COSC 4368 Fundamentals of Artificial Intelligence

Nonparametric Models
October 2nd, 2023

Parametric Model vs Nonparametric Model

- Parametric model: summarizes data with a set of parameters of fixed size
  - E.g., linear regression model,

- Nonparametric model: does not use a bounded set of parameters to characterize data
  - A.k.a. instance-based learning or memory-based learning

Nearest Neighbor

e.g., image classification

```
def train(images, labels):
  # Machine learning!
  return model
```

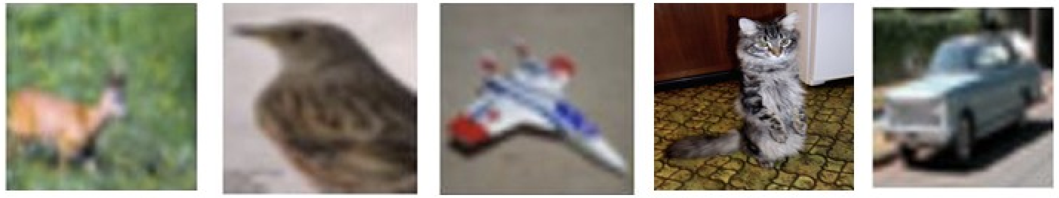
Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Predict the label of the most similar training image

Nearest Neighbor

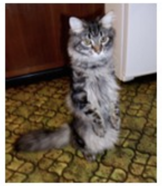
deer bird plane cat car



Training data with labels

?

query data

Distance Metric $\Big|$  ,  $\Big| \to \mathbb{R}$

Distance Metrics

- Given a query point and an example point , norm (Minkowski distance) is used to measure their distance:
- E.g., norm with :

$$\left| \begin{array}{c} \text{test image} \\ \begin{bmatrix} 56 & 32 & 10 & 18 \\ 90 & 23 & 128 & 133 \\ 24 & 26 & 178 & 200 \\ 2 & 0 & 255 & 220 \end{bmatrix} \end{array} - \begin{array}{c} \text{training image} \\ \begin{bmatrix} 10 & 20 & 24 & 17 \\ 8 & 10 & 89 & 100 \\ 12 & 16 & 178 & 170 \\ 4 & 32 & 233 & 112 \end{bmatrix} \end{array} \right| = \begin{array}{c} \text{pixel-wise absolute value differences} \\ \begin{bmatrix} 46 & 12 & 14 & 1 \\ 82 & 13 & 39 & 33 \\ 12 & 10 & 0 & 30 \\ 2 & 32 & 22 & 108 \end{bmatrix} \end{array} \xrightarrow{\text{add}} 456$$

Nearest Neighbor Classifier with L1 Distance

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Memorize the training data

For each test image:

- Find closest train image
- Predict label of the nearest image

Nearest Neighbor Classifier with L1 Distance

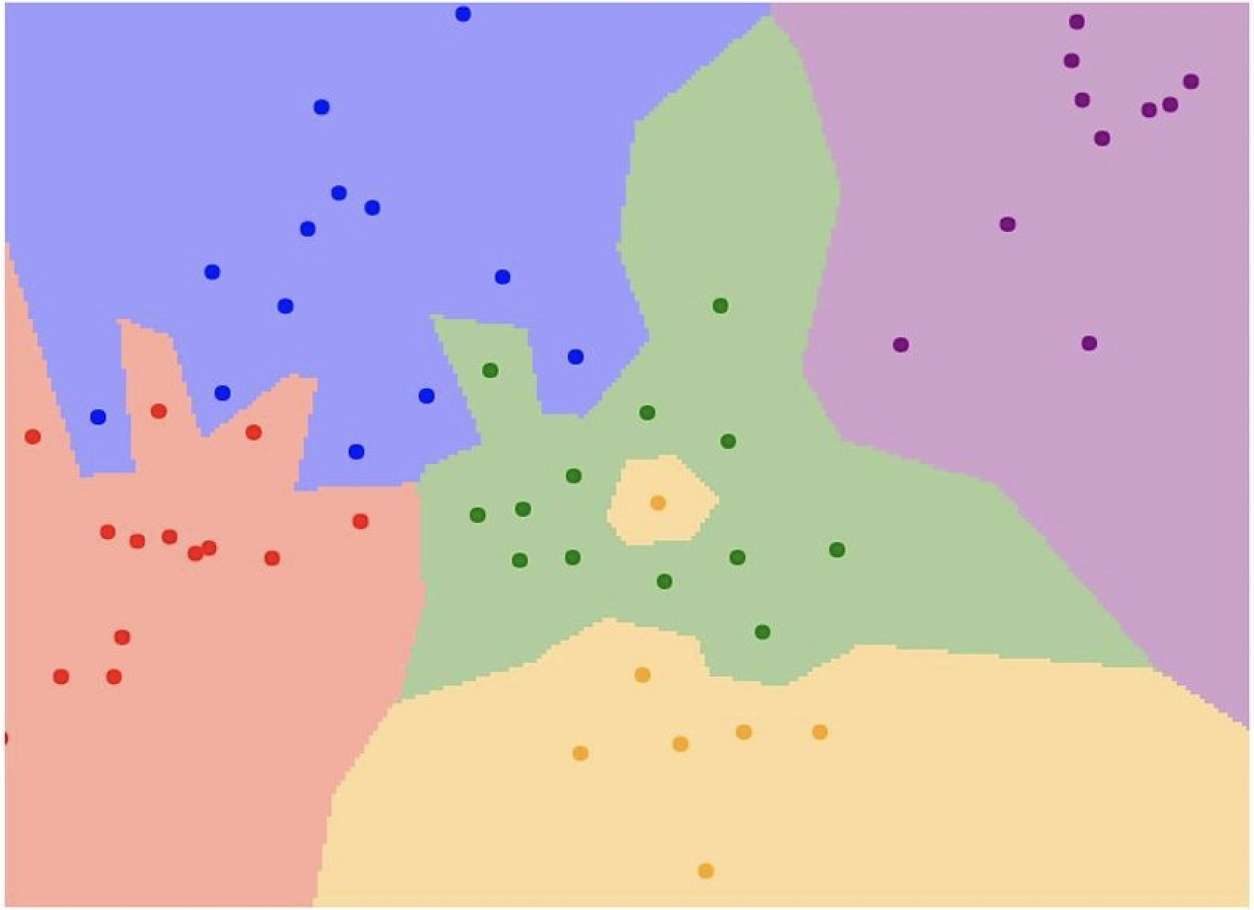
```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N examples, how fast are training and prediction?

A: Train , test

Bad: we want classifiers to be fast at prediction; slow for training is ok

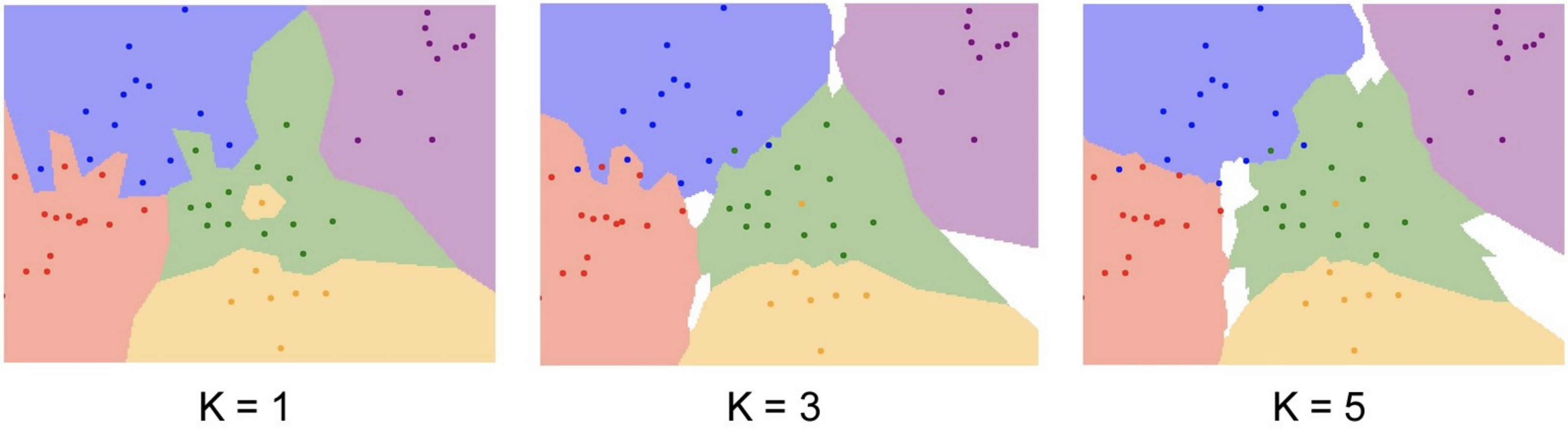
What Does It Look Like



1-nearest neighbor

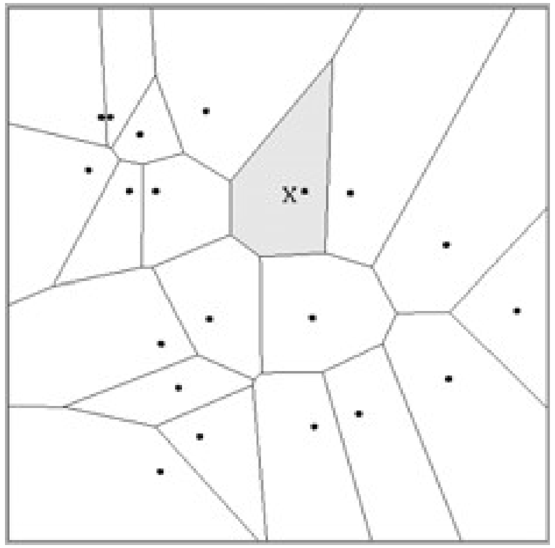
Overfitting and underfitting also exist here

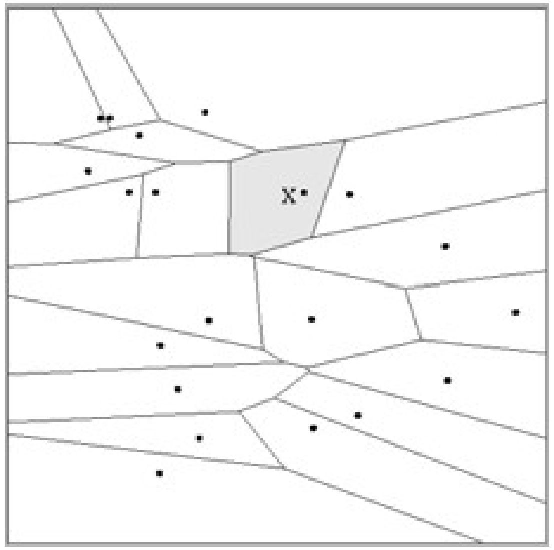
K-Nearest Neighbors



- Instead of directly copying label from nearest neighbor, take majority vote from K closest neighbors
- E.g., if K=3, neighbor values: <Yes, No, Yes>, result of vote: Yes

Different Metrics Change the Decision Surface





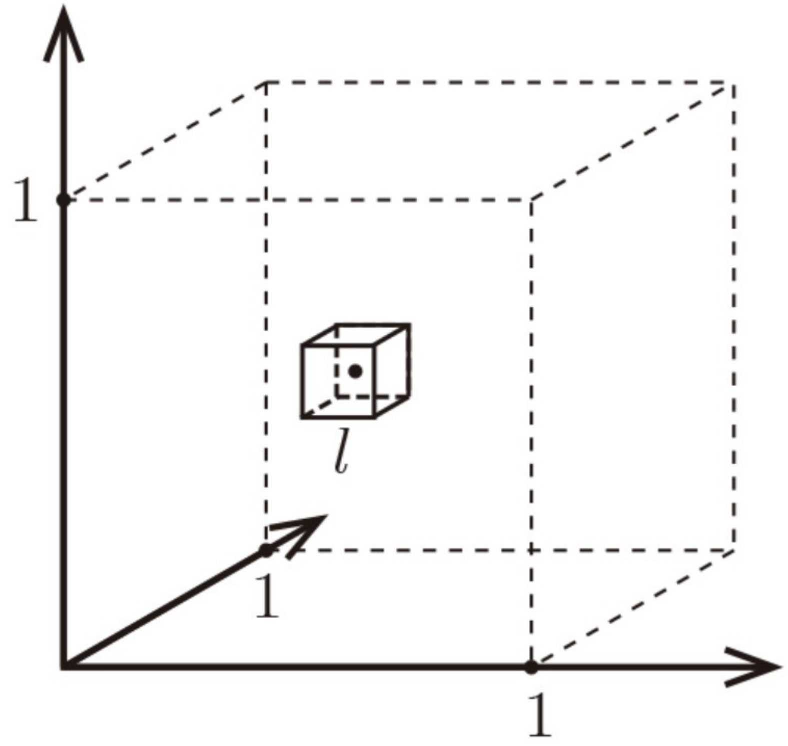
$dist(\boldsymbol{x}_q, \boldsymbol{x}_j) = (\boldsymbol{x}_{q,1} - \boldsymbol{x}_{j,1})^2 + (\boldsymbol{x}_{q,2} - \boldsymbol{x}_{j,2})^2$

$dist(\boldsymbol{x}_q, \boldsymbol{x}_j) = (\boldsymbol{x}_{q,1} - \boldsymbol{x}_{j,1})^2 + (\boldsymbol{3}\,\boldsymbol{x}_{q,2} - \boldsymbol{3}\,\boldsymbol{x}_{j,2})^2$

- The choices of the hyperparameters, e.g., distance metrics and value of , are very problem dependent

Curse of Dimensionality

- In high dimensional spaces, points that are drawn from a probability distribution tend to never be close together



- All training data uniformly locate in unit cube with dimension
- Let be the edge length of the smallest hyper-cube that contains all -nearest neighbors of a test point
- The volume of this hyper-cube is

Curse of Dimensionality

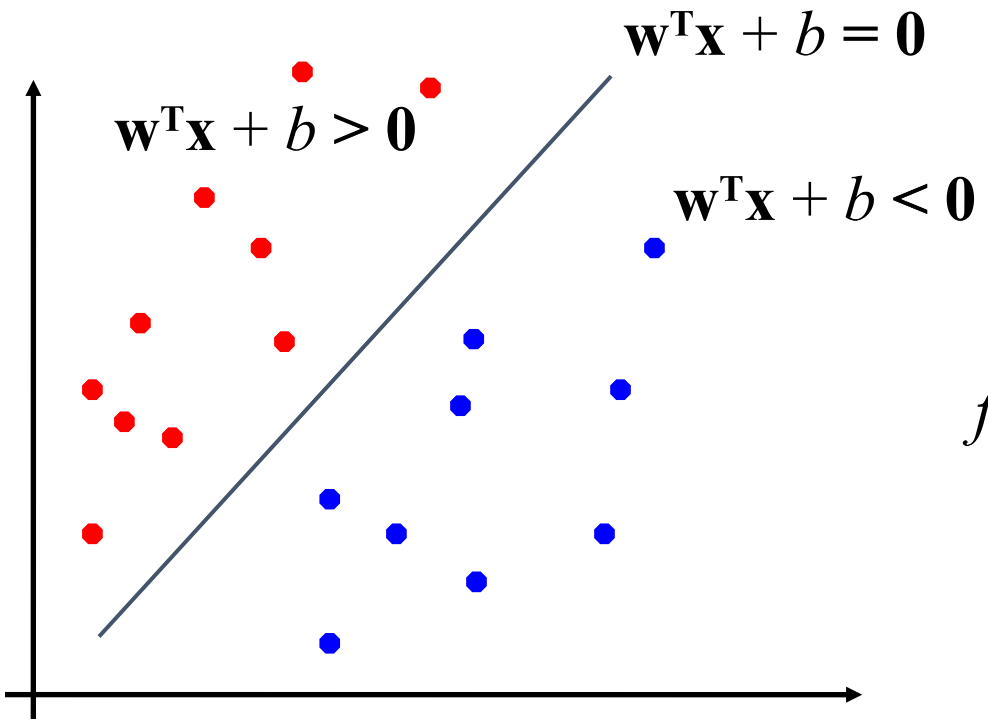
- For ,

d	ℓ
2	0.1
10	0.63
100	0.955
1000	0.9954

- Almost the entire space is needed to find the 10-nearest neighbors for large

Support Vector Machines

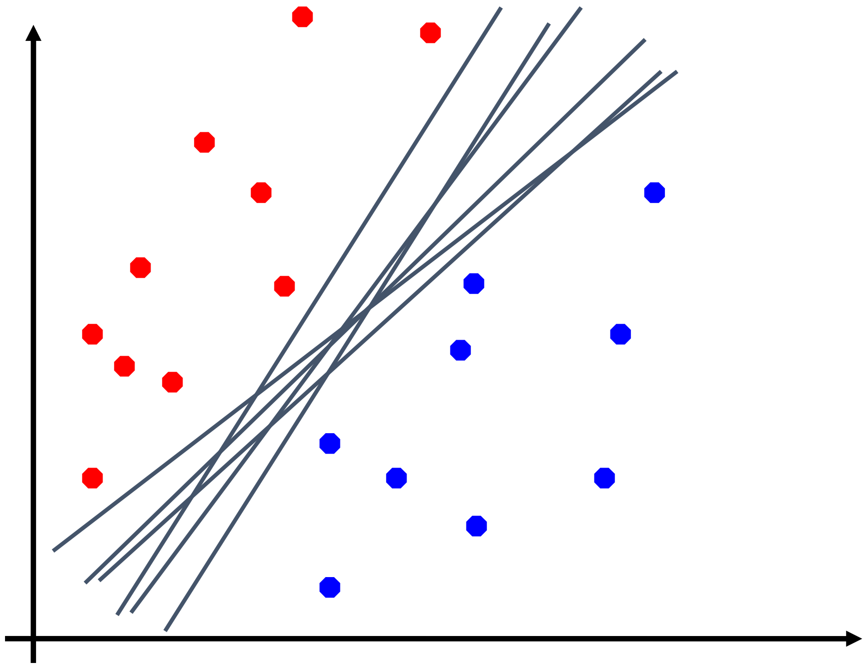
- Revisit linear classification
  - Binary classification can be viewed as the task of separating classes in feature space



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

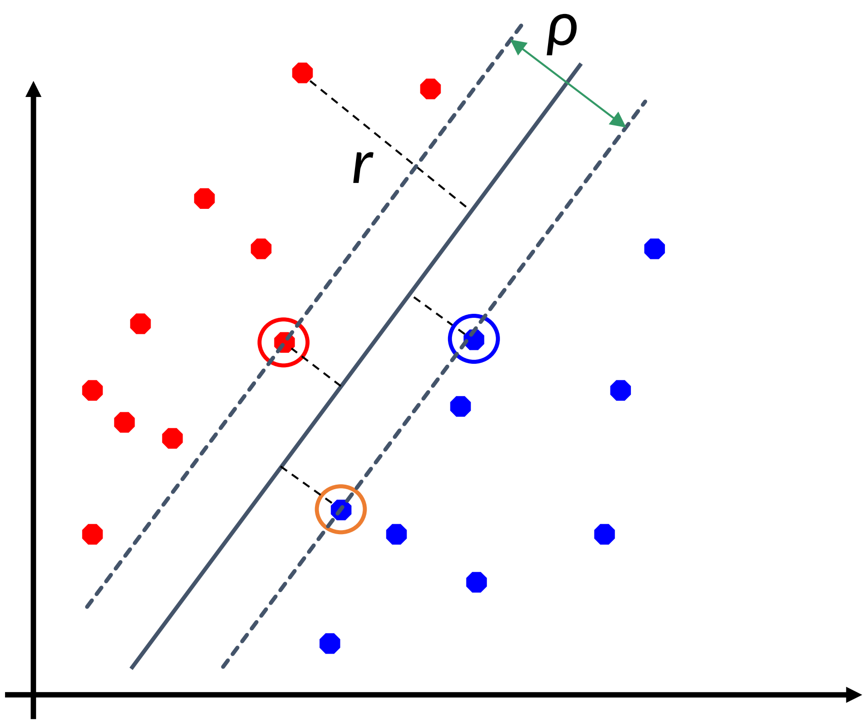
Support Vector Machines

- Which linear separator is better?



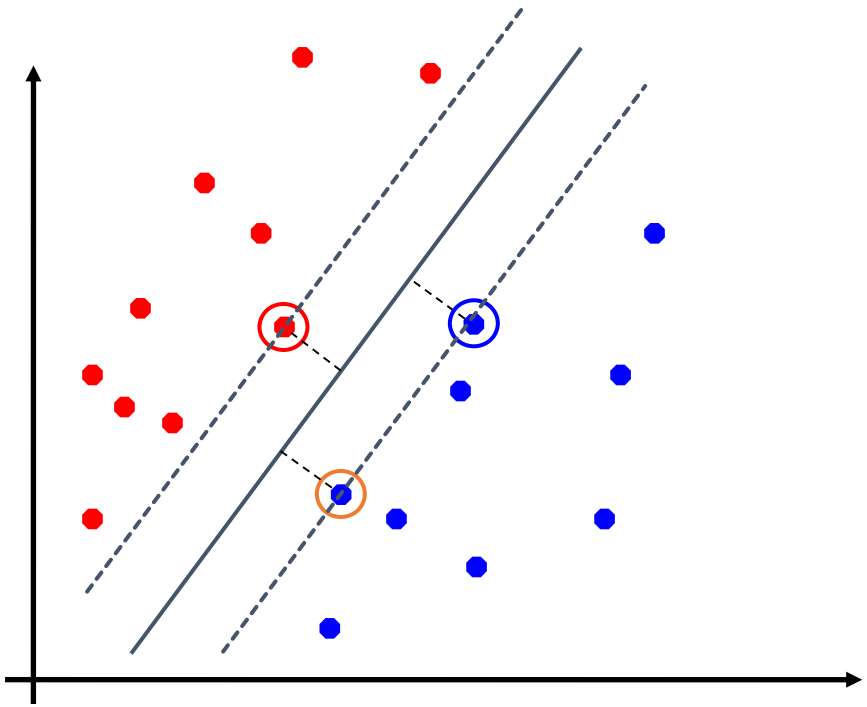
Support Vector Machines

- Classification margin:
  - Distance from example $\mathbf{x}_i$ to the separator is
  - Examples closed to the hyperplane are ***support vectors***
  - Margin ρ of the separator is the distance between support vectors



Maximum Margin Separator

- Maximizing the margin is good according to intuition and PAC 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 $(\mathbf{x}_i, y_i)$:

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

- 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) $\mathbf{w}$ and b as:

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

Linear SVM Mathematically

- 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

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

- Quadratic programming problems are a well-known class of optimization problems for which several algorithms exist
- One way to construct 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

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

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

(2) $\alpha_i \geq 0$ for all α_i

Solving the Optimization Problem

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

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

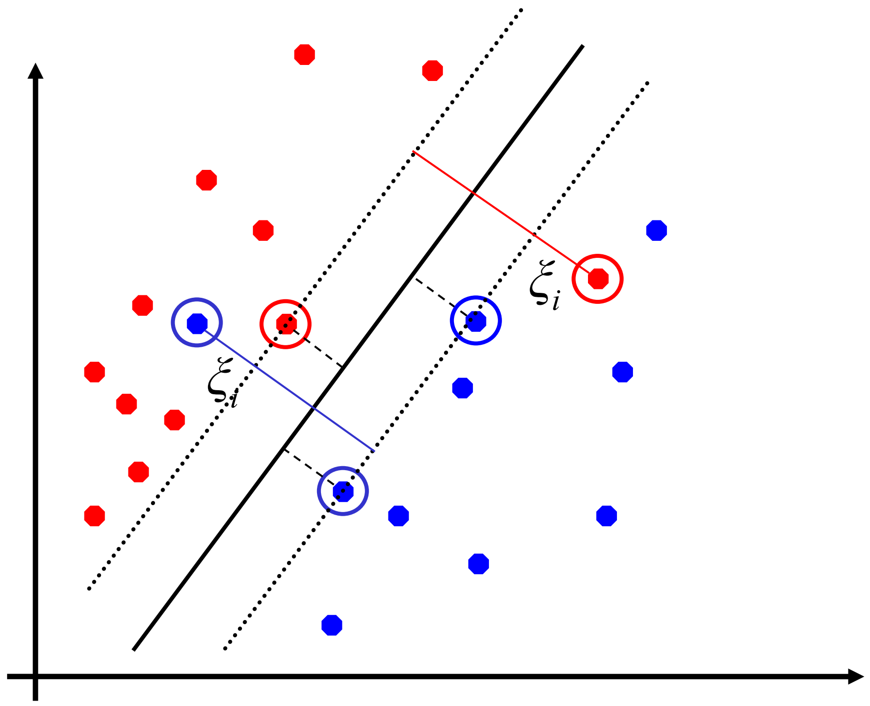
- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector.
- Then the classifying function is (note that we don't need **w** explicitly):

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

- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x}_i$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$ between all training points.

Soft Margin Classification

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



Soft Margin Classification Mathematically

- The old formulation:

Find $\mathbf{w}$ 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 : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_{i,}$, $\xi_i \geq 0$

- 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:

> Find $\alpha_1 \ldots \alpha_N$ such that
> $\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ 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:

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

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

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

Linear SVM: 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

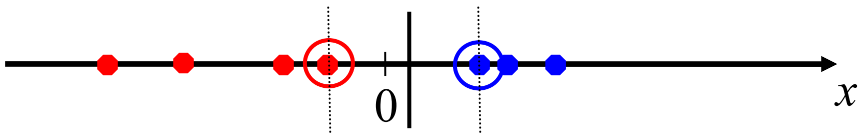
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

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

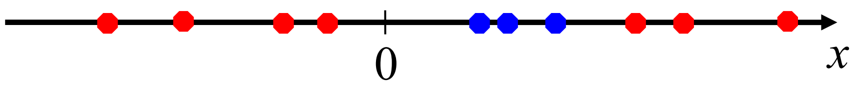
$f(\mathbf{x}) = \Sigma\alpha_i y_i \boxed{\mathbf{x}_i^T\mathbf{x}} + b$

Non-linear SVM

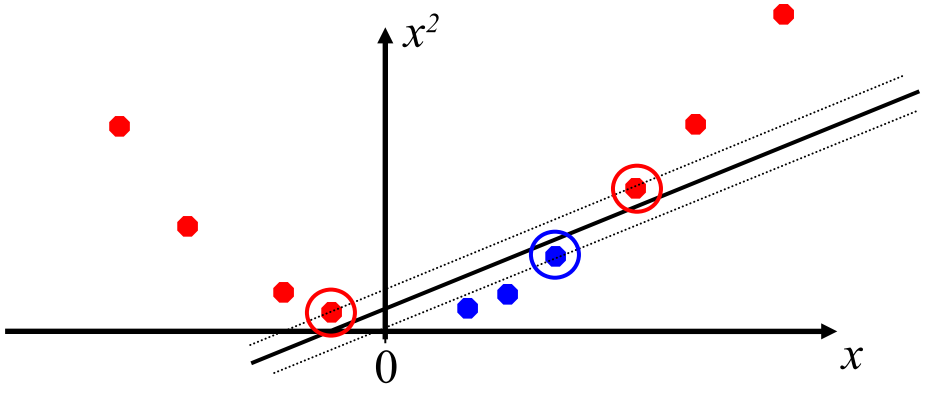
- Datasets that are linearly separable with some noise work out great:



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

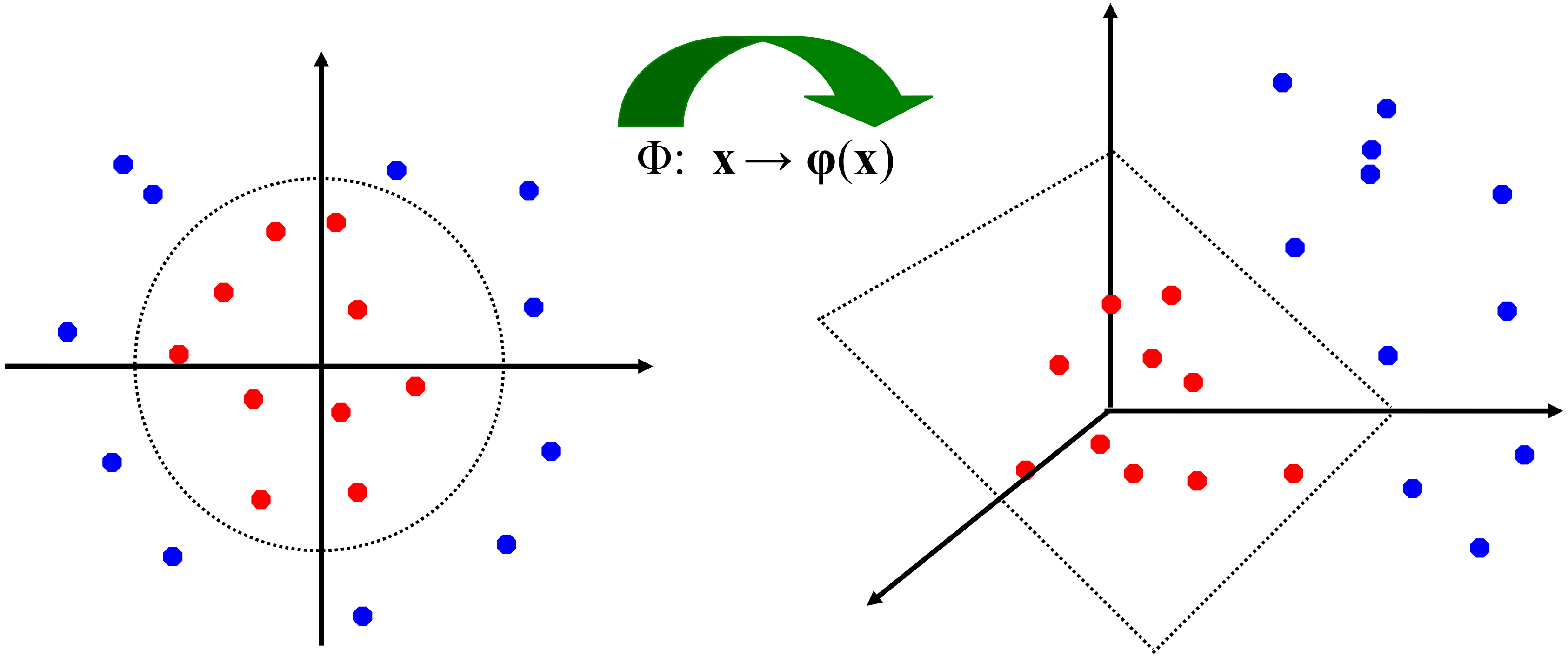


- How about… mapping data to a higher-dimensional space:



Non-linear SVM: Feature Space

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



  - In general, N data points will always be separable in spaces of N-1 dimensions or more

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 the dimension $\boldsymbol{\varphi}(\mathbf{x})$ is exponential in 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)

- Higher-dimensional space still has *intrinsic* dimensionality d, but linear separators in it correspond to *non-linear* separators in original space.

Non-linear SVM 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 Summary

- SVMs learn hyperplanes that separate two classes maximizing the *margin between them* (*the empty space between the instances of the two classes)*.

- Soft margin SVMs introduce slack variables, in the case that classes are not linear separable and trying to maximize margins while keeping the training error low.

- The most popular versions of SVMs use non-linear kernel functions to map the attribute space into a higher dimensional space, to facilitate finding "good" linear decision boundaries in the modified space (which correspond to non-linear decision boundaries in the original space).

- SVMs find "margin optimal" hyperplanes by solving a convex quadratic optimization problem. The complexity of this optimization problem is high; however, as computer got faster, using SVMs on large datasets is no longer a major challenge.

- In general, SVMs accomplish quite high accuracies, if compared to other techniques.