Data Mining & Machine Learning

CS37300
Purdue University

October 6, 2017

Extra Credit Competition Update

Public Leaderboard **Private Leaderboard**

This leaderboard is calculated with approximately 30% of the test data.
The final results will be based on the other 70%, so the final standings may be different.

Raw Data Refresh

#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1	new	**Luke Skywalker** NBC			0.74585	2	2d
2	new	**Bossk**			0.67826	2	21m
3	new	**Yoda**			0.63820	1	9h
4	new	**Revan**			0.55281	2	11h
5	new	**Boba Fett**			0.51800	3	11h
		Bank_Sample_Submission.csv			0.49008		

How to beat Skywalker…

or

Classifiers beyond NBC and Decision Trees

So far…

A few weeks ago…
we reviewed Naive Bayes Classifier
and the Decision Tree…

We now embark on a quest
to find other classifiers

Classifiers for today

- Nearest neighbors
- Linear Regression
- Support vector machines
- Logistic Regression (1-layer neural network)

Classification Task

- Data representation:
  - Training set: Paired attribute vectors and class labels <$y(i)$, $\boldsymbol{x}(i)$>

    or

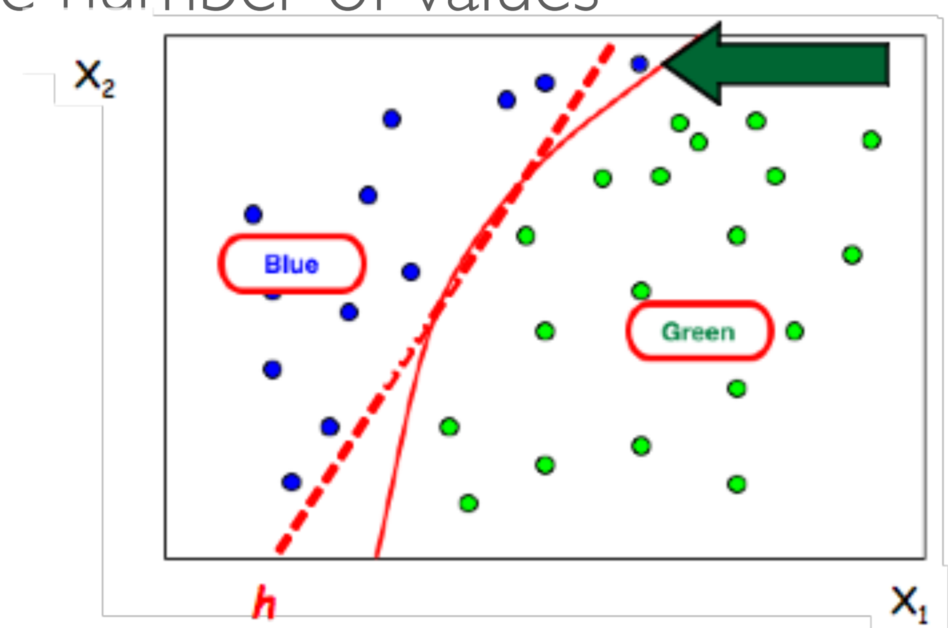
    $\mathbf{n}\times\mathbf{p}$ tabular data with class label (y) and $\mathbf{p}$-1 attributes ($\boldsymbol{x}$)

- Task: estimate a predictive function $f(\boldsymbol{x};\theta)=y$
  - Assume that there is a function $y=f(x)$ that maps data instances ($\mathbf{x}$) to class labels ($\mathbf{y}$)
  - Construct a model that approximates the mapping
    - Classification: if y is categorical (e.g., {yes, no}, {dog, cat, elephant})
    - Regression: if y is real-valued (e.g., stock prices)

Binary classification

- In its simplest form, a classification model defines a decision boundary (h) and labels for each side of the boundary

- Input: $\boldsymbol{x}=\{x_1,x_2,\ldots,x_n\}$ is a set of attributes, function f assigns a label y to input $\boldsymbol{x}$, where y is a discrete variable with a finite number of values



Nearest Neighbors

Nearest neighbor

- Instance-based method
- Learning
  - Stores training data and delays processing until a new instance must be classified
  - Assumes that all points are represented in p-dimensional space
- Prediction
  - **Nearest neighbors** are calculated using Euclidean distance
  - Classification is made based on class labels of neighbors

1NN

- Training set: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
  where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ is a feature vector of p continuous attributes
  and y_i is a discrete class label

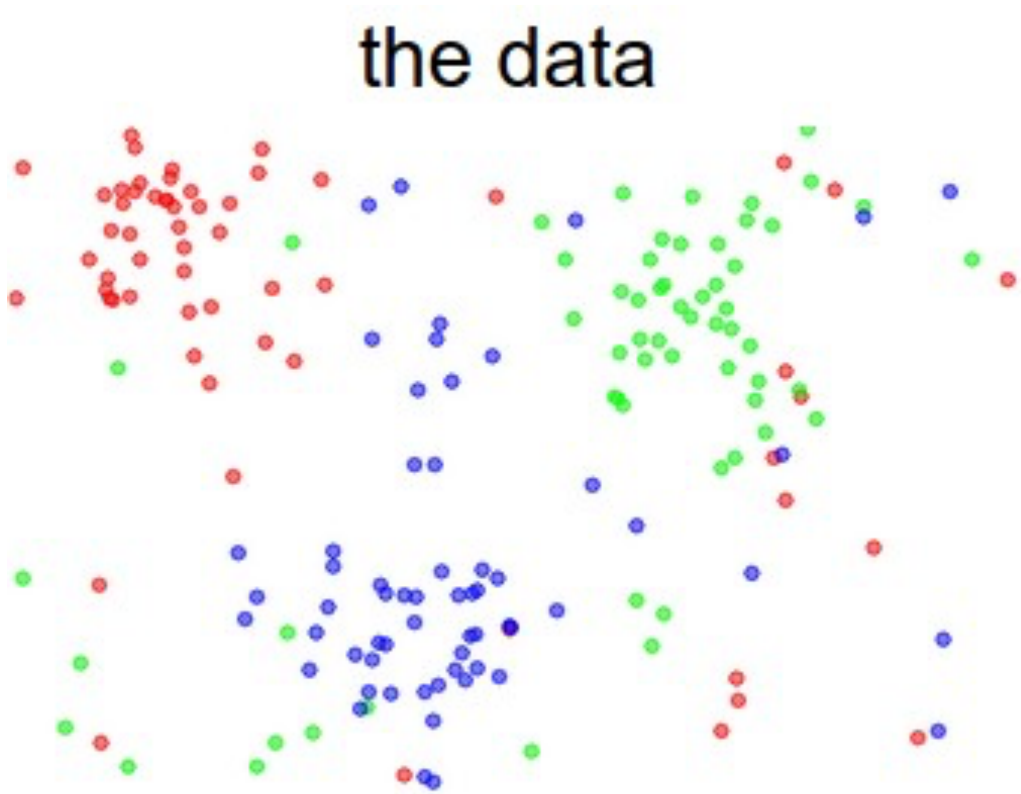
- **1NN algorithm**

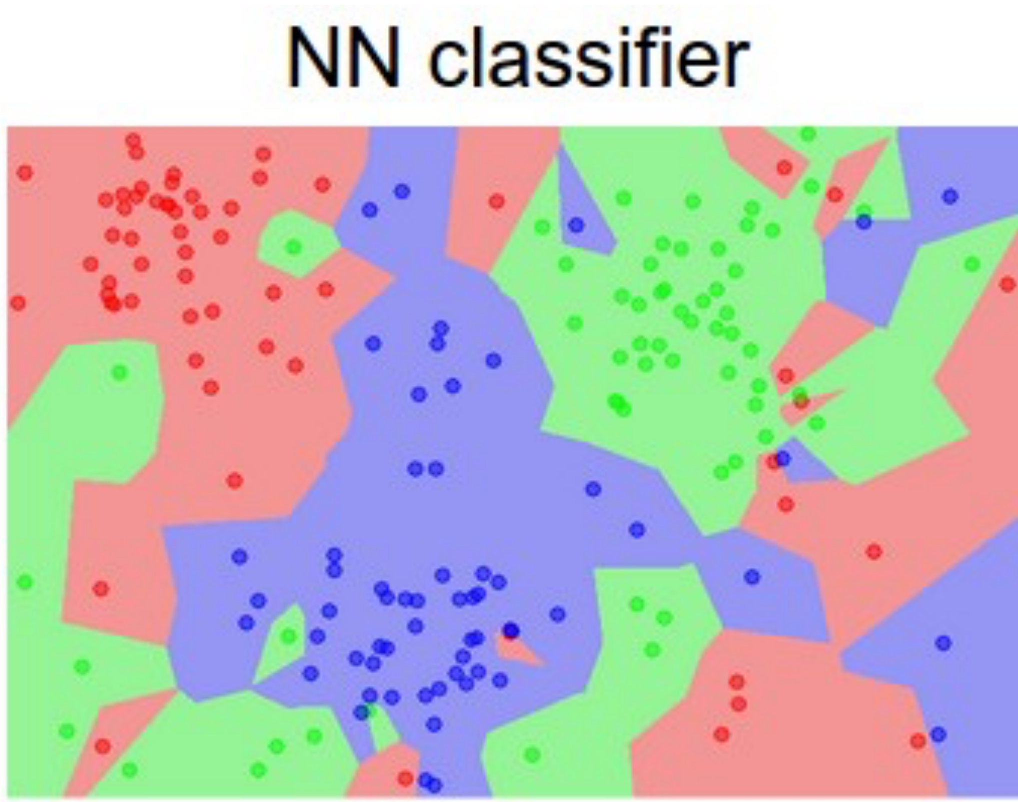
  To predict a class label for new instance j:
  Find the training instance point $\mathbf{x}_i$ such that $d(\mathbf{x}_i, \mathbf{x}_j)$ is minimized
  Let $f(\mathbf{x}_j) = y_i$

- Key idea: Find instances that are "similar" to the new instance and use their class labels to make prediction for the new instance

  - 1NN generalizes to kNN when more neighbors are considered

kNN model: decision boundaries





Source: http://cs231n.github.io/classification/

kNN

- **kNN algorithm**

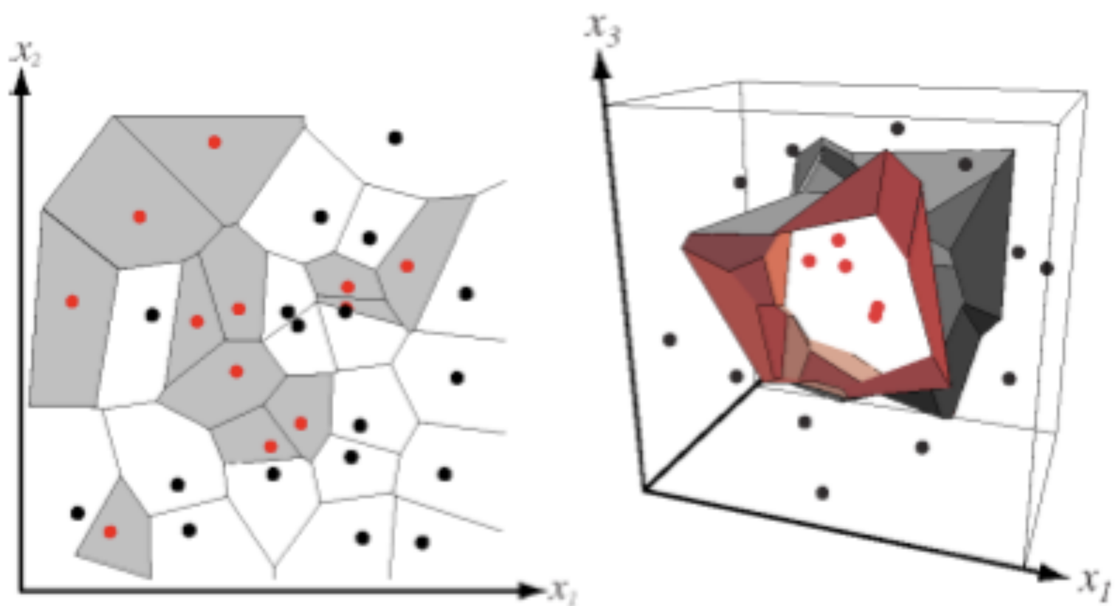
  To predict a class label for new instance j:
  Find the k nearest neighbors of j, i.e., those that minimize $d(\mathbf{x}_k, \mathbf{x}_j)$
  Let $f(\mathbf{x}_j) = g(\mathbf{y}_k)$, e.g., majority label in $\mathbf{y}_k$

- *Algorithm choices*

  - How many neighbors to consider (i.e., choice of k)?
    ... Usually a small value is used, e.g. $k<10$
  - What distance measure d( ) to use?
    ... Euclidean L2 distance is often used
  - What function g( ) to combine the neighbors' labels into a prediction?
    ... Majority vote is often used

1NN decision boundary

- For each training example i, we can calculate its **Voronoi cell**, which corresponds to the space of points for which i is their nearest neighbor
- All points in such a Voronoi cell are labeled by the class of the training point, forming a Voronoi tessellation of the feature space



Source: http://www.cs.bilkent.edu.tr/~saksoy/courses/cs551-Spring2008/slides/cs551_nonbayesian1.pdf

Nearest neighbor

- Strengths:
  - Simple model, easy to implement
  - Very efficient learning: O(1)
- Weaknesses:
  - Inefficient inference: time and space O(n)
  - Curse of dimensionality:
    - As number of features increase, you need an exponential increase in the size of the data to ensure that you have nearby examples for any given data point

k-NN learning

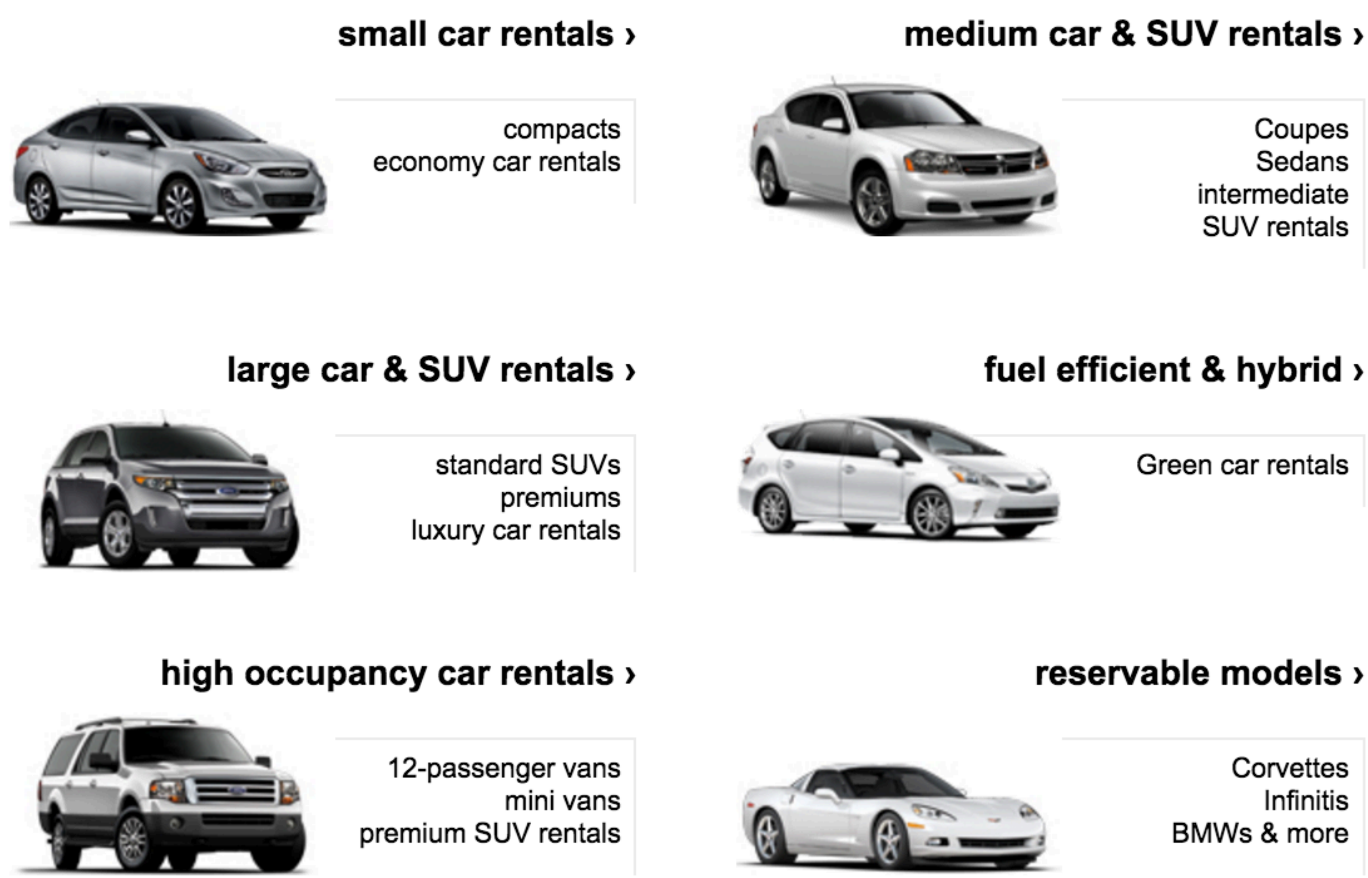
- Parameters of the model:
  - k (number of neighbors)
  - any parameters of distance measure (e.g., weights on features)

- **Model space**
  - Possible tessellations of the feature space
- **Search algorithm**
  - Implicit search: choice of k, d, and g uniquely define a tessellation
- **Score function**
  - Majority vote is minimizing misclassification rate

Least Squares Classifier

Motivation

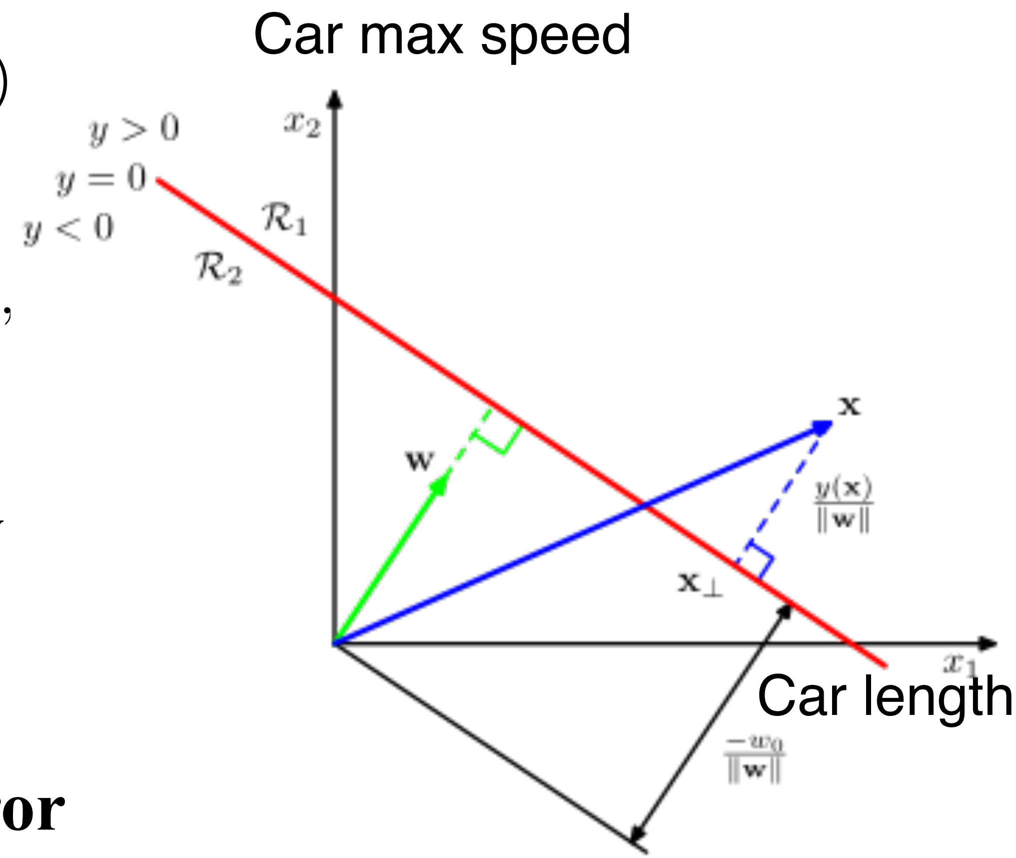
- Given x features of a car (length, width, mpg, maximum speed,...)
- Classify cars into categories based on x



Least Squares Classifier

Two classes:

- x is a real-valued vector (features)
- y is the car class



$$y_c = \begin{cases} 1 & \text{, if car } c \text{ is ``economy''} \\ -1 & \text{, if car } c \text{ is ``luxury''} \end{cases}$$

- Find linear discriminant weights $\mathbf{w}$

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

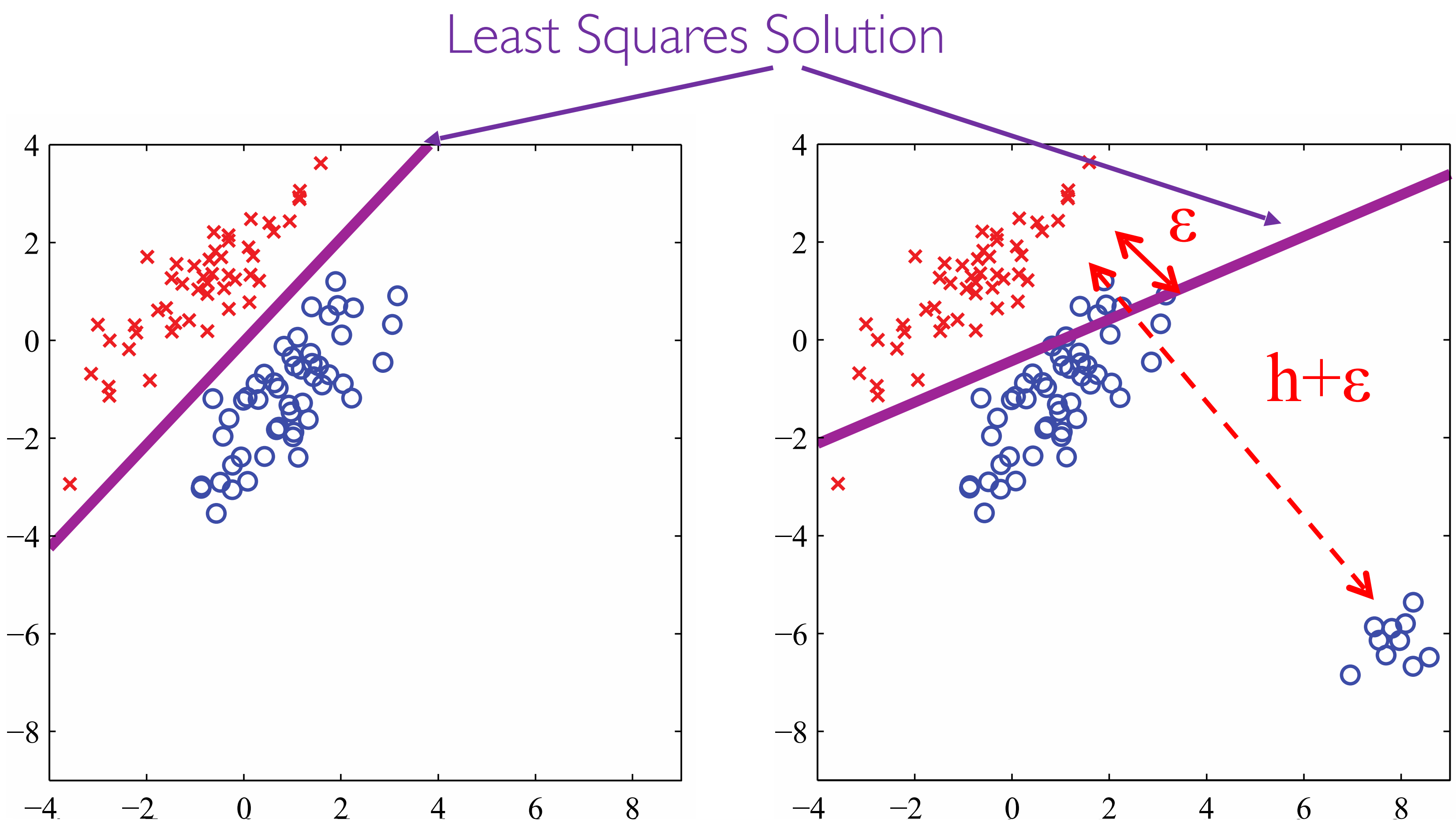
- Score function **least squares error**

$$\text{score}(\text{Test\_Data\_Cars}) = \sum_{c \in \text{Test\_Data\_Cars}} (y_c - y(x_c))^2$$

- Search function: find w, w_0 that minimize score

Issues with Least Squares Classification

$$\text{score}(\text{Test\_Data\_Cars}) = \sum_{c \in \text{Test\_Data\_Cars}} (y_c - y(x_c))^2$$

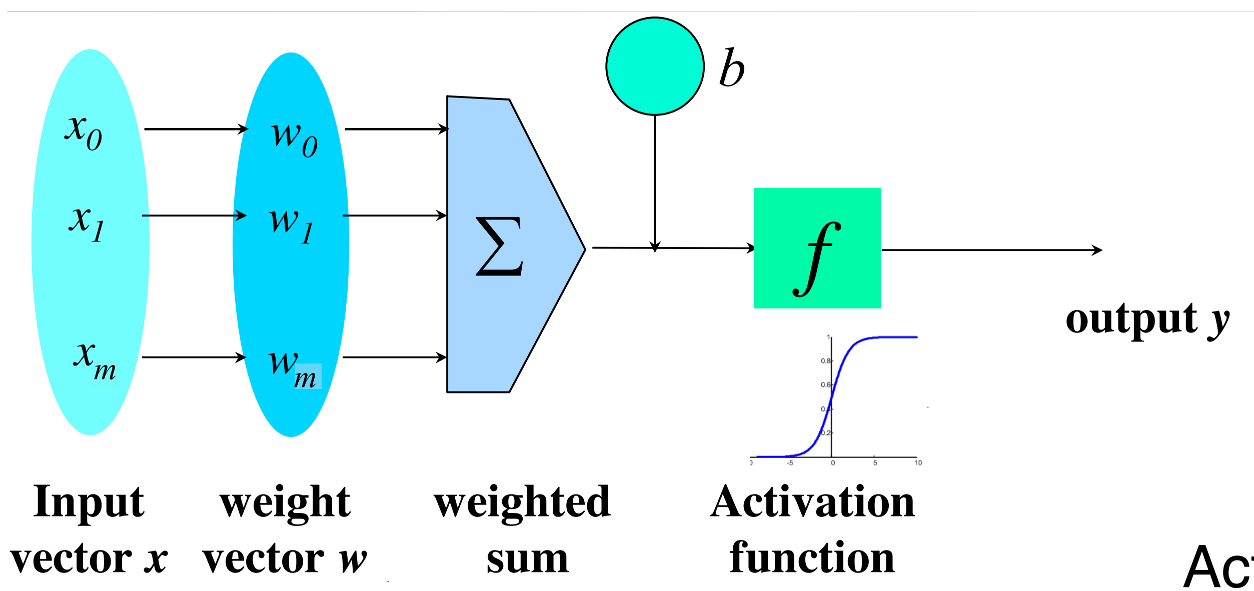


cares too much about well classified items

Neural networks

- Analogous to biological systems
- Massive parallelism is computationally efficient
- First learning algorithm in 1959 (Rosenblatt)
  - Perceptron learning rule
  - Provide target outputs with inputs for a single neuron
  - Incrementally update weights to learn to produce outputs

Neuron



Activation function

$$y(x) = \sigma(f(x))$$

$$f(x) = \sum_{i=1}^{m} w_i x_i + b$$

Single neuron (Logistic regression)

- Model: Single neuron is often used for two classes $(y=0, y=1)$

$$p(y = 1|x) = \sigma(f(x))$$

$$f(x) = \sum_{i=1}^{m} w_i x_i + b$$

where

$$\sigma(a) = \frac{\exp(a)}{1 + \exp(a)}$$

Logistic function

- Score function:

$$P[\text{Training Data}|\mathbf{w}, b] = \prod_{c \in \text{Training Data}} p(y(x_c) = y_c | x_c)$$

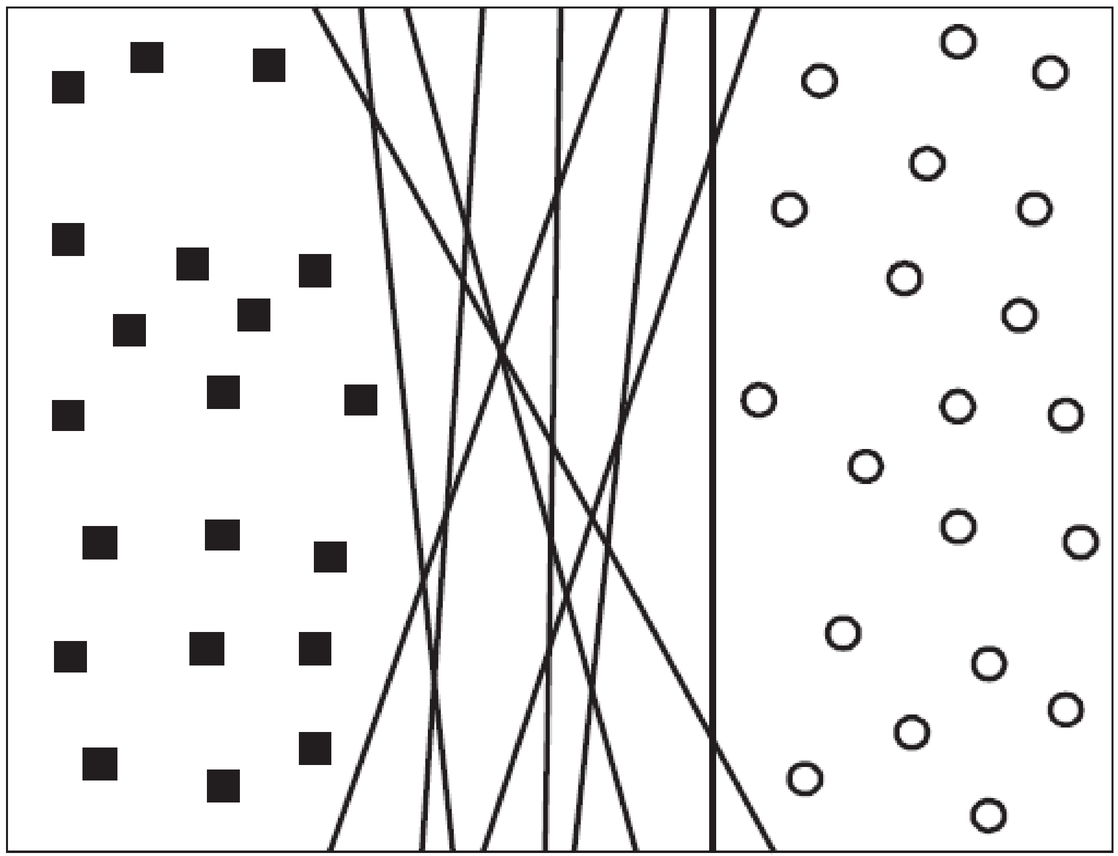
- Search: find w, b that minimize score

Support vector machines (SVMs)

Support vector machines

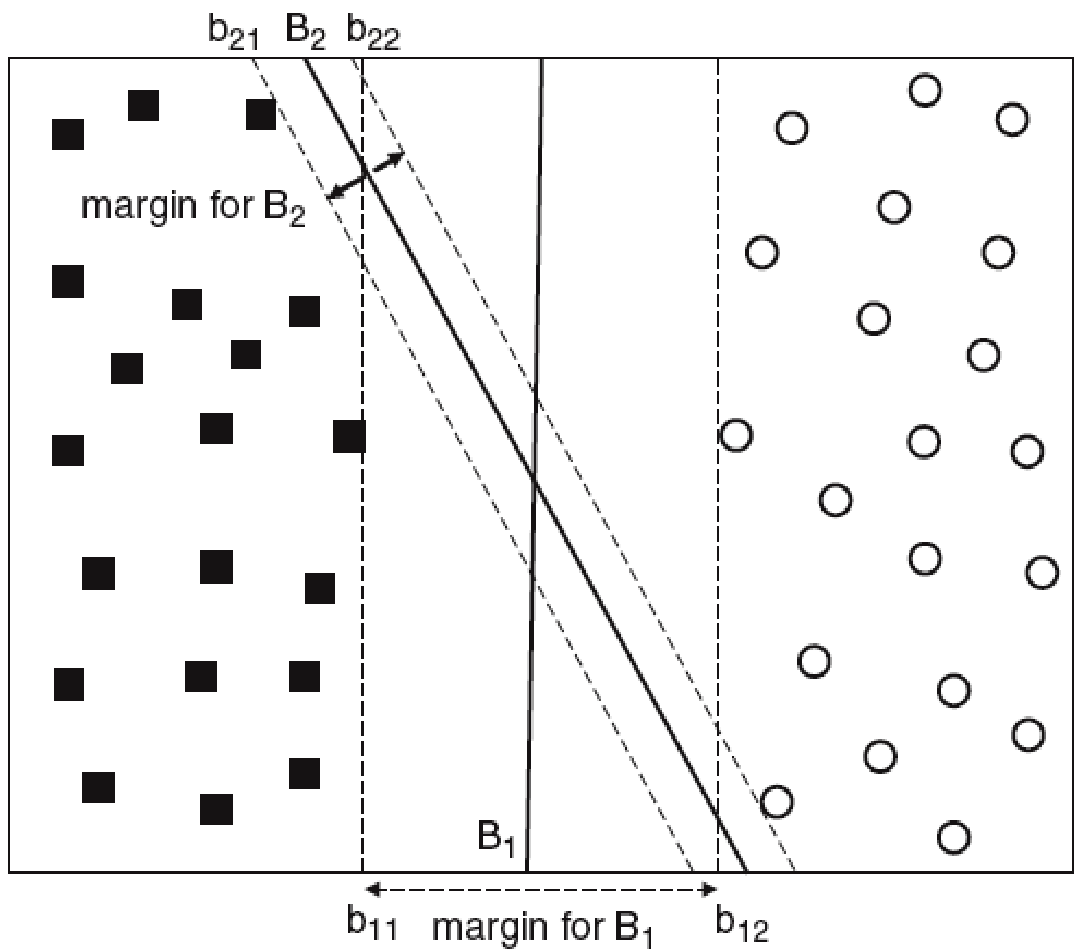
- Discriminative model
- General idea:
    - Find best boundary points (support vectors) and build classifier on top of them
- Linear and non-linear SVMs

Choosing hyperplanes to separate points



Source: Introduction to Data Mining, Tan, Steinbach, and Kumar

Among equivalent hyperplanes, choose one that maximizes "margin"

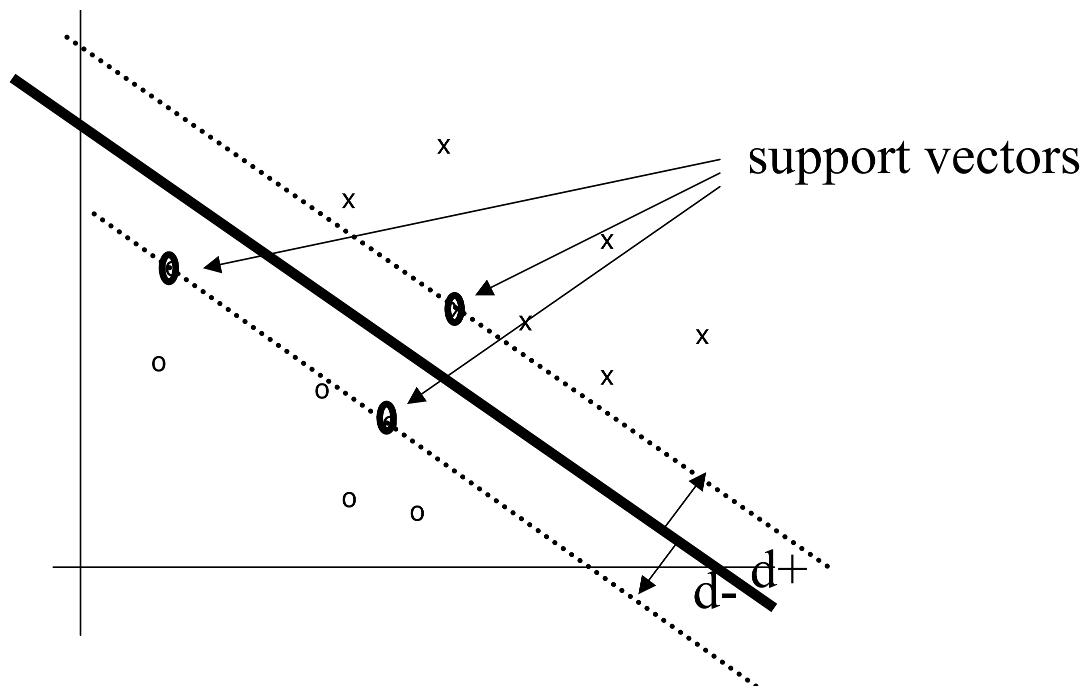


Source: Introduction to Data Mining, Tan, Steinbach, and Kumar

Linear SVMs

$$y = sign\left[\sum_{i=1}^{m} w_i x_i + b\right]$$

- Same functional form as perceptron
- Different learning procedure: Search for hyperplane with largest margin
- Margin=$d_+ + d_-$ where d_+ is distance to closest positive example and d_- is distance to closest negative example



Constrained optimization for SVMs

$$Eq1: \quad x(j) \cdot w + b \geq +1 \; for \; y(j) = +1$$

$$Eq2: \quad x(j) \cdot w + b \leq -1 \; for \; y(j) = -1$$

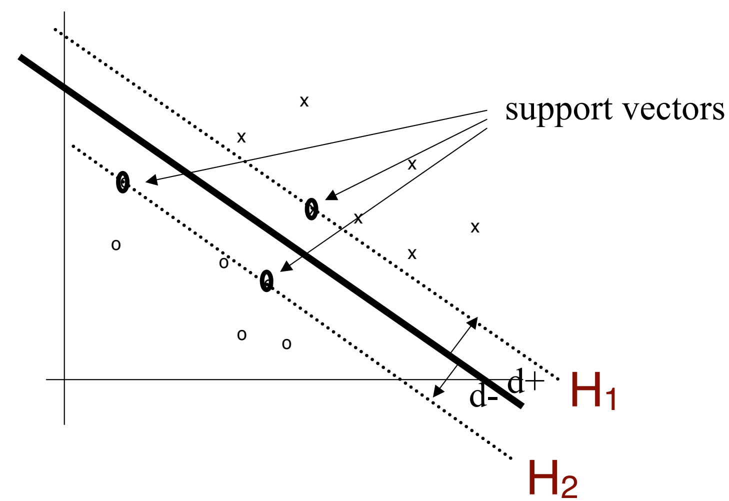
Prediction constraint

$$Eq3: \quad y(j)(x(j) \cdot w + b) - 1 \geq 0 \; \forall y(j)$$

Hyperplane boundaries

$$H_1: \quad x(j) \cdot w + b = +1$$

$$H_2: \quad x(j) \cdot w + b = -1$$



$$d_+ = d_- = \frac{1}{||w||} \qquad margin = \frac{2}{||w||}$$

- Can maximize margin by minimizing ||w|| as it defines the hyperplanes

SVM optimization

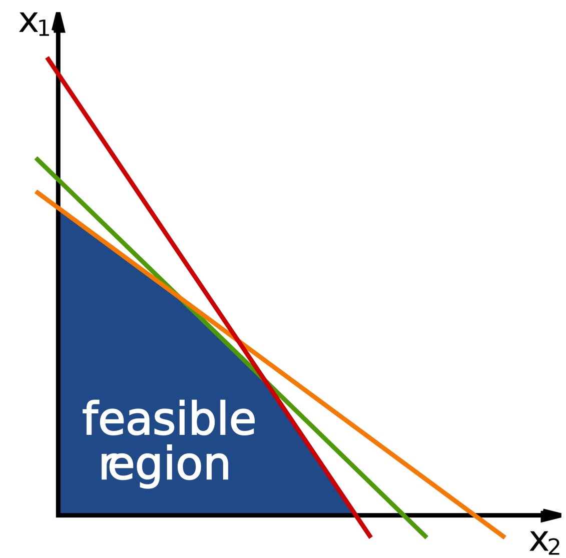
- **Search**: Maximize margin by minimizing $0.5||w||^2$ subject to constraints on Eq3
  - Note: Maximizing $2/||w||$ is equivalent to minimizing $0.5||w||^2$
- Introduce Lagrange multipliers (α) for constraints into score function to minimize:

$$L_P = \frac{1}{2}||w||^2 - \sum_{i=1}^{I} \alpha_i y(i)[x(i) \cdot w + b] + \sum_{i=1}^{I} \alpha_i$$

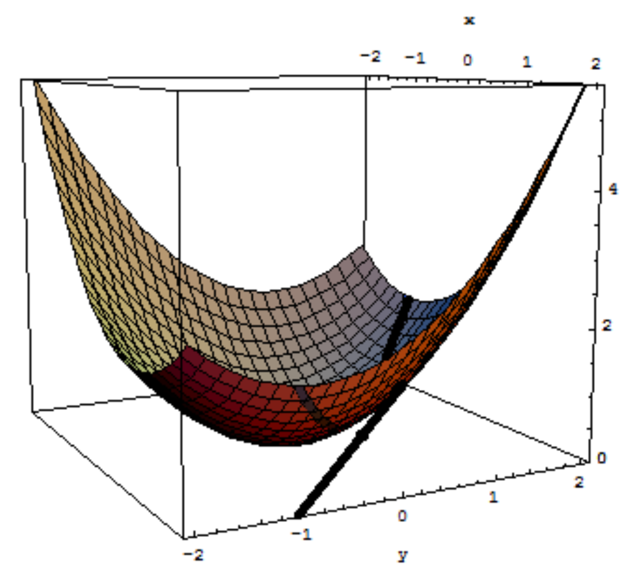
- Minimize L_P with respect to w, b, and $\alpha_N \geq 0$
- Convex programming problem
  - Quadratic programming problem with parameters w, b, α

Constrained optimization

- Linear programming (LP) is a technique for the optimization of a linear objective function, subject to linear constraints on the variables



- Quadratic programming (QP) is a technique for the optimization of a quadratic function of several variables, subject to linear constraints on these variables



SVM components

- **Model space**
    - Set of weights **w** and b (hyperplane boundary)
- **Search algorithm**
    - Quadratic programming to minimize L_p with constraints
- **Score function**
    - L_p: maximizes margin subject to constraint that all training data is correctly classified

Limitations of linear SVMs

- Linear classifiers cannot deal with:
  - Non-linear concepts
  - Noisy data
- Solutions:
  - Soft margin (e.g., allow mistakes in training data)
  - Network of simple linear classifiers (e.g., neural networks)
  - Map data into richer feature space (e.g., non-linear features) and then use linear classifier