Linear Models

Li, Jia

DSAA 5002

The Hong Kong of Science and Technology (Guangzhou)

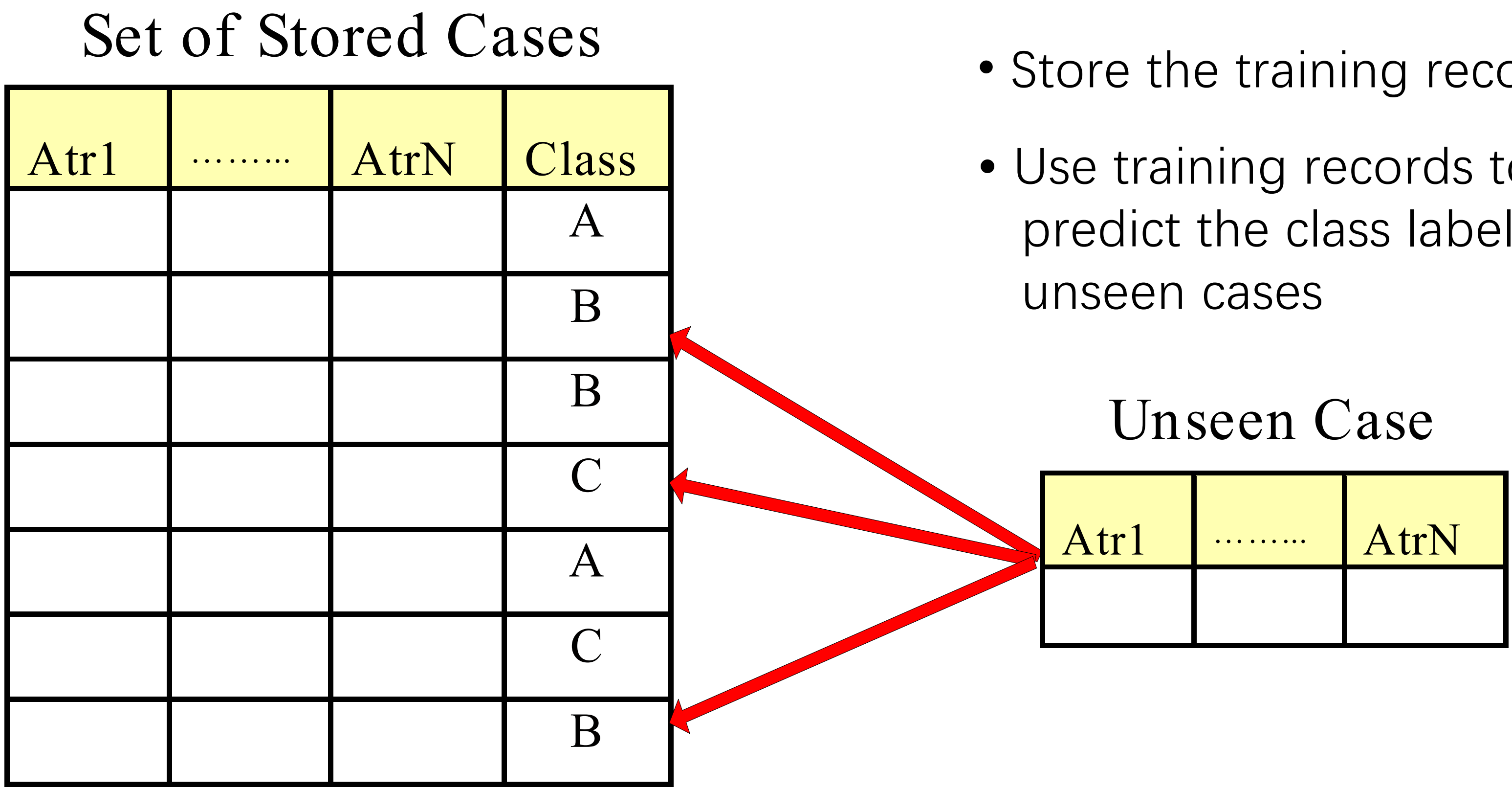
2025 Fall

Sep 15

Overview

- Instance-based classifiers, k-NN
- Perceptron Learning
- Linear Models

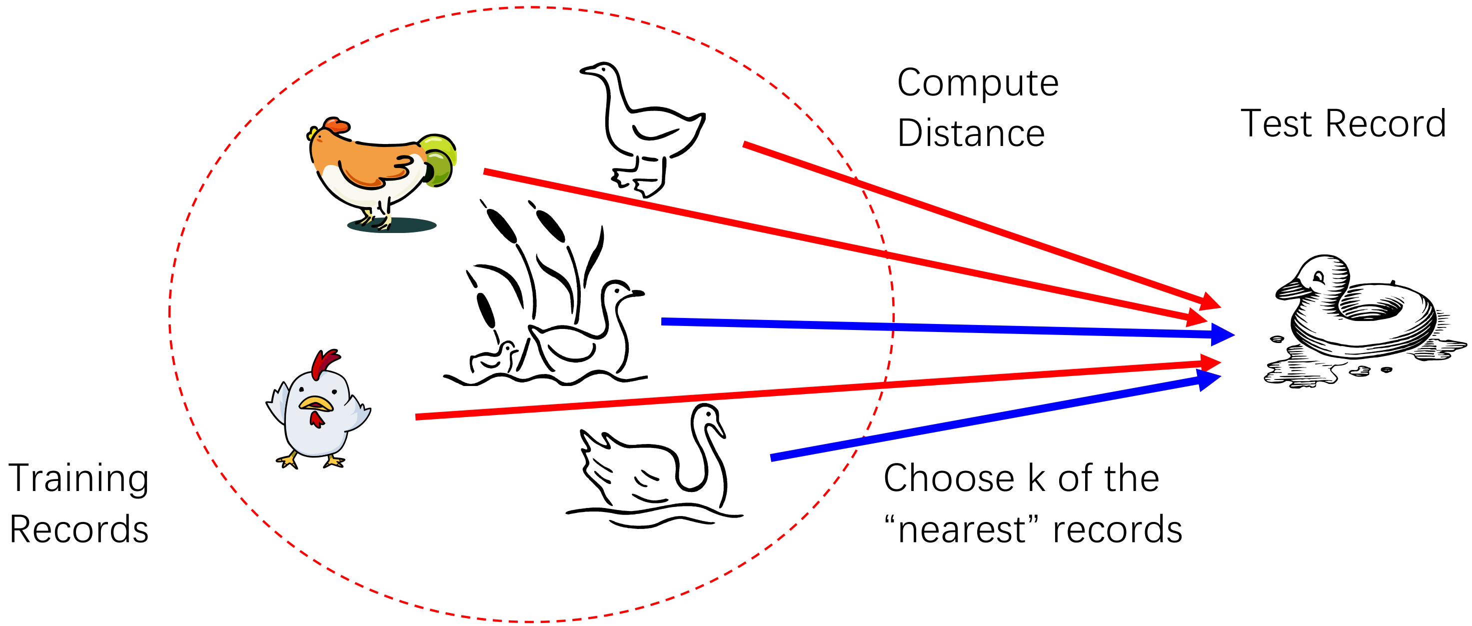
Instance-Based Classifiers



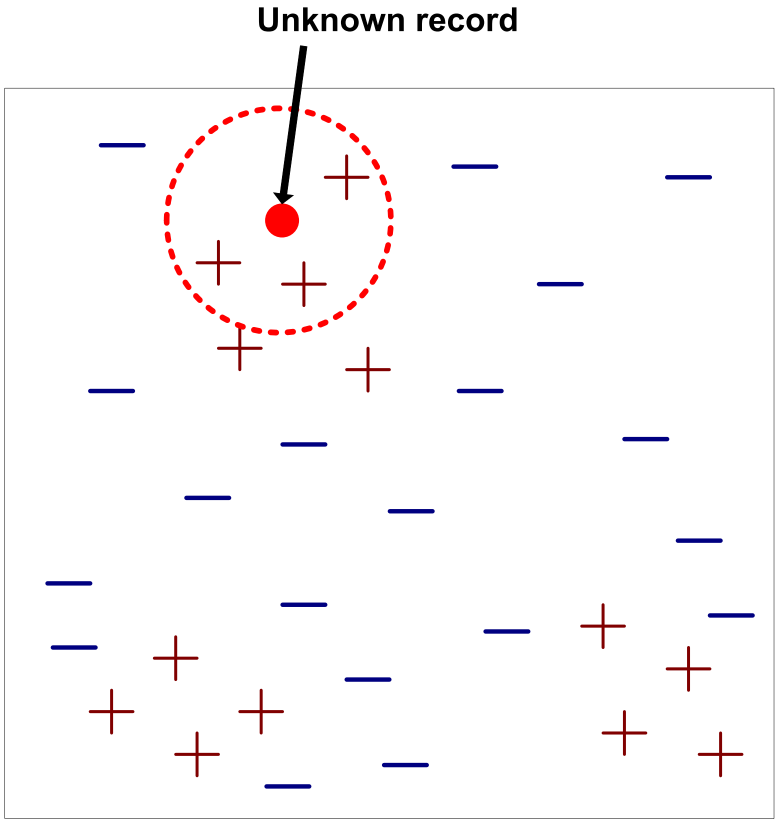
- Store the training records
- Use training records to predict the class label of unseen cases

Nearest Neighbor Classifiers

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



Nearest-Neighbor Classifiers



- 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)

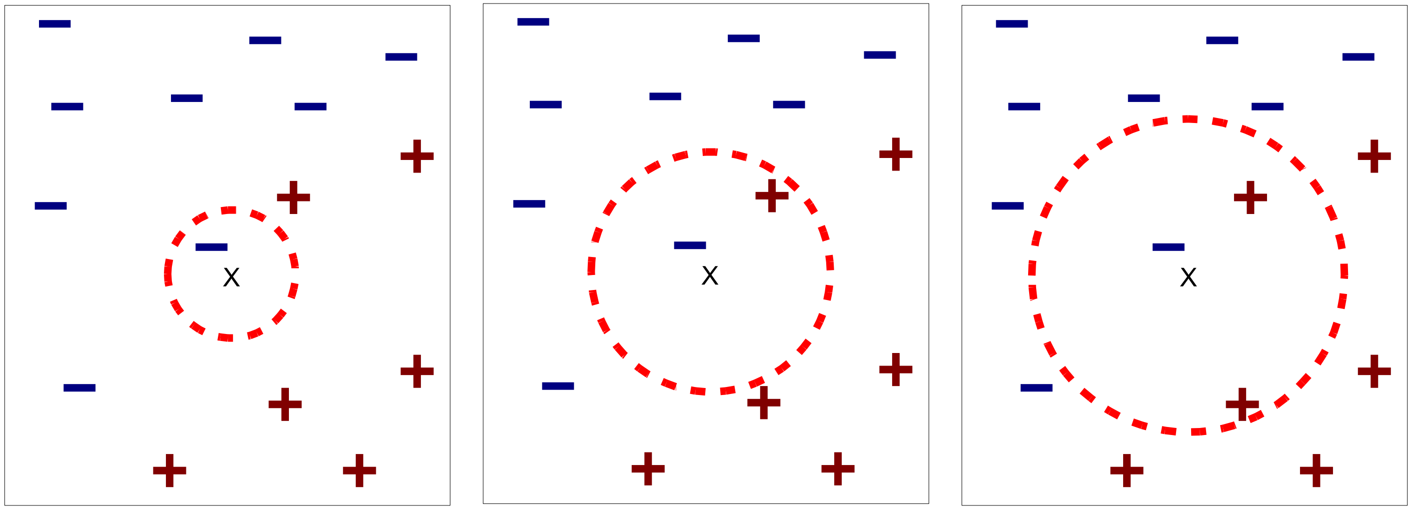
K Nearest Neighbor Classification

- 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, $w = 1/d^2$

Definition of Nearest Neighbor

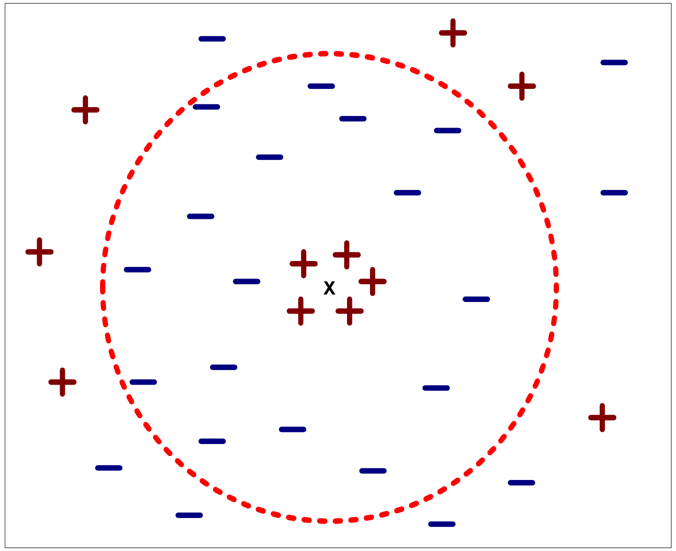


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

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

Nearest Neighbor Classification

- 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



The KNN Algorithm

Algorithm 6.2 The k-nearest neighbor classifier.

1: Let k be the number of nearest neighbors and D be the set of training examples.
2: **for** each test instance $z = (\mathbf{x}', y')$ **do**
3: Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
4: Select $D_z \subseteq D$, the set of k closest training examples to z.
5: $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
6: **end for**

- Here, v is a class lebel, y_i is the class label for one of the nearest neighbors, and $I(.)$ is an indicator function that returns 1 if its argument is true and 0 otherwise.

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

The Impacts of Distance functions

- $D_{sum}(A,B) = D_{gender}(A,B) + D_{age}(A,B) + D_{salary}(A,B)$
- $D_{norm}(A,B) = D_{sum}(A,B)/max(D_{sum})$
- $D_{euclid}(A,B) = sqrt(D_{gender}(A,B)^2 + D_{age}(A,B)^2 + D_{salary}(A,B)^2$

Table 9.9 New Customer

Recnum	*Gender*	*Age*	*Salary*
new	female	45	$100,000

Table 9.10 Set of Nearest Neighbors for New Customer

	1	*2*	*3*	*4*	*5*	*Neighbors*
$\mathbf{d_{sum}}$	1.662	1.659	1.338	1.003	1.640	4,3,5,2,1
$\mathbf{d_{norm}}$	0.554	0.553	0.446	0.334	0.547	4,3,5,2,1
$\mathbf{d_{euclid}}$	0.781	1.052	1.251	0.494	1.000	4,1,5,2,3

Table 9.11 Customers with Attrition History

Recnum	*Gender*	*Age*	*Salary*	*Attriter*
1	female	27	$19,000	no
2	male	51	$64,000	yes
3	male	52	$105,000	yes
4	female	33	$55,000	yes
5	male	45	$45,000	no
new	female	45	$100,000	?

Table 9.12 Using MBR to Determine if the New Customer Will Attrite

	Neighbors	*Neighbor Attrition*	*k = 1*	*k = 2*	*k = 3*	*k = 4*	*k = 5*
$\mathbf{d}_{\mathbf{sum}}$	4,3,5,2,1	Y,Y,N,Y,N	yes	yes	yes	yes	yes
$\mathbf{d}_{\mathbf{Euclid}}$	4,1,5,2,3	Y,N,N,Y,Y	yes	?	no	?	yes

Table 9.13 Attrition Prediction with Confidence

	k = 1	*k = 2*	*k = 3*	*k = 4*	*k = 5*
$\mathbf{d}_{\mathbf{sum}}$	yes, 100%	yes, 100%	yes, 67%	yes, 75%	yes, 60%
$\mathbf{d}_{\mathbf{Euclid}}$	yes, 100%	yes, 50%	no, 67%	yes, 50%	yes, 60%

Remarks on Lazy vs. Eager Learning

- Instance-based learning: lazy evaluation
- Decision-tree and Bayesian classification: eager evaluation
- Key differences
  - Lazy method may consider query instance *xq* when deciding how to generalize beyond the training data *D*
  - Eager method cannot since they have already chosen global approximation when seeing the query
- Efficiency: Lazy - less time training but more time predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form its implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

Artificial Neural Networks (ANN)

- **Basic Idea:** A complex non-linear function can be learned as a composition of simple processing units.

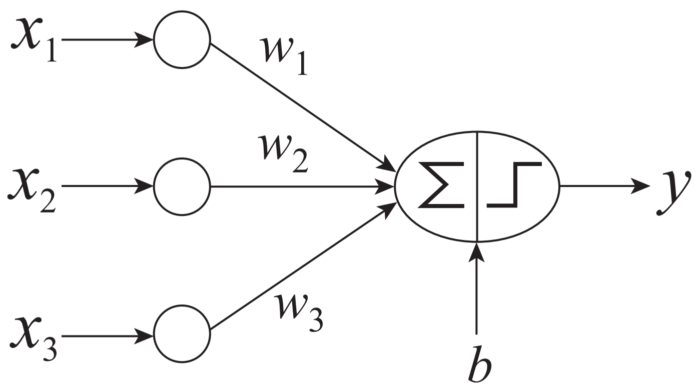
- ANN is a collection of simple processing units (nodes) that are connected by directed links (edges)
  - Every node receives signals from incoming edges, performs computations, and transmits signals to outgoing edges
  - Analogous to *human brain* where nodes are neurons and signals are electrical impulses
  - Weight of an edge determines the strength of connection between the nodes

- Simplest ANN: **Perceptron** (single neuron)

Why Do We Discuss Perceptron?

- We have discussed decision tree and k-NN.
- For decision tree, we use some of its attributes (features) to build the model.
- For k-NN, we use all its attributes/features/dimensions to clacify.
- Are there other ones?
- Perceptron is to learn weights for features, and is to find a linear decision boundary.

Basic Architecture of Perceptron



$$y = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + b > 0. \\ -1, & \text{otherwise.} \end{cases}$$

$$\hat{y} = sign(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}})$$

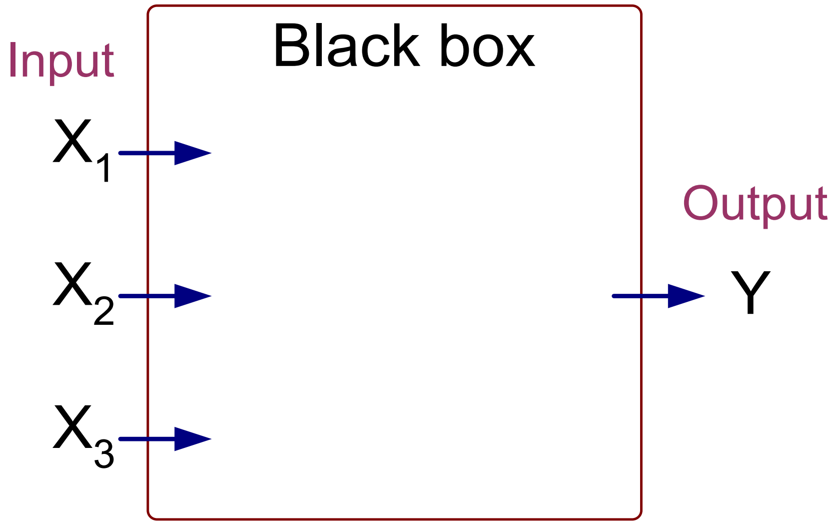
Activation Function

$$\tilde{\mathbf{w}} = (\mathbf{w}^T \ \ b)^T \qquad \tilde{\mathbf{x}} = (\mathbf{x}^T \ \ 1)^T$$

- Learns linear decision boundaries
- Related to logistic regression (activation function is sign instead of sigmoid)

Perceptron Example

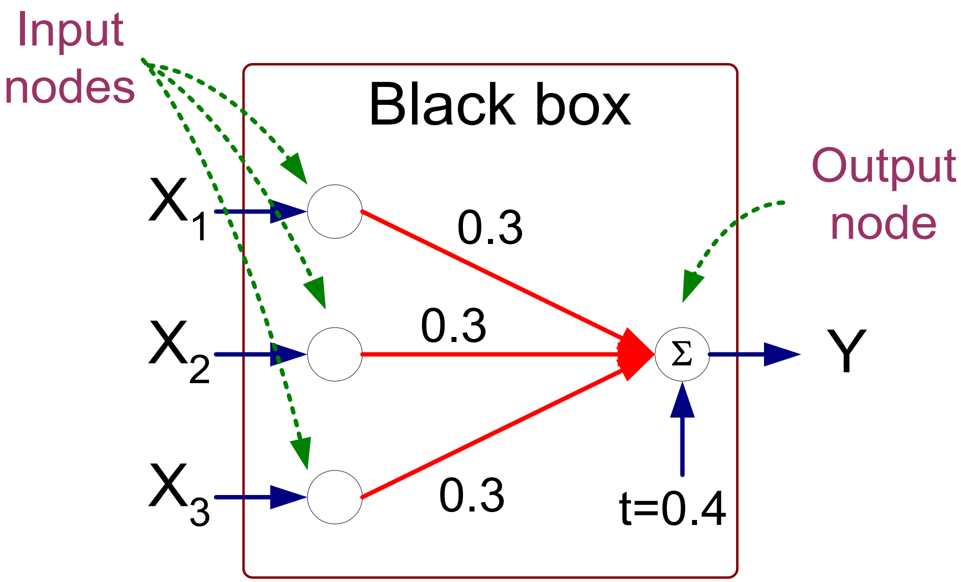
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Perceptron Example

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, ⋯, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute $\hat{y}_i$
    - Update the weights:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda\big(y_i - \hat{y}_i^{(k)}\big)x_{ij}$$

- Until stopping condition is met

- k: iteration number; λ: learning rate

Perceptron Learning Rule

- Weight update formula:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda\big(y_i - \hat{y}_i^{(k)}\big)x_{ij}$$

- Intuition:
  - Update weight based on error: e = $\big(y_i - \hat{y}_i\big)$
    - If y = $\hat{y}$, e=0: no update needed
    - If y > $\hat{y}$, e=2: weight must be increased (assuming Xij is positive) so that $\hat{y}$ will increase
    - If y < $\hat{y}$, e=-2: weight must be decreased (assuming Xij is positive) so that $\hat{y}$ will decrease

The Algorithm

Algorithm 6.3 Perceptron learning algorithm.

```
Let D.train = {(x̃_i, y_i) | i = 1, 2, ..., n} be the set of training instances.
Set k ← 0.
Initialize the weight vector w̃^(0) with random values.
repeat
    for each training instance (x̃_i, y_i) ∈ D.train do
        Compute the predicted output ŷ_i^(k) using w̃^(k).
        for each weight component w_j do
            Update the weight, w_j^(k+1) = w_j^(k) + λ(y_i − ŷ_i^(k)) x_ij.
        end for
        Update k ← k + 1.
    end for
until Σ_{i=1}^{n} |y_i − ŷ_i^(k)|/n is less than a threshold γ
```

In mathematical notation:

- Let $D.train = \{(\tilde{\mathbf{x}}_i, y_i) \mid i = 1, 2, \ldots, n\}$ be the set of training instances.
- Set $k \leftarrow 0$.
- Initialize the weight vector $\tilde{\mathbf{w}}^{(0)}$ with random values.
- **repeat**
  - **for** each training instance $(\tilde{\mathbf{x}}_i, y_i) \in D.train$ **do**
    - Compute the predicted output $\hat{y}_i^{(k)}$ using $\tilde{\mathbf{w}}^{(k)}$.
    - **for** each weight component w_j **do**
      - Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$.
    - **end for**
    - Update $k \leftarrow k + 1$.
  - **end for**
- **until** $\sum_{i=1}^{n} |y_i - \hat{y}_i^{(k)}|/n$ is less than a threshold γ

Example of Perceptron Learning

$$\lambda = 0.1$$

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

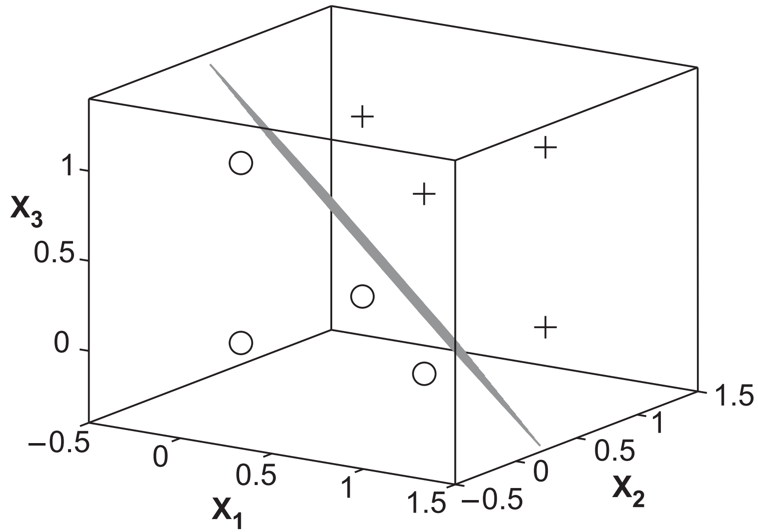
Weight updates over first epoch

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Weight updates over all epochs

Perceptron Learning

- Here, y is a linear combination of input variables, decision boundary is linear.



Overview

- Linear models
  - Perceptron: model (linear classifier) and learning algorithm (updating rules) combined as one.
  - Is there a better way to learn linear models?
- To separate models and learning algorithms
  - Learning as optimization
  - Surrogate loss function
  - Regularization

  } model design

  - Gradient descent
  - Batch and online gradients
  - Support vector machines

  } optimization

Learning as Optimization

$$\min_{\mathbf{w}} \sum_n \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0]$$

fewest mistakes

- The perceptron algorithm will find an optimal **w** if the data is separable
  - efficiency depends on the margin and norm of the data
- However, if the data is not separable, optimizing this is NP-hard
  - i.e., there is no efficient way to minimize this unless **P=NP**

Learning as Optimization

hyperparameter

$$\min_{\mathbf{w}} \sum_n \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0] + \lambda R(\mathbf{w})$$

fewest mistakes

simpler model

- In addition to minimizing training error, we want a simpler model
  - Remember our goal is to minimize generalization error
- We can add a regularization term *R(**w**)* that prefers simpler models
  - For example we may prefer decision trees of shallow depth
- Here λ is a hyperparameter of optimization problem

Learning as Optimization

hyperparameter

$$\min_{\mathbf{w}} \sum_n \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0] + \lambda R(\mathbf{w})$$

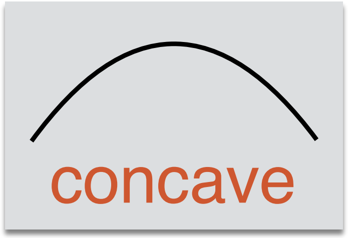
fewest mistakes

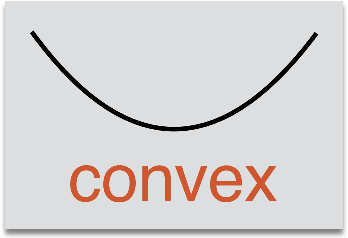
simpler model

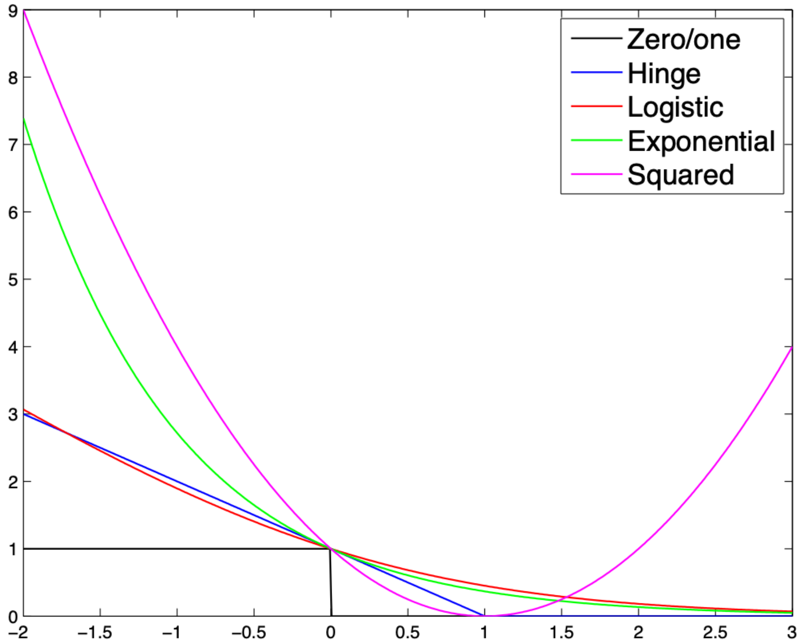
- The questions that remain are:
  - What are good ways to adjust the optimization problem so that there are efficient algorithms for solving it?
  - What are good regularizations ***R*(w)** for hyperplanes?
  - Assuming that the optimization problem can be adjusted appropriately, what algorithms exist for solving the regularized optimization problem?

Convex Surrogate Loss Functions

- Zero/one loss is hard to optimize
  - Small changes in **w** can cause large changes in the loss
- Surrogate loss: replace Zero/one loss by a smooth function
  - Easier to optimize if the surrogate loss is convex







$$y = +1 \qquad \hat{y} \leftarrow \mathbf{w}^T \mathbf{x}$$

$$\begin{aligned}
\text{Zero/one:} \quad & \ell^{(0/1)}(y, \hat{y}) = \mathbf{1}[y\hat{y} \leq 0] \\
\text{Hinge:} \quad & \ell^{(\text{hin})}(y, \hat{y}) = \max\{0, 1 - y\hat{y}\} \\
\text{Logistic:} \quad & \ell^{(\log)}(y, \hat{y}) = \frac{1}{\log 2} \log\left(1 + \exp[-y\hat{y}]\right) \\
\text{Exponential:} \quad & \ell^{(\exp)}(y, \hat{y}) = \exp[-y\hat{y}] \\
\text{Squared:} \quad & \ell^{(\text{sqr})}(y, \hat{y}) = (y - \hat{y})^2
\end{aligned}$$

Weight Regularization

- What are good regularization functions $R(\boldsymbol{w})$ for hyperplanes?
- We would like the weights —
  - To be small —
    - Change in the features cause small change to the score
    - Robustness to noise
  - To be sparse —
    - Use as few features as possible
    - Similar to controlling the depth of a decision tree
- This is a form of inductive bias

Weight Regularization

- Just like the surrogate loss function, we would like *R(**w**)* to be convex.
- Small weights regularization

$$R^{(\mathrm{norm})}(\mathbf{w}) = \sqrt{\sum_d w_d^2} \qquad\qquad R^{(\mathrm{sqrd})}(\mathbf{w}) = \sum_d w_d^2$$

- Sparsity regularization

$$R^{(\mathrm{count})}(\mathbf{w}) = \sum_d \mathbf{1}[|w_d| > 0]$$

not convex

- Family of "p-norm" regularization

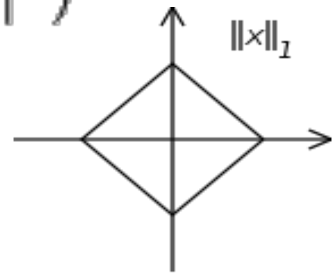
$$R^{(\text{p-norm})}(\mathbf{w}) = \left(\sum_d |w_d|^p\right)^{1/p}$$

Contours of p-norms

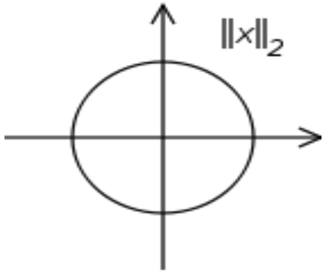
$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

convex for $p \geq 1$

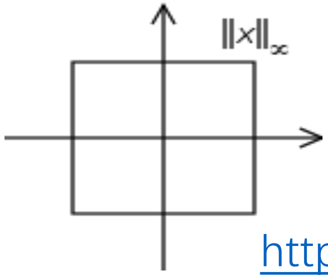
$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$



$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$



$$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$$



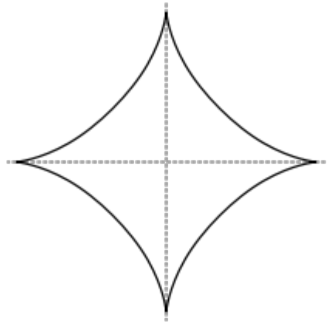
http://en.wikipedia.org/wiki/Lp_space

Contours of p-norms

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

$\boxed{\text{not convex for } 0 \leq p < 1}$

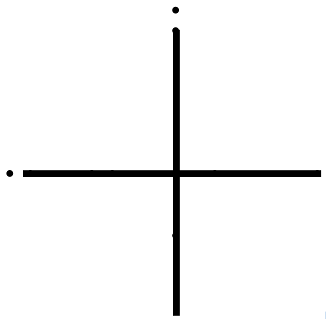
$$p = \frac{2}{3}$$



[Counting non-zeros:](http://en.wikipedia.org/wiki/Lp_space)

$$R^{(\text{count})}(\mathbf{w}) = \sum_d \mathbf{1}[|w_d| > 0]$$

$$p = 0$$



http://en.wikipedia.org/wiki/Lp_space

General Optimization Framework

hyperparameter

$$\min_{\mathbf{w}} \sum_n \ell\left(y_n, \mathbf{w}^T \mathbf{x}_n\right) + \lambda R(\mathbf{w})$$

surrogate loss

regularization

- Select a suitable:
  - convex surrogate loss
  - convex regularization
- Select the hyperparameter λ
- Minimize the regularized objective with respect to **w**
- This framework for optimization is called Tikhonov regularization or
- generally Structural Risk Minimization (SRM)

http://en.wikipedia.org/wiki/Tikhonov_regularization

The Gradient-based Approaches

- It is like blindfolded mountain climbing.
- To find the max of a function f(x), the optimizer keeps the current estimation about x, measures the gradient of the parameters of x, and take a step along the direction of the gradient as $x \leftarrow x + \eta g$, where η is the step size.

Gradient Descent

- A gradient is a multidimensional generalization of a derivative.
- Consider a function $f : \mathbb{R}^D \to \mathbb{R}$with a vector input $x = \langle x_1, x_2, \dots, x_D \rangle$ process a scalar value output y.
- To differentiate f with one of the input, x_i using $\frac{\partial f}{\partial x_i}$.
- The gradient of f is the vector with the derivate f with respect to every x_i.

The Gradient Descent Algorithm

Algorithm 21 GradientDescent($\mathcal{F}, K, \eta_1, \ldots$)

$z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing

for $k = 1 \ldots K$ **do**

  $g^{(k)} \leftarrow \nabla_z \mathcal{F}\big|_{z^{(k-1)}}$ // compute gradient at current location

  $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient

end for

return $z^{(K)}$

An Example (1)

hyperparameter

$$\min_{\mathbf{w}} \sum_{n} \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0] + \lambda R(\mathbf{w})$$

fewest mistakes

simpler model

- Here, let the exponential loss $\ell^{(exp)}(y, \hat{y}) = \exp(-y\hat{y})$ be a loss function , and the 2-norm $R(w) = \frac{1}{2}(\sum_d |w_d|^2)^{1/2}$ be a regularizer.

$$\mathcal{L}(w, b) = \sum_n \exp\left[-y_n(w \cdot x_n + b)\right] + \frac{\lambda}{2}||w||^2$$

An Example (2)

$$\mathcal{L}(\boldsymbol{w}, b) = \sum_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

- The derivatives with respect to b is as follows, and update $b \leftarrow b - \eta\frac{\partial \mathcal{L}}{\partial b}$.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b}\sum_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \frac{\partial}{\partial b}\frac{\lambda}{2}||\boldsymbol{w}||^2 \\
&= \sum_n \frac{\partial}{\partial b}\exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + 0 \\
&= \sum_n \left(\frac{\partial}{\partial b} - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right)\exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] \\
&= -\sum_n y_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right]
\end{aligned}$$

An Example (3)

$$\mathcal{L}(\boldsymbol{w}, b) = \sum_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \frac{\lambda}{2} ||\boldsymbol{w}||^2$$

- The derivatives with respect to w is as follows, and update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \mathcal{L}$.

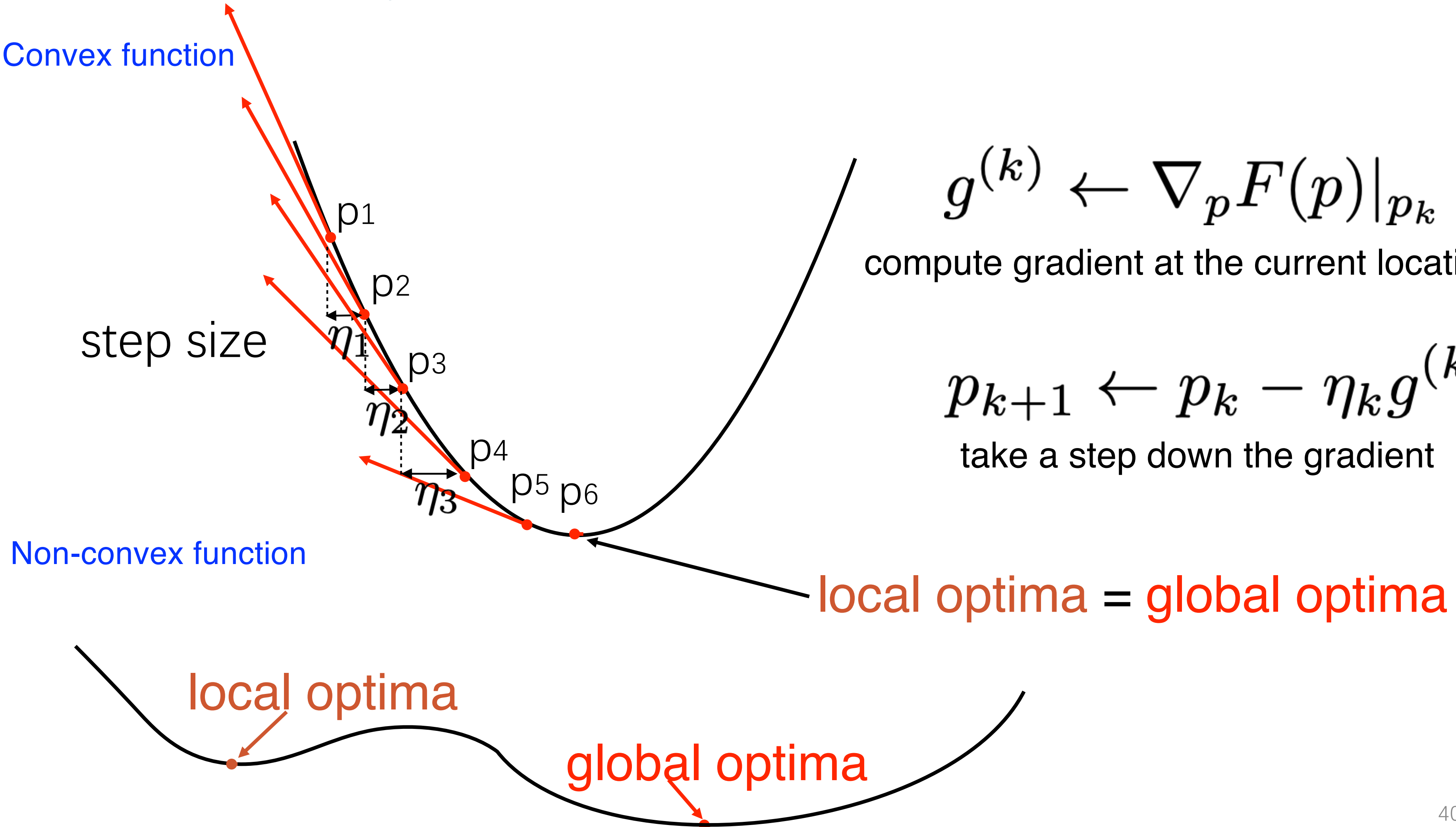
$$\nabla_{\boldsymbol{w}} \mathcal{L} = \nabla_{\boldsymbol{w}} \sum_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \nabla_{\boldsymbol{w}} \frac{\lambda}{2} ||\boldsymbol{w}||^2$$

$$= \sum_n (\nabla_{\boldsymbol{w}} - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)) \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \lambda \boldsymbol{w}$$

$$= -\sum_n y_n \boldsymbol{x}_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \lambda \boldsymbol{w}$$

- For poorly classified points, the gradient points is in the direction $-y_n x_n$, so the update is $w \leftarrow w + c y_n x_n$ where c is a scalar value, the update for the part of the gradient related to the regularizer is $w \leftarrow (1 - \lambda) w$.

Optimization by Gradient Descent



$$g^{(k)} \leftarrow \nabla_p F(p)|_{p_k}$$

compute gradient at the current location

$$p_{k+1} \leftarrow p_k - \eta_k g^{(k)}$$

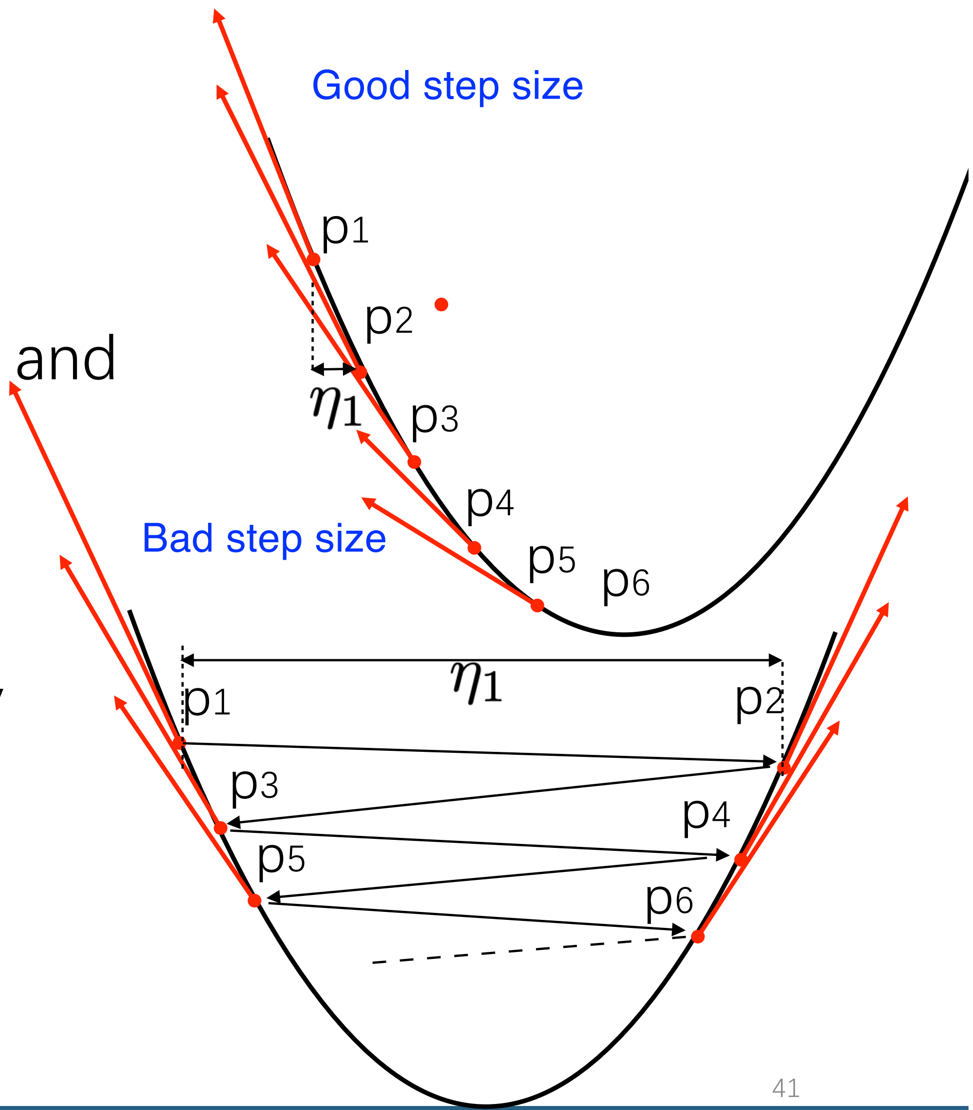
take a step down the gradient

Choice of Step Size

- The step size is important —
  - too small: slow convergence
  - too large: no convergence
- A strategy is to use large step sizes initially and small step sizes later:

$$\eta_t \leftarrow \eta_0/(t_0 + t)$$

- There are methods that converge faster by adapting step size to the curvature of the function
  - Field of convex optimization



http://stanford.edu/~boyd/cvxbook/

Exponential Loss

$$\mathcal{L}(\mathbf{w}) = \sum_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \frac{\lambda}{2}||\mathbf{w}||^2$$ objective

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -y_n \mathbf{x}_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \lambda \mathbf{w}$$ gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\sum_n -y_n \mathbf{x}_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \lambda \mathbf{w} \right)$$ update

loss term

$$\mathbf{w} \leftarrow \mathbf{w} + c y_n \mathbf{x}_n$$

high for misclassified points

regularization term

$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w}$$

shrinks weights towards zero

Batch and Online Gradients

$$\mathcal{L}(\mathbf{w}) = \sum_n \mathcal{L}_n(\mathbf{w})$$ objective

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{d\mathcal{L}}{d\mathbf{w}}$$ gradient descent

batch gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\sum_n \frac{d\mathcal{L}_n}{d\mathbf{w}} \right)$$

sum of n gradients

update weight after you see all points

online gradient

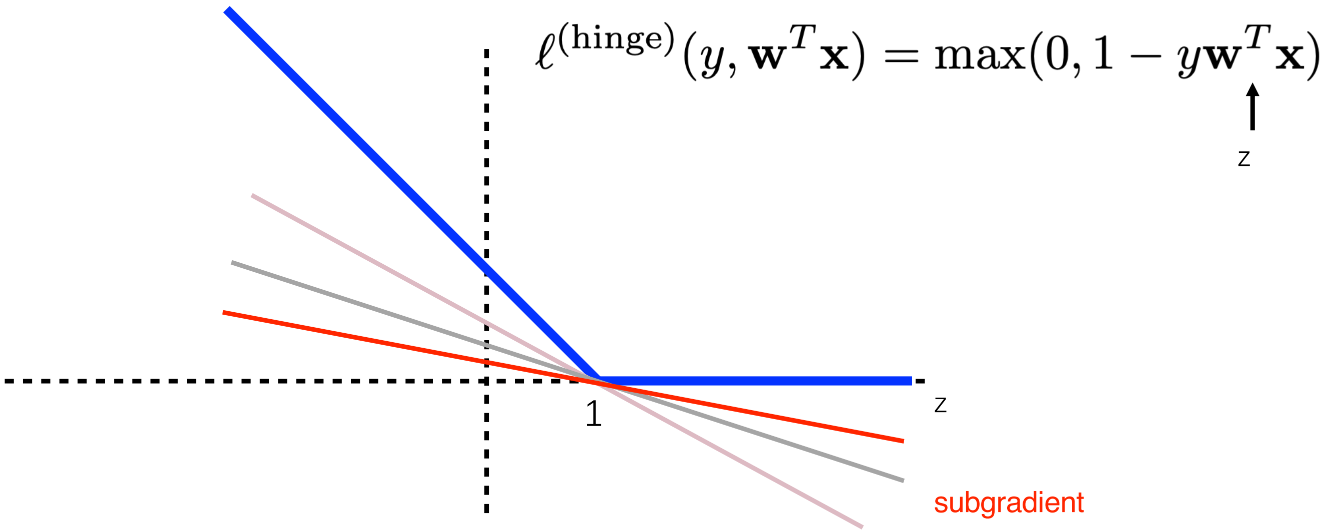
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\frac{d\mathcal{L}_n}{d\mathbf{w}} \right)$$

gradient at n^{th} point

update weights after you see each point

Online gradients are the default method for multi-layer perceptrons

Subgradient



- The hinge loss is not differentiable at z=1
- Subgradient is any direction that is below the function
- For the hinge loss a possible subgradient is:

$$\frac{d\ell^{\text{hinge}}}{d\mathbf{w}} = \begin{cases} 0 & \text{if } y\mathbf{w}^T\mathbf{x} > 1 \\ -y\mathbf{x} & \text{otherwise} \end{cases}$$

Closed-form Optimization for Squared loss

- Gradient descent is a good, generic optimization algorithm.
- There are cases where we can do better than gradient descent.
- One such case is squared error loss function and 2-norm regularizer.
- We can get a closed-form solution!
- Let training data be a matrix $\boldsymbol{X}$ of size $N \times D$, where $x_{n,d}$ is the value of the dth feature on the nth record, label as a vector $\boldsymbol{Y}$ of dimension N, and weight as a column vector $\boldsymbol{w}$ of size D.
- We have $\boldsymbol{a} = \boldsymbol{X}\boldsymbol{w}$, which has dimension N. That is $a_n = [\boldsymbol{X}\boldsymbol{w}]_n = \sum_d X_{n,d} w_d$.
- Here, we can take $\boldsymbol{a}$ as $\boldsymbol{Y}$.
- We should minimize $\frac{1}{2}\sum_n \left(\hat{Y}_n - Y_n\right)^2$ following squared error. In vector form, it is $\frac{1}{2}\left\|\widehat{\boldsymbol{Y}} - \boldsymbol{Y}\right\|^2$

Closed-form Optimization for Squared loss

$$\mathcal{L}(\mathbf{w}) = \sum_n \left(y_n - \mathbf{w}^T \mathbf{x}_n\right)^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

objective

matrix notation

$$\underbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}}_{\boldsymbol{w}} = \underbrace{\begin{bmatrix} \sum_d x_{1,d} w_d \\ \sum_d x_{2,d} w_d \\ \vdots \\ \sum_d x_{N,d} w_d \end{bmatrix}}_{\hat{\boldsymbol{Y}}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\hat{\boldsymbol{Y}}}$$

equivalent loss

$$\min_{\boldsymbol{w}} \quad \mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\,||\mathbf{X}\boldsymbol{w} - \boldsymbol{Y}||^2 + \frac{\lambda}{2}\,||\boldsymbol{w}||^2$$

Closed-form Optimization for Squared loss

$$\min_{w} \quad \mathcal{L}(w) = \frac{1}{2}\,||\mathbf{X}w - Y||^2 + \frac{\lambda}{2}\,||w||^2$$

objective

$$\begin{aligned}\nabla_w \mathcal{L}(w) &= \mathbf{X}^\top (\mathbf{X}w - Y) + \lambda w \\ &= \mathbf{X}^\top \mathbf{X}w - \mathbf{X}^\top Y + \lambda w \\ &= \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right) w - \mathbf{X}^\top Y\end{aligned}$$

gradient

At optima the gradient = 0

$$\begin{aligned} & \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right) w - \mathbf{X}^\top Y = 0 \\ \iff \quad & \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right) w = \mathbf{X}^\top Y \\ \iff \quad & w = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right)^{-1} \mathbf{X}^\top Y\end{aligned}$$

exact closed-form solution

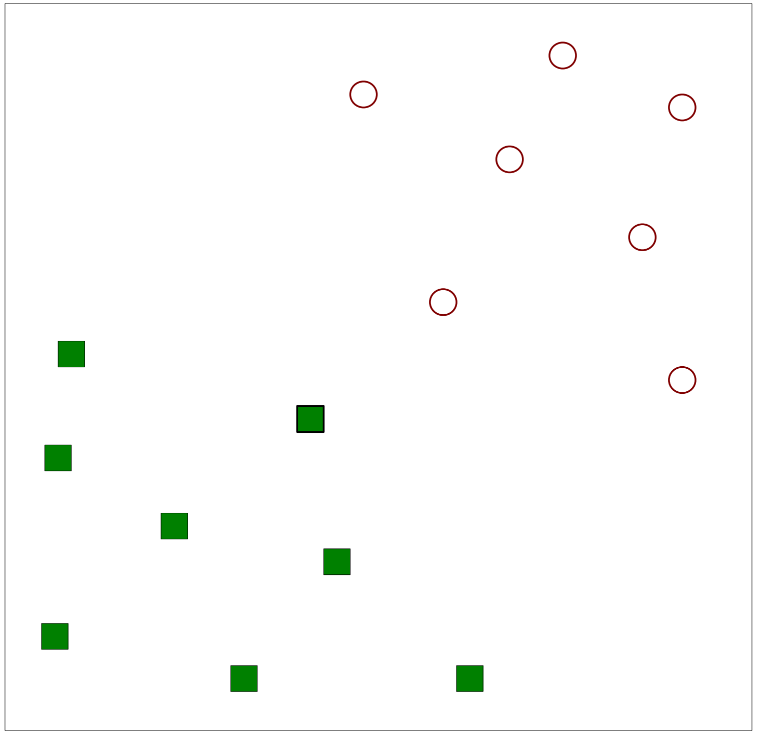
Matrix inversion vs. gradient descent

- Assume, we have D features and N points
- Overall time via matrix inversion
  - The closed form solution involves computing:

$$w = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right)^{-1} \mathbf{X}^\top \mathbf{Y}$$

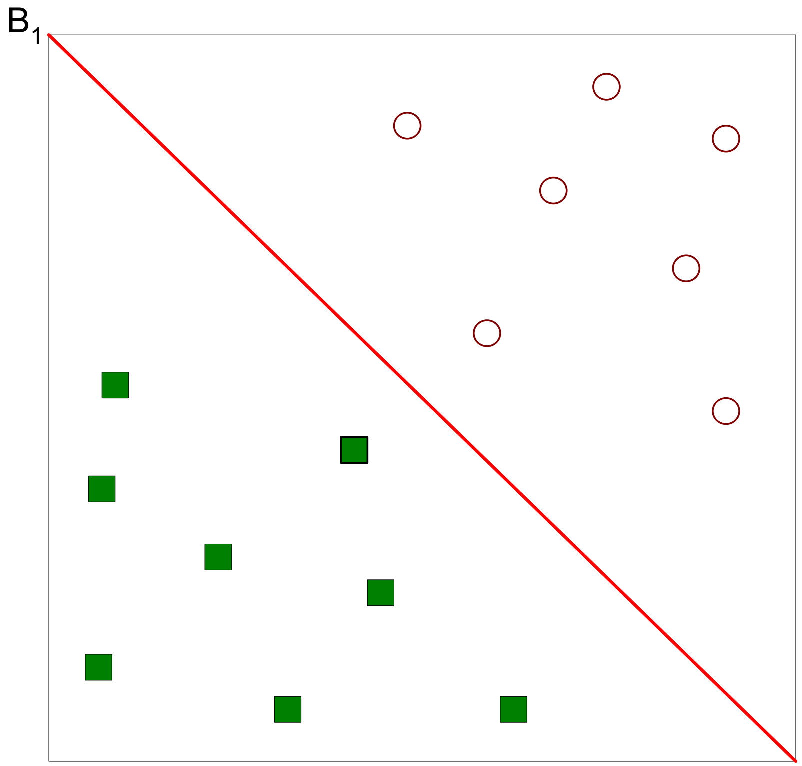
  - Total time is $O(D^2N + D^3 + DN)$, assuming $O(D^3)$ matrix inversion
  - If $N > D$, then total time is $O(D^2N)$
- Overall time via gradient descent
  - Gradient: $\dfrac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -2(y_n - \mathbf{w}^T \mathbf{x}_n)\mathbf{x}_n + \lambda \mathbf{w}$
  - Each iteration: $O(ND)$; T iterations: $O(TND)$
- Which one is faster?
  - Small problems $D < 100$: probably faster to run matrix inversion
  - Large problems $D > 10{,}000$: probably faster to run gradient descent

Support Vector Machines



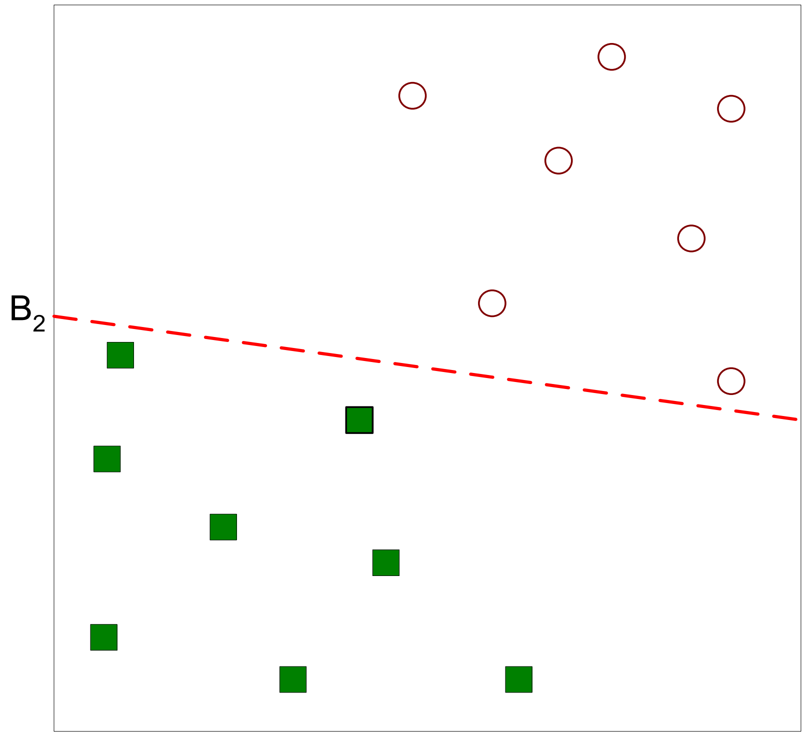
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



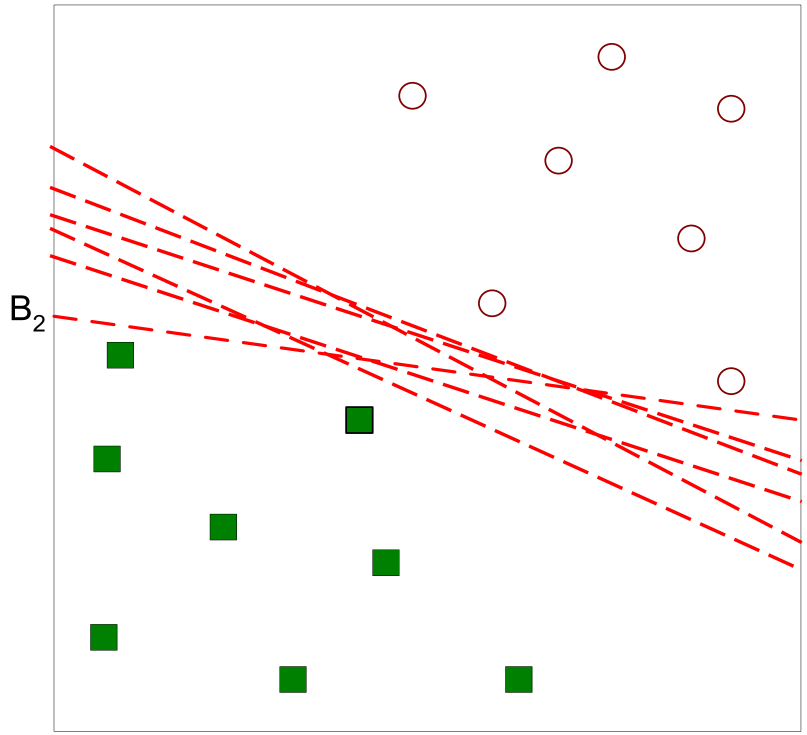
- One Possible Solution

Support Vector Machines



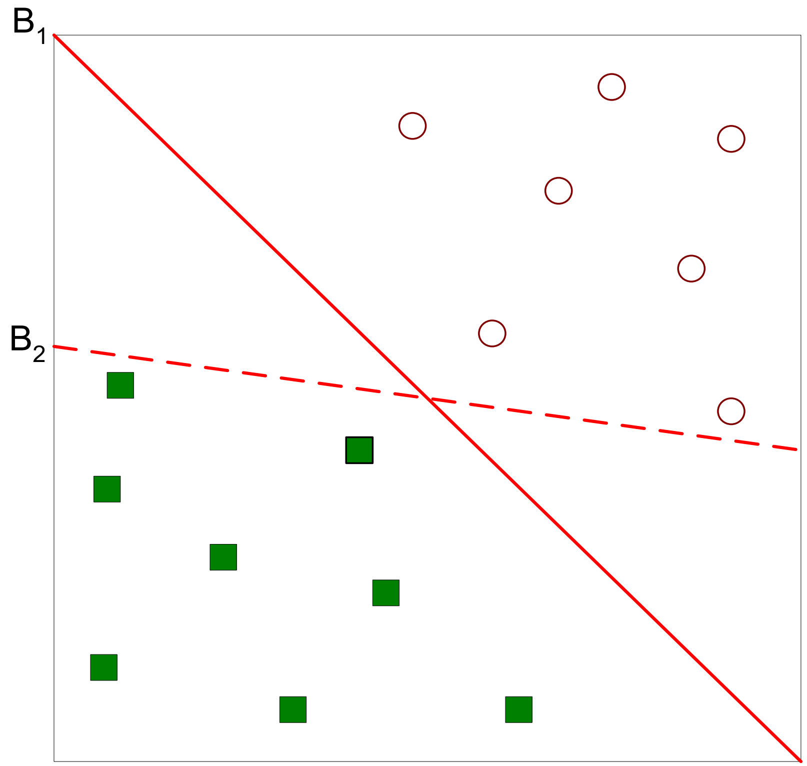
- Another possible solution

Support Vector Machines



- Other possible solutions

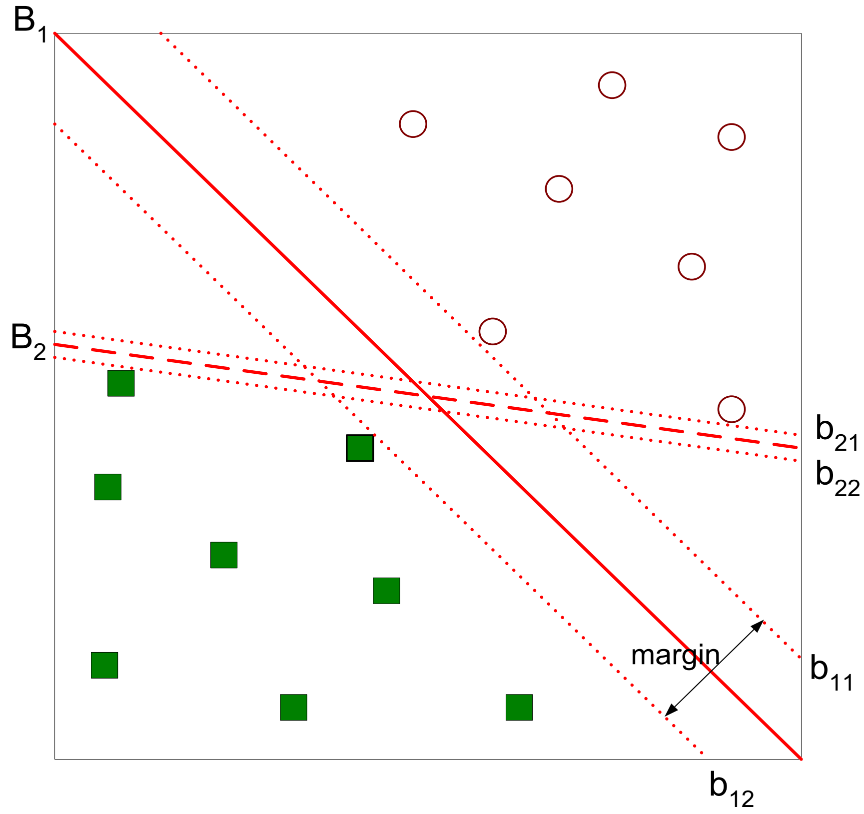
Support Vector Machines



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

Support Vector Machines

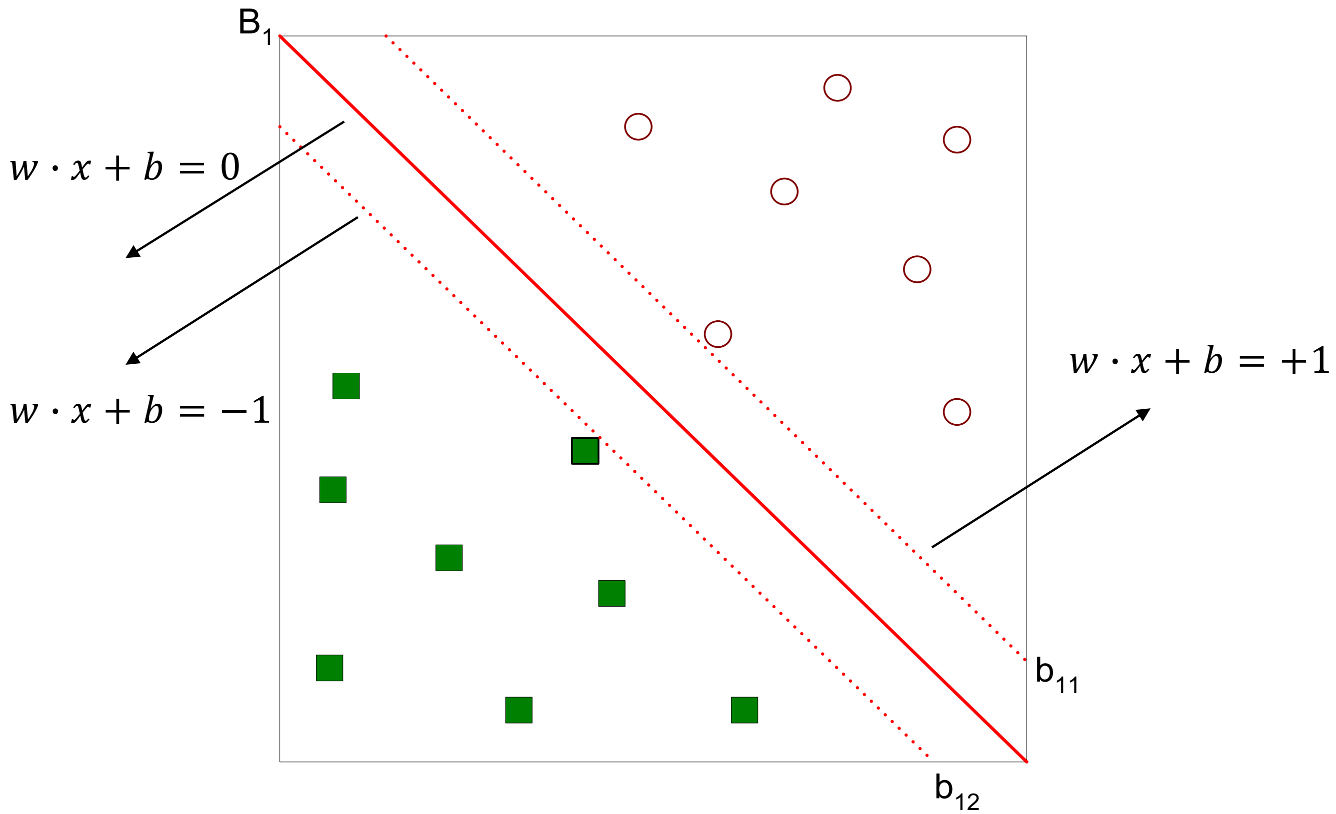
- For evert separating hyperplane B_i, there is a pair of margin hyperplanes, b_{i1} and b_{i2} such that they touch the closest instances of classes.
- The margin is the distance between the pair of margin hyperplanes.
- Find the hyperplane that maximizes the margin => B1 is better than B2.
- A larger margin tends to have abetter generalization error.



Support Vector Machines

- The distance of any point x from the hyperplane is $D(x) = \frac{|w \cdot x + b|}{||w||}$
- Let the closest points from the hyperplane be k_+ and k_-.
- We have the constraints

$$\frac{\mathbf{w}^T \mathbf{x}_i + b}{||\mathbf{w}||} \geq k_+ \qquad \text{if } y_i = 1,$$
$$\frac{\mathbf{w}^T \mathbf{x}_i + b}{||\mathbf{w}||} \leq -k_- \qquad \text{if } y_i = -1.$$



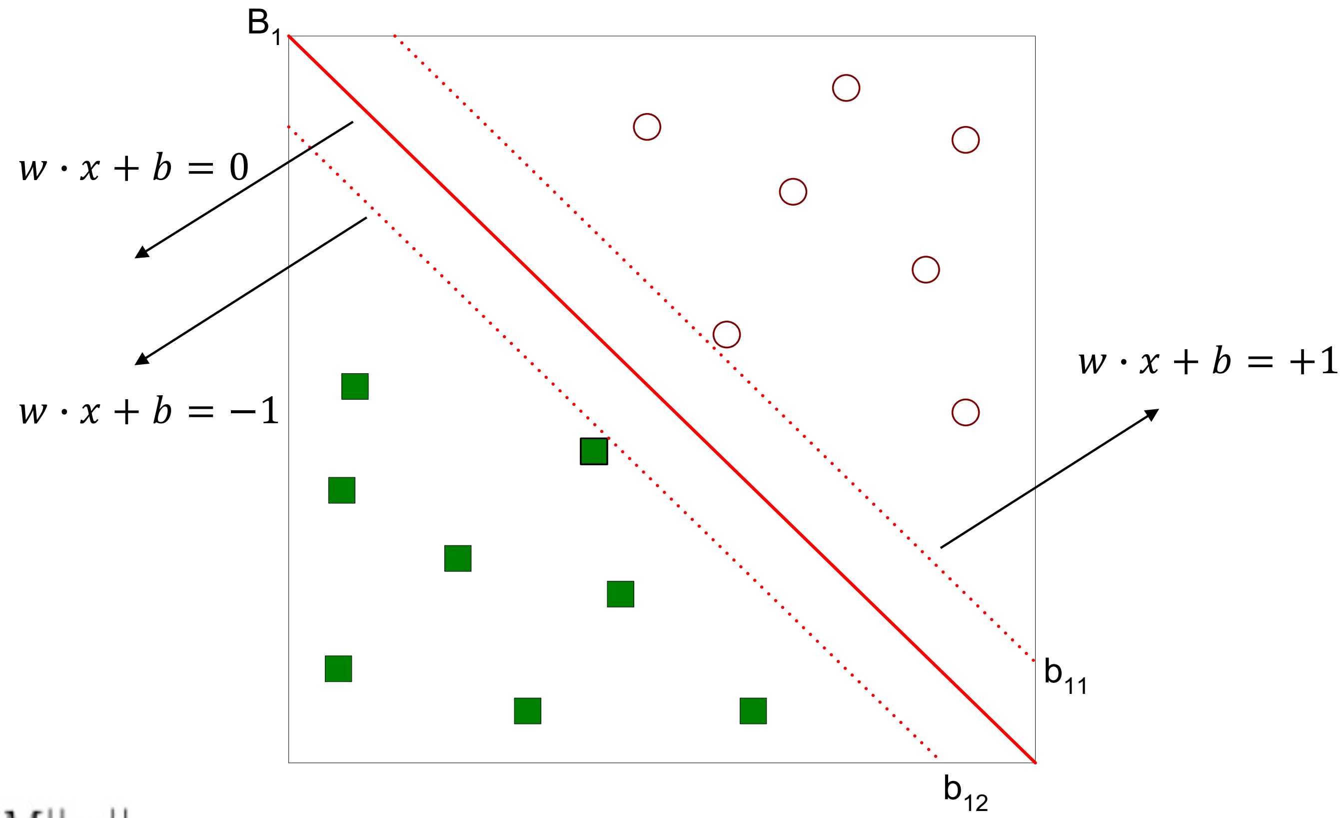
$$\text{Margin} = \frac{2}{\|\vec{w}\|}$$

Support Vector Machines

- Then, we have the product of y_i and $(w^T x_i + b)$ as $y_i(w^T x_i + b) \geq M\|w\|$ where M is a parameter.
- If $k_+ = \mathrm{M}$ and $k_- = \mathrm{M}$, then margin = $k_+ - k_- = 2\mathrm{M}$.
- The optimization problem is:

$$\max_{\mathbf{w},b} M$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq M\|\mathbf{w}\|.$$



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

Linear SVM

- We measure the distance as follows, $k_+ = \frac{1}{\|w\|} w \cdot x_+ + b - 1$, and $k_- = \frac{1}{\|w\|} w \cdot x_- + b - 1$.
- $M = \frac{1}{2}[k_+ - k_-] = \frac{1}{2}\left[\frac{1}{\|w\|} w \cdot x_+ + b - 1 - \frac{1}{\|w\|} w \cdot x_- - b + 1\right] = \frac{1}{2}\left[\frac{1}{\|w\|} w \cdot x_+ - \frac{1}{\|w\|} w \cdot x_-\right] = \frac{1}{2}\left[\frac{1}{\|w\|}(1 - b) - \frac{1}{\|w\|}(-1 - b)\right] = \frac{1}{\|w\|}$
- The size of the margin M is inversely proportional to the norm of the weight vector $\|w\|$.
- Maximize M amounts to minimizing $\|w\|^2$. The SVM is commonly represented as

$$\min_{\mathbf{w},b} \frac{||\mathbf{w}||^2}{2}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1.$$

Learning Model Parameters

$$\min_{\mathbf{w},b} \frac{||\mathbf{w}||^2}{2}$$
$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1.$$

- The equation above represents a constrained optimization problem with linear inequalities. And the objective function is convex and quadratic with respect to w.
- We can rewrite the objective function, which is known Lagrangian primal problem $L_P = \frac{1}{2}\|w\|^2 - \sum_i \lambda_i [y_i(w^T x_i + b) - 1]$, where the parameters $\lambda_i \geq 0$ correspond to the constraints and are called the Lagrange multipliers.

SVM Target

- Optimization (Quadratic Programming):

$$\min_{w,b} \frac{\|w\|^2}{2}$$
$$y_i(w^T x_i + b) \geq 1, \forall i$$

- Solved by Lagrange multiplier method:

$$L_P = \frac{1}{2}\|w\|^2 - \sum_i \lambda_i [y_i(w^T x_i + b) - 1]$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier

Lagrangian

- Consider optimization problem:

$$\min_{w} \; f(w)$$
$$h_i(w) = 0, \forall 1 \leq i \leq l$$

- Lagrangian:

$$\mathcal{L}(w, \boldsymbol{\beta}) = f(w) + \sum_i \beta_i h_i(w)$$

where β_i's are called Lagrange multipliers

Lagrangian

- Consider optimization problem:

$$\min_{w} \ f(w)$$

$$h_i(w) = 0, \forall 1 \leq i \leq l$$

- Solved by setting derivatives of Lagrangian to 0

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0; \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0$$

Generalized Lagrangian

- Consider optimization problem:

$$\min_{w} f(w)$$

$$g_i(w) \leq 0, \forall 1 \leq i \leq k$$

$$h_j(w) = 0, \forall 1 \leq j \leq l$$

- Generalized Lagrangian:

$$\mathcal{L}(w, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(w) + \sum_i \alpha_i g_i(w) + \sum_j \beta_j h_j(w)$$

where α_i, β_j's are called Lagrange multipliers

Lagrange Duality

- The primal problem

$$p^* \coloneqq \min_{w} f(w) = \min_{w} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \geq 0} \mathcal{L}(w, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

- The dual problem

$$d^* \coloneqq \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \geq 0} \min_{w} \mathcal{L}(w, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

- Always true:

$$d^* \leq p^*$$

Lagrange Duality

- Theorem: under proper conditions, there exists $(w^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ such that

$$d^* = \mathcal{L}(w^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = p^*$$

Moreover, $(w^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ satisfy Karush-Kuhn-Tucker (KKT) conditions:

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0, \qquad \alpha_i g_i(w) = 0$$

$$g_i(w) \leq 0,\ \ h_j(w) = 0, \qquad \alpha_i \geq 0$$

Learning Model Parameters

$$\min_{\mathbf{w},b} \frac{||\mathbf{w}||^2}{2}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1.$$

Given $L_P = \frac{1}{2}\|w\|^2 - \sum_i \lambda_i[y_i(w^T x_i + b) - 1]$, where the parameters $\lambda_i \geq 0$ correspond to the constraints and are called the Lagrange multipliers, to minimize the Lagrangian, we have,

$\frac{\partial L_P}{\partial w} = 0, \rightarrow w = \sum_i \lambda_i y_i x_i$

$\frac{\partial L_P}{\partial b} = 0, \rightarrow 0 = \sum_i \alpha_i y_i$

With the Karash-Kuhn-Tucker (KKT) conditions between (w, b) and λ_i, we also have $\lambda_i[y_i(w^T x_i + b) - 1] = 0$.

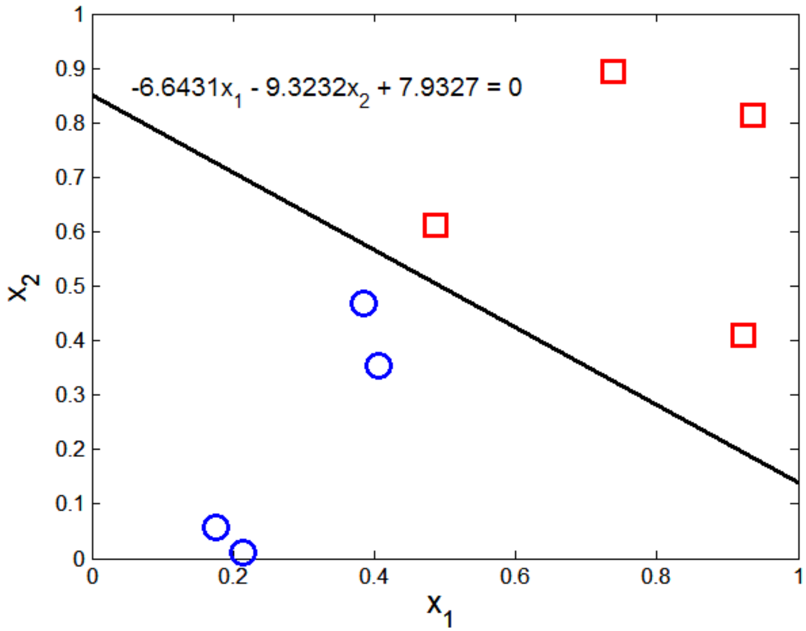
Learning Model Parameters

$$\min_{\mathbf{w},b} \frac{||\mathbf{w}||^2}{2}$$
$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1.$$

With all together, we have

$$\begin{aligned} \max_{\lambda_i} \quad & \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i\lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to} \quad & \sum_{i=1}^{n} \lambda_i y_i = 0, \\ & \lambda_i \geq 0. \end{aligned}$$

An Example of Linear SVM

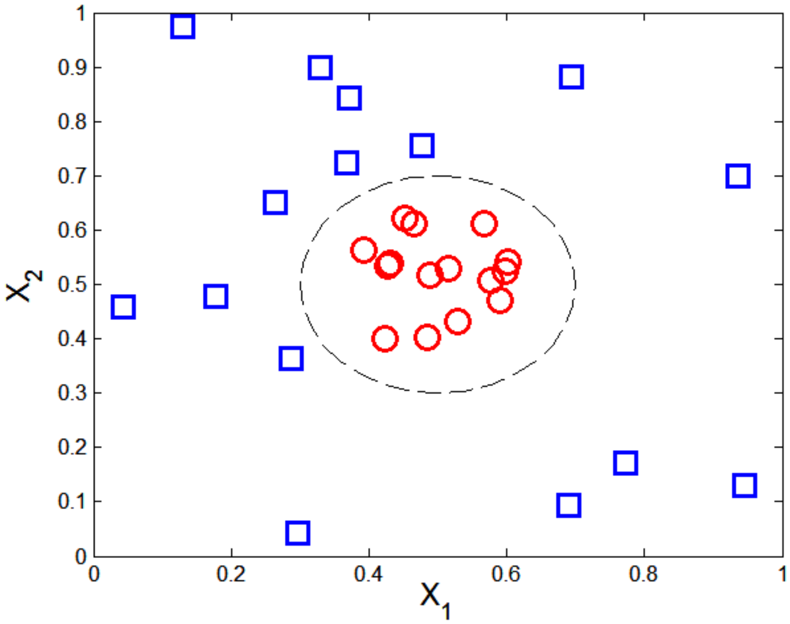


Support vectors

x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

Nonlinear Support Vector Machines

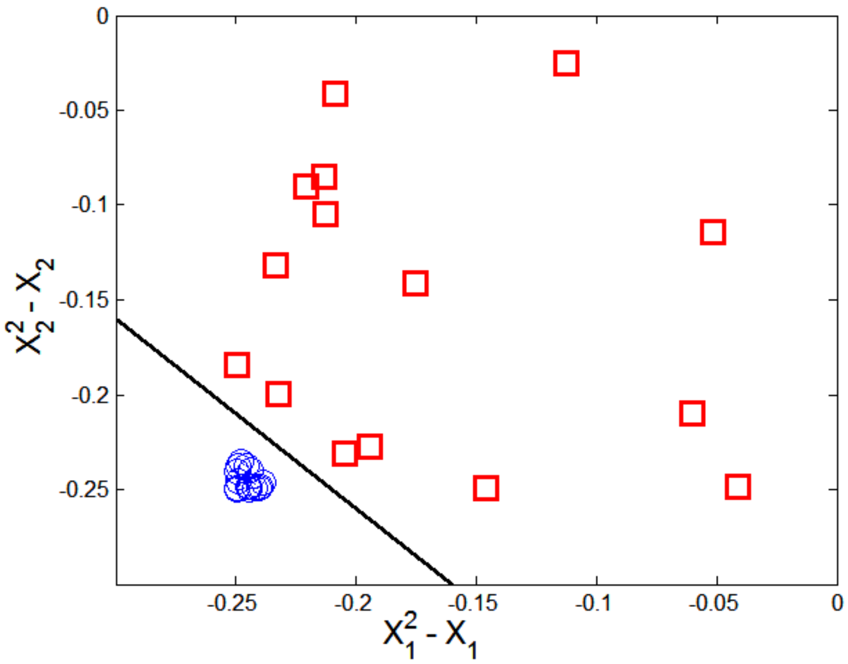
- What if decision boundary is not linear?



$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1 & \text{otherwise} \end{cases}$$

Nonlinear Support Vector Machines

- Transform data into higher dimensional space



$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46.$$

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

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

Decision boundary:

$$\vec{w} \bullet \Phi(\vec{x}) + b = 0$$

Characteristics of SVM

- The learning problem is formulated as a convex optimization problem
  - Efficient algorithms are available to find the global minima
  - Many of the other methods use greedy approaches and find locally optimal solutions
  - High computational complexity for building the model
- Robust to noise
- Overfitting is handled by maximizing the margin of the decision boundary,
- SVM can handle irrelevant and redundant attributes better than many other techniques
- The user needs to provide the type of kernel function and cost function
- Difficult to handle missing values
- What about categorical variables?

Slides Credit

[1] Subhransu Maji. Linear model in CMPSCI689.

[2] Yingyu Liang. SVM II in COS495

[3] Introduction to Machine Learning 2nd edition, Pang-Ning Tan, Michael Steinbach, Vipin Kumar, and Anuj Karpatne

[4] A Course in Machine Learning, Hal Daume III.