K means clustering

Basically, clustering is a technique for finding similarities between data rows in a dataset.

In order to realize clustering, we can use unsupervised learning, the process of finding data without any given classification and without being driven by a specific target.

One of the more famous unsupervised learning algorithms is KMean Clustering.

Given k, the k-mean algorithm work as follows :

1)choose k (random)datapoints called also seeds to be the initial centroid clusters centre

2)Assign each datapoint to the closest centroid

3)Recompute the centroids using the cluster memberships

4)If a convergence criterion is not met up steps 2 and 3 must me repeated

Here is a practical application of K_means clustering using a software tool such as Python.

First of all we need to need to load our dataset.

```
import pandas as pd
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from io import StringIO
from mpl_toolkits.mplot3d import Axes3D


csvTrain = StringIO("""User,Beatles,Beach Boys,Metallica
Nils,9,8,4
Anita,8,8,5
Tore,4,2,1
Solveig,3,2,2
Vibeke,2,4,2
Kristine,9,8,8
Ola,8,8,8
Anne,8,9,7
Per,1,1,6
Sissel,2,1,9
Ole,1,2,7
Sigurd,1,1,8
""")

df=pd.read_csv(csvTrain, sep=',')
print(df.head())
```

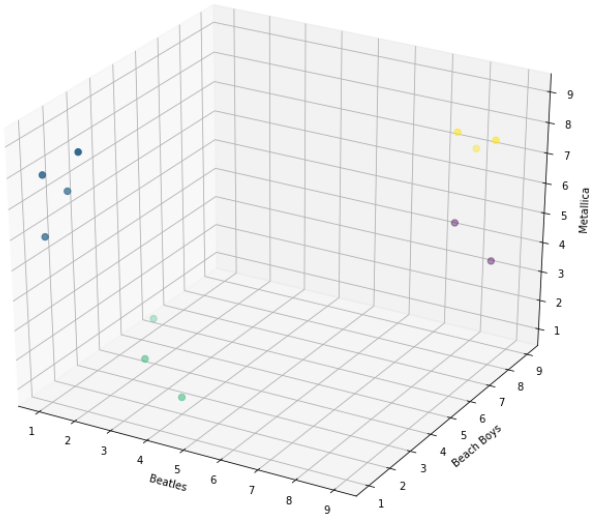
Here's is the out output of the code above(the head()method select the first 5 rows of a dataset.

	User	Beatles	Beach Boys	Metallica
0	Nils	9	8	4
1	Anita	8	8	5
2	Tore	4	2	1
3	Solveig	3	2	2
4	Vibeke	2	4	2

But as we would like just to use the ratings of the Artist we delete the names and appy the k_mean algorithm that takes in input the number of cluster-) and we invoke the fit method on it with our data values.

```
df= df.drop('User', axis=1)
data=df.values
k=4
kmeans=KMeans(n_clusters=k).fit(data)

# Plot 3d graph
fig = plt.figure(figsize=(12,10)) # figsize sets the diagram size
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data[:,0],data[:,1],data[:,2], c=kmeans.labels_, s=40)
ax.set_xlabel('Beatles')
ax.set_ylabel('Beach Boys')
ax.set_zlabel('Metallica')
plt.show()
```

The output is the following:



In [12]:

In the figure above we can see that the algorithm has found 4 clusters ,that in this case can be distinguished through the color of the points or by the position in the graph:

The four points on the top left are those who think heavy metal is cool and old pop music is lame. (Sissel, Ole, Sigurd)

The three on the bottom aren't really much into any of it.

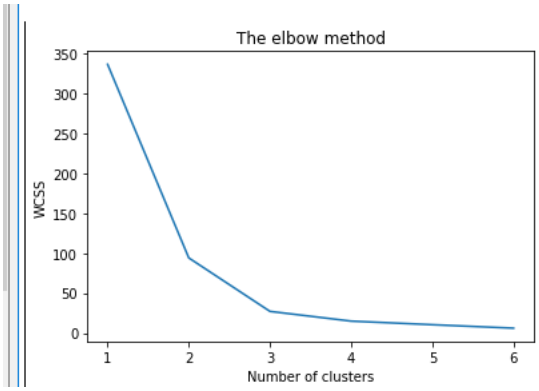
The three on the top right likes everything. (Kristine, Ole, Anne)

The two below likes everything, but are slightly less enthusiastic about heavy metal. (Nils , Anita)

The choice of k and quality of our clusters can be assessed using the elbow method. To do this we try out different values of k, and plot the resulting WCSS (within cluster sum of squares) which is a measure of the mean distance between data points and their cluster centroid.

```
wcss= []#within cluster sums of square
for k in range(1,7):
    kmeans= KMeans(n_clusters=k).fit(data)
    wcss.append(kmeans.inertia_)
# Plot the elbow analysis to find optimal k
plt.plot(range(1, 7), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```

We choose a k that is in the interval(1,7) ,where 7 is not included in the interval and we look for the point that has the sharpest angle.



-In this case the best value of k is 3 so we should divide our data into 3 clusters.

Decisions

A decision tree is a tool that can be used to assist in decision-making. To more specific, it is a technique that can be used to make predictions based on a set of variables. A tree can be trained or "grown" based on a set of known data, and then the tree can make predictions on unknown data. This is called "supervised learning". We train the tree by giving it examples, a set of values and the expected outcome.

It ca also be defined as a tree where each node represents a feature(attribute), each link(branch) represents a decision(rule) and each leaf represents an outcome(categorical or continues value).

Decision trees can be divided into two categories ,based on the outcome:

1)CLASSIFICATION TREE involves that the outcome is categorical(Boolean or set of outcomes)

2) REGRESSION TREE has a numeric (continuous) outcome.

PRACTICAL CASE

TASK'S DESCRIPTION

Given a dataset (a csv files) containing a record for each historical car purchase, with car type, price including all extras and some simple customer information like age, sex and number of children, the task is to find the type of car that nowadays a customer most likely to want to buy and what price they are willing to pay. In this case, a classification tree is suitable to find the car type and that a regression tree is suitable for finding the price.

IMPLEMENTATION OF THE TASK

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Sep  9 17:19:51 2018
4
5 @author: Andrea
6 """
7
8 import pandas as pd
9
10 # Read csv-file from disk into dataframe object
11 df = pd.read_csv('data/cars.csv', sep=';', encoding='utf-8')
12 print(df.head())
13
```

>

The first step consists in loading our dataset containing information about historical purchase.

We also print the first 5 rows of the dataset.

```
In [13]: runfile('C:/Users/Andrea/.spyder-py3/untitled1.py',
wdir='C:/Users/Andrea/.spyder-py3')
     Car  Age Sex  Children   Price
0  Sedan   23   M         0  155041
1  Sedan   24   M         0  155090
2  Sedan   19   M         0  155311
3  Sedan   22   M         0  155733
4  Sedan   21   M         1  155740
```

Then to prepare our dataset to the so called supervised learning ,we covert not numerical values into numerical values e.g. M=1,F=0.

```
cleanup_nums = {"Sex": {"M": 1, "F": 0},
                "Car": {"Sedan": 0, "Station Wagon": 1, "SUV": 2, "MPV": 3, "Sportscar": 4 }}
df.replace(cleanup_nums, inplace=True)
print(df.head())
```

Here is the output:

```
In [17]: runfile('C:/Users/Andrea/.spyder-py3/untitled1.py', wdir='C:/Users/Andrea/.spyder-
py3')
   Car  Age  Sex  Children   Price
0    0   23    1         0  155041
1    0   24    1         0  155090
2    0   19    1         0  155311
3    0   22    1         0  155733
4    0   21    1         1  155740
```

Then, as we want to predict Car and Price we to remove from the columns we used to learn by putting them in separate variables.

```
y = df['Car']
X = df.drop('Car', axis=1)
X = X.drop('Price', axis=1)
print(X.head())
```

```
   Age  Sex  Children
0   23    1         0
1   24    1         0
2   19    1         0
3   22    1         0
4   21    1         1
```

Next step is to split the data horizontally ,so that can we use parts of it to train and grow the tree and the remaining part to see how well the tree performs. In order to do this we need some imports

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
```

We split our data horizontally,

```
# We split the dataset horizontally. We use 20% for testing and 80% for training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

First, we create a tree, with two parameters to control the depth of the tree, and then we perform the training.

```
34 classifier = DecisionTreeClassifier(min_impurity_decrease=0.01, max_depth=5)
35 classifier.fit(X_train, y_train)
```

After having trained the tree,we can do some tests

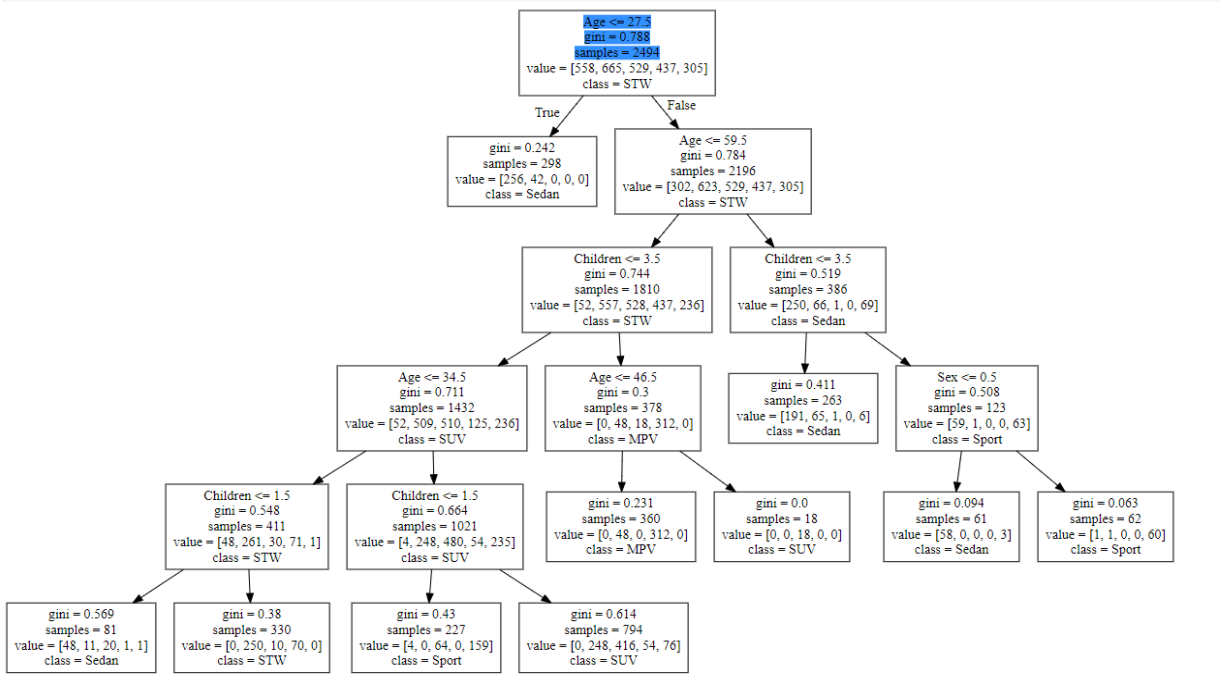
```
# Test tree
y_pred = classifier.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.88      0.97      0.92       146
          1       0.80      0.48      0.60       175
          2       0.58      0.88      0.70       138
          3       0.83      0.76      0.79        90
          4       0.84      0.75      0.79        75

avg / total       0.78      0.75      0.75       624
```

To visualize the tree we can write the following code:

```
# Display tree

dotfile = open("./dtree1.dot", 'w')

ditfile = tree.export_graphviz(classifier, out_file = dotfile, feature_names = X.columns,
class_names=['Sedan','STW','SUV','MPV','Sport'])

dotfile.close()
```

Then we open the dot file in the folder where the python code is saved, and we copy the text in it into the following site to see the tree: : http://webgraphviz.com/



In the example above, we have pruned the default tree by using two parameters, max_depth and min_impurity_decreaseThe accuracy of the tree is shown in the classification_report given above. For each of the 5 classes predicted (the 5 different types of cars), it is shown what percentage of the predictions that were correct. 0.74 average precision means that the tree classified the entries in the training set correctly in 75% of the cases.

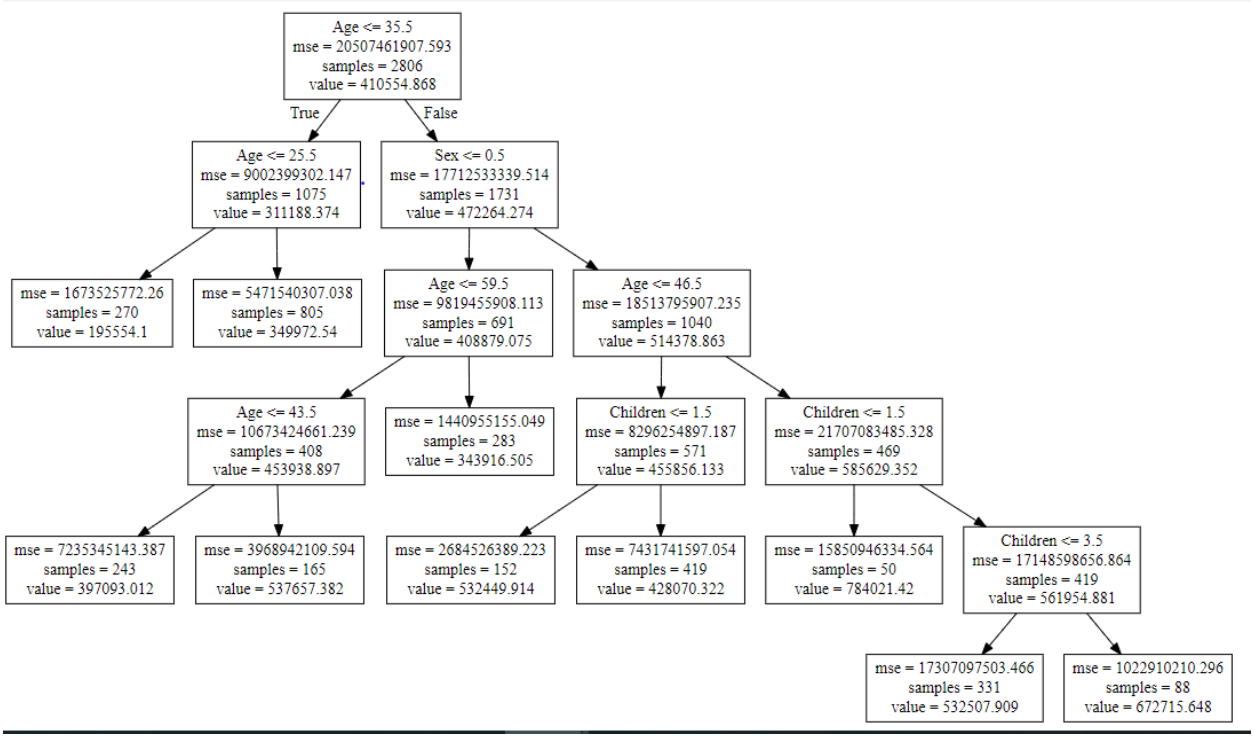
If the tree is too deep without the extra depth giving significant extra precision, it is said to be overfitted. Then it is better to prune the tree to be shallower.

The parameter max_depth is self-explanatory, but min_impurity_decrease is a bit harder to understand. Let's first look at the gini-value in the tree. The gini-value is a measure of inaccuracy. So at a given node in the tree, it says how big a percentage of the classifications would be incorrect if the tree didn't go any deeper. At the root node, the inaccuracy is 79%, which is reasonable given that there are 5 categories of cars and no decisions have been made. The inaccuracy decreases as we go deeper, and the min_impurity_decrease let's us control the minimum amount of decrease that we will allow in order for the tree to grow deeper. A higher value means a shallower tree.

We may also want to find the price that a customer is willing to pay. To do that we need a regression tree. We will do almost exactly the same as we did above, exception we will use a DecisionTreeRegressor instead of a DecisionTreeClassifier. Also we need to prepare the data set a little differently, using 'Price' as the column to predict, and dropping 'Car' from the dataset.

```
# Split the dataset vertically, so that we have the column we are predicting in y and the data in X
y = df['Price']
X = df.drop('Price', axis=1)
X = X.drop('Car', axis=1)
print(X.head())

# We split the dataset horizontally. We use 20% for testing and 80% for training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)

# Grow tree, using max_depth and/or min_impurity_decrease (mse, Mean Squared Error)
regr = DecisionTreeRegressor(max_depth=6, min_impurity_decrease=20000 ** 2)
regr.fit(X_train, y_train)

# Test tree
y_pred = regr.predict(X_test)

print("Training score: " + format(regr.score(X_train,y_train)))
print("Testing score: " + format(regr.score(X_test,y_test)))

# Display tree
dotfile = open("./dtree2.dot", 'w')
ditfile = tree.export_graphviz(regr, out_file = dotfile, feature_names = X.columns)
dotfile.close()
```

Here is the output:


Age <= 35.5
mse = 20507461907.593
samples = 2806
value = 410554.868
True
False
Age <= 25.5
mse = 9002399302.147
samples = 1075
value = 311188.374
Sex <= 0.5
mse = 17712533339.514
samples = 1731
value = 472264.274
mse = 1673525772.26
samples = 270
value = 195554.1
mse = 5471540307.038
samples = 805
value = 349972.54
Age <= 59.5
mse = 9819455908.113
samples = 691
value = 408879.075
Age <= 46.5
mse = 18513795907.235
samples = 1040
value = 514378.863
Age <= 43.5
mse = 10673424661.239
samples = 408
value = 453938.897
mse = 1440955155.049
samples = 283
value = 343916.505
Children <= 1.5
mse = 8296254897.187
samples = 571
value = 455856.133
Children <= 1.5
mse = 21707083485.328
samples = 469
value = 585629.352
mse = 7235345143.387
samples = 243
value = 397093.012
mse = 3968942109.594
samples = 165
value = 537657.382
mse = 2684526389.223
samples = 152
value = 532449.914
mse = 7431741597.054
samples = 419
value = 428070.322
mse = 15850946334.564
samples = 50
value = 784021.42
Children <= 3.5
mse = 17148598656.864
samples = 419
value = 561954.881
mse = 17307097503.466
samples = 331
value = 532507.909
mse = 1022910210.296
samples = 88
value = 672715.648