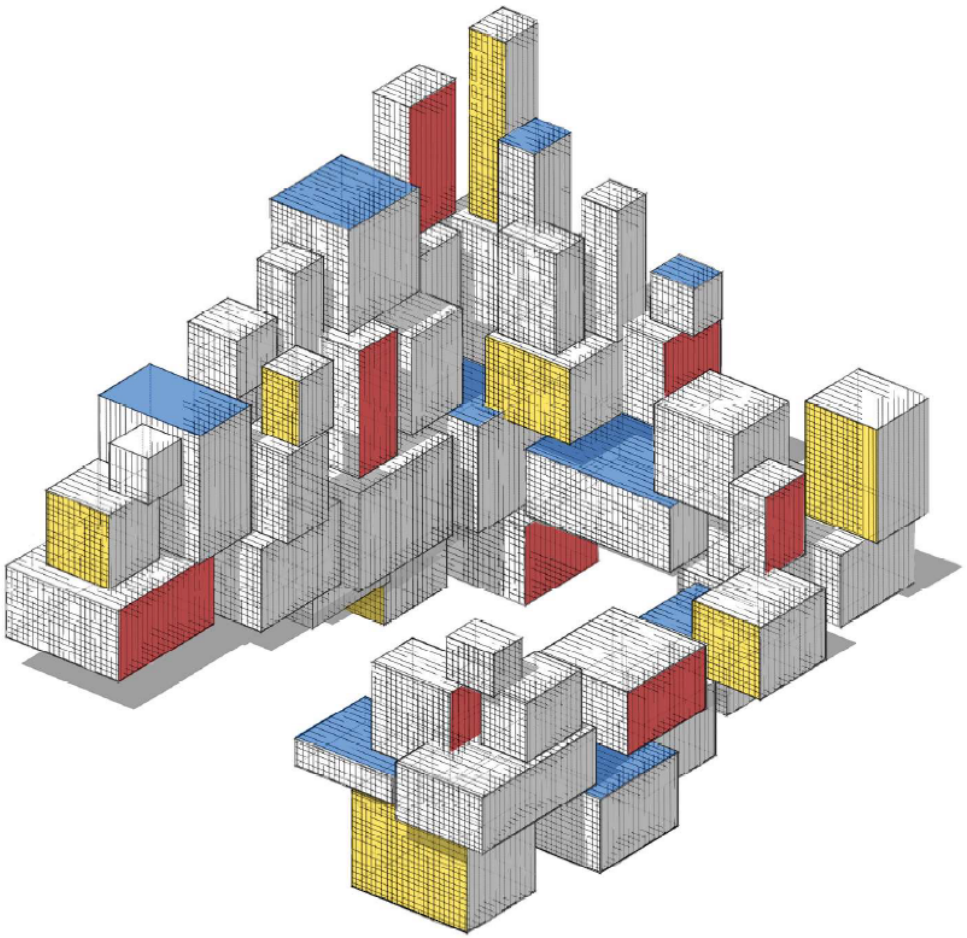
Support Vector Machines



Purpose

In this lecuture we discuss:

- Hinge loss
- Support vectors
- Support vector machines
- Classificaton with **`sklearn`**

Binary Classification

We are given the training set $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where each response y_i takes either the value -1 or 1.

We wish to construct a classifier taking values in $\{-1, 1\}$.

The optimal classification function for the indicator loss, $\mathbb{I}\{y \neq \widehat{y}\}$, is equal to

$$g^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad \mathbb{P}[Y = 1 \mid \boldsymbol{X} = \boldsymbol{x}] \geqslant 1/2, \\ -1 & \text{if} \quad \mathbb{P}[Y = 1 \mid \boldsymbol{X} = \boldsymbol{x}] < 1/2. \end{cases}$$

Hingle Loss

The function g^* can be viewed as the minimizer of the risk for the hinge loss function,

$$\text{Loss}(y, \widehat{y}) = (1 - y\,\widehat{y})_+ := \max\{0,\ 1 - y\,\widehat{y}\},$$

over all prediction functions g (not necessarily taking values only in the set $\{-1, 1\}$).

That is,

$$g^* = \underset{g}{\text{argmin}}\, \mathbb{E}\,(1 - Y\,g(\boldsymbol{X}))_+. \tag{1}$$

For a training set τ, we can approximate the risk $\ell(g) = \mathbb{E}\,(1 - Y\,g(\boldsymbol{X}))_+$ by the training loss

$$\ell_\tau(g) = \frac{1}{n}\sum_{i=1}^{n} (1 - y_i\,g(\boldsymbol{x}_i))_+,$$

and minimize this over a (smaller) class of functions to obtain the optimal prediction function g_τ.

Classifier

Finally, as the prediction function g_τ generally is not a classifier by itself (it usually does not only take values -1 or 1), we take the classifier

$$\operatorname{sign} g_\tau(\boldsymbol{x}).$$

Therefore, a feature vector $\boldsymbol{x}$ is classified according to 1 or -1 depending on whether $g_\tau(\boldsymbol{x}) \geqslant 0$ or < 0, respectively.

The optimal decision boundary is given by the set of $\boldsymbol{x}$ for which $g_\tau(\boldsymbol{x}) = 0$.

In a RKHS setting, we can consider finding the best classifier, given the training data, via the penalized goodness-of-fit optimization:

$$\min_{g \in \mathcal{H} \oplus \mathcal{H}_0} \frac{1}{n} \sum_{i=1}^{n} [1 - y_i \, g(\boldsymbol{x}_i)]_+ + \widetilde{\gamma} \, \|g\|_{\mathcal{H}}^2,$$

for some regularization parameter $\widetilde{\gamma}$.

Regularized Optimization Problem

Defining $\gamma := 2n\widetilde{\gamma}$, this is equivalent to solving:

$$\min_{g \in \mathcal{H} \oplus \mathcal{H}_0} \sum_{i=1}^{n} [1 - y_i \, g(\boldsymbol{x}_i)]_+ + \frac{\gamma}{2} \, \|g\|_{\mathcal{H}}^2.$$

We know from the representer theorem that if κ is the reproducing kernel corresponding to $\mathcal{H}$, then the solution is of the form (assuming that the null space $\mathcal{H}_0$ has a constant term only):

$$g(\boldsymbol{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}). \tag{2}$$

Substituting into the minimization expression yields the convex optimization problem:

$$\min_{\boldsymbol{\alpha}, \alpha_0} \sum_{i=1}^{n} [1 - y_i(\alpha_0 + \{\mathbf{K}\boldsymbol{\alpha}\}_i)]_+ + \frac{\gamma}{2} \, \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}, \tag{3}$$

where $\mathbf{K}$ is the Gram matrix.

Dual Optimization Problem

Defining $\lambda_i := \gamma\alpha_i / y_i$, $i = 1, \ldots, n$ and $\boldsymbol{\lambda} := [\lambda_1, \ldots, \lambda_n]^\top$, it can be shown (exercise!) that the optimal $\boldsymbol{\alpha}$ and α_0 in (3) can be obtained by solving the "dual" convex optimization problem

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & \sum_{i=1}^{n} \lambda_i - \frac{1}{2\gamma} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{subject to:} \quad & \boldsymbol{\lambda}^\top \boldsymbol{y} = 0, \ \mathbf{0} \leqslant \boldsymbol{\lambda} \leqslant \mathbf{1}, \end{aligned} \tag{4}$$

and $\alpha_0 = y_j - \sum_{i=1} \alpha_i \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for any j for which $\lambda_j \in (0, 1)$. In view of (2), the optimal prediction function (pre-classifier) g_τ is then given by

$$g_\tau(\boldsymbol{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}) = \alpha_0 + \frac{1}{\gamma} \sum_{i=1}^{n} y_i \lambda_i \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}). \tag{5}$$

Suport Vectors

From (5), the optimal pre-classifier $g(\boldsymbol{x})$ and the classifier sign $g(\boldsymbol{x})$ only depend on vectors $\boldsymbol{x}_i$ for which $\lambda_i \neq 0$.

These vectors are called the support vectors of the support vector machine.

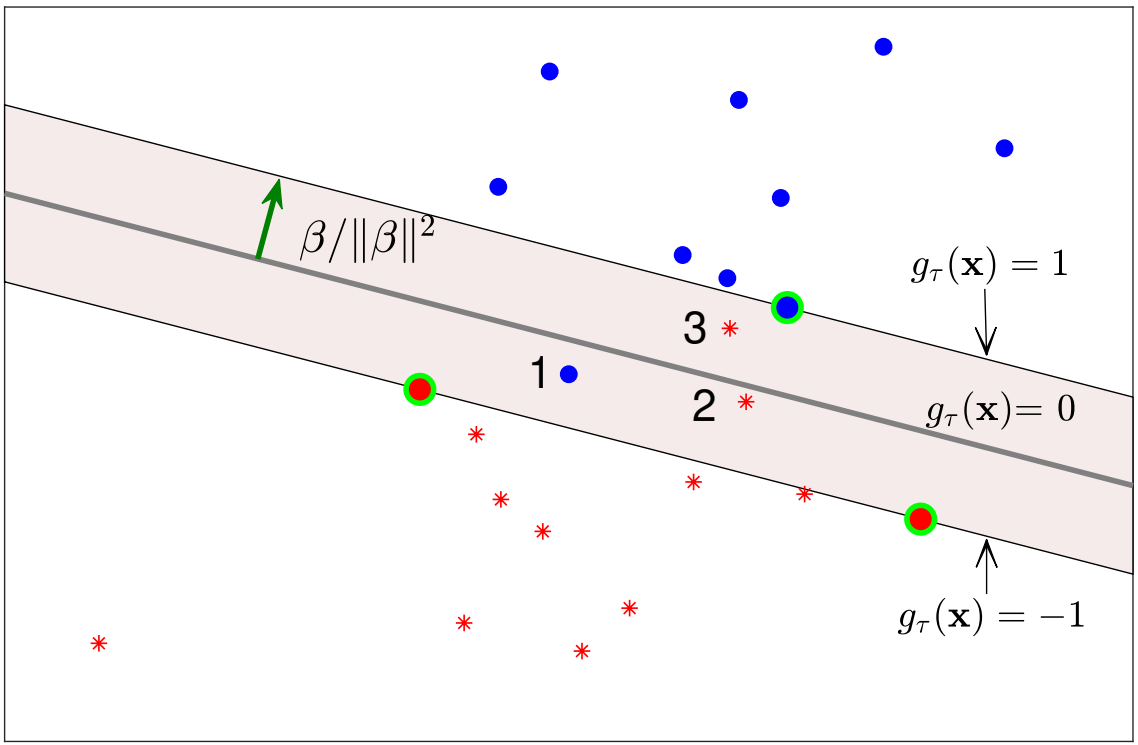
By defining $\nu_i := \lambda_i/\gamma$, $i = 1, \ldots, n$, we can rewrite (4) as

$$\begin{aligned} \min_{\nu} \quad & \frac{1}{2} \sum_{i,j} \nu_i \nu_j y_i y_j \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{n} \nu_i \\ \text{subject to:} \quad & \sum_{i=1}^{n} \nu_i y_i = 0, \;\; 0 \leqslant \nu_i \leqslant 1/\gamma =: C, \quad i = 1, \ldots, n. \end{aligned} \tag{6}$$

For perfectly separable data, we may take $C = \infty$; otherwise, C needs to be chosen via cross-validation or a test data set, for example.

Geometric interpretation

For the linear kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$, we have $g_\tau(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}$, with $\beta_0 = \alpha_0$ and $\boldsymbol{\beta} = \gamma^{-1} \sum_{i=1}^n \lambda_i y_i \boldsymbol{x}_i = \sum_{i=1}^n \alpha_i \boldsymbol{x}_i$. The decision boundary is formed by the points $\boldsymbol{x}$ such that $g_\tau(\boldsymbol{x}) = 0$. The two sets $\{\boldsymbol{x} : g_\tau(\boldsymbol{x}) = -1\}$ and $\{\boldsymbol{x} : g_\tau(\boldsymbol{x}) = 1\}$ are called the margins. The distance from the points on a margin to the decision boundary is $1/\|\boldsymbol{\beta}\|$.



Categories of Training Samples

Based on the "multipliers" $\{\lambda_i\}$, we can divide the training samples $\{(\boldsymbol{x}_i, y_i)\}$ into three categories:

- Points for which $\lambda_i \in (0, 1)$. These are the support vectors on the margins (green encircled in the figure) and are correctly classified.
- Points for which $\lambda_i = 1$. These points, which are also support vectors, lie strictly inside the margins (points 1, 2, and 3 in the figure). Such points may or may not be correctly classified.
- Points for which $\lambda_i = 0$. These are the non-support vectors, which all lie outside the margins. Every such point is correctly classified.

Separable Classes

If the classes of points $\{\boldsymbol{x}_i : y_i = 1\}$ and $\{\boldsymbol{x}_i : y_i = -1\}$ are separable by some affine plane, then there will be no points strictly inside the margins, so all support vectors will lie exactly on the margins.

In this case (3) reduces to

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0} \ \|\boldsymbol{\beta}\|^2 \\ &\text{subject to:} \quad y_i(\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}) \geqslant 1, \ i = 1, \ldots, n, \end{aligned} \tag{7}$$

using the fact that $\alpha_0 = \beta_0$ and $\mathbf{K}\boldsymbol{\alpha} = \mathbf{X}\mathbf{X}^\top\boldsymbol{\alpha} = \mathbf{X}\boldsymbol{\beta}$.

We may replace $\min \|\boldsymbol{\beta}\|^2$ in (7) with $\max 1/\|\boldsymbol{\beta}\|$, as this gives the same optimal solution. As $1/\|\boldsymbol{\beta}\|$ is equal to half the margin width, the latter optimization problem has a simple interpretation:

Separate the points via an affine hyperplane such that the margin width is maximized.

Example: Support Vector Machine

The data below were uniformly generated on the unit disc. Class-1 points (blue dots) have a radius less than 1/2 (y-values 1) and class-2 points (red crosses) have a radius greater than 1/2 (y-values -1).

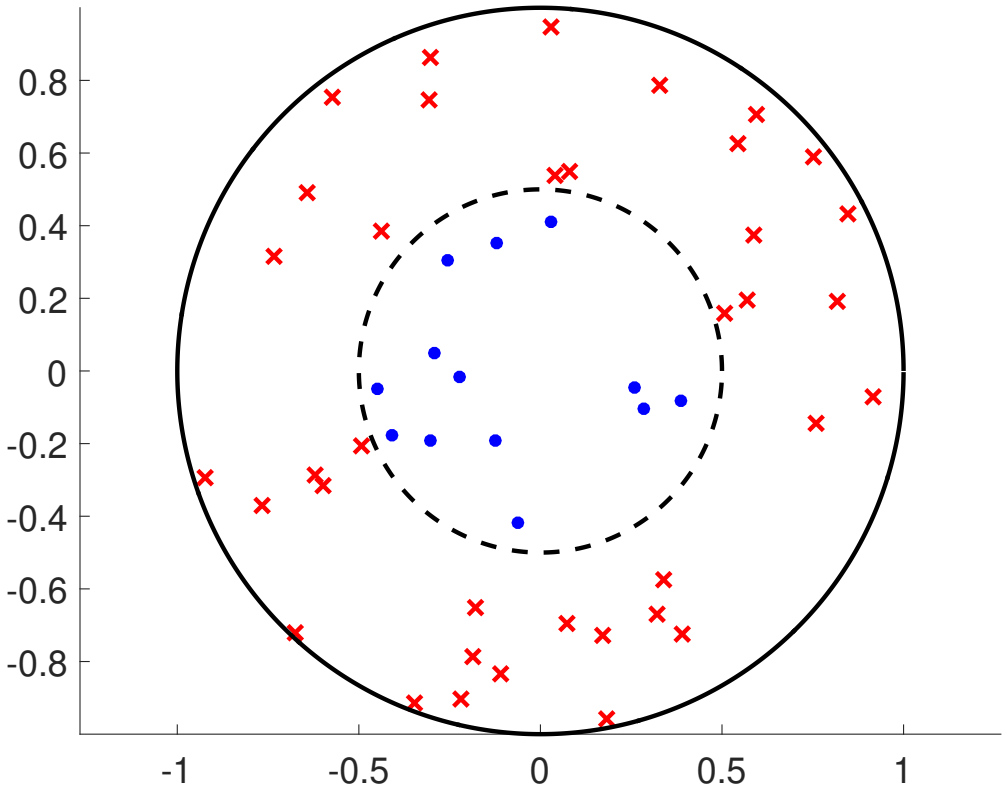


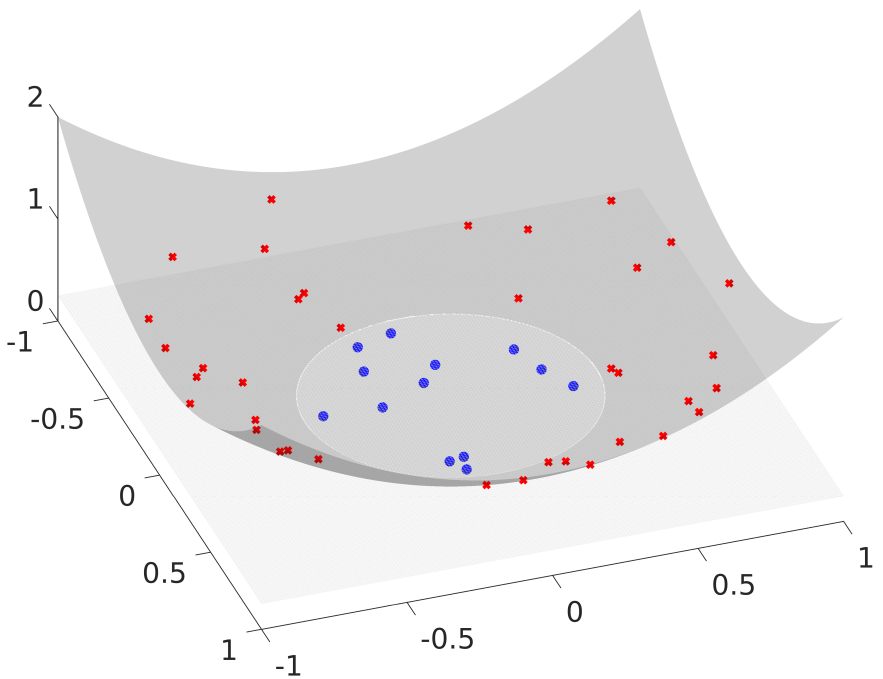
Figure: Separate the two classes.

Class Separation via a Linear Kernel

It is not possible to separate the points via a straight line in $\mathbb{R}^2$.

However, it is possible to separate them in $\mathbb{R}^3$ by considering 3-D feature vectors $\boldsymbol{z} = [z_1, z_2, z_3]^\top = [x_1, x_2, x_1^2 + x_2^2]^\top$.

For any $\boldsymbol{x} \in \mathbb{R}^2$, the corresponding feature vector $\boldsymbol{z}$ lies on a quadratic surface. In this space it is possible to separate the $\{\boldsymbol{z}_i\}$ points into two groups by means of a planar surface.



Linear Kernel

The separating plane in $\mathbb{R}^3$, using the transformed features, can be found by solving the quadratic optimization problem (6) with a linear kernel (and $C = \infty$):

svmquad.py

```
import numpy as np
from numpy import genfromtxt
from sklearn.svm import SVC

data = genfromtxt('svmcirc.csv', delimiter=',')
x = data[:,[0,1]] #vectors are rows
y = data[:,[2]].reshape(len(x),) #labels

tmp = np.sum(np.power(x,2),axis=1).reshape(len(x),1)
z = np.hstack((x,tmp))

clf = SVC(C = np.inf, kernel='linear')
clf.fit(z,y)

print("Support Vectors \n", clf.support_vectors_)
print("Support Vector Labels ",y[clf.support_])
print("Nu",clf.dual_coef_)
print("Bias",clf.intercept_)
```

Linear Kernel

This leads to the following parameters for the SVM:

$\boldsymbol{z}^\top$			y	$\alpha = \nu y$
0.0388	0.5380	0.2909	−1	−46.4925
−0.4912	−0.2056	0.2835	−1	−249.0181
−0.4507	−0.0480	0.2054	1	265.3181
−0.0611	−0.4165	0.1772	1	30.1925

It follows that the normal vector of the plane is

$$\boldsymbol{\beta} = \sum_{i \in \mathcal{S}} \alpha_i \boldsymbol{z}_i = [-0.9128, 0.8917, -24.2764]^\top,$$

where $\mathcal{S}$ is the set of indices of the support vectors.

The bias term β_0 can also be found from the table above. In particular, for any $\boldsymbol{x}^\top$ and y in the table, we have $y - \boldsymbol{\beta}^\top \boldsymbol{z} = \beta_0 = 5.6179$.

Separating Boundary

To draw the separating boundary in $\mathbb{R}^2$ we need to project the intersection of the separating plane with the quadratic surface onto the z_1, z_2 plane.

That is, we need to find all points (z_1, z_2) such that

$$5.6179 - 0.9128z_1 + 0.8917z_2 = 24.2764\,(z_1^2 + z_2^2).$$

This is the equation of a circle with (approximate) center $(0.019, -0.018)$ and radius 0.48, which is very close to the true circular boundary between the two groups, with center $(0, 0)$ and radius 0.5.

Separating Boundary

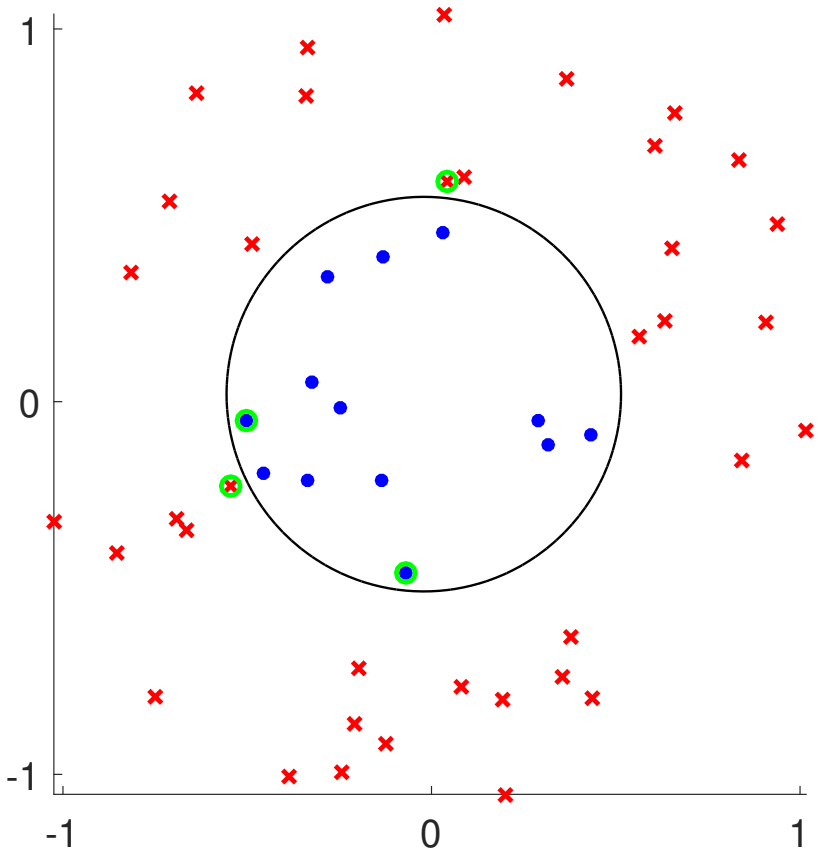


Figure: The circular decision boundary can be viewed equivalently as (a) the projection onto the x_1, x_2 plane of the intersection of the separating plane with the quadratic surface (both in $\mathbb{R}^3$), or (b) the set of points $\boldsymbol{x} = (x_1, x_2)$ for which $g_\tau(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{\phi}(\boldsymbol{x}) = 0$.

Alternative Kernel

An equivalent way to derive this circular separating boundary is to consider the feature map $\boldsymbol{\phi}(\boldsymbol{x}) = [x_1, x_2, x_1^2 + x_2^2]^\top$ on $\mathbb{R}^2$, which defines a reproducing kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}'),$$

on $\mathbb{R}^2$, which in turn gives rise to a (unique) RKHS $\mathcal{H}$. The optimal prediction function (2) is now of the form

$$g_\tau(\boldsymbol{x}) = \alpha_0 + \frac{1}{\gamma} \sum_{i=1}^{n} y_i \, \lambda_i \, \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\phi}(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{\phi}(\boldsymbol{x}), \tag{8}$$

where $\alpha_0 = \beta_0$ and

$$\boldsymbol{\beta} = \frac{1}{\gamma} \sum_{i=1}^{n} y_i \, \lambda_i \, \boldsymbol{\phi}(\boldsymbol{x}_i).$$

The decision boundary, $\{\boldsymbol{x} : g_\tau(\boldsymbol{x}) = 0\}$, is again a circle in $\mathbb{R}^2$.

svmkern.py

```
import numpy as np, matplotlib.pyplot as plt
from numpy import genfromtxt
from sklearn.svm import SVC

def mykernel(U,V):
    tmpU = np.sum(np.power(U,2),axis=1).reshape(len(U),1)
    U = np.hstack((U,tmpU))
    tmpV = np.sum(np.power(V,2),axis=1).reshape(len(V),1)
    V = np.hstack((V,tmpV))
    K = U @ V.T
    print(K.shape)
    return K

inp = genfromtxt('svmcirc.csv', delimiter=',') # read in the data
data = inp[:,[0,1]] #vectors are rows
y = inp[:,[2]].reshape(len(data),) #labels

clf = SVC(C = np.inf, kernel=mykernel, gamma='auto') # custom kernel
# clf = SVC(C = np.inf, kernel="rbf", gamma='scale') # inbuilt
clf.fit(data,y)

# output omitted (same as for the linear kernel code)
# plotting code omitted
```

Gaussian Kernel

Finally, we illustrate the use of the Gaussian kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{e}^{-c\,\|\boldsymbol{x}-\boldsymbol{x}'\|^2}, \tag{9}$$

where $c > 0$ is some tuning constant. This is an example of a radial basis function kernel, which are reproducing kernels of the form $\kappa(\boldsymbol{x}, \boldsymbol{x}') = f(\|\boldsymbol{x} - \boldsymbol{x}'\|)$, for some positive real-valued function f.

Each feature vector $\boldsymbol{x}$ is now transformed to a *function* $\kappa_{\boldsymbol{x}} = \kappa(\boldsymbol{x}, \cdot)$.

We can think of it as the (unnormalized) pdf of a Gaussian distribution centered around $\boldsymbol{x}$, and g_τ is a (signed) *mixture* of these pdfs, plus a constant; that is,

$$g_\tau(\boldsymbol{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \, \mathrm{e}^{-c\,\|\boldsymbol{x}_i-\boldsymbol{x}\|^2}.$$

Gaussian Kernel

Replacing in Line 2 of the previous code `mykernel` with `'rbf'` produces the SVM parameters.

The decision boundary is close to the true (circular) boundary $\{\boldsymbol{x} : \|\boldsymbol{x}\| = 1/2\}$. There are now seven support vectors.

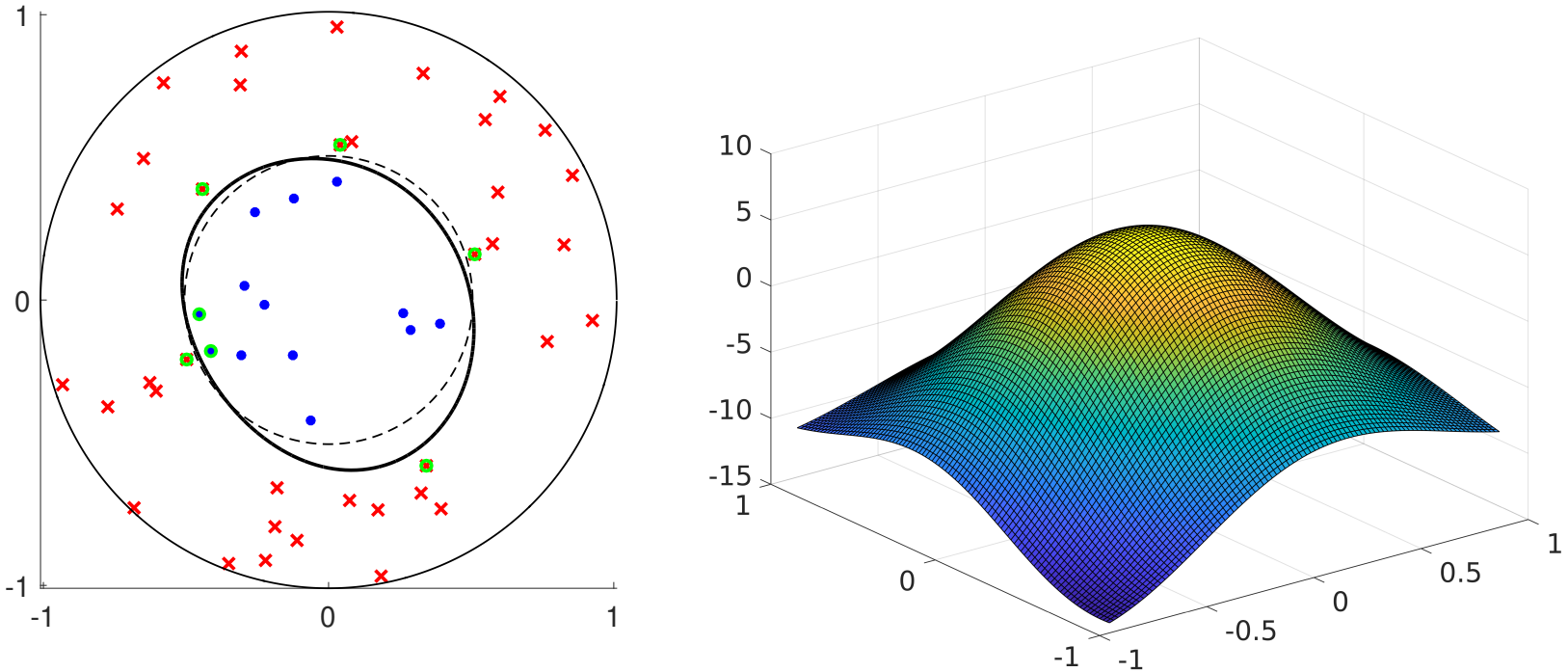


Figure: Left: Decision boundary $\{\boldsymbol{x} : g_\tau(\boldsymbol{x}) = 0\}$. Right: The graph of g_τ.

Classification with Scikit-Learn

Using the Python module **`sklearn`**, we illustrate the implementation of several classification methods applied to data from obtained from UCI's Breast Cancer Wisconsin data set.

This data set contains the measurements related to 569 images of 357 benign and 212 malignant breast masses.

The goal is to classify a breast mass as benign or malignant based on 10 features: Radius, Texture, Perimeter, Area, Smoothness, Compactness, Concavity, Concave Points, Symmetry, and Fractal Dimension of each mass.

The mean, standard error, and "worst" of these attributes were computed for each image, resulting in 30 features.

Classification with Scikit-Learn

The following Python code reads the data, extracts the response vector and model (feature) matrix and divides the data into a training and test set.

skclass1.py

```
from numpy import genfromtxt
from sklearn.model_selection import train_test_split
url1 = "http://mlr.cs.umass.edu/ml/machine-learning-databases/"
url2 = "breast-cancer-wisconsin/"
name = "wdbc.data"
data = genfromtxt(url1 + url2 + name, delimiter=',', dtype=str)
y = data[:,1] #responses
X = data[:,2:].astype('float') #features as an ndarray matrix

X_train , X_test , y_train , y_test = train_test_split(
        X, y, test_size = 0.4, random_state = 1234)
```

Visualization

The following code creates 3D scatterplot for the features mean *radius*, mean *texture*, and mean *concavity* (columns 0, 1, and 6 of the model matrix **X**).

skclass2.py

```
from skclass1 import X, y
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

Bidx = np.where(y == 'B')
Midx= np.where(y == 'M')
fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.scatter(X[Bidx,0], X[Bidx,1], X[Bidx,6],
           c='r', marker='^', label='Benign')
ax.scatter(X[Midx,0], X[Midx,1], X[Midx,6],
           c='b', marker='o', label='Malignant')
ax.legend()
ax.set_xlabel('Mean Radius')
ax.set_ylabel('Mean Texture')
ax.set_zlabel('Mean Concavity')
```

Visualization

The figure suggests that the malignant and benign breast masses could be well separated using these three features.

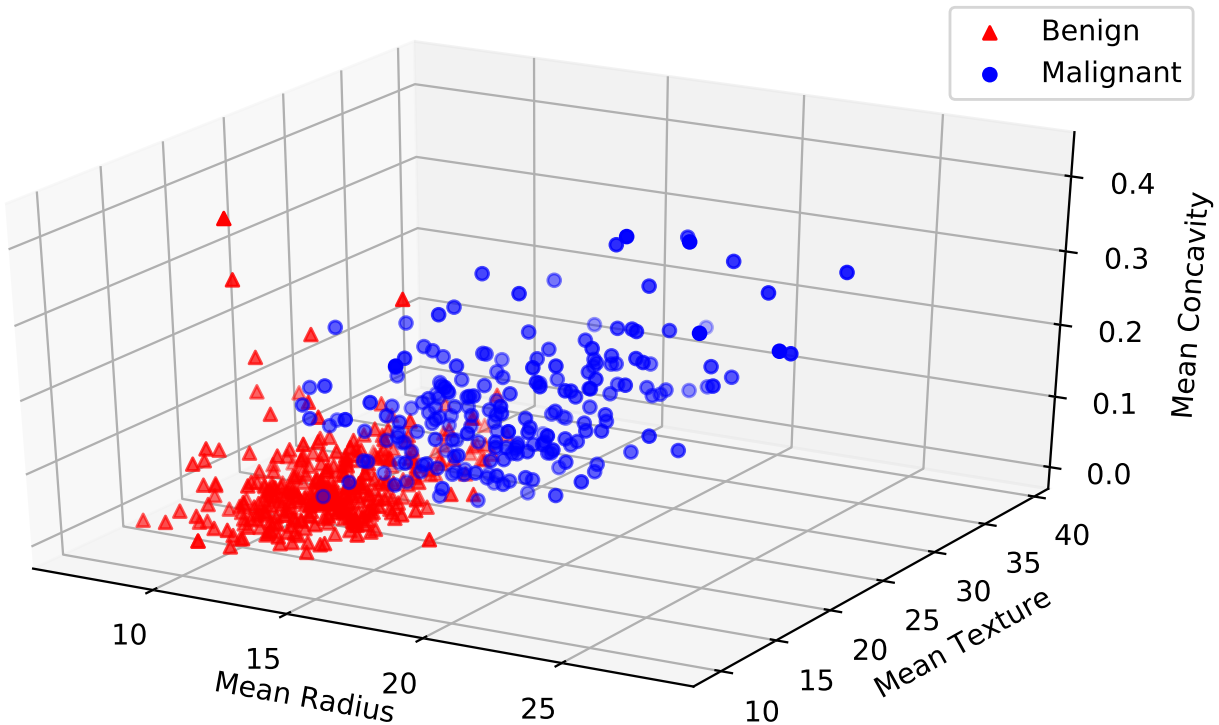


Figure: Scatterplot of three features of the benign and malignant breast masses.

Classification

skclass3.py

```
from skclass1 import X_train, y_train, X_test, y_test
from sklearn.metrics import accuracy_score

import sklearn.discriminant_analysis as DA
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

names = ["Logit","NBayes", "LDA", "QDA", "KNN", "SVM"]

classifiers = [LogisticRegression(C=1e5),
               GaussianNB(),
               DA.LinearDiscriminantAnalysis(),
               DA.QuadraticDiscriminantAnalysis(),
               KNeighborsClassifier(n_neighbors=5),
               SVC(kernel='rbf', gamma = 1e-4)]
```

```
print('Name  Accuracy\n'+14*'-')
for name, clf in zip(names, classifiers):
  clf.fit(X_train, y_train)
  y_pred = clf.predict(X_test)
  print('{:6}  {:3.3f}'.format(name, accuracy_score(y_test,y_pred)))
```

```
Name  Accuracy
--------------
Logit   0.943
NBayes  0.908
LDA     0.943
QDA     0.956
KNN     0.925
SVM     0.939
```

The training and test sets had 341 and 228 elements, respectively.

For each classifier the percentage of correct predictions (that is, the accuracy) in the test set is reported.

We see that in this case quadratic discriminant analysis gives the highest accuracy (0.956).