PATTERN RECOGNITION
ESE 588

Abhishek S Yellanki
Surya Narayana Paritala

Index:

Introduction –

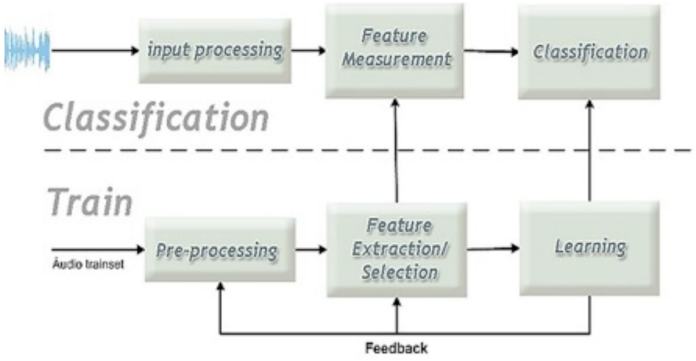
Pattern recognition is the process of classifying input data into objects or classes based on key features. These days with advancement of computers recognizing patterns is a lot easier. Also by recognizing we can easily predict the next set of data and make intelligent decisions based on the known patterns.

What is a Pattern? It is an abstraction, represented by a set of measurements describing a "physical" object. Many types of patterns exists like visual , temporal ,sonic etc.

What is Pattern Class (category)? A pattern class is a set of patterns sharing common attribute. It is a collection of "similar" not necessarily identical objects.

Pattern Recognition process –

All pattern recognition processes can be divided into two stages. Namely Training/Learning which is the first phase and the second stage is detecting or classifying stage.



In the training phase as the name suggests we train our algorithm to learn about the data so that further computations can be done like predicting or classifying the new incoming data into its corresponding classes.

Learning phase for some algorithms is hard and time consuming . The system must be exposed to several examples of each class . And based on the information we create a model for each class. Finally after the algorithm has modelled and all the classes are formed detecting and identifying the new data becomes a natural for the algorithm.

As shown in the above diagram in the training part of the algorithm we first pre-process the data and extract the features from the data and based on the extracted features the algorithm does the learning process.

Whereas in the classification phase our input is processed and the measurement of the features is done and then based on the measurements we do the classification process.

In this report we will mainly focusing on algorithms like K nearest neighbors , Decision tree and Principal component analysis.

K nearest neighbors algorithm –

k nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. This algorithms segregates unlabeled data points into well defined groups.
KNN is an *non parametric lazy learning* algorithm. This is because it does not make any assumptions on the underlying data distribution.

It is also a lazy algorithm. What this means is that it does not use the training data points to do any *generalization* i.e it doesn't learn anything from the training data but "memorizes" the training dataset .
For example in case of Decision tree the training data is used to build a tree which is later used in prediction. Since the algorithm is lazy its training phase is pretty fast .

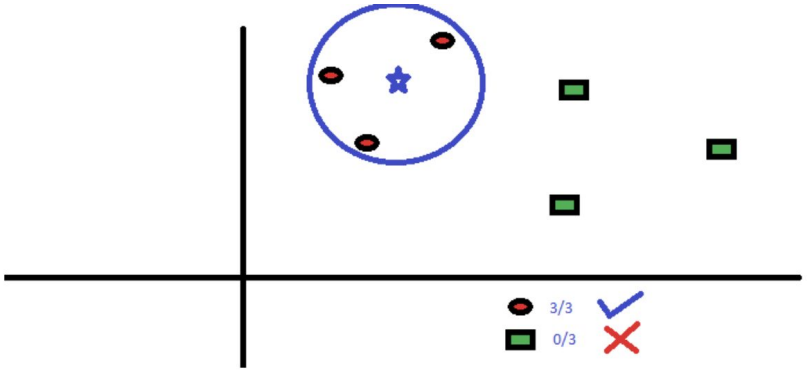
But the drawback of this would be that the "prediction" step in K-NN is relatively expensive. Each time we want to make a prediction, K-NN searches for the nearest neighbor(s) in the entire training set.

KNN (K nearest neighbor's) working explained –

As the name suggests the working of K nearest neighbors algorithms is mainly dependent on calculating the K closest points for the new data point that needs to be classified.

After finding the K nearest points we identify the dominant attribute amongst the K nearest points and assign the unknown data point with that dominant attribute.

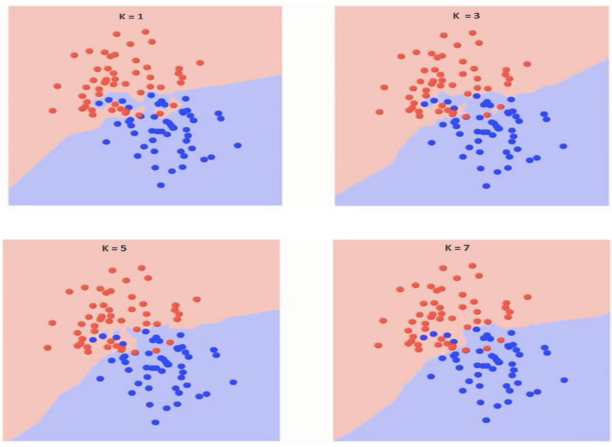
To calculate the K nearest points to the unknown data type , distances are computed between the unknown data point and all the remaining data points and K least distant points are chosen.
In our case we have used Euclidian distance to calculate the distance between two points.



As shown in the diagram if we choose the value of K as 3 then from the unknown data point which in the above diagram is the star , we find out the 3 closest points that are closest to the star.
In the above diagram a circle is drawn covering the 3 closest points near to star point which is unknown.
After finding the K (3) nearest points we compute the dominant attribute among the K points and assign it to the new point. In the above diagram we see that all the 3 nearest points to star are of the type oval . So since in this case the dominant attribute is oval , we assign the unknown star data point as oval .

Choosing K in K NN algorithm –

Selection of k will determine how well the data can be utilized to generalize the results of the kNN algorithm.
For example a large k value has benefits which include reducing the variance due to the noisy data.
But the side effect would be that it develops a bias due to which smaller patterns which may have useful insights are ignored.
Also when K is small, we are restraining the region of a given prediction and forcing our classifier to be "more blind" to the overall distribution.



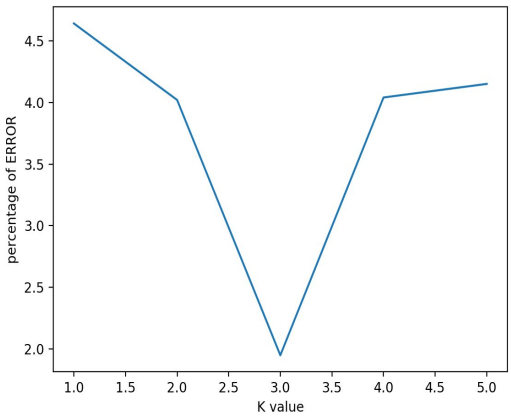
The effect of K on the algorithm can be observed from the diagram.
The scatter plot in the diagram represents the variation in boundaries of a two class data set for k -1 , 3 , 5 and 7.
As K increases the point becomes more aware of its surroundings since it compares with more of its neighbor's.
This leads to smoothing of boundaries which can be seen from the diagram.

Choosing of K value –

In most situations, it is not possible to exactly determine the exact value of K. There are many approaches for doing so. Few of the most commonly followed approaches are choosing an odd number if the number of classes is 2 and another simple approach is to select k as k= sqrt(n) where n is the number of classes.

- In our case we followed an approach which gives a more accurate solution rather than just using thumb rules.
- For multiple values of K we calculate the error percentage and plotted K vs error rate .After plotting we choose the value of K which gives the lowest error rate .

For example in case of IRIS data set the graph obtained for K was as shown in the diagram below.



The plot was drawn between K value and the percentage error obtained for it.
After plotting the graph we choose the K with lowest error percentage.

In this case for k=3 the error was minimum so for the iris dataset we have used K as 3.

Code implemented to run the algorithm –

The coding part for the KNN algorithm was done in Python.
To make the dealing with the datasets easier we have import libraries like pandas using which we can manipulate the data easily.

Apart from that we have imported and used the following libraries –
Math – To perform math functions
numpy, pandas - To easily manipulate the data
matplotlib.pyplot – To draw the graphs
sklearn import **decomposition** – To perform operations like PCA on the data
mpl_toolkits.mplot3d import **Axes3D** – To draw 3d graphs
time - To time the program execution.

The algorithm is divided into modules to increase the readability of the program. The modules are explained below.

Loaddatasets() –
This function performs the task of loading data from the CSV file into the program.
We use inbuilt functions imported from python like csv.reader(csvfile)it
To read the CSV file and save it in the program.
Also this function is responsible for splitting the data into training and testing data in the ratio of 0.67 percent.

getNeighbours() –
This functions takes in the training dataset , test dataset and K value as its parameters.
This function performs task of getting the K nearest neighbors for the each of the points in the test data set.
This function internally calls another function called euclideanDistance() to compute the Euclidean distance.

For each point in the test dataset we compute the Euclidean distance between the test data point

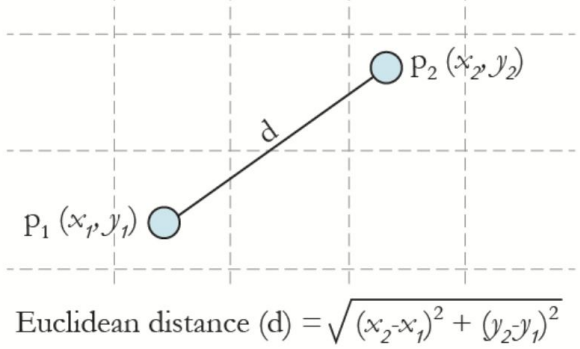
and every point the training data set. And store all distance results in a dictionary data structure with Key as the distance and value as the test point with which the distance was computed.

Further we sort the list in ascending order using inbuilt sort functions and choose the K nearest points from the list.

```
def getNeighbors(trainingSet, testInstance, k): # function to get K nearest neighoburs of a given test data point
    distances = []
    length = len(testInstance) - 1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length) # to compute the eucledian distance
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1)) # sorting the distance in ascending order.
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0]) # choosing the K nereast points from all the distances
    return neighbors
```

euclideanDistance() –

This functions computes the Euclidean distance between two test instances. This functions takes 3 arguments namely instance1 , instance2 and length. Using the Euclidean distance formula the function computes the distance between the two instances and returns it.



The formula for calculating the Euclidean distance is shown in the adjacent diagram. Given two points P1 and P2 we calculate the distance between them by sum of squares of the differences between their corresponding coordinates and then taking the square root.

The code snippet for implementing the Euclidean distance is given below.

```
def euclideanDistance(instance1, instance2, length): # to calculate the eucledean distance between two points.
    distance = 0
    for x in range(length):
        try:
            distance += pow((instance1[x] - instance2[x]), 2)
            return math.sqrt(distance)
        except TypeError:
            print instance2[x] + " and  " + instance1[x]
```

getResponse() –

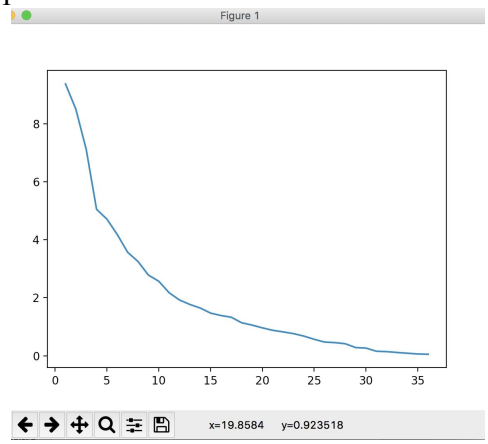
This function takes in arguments as a list which contains the K closest data points for a given test data point. Using the information about the k nearest neighbors which is stored in the list we count the dominant attribute and assign the test data point to the same class as that of the dominant attribute.

```
def getResponse(neighbors): # to count the dominant value among the neighbours and assign that value to the test data point
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:    # couning the number of attributes to determine the dominant value
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]
```

Apart from the above mentioned functions we have functions to implement PCA and plot the graph.

PCA implementation –

To apply PCA on given data we imported PCA from sklearn library and using the inbuilt functions defined in the library like PCA.fit() and PCA.transform() we computed the PCA for the data sets.
Also since applying PCA on data sometimes may lead to loss of important information leading to decrease in accuracy.
In order to avoid the loss of data we first computed PCA on all the attributes and then used **pca.explained_variance_** to find out the amount of variance explained by each of the PCA components.
And then plotted a graph between the PCA components and the variance that PCA components explains.



This was plot of PCA components (x axis) and variances explained by the corresponding PCA component (Y axis) for chess data set that we have used which has 37 attributes.
From the plot we can infer that the most of the variances in the data is explained by PCA 1 – 20 , hence to minimize the loss of data and decrease computation time we reduced the dimensions to 20.
We applied the same procedure for all our data sets to determine to what dimension we must reduce the data so that minimal data loss occurs.

The code snippet for performing PCA and plotting PCA components vs Explained variance is as given below –

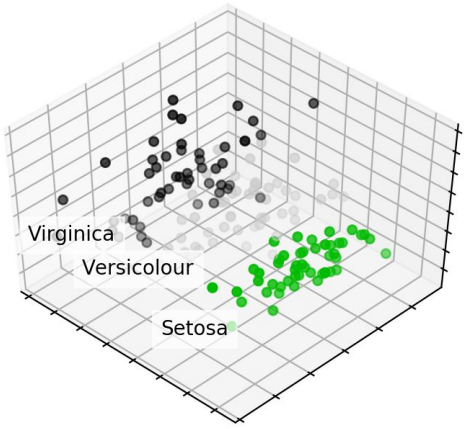
```
pca = PCA(n_components=20)  # to compute PCA for 20 PCA components
pca.fit(df)
df1 = pd.DataFrame(pca.transform(df))
df1.to_csv('chess_pca_20.csv', sep=',')
y = pca.explained_variance_ # to compute the variances expalied by PCA components and plot them
print y
x = np.arange(len(y)) + 1
plt.plot(x, y)
plt.show()
```

Datasets used for implementing KNN and their results –

Iris Data –

- It is a data set which contains details about the types of flowers based on the flower dimensions.
- It has 5 attributes and around 200 instances.
- The data set has 3 classes.

The plot for the data is as follows –



As it can be seen from the plot the data is well clustered.
Since the data was well clustered the data the algorithms efficiency was good.

Results obtained –

For K=3 the accuracy was 98%
Execution time – Around 4 seconds

Chess data set –

This was a data set from UCI machine learning repository.

- This data set is about the moves in a chess game.
- Each instance represents a move in the game.
- There are 37 attributes.
- The first 36 attributes describe the board. The last (37th) attribute is the classification: "win" or "nowin".
- Each feature corresponds to a particular position in the feature-value list
- There are around 3200 instances for this dataset.

Results obtained –
The best value of K for the data was determined to be K =3.

Applying KNN Without PCA –

- The accuracy obtained was 99 %
- Execution time was 253 seconds i.e approx 4 minutes.

Applying KNN With PCA –

- Using PCA we reduced dimensions to reduce the run time even though the efficiency was good.
- Using PCA.explained_variance_ we found out the amount of variance explained by each

of the PCA components. The plot showed that PCA 1- 20 provided most of the variances needed.

- We reduced the dimensions from 36 to 20 and then simulate the algorithm again.
- Accuracy – 89%
- Execution time – Around 20 seconds

IMPLEMENTING DECISION TREE:

Decision trees are a powerful prediction method and extremely popular. Decision trees also provide the foundation for more advanced ensemble methods such as bagging, random forests and gradient boosting.

Introduction to Classification and Regression Algorithms:

- The representation of the CART model is a binary tree. This is the same binary tree from algorithms and data structures, nothing too fancy (each node can have zero, one or two child nodes).
- A node represents a single input variable (X) and a split point on that variable, assuming the variable is numeric. The leaf nodes (also called terminal nodes) of the tree contain an output variable (y) which is used to make a prediction.
- Once created, a tree can be navigated with a new row of data following each branch with the splits until a final prediction is made.
- Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting.
- This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function.
- The split with the best cost (lowest cost because we minimize cost) is selected. All input variables and all possible split points are evaluated and chosen in a greedy manner based on the cost function.
    1. **Regression**: The cost function that is minimized to choose split points is the sum squared error across all training samples that fall within the rectangle.
    2. **Classification**: The Gini cost function is used which provides an indication of how pure the nodes are, where node purity refers to how mixed the training data assigned to each node is.
- Splitting continues until nodes contain a minimum number of training examples or a maximum tree depth is reached.

Decision Tree Algorithm:

1. **Gini index:**
    a. The Gini index is a cost function used to evaluate splits in the dataset.
    b. A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split.
    c. A perfect separation results in a Gini score of 0, whereas the worst case split that results in 50/50 classes in each group results in a Gini score of 1.0 (for a 2 class problem).

gini_index = sum(proportion * (1.0 - proportion))

2. **Create Splits:**

Creating splits involves two parts and also makes use of the Gini Index calculated in the above section.

a. Splitting the dataset
b. Evaluating the splits

Splitting the dataset:

- Splitting a dataset means separating a dataset into two lists of rows given the index of an attribute and a split value for that attribute.
- Once we have the two groups, we can then use our Gini score above to evaluate the cost of the split.
- Splitting a dataset involves iterating over each row, checking if the attribute value is below or above the split value and assigning it to the left or right group respectively.

Evaluating the splits:

- Given a dataset, we must check every value on each attribute as a candidate split, evaluate the cost of the split and find the best possible split we could make.
- Once the best split is found, we can use it as a node in our decision tree.
- This is an exhaustive and greedy algorithm.
- We will use a dictionary to represent a node in the decision tree as we can store data by name.
- When selecting the best split and using it as a new node for the tree we will store the index of the chosen attribute, the value of that attribute by which to split and the two groups of data split by the chosen split point.
- Each group of data is its own small dataset of just those rows assigned to the left or right group by the splitting process.
- You can imagine how we might split each group again, recursively as we build out our decision tree.

3. **Build a Tree:**

It involves creating a root node and adding nodes to the tree. This involves 3 parts.

a. Terminal Nodes.
b. Recursive Splitting.
c. Building a tree.

Terminal Nodes:

We need to decide when to stop growing a tree. We can do that using the depth and the number of rows that the node is responsible for in the training dataset.

- **Maximum Tree Depth**. This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes. Deeper trees are more complex and are more likely to overfit the training data.

- **Minimum Node Records**. This is the minimum number of training patterns that a given node is responsible for. Once at or below this minimum, we must stop splitting and adding new nodes. Nodes that account for too few training patterns are expected to be too specific and are likely to overfit the training data.

These two approaches will be user-specified arguments to our tree building procedure. There is one more condition. It is possible to choose a split in which all rows belong to one group. In this case, we will be unable to continue splitting and adding child nodes as we will have no records to split on one side or another.

Recursive Splitting:

- New nodes added to an existing node are called child nodes. A node may have zero children (a terminal node), one child (one side makes a prediction directly) or two child nodes. We will refer to the child nodes as left and right in the dictionary representation of a given node.
- Once a node is created, we can create child nodes recursively on each group of data from the split by calling the same function again.

Building a tree:

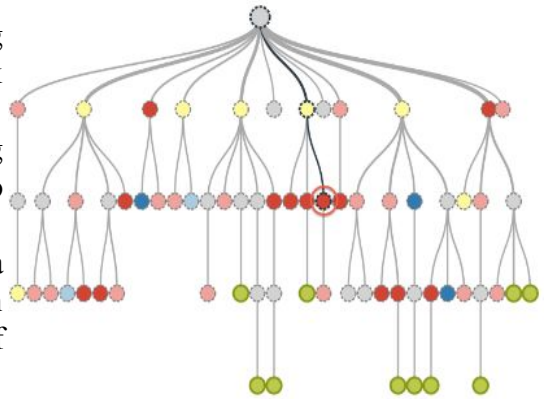
- Building the tree involves creating the root node and recursively split the data points, calculate the Gini Index to decide the best split, create new nodes and once a terminating condition is reached, exit the recursion, to build out the whole tree.

4. Make a Prediction:

- Making predictions with a decision tree involves navigating the tree with the specifically provided row of data.
- Again, we can implement this using a recursive function, where the same prediction routine is called again with the left or the right child nodes, depending on how the split affects the provided data.
- We must check if a child node is either a terminal value to be returned as the prediction, or if it is a dictionary node containing another level of the tree to be considered.

Working:

- Once the tree is constructed using the training data, we can test it using the test data and check the efficiency of the algorithm.
- Each data point is evaluated at every node starting from the root node and a decision is taken to go towards either the left node or the right node.
- The data point keeps traversing the tree until a leaf node is reached where the algorithm evaluates a different data point or terminates if there are no data points anymore.
- At the end of the algorithm the data point would



have been classified into one of the classes defined by the dataset.

Overfitting:

- Usually in decision tree algorithm, overfitting is an issue where the training set is continuously evaluated and split to increase the accuracy.
- This results in a very accurate tree but when the test data is run through the tree the accuracy decreases.
- To avoid overfitting there are two suggested methods
    - pre-pruning where the growth of the tree is stopped at a certain level
    - post-pruning where the tree is allowed to grow till the end and then resized.
- Usually the second method is preferred because it is not easy to estimate where to stop growth of the tree beforehand.

Post Pruning:

- Use a distinct dataset from the training set (called validation set), to evaluate the effect of post-pruning nodes from the tree.
- Build the tree by using the training set, then apply a statistical test to estimate whether pruning or expanding a particular node is likely to produce an improvement beyond the training set.
- Error estimation
- Significance testing (e.g., Chi-square test)
- Minimum Description Length principle : Use an explicit measure of the complexity for encoding the training set and the decision tree, stopping growth of the tree when this encoding size (size(tree) + size(misclassifications(tree)) is minimized.

Datasets:

1. kr-vs-kp.data - Chess dataset
2. HTRU_2.csv - Pulsar candidate information

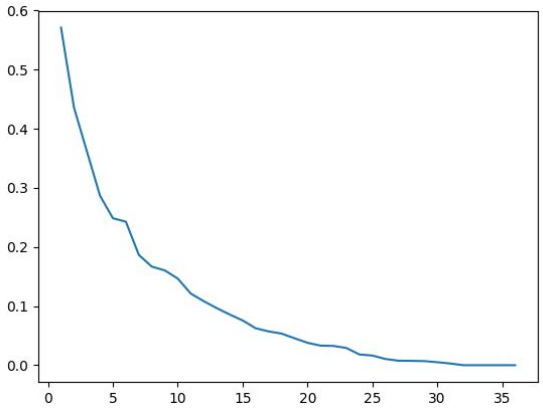
kr-vs-kp.data - Chess dataset:

- Contains the information of white's and black's moves.
- This dataset predicts if white wins in the next move or not depending on data.

- This dataset has 36 attributes and 2 classes. The 37th column has the classes into which data is classified into.
- There are about 3200 instances with 52% for white winning positions.

Without PCA:

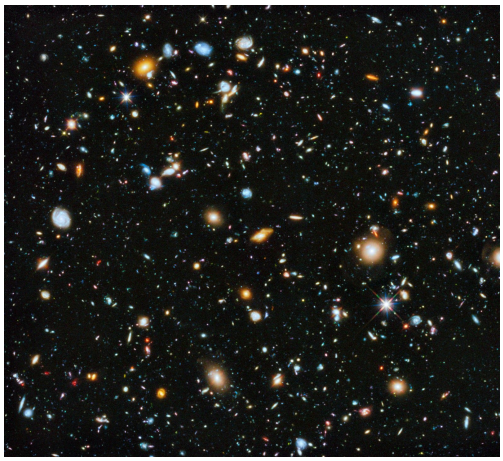
- This dataset ran in 1min 40sec with an accuracy of 91.3%.
- This running time can further be reduced by applying PCA but at the cost of accuracy.
- From the plot on the right, 20 features will have most of the data needed for classification.
- We ran the algorithm with 36 features, 20 features and 5 features.



With PCA:
The running time of the algorithm has reduced considerably but the accuracy is comprised.

- For 36 features after PCA - 0m57.260s - 90.2%
- For 5 features after PCA - 0m20.920s - 76%

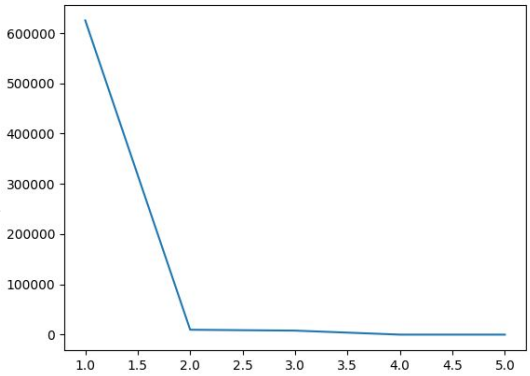
HTRU_2.csv - Pulsar candidate information:



- This dataset predicts if an object is a pulsar.
- The attributes contain the statistics obtained from the integrated pulse profile and delta modulated - signal to noise ratio.
- The data contains mean, standard deviation, excess kurtosis(measure of the sharpness of peak of the frequency distribution curve) and skewness(it is a measure of asymmetry of a probability distribution about its mean for a real-valued random variable).
- The above data is recorded for both the integrated profile and for DM-SNR.

Without PCA:

- This algorithm ran in 17sec with an accuracy of 92.2%
- There was no feature reduction.
- This running time can further be reduced by applying PCA but at the cost of accuracy.
- From the plot to the right, 2 components carry most of the data.



With PCA:

- The running time of the algorithm has reduced considerably but the accuracy is comprised.
- The running time for 5 features has dropped from 17sec to 16sec. But the accuracy dropped from 92.2% to 88.6%.
- The running time for 2 features has dropped from 17sec to 5sec. But the accuracy dropped from 92.2% to 81.2%

Which Algorithm to choose:

- The Decision Tree algorithm evaluates the data points at each node by calculating the Gini Index.
- If there are more number of features and the data is relatively inconsistent, Decision Tree takes a lot of time to run.
- Also, there could be a problem of overfitting with decision trees.

- They return better results when the data is more consistent with fewer number of attributes. The accuracy of the result is higher compared to other implementations.

Problems Faced:

- The dataset that we choose to work on had some missing attributes. To deal with it we assigned all the missing values with a constant value.
- Non numerical data set -For the computations to be done the data was to be in the form of float. But whereas in our dataset the the data was in the form of characters and strings. So we had to write a python program to encode character and string variables to float type.

Conclusion:

- While applying PCA there is a tradeoff between accuracy and execution time. Accuracy might decrease because we might loose some data while reducing the dimensions.
- But this tradeoff can be overcome by properly choosing the PCA components that explain all the variances in the data.
- If there are features that are distinct and the variance among them is large, it is rather advisable to use KNN over Decision Tree. The trade off is accuracy over time.

References:

- http://machinelearningmastery.com
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://en.wikipedia.org/wiki/Decision_tree
- https://archive.ics.uci.edu/ml/datasets
- https://www.kaggle.com/datasets
- http://deeplearning.net/datasets/