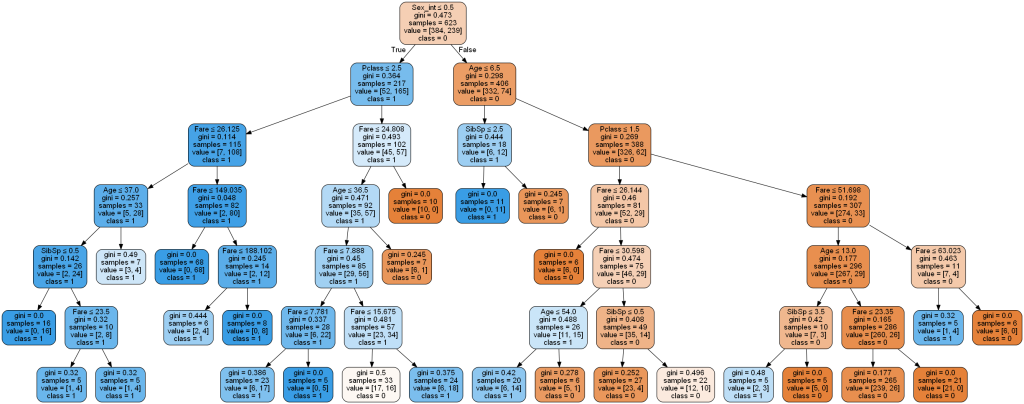
# A unique case where Decision Tree outperforms the Random Forest

Decision tree is a supervised machine learning algorithm, which can be used for classification and regression problems. Its splits the data binarily at each level. The splits are decided using Gini index. Gini is a measure of badness of split.

We will be using Kaggle's Titanic dataset for this post. We know when Titanic crashed, only some people were able to survive since lifeboats available weren't enough to accommodate all the passengers. Using the passenger’s information available, we will predict if that person survived or not. Let's look at the data dictionary first.

**Data dictionary**

|  |  |
| --- | --- |
| Variable | Definition Key |
| survival | Survival 0 = No, 1 = Yes |
| pclass | Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd |
| sex | Sex |
| Age | Age in years |
| sibsp | # of siblings / spouses aboard the Titanic |
| parch | # of parents / children aboard the Titanic |
| ticket | Ticket number |
| fare | Passenger fare |
| cabin | Cabin number |
| embarked | Port of Embarkation C = Cherbourg, Q Queenstown, S = Southampton |

**Decision Tree Model Visualized**

We have now fit a decision tree model on our training data. Let’s see how it performs on the test data.

# Confusion Matrix for Decision Tree

|  |  |  |
| --- | --- | --- |
| Predicted/Actual | Positive (1) | Negative (0) |
| Positive (1) | 150 | 15 |
| Negative (0) | 31 | 72 |

**Overall Accuracy is 0.82**

Decision tree model is very intuitive and *easy to explain* to non-technical stakeholders in Business. But it suffers from a big disadvantage. It exhibits *high variance*.

**What do you mean by high variance?**

High variance implies that there will be a big difference between the model performance on different datasets. That is, our performance on the training data will be very good but when we use the model on test data, the model performance isn't very good. This means our model has overfit the training data.

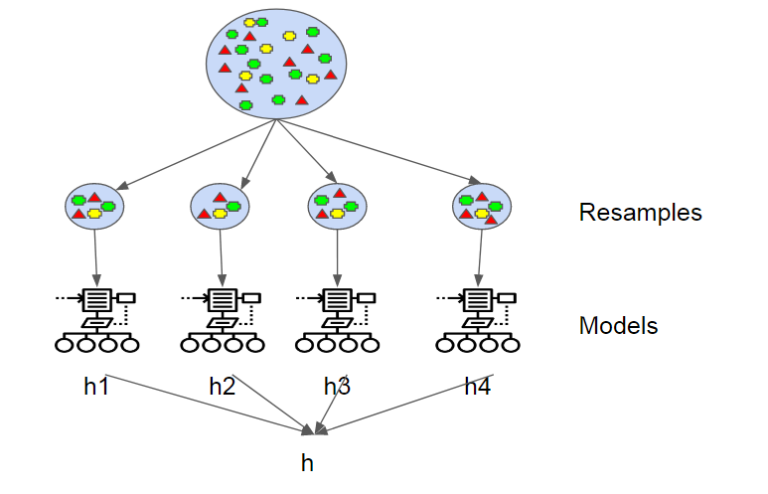
We can adjust hyper parameters like depth of the tree to reduce over-fitting and hence variance. But then it will lead to increase in bias. High bias means, our model is oversimplified and hasn't paid much attention to the training data. This is what we know as *bias - variance trade off*.

**So how do we deal with this problem?**

To tackle the problem of high variance without making a compromise on the bias, *ensemble* models were introduced.

**What's ensembling?**

In ensembling, an aggregate of many models is taken and all the models are allowed to vote on the outcome. The majority of those votes is taken as the predicted output class.



**Random Forest**

Random forest uses *feature randomness* in addition to bagging. In bagging, we use all the features in every tree but in Random Forest, we randomly select a subset of features every time and build each decision tree using those features. This solves the problem in bagging - Existence of very high correlation between the trees (If one tree classifies an observation wrongly, other trees are prone to make the same mistake). Below is a code for random forest model developed from scratch. We fit a CART model on each bootstrap sample using only a random subset of features. Then we have passed our test data as input to the model.

y\_pred\_RF = []

clf\_gini\_RF = []

features\_list = []

sequence = [i for i in range(bootstrap\_sample\_X[0].shape[1])]

from random import sample

#int(np.ceil(np.sqrt(bootstrap\_sample\_X[0].shape[1])))

for i in range(0,k):

clf\_gini\_RF.append(DecisionTreeClassifier(criterion = "gini", random\_state = 100))

feature\_subset = sample(sequence, 3)

clf\_gini\_RF[i].fit(bootstrap\_sample\_X[i].iloc[:,feature\_subset], bootstrap\_sample\_y[i])

y\_pred\_RF.append(clf\_gini\_RF[i].predict(X\_test.iloc[:,feature\_subset]))

feature\_subset = sample(sequence, int(np.ceil(np.sqrt(bootstrap\_sample\_X[0].shape[1]))))

feature\_subset

modes\_list\_RF = []

for i in range(0, len(y\_pred\_RF[0])):

ith\_obs\_list\_RF = []

for j in range(0,len(y\_pred\_RF)):

ith\_obs\_list\_RF.append(y\_pred\_RF[j][i])

#print(ith\_obs\_list)

modes\_list\_RF.append(mode(ith\_obs\_list\_RF)[0][0])

# Confusion Matrix for Random Forest

|  |  |  |
| --- | --- | --- |
| Predicted/Actual | Positive(1) | Negative(0) |
| Positive(1) | 150 | 15 |
| Negative(0) | 37 | 66 |

**Overall accuracy is just 0.80**

**So, what happened?**

Somehow our accuracy is *worse* than even basic Decision Tree model. I changed all hyper parameters and tried but still ended up getting worse results than CART every time.

Let’s use a bagged model and see how it performs.

**Bagging**

Bagging refers to Bootstrap Aggregation. As the name suggests, we create several *resamples* from our original training sample, build models using each resample, then pass our test data into each model.

The results from the models are aggregated. The aggregated output has less variance compared to the model fitted using our train sample. Therefore, we manage to reduce error due to variance without increasing bias.

# we are creating 2000 resamples of the same size as our original sample

bootstrap\_samples = []

bootstrap\_sample\_X = []

bootstrap\_sample\_y = []

for i in range(0, k):

bootstrap\_samples.append(train.sample(train.shape[0], replace = True))

bootstrap\_sample\_X.append(bootstrap\_samples[i][['Pclass','Age', 'SibSp', 'Parch', 'Fare','Sex\_int']])

bootstrap\_sample\_y.append(bootstrap\_samples[i][['Survived']])

# Fitting decision tree models with each sample

y\_pred = []

clf\_gini = []

for i in range(0,k):

clf\_gini.append(DecisionTreeClassifier(criterion = "gini", random\_state = 100))

clf\_gini[i].fit(bootstrap\_sample\_X[i], bootstrap\_sample\_y[i])

y\_pred.append(clf\_gini[i].predict(X\_test))

# Finding the aggregate measure, in this case, mode of the outputs

modes\_list = []

for i in range(0, len(y\_pred[0])):

ith\_obs\_list = []

for j in range(0,len(y\_pred)):

ith\_obs\_list.append(y\_pred[j][i])

#print(ith\_obs\_list)

modes\_list.append(mode(ith\_obs\_list)[0][0])

# Confusion Matrix for Bagging

|  |  |  |
| --- | --- | --- |
| Predicted/Actual | Positive(1) | Negative(0) |
| Positive(1) | 153 | 12 |
| Negative(0) | 29 | 74 |

**Overall accuracy is now 0.84**. This seems good. So, there is a problem with random forest. What do we do in random forest that we don’t in bagging? Feature randomness. Let’s find out if that was the issue.

**Reason**

I analysed my individual RF trees. I found that most of the trees were doing a very bad job. On closer inspection, I realised that only *some of the features* predict the outcome well and since we take a random subset every time, the trees which are built using *only bad features*, will not predict the outcome well.

There was a model which had **accuracy of just 0.52**. I looked at the tree. It had been fit using only Parch, age and SibSp. Whereas the best model had been fit using Pclass, Sex and Age. Pclass and Sex are significant factors which were not taken into account by the worst performing tree due to random feature selection.

**90** models had accuracy below 0.60. So, when so many of the ensembled models do so bad, obviously our overall predictions aren't going to be good. That's why accuracy of Random Forest was worse than Decision Tree.

**My inferences**

This problem will not be seen in Decision trees or Bagged models because a Decision Tree model uses *Gini* to find out the best split and even if only one feature is good, it will likely split using that feature many times and ignore the bad features.

When we have *a smaller number of features* and we know at the start itself what features are *significant*, then we shouldn’t use Random Forest. It’s better to use CART or a Bagged model.

References:

1. <http://www.differencebetween.net/technology/difference-between-bagging-and-random-forest/>
2. <https://www.kdnuggets.com/2019/01/random-forests-explained-intuitively.html>

**Links for reference :**

https://github.com/AjuAjit/DataScienceProjects/blob/master/DecisionTre\_RandomForest.ipynb