Project 2 - Classifying Titanic Data and Robust Visualizations

Paige Hicks, Soo Shin

Strategy 1- Random Forest Classification

What is Random Forest Classification?

Random Forest Classification is similar to the Decision Tree Classification. The main difference is that the Random Forest Classification generates multiple decision trees then merges them together to build a more accurate model.

Why did we think it would work?

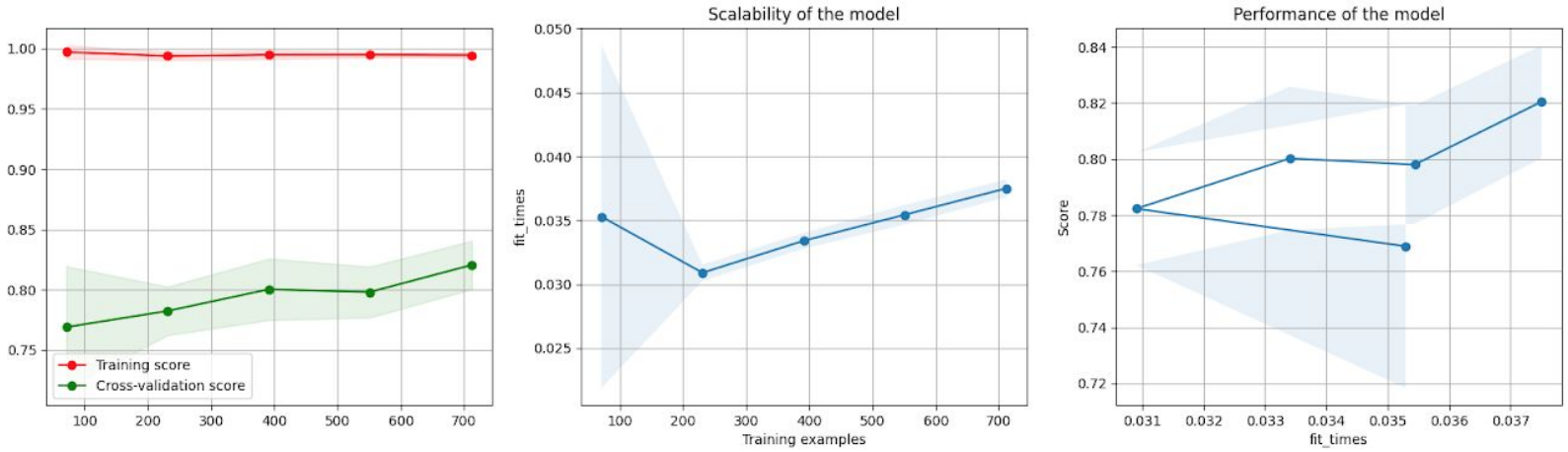
From the homeworks we knew the Decision Tree Classifier would produce a fairly accurate model. Our thought process was “more is better”, so we decided to use the Random Forest Classifier. Since this classifier takes in multiple decision trees (in our case 20 trees), having multiple trees to develop a model proved to be accurate.

Results/ performance -

The precision ranges from 0.9981818181818182 to 1.0
The recall ranges from 1.0 to 0.9970760233918129
The fscore ranges from 0.9990900818926297 to 0.9985358711566619
The support ranges from 549 to 342

These results were taken when there was only 1 misclassification. The main thing we took from Random Forest Classification is the inconsistency of results. The amount of misclassified points varied between each run of the program.

Example learning curve plot -



From our learning plot curve we know the training score is near perfect 1.00. However, our cross-validation score is low (below 85%). The cross-validation score also increases as more data points are introduced. As seen on the graph the green line is going up and it is safe to assume that it will follow that pattern. This means that our classifier has a large amount of variance. This issue can be fixed with the addition of more data or reducing the complexity of the model. In our scalability graph we can see that the Random Forest Classifier follows a linear path after 200 data points. Lastly for our performance, the overall trend of the classifier was positive, this means the performance increased over time.

First tree plot -

DISCLAIMER We did not add a picture of the tree since it is very large.

In our program we decided to only visualize the first tree produced by the Random Forest. All the trees contain important information for feature selection. When taking a look at the tree we can see what attributes were chosen to make a decision.

Strategy 2- KNearestNeighbor Classification

What is KNearestNeighbor Classification?

Each feature represents a dimension in a 10-dimensional space. Points are placed within this space, and distance between points are calculated. Ideally with two classes (survived and died), the points should cluster into two groups. Testing points are added to this space and classified based on the class of the nearest neighbors (n = 15). Testing points representing people who survived should likely end up close to more "survived" points and correctly be classified.

Why did we think it would outperform our first strategy?

Since Random Forests utilize a lot of randomness and result in a bit of variation in its results, we wanted something that would have consistent results. KNN is also a different method of classification compared to random forests. Where random forests use decision trees that only judge based on one feature at a time with thresholds, KNN uses all of the features at once with its distance metric. Therefore, the methods of classification between our two strategies are fundamentally different. KNN in our case also provided better precision, recall, and fscores consistently.

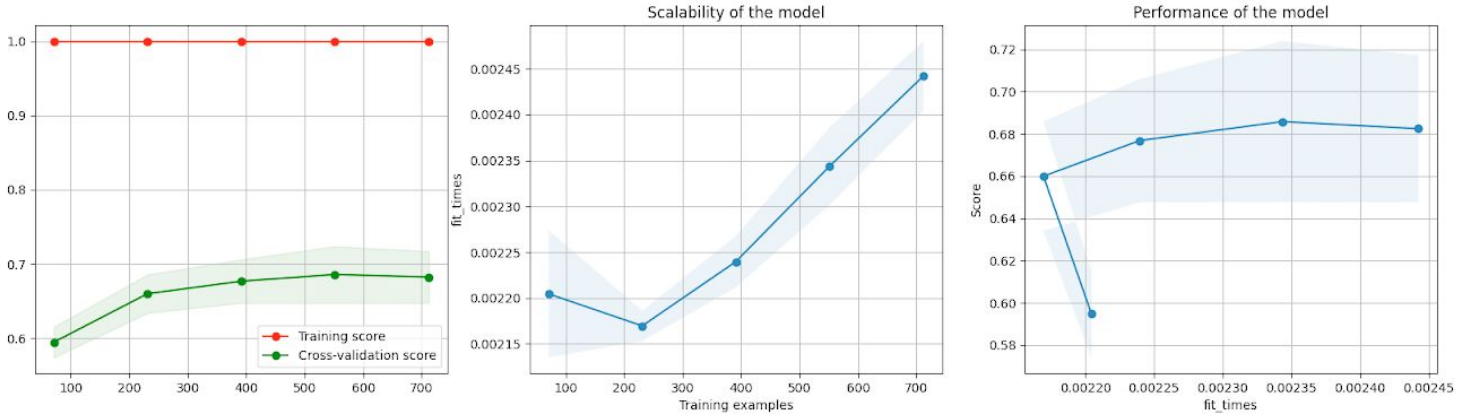
Results/ performance -

Since plotting 10 dimensions of data in 2 dimensions is not realistic and plotting only a few dimensions at a time is not useful, we stuck with just the numeric test results. Based on running the code and checking the *precision_recall_fscore_support* of the data:

The precision ranges from 1.0 to 1.0
The recall ranges from 1.0 to 1.0
The fscore ranges from 1.0 to 1.0
The support ranges from 549 to 342

The KNN algorithm shows perfect precision, recall, and fscore as it does not misclassify any of the test points. The support is the same as the random forest algorithm used above.

Example learning curve plot -



The cross-validation score increases a bit as the amount of data scales up, reaching almost 70% accuracy. The slight dip after 550 indicates that the KNN algorithm may be slightly overfitting the data. It does not perform as well with cross-validation as the random forests appears to, even though it misclassifies fewer points. The variance is pretty high, with over 30% difference between the training score and cross-validation score, indicating more records would be better. The scalability and performance of the model varies each time it is run, but overall it appears to scale well, almost linearly. The performance of the model varies a lot each time, and does not appear to have a linear relationship at any time but rather an almost random one. Performance score appears to increase with time more often than not, but it is not a strong relationship as can be seen from the large margin of error.