**CS 481 Pattern Recognition**

**Project Title**

**Summer 2022**

**Team members:**

|  |  |
| --- | --- |
| sn | Name |
| 1 | Omnia ashraf |

**Submitted to:**

Prof. Dr. Mostafa Gadal-haqq

T.A. Mohamed Rasmy

List of Content

Table of Contents

[1.1 Overview 2](#_Toc111905880)

[1.2 Problem Statement (Definition) 3](#_Toc111905881)

[1.3 The Proposed solution 3](#_Toc111905882)

[1.4 System Tools 3](#_Toc111905883)

[a. Software Requirements 3](#_Toc111905889)

[b. Hardware Requirements 3](#_Toc111905890)

[1.5 Results and Discussions 3](#_Toc111905891)

[1.6 Conclusions 3](#_Toc111905892)

[1.7 Appendix 3](#_Toc111905893)

## Overview

Heart disease is the normal term used in the health industry. The meaning of the Heart disease is that the heart is not

working properly or normally. In the medical terminology the heart attack is a condition where the supply of the blood to the

organs of the body is blocked and then it will result into the blood clot. Now-a-days there are so many heart diseases like

Coronary Artery Disease, Congestive Heart Failure and Bad Heart Rhythms etc. There are so many number of people who are

suffering from the heart diseases. The heart diseases may or may not have the symptoms before it attack the people. So we need

to predict the heart diseases for the people it effect or not. Now-a-days so much number of people is died suddenly due to the

heart attack because the life style of the people is changed rapidly. In this research paper we use the Support Vector Machine

which is the Machine Learning algorithm. The support vector machine is a supervised learning method. In the research paper

the Support Vector Machine can predict the heart disease based on the given factors like sex, age, pulse rate etc. The machine

learning algorithm support vector machine used in this research paper will give the most accurate and reliable results when

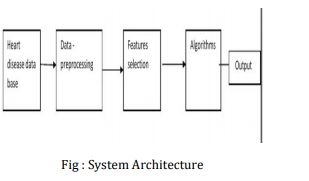
compare to the other algorithms.

## Problem Statement (Definition)

## The major challenge in heart disease is its detection. There are instruments available which can predict heart disease but either they are expensive or are not efficient to calculate chance of heart disease in human. Early detection of cardiac diseases can decrease the mortality rate and overall complications. However, it is not possible to monitor patients every day in all cases accurately and consultation of a patient for 24 hours by a doctor is not available since it requires more sapience, time and expertise. Since we have a good amount of data in today’s world, we can use various machine learning algorithms to analyze the data for hidden patterns. The hidden patterns can be used for health diagnosis in medicinal data.

**Proposed Solution**

Section depicts the overview of the proposed system and illustrates all of the components, techniques and tools are used for developing the entire system. To develop an intelligent and user friendly heart disease prediction system, an efficient software tool is needed in order to train huge datasets and compare multiple machine learning algorithms. After choosing the robust algorithm with best accuracy and performance measures, it will be implemented on the development of the smart phone-based application for detecting and predicting heart disease risk level. Hardware components like Arduino/Raspberry Pi, different biomedical sensors, display monitor, buzzer etc. are needed to build the continuous patient monitoring system. The below figure shows the process flow diagram or proposed work. First we collected the Cleveland Heart Disease Database from UCI website then preprocessed the dataset and select 16 important features..



For feature selection we used Recursive feature Elimination Algorithm using Chi2 methodand get 16 top features. After that applied ANN and Logistic algorithm individually and compute the accuracy. Finally, we used proposed Ensemble Voting method and compute best method for diagnosis of heart disease. Data Pre-processing: This file contains all the pre-processing functions needed to process all input documents andtexts. First we read the train, test and validation data files then performed some preprocessing like tokenizing, stemming etc. There are some exploratory data analysis is performed like response variable distribution and data quality checks like null or missing values etc. Feature: Extraction In this file we have performed feature extraction and selection methods from sci-kit learn python libraries. For feature selection, we have used methods like simple bag-of-words and n-grams and then term frequency like tf-tdf weighting. We have also used word2vec and POS tagging to extract the features, though POS tagging and word2vec has not been used at this point in the project. Classification: Here we have built all the classifiers for the breast cancer diseases detection. The extracted features are fed into different classifiers. We have used Naivebayes, Logistic Regression, Linear SVM, Stochastic gradient decent and Random forest classifiers from sklearn. Each of the extracted features was used in all of the classifiers. Once fitting the model, we compared the f1 score and checked the confusion matrix. After fitting all the classifiers, 2 best performing models were selected as candidatemodels for heart diseases classification. We have performed parameter tuning byimplementing GridSearchCV methods on these candidate models and chosen best performing parameters for these classifier. Finally selected model was used for heart disease detection with the probability oftruth. In Addition to this, we have also extracted the top 50 features from our term-frequencytfidf Vectorizer to see what words are most and important in each of the classes. We have also used Precision-Recall and learning curves to see how training and testset performs when we increase the amount of data in our classifiers. Prediction: Our finally selected and best performing classifier was algorithm which was then saved on disk with name final\_model.sav. Once you close this repository, this model will bec opied to user's machine and will be used by prediction.py file to classify the Heart diseases. It takes a news article as input from user then model is used for final classification output that is shown to user along with probability of truth.

# ALGORITHMS

## SUPPORT VECTOR MACHINE (SVM):

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In the 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

The followings are important concepts in SVM -

Support Vectors - Data Points that are closest to the hyperplane are called support vectors. Separating line will be defined with the help of these data points.

Hyperplane - As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.

Margin - It may be defined as the gap between two lines on the closest data points of different classes. It can be calculated as the perpendicular distance from the line to the

support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

### Types of SVM:

SVM can be of two types:

* **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
* **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

The objective of the support vector machine algorithm is to find a hyperplane in an N- dimensional space (N - the number of features) that distinctly classifies the data points.

### The advantages of support vector machines are:

* Effective in high dimensional spaces.
* Still effective in cases where the number of dimensions is greater than the number of samples.
* Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
* Versatile: different [kernel functions](https://scikit-learn.org/stable/modules/svm.html#svm-kernels) can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

### The disadvantages of support vector machines include:

* If the number of features is much greater than the number of samples, avoid over-fitting in choosing [Kernel functions](https://scikit-learn.org/stable/modules/svm.html#svm-kernels) and regularization term is crucial.

SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

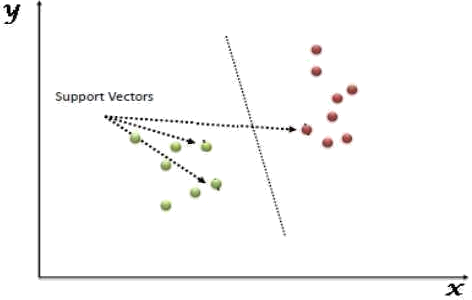


Figure: Support Vector Machine

## DECISION TREE ALGORITHM

Decision Tree is a Supervised learning technique that can be used for both classification and regression problems, but mostly it is preferred for solving classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision Tree, there are two nodes, which are the Decision Node and Leaf Node.

Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a Decision Tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A Decision Tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

The Decision Tree Algorithm belongs to the family of supervised machine learning algorithms. It can be used for both a classification problem as well as for a regression problem.

The goal of this algorithm is to create a model that predicts the value of a target variable, for which the decision tree uses the tree representation to solve the problem in which the leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision Tree:

* + - * Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
      * The logic behind the decision tree can be easily understood because it shows a tree-like structure.

In Decision Tree the major challenge is to identify the attribute for the root node in each level. This process is known as attribute selection. We have two popular attribute selection measures:

### Information Gain:

When we use a node in a Decision Tree to partition the training instances into smaller subsets, the entropy changes. Information gain is a measure of this change in entropy.

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples.

The higher the entropy the more the information content.

### Gini Index:

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower Gini index should be preferred. Sklearn supports “Gini” criteria for Gini Index and by default, it takes “gini” value.

The most notable types of Decision Tree algorithms are:-

### IDichotomiser 3 (ID3):

This algorithm uses Information Gain to decide which attribute is to be used to classify the current subset of the data. For each level of the tree, information gain is calculated for the remaining data recursively.

* 1. **C4.5:** This algorithm is the successor of the ID3 algorithm. This algorithm uses either Information gain or Gain ratio to decide upon the classifying attribute. It is a direct improvement from the ID3 algorithm as it can handle both continuous and missing attribute values.
  2. **Classification and Regression Tree (CART):** It is a dynamic learning algorithm which can produce a regression tree as well as a classification tree depending upon the dependent variable.

### Working:

In a Decision Tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

* Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
* Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
* Step-3: Divide the S into subsets that contains possible values for the best attributes.
* Step-4: Generate the Decision Tree node, which contains the best attribute.
* Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

## RANDOM FOREST ALGORITHM

Random Forest is a supervised learning algorithm. It is an extension of machine learning classifiers which include the bagging to improve the performance of Decision Tree. It combines tree predictors, and trees are dependent on a random vector which is independently sampled. The distribution of all trees are the same. Random Forests splits nodes using the best among of a predictor subset that are randomly chosen from the node itself, instead of splitting nodes based on the variables. The time complexity of the worst case of learning with Random Forests is O(M(dnlogn)) , where M is the number of growing trees, n is the number of instances, and d is the data dimension.

It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest consists of trees. It is said that the more trees it has, the more robust a forest is. Random Forests create Decision Trees on randomly selected data samples, get predictions from each tree and select the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random Forests have a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

### Assumptions:

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

* There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
* The predictions from each tree must have very low correlations.

### Algorithm Steps:

It works in four steps:

* Select random samples from a given dataset.
* Construct a Decision Tree for each sample and get a prediction result from each Decision Tree.
* Perform a vote for each predicted result.
* Select the prediction result with the most votes as the final prediction.

### Advantages:

* Random Forest is capable of performing both Classification and Regression tasks.
* It is capable of handling large datasets with high dimensionality.
* It enhances the accuracy of the model and prevents the overfitting issue.

### Disadvantages:

Although Random Forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

## System Tools



## Software Requirements

Operating System : Windows family

Technology : Python3.7

IDE : Kaggle jupyter notebook

## Hardware Requirements

Processer : Any Update Processer

Ram : Min 4GB

Hard Disk : Min 100GB

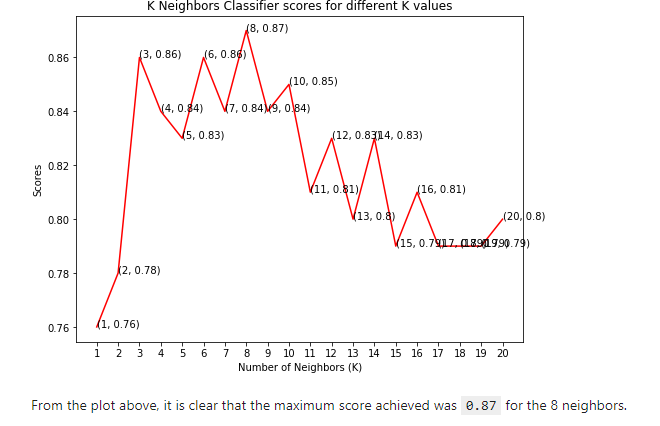
## Results and Discussions

* + - * The overall performance of the algorithms: both algorithms acted well
      * The strengths and weaknesses of the algorithms: they both handled the data properly
      * Any potential biases in the dataset or evaluation process: the dataset didn’t contain any imbalances in the distribution of classes that may have affected the performance of the algorithms

## 

## A picture containing diagram, line, plot, text Description automatically generated

## A picture containing screenshot, rectangle, colorfulness, text Description automatically generated



## Conclusions

In this project, we used Machine Learning to predict whether a person is suffering from a heart disease. After importing the data, I analysed it using plots. Then, I did generated dummy variables for categorical features and scaled other features. I then applied four Machine Learning algorithms, K Neighbors Classifier, Support Vector Classifier, Decision Tree Classifier and Random Forest Classifier. I varied parameters across each model to improve their scores. In the end, K Neighbors Classifier achieved the highest score of 87% with 8 nearest neighbors.

## Appendix

### Import libraries

Let's first import all the necessary libraries. I'll use numpy and pandas to start with. For visualization, I will use pyplot subpackage of matplotlib, use rcParams to add styling to the plots and rainbow for colors. For implementing Machine Learning models and processing of data, I will use the sklearn library.

In [1]:

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**from** matplotlib **import** rcParams

**from** matplotlib.cm **import** rainbow

**%matplotlib** inline

**import** warnings

warnings**.**filterwarnings('ignore')

For processing the data, I'll import a few libraries. To split the available dataset for testing and training, I'll use the train\_test\_split method. To scale the features, I am using StandardScaler.

In [2]:

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

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

Next, I'll import all the Machine Learning algorithms I will be using.

1. K Neighbors Classifier
2. Support Vector Classifier
3. Decision Tree Classifier
4. Random Forest Classifier

In [3]:

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.svm **import** SVC

**from** sklearn.tree **import** DecisionTreeClassifier

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

### Import dataset

Now that we have all the libraries we will need, I can import the dataset and take a look at it. The dataset is stored in the file dataset.csv. I'll use the pandas read\_csv method to read the dataset.

In [4]:

dataset **=** pd**.**read\_csv('dataset.csv')

The dataset is now loaded into the variable dataset. I'll just take a glimpse of the data using the desribe() and info() methods before I actually start processing and visualizing it.

In [5]:

dataset**.**info()

RangeIndex: 303 entries, 0 to 302

Data columns (total 14 columns):

age 303 non-null int64

sex 303 non-null int64

cp 303 non-null int64

trestbps 303 non-null int64

chol 303 non-null int64

fbs 303 non-null int64

restecg 303 non-null int64

thalach 303 non-null int64

exang 303 non-null int64

oldpeak 303 non-null float64

slope 303 non-null int64

ca 303 non-null int64

thal 303 non-null int64

target 303 non-null int64

dtypes: float64(1), int64(13)

memory usage: 33.2 KB

Looks like the dataset has a total of 303 rows and there are no missing values. There are a total of 13 features along with one target value which we wish to find.

In [6]:

dataset**.**describe()

Out[6]:

|  | **age** | **sex** | **cp** | **trestbps** | **chol** | **fbs** | **restecg** | **thalach** | **exang** | **oldpeak** | **slope** | **ca** | **thal** | **target** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| **mean** | 54.366337 | 0.683168 | 0.966997 | 131.623762 | 246.264026 | 0.148515 | 0.528053 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 | 2.313531 | 0.544554 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 | 0.612277 | 0.498835 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.000000 | 0.000000 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 | 2.000000 | 1.000000 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 | 3.000000 | 1.000000 |

The scale of each feature column is different and quite varied as well. While the maximum for age reaches 77, the maximum of chol (serum cholestoral) is 564.

### Understanding the data

Now, we can use visualizations to better understand our data and then look at any processing we might want to do.

In [7]:

rcParams['figure.figsize'] **=** 20, 14

plt**.**matshow(dataset**.**corr())

plt**.**yticks(np**.**arange(dataset**.**shape[1]), dataset**.**columns)

plt**.**xticks(np**.**arange(dataset**.**shape[1]), dataset**.**columns)

plt**.**colorbar()

Out[7]:

Taking a look at the correlation matrix above, it's easy to see that a few features have negative correlation with the target value while some have positive. Next, I'll take a look at the histograms for each variable.

In [8]:

dataset**.**hist()

Out[8]:

array([[,

,

,

],

[,

,

,

],

[,

,

,

],

[,

,

,

]],

dtype=object)

Taking a look at the histograms above, I can see that each feature has a different range of distribution. Thus, using scaling before our predictions should be of great use. Also, the categorical features do stand out.

It's always a good practice to work with a dataset where the target classes are of approximately equal size. Thus, let's check for the same.

In [28]:

rcParams['figure.figsize'] **=** 8,6

plt**.**bar(dataset['target']**.**unique(), dataset['target']**.**value\_counts(), color **=** ['red', 'green'])

plt**.**xticks([0, 1])

plt**.**xlabel('Target Classes')

plt**.**ylabel('Count')

plt**.**title('Count of each Target Class')

Out[28]:

Text(0.5, 1.0, 'Count of each Target Class')

The two classes are not exactly 50% each but the ratio is good enough to continue without dropping/increasing our data.

### Data Processing

After exploring the dataset, I observed that I need to convert some categorical variables into dummy variables and scale all the values before training the Machine Learning models. First, I'll use the get\_dummies method to create dummy columns for categorical variables.

In [10]:

dataset **=** pd**.**get\_dummies(dataset, columns **=** ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])

Now, I will use the StandardScaler from sklearn to scale my dataset.

In [29]:

standardScaler **=** StandardScaler()

columns\_to\_scale **=** ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

dataset[columns\_to\_scale] **=** standardScaler**.**fit\_transform(dataset[columns\_to\_scale])

The data is not ready for our Machine Learning application.

### Machine Learning

I'll now import train\_test\_split to split our dataset into training and testing datasets. Then, I'll import all Machine Learning models I'll be using to train and test the data.

In [12]:

y **=** dataset['target']

X **=** dataset**.**drop(['target'], axis **=** 1)

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.33, random\_state **=** 0)

#### K Neighbors Classifier

The classification score varies based on different values of neighbors that we choose. Thus, I'll plot a score graph for different values of K (neighbors) and check when do I achieve the best score.

In [13]:

knn\_scores **=** []

**for** k **in** range(1,21):

knn\_classifier **=** KNeighborsClassifier(n\_neighbors **=** k)

knn\_classifier**.**fit(X\_train, y\_train)

knn\_scores**.**append(knn\_classifier**.**score(X\_test, y\_test))

I have the scores for different neighbor values in the array knn\_scores. I'll now plot it and see for which value of K did I get the best scores.

In [14]:

plt**.**plot([k **for** k **in** range(1, 21)], knn\_scores, color **=** 'red')

**for** i **in** range(1,21):

plt**.**text(i, knn\_scores[i**-**1], (i, knn\_scores[i**-**1]))

plt**.**xticks([i **for** i **in** range(1, 21)])

plt**.**xlabel('Number of Neighbors (K)')

plt**.**ylabel('Scores')

plt**.**title('K Neighbors Classifier scores for different K values')

Out[14]:

Text(0.5, 1.0, 'K Neighbors Classifier scores for different K values')

From the plot above, it is clear that the maximum score achieved was 0.87 for the 8 neighbors.

In [15]:

print("The score for K Neighbors Classifier is {}% with {} nieghbors."**.**format(knn\_scores[7]**\***100, 8))

The score for K Neighbors Classifier is 87.0% with 8 nieghbors.

#### Support Vector Classifier

There are several kernels for Support Vector Classifier. I'll test some of them and check which has the best score.

In [16]:

svc\_scores **=** []

kernels **=** ['linear', 'poly', 'rbf', 'sigmoid']

**for** i **in** range(len(kernels)):

svc\_classifier **=** SVC(kernel **=** kernels[i])

svc\_classifier**.**fit(X\_train, y\_train)

svc\_scores**.**append(svc\_classifier**.**score(X\_test, y\_test))

I'll now plot a bar plot of scores for each kernel and see which performed the best.

In [17]:

colors **=** rainbow(np**.**linspace(0, 1, len(kernels)))

plt**.**bar(kernels, svc\_scores, color **=** colors)

**for** i **in** range(len(kernels)):

plt**.**text(i, svc\_scores[i], svc\_scores[i])

plt**.**xlabel('Kernels')

plt**.**ylabel('Scores')

plt**.**title('Support Vector Classifier scores for different kernels')

Out[17]:

Text(0.5, 1.0, 'Support Vector Classifier scores for different kernels')

The linear kernel performed the best, being slightly better than rbf kernel.

In [18]:

print("The score for Support Vector Classifier is {}% with {} kernel."**.**format(svc\_scores[0]**\***100, 'linear'))

The score for Support Vector Classifier is 83.0% with linear kernel.

#### Decision Tree Classifier

Here, I'll use the Decision Tree Classifier to model the problem at hand. I'll vary between a set of max\_features and see which returns the best accuracy.

In [19]:

dt\_scores **=** []

**for** i **in** range(1, len(X**.**columns) **+** 1):

dt\_classifier **=** DecisionTreeClassifier(max\_features **=** i, random\_state **=** 0)

dt\_classifier**.**fit(X\_train, y\_train)

dt\_scores**.**append(dt\_classifier**.**score(X\_test, y\_test))

I selected the maximum number of features from 1 to 30 for split. Now, let's see the scores for each of those cases.

In [20]:

plt**.**plot([i **for** i **in** range(1, len(X**.**columns) **+** 1)], dt\_scores, color **=** 'green')

**for** i **in** range(1, len(X**.**columns) **+** 1):

plt**.**text(i, dt\_scores[i**-**1], (i, dt\_scores[i**-**1]))

plt**.**xticks([i **for** i **in** range(1, len(X**.**columns) **+** 1)])

plt**.**xlabel('Max features')

plt**.**ylabel('Scores')

plt**.**title('Decision Tree Classifier scores for different number of maximum features')

Out[20]:

Text(0.5, 1.0, 'Decision Tree Classifier scores for different number of maximum features')

The model achieved the best accuracy at three values of maximum features, 2, 4 and 18.

In [21]:

print("The score for Decision Tree Classifier is {}% with {} maximum features."**.**format(dt\_scores[17]**\***100, [2,4,18]))

The score for Decision Tree Classifier is 79.0% with [2, 4, 18] maximum features.

#### Random Forest Classifier

Now, I'll use the ensemble method, Random Forest Classifier, to create the model and vary the number of estimators to see their effect.

In [22]:

rf\_scores **=** []

estimators **=** [10, 100, 200, 500, 1000]

**for** i **in** estimators:

rf\_classifier **=** RandomForestClassifier(n\_estimators **=** i, random\_state **=** 0)

rf\_classifier**.**fit(X\_train, y\_train)

rf\_scores**.**append(rf\_classifier**.**score(X\_test, y\_test))

The model is trained and the scores are recorded. Let's plot a bar plot to compare the scores.

In [23]:

colors **=** rainbow(np**.**linspace(0, 1, len(estimators)))

plt**.**bar([i **for** i **in** range(len(estimators))], rf\_scores, color **=** colors, width **=** 0.8)

**for** i **in** range(len(estimators)):

plt**.**text(i, rf\_scores[i], rf\_scores[i])

plt**.**xticks(ticks **=** [i **for** i **in** range(len(estimators))], labels **=** [str(estimator) **for** estimator **in** estimators])

plt**.**xlabel('Number of estimators')

plt**.**ylabel('Scores')

plt**.**title('Random Forest Classifier scores for different number of estimators')

Out[23]:

Text(0.5, 1.0, 'Random Forest Classifier scores for different number of estimators')

The maximum score is achieved when the total estimators are 100 or 500.

In [24]:

print("The score for Random Forest Classifier is {}% with {} estimators."**.**format(rf\_scores[1]**\***100, [100, 500]))

The score for Random Forest Classifier is 84.0% with [100, 500] estimators.

