CPSC 483, Project 2

Team Name: ‘AKMMR’

Team Members:

|  |  |  |  |
| --- | --- | --- | --- |
| Serial Number | Name | CWID | Email ID |
| 1 | Asit Kumar Dash | 893423350 | asitdash@csu.fullerton.edu |
| 2 | Kashyap Sodha | 893260984 | ksodha@csu.fullerton.edu |
| 3 | Melika Amini | 893423350 | melika\_am@csu.fullerton.edu |
| 4 | Milan Ghori | 893585703 | ghorimilan1994@csu.fullerton.edu |
| 5 | Rishabh Sharma | 893463695 | rishabh230795@csu.fullerton.edu |

Under guidance of Prof. Christopher Ryu

Machine Learning – CPSC483

Department of Computer Science

California State University, Fullerton

b) Brief description of the software and the tool

Description of Software and Tool:

The project was done in python 3 using the sci-kit learn library. The ide used is spyder. Spyder provides an IPython console which makes it easy for the developer to check the instances of the code at any point. We also used jupyter-notebook to research on some of our findings. Scikit learn provides the user with many tools to execute high end machine learning tasks. Scikit learn is developed by a large community of the developers and machine learning experts therefore new techniques are included in it in a very short span of time.

In order to help the users who have a hard time to find which algorithm should be used Andres Muller the developer of scikit learn made a very user friendly flow chart.

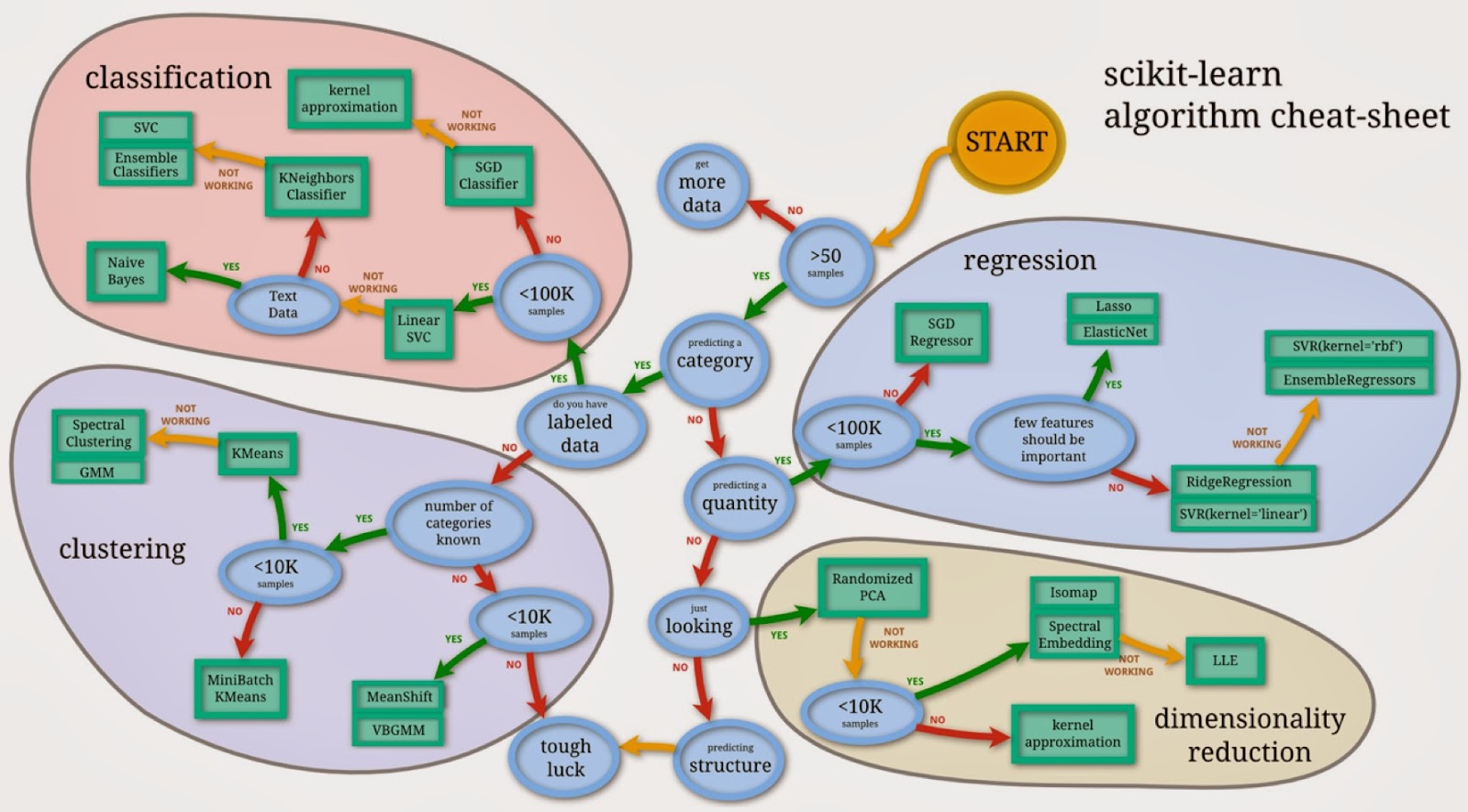


Image Source: <https://www.oreilly.com/ideas/six-reasons-why-i-recommend-scikit-learn>

C) i: Conduct classification using each of those methods specified above and come up with the best classifier from each method

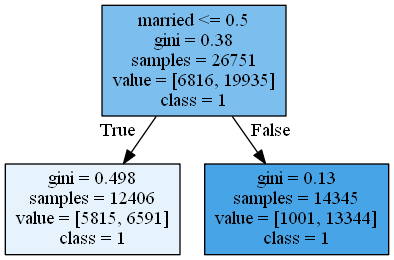
Answer: For classification we first start with the decision tree based classification method. Decision trees is a non-parametric supervised learning method that is used for classification. The goal is to **create a model that predicts the value of a target variable by rules that will be inferred with the help for the data.**

Decision trees work according to the following way:

1. Place the best attribute of the dataset at the **root** of the tree.
2. Split the training set into **subsets**. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find **leaf nodes** in all the branches of the tree.

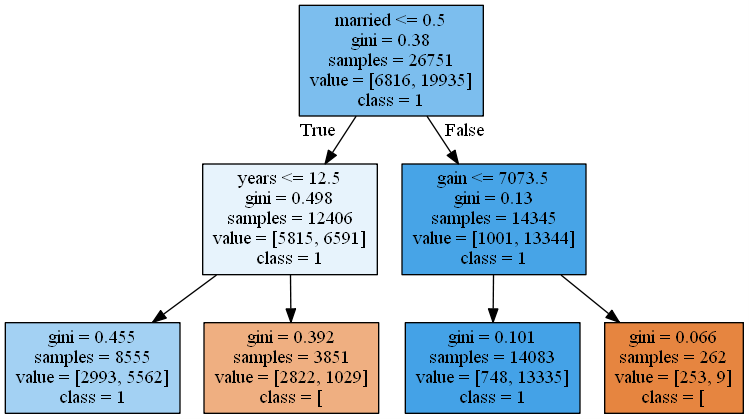
We conduct the decision tree classification and use different level of tree depth to predict the data. This give us different decision tree models that can be used to find the parameter of the dataset.

Decision tree with depth 1.



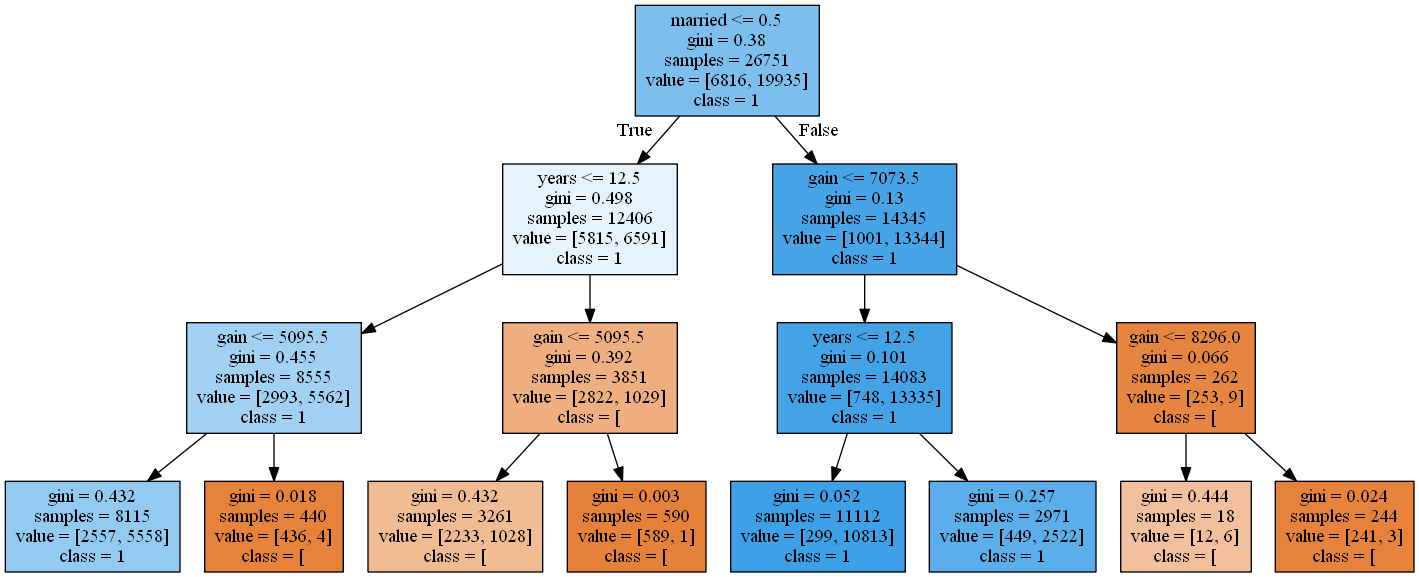
The above decision tree takes married as a classification status based on the **gini index** method and gives the above model. We can not use this method because both the datasets are classified to class 1. Class 1 means that the total income will be less that 50K and class 2 means that total income will be more than 50K. **Left branch is also the true condition and right branch is always the false condition.**

Then we increase the depth of the tree to 2.



New model gives us two leaf node that are the lowest level node with class 0 and two leaf node with class 1. Therefore at level 2 the person who is married-civ-spouse and whose years of education is more than 12.5 years can make more than 50k or else the person who has any other marital status but has a capital gain of more than 7073.5 can make more than 50K. At every node the parameter to decide the root value for that node is calculated using the gini index for all the samples at that node.

We keep on increasing our tree now to level 3 and we get the following.



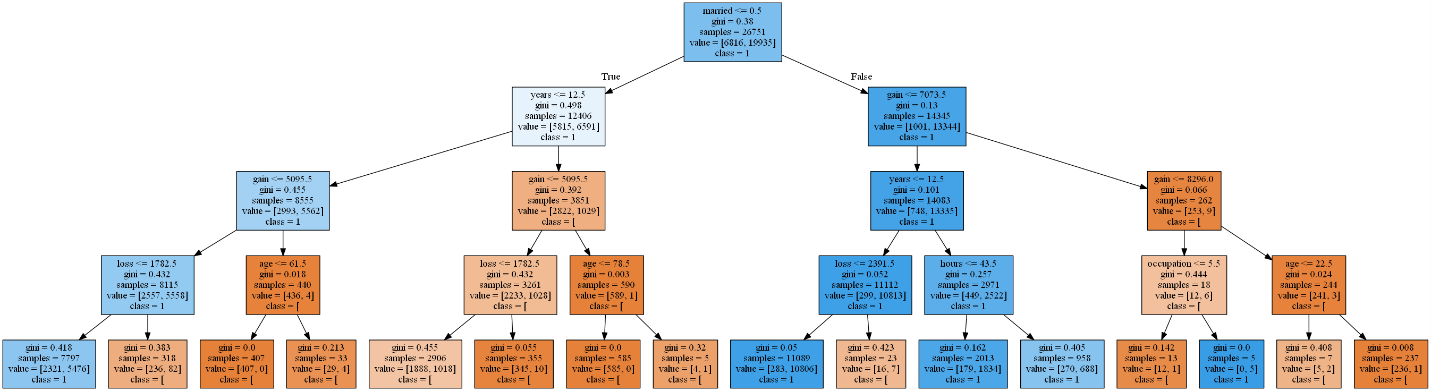
Same way as we following the above method we can find out the profile for people who will earn more than 50K.

At level 3 we get that the people who are married-civ-spouse, have more than 12.5 years of education will get more that 50K, People who are married-civ-spouse, have less than 12.5 years of education but has a capital gain more than 5095.5 will earn more than 50K and also people who have any other marital status but has a gain of more than 7073.5 will earn more than 50K.

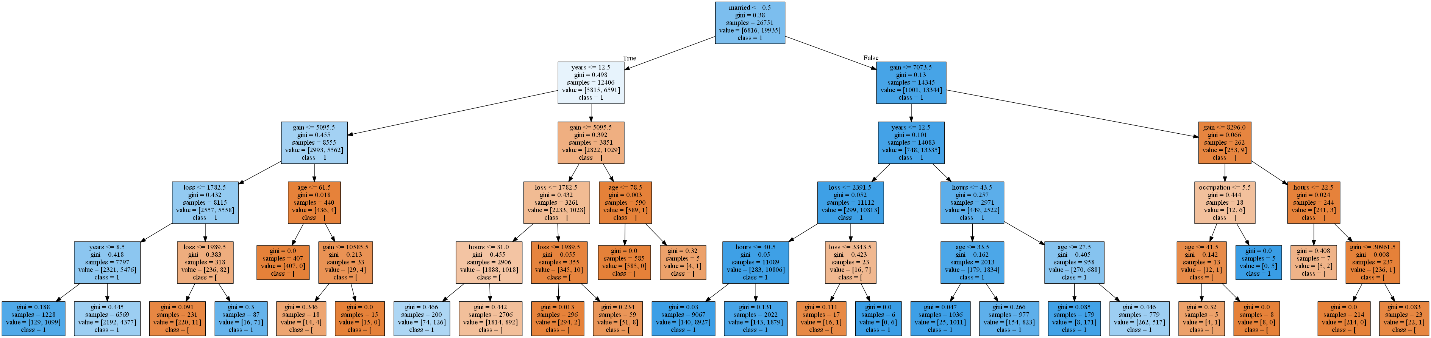
All the trees represent some model that can be used for classification and these models are the nodes represent the rules.

We made the trees to a depth of 6 and the following trees models were obtained.

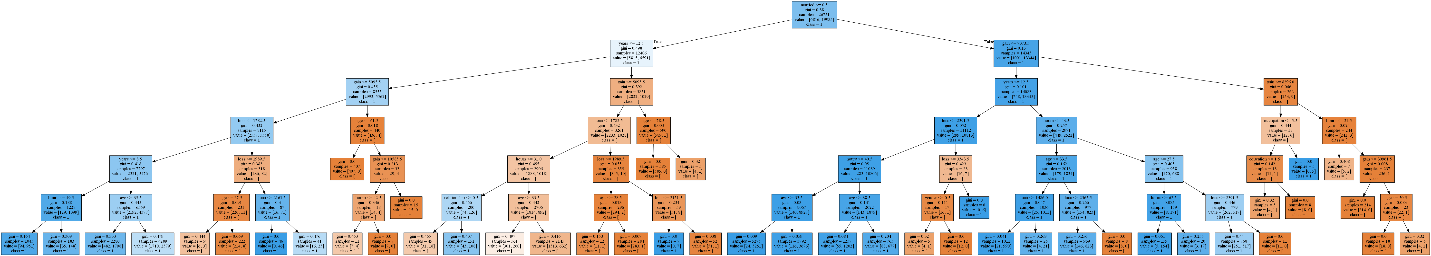
At the depth of 4:



At a depth of 5:



At a depth of 6:



We can keep on increasing the depth of our model to a specific tree depth that we want. However if we don’t pass any maximum depth value then the **algorithm keeps on computing the model till it doesn’t reach a leaf node that has an entropy of 0.**

**Naïve Bayes Algorithm.**

Naïve Bayes is a classification technique that is based on the bayes theorem. Naïve Bayes works in a way that it assumes that the presence of one feature in the dataset is not related to any other feature in the dataset. It states that if we have multiple features that are independent of each other and we want to find the class of a given sample we can do that by finding the probability of that class which is then multiplied to the probability of samples given that class is true and the product is divided by the probability of the sample.

The same can be explained mathematically by following:

P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}
                                 {P(x_1, \dots, x_n)}

So, since we took a small sample out to predict it we first get the probability of the classes.

**P(0) = 0.254**

**P(1)=0.746**

Similarly we predict rest of the probabilities.

We calculate P(Age|0), P(work class|0), P(education|0), P(years|0) and so on and then we calculate P(Age|1), P(Word class|1), P(education|1), P(years|1) for all the features. ‘1’ represents the income less than equal to 50K and ‘0’ represents the income more than 50k.

Once we get this value we can pass our profile of the user that we want to classify and the algorithm will then compute the necessary computation to decide the class of the new algorithm.

If the probability of P(X|0)P(0) is more that P(X|1)P(1) where ‘X’ is the set of all features then the sample will belong to class 0 that is the person will earn more than 50K else the person will earn less than 50K.

**Multilayer Perceptron:** A multilayer perceptron is a class of feedforward network. An MLP consists of at least three layers of nodes. Multilayer perceptron utilizes supervised learning technique called backpropagation for training. At every neuron other than the input layer uses a nonlinear activation function. Given a set of features X = {x_1, x_2, ..., x_m} and a target y, it can learn a non-linear function approximator for classification.

The following figure shows one hidden layer MLP with scalar output.

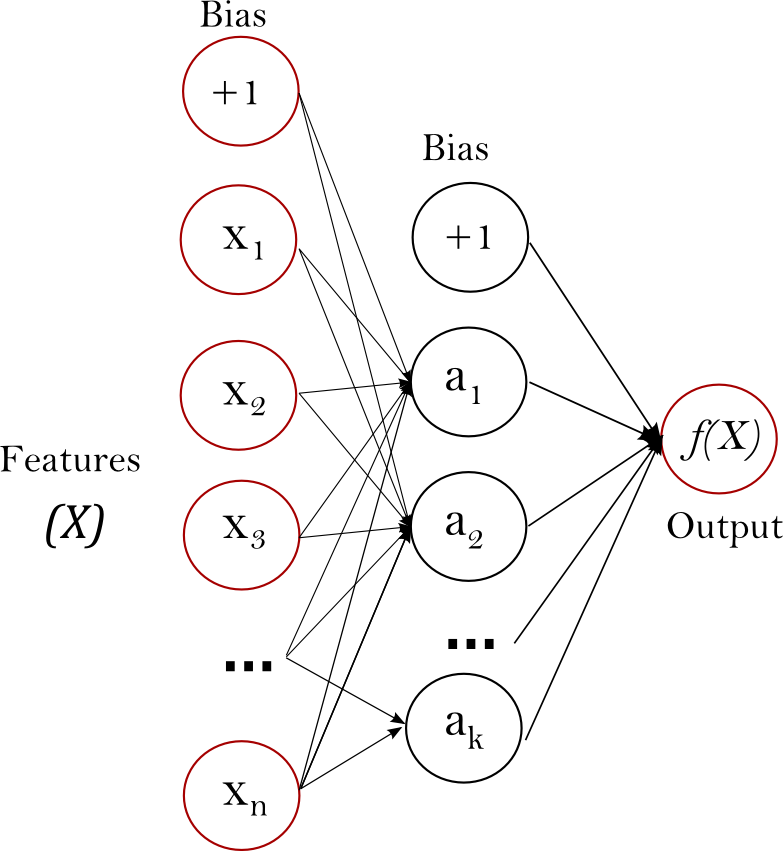


Image Source: <http://scikit-learn.org/stable/modules/neural_networks_supervised.html>

The leftmost layer, known as the input layer, consists of a set of neurons \{x_i | x_1, x_2, ..., x_m\} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation w_1x_1 + w_2x_2 + ... + w_mx_m, followed by a non-linear activation function g(\cdot):R \rightarrow R - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

We used the MLP and we tuned the neural network for 200 iterations with a learning rate of 0.001. We used the relu function as the activation. Relu is max(0,f(x)), which means that we take only the values which is the maximum to activate the neuron is passed to the next layer.

We used a multilayer perceptron with one hidden layer and 6 neuron in hidden layer and one neuron in output layer.

After optimization we get the weights associated with the input layer as.

Winput(i,j) = [[-3.46719758e-01, 4.92150458e-02, -1.36159538e-01, -9.31604169e-02, 1.28601954e-01, 8.84123701e-02], [-4.90246082e-01, 2.76595447e-01, 7.25476104e-01, -1.65516176e-01, -8.45550038e-02, 3.44927093e-01], [ 1.44260992e-02, -2.79276684e-01, 5.41365255e-02, 4.95831289e-01, 1.49782433e-01, -1.98106387e-01], [ 5.90288083e-02, 1.47479759e-01, -4.22021342e-02, -2.54312559e-01, 3.72607440e-01, -2.33973552e-01], [ 6.93767400e-01, -4.53040065e-01, -1.34728777e+00, -9.33065478e-01, 1.39763696e+00, 5.05759571e-01], [-4.55201210e-01, 4.75801895e-03, -4.59610198e-01, -5.66213771e-01, 4.16992178e-01, -4.61729438e-02], [-1.00889069e-01, -1.27129181e-01, -4.47497806e-01, 5.85257893e-02, 1.27125139e+00, 1.00847159e-01], [-2.51451730e-01, -2.48128366e-01, -1.22665211e-02, -4.15188731e-01, -1.95221641e-01, -2.80903877e-01], [-8.08476223e-02, 1.34323670e-01, -5.48780670e-05, 2.16320909e-01, 3.44416503e-01, 1.93581365e-01], [-8.98562905e-02, -3.72198718e-02, 3.26406201e-01, -7.16946390e-02, -4.38152110e-02, 1.34486349e-01], [ 4.01930344e-01, 1.08149899e-01, 2.75641353e-02, 4.77832309e-01, 2.29520807e-02, 2.24287664e-01]]

The weight at the first array shows the weight associated with the first neuron of the input layer to the neurons of the hidden layer.

Whidden(j,o) = [[ 0.27652175], [-0.90640642], [-0.27315587], [-0.25784071], [ 0.16454486], [ 0.6245477 ]]

The weight associated with the neuron of the hidden layer to the output layer.