Chapter 3 : part 3

**Support Vector Machine**

**3.3.1 Decision Boundary**

While *training* a *classifier* on a *dataset*, using a specific ***classification algorithm***, it is required to define a Set Of Hyper-Planes, called Decision Boundary, that separates the data points into specific classes, where the algorithm switches from one class to another. On one side a decision boundary, a data-points is more likely to be called as class A — on the other side of the boundary, it’s more likely to be called as class B.

* For example in our Logistic regression example: we had, and implies,

|  |  |
| --- | --- |
| * So, our current prediction function returns a ***Probability Score*** between ***0*** and ***1***. In order to map this to a ***Discrete Class (A/B)***, we select a Threshold Value or Tipping Point above which we will classify values into Class A and below which we classify values into Class B. This Threshold Value is frequently denoted by   : class=A  : class=B   * If our threshold was ***.5*** and our prediction function returned ***.7***, we would classify this observation belongs to Class A. If our prediction was ***.2*** we would classify the observation belongs to Class B. | J:\My_images\decision_boundary.png  In order to map predicted values to probabilities, we use the Sigmoid function. |

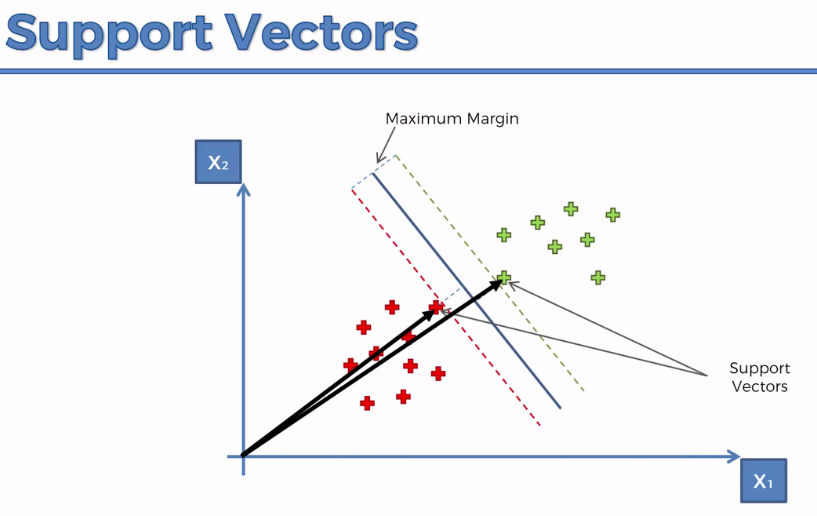
* Decision Boundary: Hence, line with ***0.5*** i.e. the horizontal line is called the Decision Boundary.

**3.3.2 Support Vector Machine**

|  |  |
| --- | --- |
| For simplicity's sake consider two columns features and . And observations are already classified.   * The problem is how we divide them? There are some following approaches that we can draw a line (Decision Boundary) that's going to separate them: we can use *vertical* or *horizontal* line or *diagonal* *lines* as follows |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| * So actually there is so many choice for Decision Boundary. But, *Decision* *Boundary* plays *Main role in classification*, because that's a *separation*. When we start *adding new points*, to *classify* those *new* *points* we need to use the *Decision* *Boundary*. So our goal is to find *"Best Decision Boundary "*. * So let's find out how the SVM actually searches for this line (Decision Boundary). | | Actually there are so many options for a diagonal line: | |

* Decision Boundary using SVM: Actually this Decision Boundary line is searched through the maximum margin. Basically it's *the line that separates these two class's of points* and at the same time it has the maximum margin.
* Maximum margins: Which means the distance from the drawn line is equidistant from the points in both sides. So the sum of these two ***distances (total margin) has to be maximized*** in order for this line to be the result of the SVM (*"Best Decision Boundary "*).

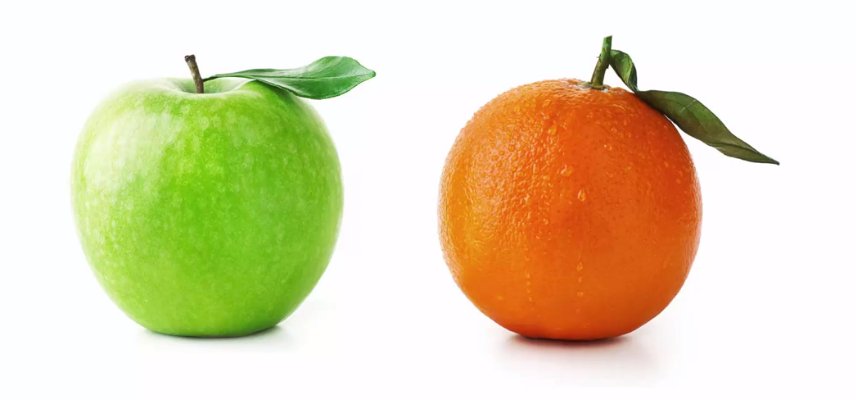


* Support vectors: The closest points on which the maximum margin depends (supported) are called the support vectors, because they *literally* supports the *decision boundary*. In above figure, the two points touching the maximum margin, are actually called the support vectors.
* These two points are supporting this whole algorithm. So even if you *get rid of all the rest of the points* the algorithm will be exactly the same.
* So these other points they don't contribute to the result of the algorithm only these two points are contributing and therefore they called the supporting vectors (you can call them supporting points but in reality they are vectors in multi-dimensions).
* Hence these two specific vectors are the ones kind of supporting this decision boundary that's why this whole algorithm is called the support vector machines.
* Maximum Margin Hyper-planes: We've got the line in the middle which is called the ***maximum*** ***margin*** ***hyperplane*** or the ***maximum*** ***margin*** ***classifier***. So in a two dimensional space this classifier is just the ***line***, but actually in a multidimensional space it's a ***hyperplane***.
* You can draw a different hyperplanes, but above is always be less because this is the one with the maximum margin.
* Positive Hyperplane And Negative Hyperplane: In the above figure, notice the *green* and the *red* *dotted* lines. The green one is called the *Positive Hyperplane* and the red one called the *Negative Hyperplane*.
* Anything to the right of the positive is classified as the green category (or the positive category) anything to the left to classify as a (negative category or) the red category in our case.

So that's how the supervision machine algorithm works of course there's some complicated mathematics behind it.

**3.3.3 What's so special about SVM**

Classify a fruit into either an apple an orange: So imagine you're trying to teach a machine how to distinguish between apples and oranges. Now in our case here you can see let's say on the right we have oranges on the left we have apples.



*Apple-ly-apple & Orangey—orange*

|  |  |
| --- | --- |
|  |  |

* Most of ML algorithms would do is they would look at the *Most Apple/ Most Orange* to match the given "fruit" (i.e. is the given fruit is most ***"Apple-ly Apple: Apple that is more like Apple"*** or ***"Most Orenge-y Orange: Orange that is more like Orange "***. Maximize the fruit character).
* But SVM does the reverse; it looks at the *marginal* *values* (i.e. ***Orenge-y Apple:*** "Apple that is more like Orange" or ***Apple-ly Orange:*** "Orange that is more like Apple"). Those are the *support vectors*. You can see that they're actually very close to the boundary so their *"Orange/apple fruit characteristics are very close to each other"*.

|  |  |
| --- | --- |
|  |  |

* Therefore the support vector machine in that sense you can think of it is like a more extreme type of algorithm a very rebellious type of algorithm a very risky type of algorithm because it looks at a very extreme case which is very close to the boundary and it uses that to construct its analysis.

And that in itself makes the *Support Vector Machine (SVM) Algorithms* very special very different to most of the other Machine Learning Algorithms.

**3.3.4 Python implementation of the SVM algorithm**

Practiced Version

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dataSet**.**iloc[:, [2,3]]**.**values

y = dataSet**.**iloc[:, 4]**.**values

        #*Feature-Scaling after Data Split*

#*Data Split*

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

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)

#*Feature-Scaling*

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

st\_x= **StandardScaler**()

X\_train= st\_x**.fit\_transform**(X\_train)

X\_test= st\_x**.transform**(X\_test)

#*Fit train set to Model classifier*

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

clsFier = **SVC**(kernel="linear", random\_state=0)

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = clsFier**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

#*Visualising the Training set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('SVM (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*Visualising the Test set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('SVM (Test set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*python prctc\_SVM.py*

* Data Preprocessing is Similar to previous KNN Example.
* Classifier: We use the ***kernel = "Linear"***. It is the part of Kernel-SVM, default is "***rbf***" (radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms) whish is mostly "Gaussian". " ***kernel = "Linear"***" means we do-not apply any Kernel-trick.

#*Fit train set to Model classifier*

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

clsFier = **SVC**(kernel="linear", random\_state=0)

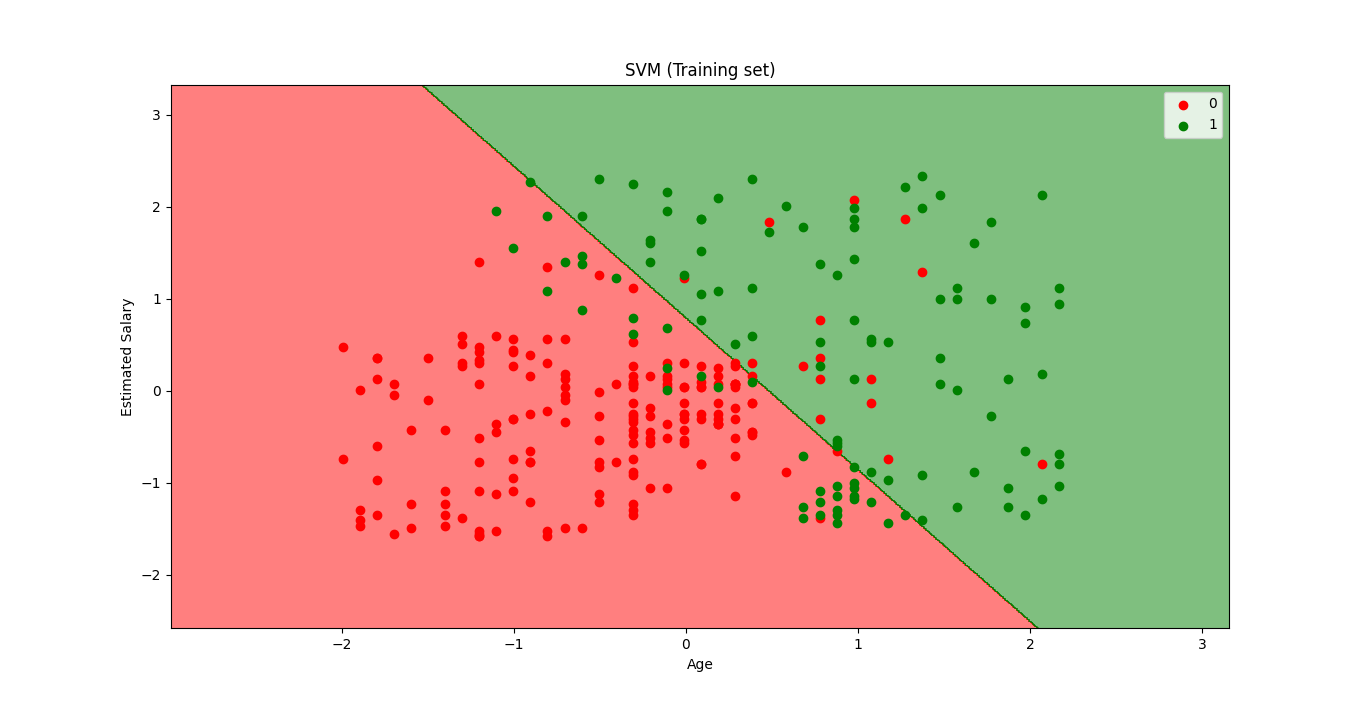
clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

* Prediction and confusion matrix: From following figures, we can see that the result is almost same as the "Previous Logistic Regression". Hence model performance is not so much improved. We use kernel-SVM in next section then the performance will increase.

|  |  |
| --- | --- |
| Predicted y-values | Confusion matrix |
|  |  |

* Visualizing the data: Same as KNN-algorithm section

Training Set



Test Set

