Classification: Support Vector Machines

BITS Pilani
Pilani Campus

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

innovate achieve lead

BITS Pilani
Pilani Campus

Session 8
Date – 27/02/2022
Time – 10 to 12.30

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Support Vector Machines in overlapping class distributions & Kernels
- Issues of overlapping class distribution for SVM
- Posing an optimization problem for SVM in overlapping class scenario
- Solving the optimization problem using Lagrange multipliers, dual representations
- Kernel Trick and Mercer's theorem
- Techniques for constructing Kernels and advantages of Kernels in SVM
- Implementation of SVM using different kernels

Solving the Optimization Problem

1. Maximize margin $2/||\mathbf{w}||$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

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

$\mathbf{\Phi}(\mathbf{w}) = \frac{\mathbf{1}}{\mathbf{2}}||\mathbf{w}||^2$ is minimized;

and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i (\mathbf{w^T x_i} + b) \geq 1$

Solving the Optimization Problem

- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

$$\boxed{L(w, b, \alpha_i) = \frac{1}{2}||\mathbf{w}||^2 - \Sigma\, \alpha_i\, [y_i\, (\mathbf{w}^T \mathbf{x_i} + b) - 1]}$$

- **Taking partial derivative with respect to w ,** $\frac{\partial L}{\partial w} = 0$
  - $w - \Sigma\, \alpha_i y_i\, \mathbf{x_i} = 0$
  - $w = \Sigma\, \alpha_i y_i\, \mathbf{x_i}$
- **Taking partial derivative with respect to b,** $\frac{\partial L}{\partial b} = 0$
  - $-\Sigma\, \alpha_i\, y_i = 0$
  - $\Sigma\, \alpha_i\, y_i = 0$

Solving the Optimization Problem

$$\boxed{L(w, b, \alpha_i) = \frac{1}{2}||\mathbf{w}||^2 - \Sigma\, \alpha_i\, [y_i\, (\mathbf{w}^T\mathbf{x_i} + b) - 1]}$$

- Expanding above equation:

$$L(w, b, \alpha_i) = \frac{1}{2}\, \mathbf{w}^T\mathbf{w} - \Sigma\, \alpha_i y_i\, \mathbf{w}^T\mathbf{x_i} - \Sigma\, \alpha_i y_i\, b \; + \Sigma\, \alpha_i$$

- Substituting w $= \Sigma\, \alpha_i y_i\, \mathbf{x_i}$ and $\Sigma\, \alpha_i\, y_i = 0$ in above equation

$$L(w, b, \alpha_i) = \frac{1}{2}\, (\textstyle\sum_i \alpha_i y_i\, \mathbf{x_i})(\sum_j \alpha_j y_\mathbf{j}\, \mathbf{x_j}) - \; (\sum_i \alpha_i y_i\, \mathbf{x_i})(\sum_j \alpha_j y_\mathbf{j}\, \mathbf{x_j}) + \Sigma\, \alpha_i$$

$$L(w, b, \alpha_i) = \Sigma\, \alpha_i - \frac{1}{2}\, (\textstyle\sum_i \alpha_i y_i\, \mathbf{x_i})(\sum_j \alpha_j y_\mathbf{j}\, \mathbf{x_j})$$

$$L(w, b, \alpha_i) = \Sigma\, \alpha_i - \frac{1}{2}\, (\textstyle\sum_i \sum_j \alpha_i\, \alpha_j y_i\, y_\mathbf{j}\, \mathbf{x_i} \cdot \mathbf{x_j})$$

The Dual Problem

- The new objective function is in terms of α_i only
- It is known as the dual problem: if we know **w**, we know all α_i; if we know all α_i, we know **w**
- The original problem is known as the primal problem
- The objective function of the dual problem needs to be maximized (comes out from the KKT theory)
- The dual problem is therefore:

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Properties of α_i when we introduce the the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

Optimization Problem

Find **w** and b such that

$\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\, \mathbf{w}^{T}\mathbf{w} = \frac{1}{2}||\mathbf{w}||^{2}$ is minimized;

and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i\,(\mathbf{w^T x_i} + b) \geq 1$

$L(w, b, \alpha_i) = \frac{1}{2}||\mathbf{w}||^{2} - \Sigma\, \alpha_i\,[y_i\,(\mathbf{w^T x_i} + b) - 1]$

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\left(\sum_i \sum_j \alpha_i\, \alpha_j\, y_i\, y_\mathbf{j}\, \mathbf{x_i} \cdot \mathbf{x_j}\right)$ is maximized and

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

(2) $\alpha_i \geq 0$ for all α_i

Support Vectors

Using KKT conditions :

$\alpha_i\,[y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1] = 0$

For this condition to be satisfied

either $\alpha_i = 0$ and $y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1 > 0$

OR

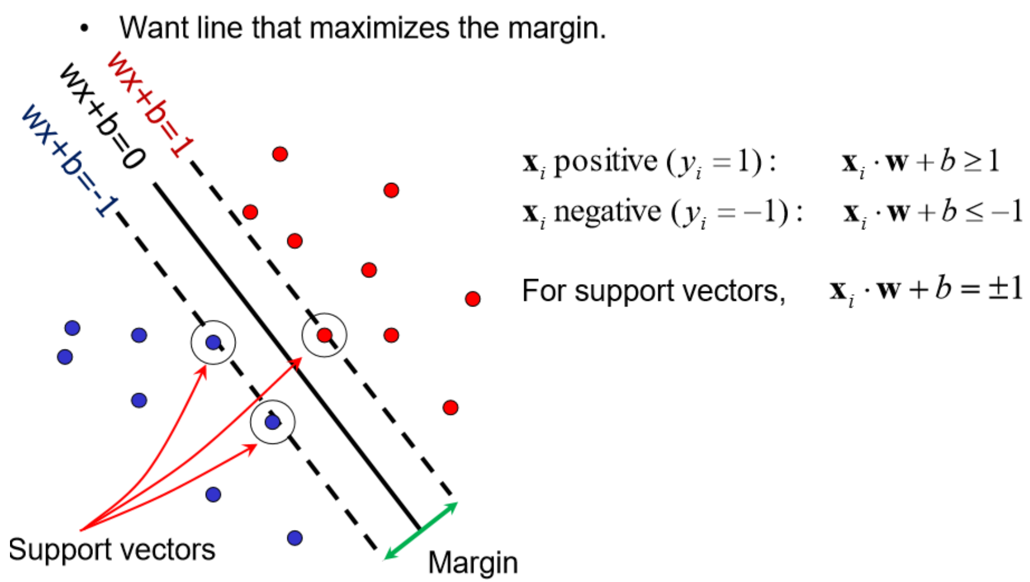
$y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1 = 0$ and $\alpha_i > 0$

For support vectors:

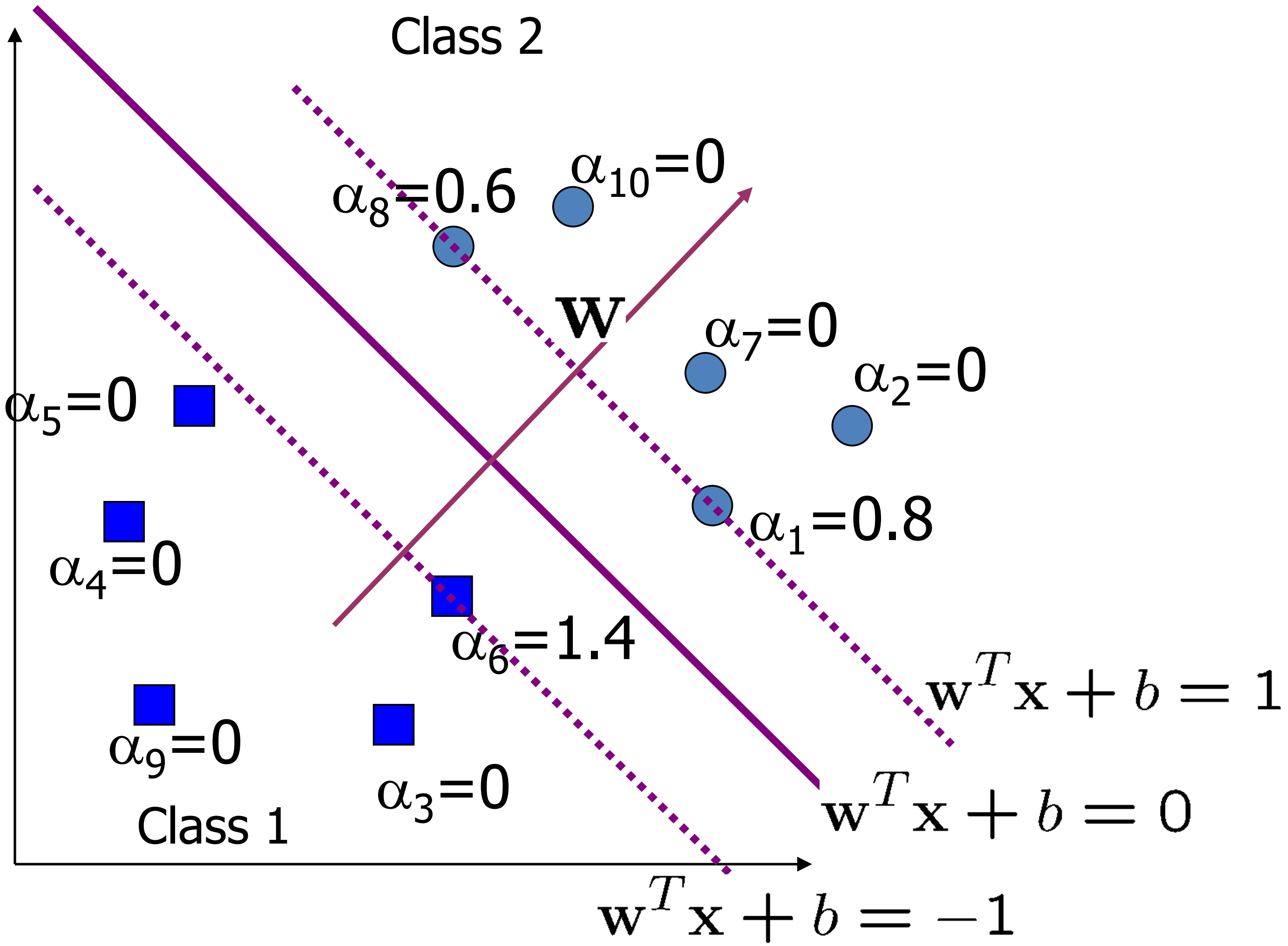
$y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1 = 0$

For all points other than support vectors:

$\alpha_i = 0$



A Geometrical Interpretation



Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

Learned weight

Support vector

Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

  $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)

- Classification function:

$$f(x) = \text{sign}\,(\mathbf{w} \cdot \mathbf{x} + \mathrm{b})$$
$$= \text{sign}\left(\sum_{\mathrm{i}} \alpha_{\mathrm{i}} y_i \boxed{\mathbf{x}_i \cdot \mathbf{x}} + b\right)$$

If f(x) < 0, classify as negative, otherwise classify as positive.

- Notice that it relies on an *inner product* between the test point $\boldsymbol{x}$ and the support vectors $\boldsymbol{x}_i$
- (Solving the optimization problem also involves computing the inner products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ between all pairs of training points)

Linear SVMs: Overview

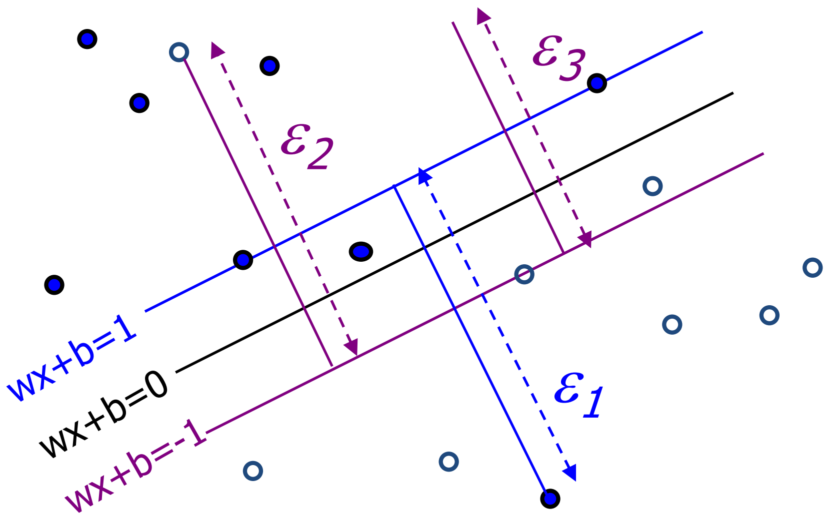
- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

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

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

Soft Margin Classification

Slack variables ξ_i **can be added to allow misclassification of difficult or noisy examples.**



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Soft Margin

Misclassification cost

\# data samples

Slack variable

$$\min_{\boldsymbol{w}} \; \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{Maximize margin}} + \underbrace{C\sum_{i=1}^{N}\xi_i}_{\text{Minimize misclassification}}$$

The *w* that minimizes…

$$\text{subject to} \quad y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i,$$

$$\xi_i \geq 0, \quad \forall i = 1, \ldots, N$$

Hard Margin versus Soft Margin

- **Hard Margin:**

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

$\boldsymbol{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i(\mathbf{w^T x_i} + \mathrm{b}) \geq 1$

- **Soft Margin incorporating slack variables:**

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

$\boldsymbol{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum \xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i(\mathbf{w^T x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

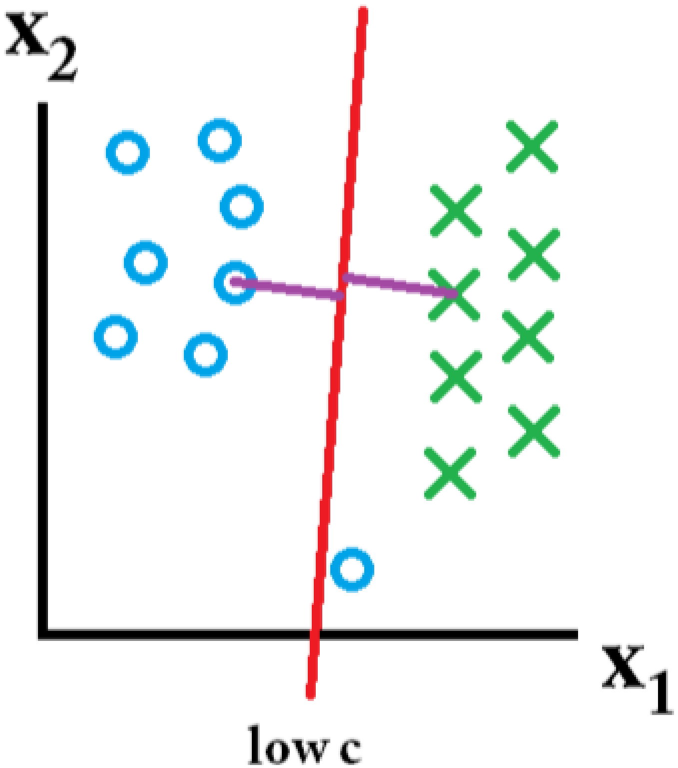
- **Parameter *C* can be viewed as a way to control overfitting.**

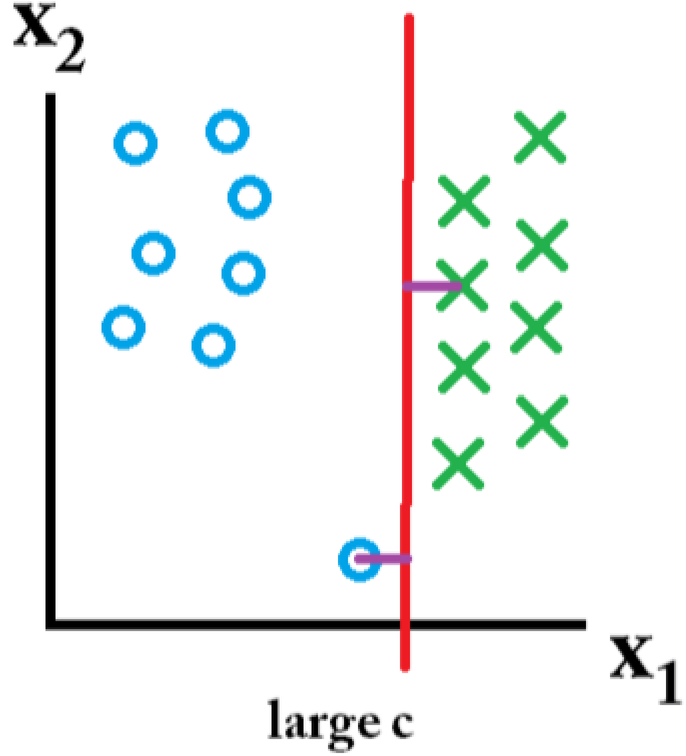
Value of C parameter

- C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Effect of Margin size v/s misclassification cost

Training set





overfitting

Misclassification ok, want large margin

Misclassification not ok

Linear SVMs: Overview

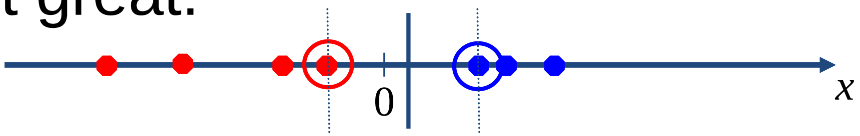
- **The classifier is a *separating hyperplane.***
- **Most "important" training points are support vectors; they define the hyperplane.**
- **Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.**
- **Both in the dual formulation of the problem and in the solution training points appear only inside dot products:**

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

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

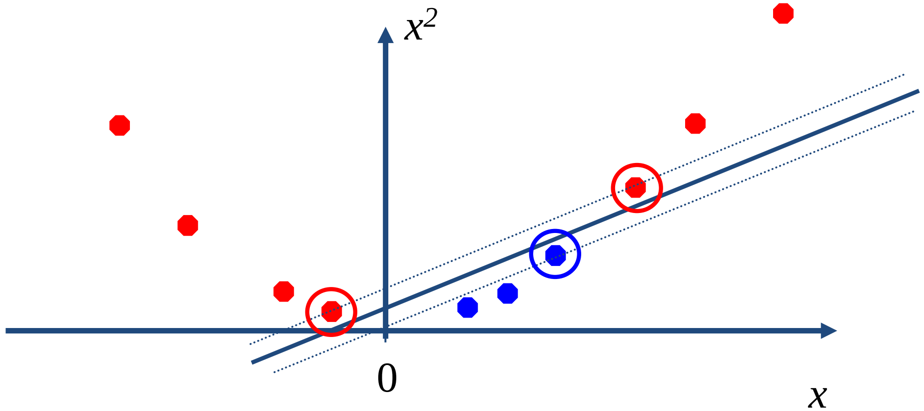
Non-linear SVMs

- Datasets that are linearly separable with some noise soft margin 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:



The "Kernel Trick"

- **The linear classifier relies on dot product between vectors**
  - $x_i^T . x_j$
- **If every data point is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the dot product becomes:**

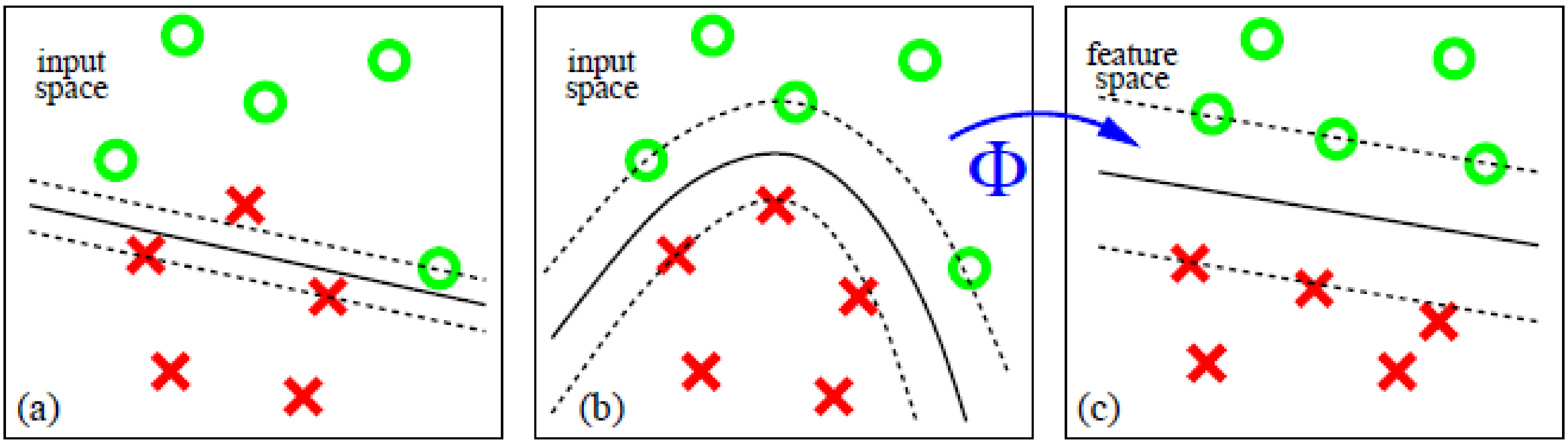
$$K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$$

- **A *kernel function* is some function that corresponds to an inner product in some expanded feature space.**

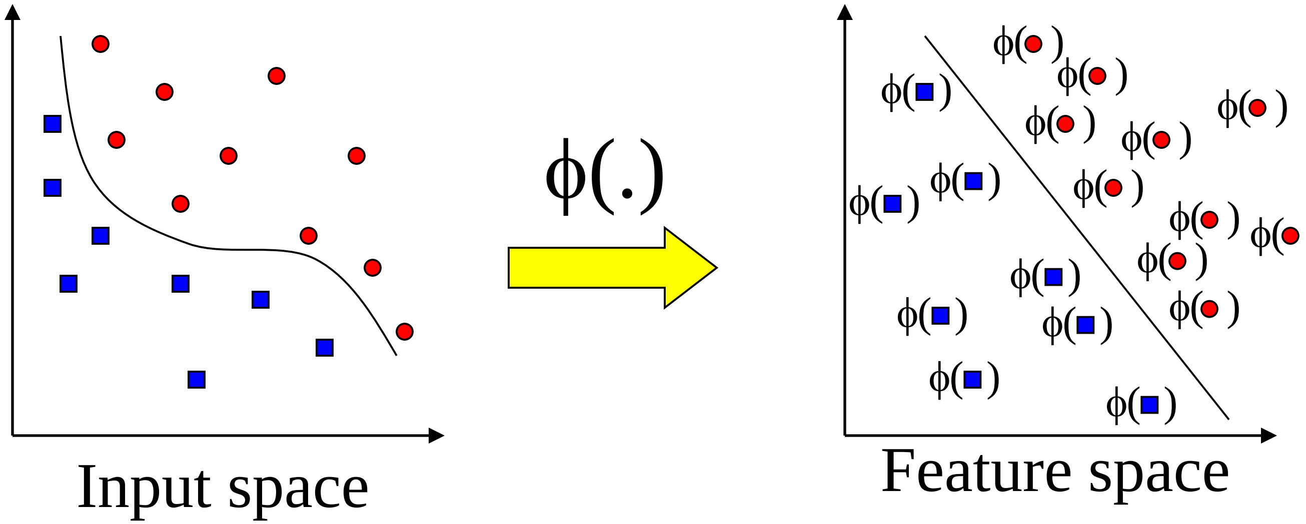
SVM Kernels

- SVM algorithms use a set of mathematical functions that are defined as the kernel.
- Function of kernel is to take data as input and transform it into the required form.
- Different SVM algorithms use different types of kernel functions. Example *linear, nonlinear, polynomial, and sigmoid etc.*

Find a feature space



Transforming the Data

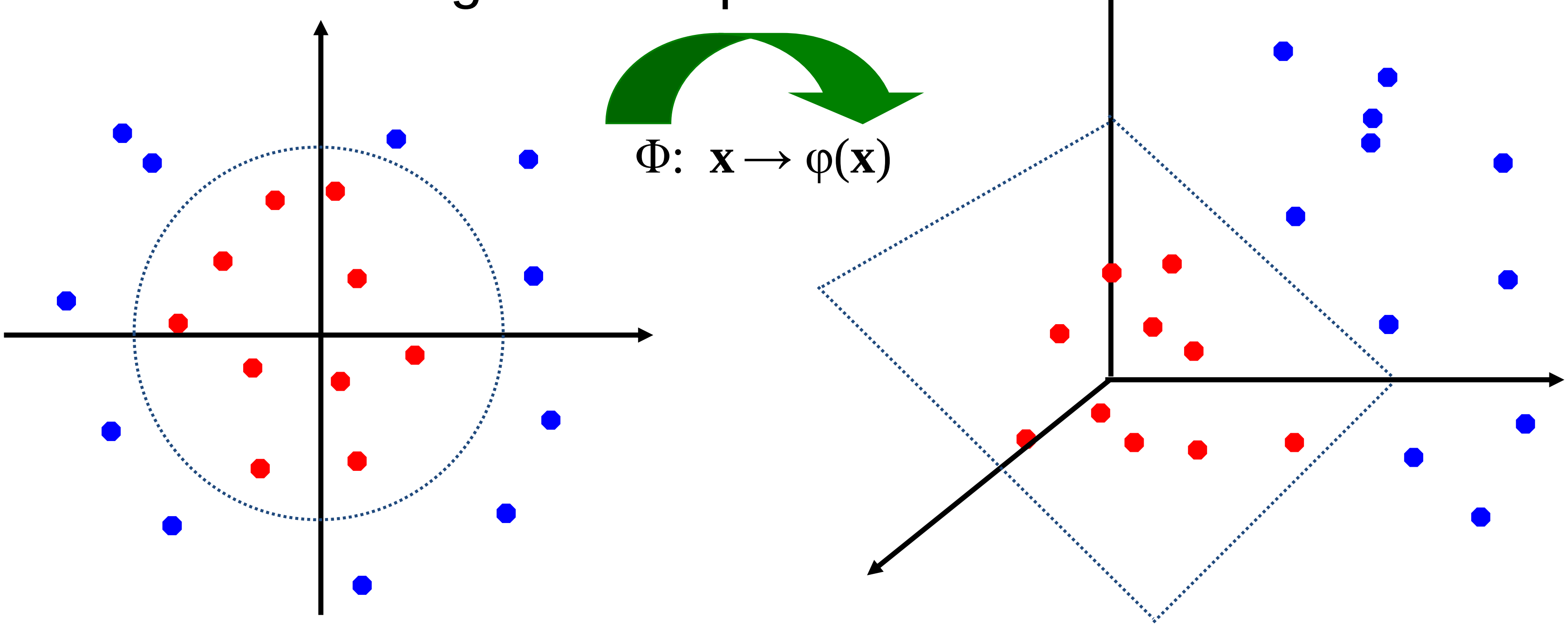


Note: feature space is of higher dimension than the input space in practice

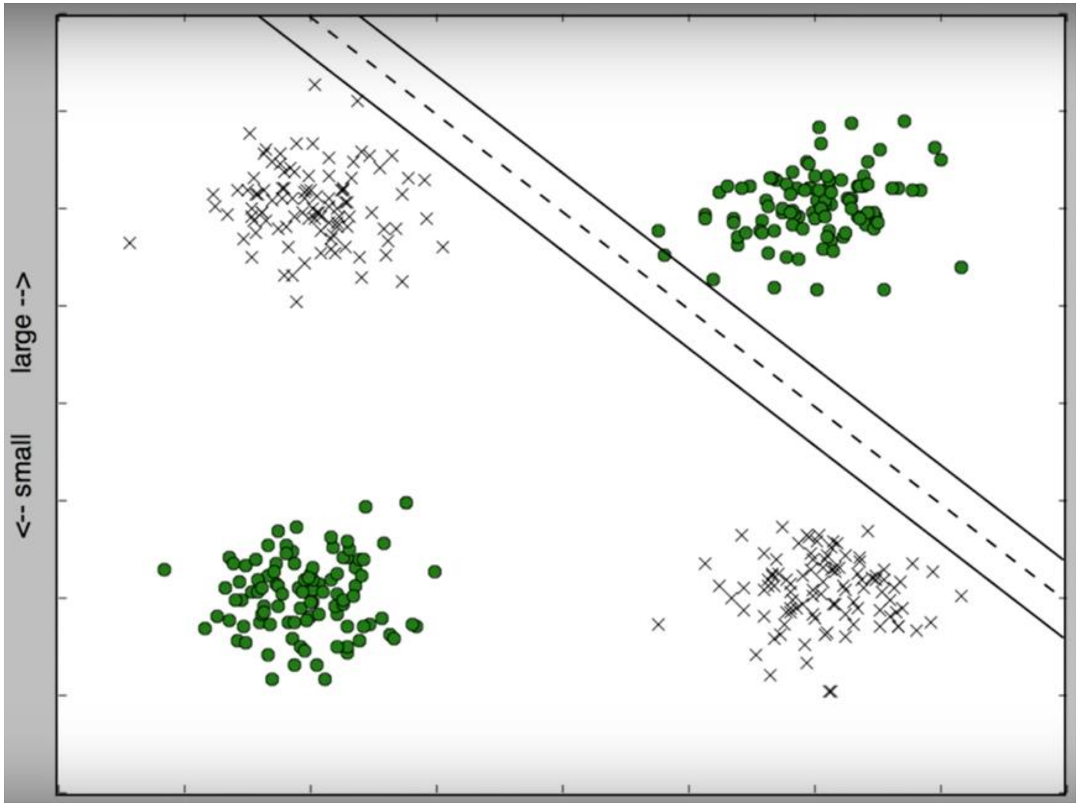
- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

Non-linear SVMs: Feature spaces

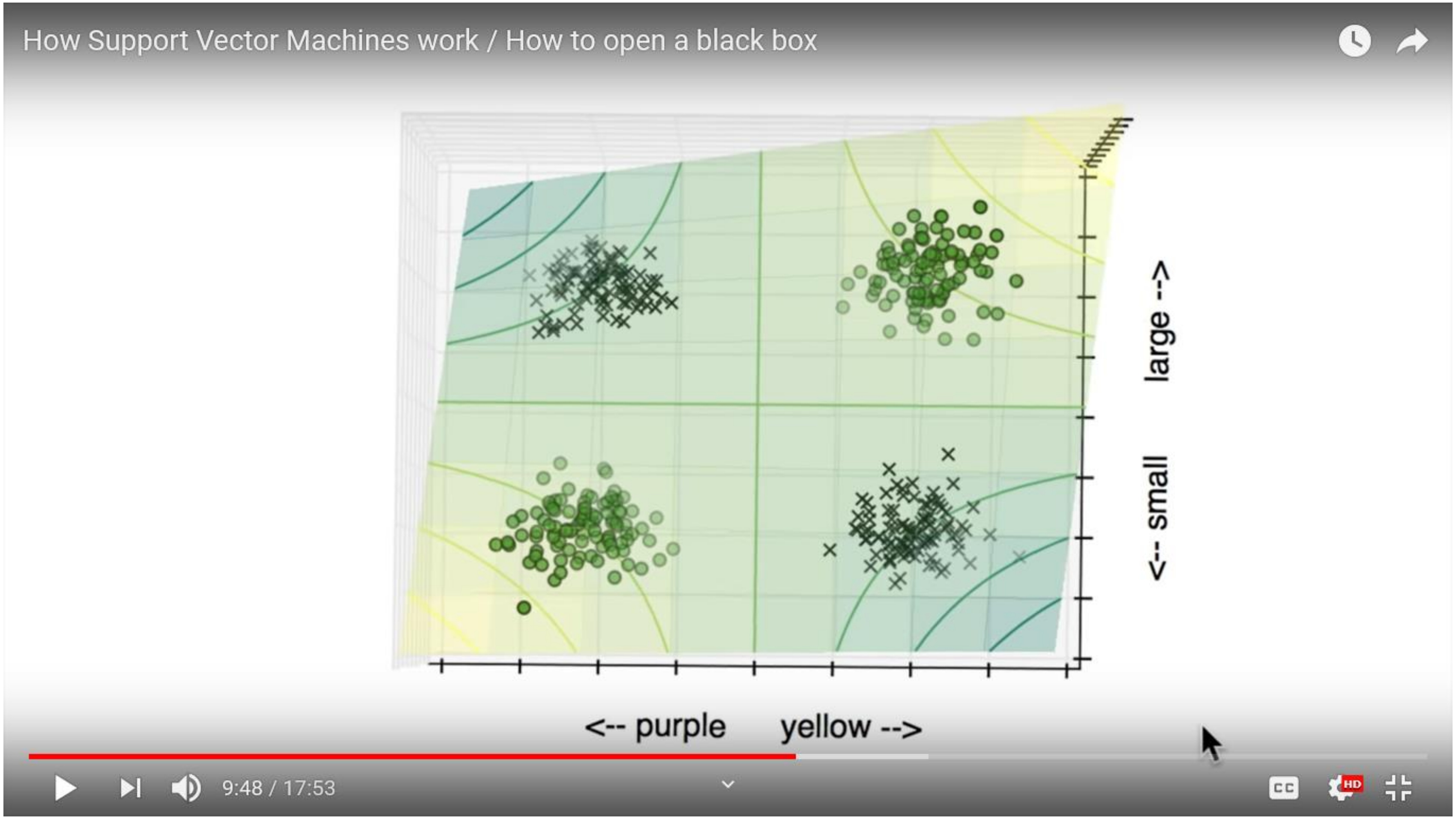
- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



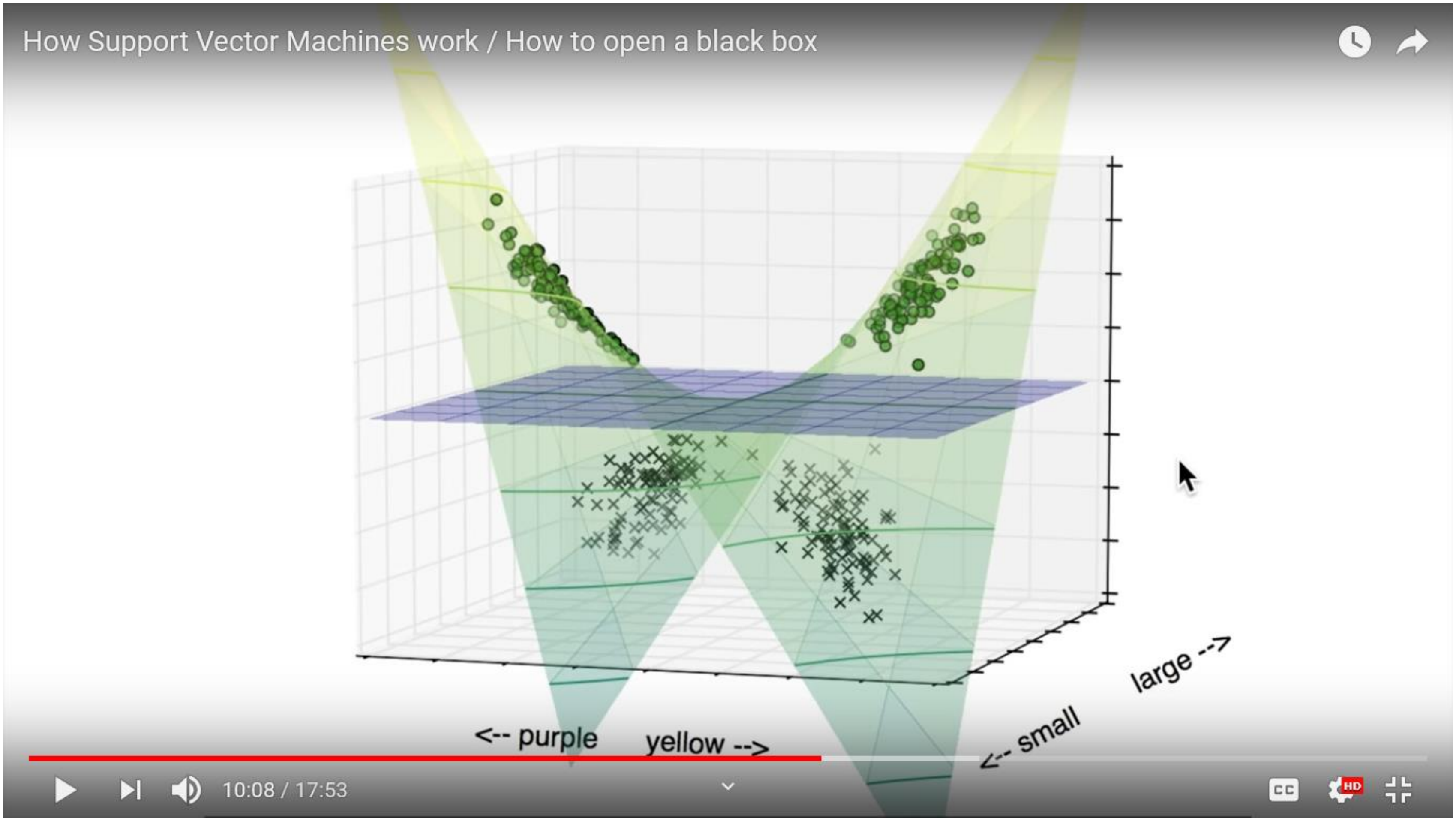
Non-linear SVMs



Non-linear SVMs: Feature spaces



Non-linear SVMs: Feature spaces



SVM – Overlapping Class Scenario

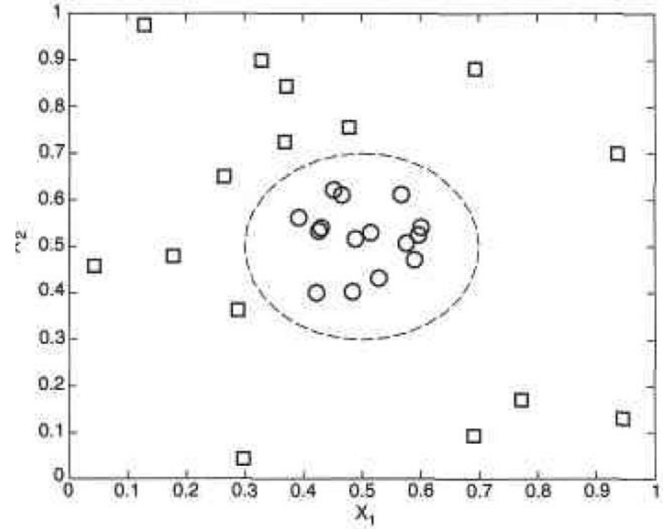
- Data is not separable linearly
- Margin will become inefficient
- Data needs to be transformed from original coordinate space **x** to a new space **Φ(x)**, so that linear decision boundary can be applied
- A non-linear transformation function is needed, like, ex:

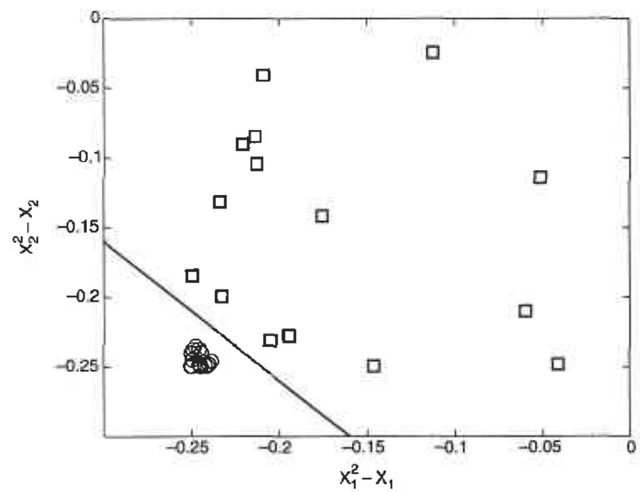
$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

- In the transformed space we can choose w = (w_0, w_1, … , w_4) such that

$$w_4 x_1^2 + w_3 x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

- The linear decision boundary in the transformed space has the following form: w. **Φ**(x) + b = 0





SVM – Overlapping Class Scenario

- The learning task for a nonlinear SVM can be formalized as:

$$\min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2}$$
$$\text{subject to} \quad y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1, \quad i = 1, 2, \ldots, N.$$

- Only difference with Linear SVM is the transformed attributes **Φ(x)**
- Similar to Linear SVM, using dual Lagrangian: $$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

- Once the Lagrange Multipliers are found using quadratic programming techniques, the parameters w and b can be derived using:

$$\mathbf{w} = \sum_i \lambda_i y_i \Phi(\mathbf{x}_i)$$
$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i) + b) - 1 \} = 0,$$

- A test instance z can be classified as: $$f(\mathbf{z}) = sign(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = sign\left(\sum_{i=1}^{n} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b \right)$$

- Only missing piece of the puzzle is transforming x to **Φ(x)** – called ‘Kernel Trick’

SVM Kernel Trick - Techniques

- Kernel trick is a method for computing similarity in the transformed space using the original attribute set
- The dot product between two input vectors u and v in the transformed space can be written as:

$$\begin{aligned}\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) &= (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, 1) \cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, 1) \\ &= u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 + 2u_2 v_2 + 1 \\ &= (\mathbf{u} \cdot \mathbf{v} + 1)^2.\end{aligned}$$

- Dot product in the transformed space can be expressed in terms of a similarity function in the original space

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^2$$

- The similarity function, K, which is computed in the original attribute space is known as the kernel function
- Kernel function should satisfy ‘Mercer’s Theorem’

What Functions are Kernels?

- **Kernel is a continuous function *$k(x,y)$* that takes two arguments *x and y* (real numbers, functions, vectors, etc.) and maps them to a real value independent of the order of the arguments, i.e., *$k(x,y)=k(y,x)$* .**

- **For some functions $K(x_i,x_j)$ checking that $K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$ can be cumbersome.**
- **Mercer's theorem:**

  Every positive-semidefinite symmetric function is a kernel

What Functions are Kernels?

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, i.e.
  - $d(x, x') \geq 0 \quad$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0 \quad$ only for $x = x'$,
  - $d(x, x') = d(x', x) \quad$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x') \quad$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

2) We can *construct kernels from other kernels*:

- if k is a kernel and $\alpha > 0$, then αk and $k + \alpha$ are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

Examples of Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j})= \mathbf{x_i}^{\mathsf{T}}\mathbf{x_j}$

- Polynomial of power p: $K(\mathbf{x_i},\mathbf{x_j})= (1+ \mathbf{x_i}^{\mathsf{T}}\mathbf{x_j})^p$

- Gaussian Kernel:

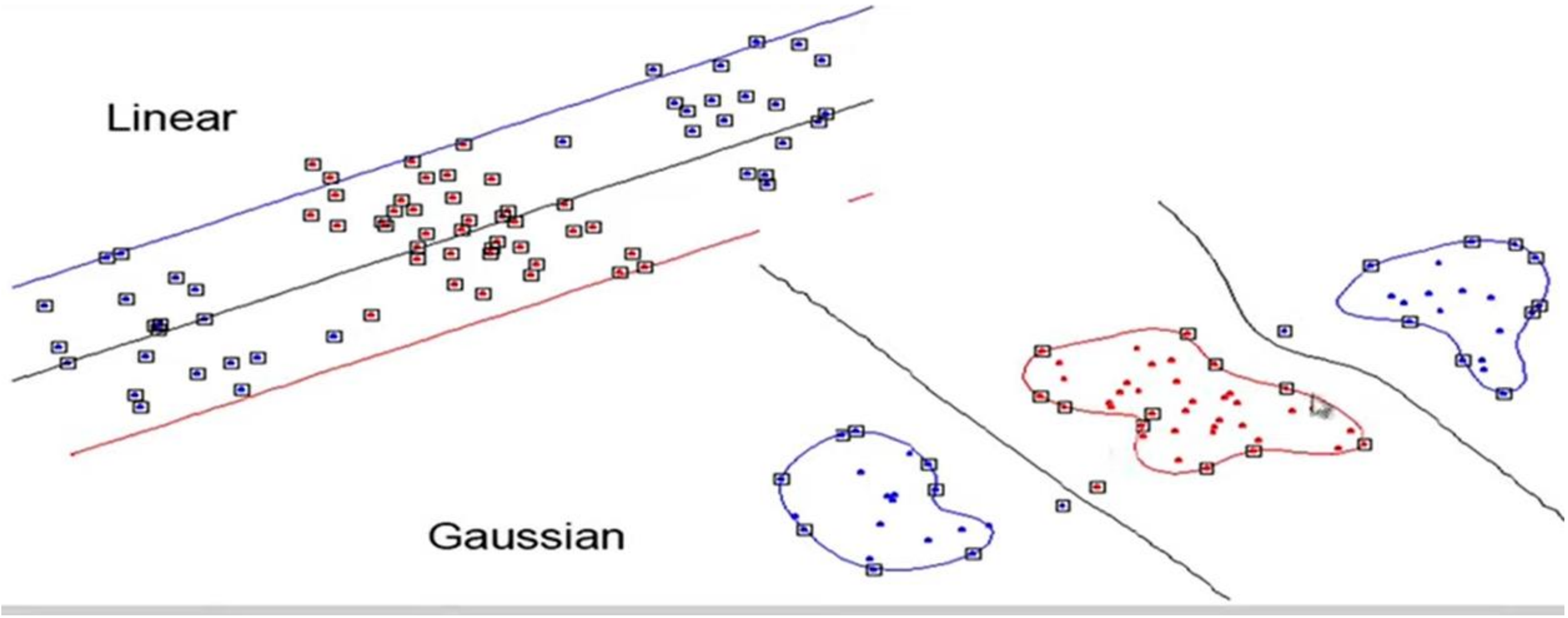
$$K(\mathbf{x_i},\mathbf{x_j}) = \exp(-\frac{\left\|\mathbf{x_i} - \mathbf{x_j}\right\|^2}{2\sigma^2})$$

SVM Kernel Functions

- **Linear kernel** takes less training time when compared to other kernel functions.
- **Linear kernel** is mostly preferred for text classification problems as it performs well for large datasets.
- **Gaussian kernels** tend to give good results when there is no additional information regarding data that is not available.
- **RBF kernel** is also a kind of Gaussian kernel which projects the high dimensional data and then searches a linear separation for it.
- **Polynomial kernels** give good results for problems where all the training data is normalized.

Name	Function	Type problem
Polynomial Kernel	$(x_i^t x_j + 1)^q$ q is degree of polynomial	Best for Image processing
Sigmoid Kernel	$\tanh(a x_i^t x_j + k)$ k is offset value	Very similar to neural network
Gaussian Kernel	$\exp^{(}\|x_i - x_j\|^2 / 2\sigma^2)$	No prior knowledge on data
Linear Kernel	$\left(1 + x_i x_j min(x_i, x_j) - \frac{(x_i + x_j)}{2} min(x_i, x_j)^2 + \frac{min(x_i, x_j)^3}{3}\right)$	Text Classification
Laplace Radial Basis Function (RBF)	$\left(e^{(}-\lambda\|x_i - x_j\|), \lambda >= 0\right)$	No prior knowledge on data

There are many more kernel functions.


Linear
Gaussian

Non-linear SVMs Mathematically

- **The solution is:**

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

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

Non-linear SVM using kernel

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

Nonlinear SVM - Summary

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Properties of SVM

- It provides a clear margin of separation.
- Flexibility in choosing a similarity function
- Sparseness of solution when dealing with large data sets

    - Only support vectors are used to specify the separating hyperplane

  – Therefore SVM also called sparse kernel machine.

  Ability to handle large feature spaces

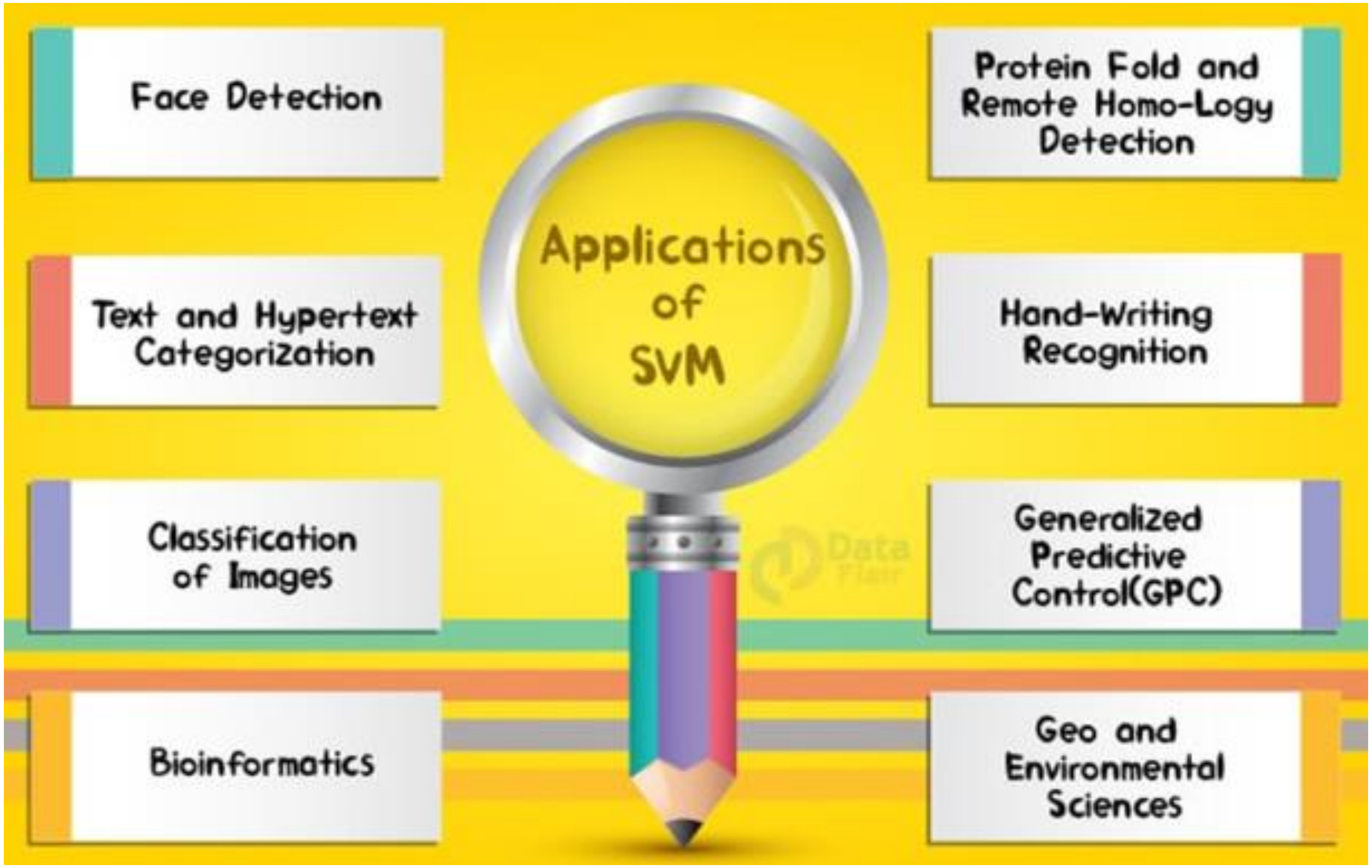
    - complexity does not depend on the dimensionality of the feature space

    - It is very effective for the dataset where the number of features are greater than the data points.

- Overfitting can be controlled by soft margin approach
- Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution

SVM Applications

SVM has been used successfully in many real-world problems



Application : Text Categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.

  A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

Text Categorization using SVM

- **The distance between two documents is $\varphi(x)\cdot\varphi(z)$**

- **$K(x,z) = \varphi(x)\cdot\varphi(z)$ is a valid kernel, SVM can be used with $K(x,z)$ for discrimination.**

- **Why SVM?**

  -High dimensional input space

  -Few irrelevant features (dense concept)

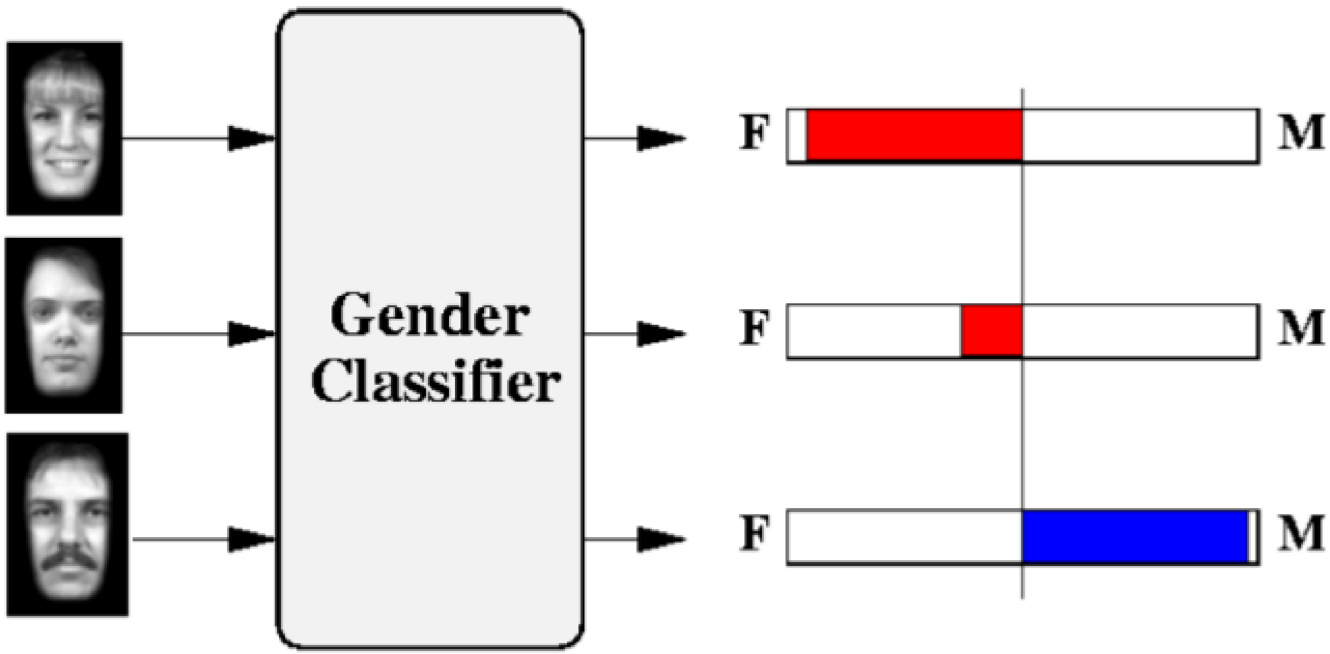
  -Sparse document vectors (sparse instances)

  -Text categorization problems are linearly separable

Using SVM

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

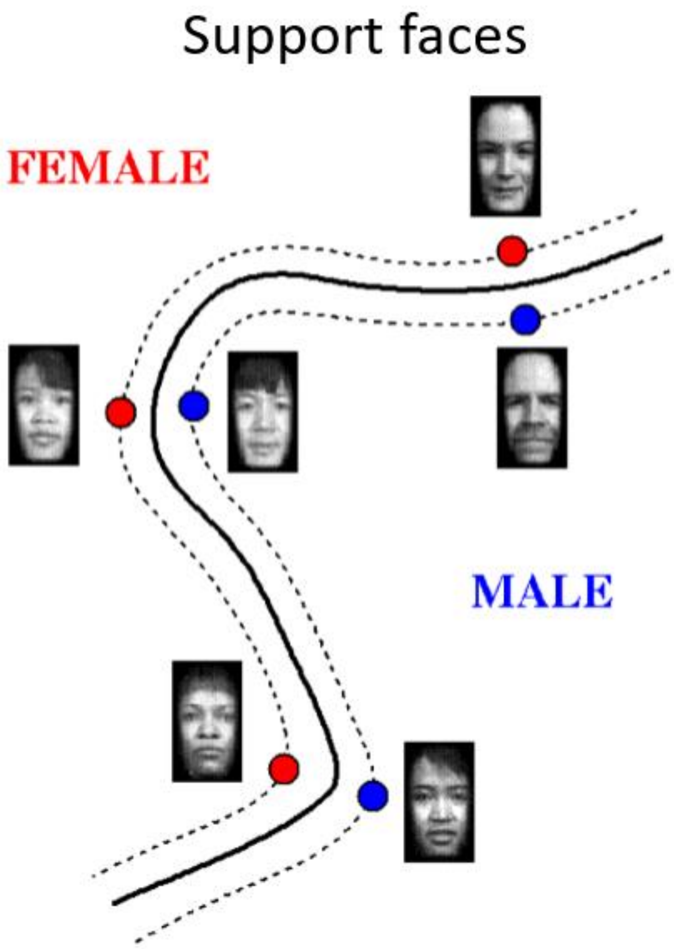
Learning Gender from image with SVM



Moghaddam and Yang, Learning Gender with Support Faces, TPAMI 2002

Moghaddam and Yang, Face & Gesture 2000

Support faces



Accuracy of SVM Classifier

Figure 6. SVM vs. Human performance

- SVMs performed better than humans, at either resolution

Some Issues

- **Sensitive to noise**

  - A relatively small number of mislabeled examples can dramatically decrease the performance

- **Choice of kernel**

  - Gaussian or polynomial kernel is default

  - if ineffective, more elaborate kernels are needed

  - domain experts can give assistance in formulating appropriate similarity measures

- **Choice of kernel parameters**

  - e.g. σ in Gaussian kernel

  - σ is the distance between closest points with different classifications

  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

- **Optimization criterion** – Hard margin v.s. Soft margin

  - a lengthy series of experiments in which various parameters are tested

SVM Implementation - Python

- **Predict credit card defaults using past credit history**
- **Data set:**
  - ID: ID of each client
  - LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
  - SEX: Gender (1=male, 2=female)
  - EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
  - MARRIAGE: Marital status (1=married, 2=single, 3=others)
  - AGE: Age in years
  - PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)
  - PAY_2: Repayment status in August, 2005 (scale same as above)
  - PAY_3: Repayment status in July, 2005 (scale same as above)
  - PAY_4: Repayment status in June, 2005 (scale same as above)
  - PAY_5: Repayment status in May, 2005 (scale same as above)
  - PAY_6: Repayment status in April, 2005 (scale same as above)
  - BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
  - BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
  - BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
  - BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
  - BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
  - BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
  - PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
  - PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
  - PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
  - PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
  - PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
  - PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
  - default.payment.next.month: Default payment (1=yes, 0=no)

SVM Implementation - Python

- Load the data

```
dataset = pd.read_csv("../input/UCI_Credit_Card.csv")
print(dataset.head())
print(f"Amount of samples: {dataset.shape[0]}")
```

```
   ID          ...          default.payment.next.month
0   1          ...                                   1
1   2          ...                                   1
2   3          ...                                   0
3   4          ...                                   0
4   5          ...                                   0

[5 rows x 25 columns]
Amount of samples: 30000
```

- Split the data

```
In [6]:
print("Getting new dataset split...")
# Get a dataset split for training and validation
x_train, y_train, x_test, y_test = trainTestSplit(dataset, 0.2)
```

Linear SVM Implementation - Python

```python
# Create Linear SVM model
lsvm = LinearSVC(max_iter=32000) # If we don't specify anything it assumed all classes have same weight
lsvm.fit(x_train, y_train)
y_pred = lsvm.predict(x_test)
linear_acc = lsvm.score(x_test, y_test)
print(f"Linear SVM Acc: {linear_acc*100} % - Validated on {y_test.shape[0]} samples")
print(classification_report(y_test, y_pred))
```

```
#### Linear SVM Results ####
Linear SVM Acc: 80.60000000000001 % - Validated on 6000 samples
              precision    recall  f1-score   support

           0       0.81      0.98      0.89      4683
           1       0.73      0.19      0.30      1317

    accuracy                           0.81      6000
   macro avg       0.77      0.58      0.59      6000
weighted avg       0.79      0.81      0.76      6000
```

SVM Polynomial Implementation - Python

```
# Create Polynomial SVM
svm = SVC(gamma='scale', kernel='poly', degree=3)
svm.fit(x_train, y_train)
poly_acc = svm.score(x_test, y_test)
y_pred = svm.predict(x_test)
print(f"Polynomial SVM Acc: {poly_acc*100} %")
print(classification_report(y_test, y_pred))
```

```
#### Polynomial SVM with Degree 3 Results ####
Polynomial SVM Acc: 82.03333333333333 %
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      4683
           1       0.68      0.34      0.46      1317

    accuracy                           0.82      6000
   macro avg       0.76      0.65      0.67      6000
weighted avg       0.80      0.82      0.80      6000
```

SVM – Python Implementation

- Load the data

```
In [2]:  dataset = pd.read_csv('../input/diabetes.csv')
```

- Split the data

```
In [7]:  X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.20)
```

SVM – Python Implementation

- Feature Scaling

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

- Implement SVM with Linear Kernel

In [9]:
```
classifier = SVC(random_state=0, kernel='rbf')
classifier.fit(X_train, y_train)
```

Out[9]:
```
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=0, shrinking=True,
  tol=0.001, verbose=False)
```

SVM – Python Implementation

- Validate the Model

```
In [10]:
y_pred = classifier.predict(X_test)
```

- Evaluate the Model

```
In [11]:
cm = confusion_matrix(y_test, y_pred)
print (cm)
print(f1_score(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
[[97 10]
 [19 28]]
0.658823529412
0.811688311688
```

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

One-vs-all (a.k.a. one-vs-others)

- Train C classifiers
- In each, pos = data from class *i*, neg = data from classes other than *i*
- The class with the most confident prediction wins
- Example:
  - You have 4 classes, train 4 classifiers
  - 1 vs others: score 3.5
  - 2 vs others: score 6.2
  - 3 vs others: score 1.4
  - 4 vs other: score 5.5
  - Final prediction: class 2
- Issues?

Multi-Class Problem

One-vs-one (a.k.a. all-vs-all)

- Train C(C-1)/2 binary classifiers (all pairs of classes)
- They all vote for the label
- Example:
  - You have 4 classes, then train 6 classifiers
  - 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, 2 vs 4, 3 vs 4
  - Votes: 1, 1, 4, 2, 4, 4
  - Final prediction is class 4

Reference

- **Support Vector Machine Classification of Microarray Gene Expression Data**, Michael P. S. Brown William Noble Grundy, David Lin, Nello Cristianini, Charles Sugnet, Manuel Ares, Jr., David Haussler
- **Text categorization with Support Vector Machines: learning with many relevant features**

  T. Joachims, ECML - 98
- Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition

Good Web References for SVM

- https://www.coursera.org/learn/machine-learning/home/week/7
- https://www.youtube.com/watch?time_continue=1&v=_PwhiWxHK8o
- https://www.youtube.com/watch?v=eh3sM4-3heo
- https://www.youtube.com/watch?time_continue=138&v=s8B4A5ubw6c
- https://data-flair.training/blogs/svm-kernel-functions/
- http://www.cs.utexas.edu/users/mooney/cs391L/
- https://www.coursera.org/learn/machine-learning/home/week/7
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- https://towardsdatascience.com/linear-svm-classifier-step-by-step-theoretical-explanation-with-python-implementation-69767437e0e6
- Kernel
- Polynimial Kernel
- Radial basis kernel
- https://ajaytech.co/python-svm-support-vector-machine/
- https://towardsdatascience.com/linear-svm-classifier-step-by-step-theoretical-explanation-with-python-implementation-69767437e0e6
- https://www.learnopencv.com/svm-using-scikit-learn-in-python/
- https://archive.ics.uci.edu/ml/index.php
- https://archive.ics.uci.edu/ml/datasets.php

Thank You