TOPIC: MACHINE LEARINING

Rahul G. Chavan

Mini Project: Breast Cancer Classification Using Support Vector Machine on UCI ML Repository dataset.

Title: Breast Cancer Classification Using SVM on UCI ML Repository Dataset – **"Breast Cancer Wisconsin (Diagnostic) Data Set"**.

Problem Definition: Apply the Support Vector Machine for classification on a dataset obtained from UCI ML repository.

Prerequisite: Knowledge of Python or R, higher Mathematical Understanding and concepts of Machine Learning.

Software Requirements: Python3, Anaconda, Jupyter Notebook

Hardware Requirements: Functional computer system with sufficient power to run Python environment and Internet Connectivity.

Theory/Important Concepts Used Ahead:

- **Machine Learning:**
    - The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
    - The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly**.**
    - The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.

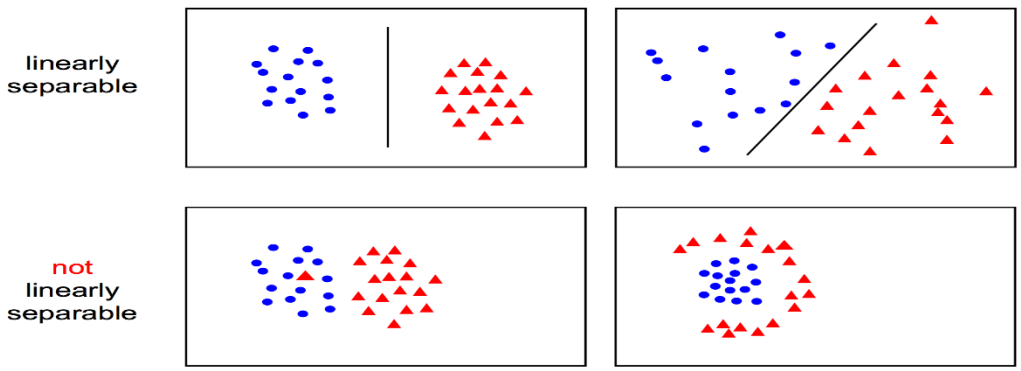
- **Classification:**
    - Classification is a process of categorizing a given set of data into classes, it can be performed on both structured or unstructured data.
    - The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories. The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables.
    - The main goal is to identify which class/category the new data will fall into. Some common terminologies used ahead about classification are:

      Classifier – It is an algorithm that is used to map the input data to a specific category.

      Classification Model – The model predicts or draws a conclusion to the input data given for training, it will predict the class or category for the data.
- **Modelling:**

- Depending on how long we've lived in a particular place and traveled to a location, we probably have a good understanding of commute times in our area.
- For example, we've traveled to work/school using some combination of the metro, buses, trains, ubers, taxis, carpools, walking, biking, etc.
- All humans naturally model the world around them.
- Over time, our observations about transportation have built up a mental dataset and a mental model that helps us predict what traffic will be like at various times and locations.
- We probably use this mental model to help plan our days, predict arrival times, and many other tasks.
- As data scientists we attempt to make our understanding of relationships between different quantities more precise through using data and mathematical/statistical structures. This process is called modeling.
- Models are simplifications of reality that help us to better understand that which we observe.
- In a data science setting, models generally consist of an independent variable (or output) of interest and one or more dependent variables (or inputs) believed to influence the independent variable.

- **Support Vector Machine (SVM):**
  - A Support Vector Machine (SVM) is a binary linear classification whose decision boundary is explicitly constructed to minimize generalization error.
  - It is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression and even outlier detection.
  - SVM is well suited for classification of complex but small or medium sized datasets.
  - It's important to start with the intuition for SVM with the special linearly separable classification case.
  - If classification of observations is "linearly separable", SVM fits the "decision boundary" that is defined by the largest margin between the closest points for each class. This is commonly called the "maximum margin hyperplane (MMH)".



- The advantages of support vector machines are:
    1. Effective in high dimensional spaces.
    2. Still effective in cases where number of dimensions is greater than the number of samples.
    3. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
    4. Versatile: different Kernel functions 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:
    1. If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
    2. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities).

Implementation:

- In this study, the task is to classify tumors into malignant (cancerous) or benign (non-cancerous) using features obtained from several cell images.
- Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
- **Attribute Information:**
    - ID number
    - Diagnosis (M = malignant, B = benign)
    - **Ten real-valued features are computed for each cell nucleus:**
        - Radius (mean of distances from center to points on the perimeter)
        - Texture (standard deviation of gray-scale values)
        - Perimeter
        - Area
        - Smoothness (local variation in radius lengths)
        - Compactness (perimeter² / area — 1.0)
        - Concavity (severity of concave portions of the contour)
        - Concave points (number of concave portions of the contour)
        - Symmetry
        - Fractal dimension (“coastline approximation” — 1)
- **Loading Python Libraries and Breast Cancer Dataset**

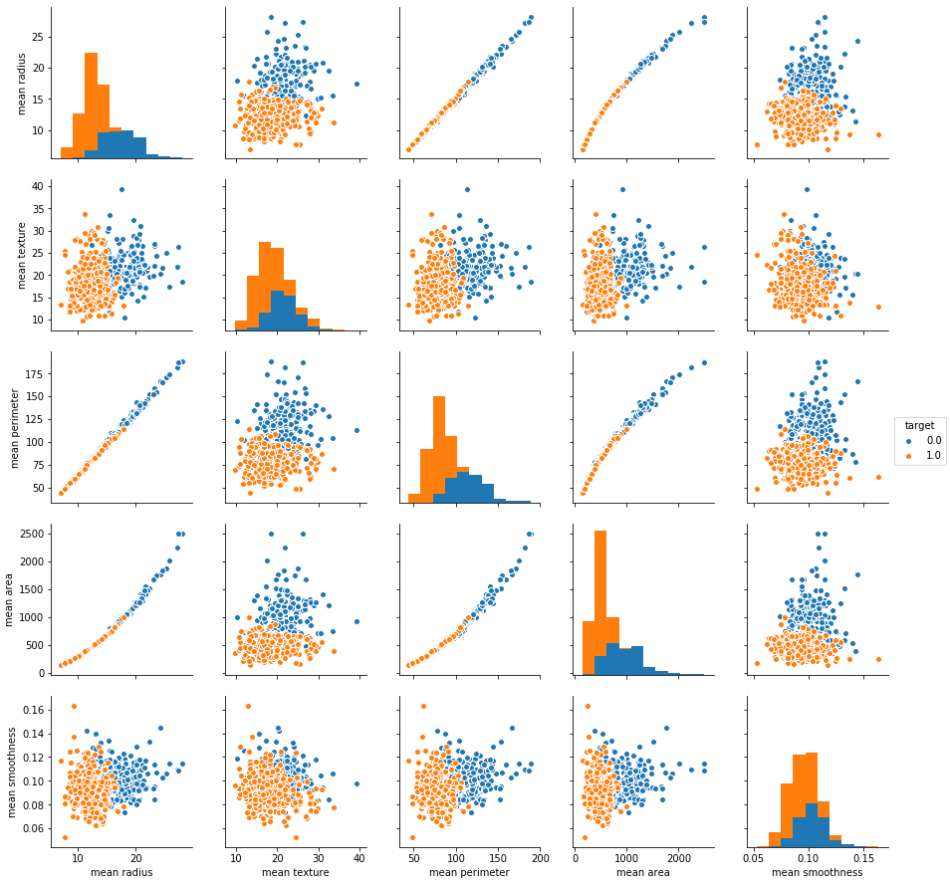
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

#Import Cancer data from the Sklearn library
# Dataset can also be found here (http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29)

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

- **Features (Columns) breakdown**

- mean radius = mean of distances from center to points on the perimeter
- mean texture = standard deviation of gray-scale values
- mean perimeter = mean size of the core tumor
- mean area =
- mean smoothness = mean of local variation in radius lengths
- mean compactness = mean of perimeter^2 / area - 1.0
- mean concavity = mean of severity of concave portions of the contour
- mean concave points = mean for number of concave portions of the contour
- mean symmetry =
- mean fractal dimension = mean for "coastline approximation" - 1
- radius error = standard error for the mean of distances from center to points on the perimeter
- texture error = standard error for standard deviation of gray-scale values
- perimeter error =
- area error =
- smoothness error = standard error for local variation in radius lengths
- compactness error = standard error for perimeter^2 / area - 1.0
- concavity error = standard error for severity of concave portions of the contour
- concave points error = standard error for number of concave portions of the contour
- symmetry error =
- fractal dimension error = standard error for "coastline approximation" - 1
- worst radius = "worst" or largest mean value for mean of distances from center to points on the perimeter
- worst texture = "worst" or largest mean value for standard deviation of gray-scale values
- worst perimeter =
- worst smoothness = "worst" or largest mean value for local variation in radius lengths
- worst compactness = "worst" or largest mean value for perimeter^2 / area - 1.0
- worst concavity = "worst" or largest mean value for severity of concave portions of the contour
- worst concave points = "worst" or largest mean value for number of concave portions of the contour
- worst fractal dimension = "worst" or largest mean value for "coastline approximation" - 1

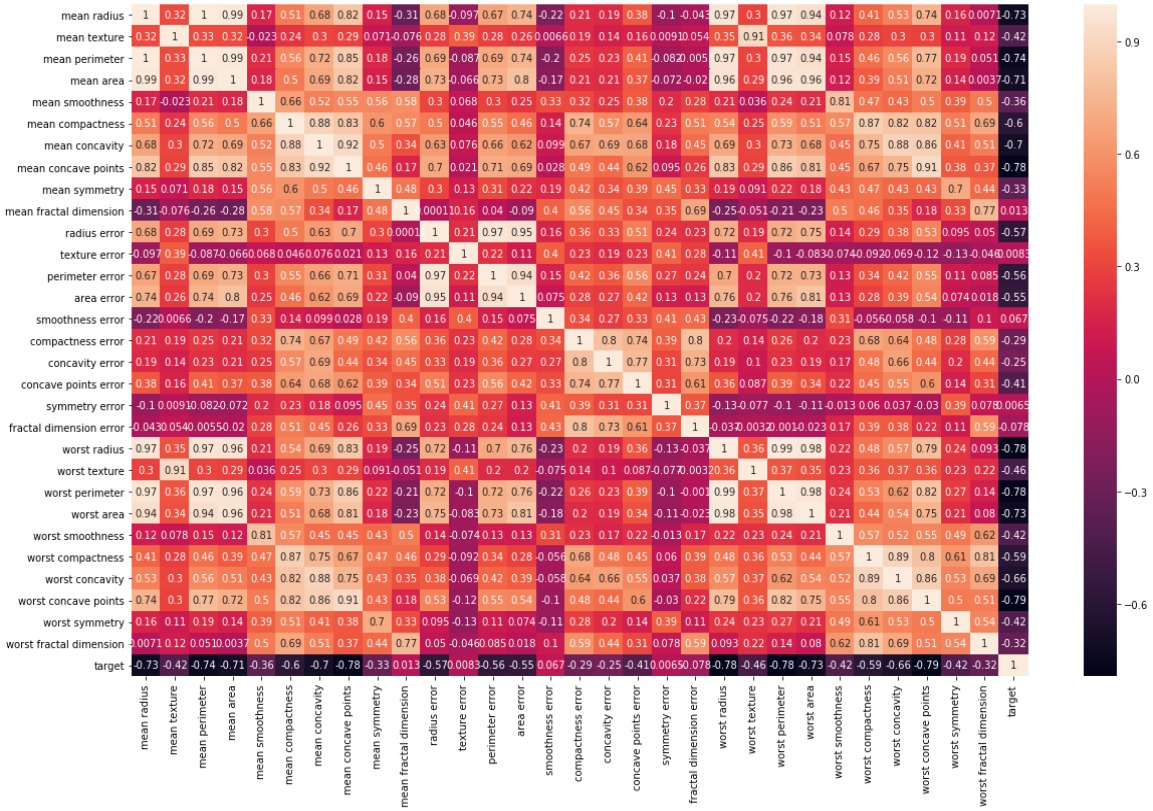
- **Visualize the relationship between our features**

```
In [7]: # Let's plot out just the first 5 variables (features)
        sns.pairplot(df_cancer, hue = 'target', vars = ['mean radius', 'mean texture', 'mean perimeter','mean area','mean smooth
```



- **Let's check the correlation between our features**

```
In [10]: plt.figure(figsize=(20,12))
         sns.heatmap(df_cancer.corr(), annot=True)
```



- There is a strong correlation between mean radius and mean perimeter, as well as mean area and mean perimeter

- **From our dataset, let's create the target and predictor matrix**
  - y" = Is the feature we are trying to predict (Output). In this case we are trying to predict if our "target" is cancerous (Malignant) or not (Benign). i.e. we are going to use the "target" feature here.
  - X" = The predictors which are the remaining columns (mean radius, mean texture, mean perimeter, mean area, mean smoothness, etc.)

```
In [11]: X = df_cancer.drop(['target'], axis = 1) # We drop our "target" feature and use all the remaining features in our dataf
         X.head()
```

Out[11]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothnes
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.137

5 rows × 30 columns

- **Create the training and testing data**

  Now that we've assigned values to our "X" and "y", the next step is to import the python library that will help us split our dataset into training and testing data.

  - Training data = the subset of our data used to train our model.
  - Testing data = the subset of our data that the model hasn't seen before (We will be using this dataset to test the performance of our model).
  - Let's split our data using 80% for training and the remaining 20% for testing.

```
In [14]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 20)
```

Let now check the size our training and testing data.

```
In [15]: print ('The size of our training "X" (input features) is', X_train.shape)
         print ('\n')
         print ('The size of our testing "X" (input features) is', X_test.shape)
         print ('\n')
         print ('The size of our training "y" (output feature) is', y_train.shape)
         print ('\n')
         print ('The size of our testing "y" (output features) is', y_test.shape)
```

```
The size of our training "X" (input features) is (455, 30)


The size of our testing "X" (input features) is (114, 30)


The size of our training "y" (output feature) is (455,)


The size of our testing "y" (output features) is (114,)
```

- **Train SVM model with our "training" dataset.**

```
In [18]: svc_model.fit(X_train, y_train)
```

```
Out[18]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

- **Use trained model to make a prediction using our testing data and compare it with output(y_test). The comparison is done using confusion matrix.**

```
In [19]: y_predict = svc_model.predict(X_test)
```

```
In [20]: # Import metric libraries

         from sklearn.metrics import classification_report, confusion_matrix
```

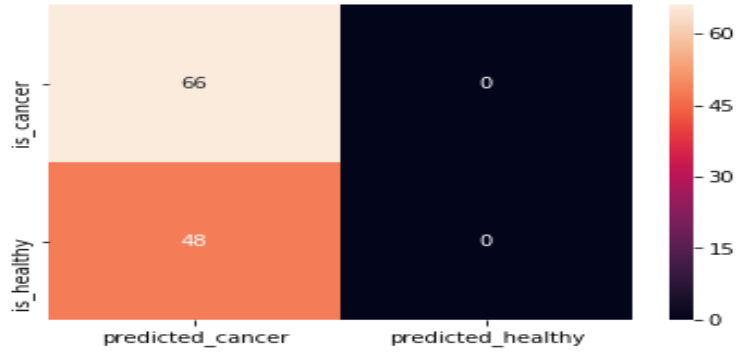
```
In [21]: cm = np.array(confusion_matrix(y_test, y_predict, labels=[1,0]))
         confusion = pd.DataFrame(cm, index=['is_cancer', 'is_healthy'],
                                  columns=['predicted_cancer','predicted_healthy'])
         confusion
```

Out[21]:

	predicted_cancer	predicted_healthy
is_cancer	66	0
is_healthy	48	0

- **Visualize our confusion matrix on a Heatmap**

```
In [22]: sns.heatmap(confusion, annot=True)
```




```
In [23]: print(classification_report(y_test, y_predict))
```

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	48
1.0	0.58	1.00	0.73	66
avg / total	0.34	0.58	0.42	114

As we can see, our model did not do a good job in its predictions. It predicted that 48 healthy patients have cancer. We only achieved 34% accuracy! So, we use normalization to improve the accuracy. Data normalization is a feature scaling process that brings all values into range [0,1].

  - **X' = (X-X_min) / (X_max — X_min)**

In [27]:
```
X_train_scaled = (X_train - X_train_min)/(X_train_range)
X_train_scaled.head()
```

Out[27]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	wo a
412	0.114345	0.391003	0.110290	0.053150	0.293907	0.126219	0.087512	0.025487	0.108081	0.401860	...	0.072394	0.418354	0.080681	0.028
461	0.967343	0.549827	0.988943	1.000000	0.605735	0.550334	0.851687	0.839463	0.505556	0.145814	...	1.000000	0.509582	1.000000	1.000
532	0.317052	0.205882	0.303849	0.183245	0.435973	0.163088	0.041050	0.093439	0.288384	0.269535	...	0.281750	0.208097	0.254943	0.144
495	0.373373	0.340138	0.361620	0.227953	0.469643	0.196522	0.159888	0.246074	0.215657	0.174884	...	0.287442	0.431579	0.266398	0.147
13	0.419755	0.469550	0.414000	0.271135	0.340828	0.247899	0.232849	0.266600	0.397475	0.079535	...	0.316969	0.409447	0.306738	0.169

5 rows × 30 columns

In [28]:
```
X_test_min = X_test.min()
X_test_range = (X_test - X_test_min).max()
X_test_scaled = (X_test - X_test_min)/X_test_range
```

- Retrain your SVM model with our scaled (Normalized) datasets.

In [29]:
```
svc_model = SVC()
svc_model.fit(X_train_scaled, y_train)
```

Out[29]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

- **Prediction with Scaled dataset**

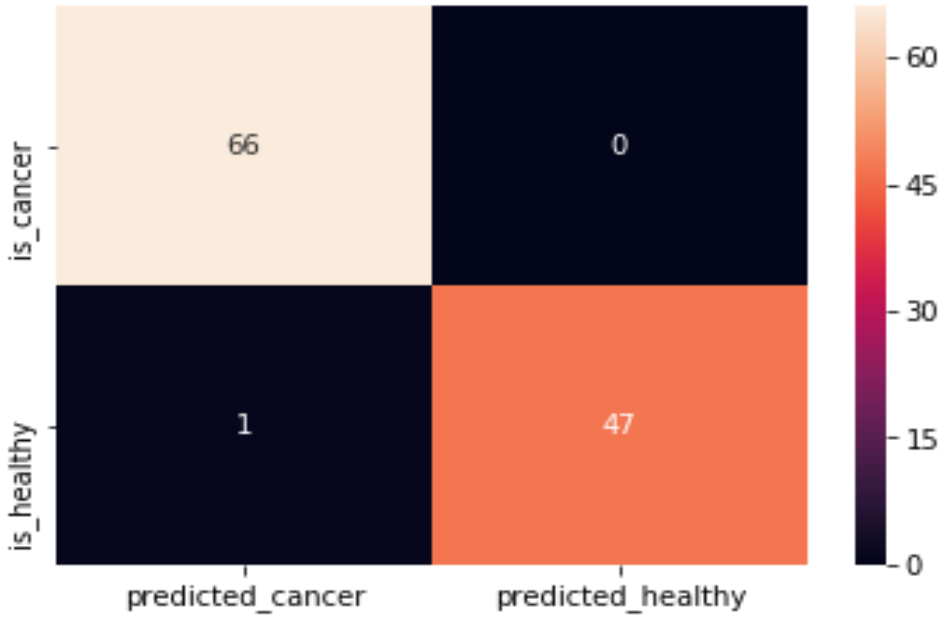
In [30]:
```
y_predict = svc_model.predict(X_test_scaled)
cm = confusion_matrix(y_test, y_predict)
```

- **Confusion Matrix on Scaled dataset**

In [31]:
```
cm = np.array(confusion_matrix(y_test, y_predict, labels=[1,0]))
confusion = pd.DataFrame(cm, index=['is_cancer', 'is_healthy'],
                         columns=['predicted_cancer','predicted_healthy'])
confusion
```

Out[31]:

	predicted_cancer	predicted_healthy
is_cancer	66	0
is_healthy	1	47




```
In [33]: print(classification_report(y_test,y_predict))
```

	precision	recall	f1-score	support
0.0	1.00	0.98	0.99	48
1.0	0.99	1.00	0.99	66
avg / total	0.99	0.99	0.99	114

Our prediction is now a lot better with only 1 false prediction (Predicted cancer instead of healthy).The accuracy of the model is 98%

Conclusion: We've thus developed a classifier model using Support Vector Machine for classification of Cancer. The model can predict a lot better than before after normalization of the data with an increased accuracy of 98% .

References:

[1]: "Breast cancer classification by using support vector machines with reduced dimension", https://ieeexplore.ieee.org/document/6044334

[2]: "Gunn, Support Vector Machines for Classification and Regression",https://www.isis.ecs.soton.ac.uk/resources/svminfo/