Introduction to SVM

Charles Moseley, Aditya Joshi, Joseph Zhang

Support Vector Machines

Support Vector Machines are supervised learning models, highly used for data analysis and recognizing patterns.

It is help for classification and regression.

We will mainly discuss the classification today.

SVM Algorithm

Given a training set, which is marked in two categories respectively.

SVM training algorithm builds a model that assigns new data into one or the other categories, to make it a non-probabilistic binary linear classifier.

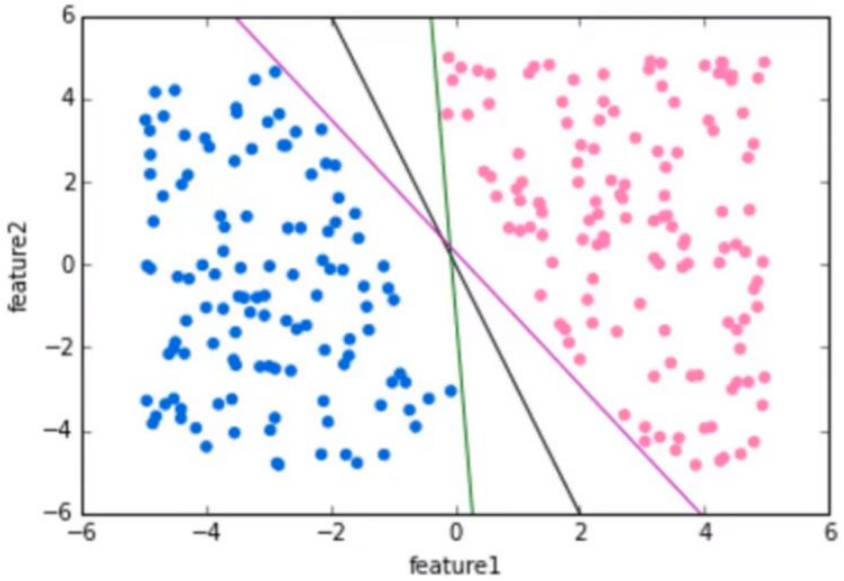
SVM model is training a clear ‘gap’ to separate two categories, as wide as possible.

Then, the original data in the two categories will be predicted to a category based on which side of the ‘gap’ they fall on.

SVM Training

Eg. Two features lie on two axes, two data categories mapped

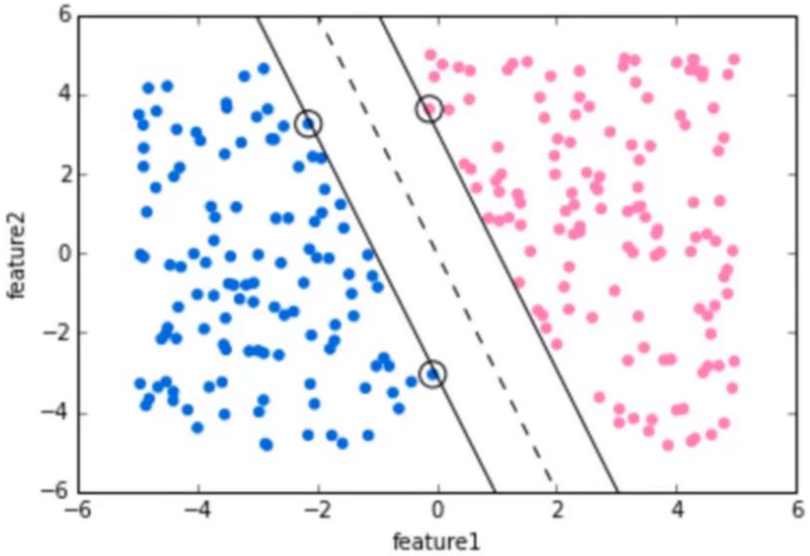
How to separate two categories of data perfectly?



SVM Training

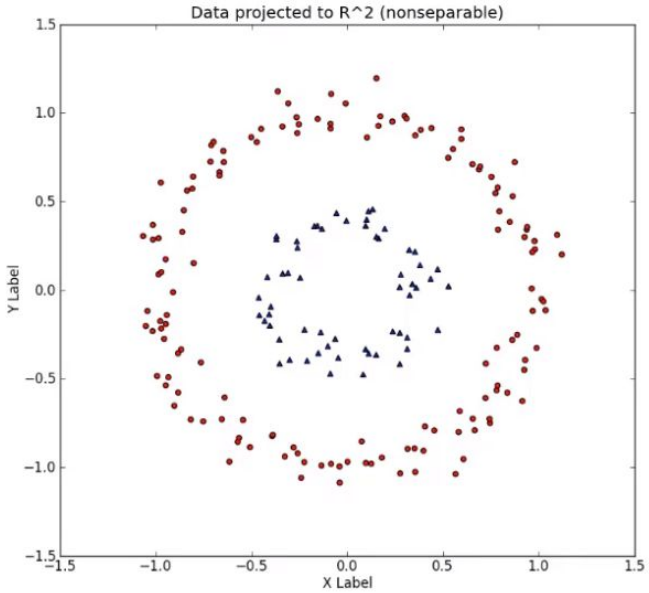
The hyperplane that maximize the margin between two categories of data

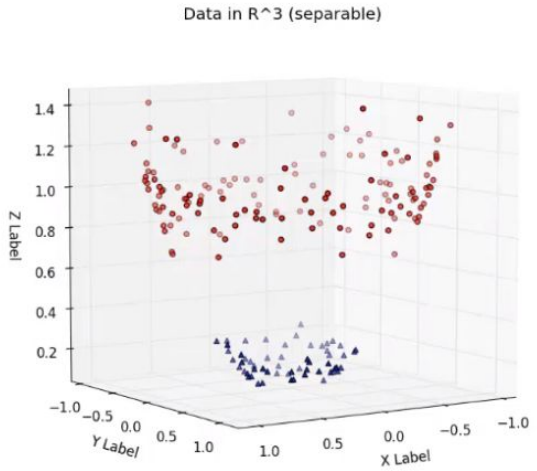
The vector data that margin lines touch are Support Vectors (circled)



SVM Training

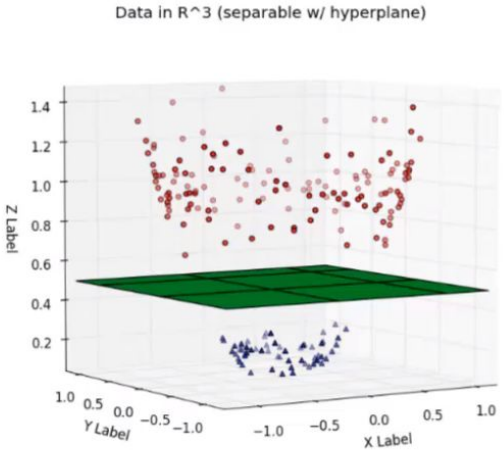
It also works on non-linearly separable dataset

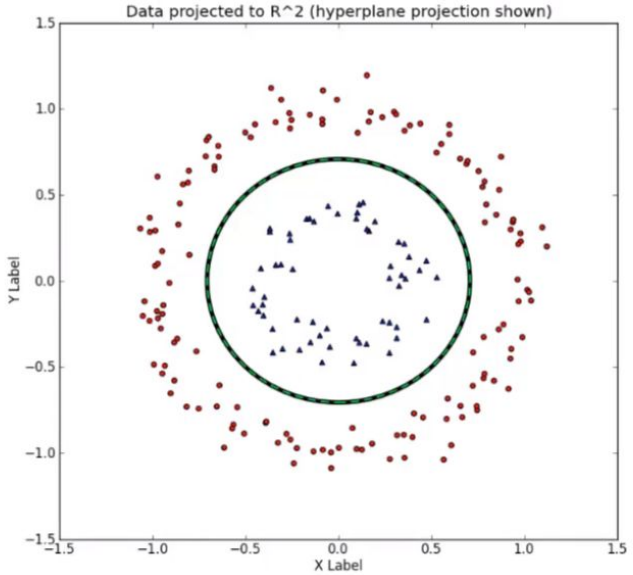




SVM Training

Use kernel trick, interpolation Z axis for non-linear data hyperplane to separate data





SVM Training

Let us see how to use it

We now use the built-in dataset Breast Cancer to train our model

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

```
# Use the built-in dataset Breast Cancer for sklearn
```

In [3]:
```
from sklearn.datasets import load_breast_cancer
```

SVM Training

Let's check our dataset:

569 instances, 30 numeric attributes, to predict the tumor is malignant or benign; # we use cancer 'data' and 'target' to do the analysis, to predict its malignant or benign.

```
In [11]: cr = load_breast_cancer()

In [12]: cr.keys()

Out[12]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
         'filename', 'data_module'])

In [15]: print(cr['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569
```

```
print(cr['data'])
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
```

```
print(cr['target'])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
```

```
print(cr['target_names'])
```

```
['malignant' 'benign']
```

Data Preparation

Sort out the attributes of cancer ‘data’ with ‘feature_names’ and cancer ‘target’ into new dataset, then split the dataset into train and test

```
crd = pd.DataFrame(cr['data'], columns=cr['feature_names'])
```

```
crd.head(2)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
0	17.99	10.38	122.8	1001.0	0.11840	0				
1	20.57	17.77	132.9	1326.0	0.08474	0				

2 rows × 30 columns

```
from sklearn.model_selection import train_test_split
```

```
x = crd
y = cr['target']


X_train, X_test, y_train, y_test = train_test_split(
x, y, test_size=0.3)                    # 30% proportion for test data split
```

Training the SVM model, using GridSearchCV for best parameter

C (Regularisation): C is the penalty parameter, it tells SVM optimisation how much error is bearable, C is high it will classify all the data points correctly, also there is a chance to overfit;

Gamma: It defines how far influences the calculation of plausible line of separation, low gamma means far away points also be considered to get the decision boundary.

Verbose: control the verbosity, 3 is a normal setting. Higher verbosity gives more messages, and may burn your

```
from sklearn.svm import SVC    # capital SVC = support vector classification
from sklearn.model_selection import GridSearchCV
```

```
param = {'C':[0.1,1,10,100,1000], 'gamma':[1,0.1,0.01,0.001,0.0001]}
```

```
gd = GridSearchCV(SVC(), param, verbose=3)  # give a number for verbose to test
```

```
gd.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END .....................C=0.1, gamma=1;, score=0.625 total time=
0.0s
[CV 2/5] END .....................C=0.1, gamma=1;, score=0.625 total time=
0.0s
[CV 3/5] END .....................C=0.1, gamma=1;, score=0.625 total time=
0.0s
[CV 4/5] END .....................C=0.1, gamma=1;, score=0.620 total time=
```

```
gd.best_params_
```

```
{'C': 1, 'gamma': 0.0001}
```

```
gd.best_estimator_
```

```
SVC
SVC(C=1, gamma=0.0001)
```

Prediction and Interpretation

After training model, test the prediction, call the confusion matrix and classification report

Confusion Matrix: Report diagonal values

```
gd_pred = gd.predict(x_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

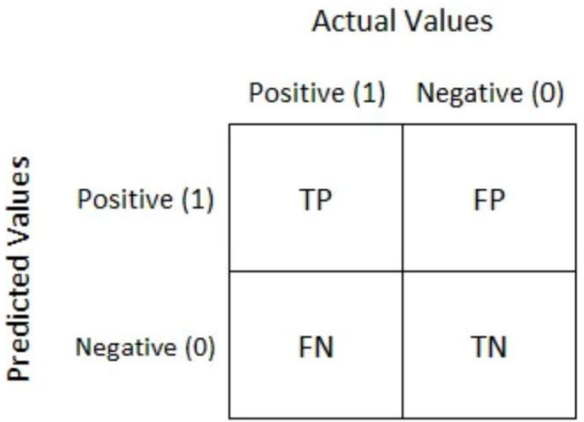
```
print(confusion_matrix(y_test,gd_pred))
print('\n')
print(classification_report(y_test, gd_pred))
```

```
[[ 56   6]
 [  4 105]]


              precision    recall  f1-score   support

           0       0.93      0.90      0.92        62
           1       0.95      0.96      0.95       109

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.94       171
weighted avg       0.94      0.94      0.94       171
```



Understand classification report

Accuracy: how often our classifier is right

Precision: how often the predicted positive is right

Recall: the model's ability to predict positive values,

How often model actually predict correct positive

F1-score: harmonic mean of recall and precision,

Taking both of them into consideration

Support: Actual occurrence in the training model

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1-Score} = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision} + \text{Recall}}$$

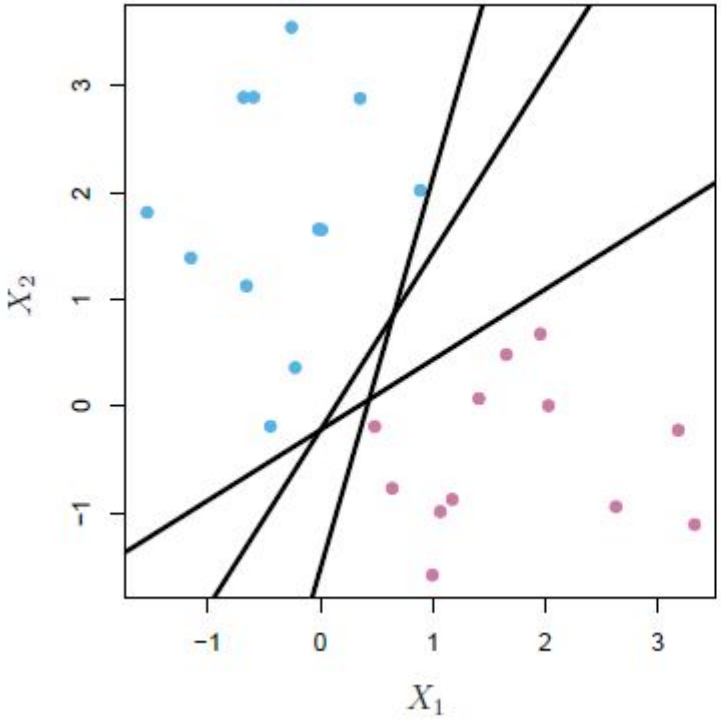
Hyperplanes

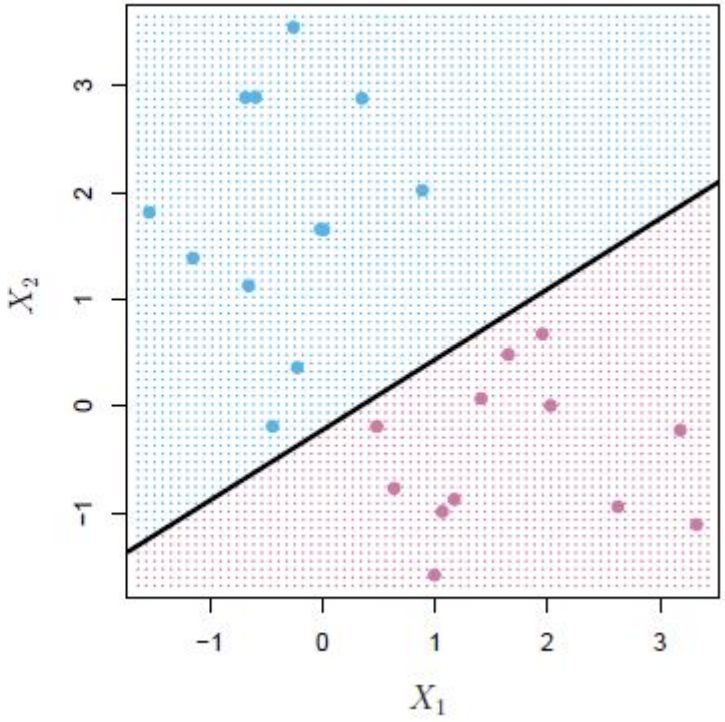
For a p dimensional data space, a hyperplane is a $p-1$ dimensional plane that divides the data into two groups of classification. For $X_1 \ldots X_p$ attributes, the hyperplane is of the form,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0, \text{ (Hastie)}$$

and satisfies properties that for $t_1 \ldots t_n \in \{1, -1\}$ and $i \in \{1 \ldots n\}$ where n is the number of observations in the data,

$$t_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0. \text{ (Hastie)}$$





(Hastie)

Maximal Margin Hyperplane

The goal of the maximal margin hyperplane, is to create a hyperplane with the greatest margin between the hyperplane and the nearest data points or support vectors.

In matrix notation,

$$t_i y(\mathbf{x}_i) = t_i(\mathbf{w}^T \mathbf{x}_i + w_0) > 0. \text{ (Bishop)}$$

The point $\mathbf{x}_i$ has a distance from the hyperplane of,

$$\frac{t_i y(\mathbf{x}_i)}{||\mathbf{w}||}. \text{ (Bishop)}$$

The margin is the distance between the hyperplane and the closest points to the hyperplane (support vectors),

$$\min_{i=1}^{n} \frac{t_i y(\mathbf{x}_i)}{||\mathbf{w}||}. \text{ (Murphy)}$$

We add constraint or rescaling factor,

$$\sum_{j=1}^{p} \beta_j^2 = 1, \text{ (Hastie)}$$

so that for a support vector $\mathbf{x}_i$, $t_i y(\mathbf{x}_i) = 1$.

So we can rewrite the previous constraint as,

$$t_i y(\mathbf{x}_i) = t_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \text{ (Bishop)}$$

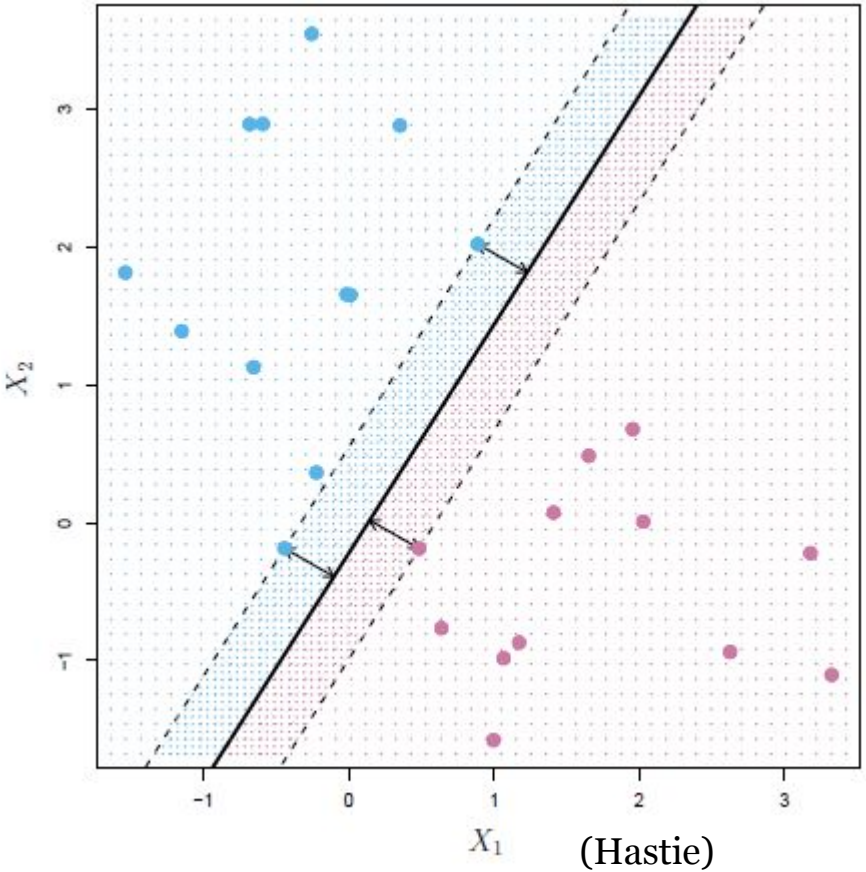
for all i.

And, now we seek the maximal margin by,

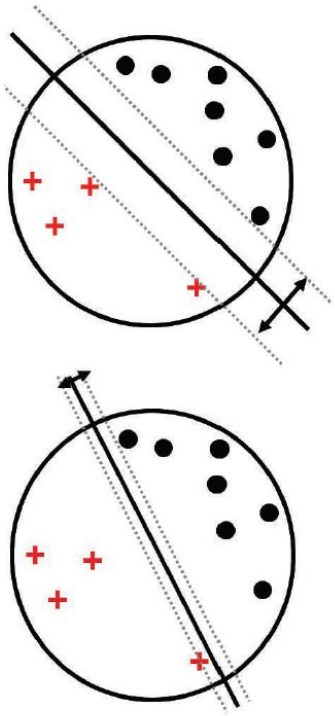
$$\max_{\mathbf{w},w_0} \min_{i=1}^{n} \frac{t_i y(\mathbf{x}_i)}{||\mathbf{w}||}. \text{ (Murphy)}$$

which can be simplified to minimizing $||\mathbf{w}||^2$,

$$\min_{\mathbf{w},w_0} \frac{1}{2}||\mathbf{w}||^2. \text{ (Murphy)}$$



(Hastie)

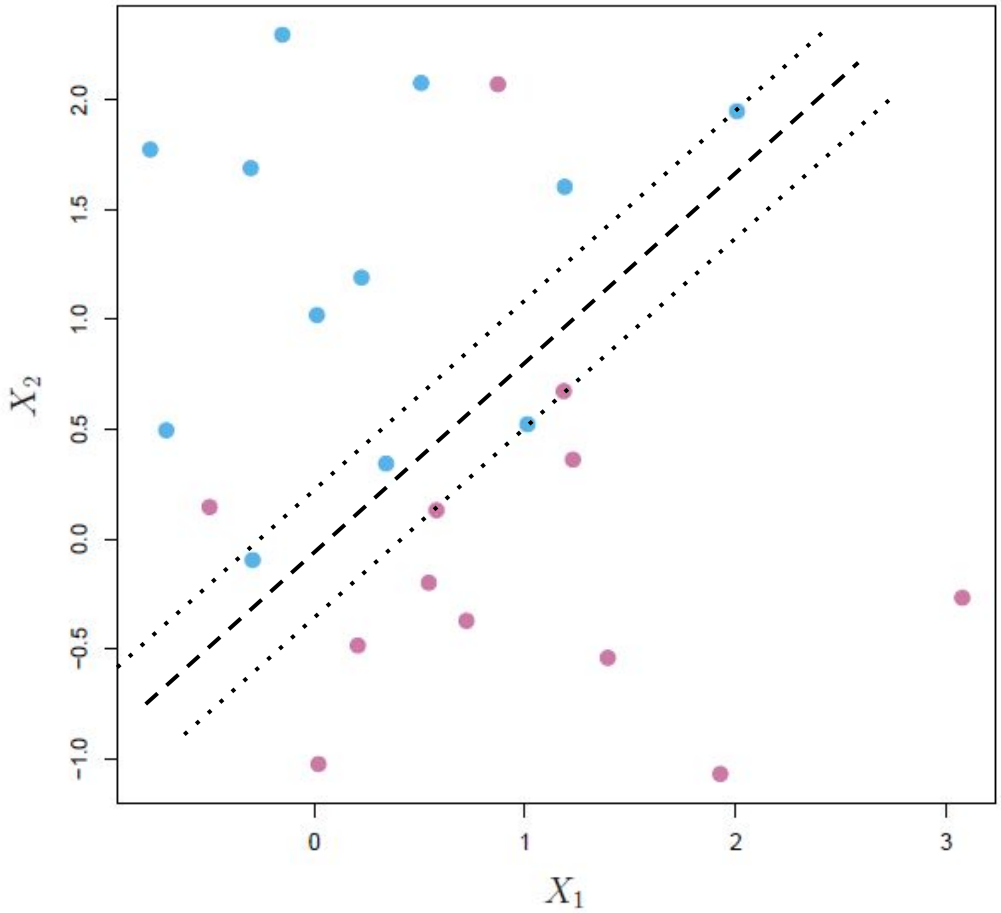


(Murphy)

Soft Margins

What if our data cannot be perfectly separated? That is where soft margins come in. We add the soft margin principle to the maximal margin classifier to get the support vector classifier or soft margin classifier. (Hastie)

The support vector classifier allows for some data points to be on the wrong side of the margin and even on the wrong side of the hyperplane subject to a tuning parameter. (Hastie)



(Hastie)
(lines added)

We introduce slack variables ϵ_i for $i \in (1 \ldots n)$ where,

$$\epsilon_i = 0, \text{for data points on the correct side of the margin,}$$

$$\epsilon_i = (t_i - y(\mathbf{x}_i)), \text{for data points on the wrong side of the margin.}$$

So, data points on the hyperplane have $\epsilon_i = 1$, and data points on the wrong side of the hyperplane have $\epsilon_i > 1$. And any points on the wrong side of the margin are considered support vectors.

Then we introduce the budget C where,

$$\sum_{i=1}^{n} \epsilon_i \leq C, \text{ (Hastie)}$$

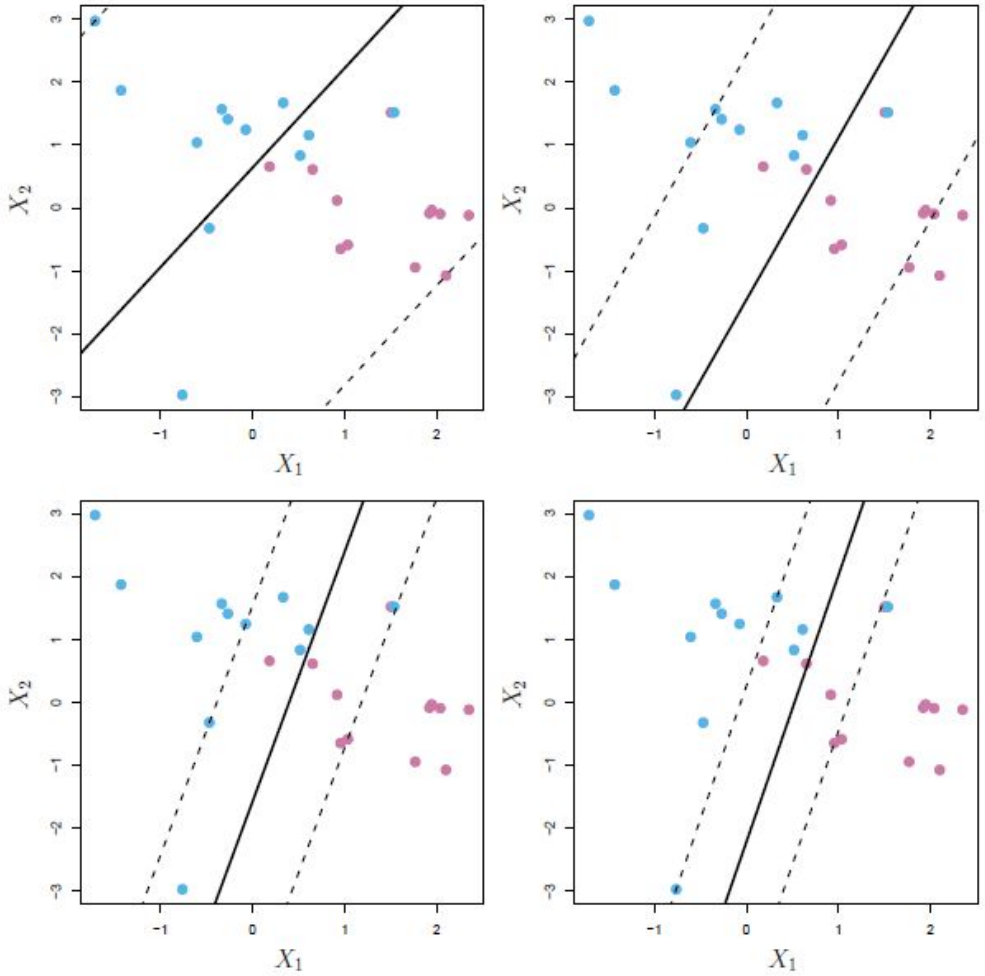
and again rewrite the previous constraint as,

$$t_i y(\mathbf{x}_i) = t_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \epsilon_i. \text{ (Bishop)}$$

And now we seek the optimal support vector classifier by,

(Hinge Loss)
$$\min_{\mathbf{w}, w_0, \epsilon} \left(\frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{n} \epsilon_i \right) . \text{ (Bishop)}$$

Different budgets, *C*, produce different support vector classifiers.
Use GridSearchCV to find an optimal budget.

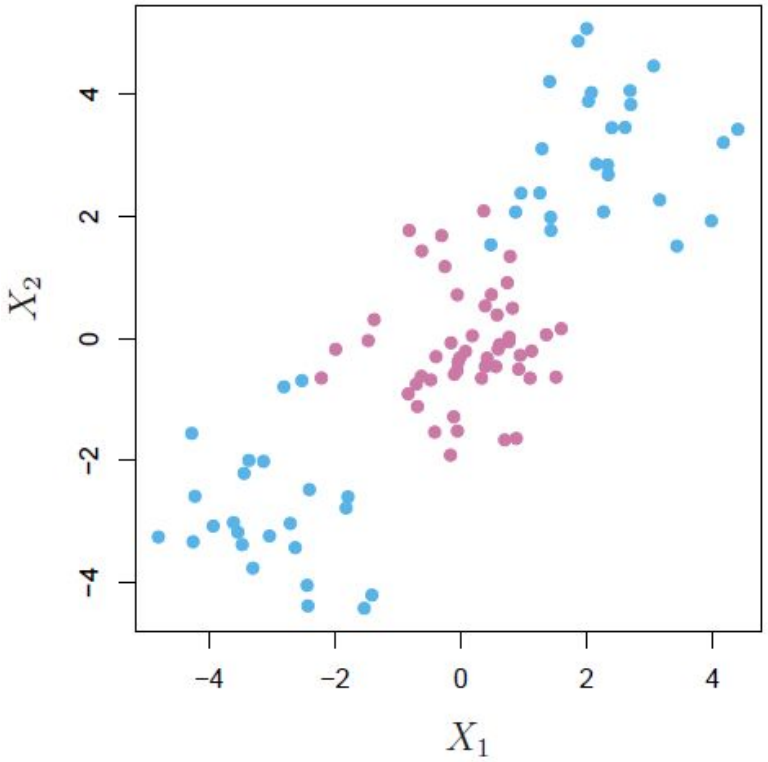


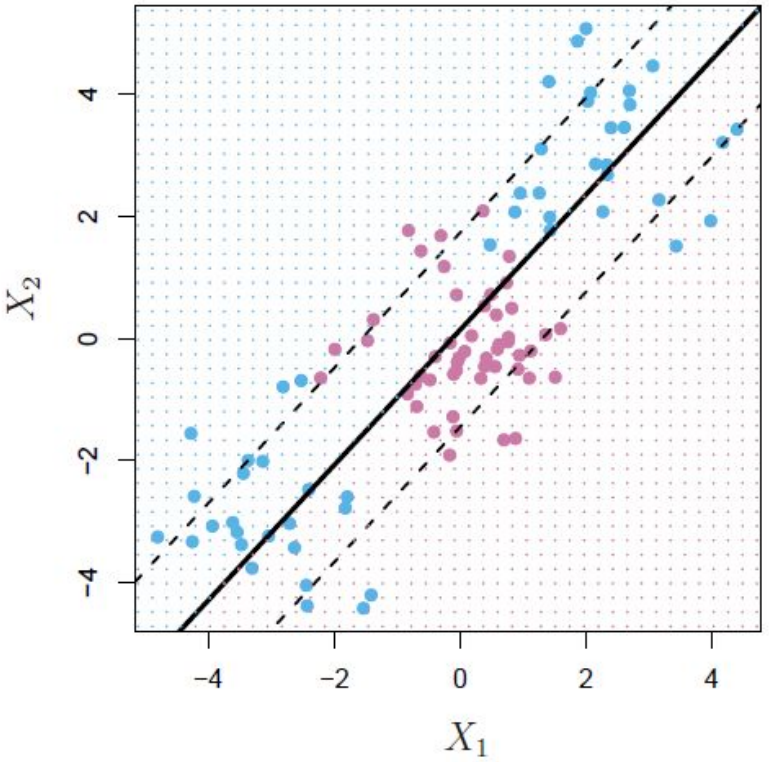
(Hastie)

Non-linear Classification

There are cases where it would be unsatisfactory to use a linear classifier for a particular data set. We need more flexibility in our model to handle these kind of data sets.

This is done by the use of kernels. Kernels are an efficient way to map our data in an extra dimension. Our support vector classifier now becomes a support vector machine. (Hastie)





(Hastie)

Kernels

Essentially, kernels, $k(\mathbf{x}, \mathbf{x}')$, measure the similarity between two data points. There are many different ways to measure similarity, and hence there are many different kernels. In the case of a linear kernel,

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}' \text{ (Bishop)}$$

We can use other kernels to create a non-linear classifier such as a polynomial kernel,

$$k(\mathbf{x}, \mathbf{x}') = 1 + (\mathbf{x}^T\mathbf{x}')^d \text{ (Hastie)}$$

where d is a positive integer, or a radial kernel,

$$k(\mathbf{x}, \mathbf{x}') = exp(-\gamma \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2) \text{ (Hastie)}$$

We use the kernels when seeking the optimal support vector classifier, then substitute the kernel into the classifier for classifying testing data.

The use of the linear kernel will return the same support vector classifier previously seen and looks like,

$$y(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i k(\mathbf{x_i}, \mathbf{x}) \text{ (Hastie)}$$

where $\alpha_i, i \in \{1 \dots n\}$ are parameters for each observation and is non-zero only for non-support vectors.

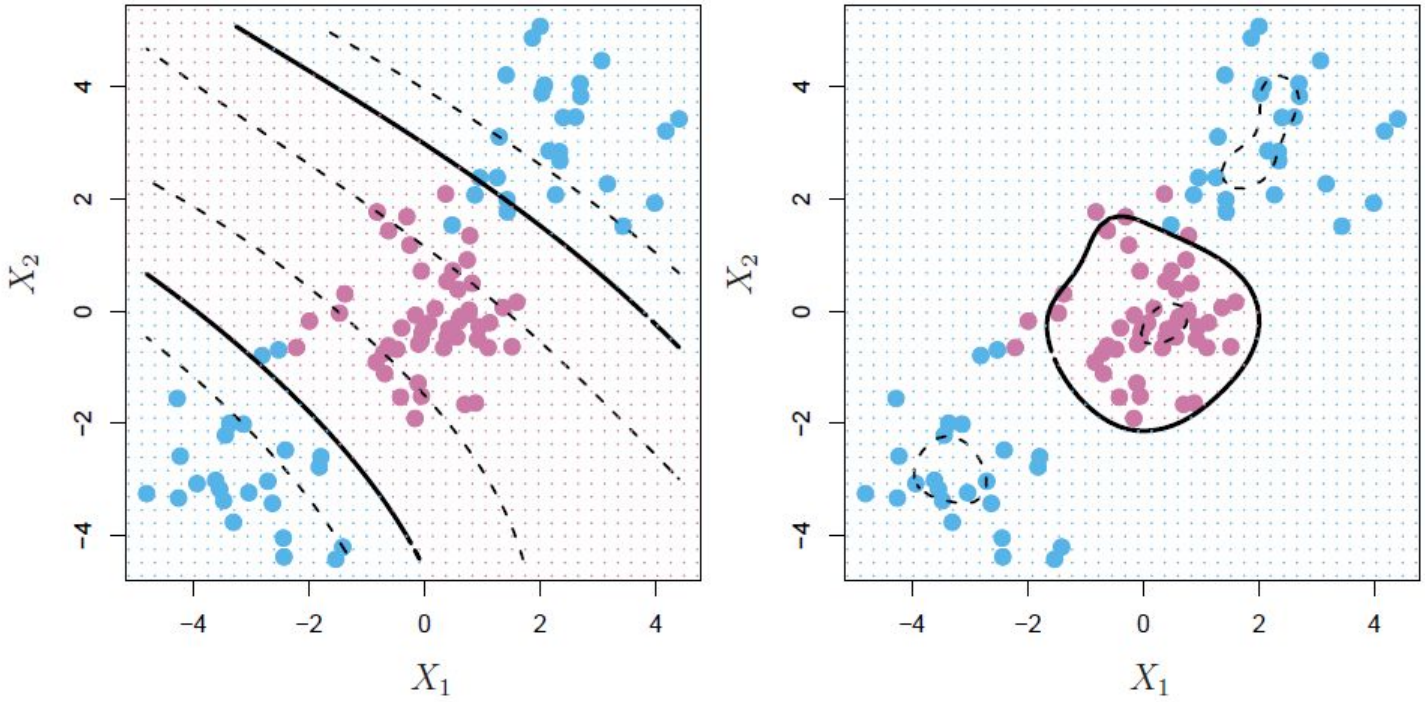
The optimal support vector machine parameters are found by minimizing,

$$\sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} a_n a_m t_n t_m k(\mathbf{x_n}, \mathbf{x_m}) \text{ (Bishop)}$$

subject to constraints,

$$0 \leq a_n \leq C,$$

$$\sum_{n=1}^{N} a_n t_n = 0$$



(Hastie)

Multiple Classes

There are a couple of ways to do multi-class SVM.

The one-vs-all method trains one class as a positive and the rest of the classes as a negative for each class.
The one-vs-one method trains each pair of classes.

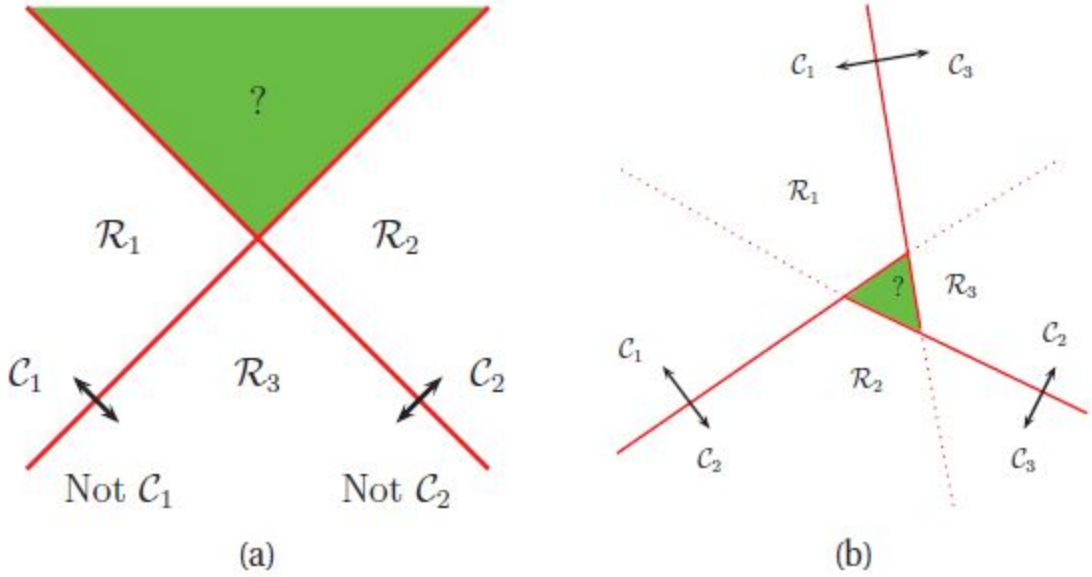


Figure 14.14 (a) The one-versus-rest approach. The green region is predicted to be both class 1 and class 2. (b) The one-versus-one approach. The label of the green region is ambiguous. Based on Figure 4.2 of (Bishop 2006a).

(Murphy)

Comparison with other Classifiers

- SVM vs Logistic Regression:

  SVM can handle non-linear solutions whereas logistic regression can only handle linear solutions.

  Linear SVM handles outliers better, as it derives maximum margin solution.

  Hinge loss in SVM outperforms log loss in LR.

- SVM vs Random Forest

  Random Forest is intrinsically suited for multiclass problems, while SVM is intrinsically two-class.

  For a classification problem Random Forest gives you probability of belonging to class. SVM gives you distance to the boundary

References

Hastie, Trevor; James, Gareth; Witten, Daniela; Tibshirani, Robert; Taylor, Jonathan. "*An Introduction to Statistical Learning with applications in Python*". "https://www.statlearning.com/". 2023.

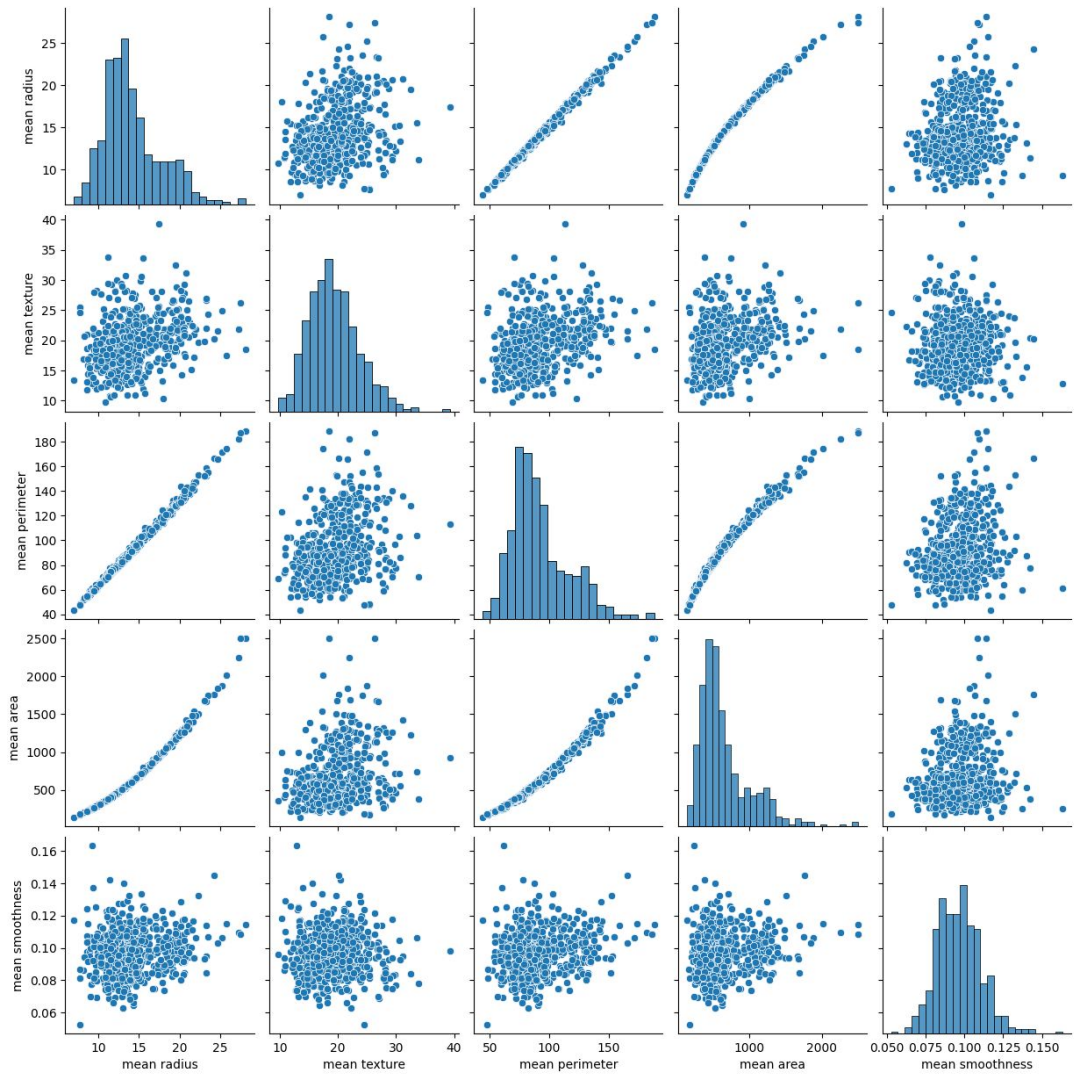
Bishop, Christopher M. *"Pattern Recognition and Machine Learning"*. Springer. 978-0387-31073-2. 2006.

Murphy, Kevin P. *"Machine Learning: A Probabilistic Perspective"*. The MIT Press. 978-0-262-01802-9. 2012

Demonstration Screenshots

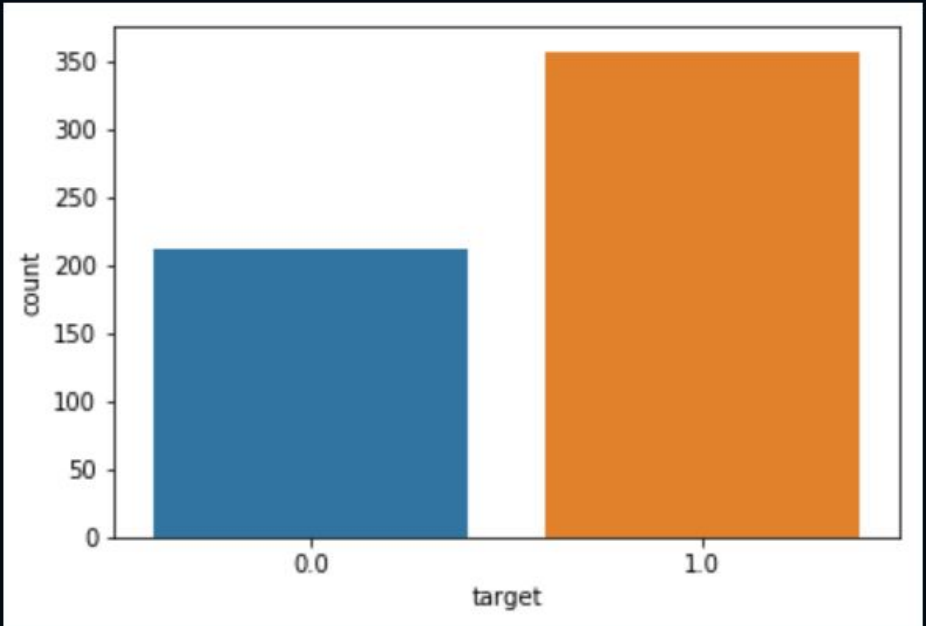
VISUALISING THE DATA

```
sns.pairplot(df_cancer , vars =['mean radius','mean texture', 'mean perimeter', 'mean area',
 'mean smoothness', 'mean compactness' ,'mean concavity',
 'mean concave points', 'mean symmetry', 'mean fractal dimension',
 'radius error', 'texture error', 'perimeter error', 'area error',
 'smoothness error', 'compactness error', 'concavity error',
 'concave points error', 'symmetry error', 'fractal dimension error',
 'worst radius', 'worst texture', 'worst perimeter', 'worst area',
 'worst smoothness', 'worst compactness', 'worst concavity',
 'worst concave points', 'worst symmetry', 'worst fractal dimension'])
```


mean radius
mean texture
mean perimeter
mean area
mean smoothness
10 15 20 25
10 20 30 40
50 100 150
1000 2000
0.050 0.075 0.100 0.125 0.150
0.06 0.08 0.10 0.12 0.14 0.16
500 1000 1500 2000 2500
40 60 80 100 120 140 160 180
10 15 20 25 30 35 40

```
#will simply tell you how many Malignent and Benign cases we have.
#Malignent= 200~ and Benign = 350~ approx.
sns.countplot(df_cancer['target'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba9340f650>
```

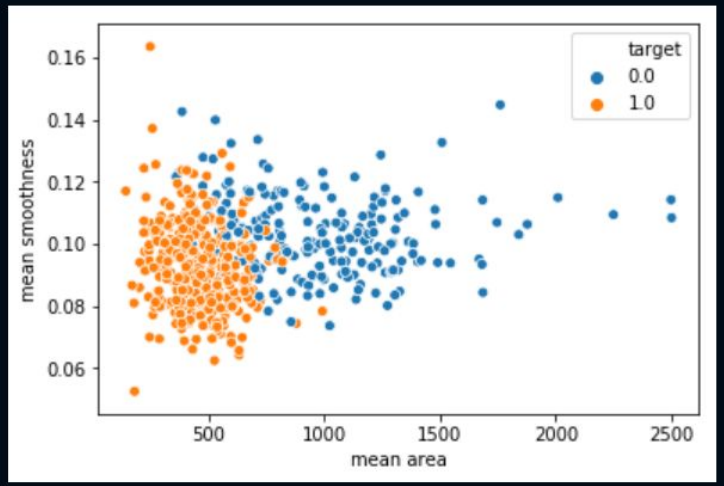


```
#plotting a scatter plot diagram for mean area anf mean smoothness, you can plot any feature combination scatterplot.
sns.scatterplot(x='mean area',y='mean smoothness',hue='target',data =df_cancer)
```

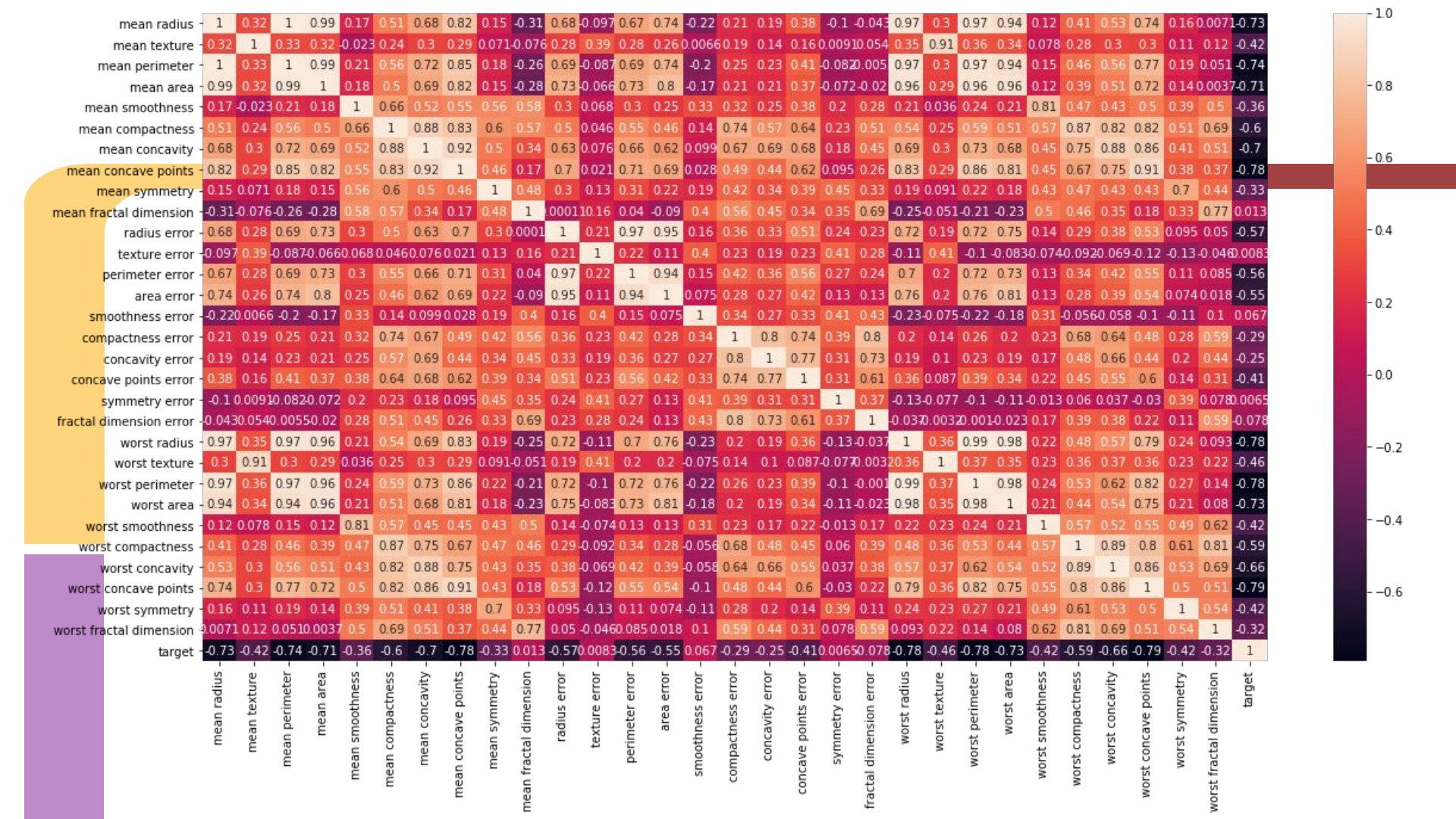
8]

Python

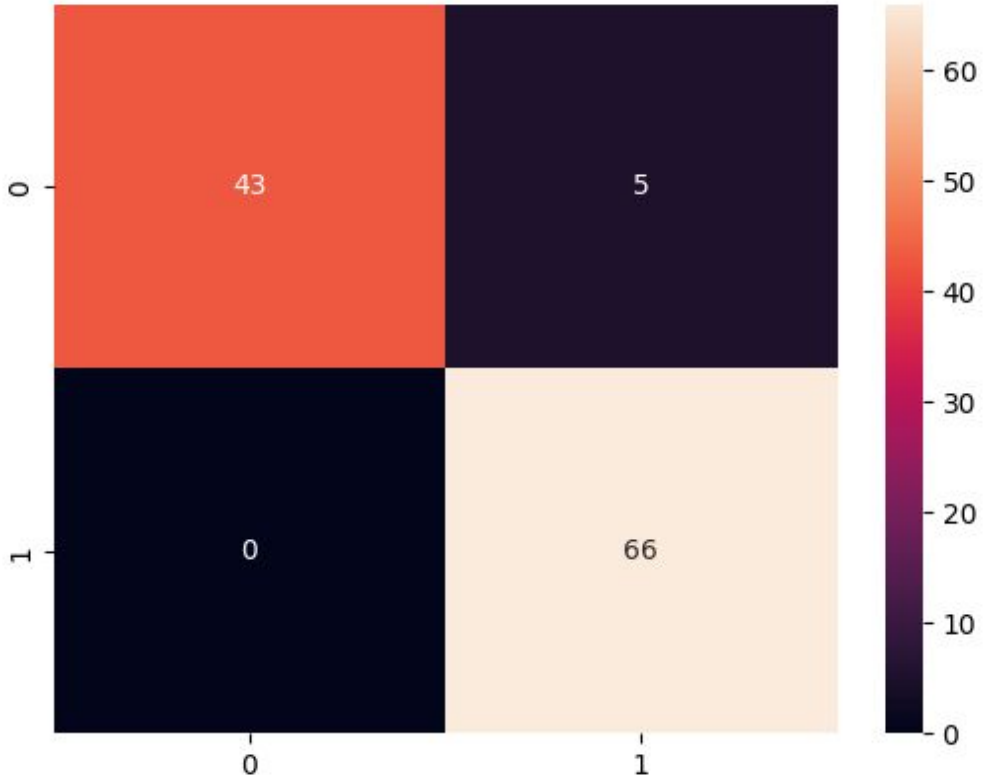
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba93971510>
```



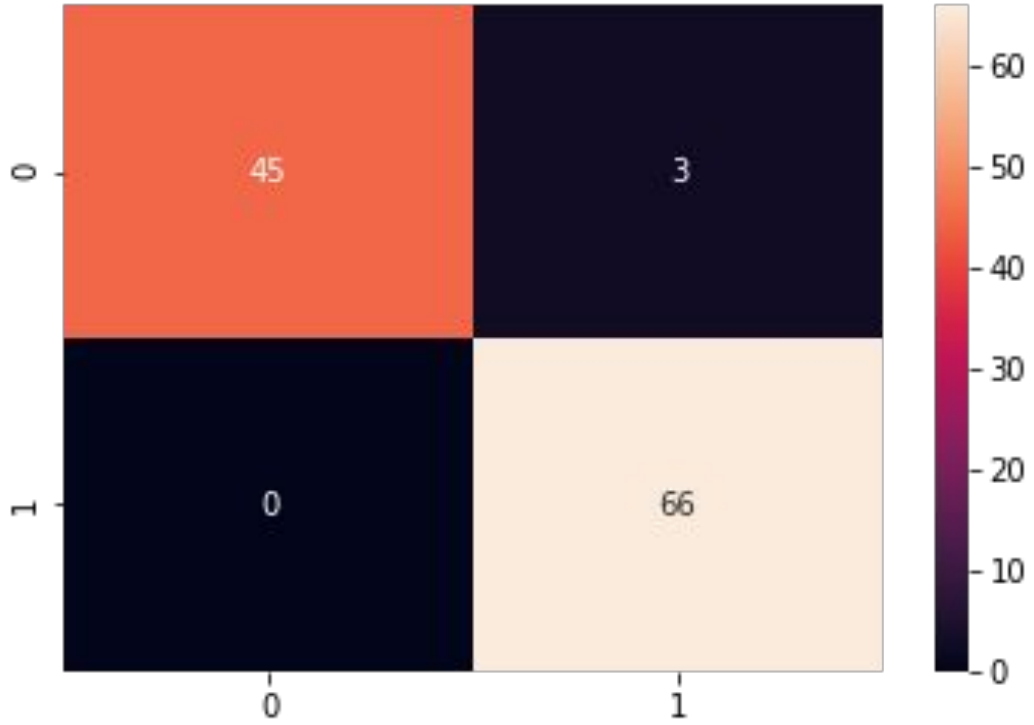
```
#here we made a heatmap figure of correlation of all the columns
plt.figure(figsize =(20,10))
sns.heatmap(df_cancer.corr(), annot =True)
```


mean radius 1 0.32 1 0.99 0.17 0.51 0.68 0.82 0.15 -0.31 0.68 -0.097 0.67 0.74 -0.22 0.21 0.19 0.38 -0.1 -0.043 0.97 0.3 0.97 0.94 0.12 0.41 0.53 0.74 0.16 0.0071 -0.73
mean texture 0.32 1 0.33 0.32 -0.023 0.24 0.3 0.29 0.071 -0.076 0.28 0.39 0.28 0.26 0.0066 0.19 0.14 0.16 0.0091 -0.054 0.35 0.91 0.36 0.34 0.078 0.28 0.3 0.3 0.11 0.12 -0.42
mean perimeter 1 0.33 1 0.99 0.21 0.56 0.72 0.85 0.18 -0.26 0.69 -0.087 0.69 0.74 -0.2 0.25 0.23 0.41 -0.082 0.005 0.97 0.3 0.97 0.94 0.15 0.46 0.56 0.77 0.19 0.051 -0.74
mean area 0.99 0.32 0.99 1 0.18 0.5 0.69 0.82 0.15 -0.28 0.73 -0.066 0.73 0.8 -0.17 0.21 0.21 0.37 -0.072 -0.02 0.96 0.29 0.96 0.96 0.12 0.39 0.51 0.72 0.14 0.0037 -0.71
mean smoothness 0.17 -0.023 0.21 0.18 1 0.66 0.52 0.55 0.56 0.58 0.3 0.068 0.3 0.25 0.33 0.32 0.25 0.38 0.2 0.28 0.21 0.036 0.24 0.21 0.81 0.47 0.43 0.5 0.39 0.5 -0.36
mean compactness 0.51 0.24 0.56 0.5 0.66 1 0.88 0.83 0.6 0.57 0.5 0.046 0.55 0.46 0.14 0.74 0.57 0.64 0.23 0.51 0.54 0.25 0.59 0.51 0.57 0.87 0.82 0.82 0.51 0.69 -0.6
mean concavity 0.68 0.3 0.72 0.69 0.52 0.88 1 0.92 0.5 0.34 0.63 0.076 0.66 0.62 0.099 0.67 0.69 0.68 0.18 0.45 0.69 0.3 0.73 0.68 0.45 0.75 0.88 0.86 0.41 0.51 -0.7
mean concave points 0.82 0.29 0.85 0.82 0.55 0.83 0.92 1 0.46 0.17 0.7 0.021 0.71 0.69 0.028 0.49 0.44 0.62 0.095 0.26 0.83 0.29 0.86 0.81 0.45 0.67 0.75 0.91 0.38 0.37 -0.78
mean symmetry 0.15 0.071 0.18 0.15 0.56 0.6 0.5 0.46 1 0.48 0.3 0.13 0.31 0.22 0.19 0.42 0.34 0.39 0.45 0.33 0.19 0.091 0.22 0.18 0.43 0.47 0.43 0.43 0.7 0.44 -0.33
mean fractal dimension -0.31 -0.076 -0.26 -0.28 0.58 0.57 0.34 0.17 0.48 1 0.0001 0.16 0.04 -0.09 0.4 0.56 0.45 0.34 0.35 0.69 -0.25 -0.051 -0.21 -0.23 0.5 0.46 0.35 0.18 0.33 0.77 0.013
radius error 0.68 0.28 0.69 0.73 0.3 0.5 0.63 0.7 0.3 0.0001 1 0.21 0.97 0.95 0.16 0.36 0.33 0.51 0.24 0.23 0.72 0.19 0.72 0.75 0.14 0.29 0.38 0.53 0.095 0.05 -0.57
texture error -0.097 0.39 -0.087 -0.066 0.068 0.046 0.076 0.021 0.13 0.16 0.21 1 0.22 0.11 0.4 0.23 0.19 0.23 0.41 0.28 -0.11 0.41 -0.1 -0.083 -0.074 -0.092 -0.069 -0.12 -0.13 -0.046 0.0083
perimeter error 0.67 0.28 0.69 0.73 0.3 0.55 0.66 0.71 0.31 0.04 0.97 0.22 1 0.94 0.15 0.42 0.36 0.56 0.27 0.24 0.7 0.2 0.72 0.73 0.13 0.34 0.42 0.55 0.11 0.085 -0.56
area error 0.74 0.26 0.74 0.8 0.25 0.46 0.62 0.69 0.22 -0.09 0.95 0.11 0.94 1 0.075 0.28 0.27 0.42 0.13 0.13 0.76 0.2 0.76 0.81 0.13 0.28 0.39 0.54 0.074 0.018 -0.55
smoothness error -0.22 0.0066 -0.2 -0.17 0.33 0.14 0.099 0.028 0.19 0.4 0.16 0.4 0.15 0.075 1 0.34 0.27 0.33 0.41 0.43 -0.23 -0.075 -0.22 -0.18 0.31 -0.056 0.058 -0.1 -0.11 0.1 0.067
compactness error 0.21 0.19 0.25 0.21 0.32 0.74 0.67 0.49 0.42 0.56 0.36 0.23 0.42 0.28 0.34 1 0.8 0.74 0.39 0.8 0.2 0.14 0.26 0.2 0.23 0.68 0.64 0.48 0.28 0.59 -0.29
concavity error 0.19 0.14 0.23 0.21 0.25 0.57 0.69 0.44 0.34 0.45 0.33 0.19 0.36 0.27 0.27 0.8 1 0.77 0.31 0.73 0.19 0.1 0.23 0.19 0.17 0.48 0.66 0.44 0.2 0.44 -0.25
concave points error 0.38 0.16 0.41 0.37 0.38 0.64 0.68 0.62 0.39 0.34 0.51 0.23 0.56 0.42 0.33 0.74 0.77 1 0.31 0.61 0.36 0.087 0.39 0.34 0.22 0.45 0.55 0.6 0.14 0.31 -0.41
symmetry error -0.1 0.0091 -0.082 -0.072 0.2 0.23 0.18 0.095 0.45 0.35 0.24 0.41 0.27 0.13 0.41 0.39 0.31 0.31 1 0.37 -0.13 -0.077 -0.1 -0.11 -0.013 0.06 0.037 -0.03 0.39 0.078 0.0065
fractal dimension error -0.043 0.054 -0.005 -0.02 0.28 0.51 0.45 0.26 0.33 0.69 0.23 0.28 0.24 0.13 0.43 0.8 0.73 0.61 0.37 1 -0.037 0.0032 0.001 -0.023 0.17 0.39 0.38 0.22 0.11 0.59 -0.078
worst radius 0.97 0.35 0.97 0.96 0.21 0.54 0.69 0.83 0.19 -0.25 0.72 -0.11 0.7 0.76 -0.23 0.2 0.19 0.36 -0.13 -0.037 1 0.36 0.99 0.98 0.22 0.48 0.57 0.79 0.24 0.093 -0.78
worst texture 0.3 0.91 0.3 0.29 0.036 0.25 0.3 0.29 0.091 -0.051 0.19 0.41 0.2 0.2 -0.075 0.14 0.1 0.087 -0.077 -0.003 0.36 1 0.37 0.35 0.23 0.36 0.37 0.36 0.23 0.22 -0.46
worst perimeter 0.97 0.36 0.97 0.96 0.24 0.59 0.73 0.86 0.22 -0.21 0.72 -0.1 0.72 0.76 -0.22 0.26 0.23 0.39 -0.1 -0.001 0.99 0.37 1 0.98 0.24 0.53 0.62 0.82 0.27 0.14 -0.78
worst area 0.94 0.34 0.94 0.96 0.21 0.51 0.68 0.81 0.18 -0.23 0.75 -0.083 0.73 0.81 -0.18 0.2 0.19 0.34 -0.11 -0.023 0.98 0.35 0.98 1 0.21 0.44 0.54 0.75 0.21 0.08 -0.73
worst smoothness 0.12 0.078 0.15 0.12 0.81 0.57 0.45 0.45 0.43 0.5 0.14 -0.074 0.13 0.13 0.31 0.23 0.17 0.22 -0.013 0.17 0.22 0.23 0.24 0.21 1 0.57 0.52 0.55 0.49 0.62 -0.42
worst compactness 0.41 0.28 0.46 0.39 0.47 0.87 0.75 0.67 0.47 0.46 0.29 -0.092 0.34 0.28 -0.056 0.68 0.48 0.45 0.06 0.39 0.48 0.36 0.53 0.44 0.57 1 0.89 0.8 0.61 0.81 -0.59
worst concavity 0.53 0.3 0.56 0.51 0.43 0.82 0.88 0.75 0.43 0.35 0.38 -0.069 0.42 0.39 -0.058 0.64 0.66 0.55 0.037 0.38 0.57 0.37 0.62 0.54 0.52 0.89 1 0.86 0.53 0.69 -0.66
worst concave points 0.74 0.3 0.77 0.72 0.5 0.82 0.86 0.91 0.43 0.18 0.53 -0.12 0.55 0.54 -0.1 0.48 0.44 0.6 -0.03 0.22 0.79 0.36 0.82 0.75 0.55 0.8 0.86 1 0.5 0.51 -0.79
worst symmetry 0.16 0.11 0.19 0.14 0.39 0.51 0.41 0.38 0.7 0.33 0.095 -0.13 0.11 0.074 -0.11 0.28 0.2 0.14 0.39 0.11 0.24 0.23 0.27 0.21 0.49 0.61 0.53 0.5 1 0.54 -0.42
worst fractal dimension 0.0071 0.12 0.051 0.0037 0.5 0.69 0.51 0.37 0.44 0.77 0.05 -0.046 0.085 0.018 0.1 0.59 0.44 0.31 0.078 0.59 0.093 0.22 0.14 0.08 0.62 0.81 0.69 0.51 0.54 1 -0.32
target -0.73 -0.42 -0.74 -0.71 -0.36 -0.6 -0.7 -0.78 -0.33 0.013 -0.57 0.0083 -0.56 -0.55 0.067 -0.29 -0.25 -0.41 0.0065 -0.078 -0.78 -0.46 -0.78 -0.73 -0.42 -0.59 -0.66 -0.79 -0.42 -0.32 1
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
target
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Confusion Matrix for LR



Confusion Matrix for SVM



Confusion Matrix for RF