All About ML — Part 6: Bagging, Random Forests and Boosting

Dharani J Follow
Apr 27, 2020 · 10 min read

When ever we deal with trees in Machine Learning, bagging and boosting are two commonly heard words. They are usually methods of **ensemble modelling**. It is similar to dividing a big task into numerous small tasks and aggregating them to achieve desired result from it.

Have you ever faced the issue of over fitting in decision trees? we can try changing the parametres like adjusting max_depth etc., but it won't differ much in some cases. If we can recall that in decision trees, there is a high probability of model performing excellently on train data and poor performance in test set. As the decision trees are sensitive to the data they are trained, so any new observation or changes to train data will result in highly fluctuated values, resulting in high variance. To avoid this we have bagging and boosting.

Look at the dataset where we have heights of 20 persons S={173, 179, 188, 163, 165, 176, 178, 157, 160, 157, 185, 188, 167, 168, 158, 178, 178, 160, 166, 171}. Mean or Average height=170.75 and variance is 98.28

Let us now divide the data randomly into 4 samples of 6 observations each. S1= {173, 179, 188, 163, 165,188} , S2={176, 178, 157, 160, 157,160}, S3={ 185, 188, 167, 168, 158,166}, S4={178, 178, 160, 166, 171,157}

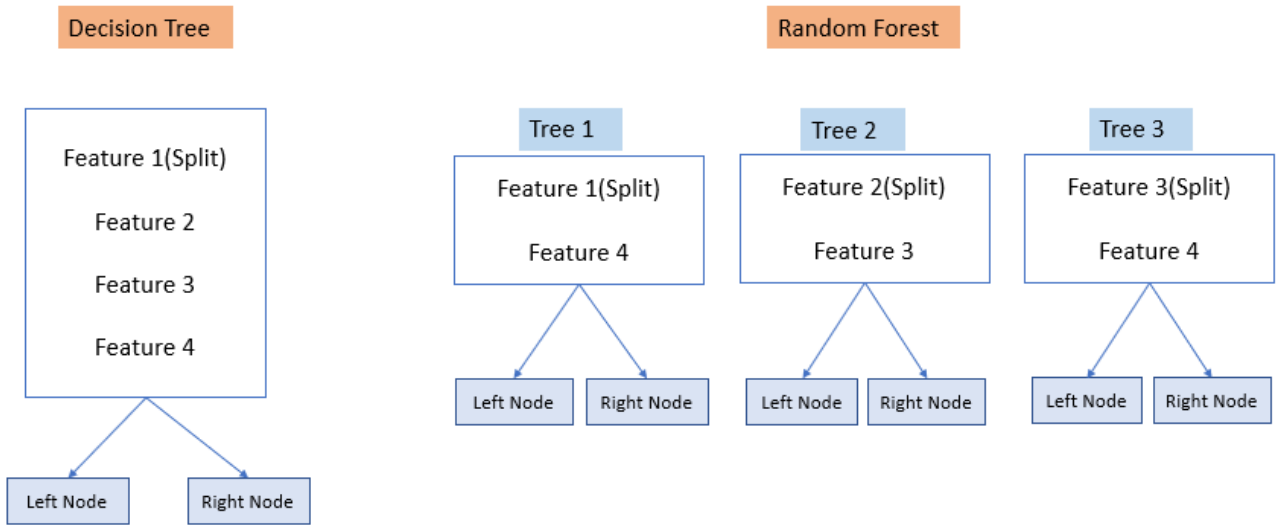
Mean of S1=176, S2=164.66, S3=172, S4=168.33. Mean of means of S1, S2, S3, S4 ([176+164.44+172+168.33]/4) is 170.25 close to mean of S= 170.75

Variance of the means of S1, S2, S3, S4 is 17.76 (very low variance compared to variance of S — 98.28). It is clear from this example that there is no loss in mean

but variance is greatly reduced and we can work with samples and combine them later by averaging. This is the basic idea of bagging — "*Averaging reduces variance*". The process of randomly splitting samples S1 to S4 is called **bootstrap aggregating**. If the sample size is same as original data set size then it is called **boost strapping.** Bagging is a parallel process with an applications found in Random forests.

> *The process of using bootstrapped samples with replacement and applying a model on each of them and then average out the results to avoid high variance is called bagging.*

Random Forests



Difference between Decision Tree and Random Forests

I was working on prediction of house prices data set to understand the algorithms on regression. One successful attempt was using Lasso and Ridge to reduce over fitting on high dimensional data (In case you are interested in full code it is at the end of the blog). Then I applied decision trees which produced decent accuracy but when I want to improve it, I tried different options like changing max_depth parametre. As max_depth is an important tuning feature which decides the complexity of the model, we can tune it in such a way that it does not overfit on the data. Increasing the depth will let the tree capture all the splits and patterns in the train data accurately and fail on a new data point in test set so reducing max_depth is one way to combat overfitting. The observations on house prices data are:

```
from sklearn.tree import DecisionTreeRegressor
dtreg = DecisionTreeRegressor(random_state = 100,max_depth=5)
dtreg.fit(x_train, y_train)

dtr_pred = dtreg.predict(x_val)
dtr_pred= dtr_pred.reshape(-1,1)

dtr_x_pred = dtreg.predict(x_train)
dtr_x_pred = dtr_x_pred.reshape(-1,1)
```

```
x_train_accuracy
MAE: 0.2703293234276162
MSE: 0.13159263028147444
RMSE: 0.36275698515876226
R2 Score: 0.8684073697185255
x_val_accuracy
MAE: 0.324728379527754
MSE: 0.24074576384770086
RMSE: 0.49065850022974317
R2 Score: 0.7592542361522991
```

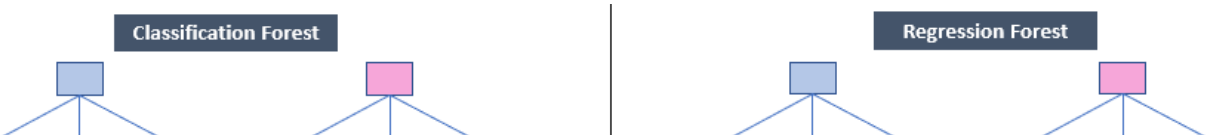
Max_depth = 5

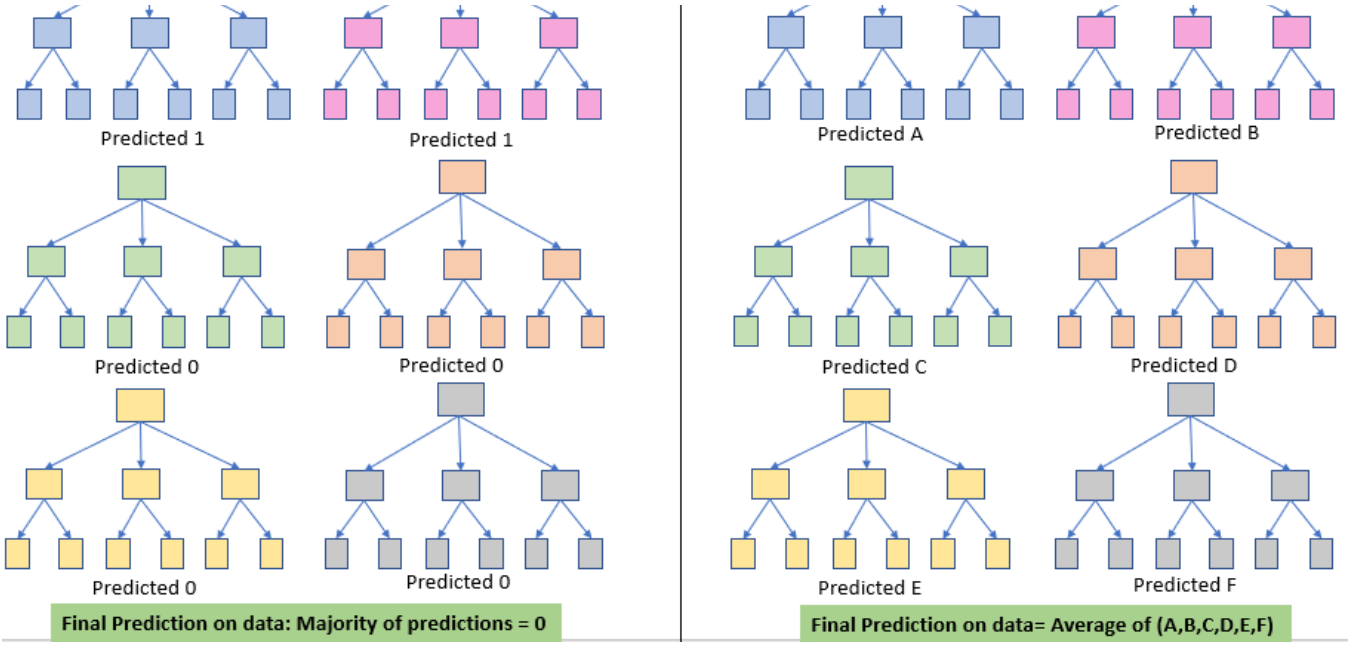
```
x_train_accuracy
MAE: 0.22191390250546256
MSE: 0.08637105146508792
RMSE: 0.2938895225507162
R2 Score: 0.913628948534912
x_val_accuracy
MAE: 0.34414796053797925
MSE: 0.33714341470219505
RMSE: 0.5806405210646214
R2 Score: 0.662856585297805
```

Max_depth = 6

max_depth here is 5 and decent scores of RMSE and R2-Score of training and validation data set. When max_depth is changed to 6, the model is over fitting as the metrics are really good on training data but on validation set, they are not upto the mark. I experimented a bit on depths 7 to 10 but the over fitting increased a lot, leading bad accuracy on validation data.

In bagging, the whole data is bootstrapped into various samples with replacement and on each of this samples, a decision tree is built with all the features and the results are averaged out to get a final prediction. If we apply this concept with some twist on number of features and correlated trees, then the result is a random forest.





Random Forest Application

In each decision tree here, the split is made such that it considers only 'm' number of features which are randomly picked from the total of 'p' features. A new sample of 'm' features is picked at each split and we choose $\mathbf{m \approx \sqrt{p}}$ **features** in general.

> *Therefore, while building each tree in random forest, not all the features are considered at each split.*

In bagging, we take all the features at each split unlike discounting them in random forests. Assume there is one strong feature which is more effective in finding out the patterns in the predictions. In bagging, if we consider all the features, then all the small trees will produce similar predictions which are correlated and averaging out will not reduce any variance, in many cases this might produce bad results. But consider random forests, we consider only m features at each split and that one string feature may or may not be there in the selected features, so each decision tree will produce uncorrelated results and averaging them will result in substantial reduction in variance. This processing of decorrelating the trees is more grounded and valid.

> *Uncorrelated trees are another important feature of random forests which finds its uses in real time data sets for regression and classification.*

In python sklearn.ensemble package we have RandomForestRegressor and RandomForestClassifier models. Lets check out the commonly used parameters involved which upon changing could produce a great performing model.

n_estimators: This defines the number of decision trees we would like to have in the random forest (default is 100)

min_samples_split: The minimum number of sample required to split a node which is internal(Neither the root node or leaf node is internal node)

max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples

max_features: Maximum number of features for best split, default is square root of p(total number of features) as discussed above. We can also specify log2(p) or a fraction of p

I have used RandomForestRegressor on house prices predicting data set to check the difference of using Decision Tree and RF,

n_estimators is the number of trees in the forest we would like to input, as the data size is around 1200 rows, we can experiment around with 10–15 trees.

max_depth can be changed and experimented with and we can come up with an optimal one after few observations. As a start I took 4 and checked with 3,5,6,7,as well.

optimal number of max features at split as discussed above would be square root of total number of features. In house prices data set, we have around 205 features so the max_features I took is 16 and min_sample_split can be given from a range of 1–40 for decision trees, higher values might lead to overfitting here I chose 4 and by default it is 2 if we do not specify and the results are:

```
from sklearn.ensemble import RandomForestRegressor
rfreg = RandomForestRegressor(n_estimators=15,random_state = 100,
max_depth=4, max_features = 16,min_samples_split=4)
rfreg.fit(x_train, y_train)

rf_pred = rfreg.predict(x_val)
rf_pred= rf_pred.reshape(-1,1)

rf_x_pred = rfreg.predict(x_train)
rf_x_pred = rf_x_pred.reshape(-1,1)
```

max_depth=4 and max_features=16

max_depth=4, max_features = 20

If we play around with the parameters a bit, I observed that max_depth =4 is optimal and when increased to 5 or more there is a sign of overfit. Increasing other features like n_estimators or min_sample_split resulted in high accuracy on train data but less on test data.

When DecisionTreeRegressor with max_depth = 5 is used, RMSE score is 0.49 and R2 is 0.75 which is a good score but with RandomForestRegressor max_depth=4 and max_features=20, RMSE has reduced to 0.43 and R2 score has increased to 0.80 on validation set, with no sign of overfitting. Thus, Random Forests finds a great use in Regression and Classification problems when we want to have best accuracy measures.

Random Forests finds the use in many fields as they have the property of performing well on large data sets with high dimensions. Predominantly they are used in banking(fraud detection), finance and stock markets due to the property of uncorrelated trees that reduce correlation.

Boosting

The concept of feedback is used in boosting — *Learn from the weak to get strong*. Boosting is a sequential process. Boosting involves combining a large number of decision trees let them be ˆf1,…, ˆfB (B is number of trees)

In Boosting we have 3 parametres that decide the performance:

1. Number of trees B. In bagging and random forests if we increase B then there is no effect but in boosting, it can over fit if B is large. The optimal number of B is found by using cross validation.

2. Each prediction of data points by the model is called **residual**, the shrinkage parameter λ(small positive number) slows the process down even further, allowing more and different shaped trees to change the residuals. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance

3. The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

An overview of Boosting algorithm,

1. Set ˆf(x) = 0 and ri = yi for all i in the training set

2. For b=1,2,3… B repeat the following steps:

— Fit a tree ˆfb with d splits (d+1 terminal nodes) to the training data (X,r).

— Update ˆf by adding the new version of the tree that is formed by shrinking:

— Update the residuals(weights),

3. Final model will be the consolidated one with all the trees,

Difference between Bagging and Boosting

In simple words, the original data is taken and sequentially the data points are updated in boosting (unlike taking samples in bagging) and we give extra weightage to the wrongly predicted observations, the next models will try to correct them and thus the **error(bias) is minimized**. Boosting learns the patterns in the data from weak learners to finally develop a strong model.

There is always an ambiguity in deciding when to use bagging or boosting in the data sets that we will encounter. Bagging will give best results when it is in concern with over fitting or high variance and Boosting will perform well with models that have high errors and reduces them as it enhances and reduces the risk of errors by adjusting weights to weak learners. Boosting is not much recommended for reducing over fitting as it self has the disadvantage if it. Bagging is parallel and Boosting is sequential thus it makes it quite slow in computation with huge data sets.

Widely used separate algorithms used for these ensemble methods are:

1. Random Forest — bagging.

2. Ada Boost, Gradient Boosting and XG Boost — boosting.

Hope this gives a good understanding of the ensembling methods :)

The code I used for House prices *data set* with Lasso andRidge, Decision Trees and Random Forest

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn import ensemble
from sklearn.linear_model import Lasso,Ridge#load train data
df_data=pd.read_csv("data_price.csv")
df_data.head()#to know each and every column execute the following
print(df_data.columns)
print(df_data.shape)total =
df_data.isnull().sum().sort_values(ascending=False)
percent =
(df_data.isnull().sum()/df_data.isnull().count()).sort_values(ascen
ding=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])
missing_data.head(20)df_data=
df_data.drop(missing_data[missing_data['Total']>1].index.values,1)
df_data=
df_data.drop(df_data.loc[df_data['Electrical'].isnull()].index)corr
_mat=df_data.corr()fi,ax=plt.subplots(figsize=(20,20))
sns.heatmap(corr_mat,square=True)del df_data['Id']le=LabelEncoder()
cat_mask= df_data.dtypes=='object'
cat_cols= df_data.columns[cat_mask].tolist()
cat_cols#Lets convert the columns to one ht encoding
df_data[cat_cols]=df_data[cat_cols].apply(lambda x:
le.fit_transform(x.astype(str)))
df_data_c = df_data.copy()#get_dummies is used for one hot encoding
df_data_c = pd.get_dummies(df_data_c,columns=cat_cols)x_train,
x_test, y_train, y_test =
train_test_split(df_data_c.drop('SalePrice',axis=1),df_data_c['Sale
Price'], test_size =0.25,random_state=120)
y_train= y_train.values.reshape(-1,1)
y_test= y_test.values.reshape(-1,1)sc_X = StandardScaler()
sc_y = StandardScaler()
x_train = sc_X.fit_transform(x_train)
x_test = sc_X.fit_transform(x_test)
y_train = sc_X.fit_transform(y_train)
y_test = sc_y.fit_transform(y_test)#Linear Regression
lm = LinearRegression()
lm.fit(x_train,y_train)#predictions on train data
x_pred = lm.predict(x_train)
x_pred = x_pred.reshape(-1,1)#Prediction of test data
```

```
y_pred = lm.predict(x_test)
y_pred= y_pred.reshape(-1,1)def scores_(y,x):
    print('MAE:', metrics.mean_absolute_error(y, x))
    print('MSE:', metrics.mean_squared_error(y, x))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y, x)))
    print('R2 Score:'
,metrics.r2_score(y,x))print('InSample_accuracy')
scores_(y_train, x_pred)
print('--------------------------')print('OutSample_accuracy')
scores_(y_test,y_pred)def regularization_model(model,alpha_range):
 rmse_score_insample=[]
 rmse_score_outsample=[]
 r2_score_insample=[]
 r2_score_outsample=[]
 for i in alpha_range:
 regularization = model(alpha=i,normalize=True)
 regularization.fit(x_train,y_train)
 y_pred_train = regularization.predict(x_train)
 y_pred_train = y_pred_train.reshape(-1,1)
 y_pred_test=regularization.predict(x_test)
 y_pred_test = y_pred_test.reshape(-1,1)

rmse_score_insample.append(np.sqrt(metrics.mean_squared_error(y_tra
in,y_pred_train )))

rmse_score_outsample.append(np.sqrt(metrics.mean_squared_error(y_te
st, y_pred_test)))
 r2_score_insample.append(metrics.r2_score(y_train, y_pred_train))
 r2_score_outsample.append(metrics.r2_score(y_test,
y_pred_test))df=pd.DataFrame()
 df['alpha']=alpha_range
 df['rmse_score_insample'] = rmse_score_insample
 df['rmse_score_outsample']= rmse_score_outsample
 df['r2_score_insample'] = r2_score_insample
 df['r2_score_outsample'] = r2_score_outsample
 return df.plot(x = 'alpha', y =
['rmse_score_insample','rmse_score_outsample'])alpha_range_lasso =
np.arange(0.001,0.03,0.001)
print(regularization_model(Lasso,alpha_range_lasso))alpha_range_rid
ge = np.arange(0.001,1,0.1)
print(regularization_model(Ridge,alpha_range))

from sklearn.tree import DecisionTreeRegressor
dtreg = DecisionTreeRegressor(random_state = 100,max_depth=5)
dtreg.fit(x_train, y_train)

dtr_pred = dtreg.predict(x_val)
dtr_pred= dtr_pred.reshape(-1,1)

dtr_x_pred = dtreg.predict(x_train)
dtr_x_pred = dtr_x_pred.reshape(-1,1)

from sklearn.ensemble import RandomForestRegressor
rfreg = RandomForestRegressor(n_estimators=15,random_state = 100,
max_depth=4, max_features = 16,min_samples_split=4)
rfreg.fit(x_train, y_train)
```

```
rf_pred = rfreg.predict(x_val)
rf_pred= rf_pred.reshape(-1,1)

rf_x_pred = rfreg.predict(x_train)
rf_x_pred = rf_x_pred.reshape(-1,1)
```

Thank you!

References: An Introduction to Statistical Learning: With Applications in R

Machine Learning Random Forest Bagging Boosting Bootstrapping