Chapter 3 : part 7

**Random Forest Classification**

**3.7.1 Random Forest Classification**

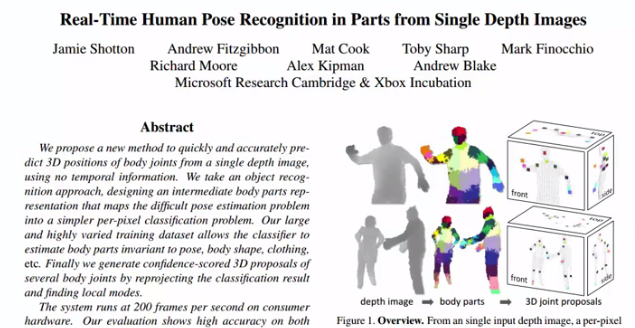
Random Forest is a version of ensemble learning (such as gradient boosting). Ensemble learning is when you take multiple algorithms or the same algorithm multiple times and you put them together to make something much more powerful than the original.

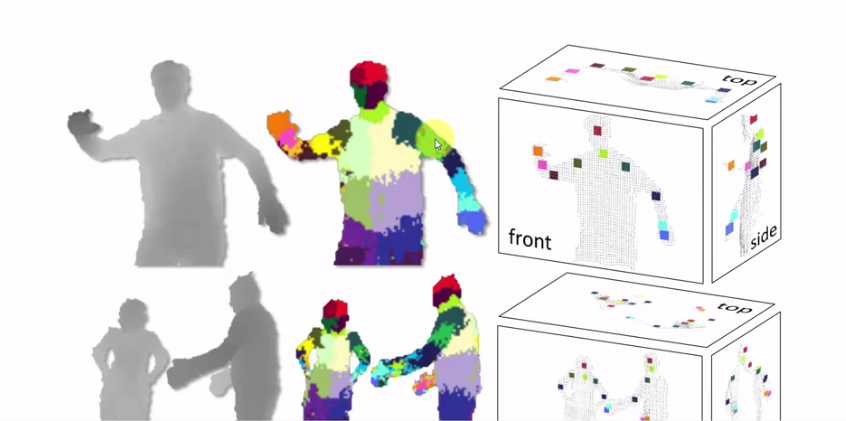
* How it actually works:

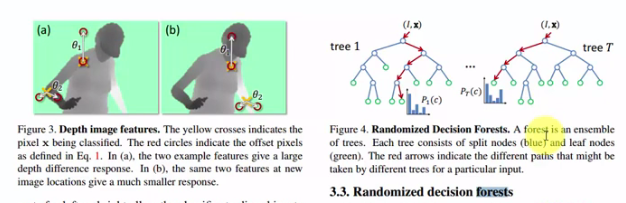
1. STEP 1: Pick at ***random*** ***K*** ***data*** ***points*** from the Training set.
2. STEP 2: Build the ***Decision Tree*** associated to these ***K data points***.
3. STEP 3: Choose the number ***Ntree*** of trees you want to build and repeat STEPS 1 & 2
4. STEP 4: For a *new data point*, make *each* 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*.

**One of decision trees practical example is Microsoft's Xbox's Body-motion Sensor.**









**3.7. 2 Python Implementation of 3.7.1 Random Forest Classification**

#*Fit train set to Random forest classifier: No parmeter is needed*

**from** sklearn**.**ensemble **import** RandomForestClassifier

clsFier = **RandomForestClassifier**(n\_estimators= 10, criterion="entropy", random\_state=0)

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

|  |  |
| --- | --- |
| * Parameters: * ***n\_estimators= 10***, Number of trees to be used. Try different number of trees to detect overfitting. * Similar to Decision tree we use ***criterion = "entropy"*** * Confusion matrix: * Some Test values and Predicted values are given to the right: * Below is the confusion matrix (with 10 trees/estimators) : |  |

Practiced Version

#*----------- Naive Bayes model ----------*

#*Library*

**import** pandas **as** pd

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

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dataSet**.**iloc[:, [2,3]]**.**values

y = dataSet**.**iloc[:, 4]**.**values

        #*Feature-Scaling after Data Split*

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)

#*Feature-Scaling*

**from** sklearn**.**preprocessing **import** StandardScaler

st\_x= **StandardScaler**()

X\_train= st\_x**.fit\_transform**(X\_train)

X\_test= st\_x**.transform**(X\_test)

#*Fit train set to Random forest classifier: No parmeter is needed*

**from** sklearn**.**ensemble **import** RandomForestClassifier

clsFier = **RandomForestClassifier**(n\_estimators= 10, criterion="entropy", random\_state=0)

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = clsFier**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

#*Visualising the Training set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('Random forest (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*Visualising the Test set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('Random forest (Test set)')

pLt**.xlabel**('Age')

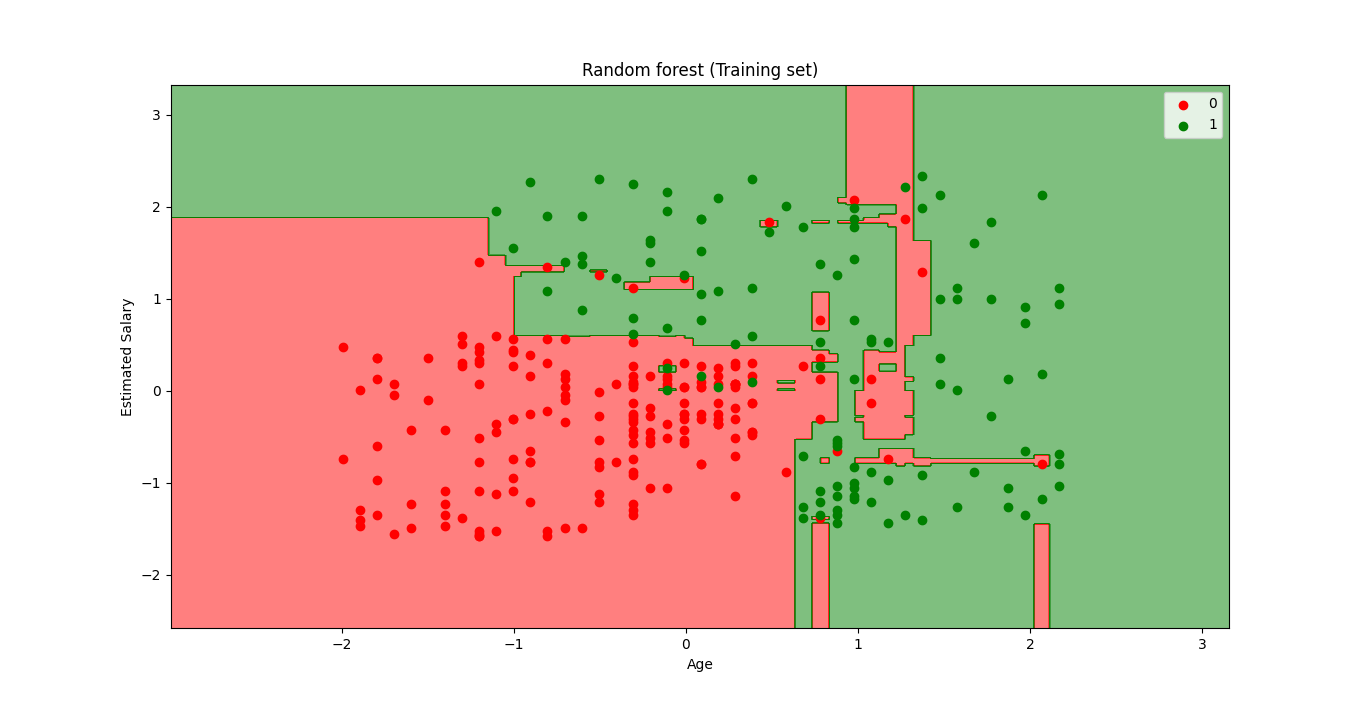
pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

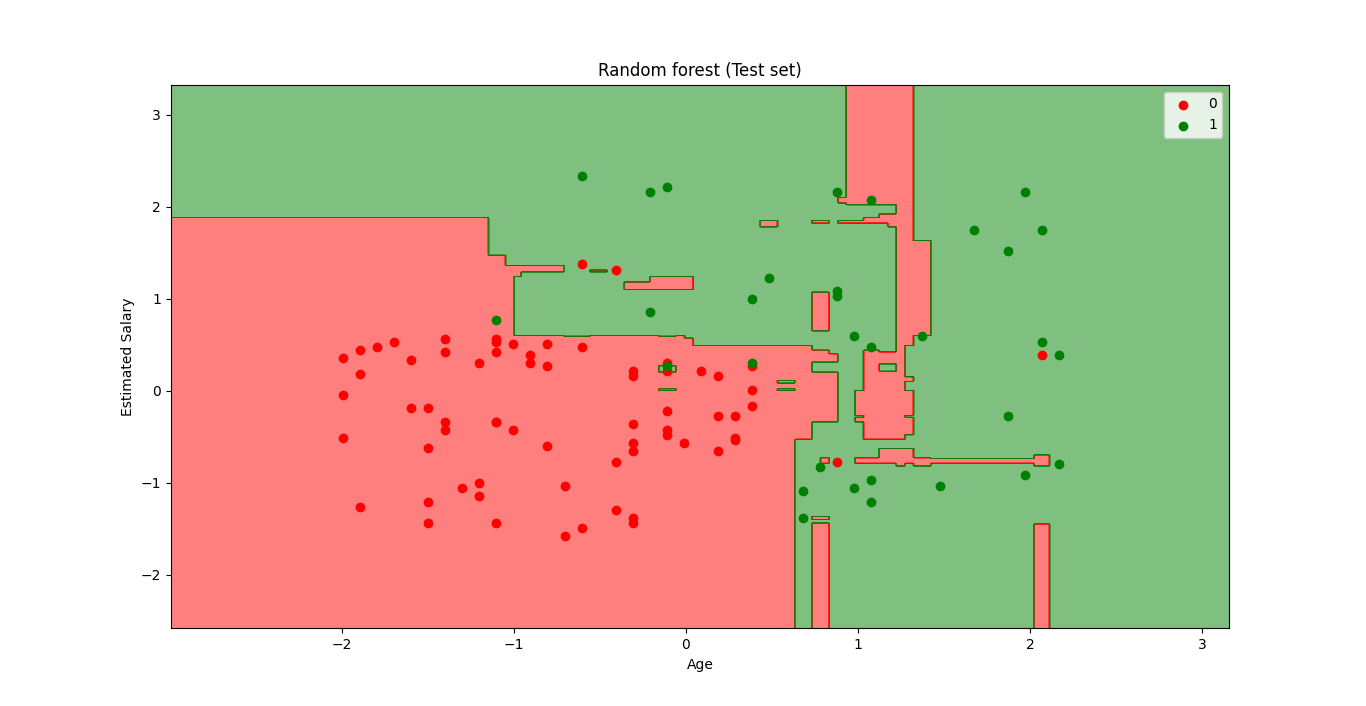
pLt**.show**()

#*python prctc\_RFC.py*

Train set Plot



Test set Plot



* Some overfitting is appearing in the train set. Notice the *small red region* in the *Green area*.
* So as a conclusion we had the best classifier as Kernel-SVM classifier and Naïve Bayes Classifier. For our problem we just gonna use Kernel-SVM classifier.
* We can start thinking how to choose the best classifier for our problem. And it's always related to this battle between accuracy (getting the maximum number of correct predictions) and preventing overfitting.