whether a flight has a delayed arrival. These three features were then used to retrain a new decision tree classifier and train a simple neural network and logistic regression. A ten fold cross validation was used to train the models chosen. These models were tested using 30 thousand samples with an almost equal split between flights with and without arrival delay. The test results showed that all three models, decision tree, simple neural network, and logistic regression, had an f-1 score of 0.91. In contrast, in training the models, this study used cross validation to remove biases from over training and used a greater selection of features and samples to train and test the machine learning algorithms. Moreover, this work uniquely tested more complex versions of the classic decision tree model through bagging and boosting algorithms, and used regression models for continuous arrival delay predictions. Random forest and AdaBoost, examples of bagging and boosting algorithms,

were designed as improvements to the classic decision tree model, but whether these more complicated models work better was explored in this study.

3 Dataset

The dataset used to train and test the machine learning models came from the US Bureau of Transportation Statistics. The dataset contained information about US domestic flights that were scheduled throughout 2015 with many features. Features chosen to train the models were ones that are typically known ahead of the departure of the flight: month, day of the week, airline, origin airport, destination airport, departure time, scheduled departure, departure delay, scheduled time, and scheduled arrival. These features were the parameters that would help the models predict the delay of an airplane. The final feature looked at was the arrival delay of each flight; this variable was the output feature. To clean the data, the flights in the dataset that were cancelled were deleted. A cancelled flight had no substantial information recorded because the flight was cancelled. For the purpose of this work, deleting cancelled flights did not have an adverse effect on the models. Additionally, in the real world, there is no purpose in predicting whether a cancelled flight will have a delayed arrival because it will never reach its destination; inherently, it made sense to delete these flights because without a recorded arrival delay, cancelled flights cannot contribute to model creation.

3.1 Categorical Values Replaced with Dummy Data

The python sklearn library for machine learning regression models can only take in features that can be converted into float format. As a result, they cannot take in non-numerical categorical data as features that contribute to the output variable prediction. In response to this issue, the categorical variables in the dataset, airline, origin airport, and destination airport, were converted into dummy variables. The essential idea behind this concept is to take string values for categorical data and assign numerical values to them. For example, a number was assigned to each airline and replaced the string value representing the specific airline of each flight. Now, categorical data can still be incorporated into the model because the data type has been changed to an integer which can be changed to a float. Even though, to a human, the airline column has been reduced to a bunch of numbers, the sklearn models can now incorporate this feature into its output prediction. Unfortunately, the flights from the month of October had to be removed because the origin and destination airports were given in a different format compared to the other months. As a result, dummy variables could not be assigned, so they were removed. Because deleting a two features, origin and destination airport, from the data set is much worse than just deleting the month of October, the latter was done.

4 Methods

The full five million sample data set was first subsetted to one million samples because of the computational limits of the computer used and the time constraints. The one million samples were then split into a training and testing set with a 80-20 split respectively. The Models, decision tree regressor, random forest regressor and AdaBoost regressor, came from the sklearn python library. Using the training data, a five-fold cross validation was run on three models selected to initially train the models with various hyperparameters. Then, after training the data, the combination of hyperparameters that had the highest cross validation test score for each model was selected for the testing set that was split off earlier. The models took in the input features in the testing set and predicted a continuous value for the output variable: arrival delay. The model predictions for the testing set were compared to the actual arrival delays of the flights in the testing set and and scored using the r-squared metric. The r-squared metric was used to score the three regression models selected.

4.1 Regression Model

Since this study focused on predicting a flight's arrival delay as a continuous output, a decision tree regressor opposed to its classification counterpart was used. Likewise, random forests and AdaBoost regressors were used. Regression models yield a continuous variable while classification models have a binary output. Both types of models have their uses based on the specific situation and the purpose of the machine learning algorithm. In this specific situation with flight delays, a regressor offers more specific information about the output variable, arrival delay. A classification model will predict whether there will be arrival delay, but a regression model will predict how late a specific flight will be. In relation to airport traffic management, an estimated time of a flight's arrival delay has more importance and value compared to a binary value of delay. Knowing approximately how late a flight will arrive can help airport traffic management make more efficient decisions to address the delay; however, simply knowing whether a flight will arrive late or not does not offer the same utility to airport traffic management. As a result, regressor versions of the decision tree, random forest, and AdaBoost models were used for the study.

4.2 Decision Tree

Decision trees are straightforward models that focus on creating statistically significant splits to eventually predict the outcome. This model resembles a tree and starts off with a root node that focuses on a specific feature of the dataset. Based on the value of the feature selected for that specific data point, the model will move onto a system of internal nodes that will focus on different features of the data point. Eventually, the internal nodes will lead to a leaf node where the outcome variable is predicted. From the perspective of this specific dataset, at the root node, the model might focus on the month of the flight. If the month that the flight took place was in winter, the data point will be taken along a specific route along the decision tree. However, if the month that the flight took place was in summer, the data point will go through a different set of internal nodes. Since the season has an important impact on flight arrival delay, splitting the data based on the month feature can help predict the arrival delay. Regardless of the path of internal nodes taken, the datapoint will eventually reach a leaf node where the delay time is predicted. The leaf nodes predict the outcome variable based on the internal and root nodes. Data points with very similar features will take similar paths through the decision tree and end at similar leaf nodes. A decision tree uses one large model that focuses on a variety of features in a data set. Compared to other machine learning models, decision trees are simple, easy to understand, and do not have any data type constraints. However, being such a simple model, there are inherent problems like overfitting and poor performance when introducing new data and outliers. When new data and outliers are applied to very convoluted decision trees, the model will not perform well since it has been overfitted to the training data. When decision trees are overtrained, they can become huge and very complicated with many unnecessary parts that do not contribute to the outcome prediction. This is a heavy disadvantage of decision trees. Regardless, decision trees are simple and very useful.

4.3 Random Forest

To fix the inherent problems with decision trees, an updated model, random forest, was created based on the principle ideas behind a decision tree. Instead of using one large decision tree, random forests utilize a technique called bagging that uses multiple smaller decision trees called weak learners. Random forests, a bagging algorithm, create weak learners in a parallel fashion: one after the other. Each weak learner produces an output prediction for a data point, but the final outcome prediction is based on the results of all of the smaller learners. For example, in a random forest regressor, the output prediction from the multiple weak learners will be averaged to produce a more accurate result. Because multiple weaker decision trees are used in a random forest, one larger convoluted model will be avoided. In turn, this would prevent overfitting and poor performances when introduced to outliers and new data. When using a significant amount of weak learners to predict a specific outcome based on a variety of features, the disadvantages related to regular decision trees can be avoided. However, the main key drawback of random forests and other more complex models is that they are very slow and take a significant amount of time to run. In the real world, dealing with large data sets, creating multiple decision trees at the same time, and using all of them to predict the outcome of one data point in a database will take an extended period of time to complete.

4.4 AdaBoost

Furthermore, instead of creating weak learners in a parallel fashion where all trees have equal weights in the final outcome prediction, AdaBoost varies the weights of the trees based on its ability to predict the outcome feature. AdaBoost is a sequential algorithm that uses a method called boosting. Boosting is when iterations are made to the weak learners based on the previous one's flaws and mistakes. Therefore, each weak learner benefits from the previous ones that have already been created. Unlike random forests, each weak learner is given a different weight when predicting the final outcome. The weights for each weak learner are determined based on the iterations and mistakes each learner makes. Through boosting, a more accurate model can be created. When compared to random forests which average the results from the multiple weak learners, AdaBoost builds on each weak learner through an iterative process and assigns weights to each specific learner; as a result, AdaBoost uses a theoretically more accurate process compared to its predecessors.

4.5 Hyperparameters Selection

The specific hyperparameters that were manipulated for the decision tree model were the number of leaf nodes and max depth. The number of lead nodes was varied because these are the final outputs to the decision tree. Changing how many leaf nodes a decision tree has will impact its model performance. Additionally, the max depth of the decision tree will change the max amount of levels the decision tree is allowed to have. Making the max depth too large can over complicate the model, creating areas that do not contribute at all to the overall decision tree. Making the max depth too small can limit the amount of features used to predict the outcome variable. Therefore, changing max depth will also influence the performance of the decision tree model. The range for the leaf nodes was from 23 to 26 and the max depth varied between 15 and 19.

For the random forest model, the number of weak learners and max depth of each learner was varied. Since random

forest uses multiple weak learners to predict an outcome variable, this hyperparameter is important to vary. The number of weak learners ranged from 42 to 50 across all of the random forest models trained. Similar to the decision trees, the max depth was varied for the random forests; however, the values used were much lower because random forests use much smaller weaker learners. As a result, the max depth for the random forest weak learners was either three of four.

Finally, the hyperparameters that were varied for the AdaBoost model were the number of weak learners, max depth of each learner, and learning rate. Because AdaBoost is more complex compared to random forest, in the interest of time, a range of lower numbers for the number of weak learners was used: between 25 and 29. Increasing the number of weak learners used will increase the amount of time needed to create and train the models. Similar to the random forest models, the max depth for each weak learner was either three of four. Finally, the learning rate for the AdaBoost varied between .088, .089, and .090. The learning rate manipulates the weights of each new weak learner to the overall prediction.

4.6 Cross Validation

The data set must be used to both train and test the selected machine learning models. However, the approach taken to train and test the models must avoid overtraining and other related issues. Using a full set of data to both train and test the models can cause major issues. If the model is tested using the same data that was used to train it, the recorded model performance score from testing can be misleading. When these models are applied to new data in the real world, the algorithms may not perform nearly as well as they did in the testing phase. Instead, the model performance score from the testing phase should be an accurate indicator of how the models will do in the real world with new data that it was not trained on. This goal can be accomplished by doing a cross validation where the large data set is split up into pieces that each play a separate role in creating the model. Cross validations are run to prevent the models from being tested on the same dataset they were trained on.

A cross validation was performed on the models using the training set that contained 80 percent of the subsetted data meaning a total of 800 thousand flights were used to train the models. The 800 thousand flights were used to perform a five-fold cross validation. This means the training set was split into five groups. One of these groups would be left from training the model and used to test the model after; this would yield a training and testing score. The model would be trained five separate times where each time one of the five groups would be left out to test the model. As a result, there will be five different training and testing scores. To measure overall performance of a model on the training set, the mean training and testing score was found from the five-fold cross validation. This process was done for each set of hyperparameters for each of the three types of models.

After running all of the cross validations, the set of hyperparameters with the highest cross validation test score for each of the three models was selected. Moreover, the difference between the train and test score was also looked at to eliminate sets of hyperparameters that created a model that tested poorly in the cross validation. A huge difference between train and test scores in a cross validation indicates that the model will not do well when presented with new data that it was not trained on. In short, the best combination of hyperparameters from the range of values inputted will be determined by using the cross validation mean test score.

TABLE 1: Decision Tree Cross Validation Results

Max Leaf Nodes	Max Depth	Test	Train
26	19	0.887	0.887
26	18	0.887	0.887
26	17	0.887	0.887

TABLE 2: Random Forest Cross Validation Results

Learners	Max Depth	Test	Train
50	4	0.884	0.885
48	4	0.884	0.885
49	4	0.884	0.885

TABLE 3: AdaBoost Cross Validation Results

Learners	Learn Rate	Max Depth	Test	Train
27	0.09	4	0.889	0.889
28	0.09	4	0.889	0.889
29	0.09	4	0.889	0.889

Train and test columns in Tables 1, 2 and 3 refer to the mean train and test scores from the five fold cross validation. These tables show the results of the top three sets of hyperparameters in terms of test scores for the three machine learning models. The cross validation train and test scores for all three models are similar and not significantly different.

5 Results and Discussion

The testing set that was initially split off and set aside had 200 thousand flights as per the 80-20 split of the original 1 million. This testing set was used to test the final three models that were chosen. The models were chosen by looking at the mean test score from the five fold cross validation performed. One model was picked from each of the three different machine learning algorithms. The set of hyperparameters used to yield such models were used to create the final algorithms that would be tested on the testing set. As shown in Table 1, the hyperparameters that yielded the best

decision tree model were a max leaf nodes of 26 and a max depth of 19. Furthermore, as shown in table 2, the random forest model with the best mean cross validation test score was the one with 50 weak learners which each had a max depth of four. Lastly, as displayed in table 3, the AdaBoost model with the best mean cross validation test score was the one with 27 estimators, a learning rat of .09 and a max depth of 4 for each weak learner.

TABLE 4: Table of testing and cross validation performance of the models selected for the final testing set. Fit time for the models is also included. Cross validation is represented as CV in the table

Model	Decision Tree	Random Forest	AdaBoost
Testing Score	0.874	0.867	0.861
Fit Time	2.00	69.92	55.15
CV Train Score	0.887	0.885	0.889
CV Test Score	0.887	0.884	0.889

The decision tree, random forest, and AdaBoost models with these specific hyperparameters used the features from the testing set to predict the continuous arrival delay output. These predictions were then compared to the actual arrival delay of the flights in the testing set; then, a score was calculated using the r-squared scoring metric. Because regression models were used, r-squared seemed an appropriate scoring metric. The testing scores of the three algorithms are shown in Table 4.

The decision tree model did the best among the three models, followed by the random forest model and the AdaBoost model. Ironically, the least complex model actually performed the best on the testing set. This is very surprising because random forest and AdaBoost models were created to improve on decision tree algorithms. The bagging and boosting algorithms in this case did not do better than the base decision tree algorithm. It is very important to point out that the testing scores for the three models are not significantly different from each other, so the small differences in the r-squared metric could be explained by the randomization of the selection of flights in the testing set; however, the results from the study are still significant.

5.1 Complexity and Run Time Tradeoff

The bagging and boosting algorithms, theoretically supposed to perform better than the base algorithm, did not have a higher testing score. This study is evidence for the claim that bagging and boosting algorithms may not perform significantly better than simpler models 100 percent of the time. With model fit time in consideration, a simpler model that performs slightly worse or maybe even better than more complex models in significantly less time may be of better value. Bagging and boosting algorithms create multiple decision trees and use iterative processes requiring a significant amount of time to run; comparatively, the base algorithm, decision tree, takes significantly less time to run. In a world which emphasizes quick results, simpler models may be more useful in certain situations.

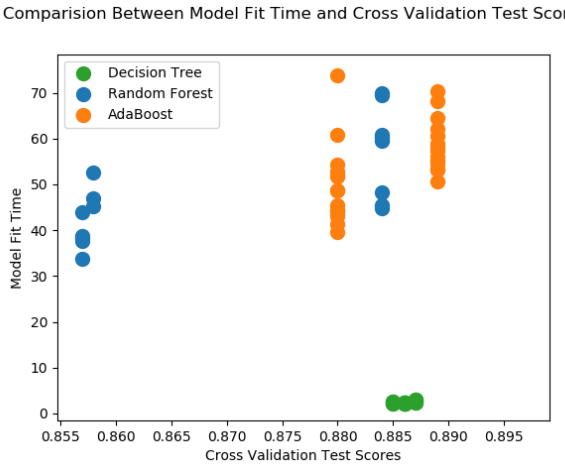

FIGURE 1: This scatter plot shows the three models selected and the relationship between their cross validation test scores and fit time.

To expand on the tradeoff between complexity and time, Figure 1 shows how, in this case, the decision tree models were in the optimal place with a low fit time and a similar cross validation test performance when compared to the other more complex models. In this case, keeping this tradeoff in mind, the best model to predict a flight's arrival delay is the decision tree model. Even though the decision tree model with the best set of hyperparameters does not significantly out-perform the bagging and boosting algorithms on the testing set, the fit time is significantly less; therefore, the decision tree model with the best hyperparameters is more useful in this case because of its relatively quick run time compared to its more complex successors.

6 Conclusion and Future Works

Finish this section

References

Bureau of Transportation Statistics (2015). https://www.transtats.bts.gov/ONTIME/Departures.aspx.

Kuhn, N., Jamadagni, N. (2017). Application of Machine Learning Algorithms to Predict Flight Arrival Delays. Project Posters and Reports, Fall 2017. http://cs229.stanford.edu/proj2017/.