Support Vector Machines

Edgar Acuna
Department of Mathematical Science
University of Puerto Rico at Mayaguez

January 2022

Introduction

Linear SVM was introduced by V. Vapnik en 1979 (before at ATT, now at Facebook). Non-linear SVM were introduced by Boser, Guyon and Vapnik in 1992 and become very popular by the end of the 1990's.

SVM is one of the best classifier for bioinformatics data and textual data.

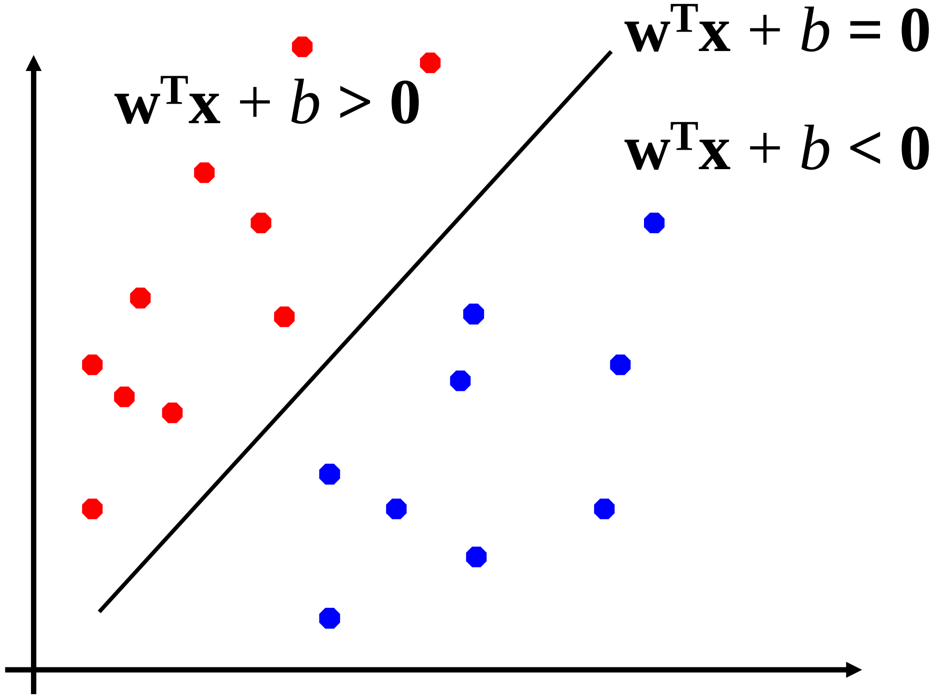
SVM's can be applied to a variety of complex data choosing very convenient kernel functions.

Application of SVM has been extended to other machine learning tasks such as regression [Vapnik *et al.* '97], principal component analysis [Schölkopf *et al.* '99], Outlier detection[Tax, 2001], etc.

The tuning of SVMs such as kernel selection and parameters models is done heuristically.

Linear Separators

- Binary classification can be seen as the task to separate classes in the attribute space.


$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$
$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b > 0$
$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b < 0$

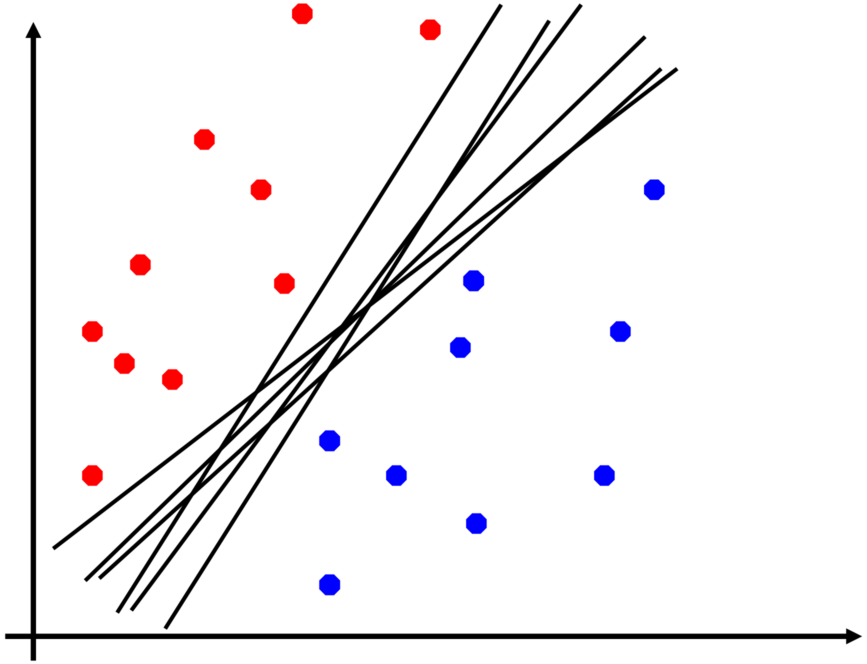
$a_1x_1+a_2x_2=b$ Linear equation

$\mathbf{x}=(x_1,x_2)$ $\mathbf{w}=(a_1,a_2)$

$$f(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$$

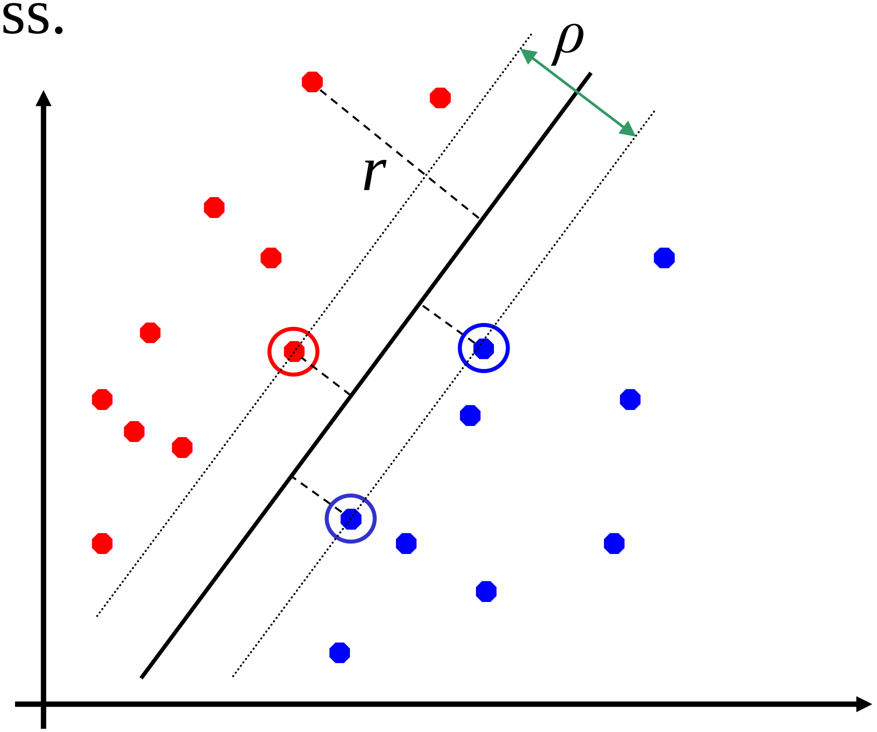
Linear Separators

Which of these linear separators is optimal ?



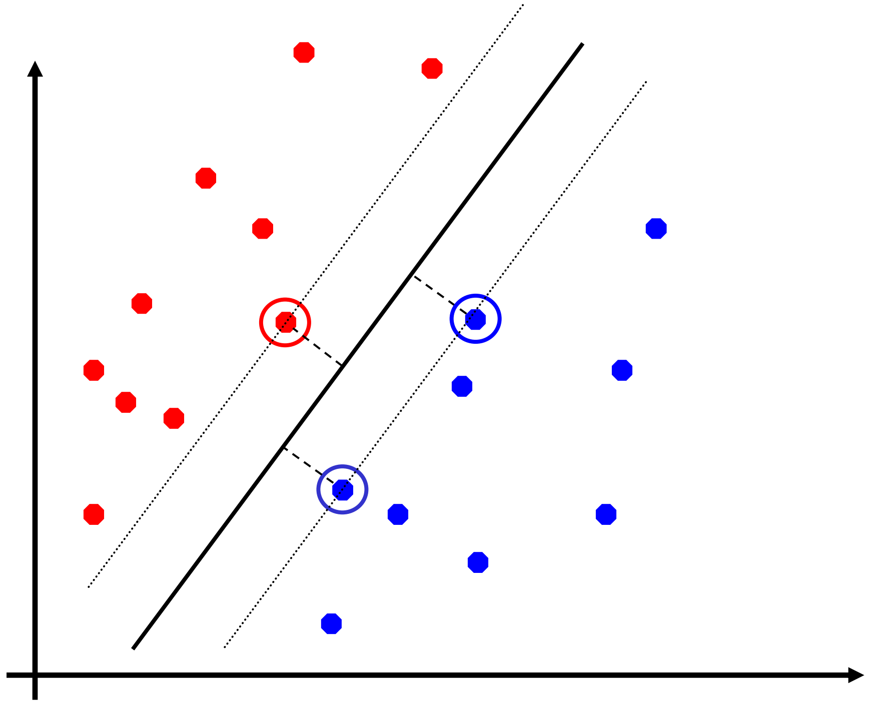
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are called ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors from oppositive class.



Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC (Probably approximately correct) theory.
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbf{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example $(\mathbf{x}_i, y_i)$:

$$\begin{array}{ll} \mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b \leq -\rho/2 & \text{if } y_i = -1 \\ \mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b \geq \rho/2 & \text{if } y_i = 1 \end{array} \quad \Leftrightarrow \quad y_i(\mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b) \geq \rho/2$$

- For every support vector $\mathbf{x}_s$ the above inequality is an equality. After rescaling $\mathbf{w}$ and b by $\rho/2$ in the equality, we obtain that distance between each $\mathbf{x}_s$ and the hyperplane is $r = \frac{\mathrm{y}_s(\mathbf{w}^T\mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$

- Then the margin can be expressed through (rescaled) **w** and b as:

$$\rho = 2r = \frac{2}{\|\mathbf{w}\|}$$

Linear SVMs Mathematically (cont.)

- Then, we can formulate the *quadratic optimization problem:*

Find $\mathbf{w}$ and b such that

$$\rho = \frac{2}{\|\mathbf{w}\|} \text{ is maximized}$$

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find $\mathbf{w}$ and b such that

$\mathbf{\Phi}(\mathbf{w}) = ||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$ is minimized

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i\,(\mathbf{w}^\mathbf{T}\mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathbf{T}\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- Given a solution $\alpha_1 \ldots \alpha_n$ to the dual problem, solution to the primal is:

$$\boxed{\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x}_i \qquad b = y_k - \Sigma\alpha_i y_i \mathbf{x}_i{}^{\mathbf{T}}\mathbf{x}_k \quad \text{for any } \alpha_k > 0}$$

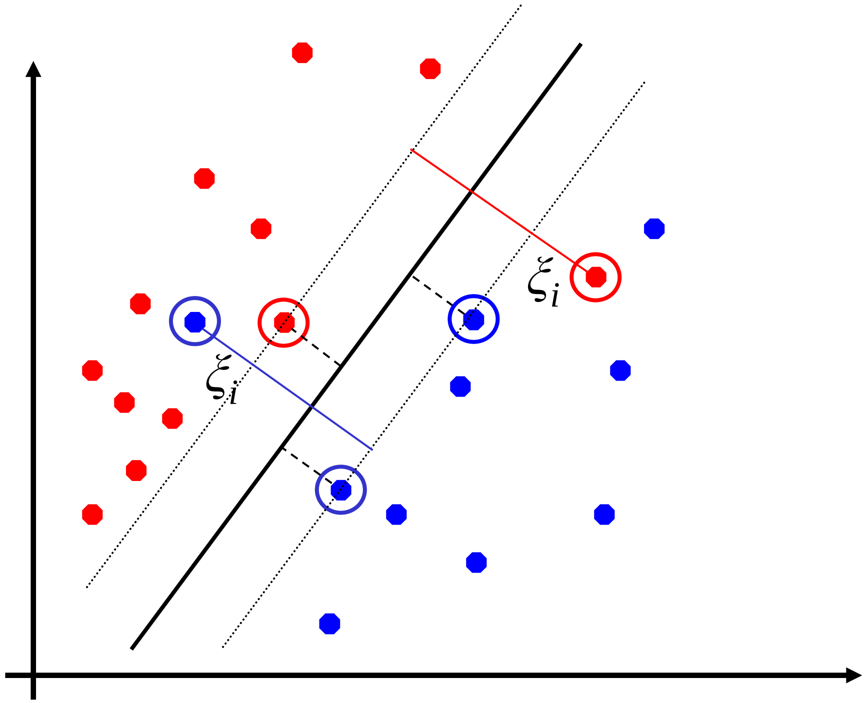
- Each non-zero α_i indicates that corresponding $\mathbf{x}_{\boldsymbol{i}}$ is a support vector.
- Then the classifying function is:

$$\boxed{f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i{}^{\mathbf{T}}\mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x}_i$
- Also, solving the optimization problem involved computing the inner products $\mathbf{x}_i{}^{\mathbf{T}}\mathbf{x}_j$ between all training points.

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Soft Margin Classification Mathematically

- The old formulation:

Find **w** and b such that

$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized

and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i (\mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find **w** and b such that

$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w} + C\Sigma\xi_i$ is minimized

and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i (\mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b) \geq 1 - \xi_{i,}$, $\xi_i \geq 0$

- Parameter *C* can be viewed as a way to control overfitting: it “trades off” the relative importance of maximizing the margin and fitting the training data.

Soft Margin Classification – Solution

- Dual problem is identical to separable case (would *not* be identical if the 2-norm penalty for slack variables $C\Sigma\xi_i^2$ was used in primal objective, we would need additional Lagrange multipliers for slack variables):

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

- Again, $\mathbf{x}_i$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x}_i$

$b = y_k(1 - \xi_k) - \Sigma\alpha_i y_i \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_k$ for any k s.t. $\alpha_k > 0$

Again, we don't need to compute **w** explicitly for classification:

$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^{\mathrm{T}}\mathbf{x} + b$

Linear SVMs: Overview

- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Using a technique called Platt scaling outputs from SVM can be mapped to posterior probabilities.

- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.

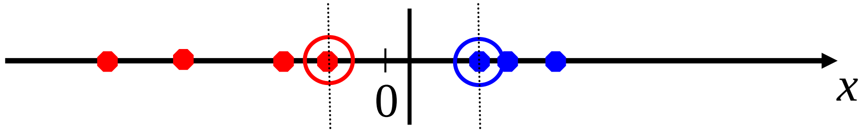
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \boxed{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

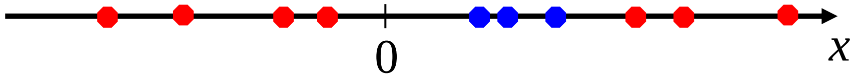
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \boxed{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}} + b$$

Non-linear SVMs

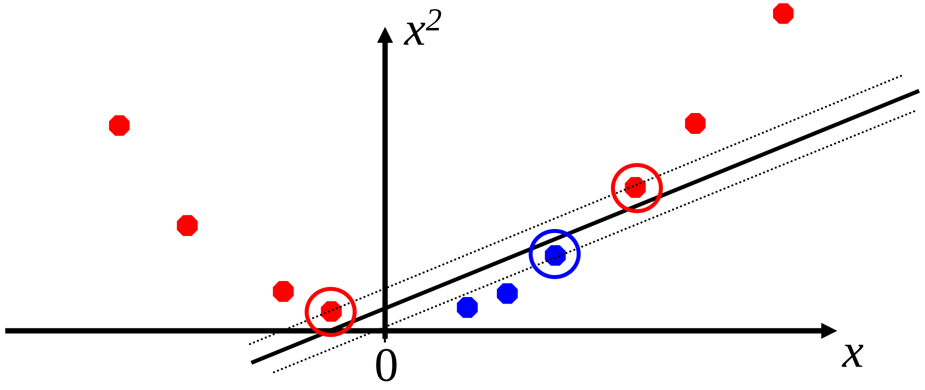
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

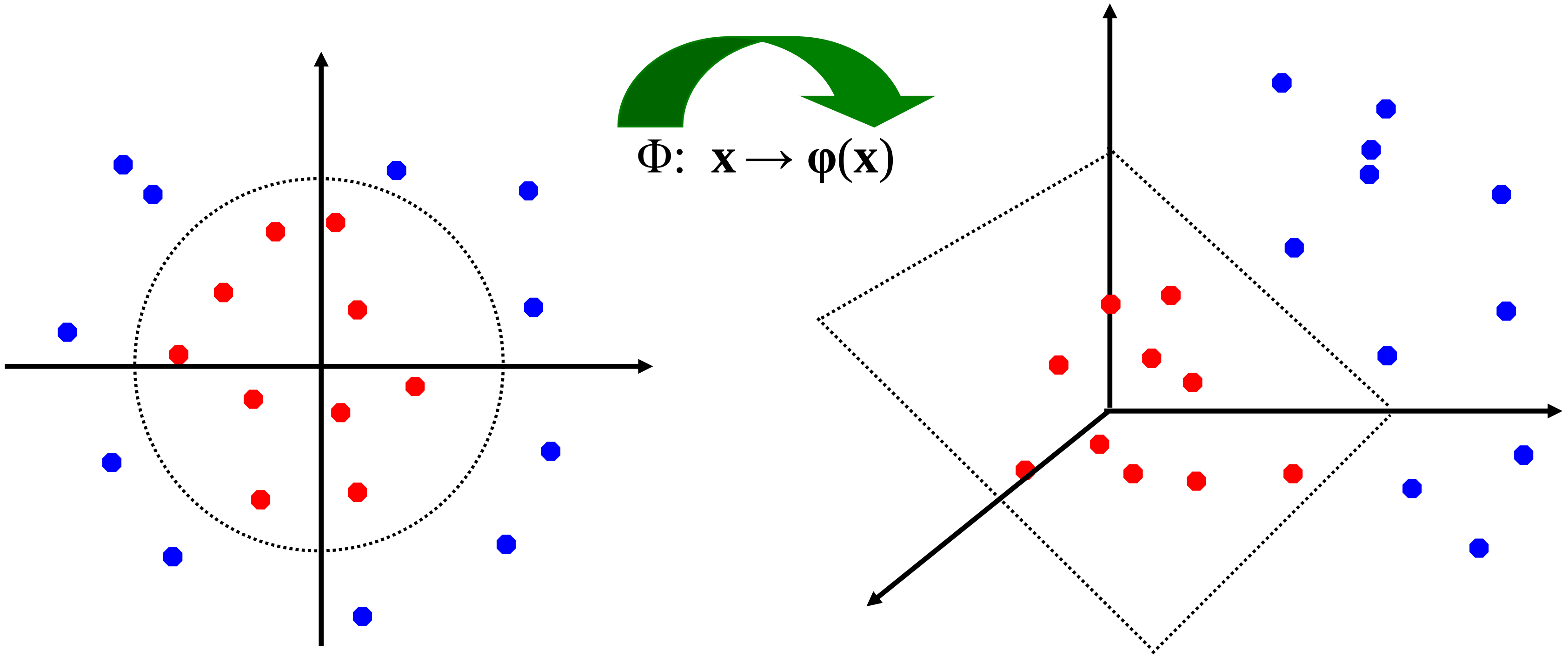


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



$\Phi:\ \mathbf{x} \rightarrow \varphi(\mathbf{x})$

The "Kernel Trick"

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\varphi}(\mathbf{x}_j)$$

- A *kernel function* is a function that is equivalent to an inner product in some feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1 \;\; x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_\mathbf{j})= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\varphi}(\mathbf{x}_j)$:

$$K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^2_{,}= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}=$$

$$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^{\mathbf{T}}\, [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}] =$$

$$= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\varphi}(\mathbf{x}_j), \quad \text{where } \boldsymbol{\varphi}(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$$

- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\varphi}(\mathbf{x})$ explicitly).

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i,\mathbf{x}_j)$ checking that $K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^{\mathbf{T}}\varphi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_n)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_n)$
…	…	…	…	…
$K(\mathbf{x}_n,\mathbf{x}_1)$	$K(\mathbf{x}_n,\mathbf{x}_2)$	$K(\mathbf{x}_n,\mathbf{x}_3)$	…	$K(\mathbf{x}_n,\mathbf{x}_n)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is $\mathbf{x}$ itself

- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^p$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ has dimensions $\binom{d+p}{p}$

- Gaussian (radial-basis function): $K(\mathbf{x}_i,\mathbf{x}_j) = e^{-\frac{\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2}{2\sigma^2}}$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is *infinite-dimensional*: every point is mapped to *a function* (a Gaussian); combination of functions for support vectors is the separator.

- Higher-dimensional space still has *intrinsic* dimensionality d (the mapping is not *onto*), but linear separators in it correspond to *non-linear* separators in original space.

Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \ldots \alpha_n$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(\mathbf{x}) = \Sigma\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b$$

- Optimization techniques for finding α_i's remain the same!

SVM in Python

The library scikit-learn includes two functions to perform SVM. The first one is LinearSVM for performing SVM using linear Kernel, and the second one is SVM for performing SVM with any other kernel.

Like in R, these functions are a wrapper for *LIBSVM: a C library for Support Vector Machines (2011),* Chang, C. C. and Lin, C.J.

In order to apply SVM classifier it is desirable to work with normalized predictors.

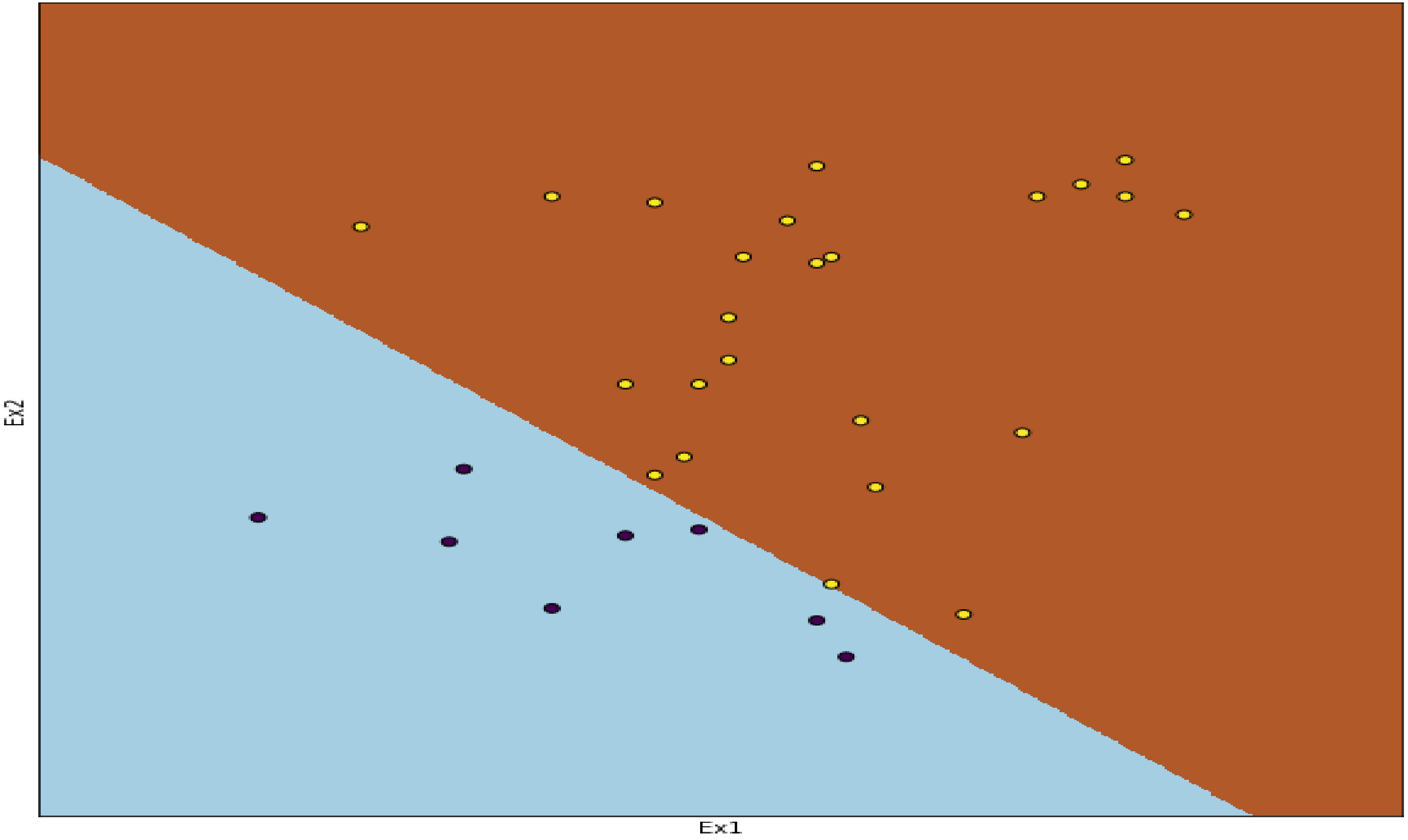
```
#Example 1. SVM applied to the prediction of  final grade in a class based on exams E1 and E2
df=pd.read_csv("http://academic.uprm.edu/eacuna/eje1dis.csv")
#Normalizing the predictor matrix
y=df['Nota']
X=df.iloc[:,0:2]
X1=X.as_matrix()
scaler = StandardScaler()
```

SVM in Python

```
scaler.fit(X1)
X1= scaler.transform(X1)
#Applying a linear SVM classifier and calculating the percentage of accuracy
lin_clf = svm.LinearSVC()
lin_clf.fit(X1, y)
#The confusion matrix
pred=lin_clf.predict(X1)
print(confusion_matrix(y,pred))
[[ 8 0]
 [ 0 24]]
```

The accuracy estimated by resubstitution is 100%

Visualizing the decision boundary of the Linear SVM

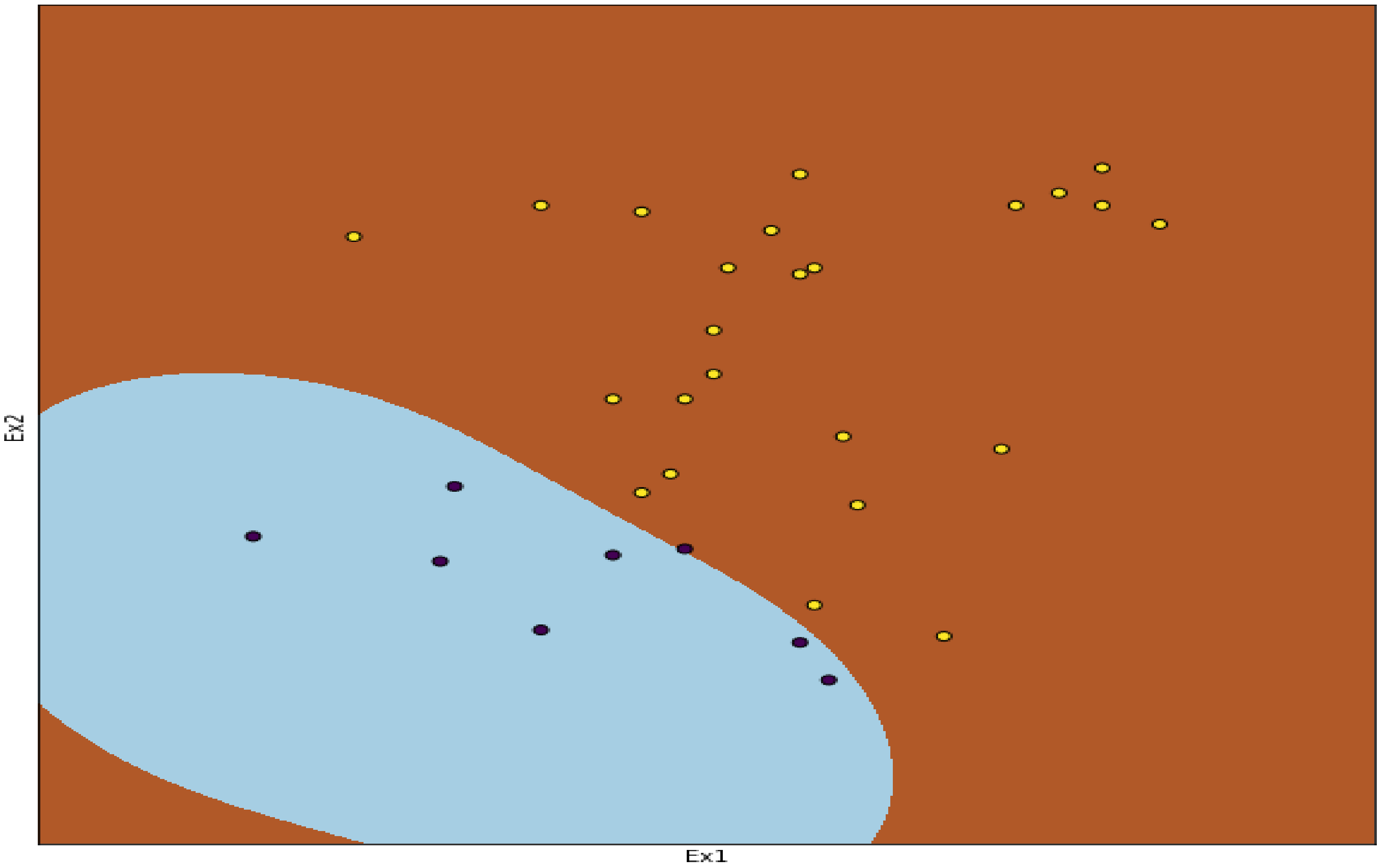


SVM in Python

```
#Applying a nonlinear SVM classifier and calculating the percentage of accuracy
clf = svm.SVC()
clf.fit(X1, y)
#The support vectors
clf.support_vectors_
clf.n_support_
#The confusion matrix
pred=clf.predict(X1)
print(confusion_matrix(y,pred))
[[ 7 1]
 [ 0 24]]
```

The accuracy estimated by resubstitution is 96.9%

Visualizing the decision boundary of the nonlinear SVM



Nonlinear SVM for the Diabetes dataset

```
#Applying a nonlinear SVM classifier and calculating the percentage of accuracy
url= "http://academic.uprm.edu/eacuna/diabetes.dat"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_table(url, names=names)
X=data.iloc[:,0:8]
lb_make = LabelEncoder()
data["class"] = lb_make.fit_transform(data["class"])
y2=data['class']
X1=X.as_matrix()
scaler = StandardScaler()
scaler.fit(X_train)
X_train= scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Nonlinear SVM for the Diabetes dataset

```
#Applying a nonlinear SVM classifier and calculating the percentage of accuracy
clf=svm.SVC()
clf.fit(X, Cy2)
clf.support_vectors_
clf.n_support_
#Accuracy using resubstitution
clf.score(X,y2)
1.0
#Accuracy using cross validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X1, y2, cv=10)
print 'The accuracy estimated by CV is:', scores.mean()
The accuracy estimated by CV is: 0.761688311688
```

Multiclass SVM

Let us suppose that the dataset has C classes, C>2.. SVMs are inherently two-class classifiers.

The traditional way to do multiclass classification with SVMs is to build C one-versus-rest classifiers (commonly referred to as ``one-versus-rest' or OVR classification), and to choose the class which classifies the test datum with greatest margin.

Another strategy is to build a set of one-versus-one classifiers (OVO), and to choose the class that is selected by the most classifiers. While this involves building C(C-1)/2 classifiers, the time for training classifiers may actually decrease, since the training data set for each classifier is much smaller.

SVC uses the ‘one-against-one’-approach.

On the other hand, LinearSVC implements “one-vs-the-rest” multi-class strategy.

```
#Loading the Landsat dataset
url='http://academic.uprm.edu/eacuna/landsat.txt'
data = pd.read_table(url, header=None,delim_whitespace=True)
```

Linear SVM for the Landsat dataset

```
y=data.iloc[:,36]
X=data.iloc[:,0:36]
X_train, X_test, y_train, y_test = train_test_split(X, y)
scaler = StandardScaler()
scaler.fit(X_train)
X_train= scaler.transform(X_train)
X_test = scaler.transform(X_test) #Accuracy estimation by the holdout method
X_train, X_test, y_train, y_test = train_test_split(X, y)
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
pred=lin_clf.predict(X_test)
print (pred==y_test).mean()
```

The accuracy is: 0.766456266907

SVM summary

- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- Sequential minimal optimization (SMO) (J. Platt, 1998) is the most used algorithm for solving the quadratic programming (QP) problem that arises in SVMs.
- Using a technique called Platt scaling outputs from SVM can be mapped to posterior probabilities.
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.
- The SVM time complexity is more than quadratic on the number of samples. This makes it hard to use in dataset with more than a couple of 10000 samples.