Ye cannot bear all things now; nevertheless, be of good cheer, for I will lead you along.

Doctrine & Covenants 78:18

I will give unto the children of men line upon line, precept upon precept, here a little and there a little.

2 Nephi 28:30

And see that all these things are done in wisdom and order; for it is not requisite that [you] *should run faster than* [you have] *strength. And again, it is expedient that* [you] *should be diligent, that thereby* [you] *might win the prize; therefore, all things must be done in order.*[8]

Mosiah 4:27

It has been my experience that the Lord gives us just as much as we have the strength to handle, plus a little extra, so that we can increase our faith and strength. Just as with all of Heavenly Father's creations, you were designed to grow and progress. You were not meant to stay the way you are. Change and improvement are built into your eternal DNA.

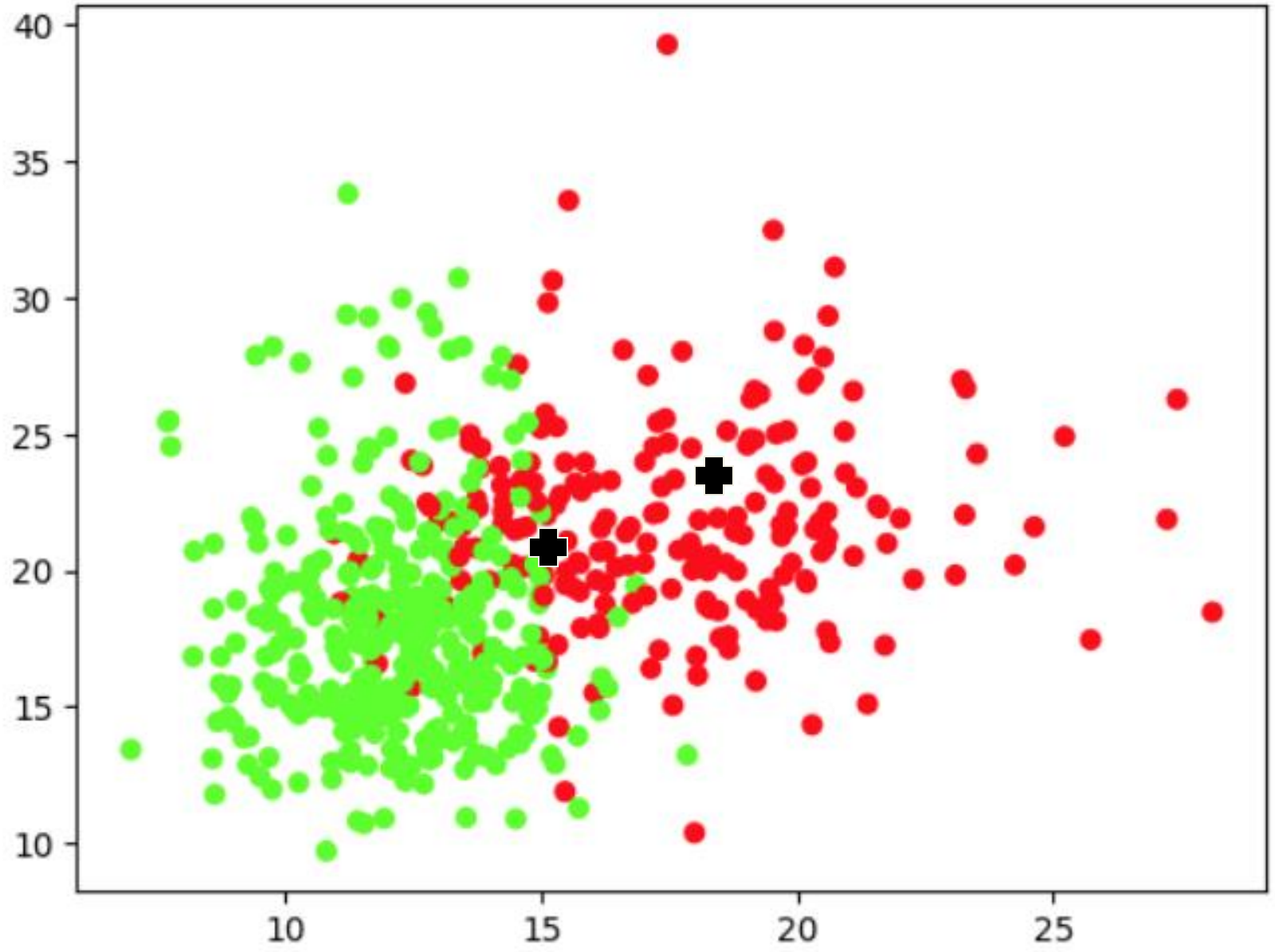
Jan E. Newman – 2nd Counselor, Sunday School General Presidency
BYU Devotional – Feb 6, 2024

ML MODELS

Classification

- k-nearest neighbor classifier
- Support Vector Machines
- Naïve Bayes
- Logistic Regression

NEAREST NEIGHBOR CLASSIFICATION



How should we classify this instance?

And this one?

Instance-Based Classifiers

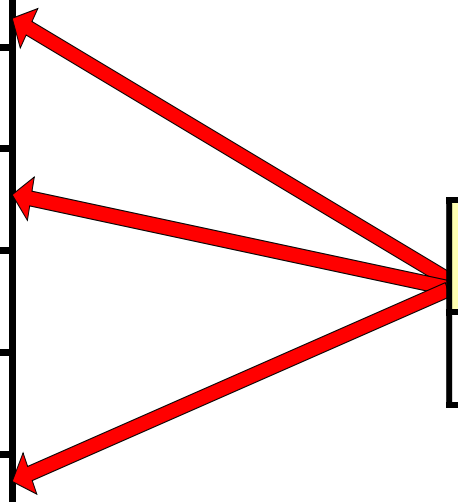
Set of Stored Cases

Atr1	……	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

- Store the training records
- Use training records to predict the class label of unseen cases

Unseen Case

Atr1	……	AtrN

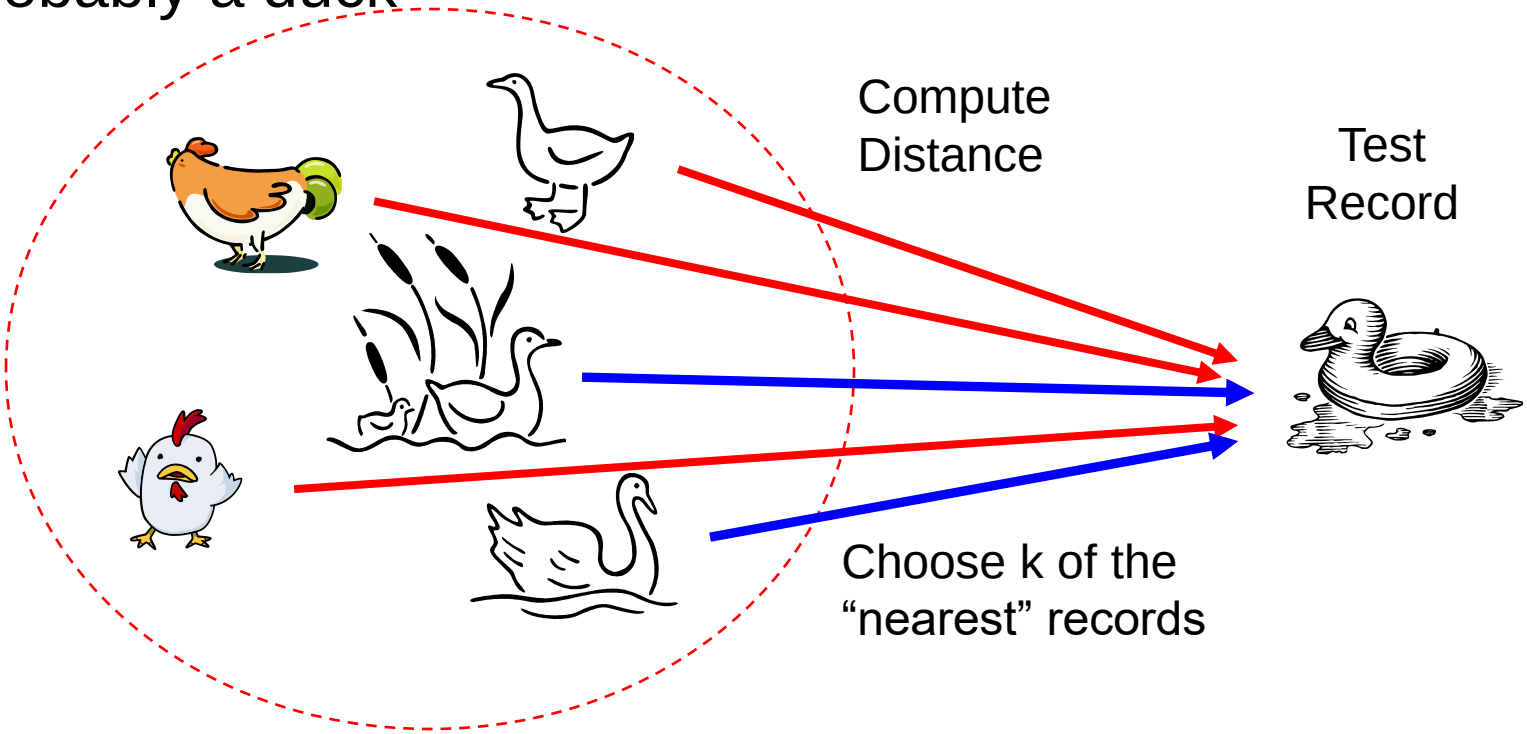


Instance Based Classifiers

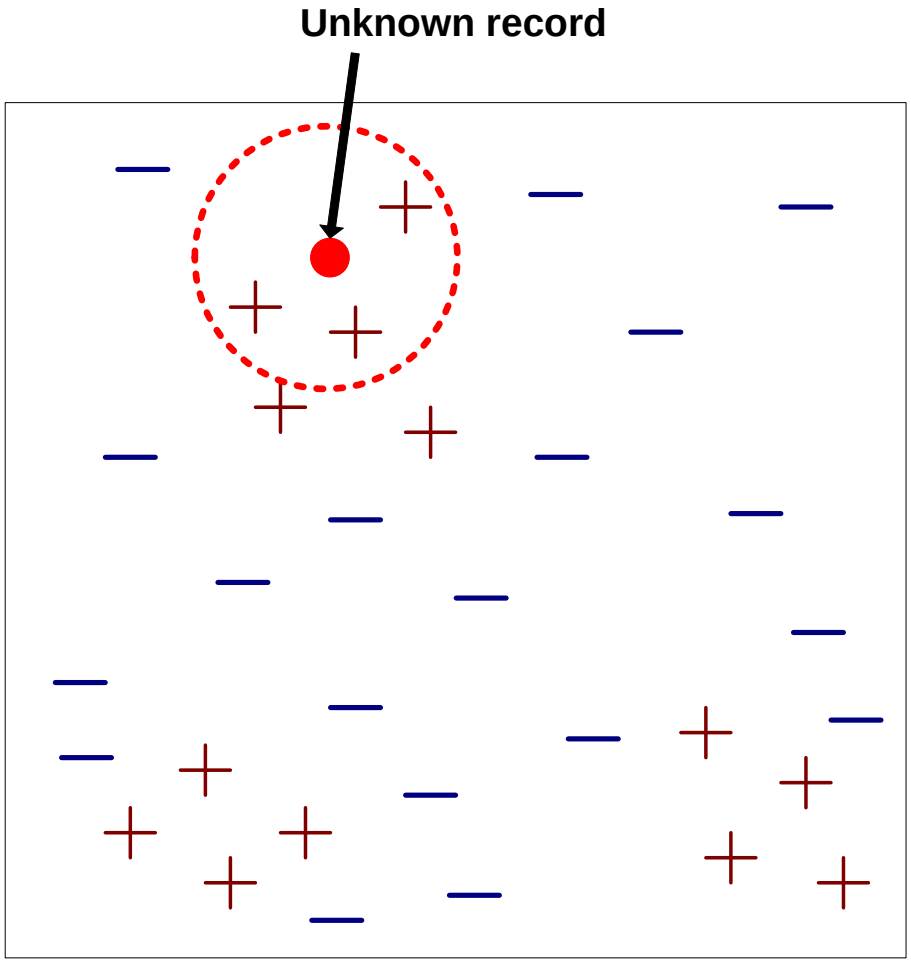
- Examples:
  - Rote-learner
    - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
  - Nearest neighbor
    - Uses k “closest” points (nearest neighbors) for performing classification

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck and quacks like a duck, then it's probably a duck

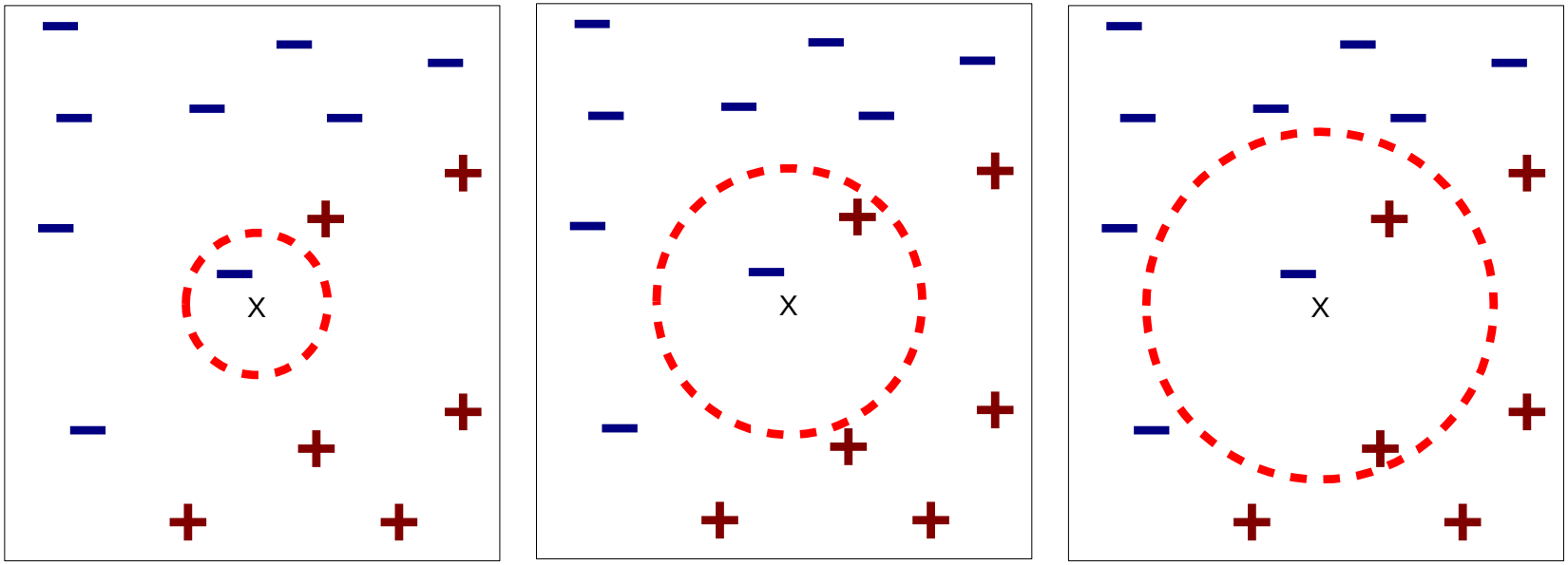


Nearest-Neighbor Classifiers



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve
- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Definition of Nearest Neighbor

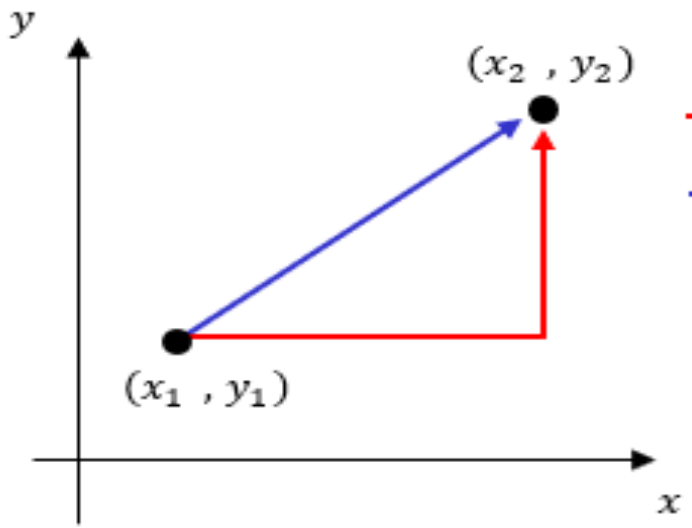


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Nearest Neighbor Classification

- Compute distance between two points:
  - Euclidean distance or Manhattan distance typically



Manhattan Distance L^1

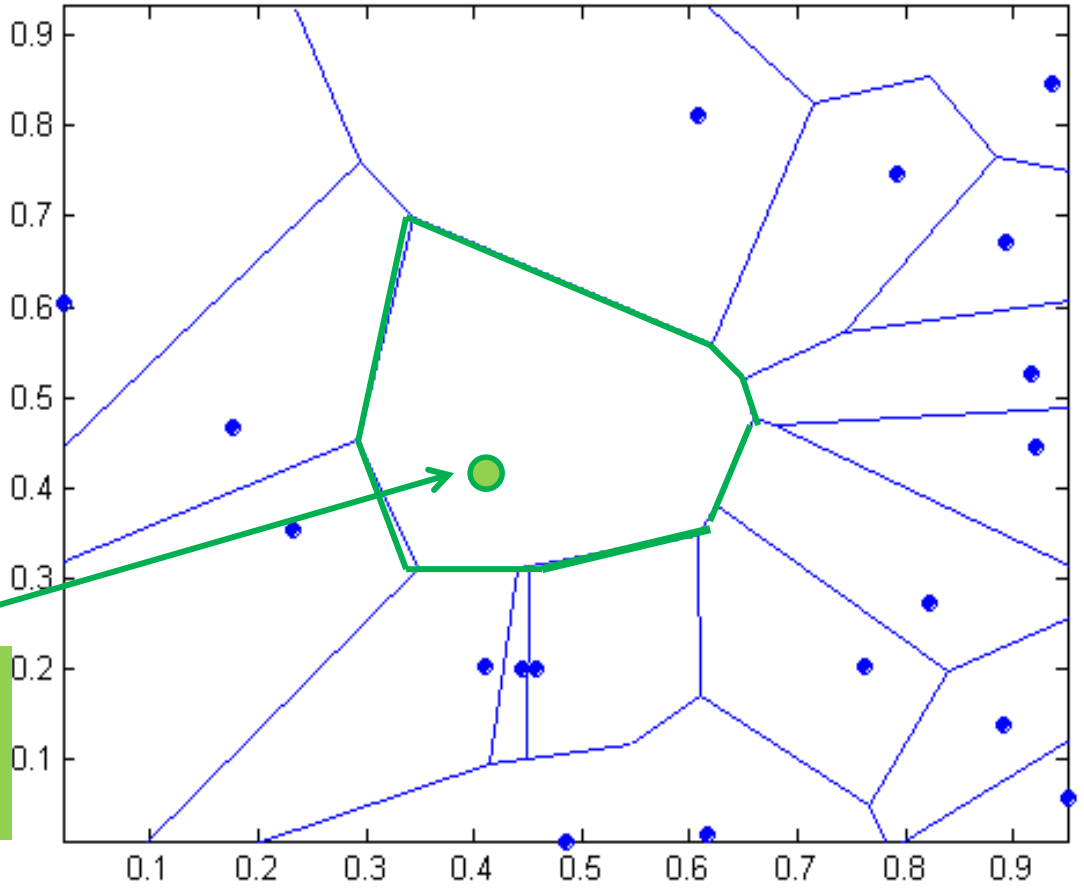
Euclidean Distance L^2

$$L^1 = |x_2 - x_1| + |y_2 - y_1|$$

$$L^2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

1 nearest-neighbor

Voronoi Diagram defines the classification boundary



The area takes the class of the green point

Nearest Neighbor Classification

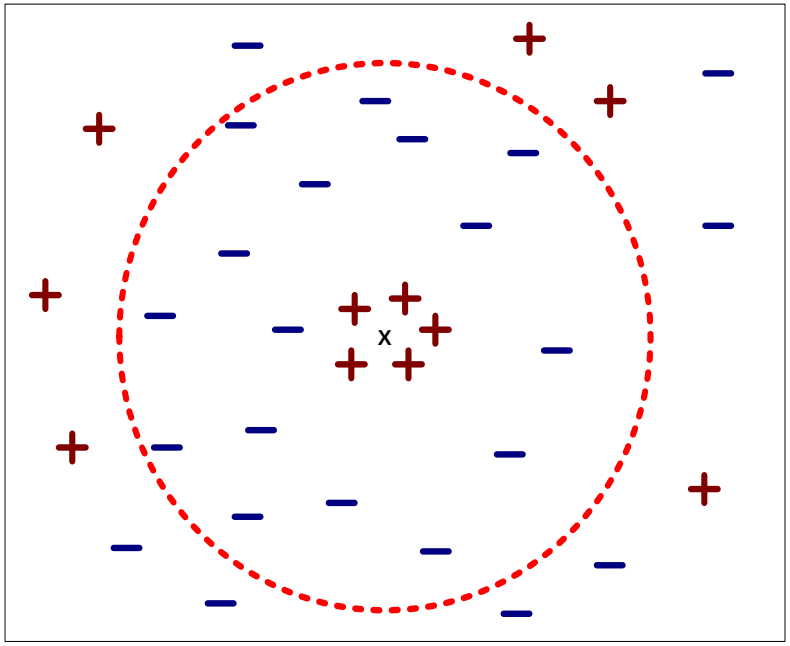
- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



Nearest Neighbor Classification…

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

    - What happens when you calculate distance?

Nearest neighbor Classification…

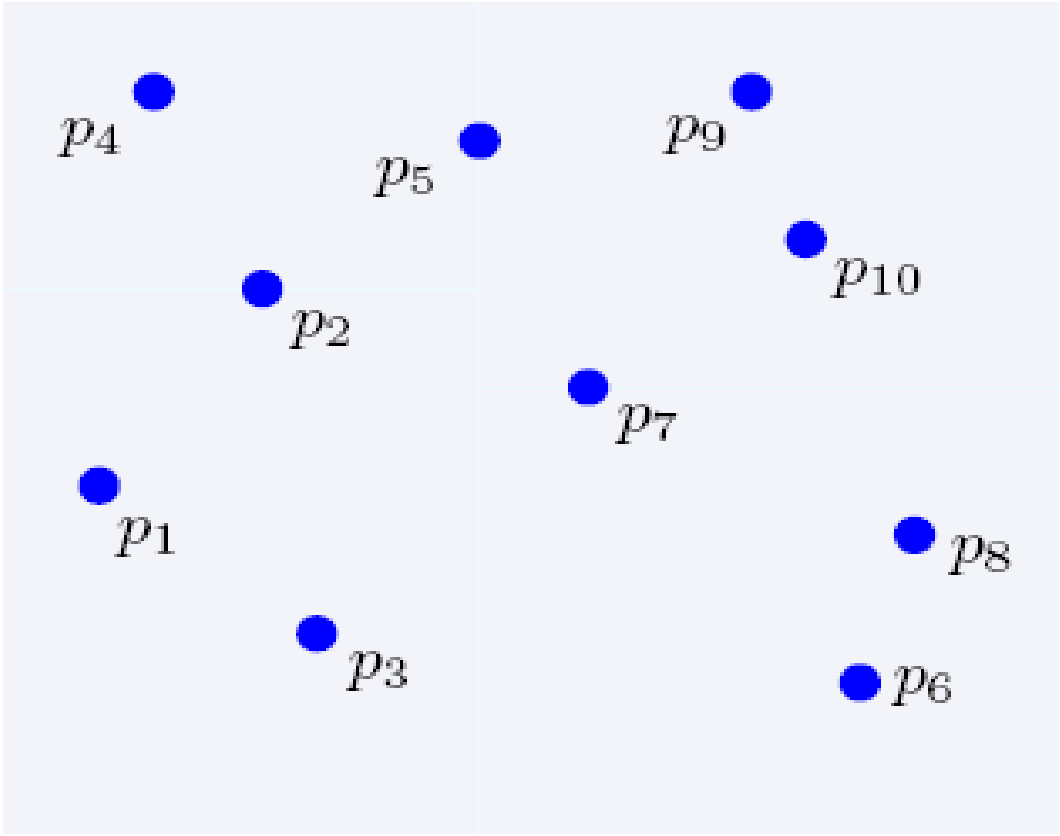
- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction and rule-based systems
- Classifying unknown records is relatively expensive
  - Naïve algorithm: O(n)
  - Need for structures to retrieve nearest neighbors fast.
    - The Nearest Neighbor Search problem.

Nearest Neighbor Search

- Two-dimensional kd-trees
  - A data structure for answering nearest neighbor queries in R^2

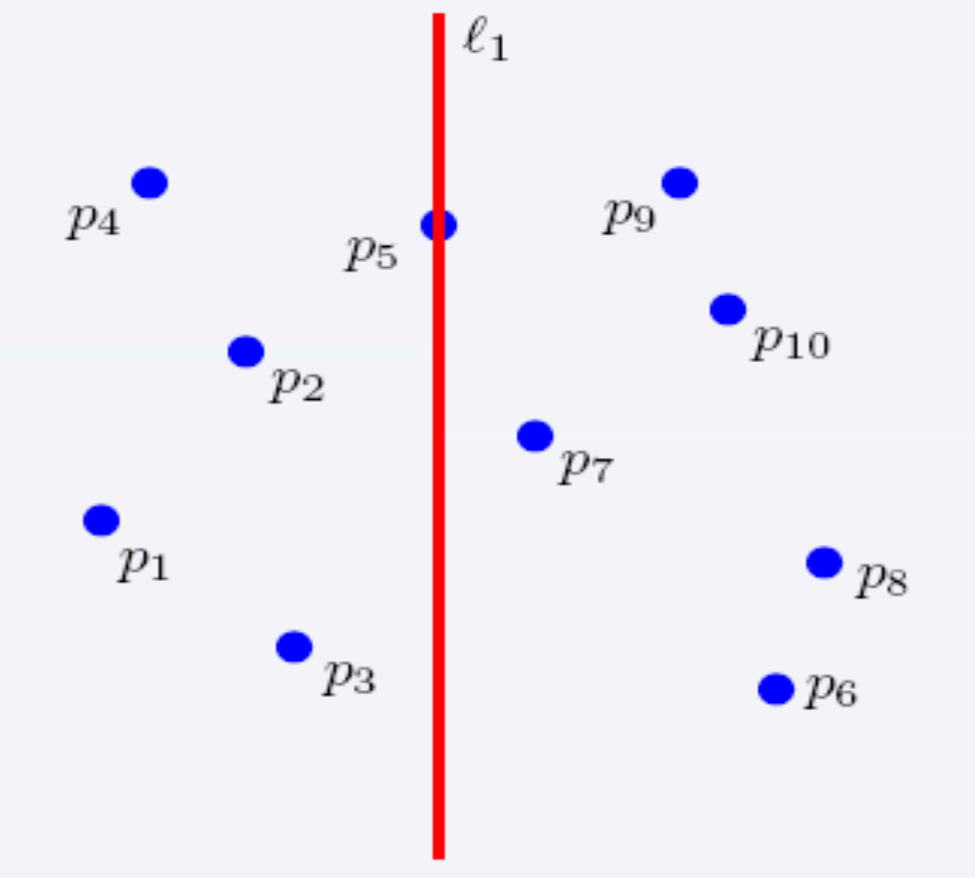
- kd-tree construction algorithm
  - Select the x or y dimension (alternating between the two)
  - Partition the space into two with a line passing from the median point
  - Repeat recursively in the two partitions as long as there are enough points

Nearest Neighbor Search



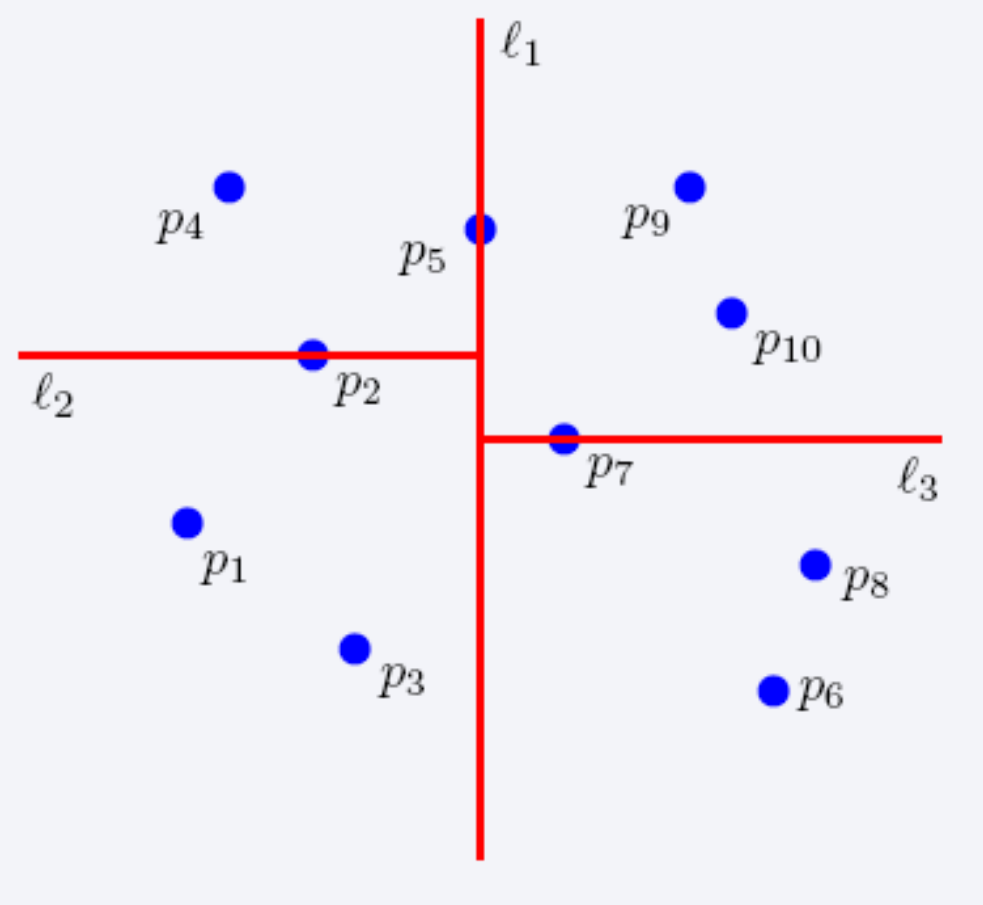
2-dimensional kd-trees

Nearest Neighbor Search



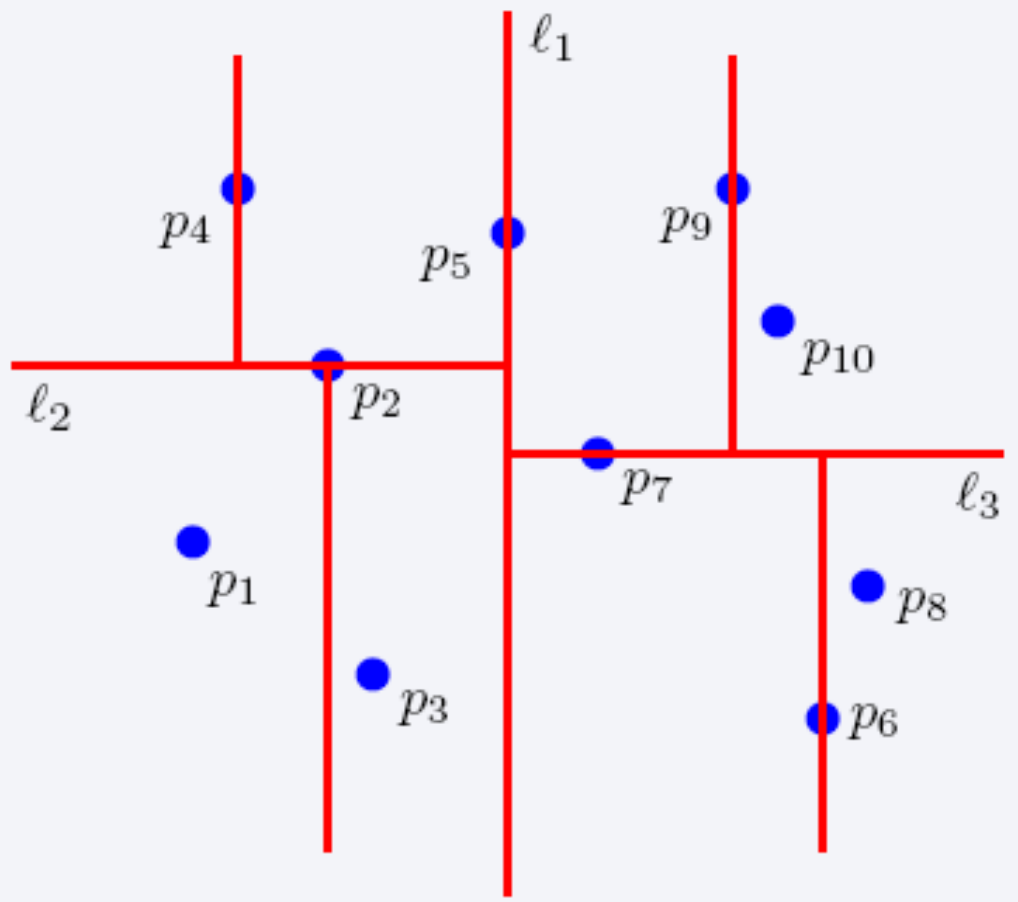
2-dimensional kd-trees

Nearest Neighbor Search



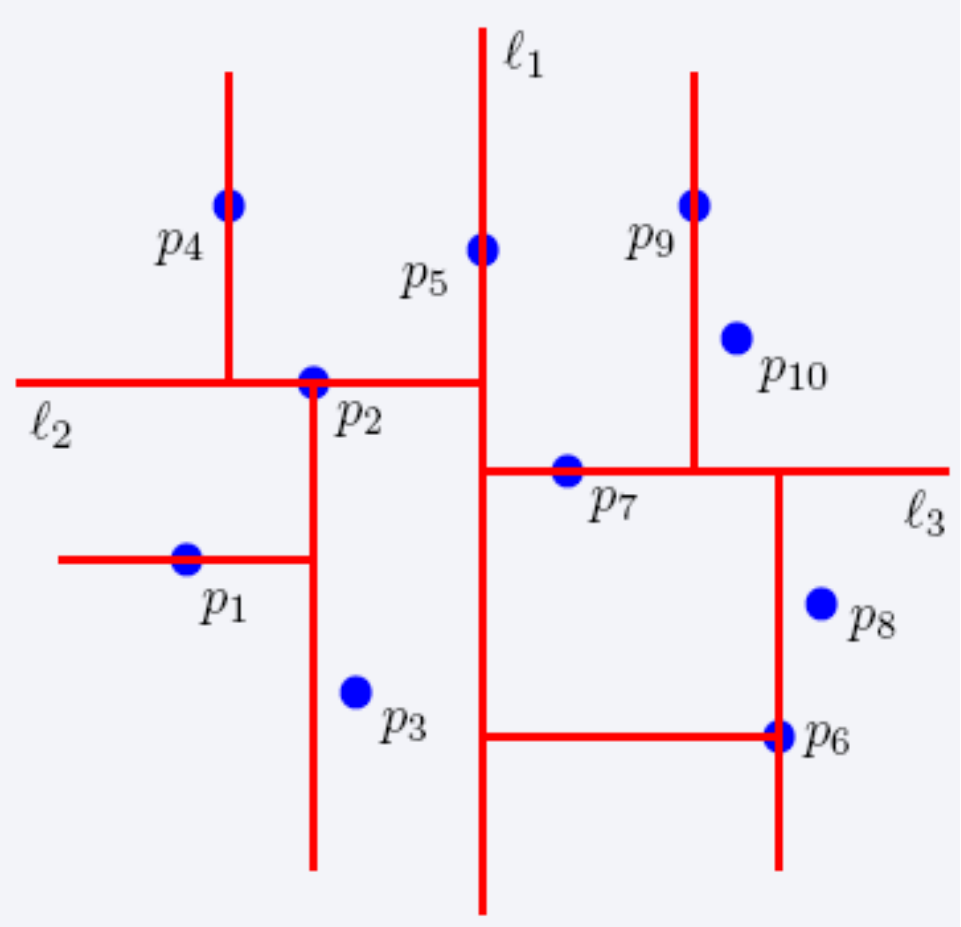
2-dimensional kd-trees

Nearest Neighbor Search



2-dimensional kd-trees

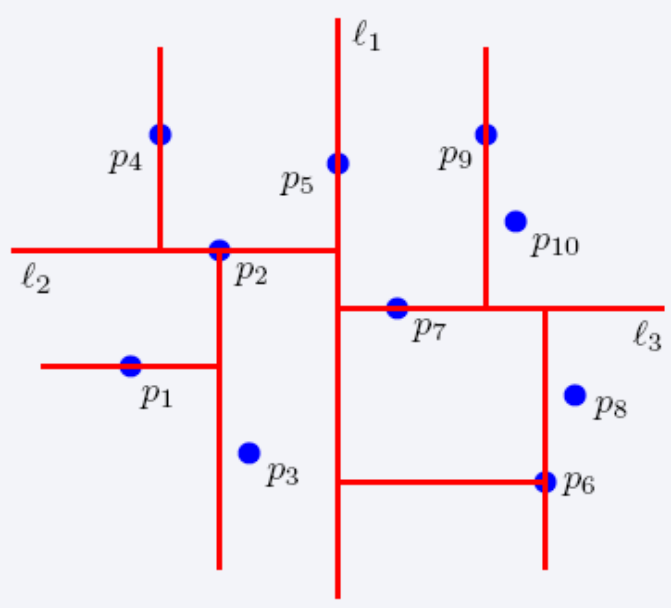
Nearest Neighbor Search

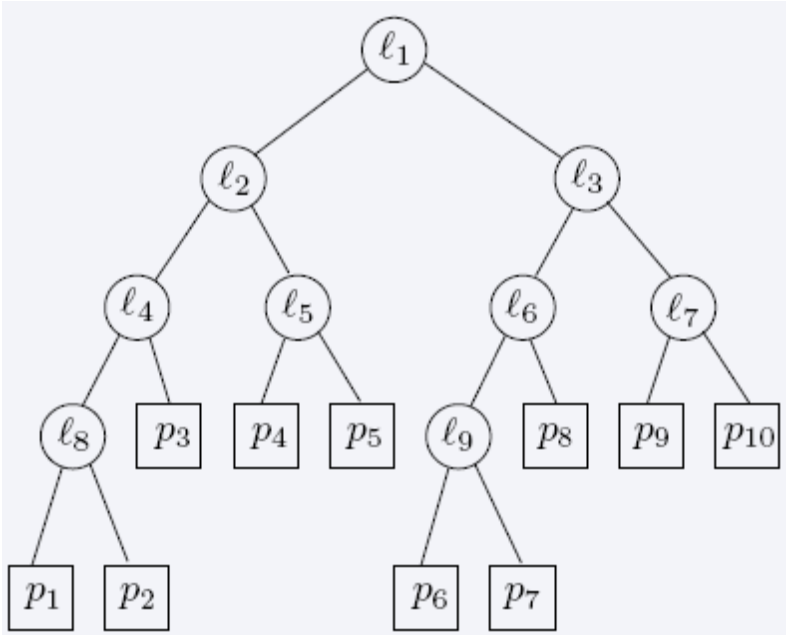


2-dimensional kd-trees

Nearest Neighbor Search

2-dimensional kd-trees

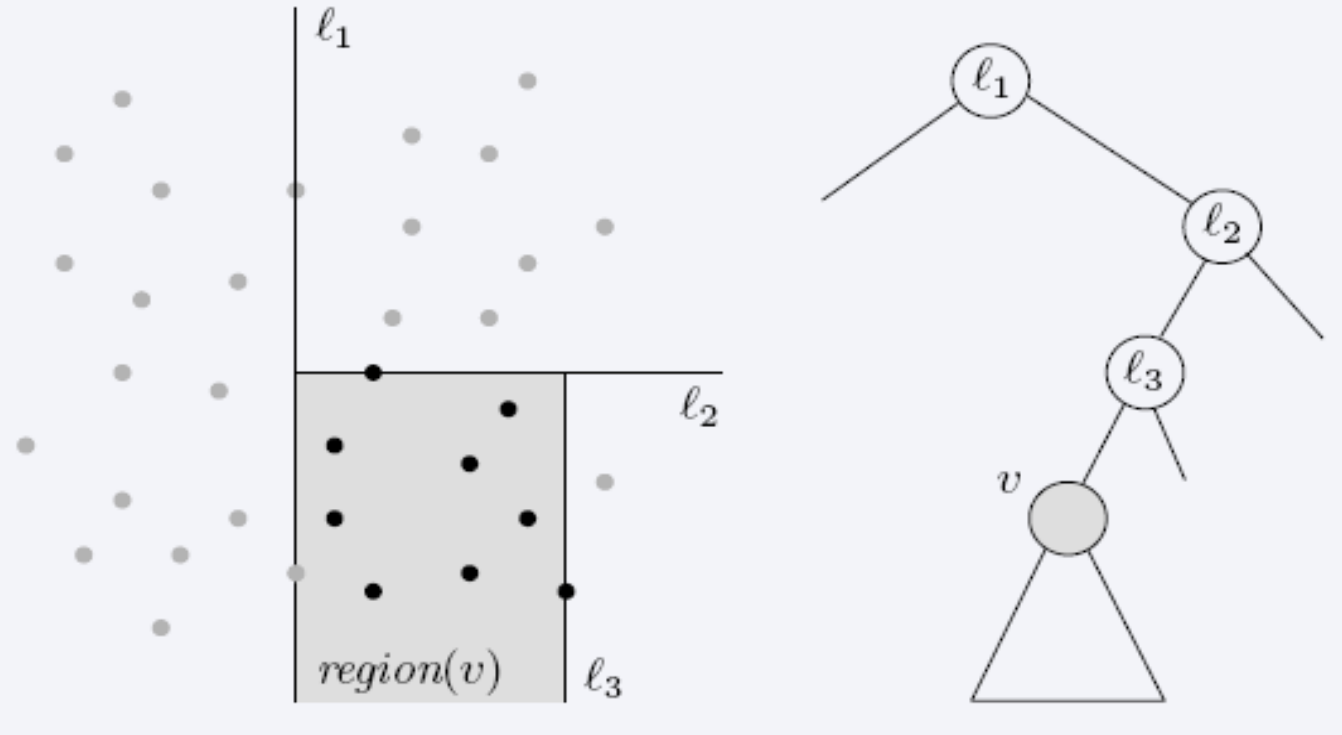




Nearest Neighbor Search

2-dimensional kd-trees

region(u) – all the black points in the subtree of u



Nearest Neighbor Search

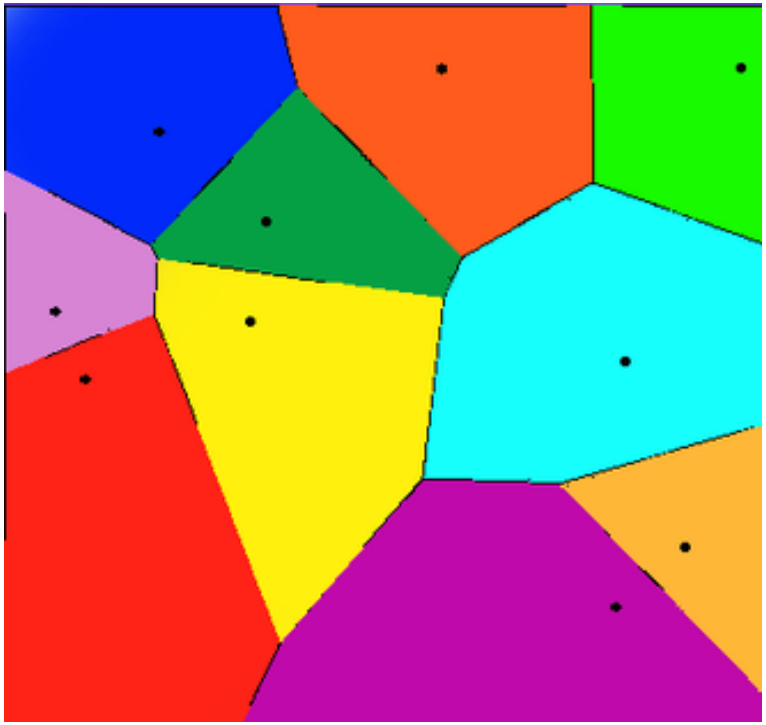
2-dimensional kd-trees

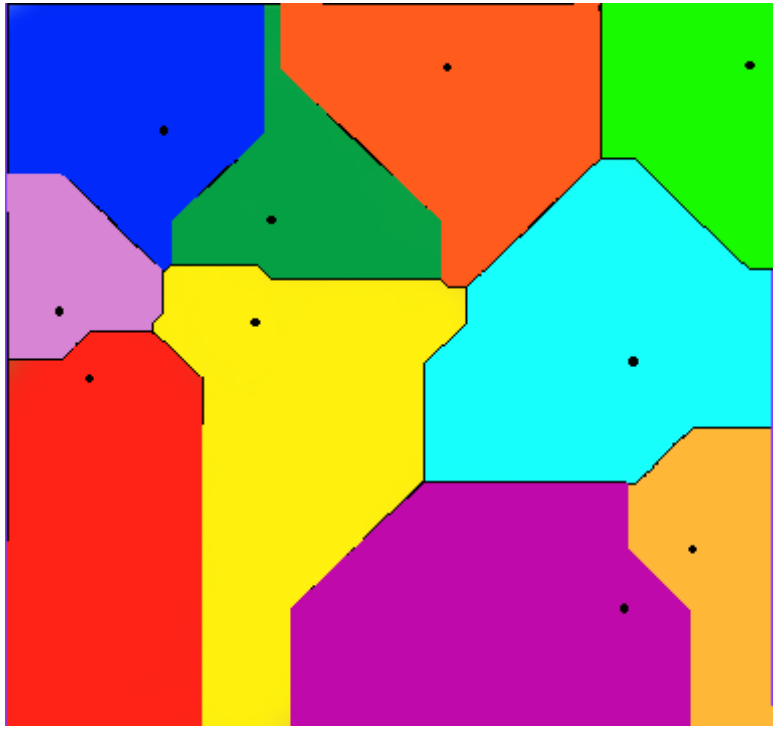
- A binary tree:
  - Size **O(n)**
  - Depth **O(logn)**
  - Construction time **O(nlogn)**
  - Query time: worst case **O(n),** but for many cases **O(logn)**

Generalizes to d dimensions

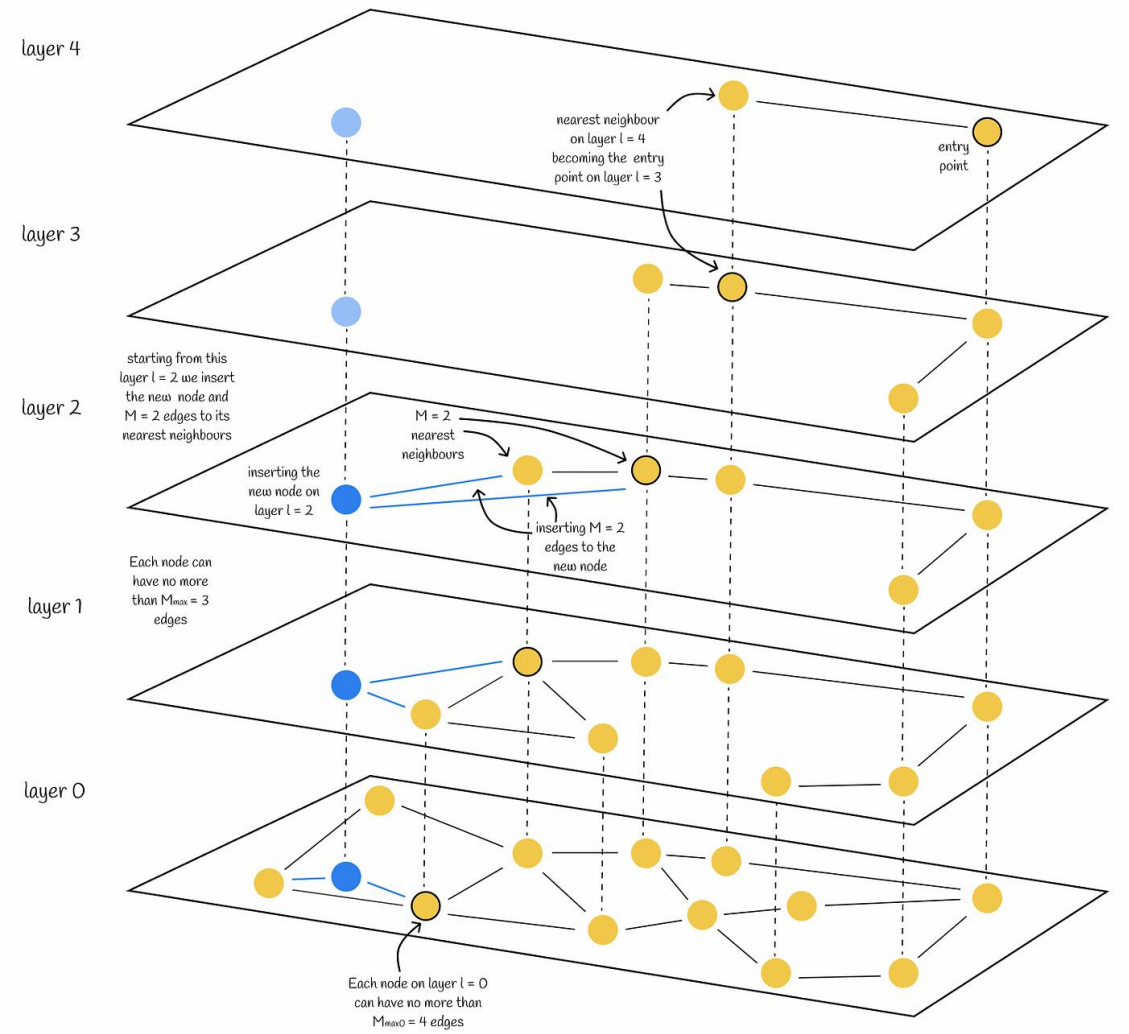
Nearest Neighbor Search

- Can also use the Voronoi diagram
  - Generalizes to d dimensions
  - Need to generalize for k nearest neighbors





Hierarchical Navigable Small Worlds



Used in vector databases

approaches Log N

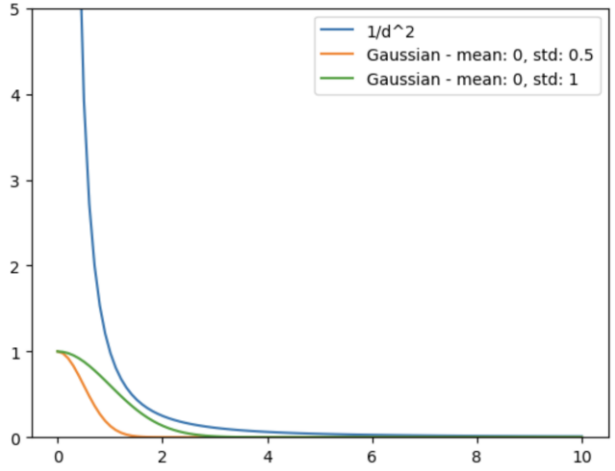
outperforms KD-tree and brute force

k-Nearest Neighbor (cont)

- Usually do distance weighted voting where each nearest neighbor vote is scaled inversely to its distance
- Inverse of distance squared is a common weight

$$w_i = \frac{1}{dist(x_q, x_i)^2}$$

- Gaussian is another common distance weight
- In this case the *k* value is more robust, could let *k* be even and/or be larger (even all points if desired), because the more distant points have negligible influence
- *w* = 1 for the non-weighted version



Challenge Question - *k*-Nearest Neighbor

- Assume the following data set
- Assume a new point (2, 6)
  - For nearest neighbor distance use Manhattan distance
  - What would the output be for 3-nn with no distance weighting? What is the total vote?
  - What would the output be for 3-nn with squared inverse distance weighting? What is the total vote?

A. A A
B. A B
C. B A
D. B B
E. None of the above

x	*y*	*Label*
1	5	A
0	8	B
9	9	B
10	10	A

$$w_i = \frac{1}{dist(x_q, x_i)^2}$$

Challenge Question - *k*-Nearest Neighbor

- Assume the following data set
- Assume a new point (2, 6)
  - For nearest neighbor distance use Manhattan distance
  - What would the output be for 3-nn with no distance weighting? What is the total vote? – B wins with vote 2 out of 3
  - What would the output be for 3-nn with distance weighting? What is the total vote? A wins with vote .25 vs B vote of .0625+.01=.0725

x	*y*	*Label*	*Distance*	*Weighted Vote*
1	5	A	1 + 1 = 2	$1/2^2 = .25$
0	8	B	2 + 2 = 4	$1/4^2 = .0625$
9	9	B	7 + 3 = 10	$1/10^2 = .01$
10	10	A	8 + 4 = 12	$1/12^2 = .0069$

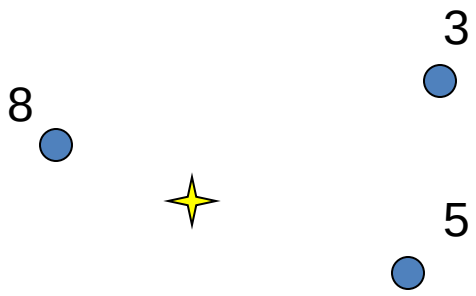
Regression with *k*-nn

- Can do regression by letting the output be the mean of the *k* nearest neighbors
- Can also do weighted regression by letting the output be the weighted mean of the *k* nearest neighbors
- For distance weighted regression

$$\hat{f}(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \qquad w_i = \frac{1}{dist(x_q, x_i)^2}$$

- Where $f(x)$ is the output value for instance x
- $w = 1$ for non-weighted

Regression Example



$$\hat{f}(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

$$w_i = \frac{1}{dist(x_q, x_i)^2}$$

- What is the value of the new instance?
- Assume dist(x_q, n_8) = 2, dist(x_q, n_5) = 3, dist(x_q, n_3) = 4
- $f(x_q) = (8/2^2 + 5/3^2 + 3/4^2)/(1/2^2 + 1/3^2 + 1/4^2) = 2.74/.42 = 6.5$
- The denominator renormalizes the value

k-Nearest Neighbor Homework

- Assume the following training set
- Assume a new point (.5, .2)
  - Use Manhattan distance and show work
  - What would the output class for 3-nn be with no distance weighting?
  - What would the output class for 3-nn be with squared inverse distance weighting?
  - What would the 3-nn regression value be for the point if we used the regression values rather than class labels? Show results for *both* no distance weighting and squared inverse distance weighting.

x	*y*	*Class Label*	*Regression Label*
.3	.8	A	.6
-.3	1.6	B	-.3
.9	0	B	.8
1	1	A	1.2

Attribute Weighting

- Normalize Features!

- One of the main weaknesses of nearest neighbor is irrelevant features, since they can dominate the distance
  - Example: assume 2 relevant and 10 irrelevant features
- Most learning algorithms weight the attributes
  - MLP and Decisions Trees do higher order weighting of features

- Could do attribute weighting - No longer lazy evaluation since you need to come up with a portion of your hypothesis (attribute weights) before generalizing

- Still an open area of research
  - Higher order weighting – 1st order helps, but not enough
  - Even if all features are relevant features, all distances become similar as number of features increases, since not all features are relevant at the same time, and the currently irrelevant ones can dominate distance
  - An issue with all pure distance based techniques, need higher-order weighting to ignore *currently* irrelevant features
  - Dimensionality reduction can be useful (feature pre-processing, PCA, NLDR, etc.)

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T Q(\boldsymbol{x} - \boldsymbol{y}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x} - \boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_{\boldsymbol{x}}$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \mathrm{sign}(x_i - x_j)\,\mathrm{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions.
($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

Value Difference Metric

- Distance of two attribute values is a measure of how similar they are in inferring the output class

- Assume a 2-output class task (A, B)
- Attribute 1 = Shape (Round, Square, Triangle, etc.)
- 10 total round instances
  - 6 class A and 4 class B
- 5 total square instances
  - 3 class A and 2 class B
- Since both attribute values suggest the same probabilities for the output class, the distance between Round and Square would be 0
  - If triangle and round suggested very different outputs, triangle and round would have a large distance

Value Difference Metric (VDM)

[Stanfill & Waltz, 1986]

Providing appropriate distance measurements for nominal attributes.

$$vdm_a(x,y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2$$

$N_{a,x}$ = # times attribute a had value x

$N_{a,x,c}$ = # times attribute a had value x and class was c

C = # output classes

Two values are considered closer if they have more similar classifications, i.e., if they have more similar correlations with the output classes.

VDM Example

$$vdm_a(x,y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2$$

$N_{a,x}$ = # times attribute a had value x
$N_{a,x,c}$ = # times attribute a=x and class was c
C = # output classes

Color →	Class
red	0
green	0
red	1
blue	0
blue	1
red	1
blue	1
blue	1

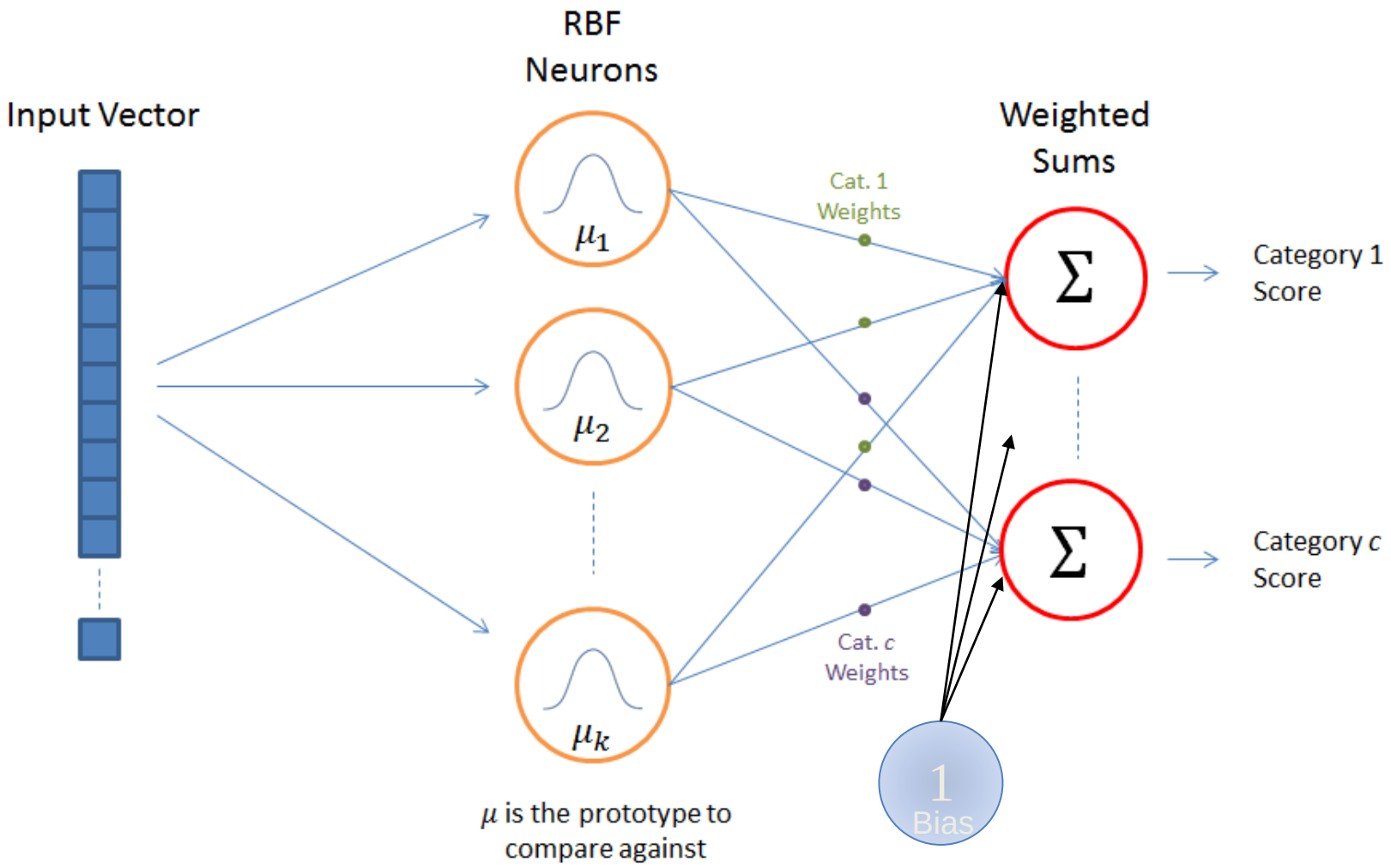
x	$N_{1,x}$	class	$N_{1,x,c}$	$N_{1,x,c}/N_{1,x}$
red	3	0:	1	1/3=.33
		1:	2	2/3=.67
green	1	0:	1	1/1=1.0
		1:	0	0/1=0.0
blue	4	0:	1	1/4=.25
		1:	3	3/4=.75

vdm_{color}(red,green) = 0.889
vdm_{color}(red,blue) = 0.014
vdm_{color}(green,blue)=1.125

I got these values

0.8978
0.0128
1.125

Radial Basis Function Networks

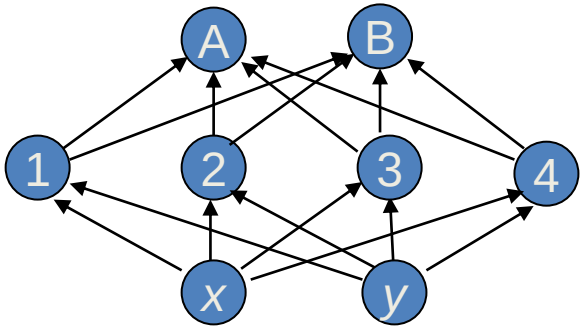


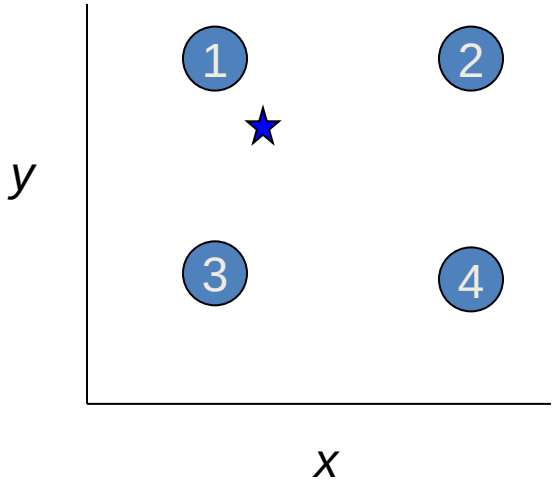
[redacted]

Where $f(x)$ is the output of each linear output node, K is RBF (kernel function) and d is distance

Radial Basis Function (RBF) Networks

- One linear output node per class – with weights and bias
- Each hidden (prototype RBF neurons) node computes the distance from itself to the input instance (Gaussian is common)
  - – *not* like an MLP hidden node
- An arbitrary number of prototype nodes form a hidden layer in the Radial Basis Function network – prototype nodes typically non-adaptive
- The prototype layer expands the input space into a new prototype space. Translates the data set into a new set with more features
- Output layer weights are learned with the linear model delta rule
  - Not a preset label vote like in k-nearest neighbor
- Thus, output nodes learn 1st order prototype weightings for each class

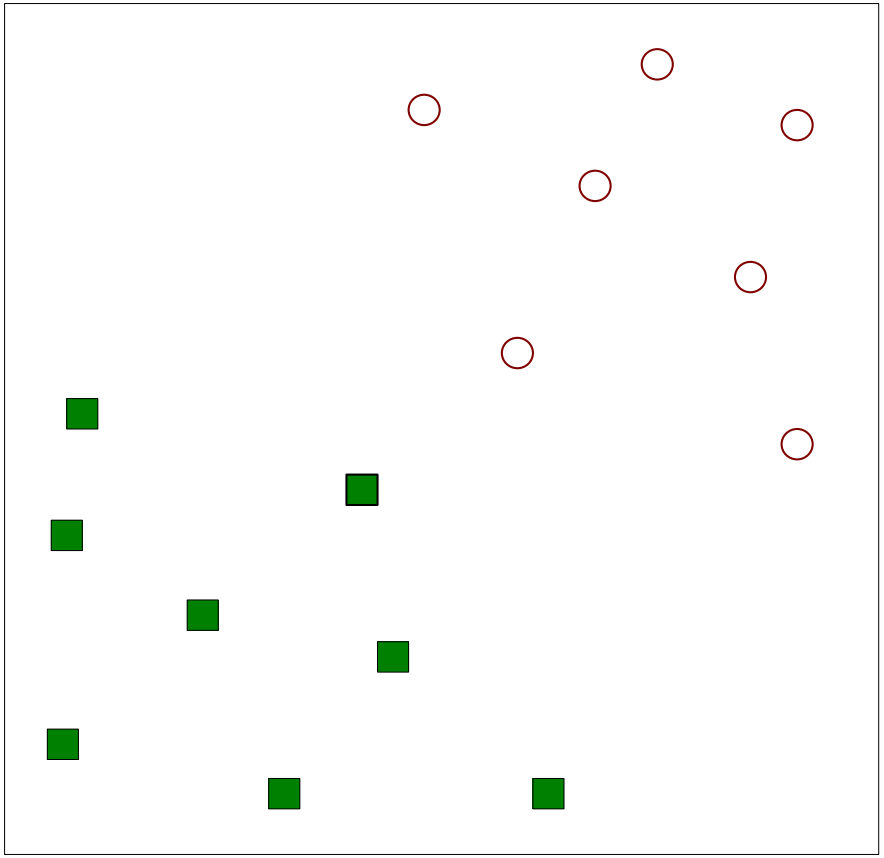




$$\Delta w_i = c(t - net)x_i$$

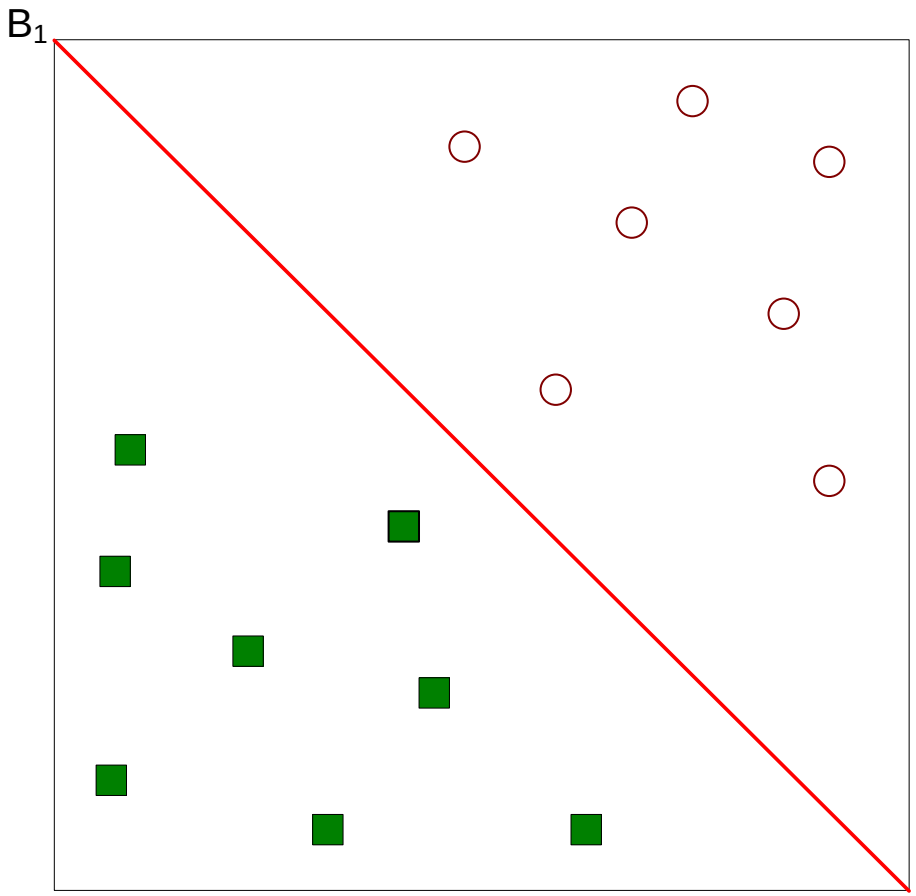
SUPPORT VECTOR MACHINES

Support Vector Machines



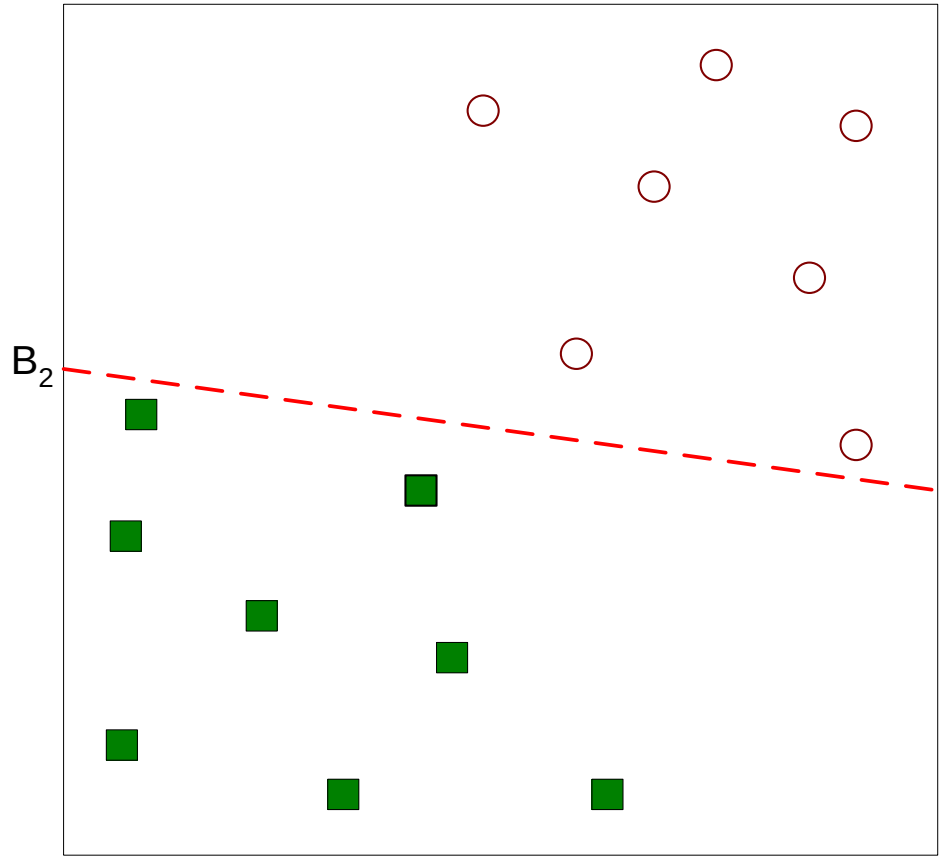
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



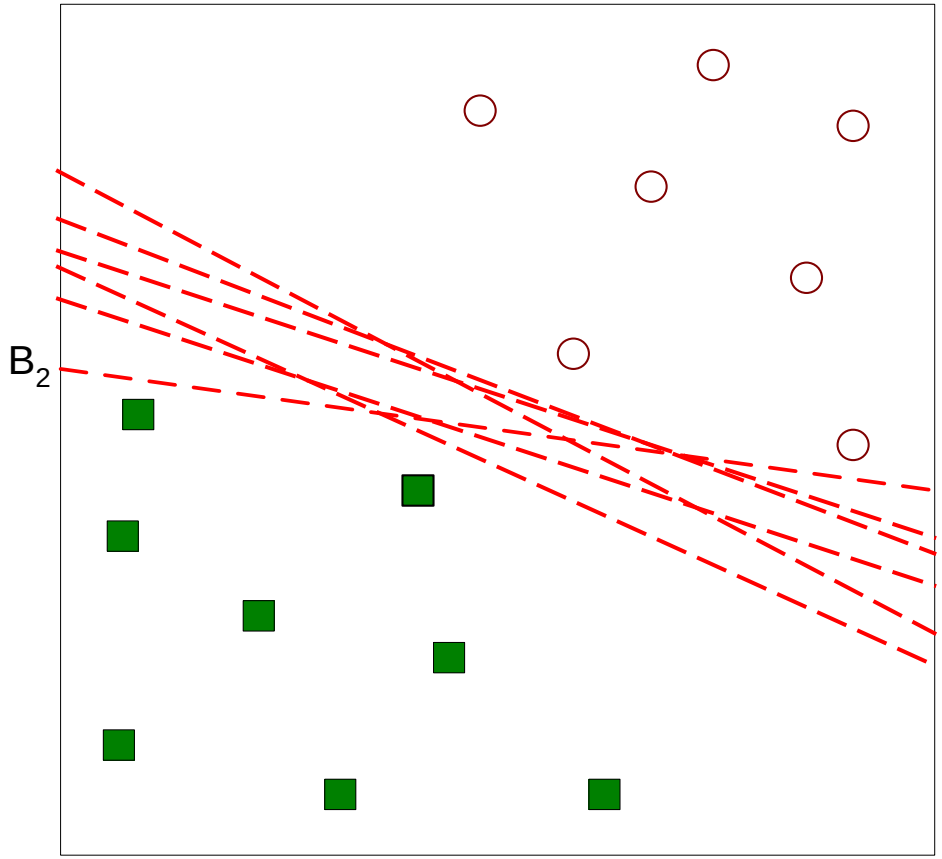
- One Possible Solution

Support Vector Machines



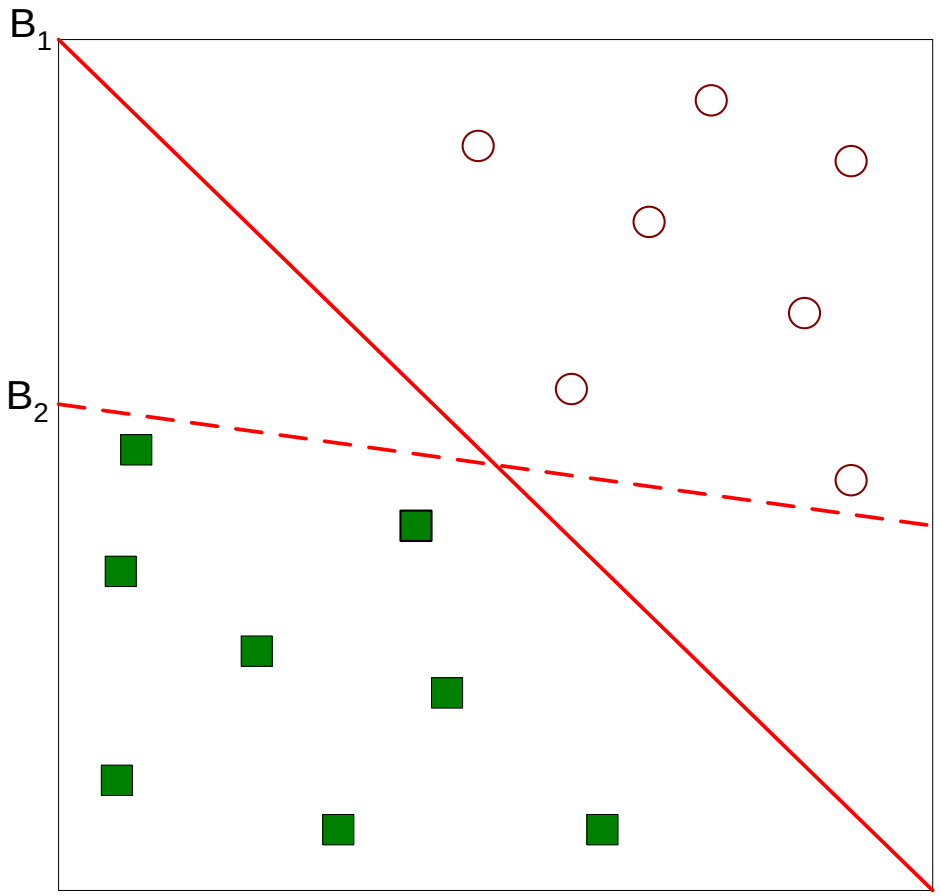
- Another possible solution

Support Vector Machines



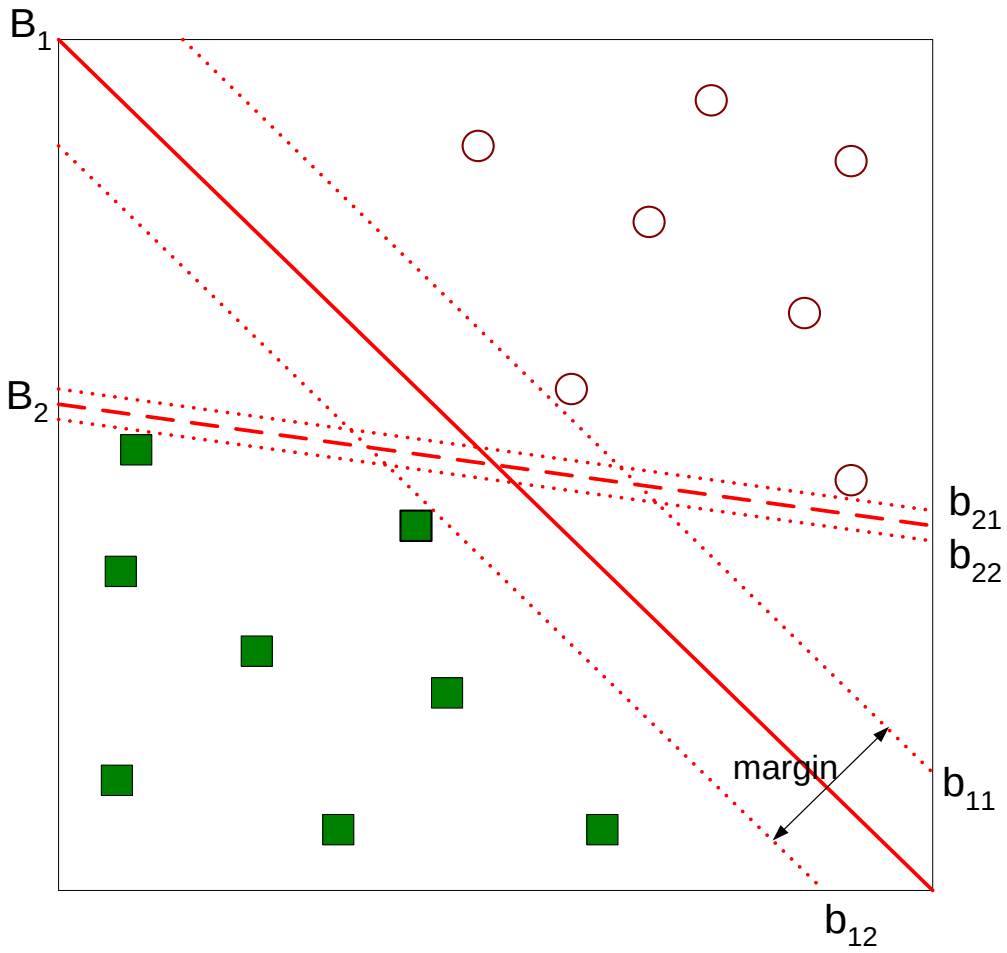
- Other possible solutions

Support Vector Machines



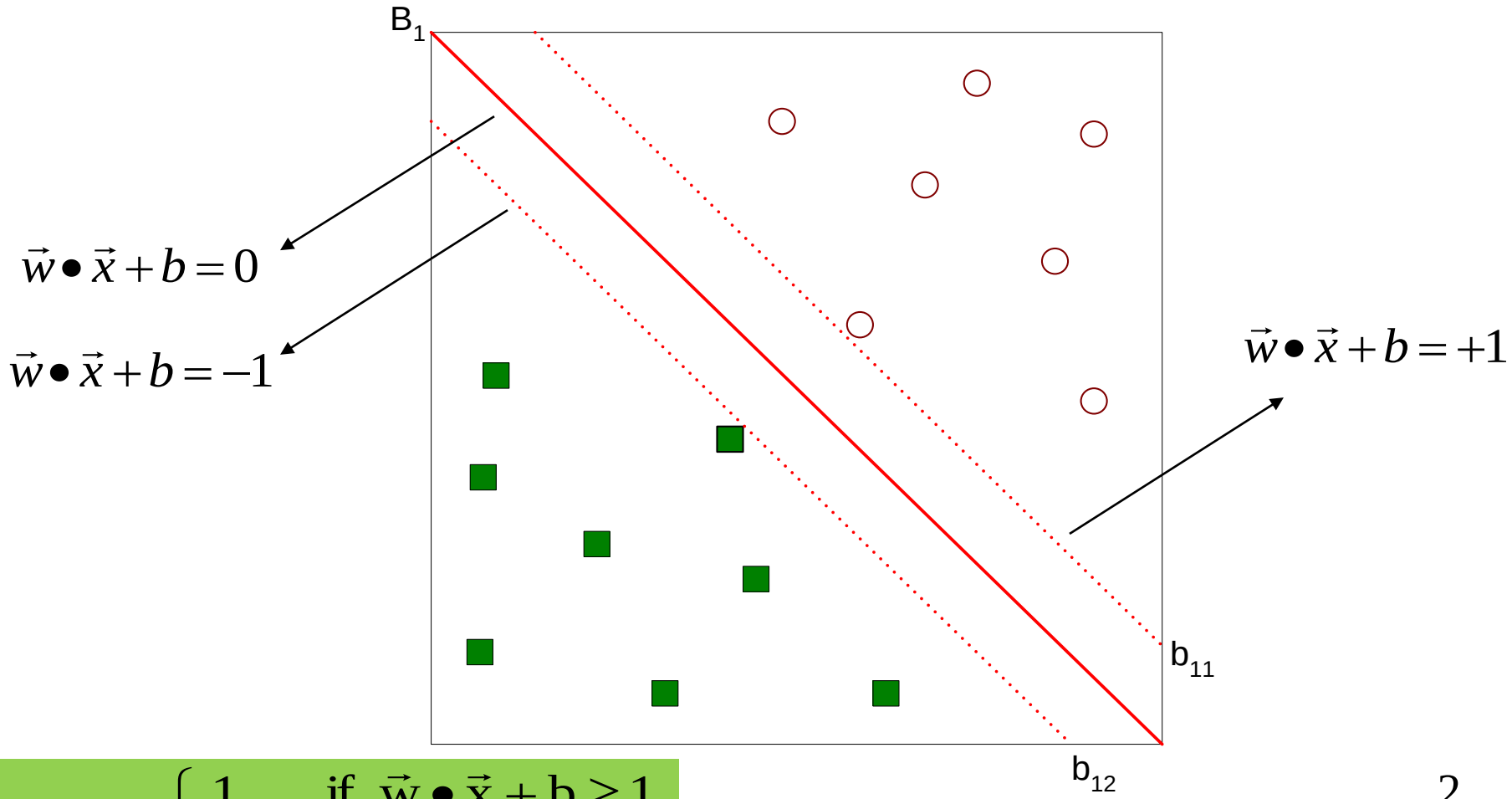
- Which one is better? B1 or B2?
- How do you define better?

Support Vector Machines



- Find hyperplane maximizes the margin => B1 is better than B2. Why?

Support Vector Machines



$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \leq -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\| \vec{w} \|}$$

Support Vector Machines

- We want to maximize: $\text{Margin} = \frac{2}{\| \vec{w} \|^2}$
  - Which is equivalent to minimizing: $L(w) = \frac{\| \vec{w} \|^2}{2}$
  - But subjected to the following constraints:

$$\vec{w} \cdot \vec{x_i} + b \geq 1 \text{ if } y_i = 1$$
$$\vec{w} \cdot \vec{x_i} + b \leq -1 \text{ if } y_i = -1$$

    - This is a constrained optimization problem
      - Numerical approaches to solve it (e.g., quadratic programming)

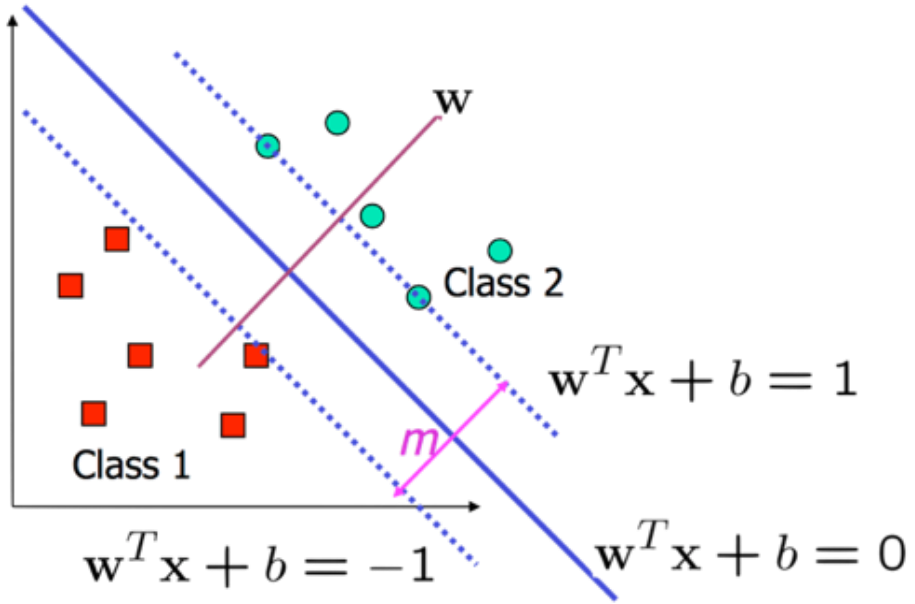
Maximizing Margins (cont.)

- Notice that maximizing the distance of ***all points*** to the decision boundary, is exactly the same as maximizing the distance to the ***closest points***.
- The points closest to the decision boundary are called ***support vectors***.

Maximizing Margins Illustration

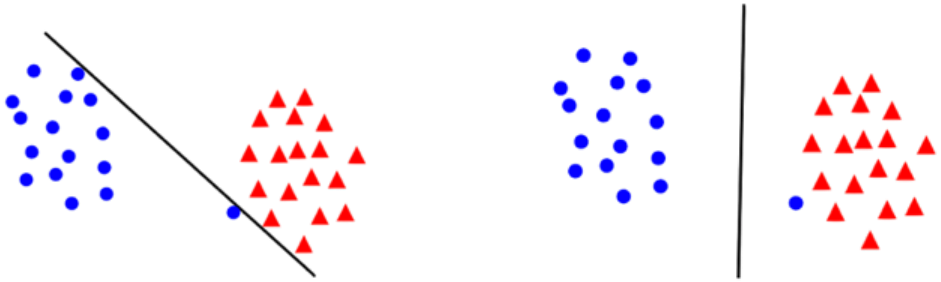
- For points on planes $w^T x + b = \pm 1$, their distance to the decision boundary is $\pm 1/\|w\|$.

  So we can define the ***margin*** of a decision boundary as the distance to its support vectors, $m = 2/\|w\|$.



Geometry of Data

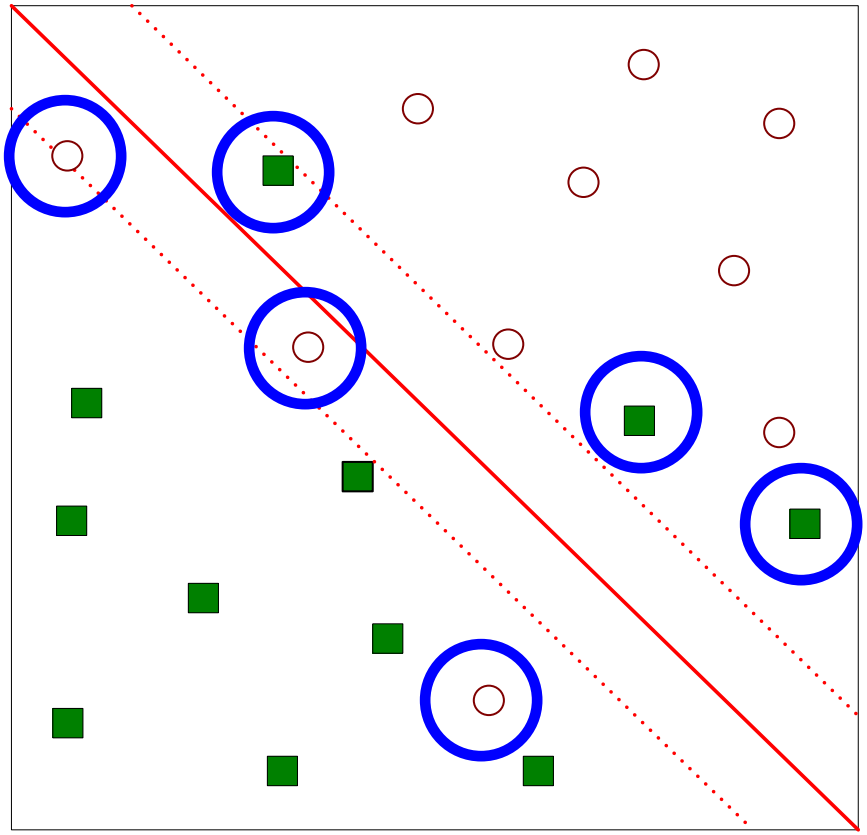
- Maximizing the margin is a good idea as long as we assume that the underlying classes are linear separable and that the data is noise free.
- If data is noisy, we might be sacrificing generalizability to minimize classification error with a very narrow margin:



- With every decision boundary, there is a trade-off between maximizing margin and minimizing the error.

Support Vector Machines

- What if the problem is not linearly separable?



Support Vector Machines

- If the problem is not linearly separable
  - Introduce slack variables
    - Need to minimize:

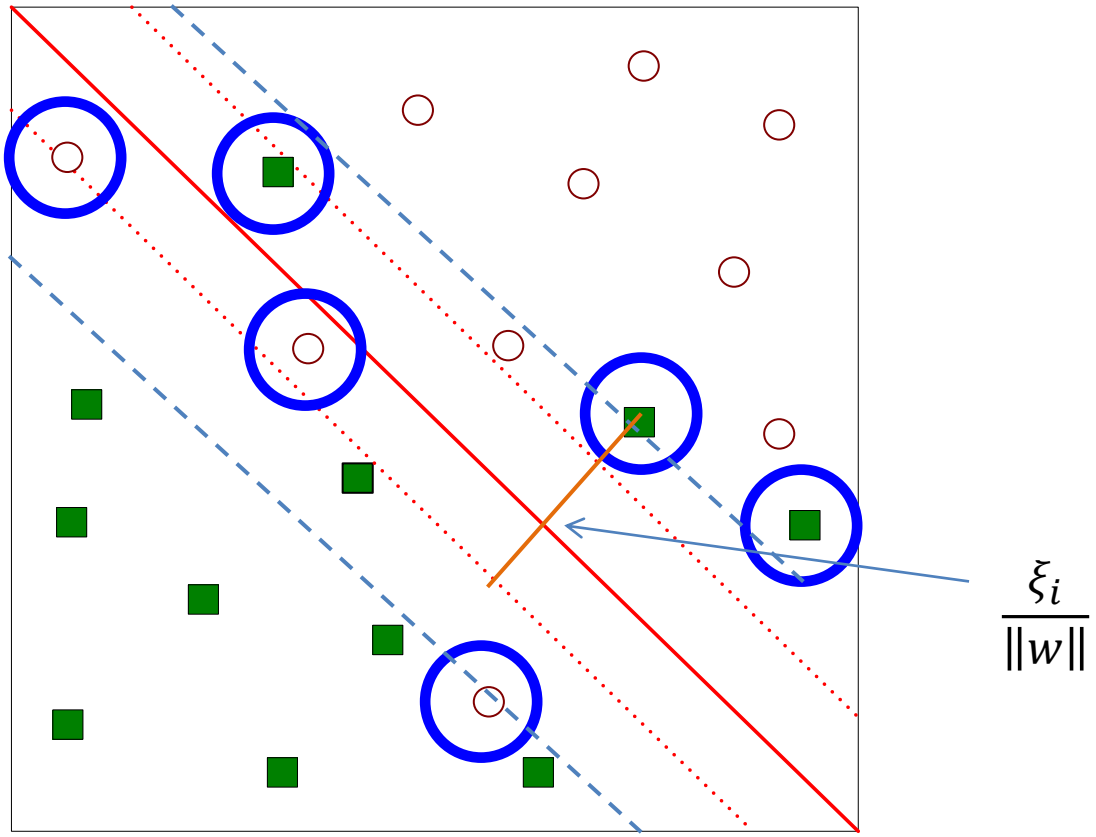
$$L(w) = \frac{\| \vec{w} \|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k \right)$$

    - Subject to:

$$\vec{w} \cdot \vec{x_i} + b \geq 1 - \xi_i \text{ if } y_i = 1$$
$$\vec{w} \cdot \vec{x_i} + b \leq -1 + \xi_i \text{ if } y_i = -1$$

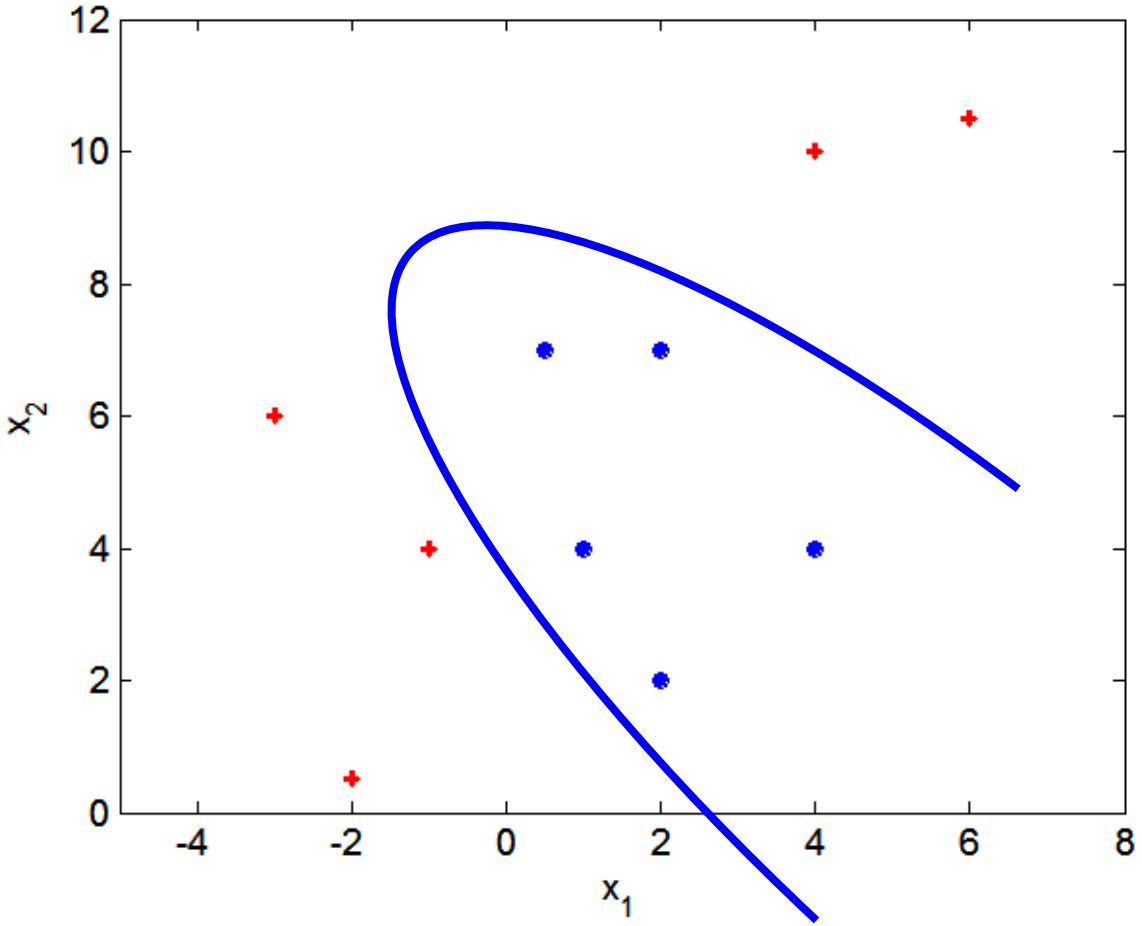
Support Vector Machines

- What if the problem is not linearly separable?



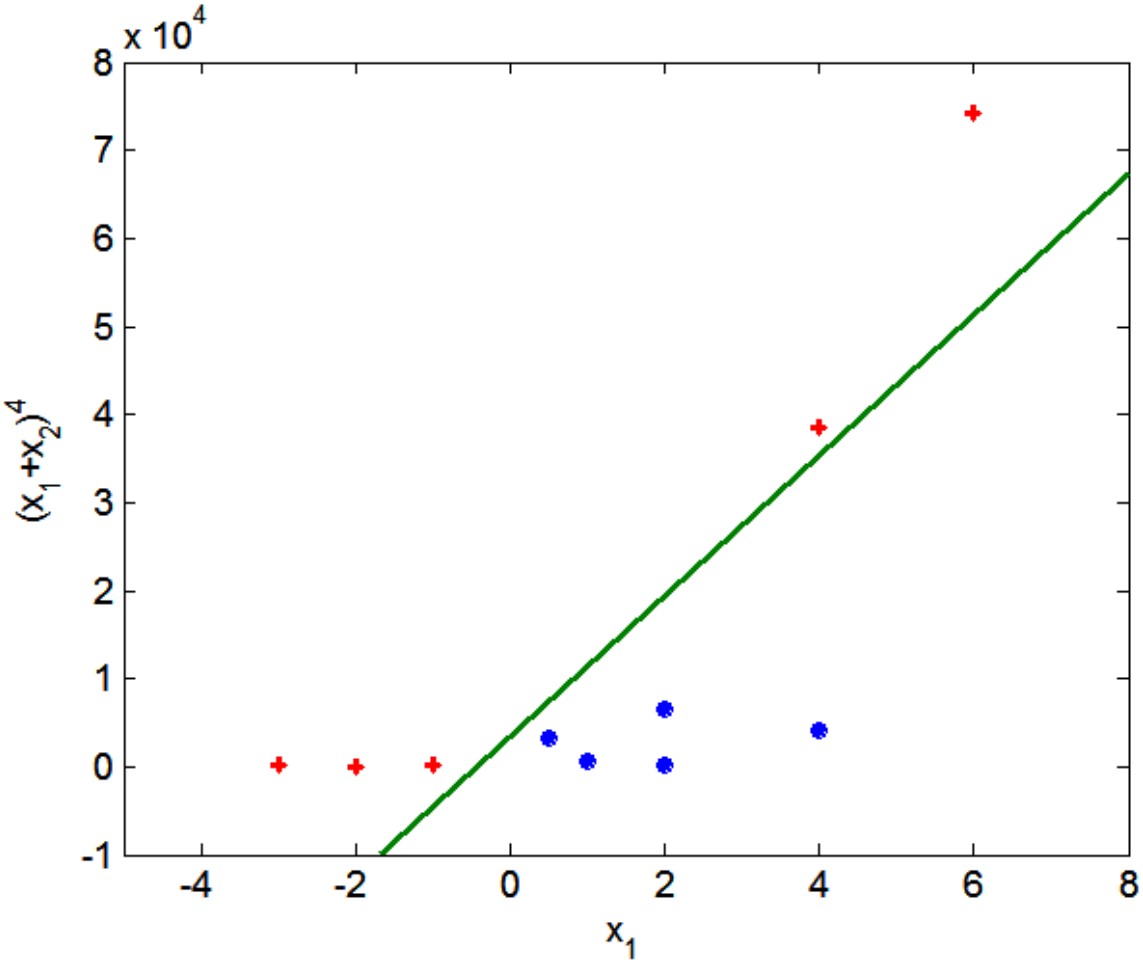
Nonlinear Support Vector Machines

- What if decision boundary is not linear?



Nonlinear Support Vector Machines

- Transform data into higher dimensional space



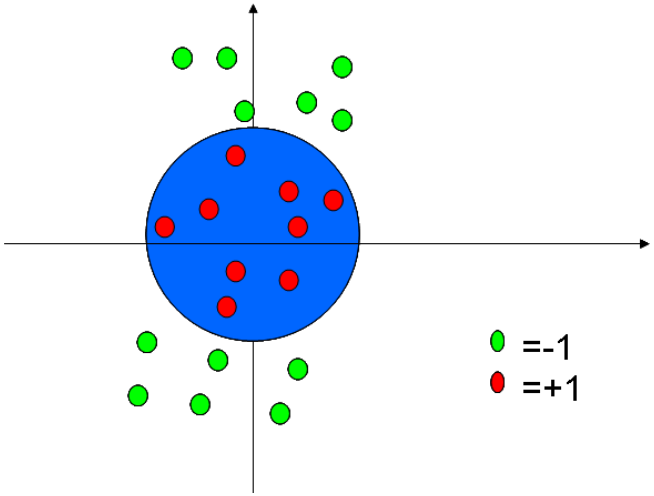
The Kernel Trick

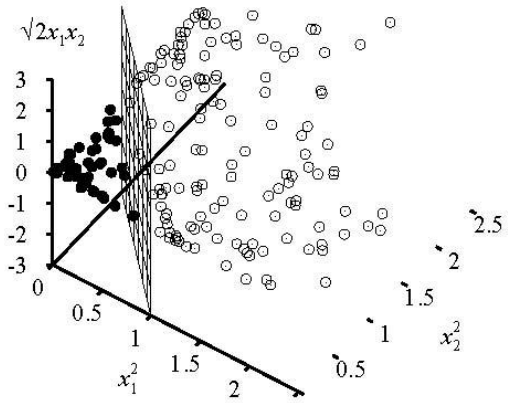
- The linear classifier relies on an inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

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

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

Kernel Trick





Data points are linearly separable in the space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$

We want to maximize $\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \left\langle F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) \right\rangle$

Define $K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle F(\mathbf{x}_i) \cdot F(\mathbf{x}_j) \right\rangle$

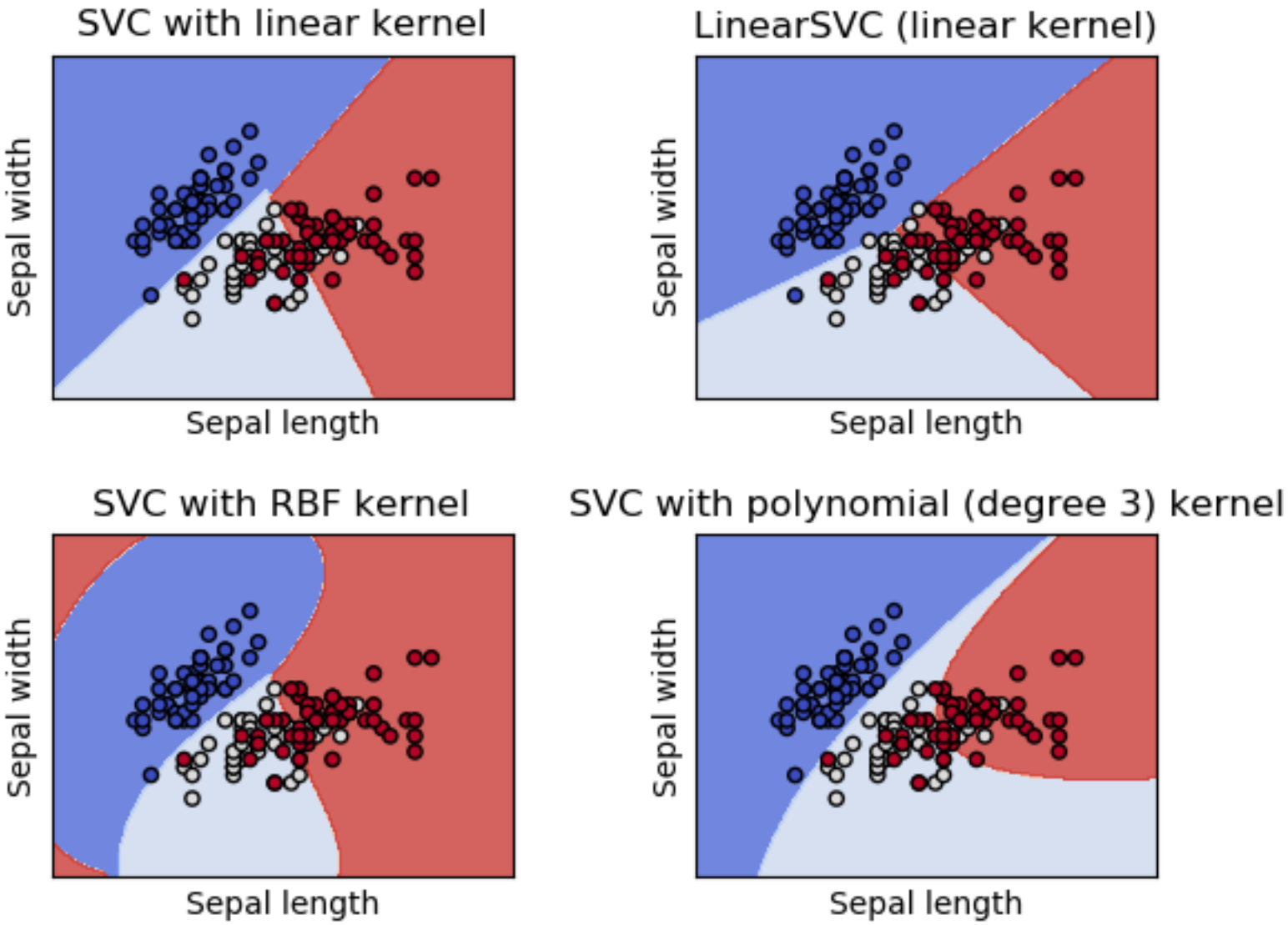
Cool thing : K is often easy to compute directly! Here,

$K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \mathbf{x}_i \cdot \mathbf{x}_j \right\rangle^2$

Other Kernels

The polynomial kernel
$K(x_i,x_j) = (x_i \bullet x_j + 1)^p$, where p is a tunable parameter.
Evaluating K only requires one addition and one exponentiation more than the original dot product.

Gaussian kernels (also called radial basis functions)
$K(x_i,x_j) = \exp(||x_i - x_j||^2/2\sigma^2)$


SVC with linear kernel
LinearSVC (linear kernel)
SVC with RBF kernel
SVC with polynomial (degree 3) kernel
Sepal width
Sepal length

sklearn.svm.SVC

class `sklearn.svm.`**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using **`LinearSVC`** or **`SGDClassifier`** instead, possibly after a **`Nystroem`** transformer or other Kernel Approximation.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

To learn how to tune SVC's hyperparameters, see the following example: Nested versus non-nested cross-validation

Read more in the User Guide.

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`. For an intuitive visualization of different kernel types see Plot classification boundaries with different SVM Kernels.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

NAÏVE BAYES CLASSIFIER

Two Approaches to Statistics

- Frequentist
  - All about probability in the long run
    - ie. Flipping a coin in the long run is 50% heads
  - Assume a null hypothesis
  - Collect data, analyze, and ask "How surprising is my result?"
  - Is it surprising enough to reject the null hypothesis?
  - Confidence intervals, P values
- Bayesian
  - Data is fixed, parameters and hypotheses are probability distributions
  - The probability distribution that summarizes what is known before is the prior distribution or just 'the prior'
  - We start with prior beliefs, and we update our beliefs based on new information

An Analogy

- I have misplaced my phone somewhere in my home. I can use my Watch to activate the beeping on my phone.
  - Problem: Which area of my home should I search?
- Frequentist Reasoning
  - I can hear the phone beeping. I also have a mental model which helps me identify the area from which the sound is coming. Therefore, upon hearing the beep, I infer the area of my home I must search to locate the phone.
- Bayesian Reasoning
  - I can hear the phone beeping. Apart from a mental model which helps me identify the area from which the sound is coming from, I also know the locations where I have misplaced the phone in the past. So, I combine my inferences using the beeps and my prior information about the locations I have misplaced the phone in the past to identify an area I must search to locate the phone. When I find the phone, I naturally update my mental model of locations that I have misplaced my phone.

Some Background Math

- A, C are random variables
  - We will often use C to represent the Class or label
- Joint Probability: P(A=a, C=c) or often P(A,C)
- Conditional Probability: P(C=c | A=a) or P(C|A)
- Relationship between Joint and Conditional
  - $P(A, C) = P(A \mid C) * P(C) = P(C \mid A) * P(A)$
- Assuming A is a set of Independent Attributes
  - $P(A) = P(a_1) * P(a_2) * P(a_3) * \ldots * P(an)$
  - $P(A \mid C) = P(a_1 \mid C) * P(a_2 \mid C) * P(a_3 \mid C) * \cdots * P(a_n \mid C)$

Bayes Classifier

- A probabilistic framework for solving classification problems
- Calculate the probability of each class and then pick the one with highest probability

- **Bayes Theorem**:

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

Likelihood of the Attribute or Attributes given the Class

Prior Probability of the Class

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

Posterior Probability of a Class given an Attribute or set of Attributes

Prior Probability of the Attribute or Attributes

Note: The right side of the equation is based on our current data

Example of Bayes Theorem

- Given:
  - A doctor knows that meningitis causes stiff neck 50% of the time
  - Prior probability of any patient having meningitis is 1/50,000
  - Prior probability of any patient having stiff neck is 1/20

- If a patient has stiff neck, what's the probability he/she has meningitis?

$$P(M \mid S) = \frac{P(S \mid M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002$$

Bayesian Classifiers

- Consider each attribute and class label as random variables

- Given a record with attributes (A_1, A_2,…,A_n)
  - Goal is to predict class C
  - Specifically, we want to find the value of C that maximizes $P(C| A_1, A_2,…,A_n)$

- Can we estimate $P(C| A_1, A_2,…,A_n)$ directly from data?

Bayesian Classifiers

- Approach:
  - compute the posterior probability $P(C \mid A_1, A_2, \ldots, A_n)$ for all values of C using the Bayes theorem

$$P(C \mid A_1 A_2 \ldots A_n) = \frac{P(A_1 A_2 \ldots A_n \mid C) P(C)}{P(A_1 A_2 \ldots A_n)}$$

  - Choose value of C that maximizes

    $P(C \mid A_1, A_2, \ldots, A_n)$

  - Equivalent to choosing value of C that maximizes

    $P(A_1, A_2, \ldots, A_n \mid C)\ P(C)$

- How to estimate $P(A_1, A_2, \ldots, A_n \mid C)$?

Bayesian Classifiers

- In Machine Learning, we are often interested in determining the best hypothesis from some space H, given the the observed training data D
- One way to specify what is meant by best hypothesis is to say that we demand the most probable hypothesis, given the data D together with any initial knowledge about the prior probabilities of the various hypotheses in H
  - Referred to as the Maximum a Posteriori (MAP) estimation

- For each hypothesis $h \in H$
  - Calculate $P(h \mid D)$ // using Bayes Theorem
- Return $h_{MAP} = \text{argmax}_{h \in H} P(h \mid D)$

- Guaranteed “best” BUT often impractical for large hypothesis spaces: mainly used as a standard to gauge the performance of other learners

Naïve Bayes Classifier

- Assume independence among attributes A_i when class is given:
  - $P(A_1, A_2, \ldots, A_n|C_j) = P(A_1|C_j)\, P(A_2|C_j) \cdots P(A_n|C_j)$
  - We can estimate $P(A_i|\, C_j)$ for all A_i and C_j.
  - New point X is classified to C_j if

$$P(C_j|X) = P(C_j) \prod_i P(A_i|C_j)$$

is maximal.

Naïve Bayes Classifier

- Involves a learning step in which the various $P(c_j)$ and $P(a_i|c_j)$ terms are estimated
  - Based on frequencies in the training data
- Store the values in a table
- Use the table to perform calculations of probabilities

How to Estimate Probabilities from Data?

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Class: $P(C) = N_c/N$
  - e.g., P(No) = 7/10,
    P(Yes) = 3/10
- For discrete attributes:

$$P(A_i \mid C_k) = |A_{ik}| / N_{c_k}$$

  - where $|A_{ik}|$ is number of instances having attribute A_i and belongs to class C_k
  - Examples:

P(Status=Married|No) = 4/7
P(Refund=Yes|Yes)=0/3 = 0

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

Frequency Table		Play	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	3	2

Frequency Table		Play	
		Yes	No
Humidity	High	3	4
	Normal	6	1

Frequency Table		Play	
		Yes	No
Wind	Strong	6	2
	Weak	3	3

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

$$P(Yes|Rainy) = \frac{P(Rainy|Yes)P(Yes)}{P(Rainy)}$$

Likelihood table for Outlook

Likelihood Table		Play		
		Yes	No	
Outlook	Sunny	2/9	3/5	6/14
	Overcast	4/9	0/5	4/14
	Rainy	3/9	2/5	5/14
		9/14	5/14	

P(Yes | Rainy) = 0.33 x 0.64 / 0.36 = 0.586

P(No | Rainy) = 0.4 x 0.36 / 0.36 = 0.4

Likelihood table for Humidity

Likelihood Table		Play		
		Yes	No	
Humidity	High	3/9	4/5	7/14
	Normal	6/9	1/5	7/14
		9/14	5/14	

P(Yes|High) = 0.33 x 0.6 / 0.5 = 0.42

P(No|High) = 0.8 x 0.36 / 0.5 = 0.58

Likelihood table for Wind

Likelihood Table		Play		
		Yes	No	
Wind	Weak	6/9	2/5	8/14
	Strong	3/9	3/5	6/14
		9/14	5/14	

P(Yes|Weak) = 0.67 x 0.64 / 0.57 = 0.75

P(No|Weak) = 0.4 x 0.36 / 0.57 = 0.25

Suppose we have a **Day** with the following values :
Outlook = Rain
Humidity = High
Wind = Weak

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

$$P(\text{Yes} \mid \text{Rain, High, Weak}) = \frac{P(\text{Rain, High, Weak} \mid \text{Yes}) * P(\text{Yes})}{P(\text{Rain, High, Weak})}$$

So, with the data, we have to predict whether "we can play on that day or not."

We will need P(A*) = P(Rain, High Humidity, Weak Wind)
P(A*) = P(Outlook = Rain) * P(Humidity = High) * P(Wind = Weak)
P(A*) = 5/14 * 7/14 * 8/14 = 0.10204

Likelihood of 'Yes' on that Day
= P(Outlook = Rain|Yes)*P(Humidity= High|Yes)* P(Wind= Weak|Yes)*P(Yes) / P(A*)
= 3/9 * 3/9 * 6/9 * 9/14 / 0.10204 = 0.466
Likelihood of 'No' on that Day
= P(Outlook = Rain|No)*P(Humidity= High|No)* P(Wind= Weak|No)*P(No) / P(A*)
= 2/5 * 4/5 * 2/5 * 5/14 / 0.10204 = 0.448

Note: We could just maximize P(A*)P(C) since we are dividing by P(A) in both cases

Normalize to get probabilities
P(Yes) = 0.466 / (0.466 + 0.448) = 0.51
P(No) = 0.448 / (0.466 + 0.448) = 0.49

Our model predicts that there is a **51%** chance there will be a game.

How to Estimate Probabilities from Data?

- For continuous attributes:
  - Discretize the range into bins
    - one ordinal attribute per bin
    - violates independence assumption
  - Two-way split: $(A < v)$ or $(A > v)$
    - choose only one of the two splits as new attribute
  - Probability density estimation:
    - Assume attribute follows a normal distribution
    - Use data to estimate parameters of distribution (e.g., mean and standard deviation)
    - Once probability distribution is known, can use it to estimate the conditional probability $P(A_i|c)$

How to Estimate Probabilities from Data?

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Normal distribution:

$$P(A_i \mid c_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(A_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

  - One for each (A_i,c_i) pair
- For (Income, Class=No):
  - If Class=No
    - sample mean = 110
    - sample variance = 2975

$$P(Income = 120 \mid No) = \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

Naïve Bayes
Revisit Bayesian Classification

- $P(c|x) = P(x|c)P(c)/P(x)$
- $P(c)$ - Prior probability of class c – How do we know?
  - Just count up and get the probability for the Training Set – Easy!
- $P(x|c)$ - Probability "likelihood" of data vector X given that the output class is c
  - We use $P(x_1, \ldots, xn|c_j)$ as short for $P(x_1 = val_{1_{, \ldots,}} x_n = val_n|c_j)$
- How do we really do this?
  - If x is nominal we can just look at the training set and count to see the probability of x given the output class c
  - If x is real valued, we can use the probability distribution to estimate

Bayes Classification

- Rephrased

$$P(c|X) = \frac{P(x_1|c) * P(x_2|c) * \ ... * P(x_n|c) * P(c)}{P(x_1) * P(x_2) * \cdots * P(xn)}$$

- Since the denominator remains constant for all the classes

$$P(c|x_1, x_2, \ldots, xn) \approx P(c) \prod_{i=1}^{n} P(xi|c)$$

- Combine this model with a rule to pick the hypothesis that is most probable

$$c_{NB} = argmaxy\ P(c) \prod_{i=1}^{n} P(xi|c)$$

Bayes Classification

- The rule to pick the hypothesis doesn't yield the probability of the hypotheses.
- Most often, we simply normalize the values for each of the classes

$$P(c_i|X) = \frac{P(ci)\prod_{i=1}^{n} P(xi|ci)}{P(c_1) + P(c_2) + \ ... + P(cn)}$$

- Sometimes referred to as the true probability

Naïve Bayes Classifier

- Conditional independence may not be a reasonable assumption… (heart rate and blood pressure), but
  - Low complexity simple approach
  - Need only store all $P(c_j)$ and $P(x_i|c_j)$ terms
  - Assume nominal features for the moment
  - Easy to calculate the $|attribute\ values| \times |classes|$ terms
  - There is often enough data to make the independent terms reasonably accurate
  - Effective and common for many large applications (Document classification, etc.)

Naïve Bayes Example

Size (L, S)	Color (R,G,B)	Output (P,N)
L	R	N
S	B	P
S	G	N
L	R	N
L	G	P

For the given training set:

1. Create a table of the statistics needed to do Naïve Bayes
2. What would be the best output for a new instance which is Large and Blue? (e.g. the class which wins the argmax)
3. What is the normalized probability for each output class (P or N) for Large and Blue?

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_i P(x_i | c_j)$$

What do we need?

Size (L, S)	Color (R,G,B)	Output (P,N)
L	R	N
S	B	P
S	G	N
L	R	N
L	G	P

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_i P(x_i|c_j)$$

P(P)	
P(N)	
P(Size=L|P)	
P(Size=S|P)	
P(Size=L|N)	
P(Size=S|N)	
P(Color=R|P)	
P(Color=G|P)	
P(Color=B|P)	
P(Color=R|N)	
P(Color=G|N)	
P(Color=B|N)	

$P(c_j)$

$P(x_i|c_j)$

Challenge Question
Finish and give the true Probabilities of P and N for Size = L and Color = B

Size (L, S)	Color (R,G,B)	Output (P,N)
L	R	N
S	B	P
S	G	N
L	R	N
L	G	P

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_i P(x_i|c_j)$$

P(P)	2/5	$P(c_j)$
P(N)	3/5	
P(Size=L|P)	1/2	$P(x_i\|c_j)$
P(Size=S|P)	1/2	
P(Size=L|N)	2/3	
P(Size=S|N)		
P(Color=R|P)		
P(Color=G|P)		
P(Color=B|P)		
P(Color=R|N)		
P(Color=G|N)		
P(Color=B|N)		

P(P) = 2/5 * 1/2 * P(Color = B|P) = ? P(N) = ?

Challenge Question
Finish and give the true Probabilities of P and N for Size = L and Color = B

Size (L, S)	Color (R,G,B)	Output (P,N)
L	R	N
S	B	P
S	G	N
L	R	N
L	G	P

P(P)	2/5
P(N)	3/5
P(Size=L|P)	1/2
P(Size=S|P)	1/2
P(Size=L|N)	2/3
P(Size=S|N)	1/3
P(Color=R|P)	0/2
P(Color=G|P)	1/2
P(Color=B|P)	1/2
P(Color=R|N)	2/3
P(Color=G|N)	1/3
P(Color=B|N)	0/3

$P(c_j)$

$P(x_i|c_j)$

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_i P(x_i|c_j)$$

Normalized Probabilities
P(P) = 1/10/(1/10+0) = 1
P(N) = 0/(1/10+0) = 0

P(P) = 2/5 * 1/2 * 1/2 = 1/10

P(N) = 3/5 * 2/3 * 0/3 = 0

Example of Naïve Bayes Classifier

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

A: attributes

M: mammals

N: non-mammals

$$P(A|M) = P(Birth|M) * P(Fly|M) * P(Water|M) * P(Legs|M)$$

$$P(A \mid M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$P(A \mid N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$P(A \mid M)P(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$P(A \mid N)P(N) = 0.004 \times \frac{13}{20} = 0.0027$$

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

P(A|M)P(M) > P(A|N)P(N)

=> Mammals

Naïve Bayes Homework

Size (L, S)	Color (R,G,B)	Output (P,N)
L	R	P
S	B	P
S	B	N
L	R	N
L	B	P
L	G	N
S	B	P

For the given training set:

1. Create a table of the statistics needed to do Naïve Bayes
2. What would be the best output for a new instance which is Small and Blue? (e.g. the class which wins the argmax)
3. What is the true probability for each output class (P or N) for Small and Blue?

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_i P(x_i | c_j)$$

Text Classification Example

- A text classification approach
  - Want P(*class*|*document*) – How to represent document?

Text Classification Example

- A text classification approach
  - Want P(*class*|*document*) - Use a "Bag of Words" approach – order independence assumption (valid?)
    - Variable length input of query document is fine
  - Calculate bag of words for every word/token in the language and each output class based on the training data. Words that occur in testing but do not occur in the training data are ignored.
  - Good empirical results. Can drop filler words (the, and, etc.) and words found less than *z* times in the training set.

Text Classification Example

- A text classification approach
  - Want P(*class*|*document*) - Use a "Bag of Words" approach – order independence assumption (valid?)
    - Variable length input of query document is fine
  - Calculate bag of words for every word/token in the language and each output class based on the training data. Words that occur in testing but do not occur in the training data are ignored.
  - Good empirical results. Can drop filler words (the, and, etc.) and words found less than *z* times in the training set.
  - $P(class|document) \approx P(class|BagOfWords)$ //assume word order independence

    $= P(BagOfWords|class)*P(class)/P(document)$ //Bayes Rule

    // But *BagOfWords* usually unique

    // $P(document)$ same for all classes

    $\approx P(class)*\Pi P(word|class)$ // Thus Naïve Bayes

Naïve Bayes Classifier

- If one of the conditional probability is zero, then the entire expression becomes zero
- Probability estimation:

$$\text{Original}: P(A_i \mid C) = \frac{N_{ic}}{N_c}$$

$$\text{Laplace}: P(A_i \mid C) = \frac{N_{ic} + 1}{N_c + N_i}$$

$$\text{m-estimate}: P(A_i \mid C) = \frac{N_{ic} + mp}{N_c + m}$$

N_i: number of attribute values for attribute A_i

p: prior probability

m: parameter

Implementation details

- Computing the conditional probabilities involves multiplication of many very small numbers
  - Numbers get very close to zero, and there is a danger of numeric instability
- We can deal with this by computing the logarithm of the conditional probability

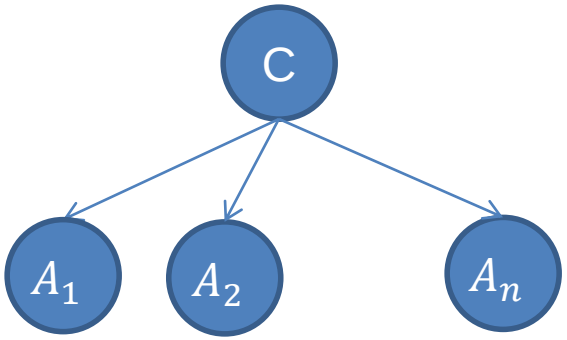
$$\log P(C|A) \sim \log P(A|C) + \log P(A)$$
$$= \sum_i \log P(A_i|C) + \log P(A)$$

Naïve Bayes (Summary)

- Robust to isolated noise points
- Handle missing values by ignoring the instance during probability estimate calculations
- Robust to irrelevant attributes
- Independence assumption may not hold for some attributes
  - Use other techniques such as Bayesian Belief Networks (BBN)
- Naïve Bayes can produce a probability estimate, but it is usually a very biased one
  - Logistic Regression is better for obtaining probabilities.

Generative vs Discriminative models

- Naïve Bayes is a type of a generative model
  - Generative process:
    - First pick the category of the record
    - Then given the category, generate the attribute values from the distribution of the category
    - Conditional independence given C



- We use the training data to learn the distribution of the values in a class

Generative vs Discriminative models

- Logistic Regression and SVM are discriminative models
  - The goal is to find the boundary that discriminates between the two classes from the training data

- In order to classify the language of a document, you can
  - Either learn the two languages and find which is more likely to have generated the words you see
  - Or learn what differentiates the two languages.

sklearn.naive_bayes

- sklearn has 5 different implementations
  - Based on probability distribution types

`sklearn.naive_bayes`: Naive Bayes

The `sklearn.naive_bayes` module implements Naive Bayes algorithms. These are supervised learning methods based on applying Bayes' theorem with strong (naive) feature independence assumptions.

User guide: See the Naive Bayes section for further details.

`naive_bayes.BernoulliNB`(*[, alpha, ...])	Naive Bayes classifier for multivariate Bernoulli models.
`naive_bayes.CategoricalNB`(*[, alpha, ...])	Naive Bayes classifier for categorical features.
`naive_bayes.ComplementNB`(*[, alpha, ...])	The Complement Naive Bayes classifier described in Rennie et al. (2003).
`naive_bayes.GaussianNB`(*[, priors, ...])	Gaussian Naive Bayes (GaussianNB).
`naive_bayes.MultinomialNB`(*[, alpha, ...])	Naive Bayes classifier for multinomial models.

The Complement Naive Bayes classifier described in Rennie et al. (2003).
ComplementNB implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the *complement* of each class to compute the model's weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks.

LOGISTIC REGRESSION

Classification via regression

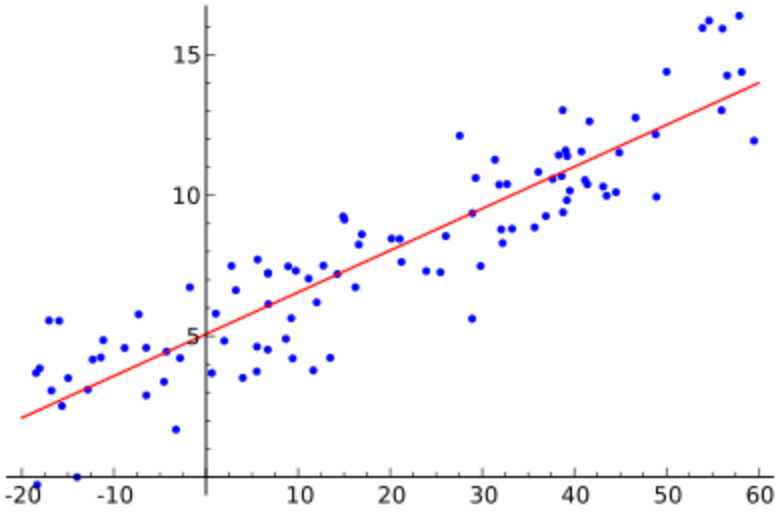
- Instead of predicting the class of a record we want to predict the probability of the class given the record
- The problem of predicting continuous values is called regression problem
- General approach: find a continuous function that models the continuous points.

Example: Linear regression

- Given a dataset of the form $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ find a linear function that given the vector x_i predicts the y_i value as $y_i' = w^T x_i$
  - Find a vector of weights w that minimizes the sum of square errors

$$\sum_i (y_i' - y_i)^2$$

  - Several techniques for solving the problem.



Classification via regression

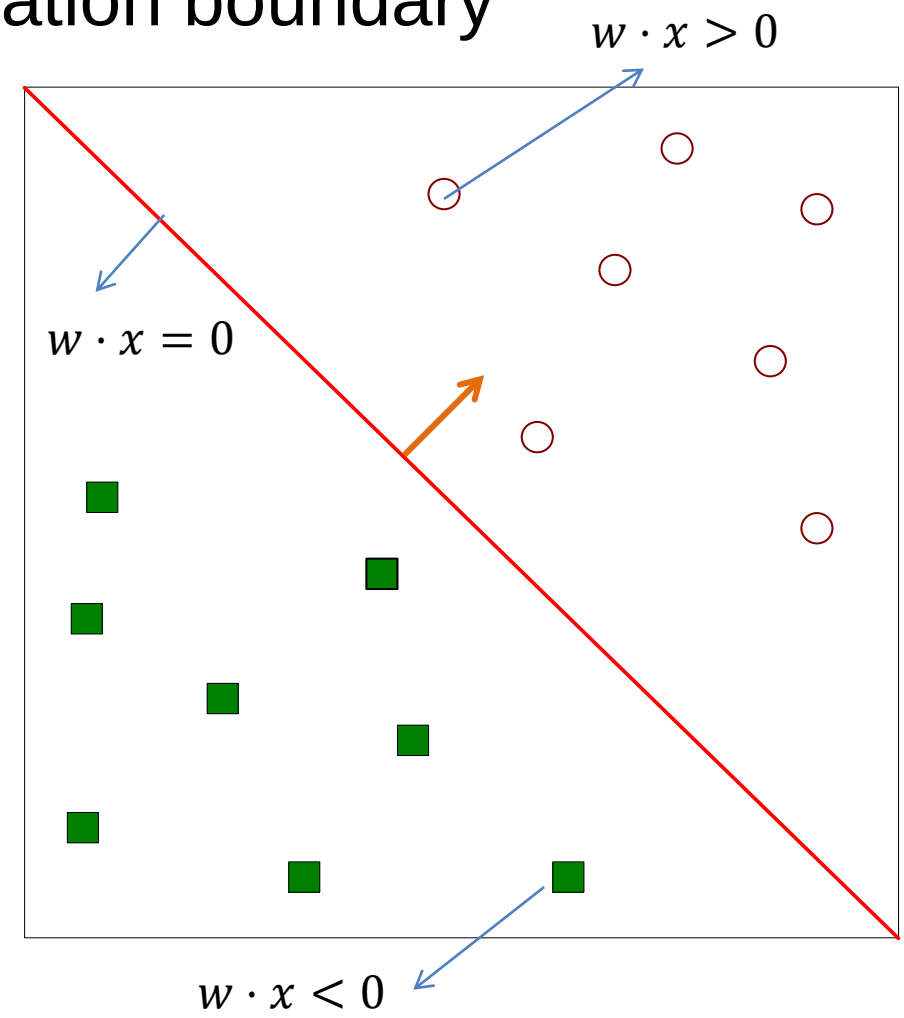
- Assume a linear classification boundary

For the positive class the bigger the value of $w \cdot x$, the further the point is from the classification boundary, the higher our certainty for the membership to the positive class

- Define $P(C_+|x)$ as an increasing function of $w \cdot x$

For the negative class the smaller the value of $w \cdot x$, the further the point is from the classification boundary, the higher our certainty for the membership to the negative class

- Define $P(C_-|x)$ as a decreasing function of $w \cdot x$



Logistic Regression

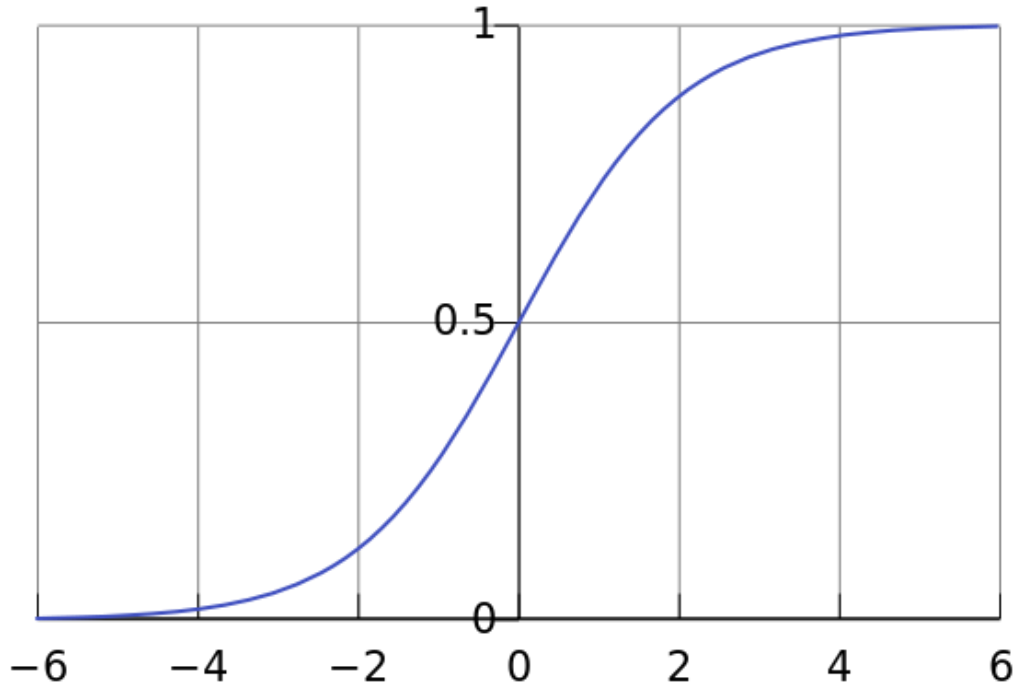
The logistic function

$$f(t) = \frac{1}{1 - e^{-t}}$$

$$P(C_+|x) = \frac{1}{1 - e^{-w \cdot x}}$$

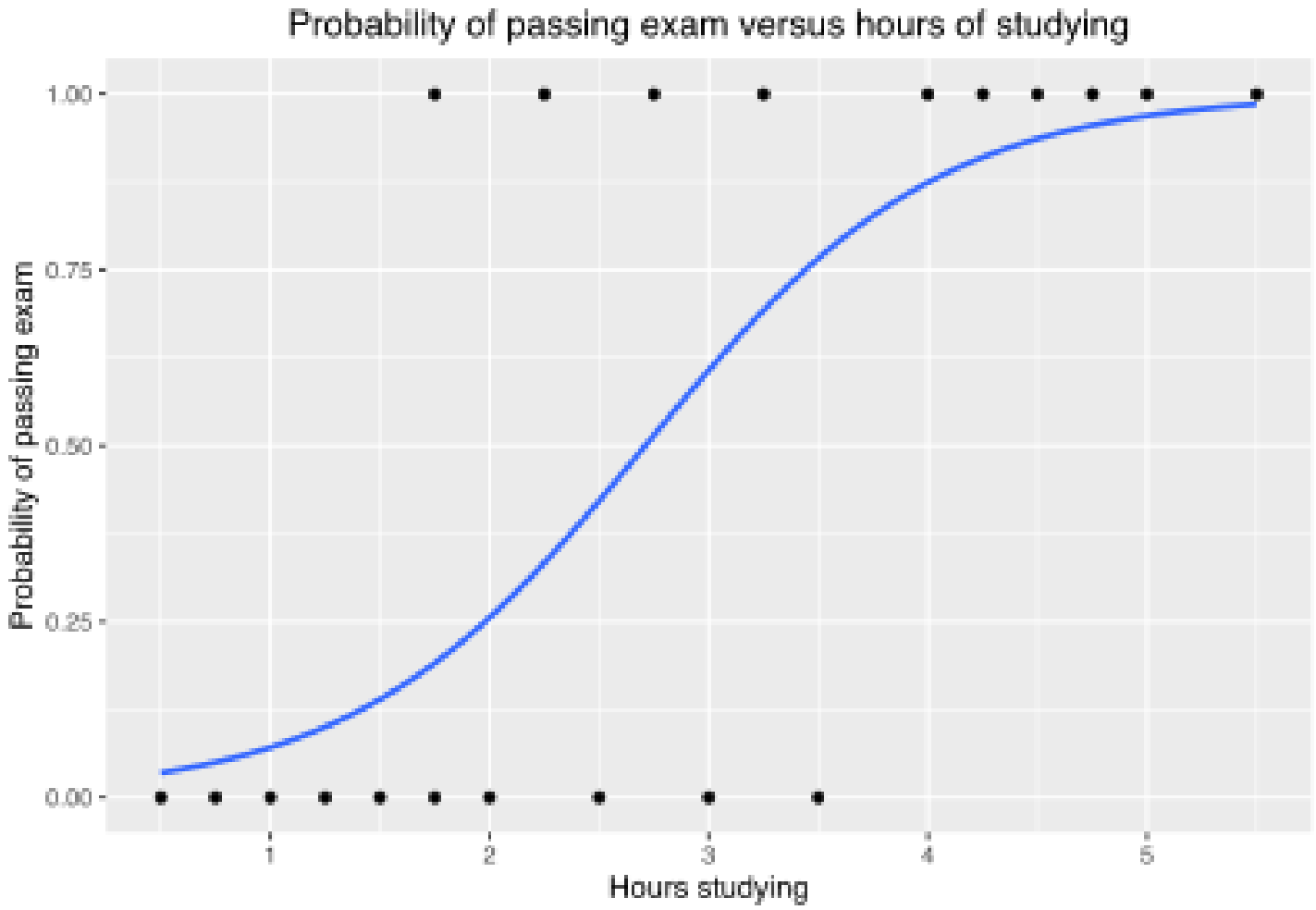
$$P(C_-|x) = \frac{e^{-w \cdot x}}{1 - e^{-w \cdot x}}$$

$$\log \frac{P(C_+|x)}{P(C_-|x)} = w \cdot x$$



Logistic Regression: Find the vector w that maximizes the probability of the observed data

Logistic Regression



Logistic Regression

- Produces a probability estimate for the class membership which is often very useful.
- The weights can be useful for understanding the feature importance.
- Works for relatively large datasets
- Fast to apply.