TRUMAN STATE UNIVERSITY

Professional Program in DATA SCIENCE

PDAT617: Python

Module 7 – Classification

DISTINCT BY DESIGN

7.1What is supervised learning

- A supervised learning algorithm takes a known set of input data (the training set) and known responses to the data (output), and trains a model to generate reasonable predictions for the response to new input data.
- Use supervised learning if you have existing data for the output you are trying to predict.
- There are two main types of supervised learning problems: they are classification that involves predicting a class label and regression that involves predicting a numerical value.

7.1 What is Classification?

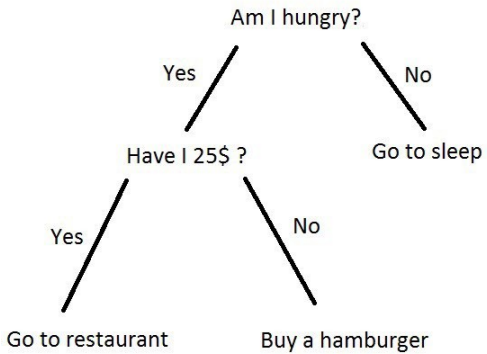
- Given data on predictor variables (inputs, X) and a **categorical response variable** (output, Y) build a model for:

  - Predicting the value of the response from the predictors.
  - Understanding the relationship between the predictors and the response
  - e.g. predict a person's **5-year-survival (yes/no)** based on their age, height, weight, etc

7.1 Types Of Classification

Decision Tree

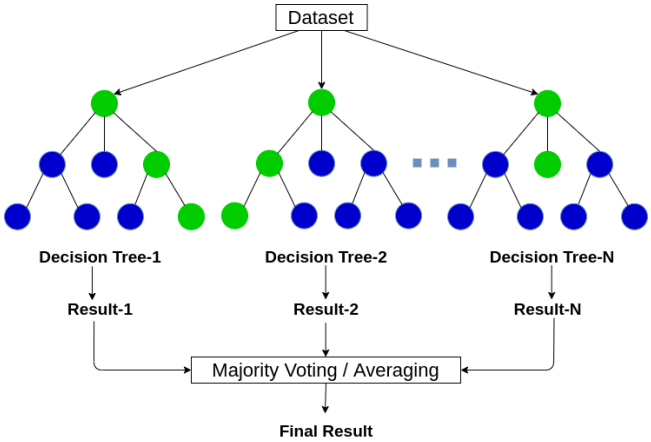
- ✓Graphical representation of all the possible solutions to a decision
- ✓Decisions are based on some conditions
- ✓Decision made can be easily explained



7.1 Types Of Classification

Random Forest

- Builds multiple decision trees and merges them together
- More accurate and stable prediction
- Random decision forests correct trees' habit of overfitting to their training set
- Trained with "bagging" method



7.1 Types Of Classification

- Naïve Bayes algorithms: is a classification technique based on applying Bayes' theorem with a strong assumption that all the predictors are independent to each other.

- K-nearest neighbors (KNN) algorithm: uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set

7.2 Decision Tree

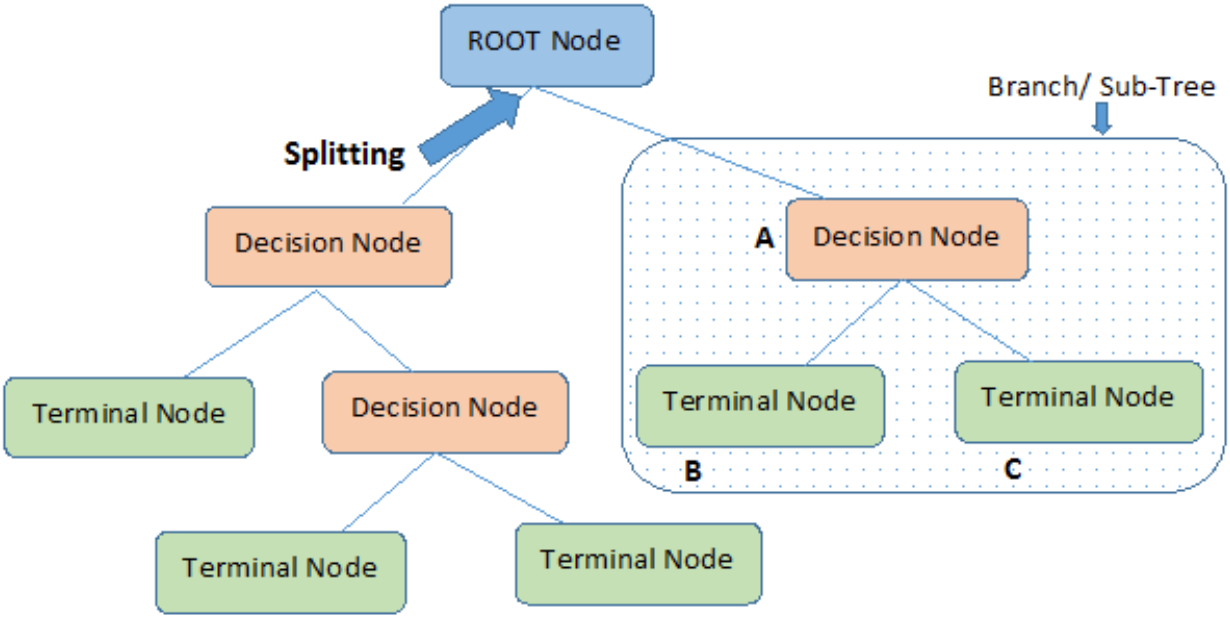
- Decision trees have a long history in machine learning
- The first popular algorithm dates back to 1979
- Very popular in many real world problems
- Intuitive to understand
- Easy to build

7.2 Decision Tree

- Decision Tree Terminology

  - Root Node: A root node is at the beginning of a tree. It represents entire population being analyzed.
  - Splitting: It is a process of dividing a node into two or more sub-nodes.
  - Decision Node: When a sub-node splits into further sub-nodes, it's a decision node.
  - Leaf Node or Terminal Node: Nodes that do not split are called leaf or terminal nodes.
  - Pruning: Removing the sub-nodes of a parent node is called pruning. A tree is grown through splitting and shrunk through pruning.
  - Branch or Sub-Tree: A sub-section of decision tree is called branch or a sub-tree, just as a portion of a graph is called a sub-graph.
  - Parent Node and Child Node: These are relative terms. Any node that falls under another node is a child node or sub-node, and any node which precedes those child nodes is called a parent node.

7.2 Decision Tree



Note:- A is parent node of B and C.

7.2 Decision Tree

- Node splitting, or simply splitting, is the process of dividing a node into multiple sub-nodes to create relatively pure nodes.
- Decision Tree Split Method:

  - Continuous Target Variable(Regression Problems): Reduction in Variance

  - Categorical Target Variable:

    1. Gini Impurity
    2. Information Gain
    3. Chi-Square

7.2 Decision Tree

- Information Gain: is the decrease in entropy after a dataset is split on the basis of an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain.
  - Information Gain = 1- Entropy

$$Entropy = -\sum_{i=1}^{n} p_i \log_2 p_i$$

7.2 Decision Tree

- Gini Impurity: The measure of impurity(or purity) used in building decision tree in CART.

$$Gini\ Impurity = 1 - \sum_{i=1}^{n} p_i^2$$

- Lower the Gini Impurity, higher is the homogeneity of the node.

7.2 Decision Tree

- Chi Square: is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node.

$$Chi\text{-}Square = \sqrt{\frac{(Actual - Expected)^2}{Expected}}$$

- Higher the value, higher will be the differences between parent and child nodes, i.e., higher will be the homogeneity.

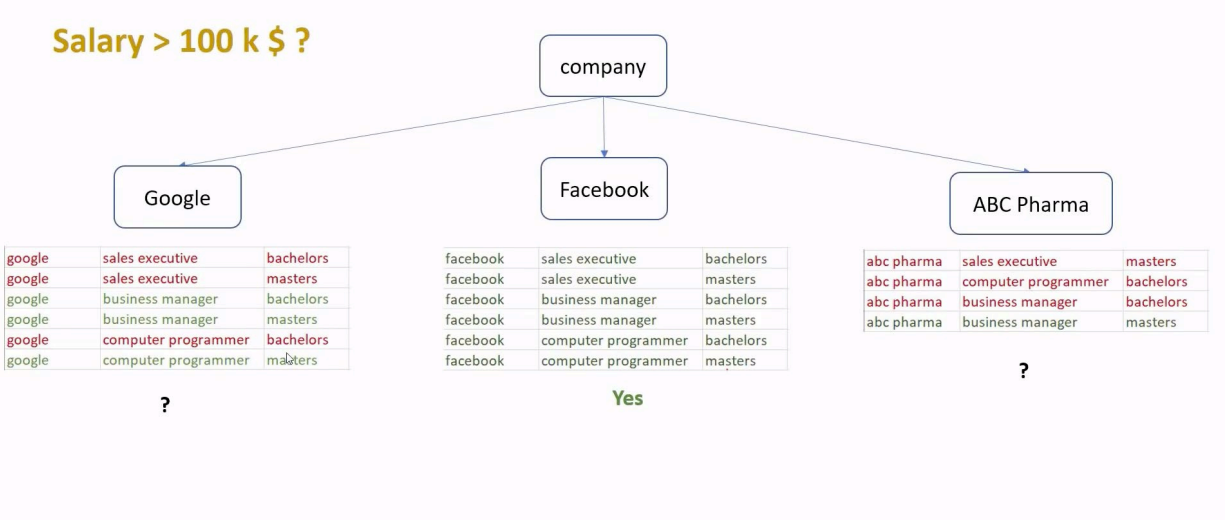
7.2 Decision Tree

- Example: We want to predict if a person's salary is more that 100k$ based on company, job title or the degree.

Company	Job	Degree	Salary_more_then_100k
google	sales executive	bachelors	0
google	sales executive	masters	0
google	business manager	bachelors	1
google	business manager	masters	1
google	computer programmer	bachelors	0
google	computer programmer	masters	1
abc pharma	sales executive	masters	0
abc pharma	computer programmer	bachelors	0
abc pharma	business manager	bachelors	0
abc pharma	business manager	masters	1
facebook	sales executive	bachelors	1
facebook	sales executive	masters	1
facebook	business manager	bachelors	1
facebook	business manager	masters	1
facebook	computer programmer	bachelors	1
facebook	computer programmer	masters	1

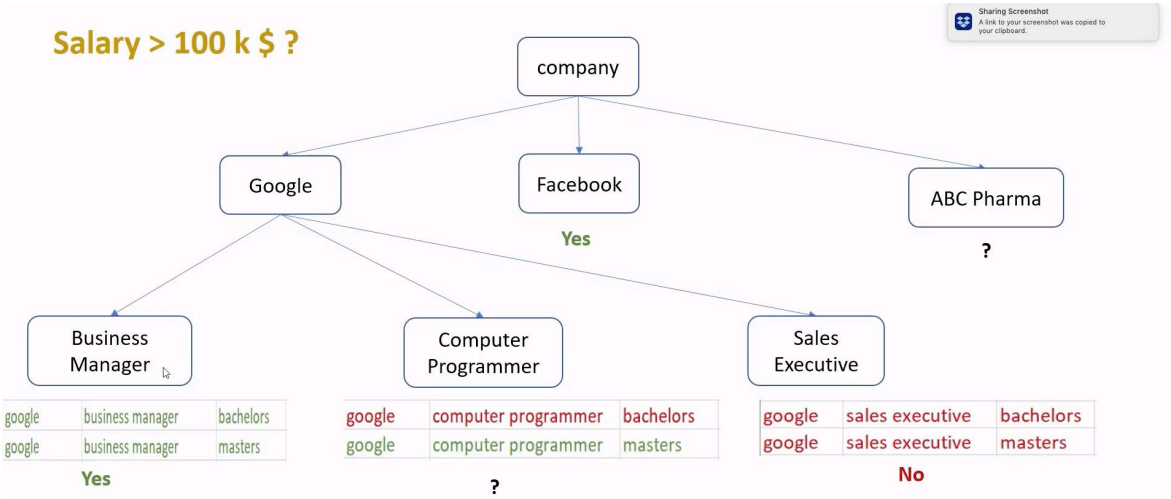
7.2 Decision Tree

- First, we may split data by company



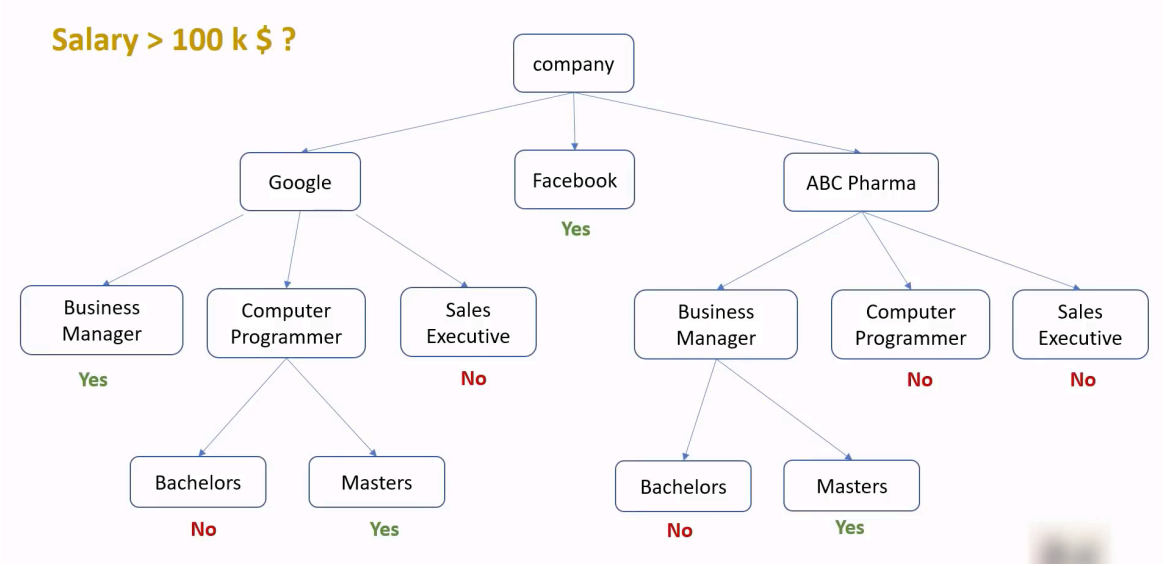
7.2 Decision Tree

- Then, we can split each company by job title



7.2 Decision Tree

- Finally, we may get a tree like this:



7.2 Decision Tree


```
import pandas as pd
df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis=1)
target = df['salary_more_then_100k']
from sklearn.preprocessing import LabelEncoder
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()
inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_job.fit_transform(inputs['job'])
inputs['degree_n'] = le_degree.fit_transform(inputs['degree'])
inputs_n = inputs.drop(['company','job','degree'],axis='columns')
```

]:

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

7.2 Decision Tree


```
from sklearn import tree
model = tree.DecisionTreeClassifier()
model.fit(inputs_n, target)
model.score(inputs_n,target)
model.predict([[2,1,0]])
model.predict([[2,1,1]])
tree.plot_tree(model)
```

```
model.score(inputs_n,target)
```

```
1.0
```

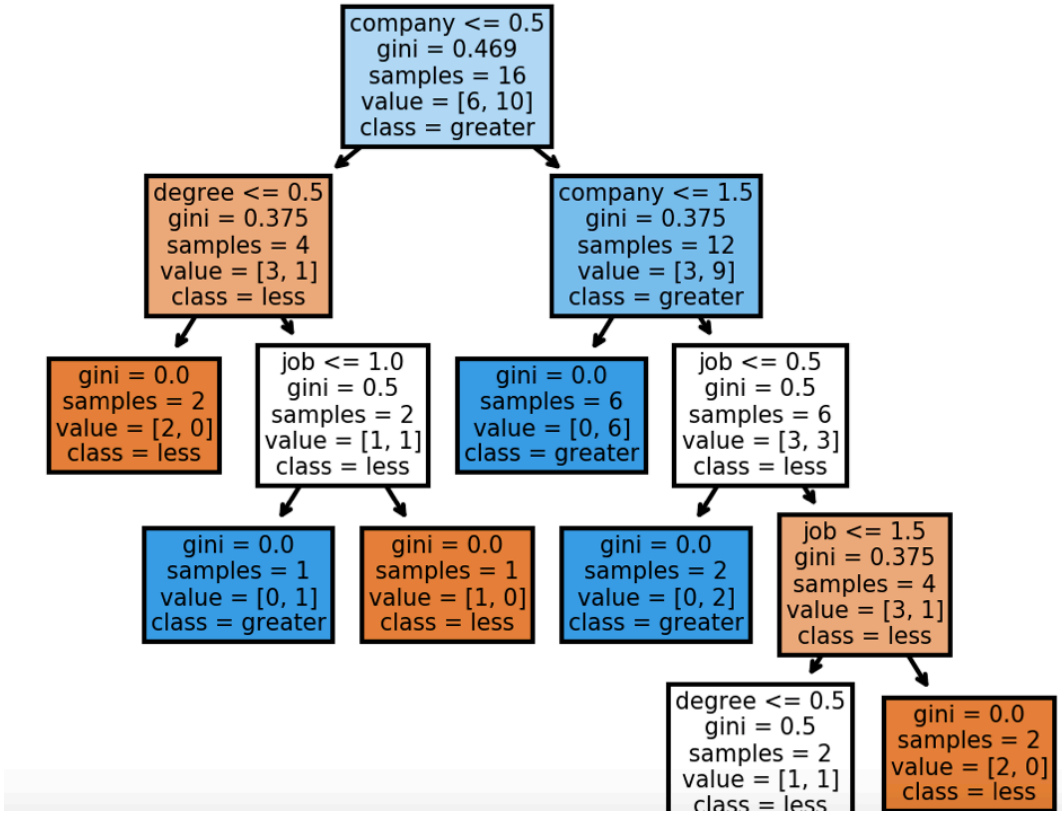
```
# Is salary of Google, Computer Engineer, Bachelors degree > 100 k ?
model.predict([[2,1,0]])
```

```
array([0])
```

```
# Is salary of Google, Computer Engineer, Masters degree > 100 k ?
model.predict([[2,1,1]])
```

```
array([1])
```

7.2 Decision Tree



7.3 Random Forest

What is Random Forest

1. Random Forest is the most used supervised learning algorithm for classification and regression.

2. An ensemble classifier using many decision tree models

3. Accuracy and variable importance information is provided with the results

7.3 Random Forest

Why Random Rorest:

- NO overfitting: Use Multiple trees to reduce the risk of overfitting
- High Accuracy: For large data, it produce high accurate predictions
- Estimates Missing Data: Random Forest can maintain accuracy when a large proportion of data is missing.

7.3 Random Forest

- Random Forest Terminology

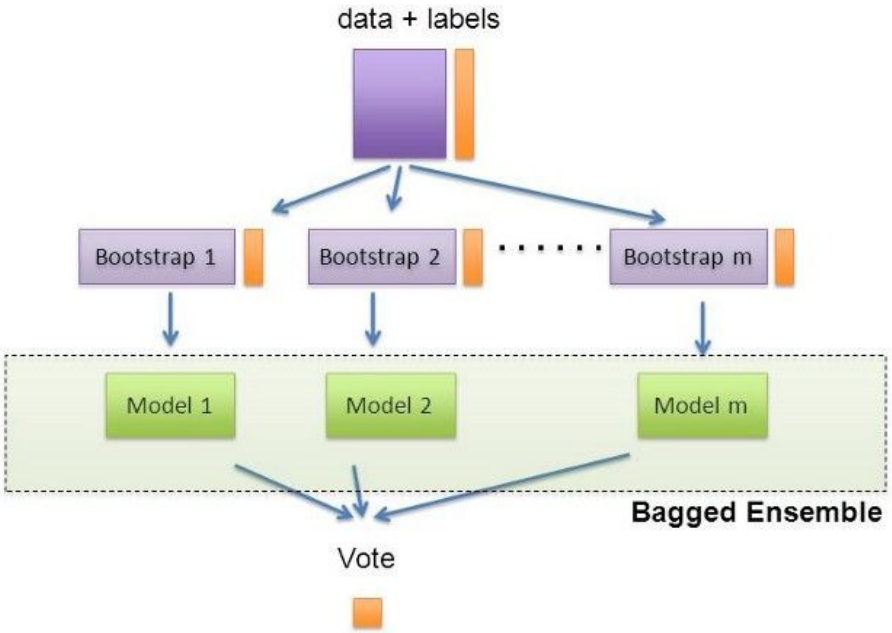
  - Ensemble Learning: Random Forest use ensemble learning method in which the predictions are based on the combined results of various individual result.

  - Ensemble Method: Bagging and Boosting.

    - Bagging(Used in RF): **Bootstrapping** the data and using it aggregate to make a decision is known as bagging (parallel algorithm).
    - Boosting is training a bunch of individual models in a sequential way.
      Each individual model learns from mistake made by the previous model

7.3 Random Forest

"Bagging" : **B**ootstrap **AGG**regat**ING**



7.3 Random Forest

- Before we dive into bagging algorithm we should go over bootstrap:

1. Used in statistic when you want to estimate a statistic of a random sample (a statistic: mean, variance, mode, etc...)
2. Using bootstrap we diverge from traditional parametric statistic, we do not assume a distribution of the random sample
3. What we do is sample our only data set (aka random sample) with replacement. We take up to the number of observations in our original data.
4. We repeat step 3 for a large number of time, B times. Once done we have B number of bootstrap random samples.
5. We then take the statistic of each bootstrap random sample and average it

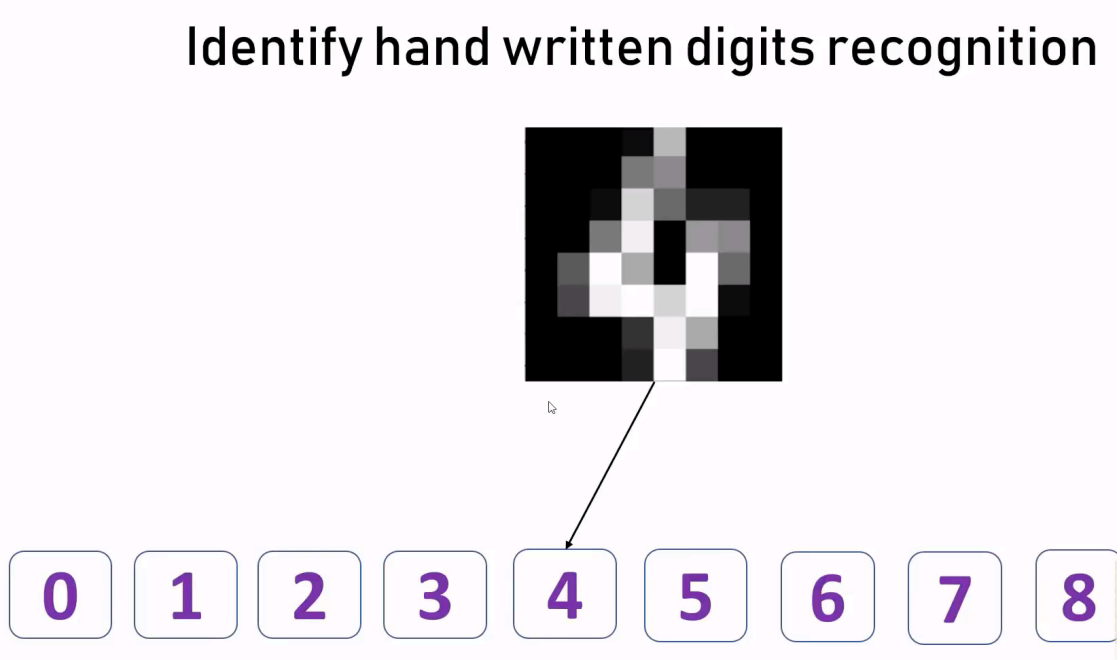
7.3 Random Forest

- Bagging(Bootstrapping Aggregation) Algorithm:

1. Take a random sample of size N with replacement from the data
2. Construct a classification tree as usual
3. Assign a class to each terminal node, and store the class attached to each case coupled with the predictor values for each observation
4. Repeat Steps 1-3 a large number of times.
5. For each observation in the dataset, count the number of times over tress that it is classified in one category and the number of times over trees it is classified in the other category
6. Assign each observation to a final category by a majority vote over the set of tress. Thus, if 51% of the time over a large number of trees a given observation is classified as a "1", that becomes its classification.

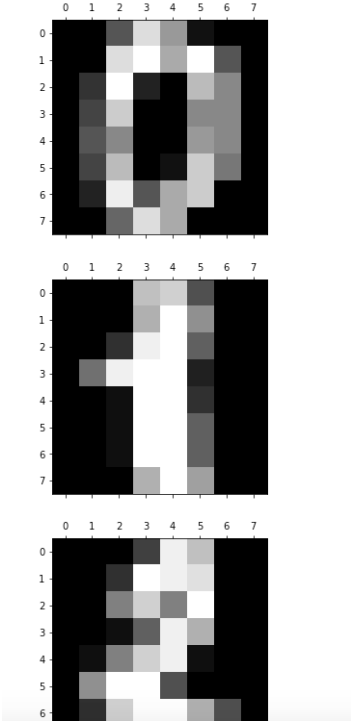
7.3 Random Forest

- Example:



7.3 Random Forest


```
import pandas as pd
from sklearn.datasets import load_digits
digits = load_digits()
%matplotlib inline
import matplotlib.pyplot as plt
plt.gray()
for i in range(4):
    plt.matshow(digits.images[i])
```



7.3 Random Forest


```
df = pd.DataFrame(digits.data)
df['target'] = digits.target
# Train and the model and prediction
X = df.drop('target',axis='columns')
y = df.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=15, criterion='entropy')
model.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=15,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

7.3 Random Forest


```
model.score(X_test, y_test)
y_predicted = model.predict(X_test)
y_predicted
from sklearn.metrics import
confusion_matrix
cm = confusion_matrix(y_test,
y_predicted)
Cm
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
model.score(X_test, y_test)
```

```
0.975
```

```
Text(69.0, 0.5, 'Truth')
```