EECS 545: Machine Learning

Lecture 6. Classification 3

Honglak Lee

1/31/2020

Announcements

- HW1 due by today, 23:55
- HW2 due by Feb/11, 23:55
- Google calendar for the class schedule
  - Subscribe it!

Outline

- Probabilistic generative models
  - Gaussian discriminant analysis (already covered)
  - Naive Bayes
- Discriminant functions
  - Fisher's linear discriminant
  - Perceptron learning algorithm

Recap: Learning the Classifier

- Goal: Learn the distributions $p(C_k\ |\mathbf{x})$.

(a) Discriminative models: Directly model $p(C_k|\mathbf{x})$ and learn parameters from the training set.

- Logistic regression
- Softmax regression

(b) Generative models: Learn class densities $p(\mathbf{x}|C_k)$ and priors $p(C_k)$

- Gaussian Discriminant Analysis
- Naive Bayes

Recap: Learning the Classifier

- Goal: Learn the distributions $p(C_k \,|\, \mathbf{x})$.

(a) Discriminative models: Directly model $p(C_k|\mathbf{x})$ and learn parameters from the training set.

- Logistic regression
- Softmax regression

(b) Generative models: Learn class densities $p(\mathbf{x}|C_k)$ and priors $p(C_k)$

- Gaussian Discriminant Analysis
- **Naive Bayes**

Naive Bayes classifier

Naive Bayes classifier

- Prior distribution:
  - $p(C_k)$: Constant (e.g., Bernoulli)
- Likelihood
  - Naive Bayes assumption: $P(\mathbf{x}|C_k)$ is factorized
    (Each coordinate of **x** is conditionally independent of other coordinates given the class label)

$$P(x_1, ..., x_M|C_k) = P(x_1|C_k) \cdots P(x_M|C_k) = \prod_{j=1}^{M} P(x_j|C_k)$$

- Classification: use Bayes rule

(binary) $$P(C_1|\mathbf{x}) = \frac{P(C_1, \mathbf{x})}{P(\mathbf{x})} = \frac{P(C_1, \mathbf{x})}{P(C_1, \mathbf{x}) + P(C_2, \mathbf{x})}$$

Naive Bayes classifier

- When classifying, we can simply take the MAP (Maximum a Posteriori) estimation:

$$\arg\max_k P(C_k|\mathbf{x}) = \arg\max_k P(C_k, \mathbf{x})$$

Naive Bayes classifier

- When classifying, we can simply take the MAP (Maximum a Posteriori) estimation:

$$\begin{aligned} \arg\max_k P(C_k|\mathbf{x}) &= \arg\max_k P(C_k, \mathbf{x}) \\ &= \arg\max_k P(C_k) P(\mathbf{x}|C_k) \end{aligned}$$

Naive Bayes classifier

- When classifying, we can simply take the MAP (Maximum a Posteriori) estimation:

$$\arg\max_k P(C_k|\mathbf{x}) = \arg\max_k P(C_k, \mathbf{x})$$

$$= \arg\max_k P(C_k)P(\mathbf{x}|C_k)$$

Naive Bayes assumption

$$= \arg\max_k P(C_k)\prod_{j=1}^{M} P(x_j|C_k)$$

Example: Spam mail classification

- Label: y=1 (spam), y=0 (non-spam)
- Features:
  - x_j: j-th word in the mail, where M is the vocabulary size.
  - Multinomial variable (M-dimensional binary vector with only one coordinate with 1)
- Naive Bayes Assumption:
  - Given a class label y, each word in a mail is a independent multinomial variable.

Naive Bayes Spam classifier

- Model

$$P(\text{spam}) = Bernoulli(\phi)$$
$$P(\text{word}|\text{spam}) = Multinomial(\mu_1^s, \ldots, \mu_M^s)$$
$$P(\text{word}|\text{nonspam}) = Multinomial(\mu_1^{ns}, \ldots, \mu_M^{ns})$$

- Goal

: Find ϕ, μ^s, μ^{ns} that best fits the data $\{(x^{(1)}, y^{(1)}),\ldots, (x^{(N)}, y^{(N)})\}$

Naive Bayes Spam classifier

- Model

$$P(\text{spam}) = Bernoulli(\phi)$$
$$P(\text{word}|\text{spam}) = Multinomial(\mu_1^s, \ldots, \mu_M^s)$$
$$P(\text{word}|\text{nonspam}) = Multinomial(\mu_1^{ns}, \ldots, \mu_M^{ns})$$

- Goal

: Find ϕ, μ^s, μ^{ns} that best fits the data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$

- Likelihood

$$\prod_{i=1}^{N} P(\mathbf{x}^{(i)}, y^{(i)})$$

Naive Bayes Spam classifier

- Model

$$P(\text{spam}) = Bernoulli(\phi)$$
$$P(\text{word}|\text{spam}) = Multinomial(\mu_1^s, \ldots, \mu_M^s)$$
$$P(\text{word}|\text{nonspam}) = Multinomial(\mu_1^{ns}, \ldots, \mu_M^{ns})$$

- Goal

: Find ϕ, μ^s, μ^{ns} that best fits the data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$

- Likelihood

$$\prod_{i=1}^{N} P(\mathbf{x}^{(i)}, y^{(i)})$$
$$= \prod_{i=1}^{N} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)})$$
$$= \underbrace{\boxed{\left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)})\right)}}_{\text{Spam}} \underbrace{\left(\prod_{i:y^{(i)}=0} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)})\right)}_{\text{Non-spam}}$$

Naive Bayes Spam classifier

- Likelihood - spam

$$\left(\prod_{i:y^{(i)}=1} \underline{P(\mathbf{x}^{(i)}|y^{(i)})}\,\underline{P(y^{(i)})}\right)$$

$$x_k^{(i)}$$

i-th mail

k-th word

- Naive Bayes assumption:

$$P(\text{spam}) = Bernoulli(\phi)$$
$$P(\text{word}|\text{spam}) = Multinomial(\mu_1^s, \ldots, \mu_M^s)$$

$$\underline{P(x^{(i)}|y^{(i)}=1)} = \prod_{k=1}^{len(x^{(i)})} P(x_k^{(i)}|y^{(i)}=1)$$
$$= \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} (\mu_j^s)^{I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)}$$
$$\underline{P(y^{(i)}=1)} = \phi$$

Naive Bayes Spam classifier

- Likelihood - spam (cont')

$$\left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)})\right)$$

$$= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} (\mu_j^s)^{I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)} \phi\right)$$

Naive Bayes Spam classifier

- Likelihood - spam (cont')

$$\left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)})\right)$$

$$= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} (\mu_j^s)^{I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)} \phi\right)$$

$$= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} (\mu_j^s)^{I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)}\right) \left(\prod_{i:y^{(i)}=1}^{N} \phi\right)$$

Naive Bayes Spam classifier

- Likelihood - spam (cont')

$$\left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)}) \right)$$

$$= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} (\mu_j^s)^{I\left(x_k^{(i)}=\text{"}j\text{" th word}\right)} \phi \right)$$

$$= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} (\mu_j^s)^{I\left(x_k^{(i)}=\text{"}j\text{" th word}\right)} \right) \left(\prod_{i:y^{(i)}=1}^{N} \phi \right)$$

$$= \left(\prod_{j=1}^{M} (\mu_j^s)^{\sum_{i:y^{(i)}=1}^{N} \sum_{k=1}^{len(x^{(i)})} I\left(x_k^{(i)}=\text{"}j\text{" th word}\right)} \right) \left(\prod_{i:y^{(i)}=1}^{N} \phi \right)$$

Naive Bayes Spam classifier

- Likelihood - spam (cont')

$$
\begin{aligned}
&\left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)})\right) \\
&= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} \left(\mu_j^s\right)^{I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)} \phi\right) \\
&= \left(\prod_{i:y^{(i)}=1}^{N} \prod_{k=1}^{len(x^{(i)})} \prod_{j=1}^{M} \left(\mu_j^s\right)^{I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)}\right) \left(\prod_{i:y^{(i)}=1}^{N} \phi\right) \\
&= \left(\prod_{j=1}^{M} \left(\mu_j^s\right)^{\sum_{i:y^{(i)}=1}^{N} \sum_{k=1}^{len(x^{(i)})} I\left(x_k^{(i)}=\text{"}j\text{"th word}\right)}\right) \left(\prod_{i:y^{(i)}=1}^{N} \phi\right) \\
&= \left(\prod_{j=1}^{M} \left(\mu_j^s\right)^{N_j^{spam}}\right) \phi^{N^{spam}}
\end{aligned}
$$

Maximum likelihood estimation

- Putting together:

$$\prod_{i=1}^{N} P(\mathbf{x}^{(i)}, y^{(i)})$$

$$= \left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)}) P(y^{(i)}) \right) \left(\prod_{i:y^{(i)}=0} P(\mathbf{x}^{(i)}|y^{(i)}) P(y^{(i)}) \right)$$

Maximum likelihood estimation

- Putting together:

$$\prod_{i=1}^{N} P(\mathbf{x}^{(i)}, y^{(i)})$$

$$= \left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)}) P(y^{(i)}) \right) \left(\prod_{i:y^{(i)}=0} P(\mathbf{x}^{(i)}|y^{(i)}) P(y^{(i)}) \right)$$

$$= \left(\phi^{N^{spam}} \prod_{word\, j} \left(\mu_j^s\right)^{N_j^{spam}} \right) \left((1-\phi)^{N^{nonspam}} \prod_{word\, j} \left(\mu_j^{ns}\right)^{N_j^{nonspam}} \right)$$

Maximum likelihood estimation

- Putting together:

$$
\begin{aligned}
&\prod_{i=1}^{N} P(\mathbf{x}^{(i)}, y^{(i)}) \\
= \quad &\left(\prod_{i:y^{(i)}=1} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)}) \right) \left(\prod_{i:y^{(i)}=0} P(\mathbf{x}^{(i)}|y^{(i)})P(y^{(i)}) \right) \\
= \quad &\left(\phi^{N^{spam}} \prod_{word\, j} \left(\mu_j^s\right)^{N_j^{spam}} \right) \left((1-\phi)^{N^{nonspam}} \prod_{word\, j} \left(\mu_j^{ns}\right)^{N_j^{nonspam}} \right)
\end{aligned}
$$

- Log-likelihood

$$
\begin{aligned}
&\log P(\mathcal{D}) \\
= \quad &\log \prod_{i=1}^{N} P(x^{(i)}, y^{(i)}) \\
= \quad &N^{spam} \log \phi + \sum_{word\, j} N_j^{spam} \log \mu_j^s + N^{nonspam} \log(1-\phi) + \sum_{word\, j} N_j^{nonspam} \log \mu_j^{ns}
\end{aligned}
$$

Maximum likelihood estimation

- Log-likelihood

$$
\begin{aligned}
& \log P(\mathcal{D}) \\
= \quad & \log \prod_{i=1}^{N} P(x^{(i)}, y^{(i)}) \\
= \quad & N^{spam} \log \phi + \sum_{word\, j} N_j^{spam} \log \mu_j^s + N^{nonspam} \log(1-\phi) + \sum_{word\, j} N_j^{nonspam} \log \mu_j^{ns}
\end{aligned}
$$

- Maximum-likelihood
  - Take the derivative of log-likelihood w.r.t. the parameters, and set it to zero.

Maximum likelihood estimation

- From $$\frac{\partial l}{\partial \phi} = \frac{1}{\phi} N^{spam} - \frac{1}{1-\phi} N^{nonspam} = 0$$
  - We get $$\phi = \frac{N^{spam}}{N^{spam} + N^{nonspam}}$$

- Make the parameters μ independent:

$$\sum_{word\, j=1}^{M} N_j^{spam} \log \mu_j^s = \sum_{word\, j=1}^{M-1} N_j^{spam} \log \mu_j^s + N_M^{spam} \log(1 - \sum_{j=1}^{M-1} \mu_j^s)$$

$$\frac{\partial}{\partial \mu_j^s} \left(\sum_{word\, j=1}^{M} N_j^{spam} \log \mu_j^s \right) = \frac{N_j^{spam}}{\mu_j^s} - \frac{N_M^{spam}}{1 - \sum_{j=1}^{M-1} \mu_j^s} = 0$$

Maximum likelihood estimation

- From $$\frac{\partial l}{\partial \phi} = \frac{1}{\phi} N^{spam} - \frac{1}{1-\phi} N^{nonspam} = 0$$
  – We get $$\phi = \frac{N^{spam}}{N^{spam} + N^{nonspam}}$$

- Make the parameters μ independent:

$$\sum_{word\, j=1}^{M} N_j^{spam} \log \mu_j^s = \sum_{word\, j=1}^{M-1} N_j^{spam} \log \mu_j^s + N_M^{spam} \log(1 - \sum_{j=1}^{M-1} \mu_j^s)$$

$$\frac{\partial}{\partial \mu_j^s} \left(\sum_{word\, j=1}^{M} N_j^{spam} \log \mu_j^s \right) = \frac{N_j^{spam}}{\mu_j^s} - \frac{N_M^{spam}}{1 - \sum_{j=1}^{M-1} \mu_j^s} = 0$$

$$\frac{N_j^{spam}}{\mu_j^s} = constant, \forall j$$

We finally get $$\mu_j^s = \frac{N_j^{spam}}{\sum_j N_j^{spam}}$$

Maximum likelihood estimation

- Summary:

$$P(spam) = \phi = \frac{N^{spam}}{N^{spam} + N^{nonspam}}$$

$$P(word = j|spam) = \mu_j^s = \frac{N_j^{spam}}{\sum_j N_j^{spam}}$$

$$P(word = j|non-spam) = \mu_j^{ns} = \frac{N_j^{nonspam}}{\sum_j N_j^{nonspam}}$$

Recall:
N^{spam}: total # examples for spam
$N^{nonspam}$: total # examples for non-spam

N_j^{spam}: total # word j from the entire spam emails
$N_j^{nonspam}$: total # word j from the entire nonspam emails

Laplace smoothing

- Maximum likelihood is problematic when a specific word count is 0
  - Leads to probability of 0!
- Solution: Put “imaginary” counts for each word
  - prevent zero probability estimates (overfitting)!
  - E.g.: Adding “1” as imaginary count for each word

$$P(spam) = \phi = \frac{N^{spam}}{N^{spam} + N^{nonspam}}$$

$$P(word = j|spam) = \mu_j^s = \frac{N_j^{spam} \boxed{+1}}{\sum_j N_j^{spam} \boxed{+M}}$$

$$P(word = j|non - spam) = \mu_j^{ns} = \frac{N_j^{nonspam} \boxed{+1}}{\sum_j N_j^{nonspam} \boxed{+M}}$$

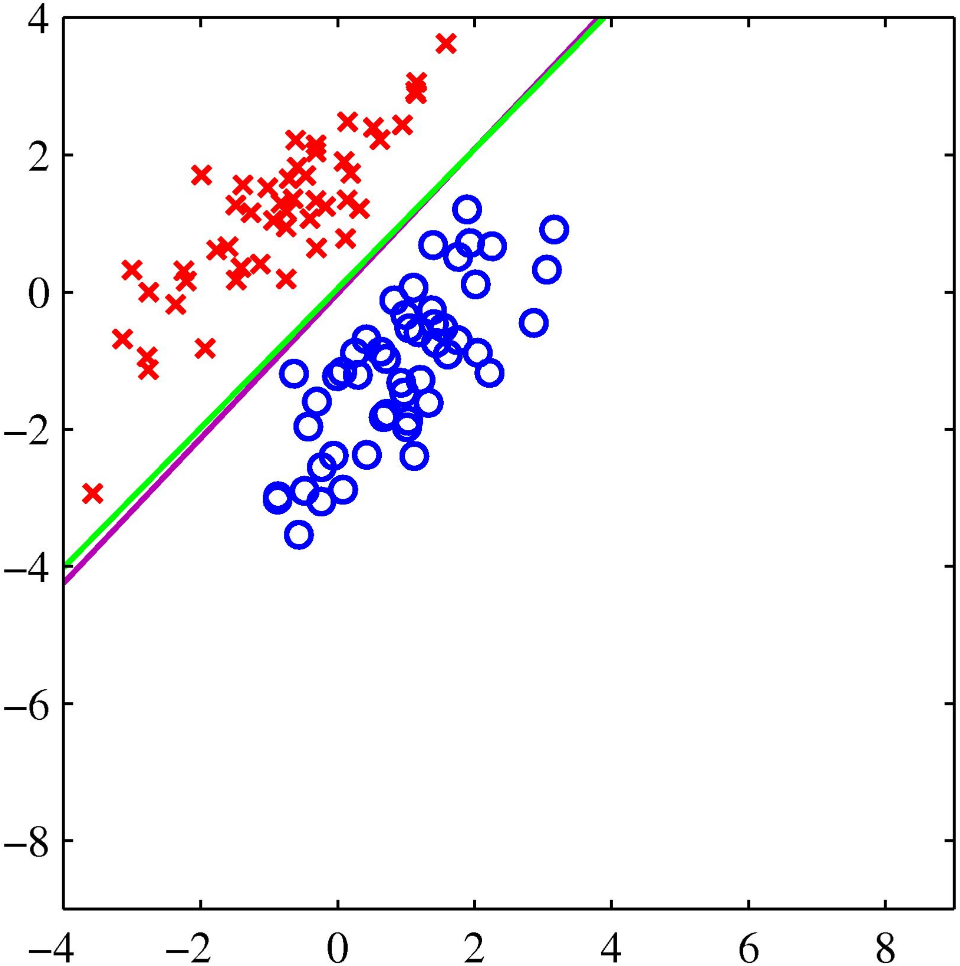
Discriminant functions

Linear Discriminant functions: Discriminating two classes

- Specify a weight vector **w** and a bias w0

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

- Assign **x** to C_1 if $h(x) \geq 0$ and to C_0 otherwise.
- Q: How to pick **w**?



Linear Discriminant functions: Discriminating K>2 classes

- Instead each class C_k gets its own function

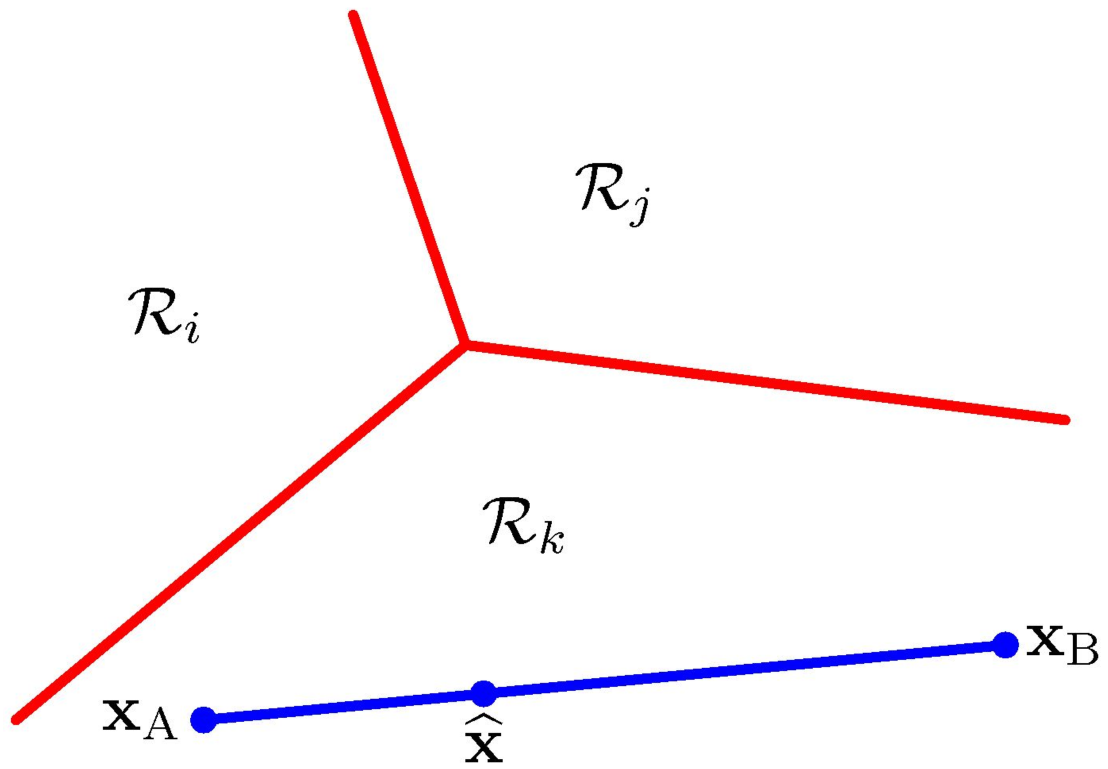
$$h_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$

  – Assign **x** to C_k if

$$h_k(\mathbf{x}) > h_j(\mathbf{x}) \text{ for all } j \neq k$$

- The decision regions are convex polyhedra.

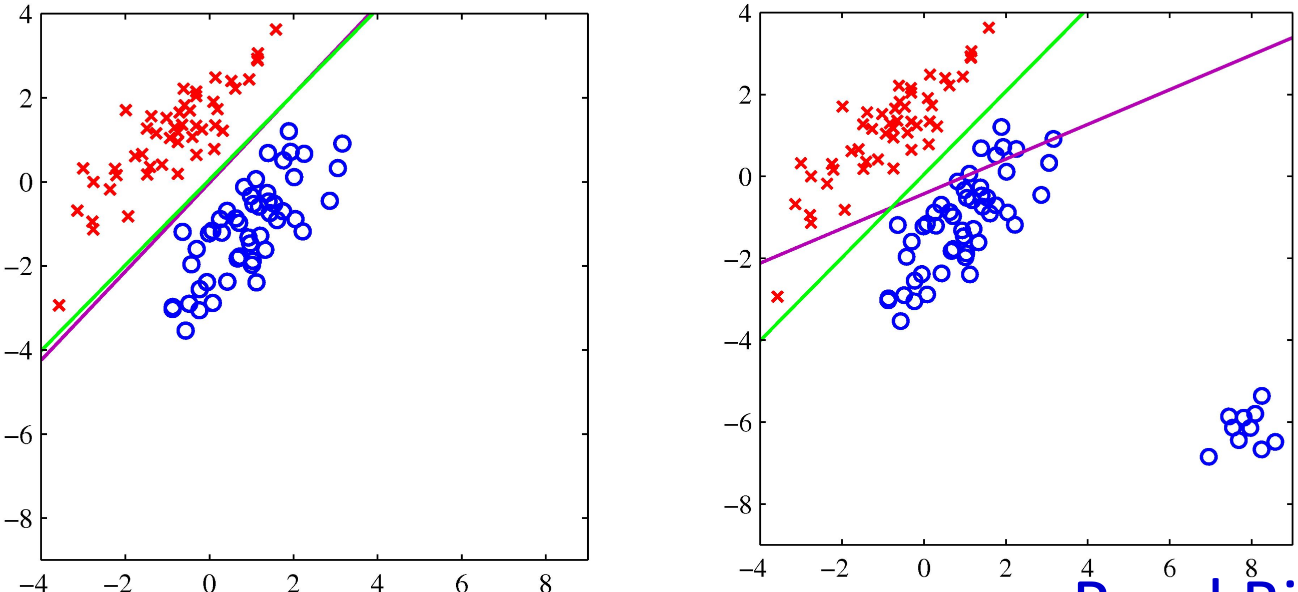
Decision Regions



- Decision regions are convex, with piecewise linear boundaries.

How do we set the weights w?

- How about w that minimizes squared error?
  - Label **y** versus linear prediction h(w).
  - Least squares is too sensitive to outliers. (why?)



Read Bishop book

Learning linear discriminant functions (this lecture)

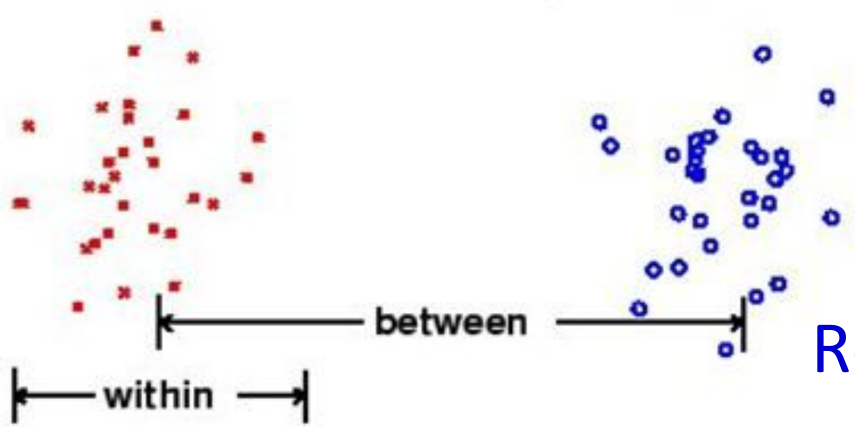
- Fisher’s linear discriminant
- Perceptron learning algorithm

Fisher's Linear Discriminant

- Use w to project x to one dimension.

$$\text{if } \mathbf{w}^T\mathbf{x} \geq -w_0 \text{ then } C_1 \text{ else } C_0$$

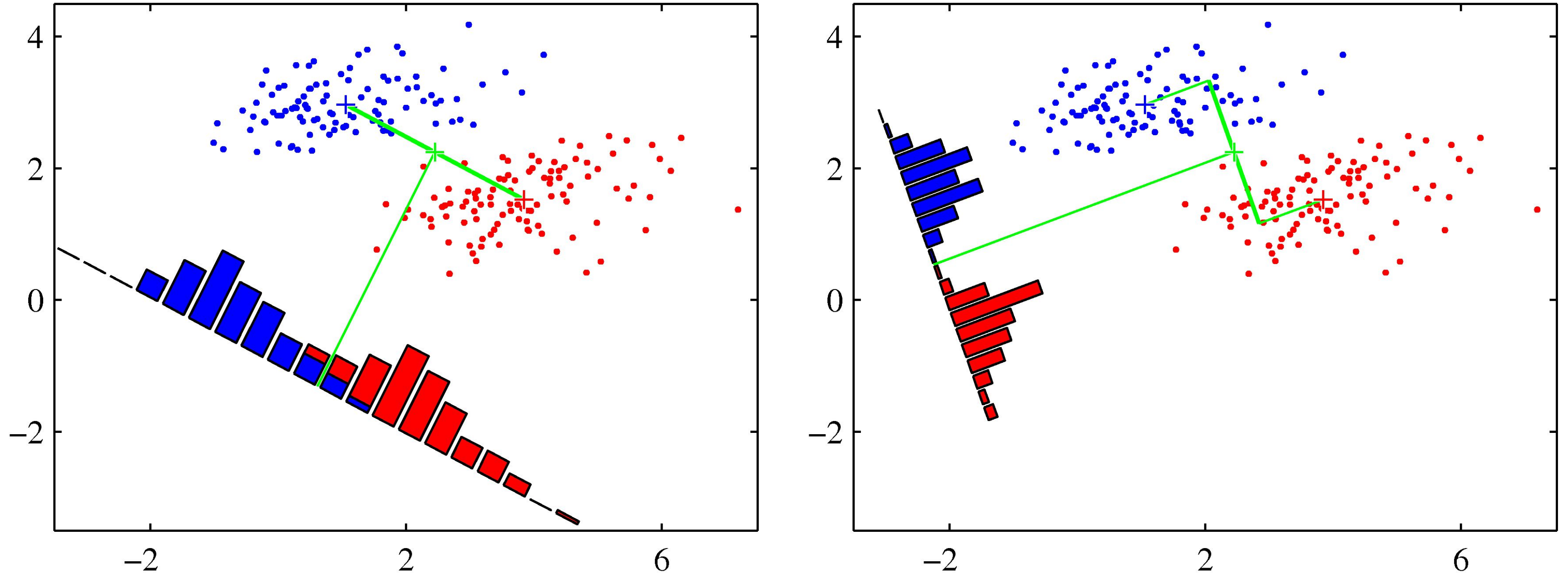
- Select w that best separates the classes.
- By "separating", the algorithm simultaneously
  - maximizes between-class variances
  - minimizes within-class variances



Read Bishop book

Fisher's Linear Discriminant

- Maximizing separation alone doesn't work.
  - Minimizing class variance is a big help.



See Bishop book

Objective function

- We want to maximize the "distance between classes"

$$\underline{m_2} - m_1 \equiv \mathbf{w}^T(\underline{\mathbf{m}_2} - \mathbf{m}_1) \qquad \text{where } \mathbf{m}_k = \frac{1}{N_k}\sum_{n \in C_k} \mathbf{x}_n$$

Projected mean

Mean

See Bishop book

Objective function

- We want to maximize the "distance between classes"

$$\underline{m_2} - m_1 \equiv \mathbf{w}^T(\underline{\mathbf{m}_2} - \mathbf{m}_1) \qquad \text{where } \mathbf{m}_k = \frac{1}{N_k}\sum_{n \in C_k} \mathbf{x}_n$$

Projected mean Mean

- While minimizing the "distance within each class"

$$s_1^2 + s_2^2 \equiv \sum_{n \in C_1} (\mathbf{w}^T \mathbf{x}_n - m_1)^2 + \sum_{n \in C_2} (\mathbf{w}^T \mathbf{x}_n - m_2)^2$$

See Bishop book

Objective function

- We want to maximize the "distance between classes"

$$\underline{m_2} - m_1 \equiv \mathbf{w}^T(\underline{\mathbf{m}_2} - \mathbf{m}_1) \qquad \text{where } \mathbf{m}_k = \frac{1}{N_k}\sum_{n \in C_k} \mathbf{x}_n$$

Projected mean Mean

- While minimizing the "distance within each class"

$$s_1^2 + s_2^2 \equiv \sum_{n \in C_1} (\mathbf{w}^T\mathbf{x}_n - m_1)^2 + \sum_{n \in C_2} (\mathbf{w}^T\mathbf{x}_n - m_2)^2$$

- Objective function: $$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

See Bishop book

Derivation of objective

- Numerator: $m_2 - m_1 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)$:
  - $\|m_2 - m_1\|^2 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T\mathbf{w}$

See Bishop book

Derivation of objective

- Numerator: $m_2 - m_1 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)$:
  - $\|m_2 - m_1\|^2 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T\mathbf{w}$

- Denominator:
  - $s_k^2 = \sum_{n\in C_k}(\mathbf{w}^T\mathbf{x}_n - m_k)^2$
    $= \sum_{n\in C_k}\mathbf{w}^T(\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T\mathbf{w}$
  - $s_1^2 + s_2^2 = \mathbf{w}^T\left[\sum_{k=1,2}\sum_{n\in C_k}(\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T\right]\mathbf{w}$

See Bishop book

Derivation of objective

- Numerator: $m_2 - m_1 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)$:
  - $\|m_2 - m_1\|^2 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T\mathbf{w}$

- Denominator:
  - $s_k^2 = \sum_{n \in C_k}(\mathbf{w}^T\mathbf{x}_n - m_k)^2$
    $= \sum_{n \in C_k} \mathbf{w}^T(\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T\mathbf{w}$
  - $s_1^2 + s_2^2 = \mathbf{w}^T\left[\sum_{k=1,2}\sum_{n \in C_k}(\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T\right]\mathbf{w}$

- After definition of terms, we get

$$J(w) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$$

  - Solution: $\mathbf{w} \propto S_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$

See Bishop book

The Perceptron

- A "generalized linear function"

$$h(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

- Where

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

- Uses target code: y=+1 for C_1, y=-1 for C_2.
- Means we always want:

$$\mathbf{w}^T \phi(\mathbf{x}_n) y_n > 0$$

The Perceptron Criterion

- Only count errors from misclassified points:

$$E_P(\mathbf{w}) = - \sum_{\mathbf{x}_n \in \mathcal{M}} \mathbf{w}^T \phi(\mathbf{x}) y_n$$

  – where *M* are the misclassified points.

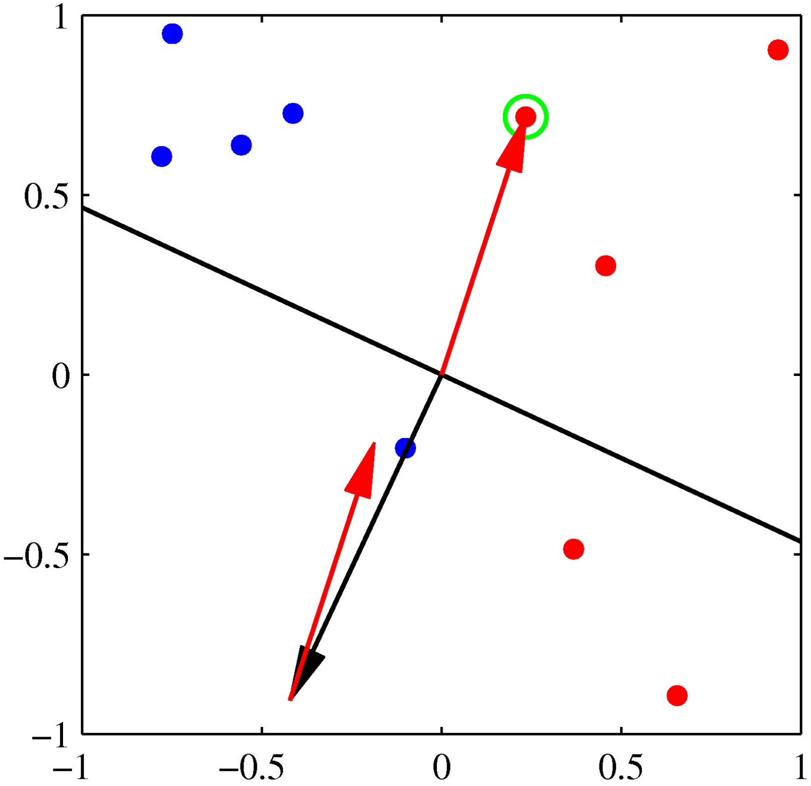
- Stochastic gradient descent:
  – Update the weight vector according to the misclassified points:

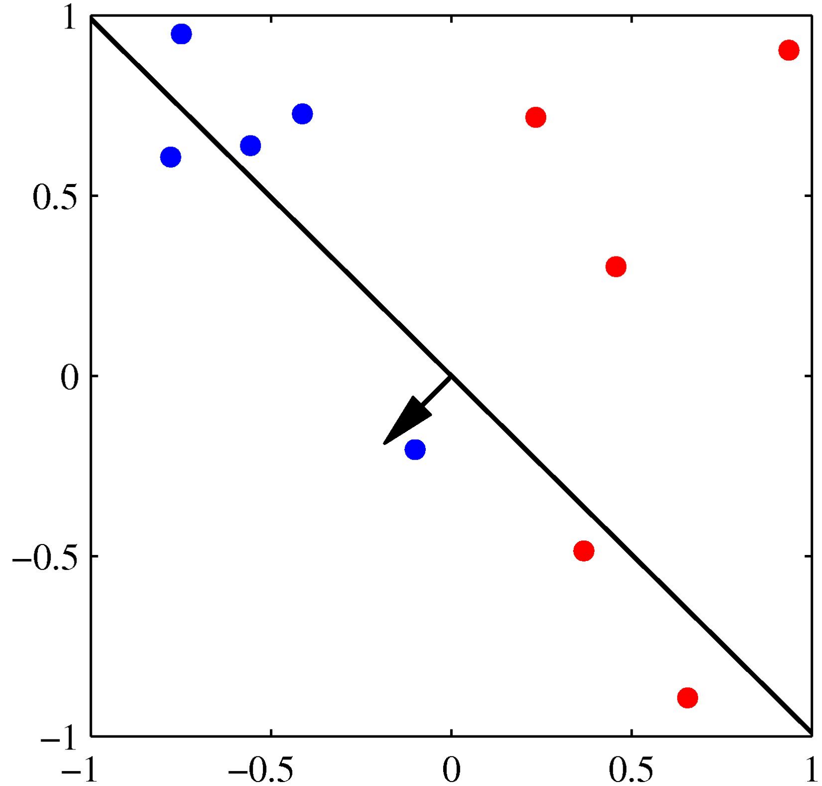
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \phi(\mathbf{x}) y_n$$

Note: update only for misclassified examples

Perceptron Learning (1)

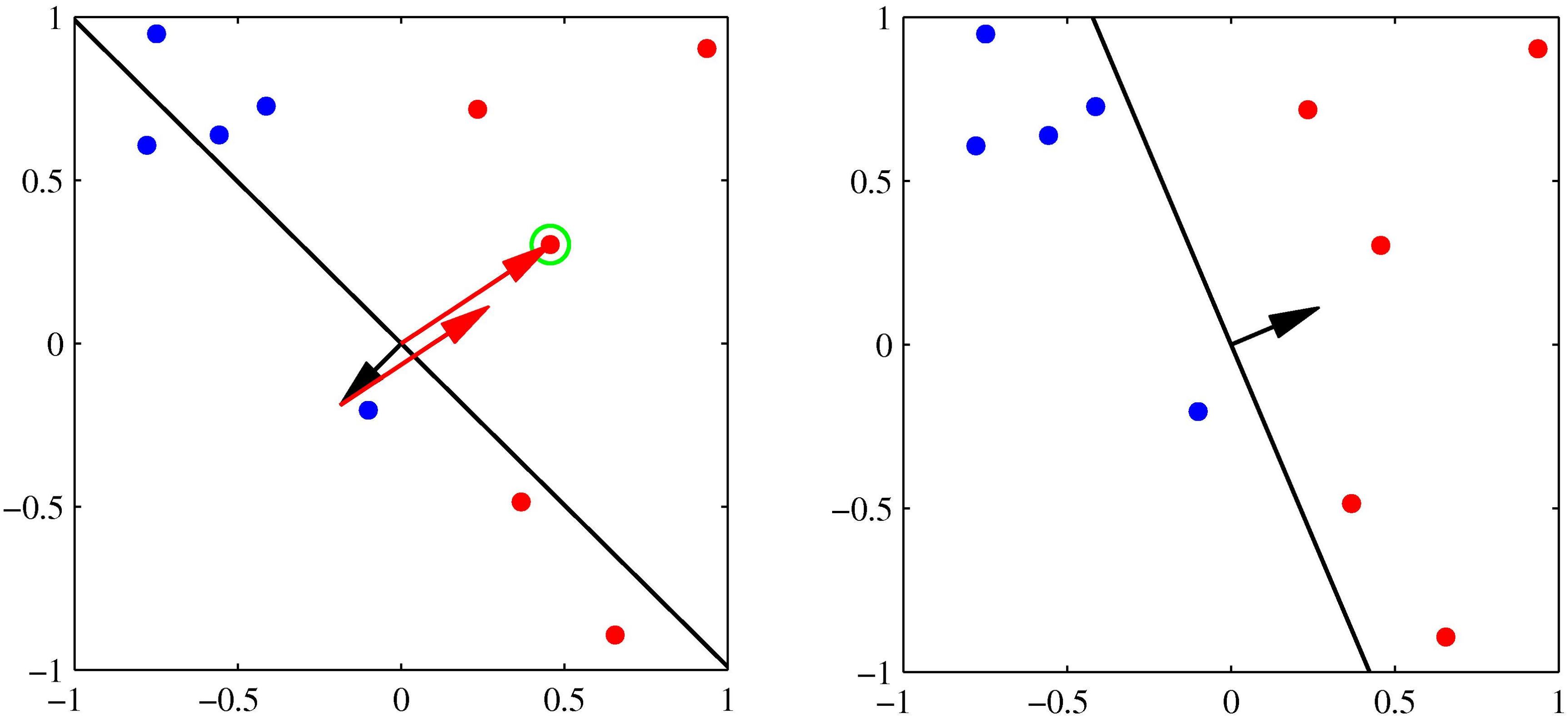
- If x_n is misclassified, add $\phi(\mathbf{x}_n)$ into w.





Perceptron Learning (2)

- If x_n is misclassified, add $\phi(\mathbf{x}_n)$ into w.



Perceptron Learning

- Perceptron Convergence Theorem:
  - If there exists an exact solution (i.e., if the training data is linearly separable)
  - then the learning algorithm will find it in a finite number of steps.

- Limitations of perceptron learning:
  - The convergence can be very slow.
  - If dataset is not linearly separable, it won't converge.
  - Does not generalize well to K>2 classes.

Next class

- Kernel methods