Chapter 2 : Section 6

**Random Forest Regression**

**2.6.1 Ensemble Learning**

* Ensemble Learning: Ensemble learning is when you take *multiple algorithms* or the *same algorithm multiple times* and you put them together to make something much more powerful than the original.
* Ensemble Methods: In *statistics* and *machine learning*, ensemble methods use *multiple learning algorithms* to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

[Unlike a ***statistical*** ***ensemble*** in statistical mechanics, which is usually infinite, a machine learning ensemble consists of only a concrete ***finite set of alternative models***, but typically allows for much more flexible structure to exist among those alternatives.]

* Bootstrapping: In general, bootstrapping usually refers to a self-starting process that is supposed to continue or grow without external input.

**2.6.2 Types of Ensemble Methods**

Main Types of Ensemble Methods

1. Bagging: Bagging, the short form for bootstrap aggregating, is mainly applied in classification and regression. It ***increases the accuracy*** of models through Decision Trees, which ***reduces variance*** to a large extent. The reduction of variance increases *accuracy*, eliminating *overfitting*, which is a challenge to many predictive models.

* Bagging is classified into two types, i.e., *bootstrapping* and *aggregation*.

1. Bootstrapping is a *sampling* *technique* where samples are derived from the whole population (set) using the *replacement* *procedure*. The ***sampling with replacement*** method helps make the ***selection procedure randomized***. The base learning algorithm is run on the samples to complete the procedure.
2. Aggregation in bagging is done to incorporate *all possible outcomes of the prediction* and *randomize the outcome*. ***Without aggregation, predictions will not be accurate*** because all outcomes are not put into consideration. Therefore, the aggregation is based on the ***probability bootstrapping procedures*** or on the basis of all outcomes of the predictive models.

Bagging is advantageous since weak base learners are combined to form a single strong learner that is more stable than single learners. It also eliminates any variance, thereby reducing the overfitting of models. One limitation of bagging is that it is computationally expensive. Thus, it can *lead to more bias* in models when the *proper procedure of bagging is* ***ignored***.

1. Boosting: Boosting is an *ensemble* *technique* that learns from ***previous predictor mistakes to make better predictions in the future***. The technique ***combines several weak base learners*** to form one strong learner, thus significantly improving the predictability of models. *Boosting works* by *arranging weak learners* in a sequence, such that *weak learners learn from the next learner* in the sequence to *create better predictive models*.

* *Boosting takes many forms*, including GRADIENT BOOSTING, ADAPTIVE BOOSTING (ADABOOST), and XGBOOST (EXTREME GRADIENT BOOSTING).

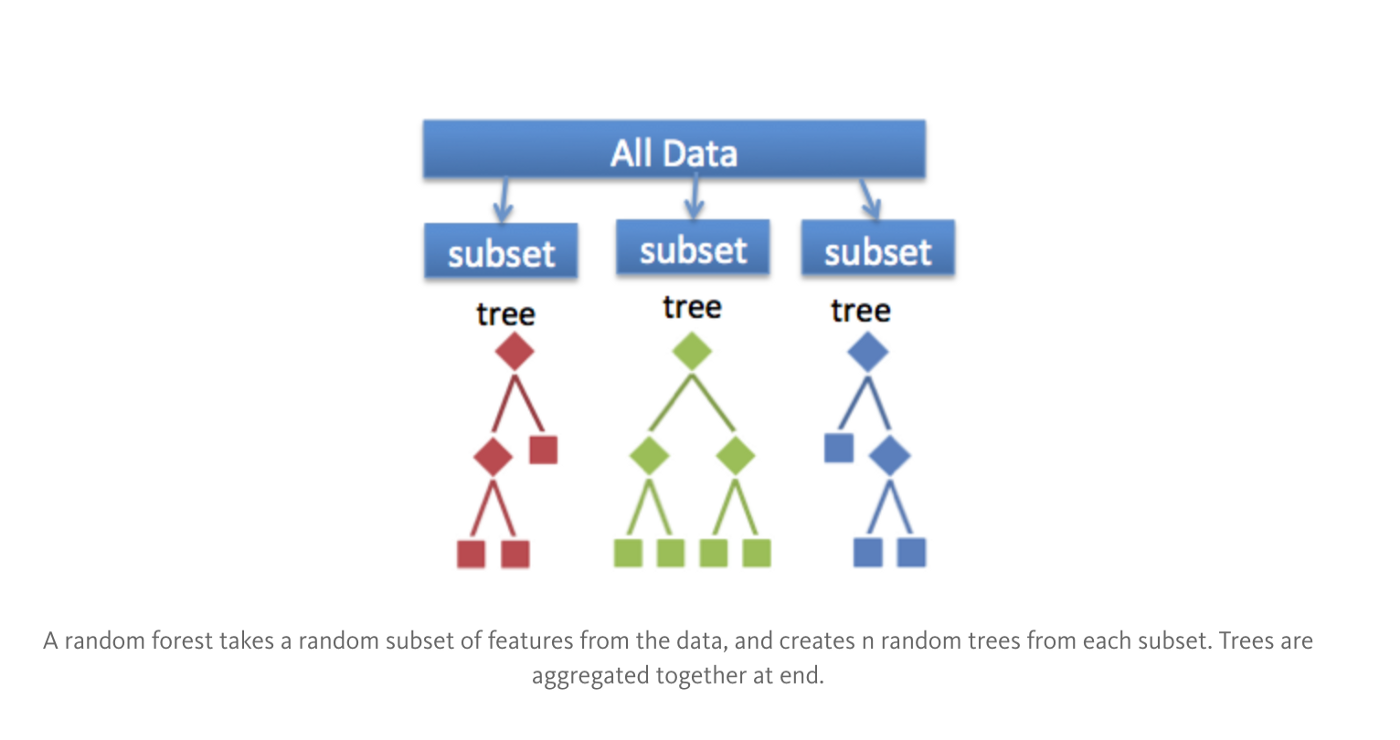
1. AdaBoost uses weak learners in the form of Decision Trees, which mostly include one split that is popularly known as *decision* *stumps*. AdaBoost’s main decision stump comprises observations carrying similar weights.
2. Gradient boosting adds *predictors* *sequentially* to the *ensemble*, where ***preceding******predictors*** *correct their* ***successors***, thereby increasing the model’s accuracy. New predictors are fit to counter the effects of errors in the previous predictors. The gradient of descent helps the gradient booster identify problems in learners’ predictions and counter them accordingly.
3. XGBoost makes use of *Decision* *Trees* with *Boosted* *Gradient*, providing improved speed and performance. It relies heavily on the computational speed and the performance of the target model. Model training should follow a sequence, thus making the implementation of gradient boosted machines slow.
4. Stacking: Stacking, another ensemble method, is often referred to as StackedGeneralization. This technique works by allowing a ***training algorithm*** to ***ensemble several other similar learning algorithm predictions***. Stacking has been successfully implemented in Regression, Density Estimations, Distance Learning, and Classifications. It can also be used to measure the error rate involved during BAGGING.

* Variance Reduction by Ensemble methods: Ensemble methods are ideal for reducing the variance in models, thereby increasing the accuracy of predictions. The *variance is eliminated* when *multiple models* are combined to form a *single prediction* that is chosen from all other possible predictions from the combined models. An *ensemble of models* combines various models to ensure that the resulting prediction is the best possible, based on the consideration of all predictions.

**2.6.3 BAGGing and Random Forest**

|  |  |
| --- | --- |
| * *BAGGing* gets its name because it combines *Bootstrapping* and *Aggregation* to form one *ensemble* *model*. * Given a sample of data, *multiple* *bootstrapped* *subsamples* are pulled. * A Decision Tree is formed on each of the *bootstrapped* *subsamples*. * After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor.   Image on the right will help explain: | J:\My_images\bagging.png  Given a Dataset, bootstrapped subsamples are pulled. A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor. |

* Random Forest Models in BAGGING: The random forest algorithm is actually a bagging algorithm. *Random* *Forest* *Models* can be thought of as *BAGGing*, with a slight tweak.
* When deciding where to split and how to make decisions, *BAGGed* *Decision* *Trees* have the full disposal of features to choose from. Therefore, although the *bootstrapped* *samples* may be *slightly* *different*, the *data is largely going to break off at the same features throughout each model*.
* In contrary, RandomForestmodels ***decide where to split based on a random selection of features***. *Rather* than *splitting at similar features at each node throughout*, ***Random Forest models implement a level of differentiation*** because *each tree will split based on different features*. This level of differentiation provides a greater ensemble to aggregate over, ergo producing a more accurate predictor. Refer to the image for a better understanding.



* Similar to *BAGGing*, *bootstrapped* *subsamples* are pulled from a larger dataset. A Decision Tree is formed on each subsample. HOWEVER, the decision tree is split on *different* *features* (in this diagram the *features* are represented by *shapes*).

**2.6.4 Random Forest**

We will discuss about Random Forest applied to Regression Trees. We first create N decision trees on N subset of data (picking data points randomly) and then we predict a new data-point using N- N decision trees. Following are the steps for Random Forrest:

* STEP 1: Pick at ***random K data points*** from the Training set. i.e. make a subset of data point.
* STEP 2: Build the Decision Tree associated to these K data points. i.e. create a Decision Tree on the selected subset.
* STEP 3: Choose the number N-tree of trees you want to build and repeat STEPS 1 & 2
* STEP 4: For a new data point, make each one of your Ntree trees predict the value of Y to for the data point in question, and assign the new data point the average across all of the predicted Y values.
* In that way you're ***not just predicting one tree but on a forest of trees***. And that improves the accuracy of your prediction because it is you're taking the *average* *of* *many* *predictions*.
* Therefore even if some tree is too perfect (overfitting) or too bad at prediction those extreme case are ignored i.e. impact on a forest of trees is negligible. So you're going to get a more accurate prediction.
* This ensemble algorithms are more stable because any changes in your data set could really impact one tree but it is hard to impact on a forest of trees.

**2.6.5 Random Forest in Python**

Random forest is just a team of Decision Trees each one making some *prediction* of your *dependent* *variable* and the ultimate prediction of the ***Random forest itself*** is simply the average of the different predictions of all the different trees in the forest.

|  |  |  |  |
| --- | --- | --- | --- |
| * Always 3 steps: | 1. import class | 1. create object | 1. fit the dataset |

#*Fit dataset to Random Forest Regression*

**from** sklearn**.**ensemble **import** RandomForestRegressor #*import class*

regressor = **RandomForestRegressor**(n\_estimators= 10 ,random\_state= 0) #*create object*

regressor**.fit**(X, y) #*fit the dataset*

* Parameters for **RandomForestRegressor**(): ***n\_estimator***: no of trees, ***random\_state*** = 0, ***criterion*** = "mse" (mse = mean square method), ***max\_features*** = "auto" (improve your model), (other max-min parameters).
* Finding best team of trees: Now all code works as the previous model, we just need to select the no. of trees ***n\_estimator*** to make a better prediction.

|  |  |
| --- | --- |
| * Here we have *several* *trees* hence we have *several* *stairs* . This is expected for non-continuous models. * So we have a lot more of splits of the whole range of levels and therefore a lot more *intervals* of the different levels. So each straight *horizontal* *line* here separate by the *vertical* *lines*. * Now if we get prediction for the ***6.5*** level (which is ***167000***), what happened with this prediction is that- ***we had 10 trees voting on which step the salary of the 6.5 level position would be*** and then the *Random forest* takes the *average of all the different predictions* of the salary of the 6.5 level made by all the different trees in the forest. And the average of the predictions is ***167000***. | 10 trees : |

* Stairs get into certain shape: If we add a lot more trees in our random forest, it *doesn't mean we'll get a lot more steps* on the stairs because the *more you add some trees the more the average of the different predictions made by the trees is converging to the same average* this is based on the same technique Entropy and Information Gain.
* So the more you add trees the more the average of these votes will converge to the same Ultimate Average and therefore it will converge to some Certain Shape of stairs here.

|  |  |
| --- | --- |
| ***10 trees***, predicted salary for 6.5 level is ***167000***.  **RandomForestRegressor**(n\_estimators= 10 ,random\_state= 0) |  |
| ***100 trees***, predicted salary for 6.5 level is ***158300***.  **RandomForestRegressor**(n\_estimators= 100 ,random\_state= 0) |  |
| ***300 trees***, predicted salary for 6.5 level is ***160333.33333333***.  **RandomForestRegressor**(n\_estimators= 300 ,random\_state= 0) |  |

* From above we can see that the *no. of steps in the stairs is not changing*. But the stair it taking a ***unique*** ***shape***
* Conclusion:
* So as a *comparison* this *Random forest model* is even better than the *Polynomial* *model*.
* In Chapter-10 we will build some "Ensemble ML models". Some models that are a combination of several ML models and these Ensemble ML models are actually the best models. When you have a team of several ML models they can actually make an awesome prediction.
* In this section we had a *team of same ML models* which were Decision Tree regression models. But in the future we'll make a team of different ML models.
* Note: Continuous model doesn't mean the dataset is continuous, but the ***"Model/Mathematical-Model"*** is continuous.

Practiced version

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Position\_Salaries.csv")

X = dataSet**.**iloc[:, 1:2]**.**values

y = dataSet**.**iloc[:, 2]**.**values

#*# Feature-Scaling*

#*from sklearn.preprocessing import StandardScaler*

#*sc\_x = StandardScaler()*

#*sc\_y = StandardScaler()*

#*X\_scaled = sc\_x.fit\_transform(X)*

#*y\_scaled = sc\_y.fit\_transform(y.reshape(-1, 1))*

#*Data Split : No need for this example*

#*Fit dataset to Random Forest Regression*

**from** sklearn**.**ensemble **import** RandomForestRegressor #*import class*

regressor = **RandomForestRegressor**(n\_estimators= 300 ,random\_state= 0) #*create object*

regressor**.fit**(X, y) #*fit the dataset*

#*Predict*

y\_pred = regressor**.predict**([[6.5]])

**print**("The predicte value ffor 6.5 is : ", y\_pred)

#*plot the model*

X\_grid = np**.arange**(min(X), max(X), 0.01)

X\_grid = X\_grid**.reshape**(len(X\_grid), 1) #*reshape matrix/array*

pLt**.scatter**(X, y, color = "red")

pLt**.plot**(X\_grid, regressor**.predict**(X\_grid), color = "green")

pLt**.title**("Truth or Bluff (Random Forest Regression)")

pLt**.xlabel**("Position level")

pLt**.ylabel**("Salary")

pLt**.show**()

**2.6.6 FAQ**

* What is the advantage and drawback of Random Forests compared to Decision Trees?
* Advantage: *Random* *Forests* can give you a *better* *predictive* power than *Decision* *Trees*.
* Drawback: *Decision* *Tree* will give you more *interpretability* than *Random* *Forests*, because you can plot the graph of a *Decision* *Tree* to see the different splits leading to the prediction, as seen in the Intuition Lecture. That’s something you can’t do with *Random* *Forests*.
* When to use Random Forest and when to use the other models?
* The best answer to that question is: try them all!

Indeed, thanks to the templates it will only take you 10 minutes to try all the models, which is very little compared to the time dedicated to the other parts of a data science project (like Data Preprocessing for example). So just don’t be scared to try all the regression models and compare the results (through cross validation which we will see in Chapter 10). That’s we gave you the maximum models in this course for you to have in your toolkit and increase your chance of getting better results.

* However then, if you want some shortcuts, here are some rules of thumbs to help you decide which model to use: First, you need to figure out whether your problem is linear or non linear. You will learn how to do that in Chapter 10 - Model Selection.
* Then: If your problem is *linear*, you should go for *Simple Linear Regression* if you only have *one* feature, and *Multiple Linear Regression* if you have *several* *features*.
* If your problem is non linear, you should go for Polynomial Regression, SVR, Decision Tree or Random Forest.
* Then which one should you choose among these four? That you will learn in Chapter - 10 – Model Selection. The method consists of using a very relevant technique that evaluates your models performance, called k-Fold Cross Validation, and then picking the model that shows the best results. Feel free to jump directly to Part 10 if you already want to learn how to do that.
* How do I know how many trees I should use?
* First, I would recommend to choose the number of trees by experimenting. It usually takes less time than we think to figure out a best value by *tweaking* and *tuning* your model *manually*. That’s actually what we do in *general* when we build a *Machine Learning model:* we do it in *several* *shots*, by *experimenting* *several* values of *hyperparameters* like the number of trees. However, also know that in Chapter 10 we will cover k-Fold Cross Validation and Grid Search, which are powerful techniques that you can use to find the optimal value of a hyperparameter, like here the number of trees.
* You should use enough trees to get a good accuracy, but you shouldn’t use too many trees because that could cause overfitting. You will learn how to find the optimal number of trees in the first section of Chapter 10 - Model Selection. It’s done with a technique called Parameter Tuning (Grid Search with k-Fold Cross Validation).
* Why do we get different results between Python and R?
* The difference is likely due to the ***random*** ***split*** of ***data***. If we did a ***cross-validation*** (see Chapter 10) on all the models in ***both*** ***languages***, then you would likely get a ***similar*** ***mean*** ***accuracy***.
* No p-valuefor Random Forest
* You couldn’t use *p-value* because *Random Forests* are *not linear models*, and *p-values apply only to linear models*. Therefore feature selection is out of the question. But you could do feature extraction, which you will see in Chapter 9 - Dimensionality Reduction. That you can apply to Random Forests, and it will reduce the number of your features.