

EECS 545: Machine Learning

Lecture 6. Classification 3

Honglak Lee

1/29/2025



Logistics/Announcements

- HW 2 due on Feb 11th
- Project Proposal due
 - No point deduction for HW1 through Jan 31th, Friday (3 late days with *consuming tokens* afterward)
 - 1 point per day from onwards

Outline

(grey: already covered)

- Probabilistic discriminative models
 - ✓ Logistic Regression
 - ✓ Softmax Regression
- Probabilistic generative models
 - ✓ Gaussian discriminant analysis
 - Naive Bayes (continued)
- Discriminant functions (non-probabilistic)
 - Fisher's linear discriminant
 - Perceptron learning algorithm

Recap: Discriminative vs Generative Probabilistic Classifiers

- For classification, we want to compute $p(C_k | \mathbf{x})$
 - (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 joint densities $p(\mathbf{x}, C_k)$ by learning $p(\mathbf{x} | C_k)$ and $p(C_k)$, and then use Bayes rule for predicting the class C_k given \mathbf{x} :
 - Gaussian Discriminant Analysis
 - Naive Bayes

Naive Bayes classifier (continued)

Naive Bayes classifier

- Probability of class label:
 - $p(C_k)$: Constant (e.g., Bernoulli)
- Conditional probability of data given the class
 - Naive Bayes assumption: $p(\mathbf{x} | C_k)$ is factorized
(Each coordinate of \mathbf{x} is conditionally independent of other coordinates given the class label)

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

- Classification: use Bayes rule

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

Naive Bayes classifier

- When classifying, we can simply find the class C_k that maximizes $P(C_k|\mathbf{x})$ using the Bayes rule:

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

Naive Bayes classifier


- When classifying, we can simply find the class C_k that maximizes $P(C_k|\mathbf{x})$ using the Bayes rule:

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

Naive Bayes classifier

- When classifying, we can simply find the class C_k that maximizes $P(C_k|\mathbf{x})$ using the Bayes rule:

$$\begin{aligned}\arg \max_k P(C_k|\mathbf{x}) &= \arg \max_k P(C_k, \mathbf{x}) \\ &= \arg \max_k P(C_k)P(\mathbf{x}|C_k) \\ &= \arg \max_k P(C_k) \prod_{j=1}^M P(x_j|C_k)\end{aligned}$$

Naive Bayes assumption 

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

$$P(\text{spam}) = \text{Bernoulli}(\phi)$$

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

$$P(\text{word}|\text{nospam}) = \text{Multinomial}(\mu_1^{ns}, \dots, \mu_M^{ns})$$

Showing top 17 of 885 possible words

ai (10) CS (12) dear (17) funding (12) icml (12) mail (13)
manuscript (33) neurips (14)
proposals (12) requests (16) research (10) reviewers (18)
teaching (12) version (14) view (10) visiting (19) week (15)

top words from my **non-spam** emails

Showing top 12 of 1077 possible words

choice (9) congratulations (8) deals (12)
exclusive (7) gift (20) giveaway (6) limited (9)
plan (5) sale (6) select (8) special (13) top (5)

top words from my **spam** emails

Naive Bayes Spam classifier

- Model

$$P(\text{spam}) = \text{Bernoulli}(\phi)$$

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

$$P(\text{word}|\text{nospam}) = \text{Multinomial}(\mu_1^{ns}, \dots, \mu_M^{ns})$$

- Goal

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

Naive Bayes Spam classifier

- Model

$$P(\text{spam}) = \text{Bernoulli}(\phi)$$

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

$$P(\text{word}|\text{nospam}) = \text{Multinomial}(\mu_1^{ns}, \dots, \mu_M^{ns})$$

- Goal

Find ϕ , μ^s , μ^{ns} that best fits the data $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$ by maximizing the joint likelihood:

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

- Joint Likelihood (joint probability of inputs/labels)
 - Note that the joint likelihood is conditioned on parameters ϕ , μ^s , μ^{ns}

Naive Bayes Spam classifier

- Model

$$P(\text{spam}) = \text{Bernoulli}(\phi)$$

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

$$P(\text{word}|\text{nospam}) = \text{Multinomial}(\mu_1^{ns}, \dots, \mu_M^{ns})$$

- Goal

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

- Likelihood - conditioned on parameters ϕ, μ^s, μ^{ns}

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

Naive Bayes Spam classifier

- Likelihood - spam

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

$x_k^{(i)}$ \nearrow i-th mail
 \searrow k-th word

- Naive Bayes assumption:

$$P(\text{spam}) = \text{Bernoulli}(\phi)$$

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

$$\underline{P(\mathbf{x}^{(i)}|y^{(i)} = 1)} = \prod_{k=1}^{\text{len}(\mathbf{x}^{(i)})} P(x_k^{(i)}|y^{(i)} = 1)$$

$$= \prod_{k=1}^{\text{len}(\mathbf{x}^{(i)})} \prod_{j=1}^M (\mu_j^s)^{\mathbb{I}(x_k^{(i)} = \text{"j" th word})}$$

$$\underline{P(y^{(i)} = 1)} = \phi$$

Naive Bayes Spam classifier

- Likelihood - spam (cont')

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

Naive Bayes Spam classifier

- Likelihood - spam (cont')

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

Naive Bayes Spam classifier

- Likelihood - spam (cont')

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

Naive Bayes Spam classifier

- Likelihood - spam (cont')

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

Definition:

N^{spam} : total # examples for spam

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

Naive Bayes Spam classifier

- Likelihood - non-spam

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

- Similarly for non-spam mails,

$$\begin{aligned}
 P(\mathbf{x}^{(i)}|y^{(i)} = 0) &= \prod_{k=1}^{\text{len}(\mathbf{x}^{(i)})} P(x_k^{(i)}|y^{(i)} = 0) \\
 &= \prod_{k=1}^{\text{len}(\mathbf{x}^{(i)})} \prod_{j=1}^M (\mu_j^{ns})^{I(x_k^{(i)} = \text{"j" th word})} \\
 P(y^{(i)} = 0) &= 1 - \phi
 \end{aligned}$$

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

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:

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

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

Maximum likelihood estimation

- Putting together:

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

- Joint Log-likelihood

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

Maximum likelihood estimation

- Joint Log-likelihood

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

- Maximum-likelihood

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

Maximum likelihood estimation

- From $\frac{\partial l}{\partial \phi} = \frac{1}{\phi} N^{spam} - \frac{1}{1 - \phi} N^{nonsпам} = 0$

We get $\phi = \frac{N^{spam}}{N^{spam} + N^{nonsпам}}$

- Removing dependent variables:

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

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

$$\text{s.t.} \quad \sum_j \mu_j^s = 1$$

$$\sum_j \mu_j^{ns} = 1$$

Maximum likelihood estimation

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

We get $\phi = \frac{N^{spam}}{N^{spam} + N^{nonspam}}$

- Removing dependent variables:

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

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

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

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

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

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

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

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

Outline

(grey: already covered)

- Probabilistic discriminative models
 - ✓ Logistic Regression
 - ✓ Softmax Regression
- Probabilistic generative models
 - ✓ Gaussian discriminant analysis
 - ✓ Naive Bayes
- Discriminant functions (non-probabilistic)
 - Fisher's linear discriminant
 - Perceptron learning algorithm

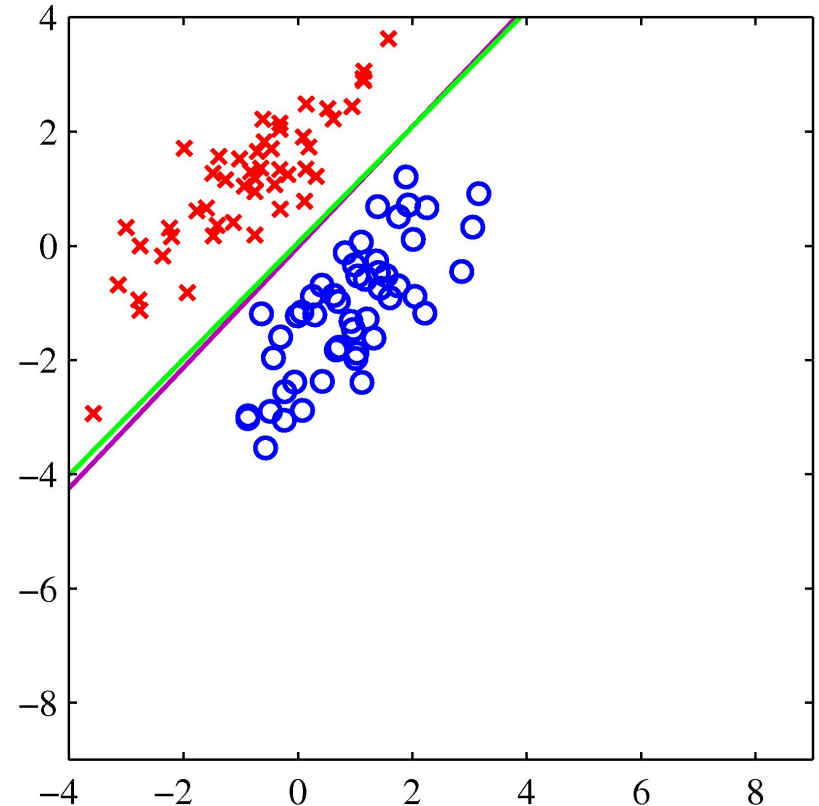
Discriminant Functions

Linear Discriminant functions: Discriminating two classes

- Specify a weight vector \mathbf{w} and a bias w_0

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

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



Linear Discriminant functions: Discriminating $K > 2$ classes

- Instead each class C_k gets its own function

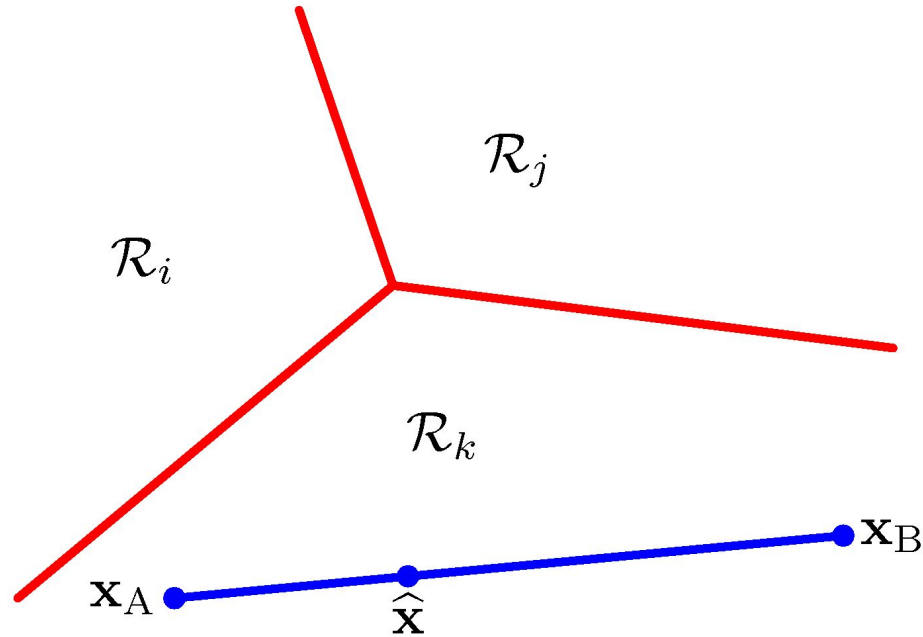
$$h_k(\mathbf{x}) = \mathbf{w}_k^\top \mathbf{x} + w_{k,0}$$

- Assign \mathbf{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.

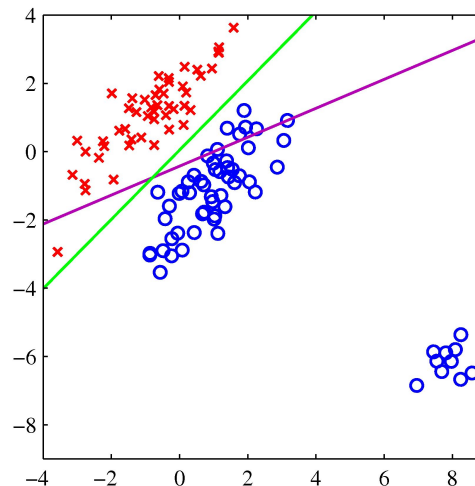
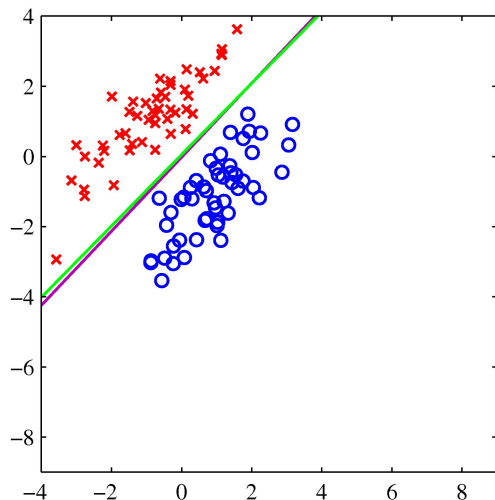
Decision Regions



- 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?)



Learning Linear Discriminant Functions

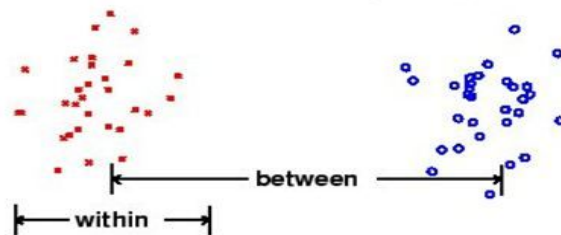
- Fisher's linear discriminant
- Perceptron learning algorithm

Fisher's Linear Discriminant

- Let's consider binary classification case.
- Use \mathbf{w} to project \mathbf{x} to one dimension.

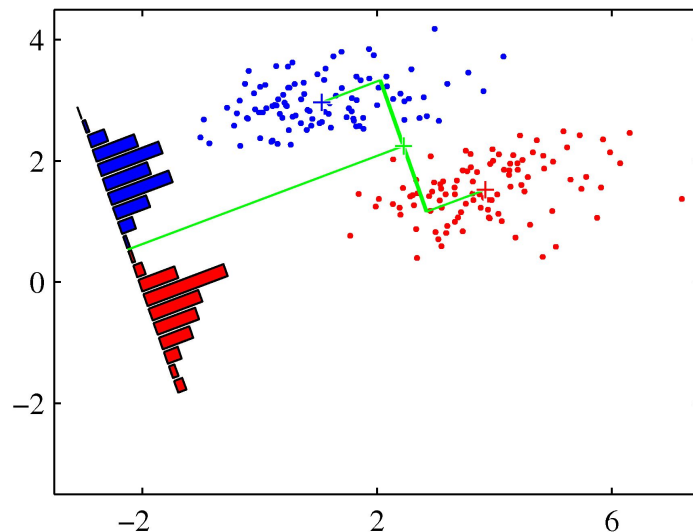
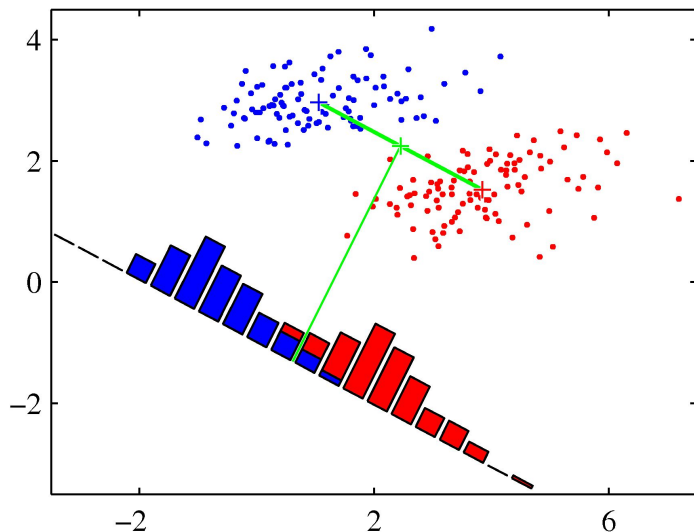
$$\text{if } \mathbf{w}^\top \mathbf{x} \geq -w_0 \text{ then } C_1 \text{ else } C_0$$

- Select \mathbf{w} that best separates the classes.
- By “separating”, the algorithm simultaneously
 - maximizes between-class (inter-class) variances
 - minimizes within-class (intra-class) variances



Fisher's Linear Discriminant

- Maximizing separation alone is not enough.
 - Minimizing class variance is a big help.



Objective function

- We want to maximize the “distance between classes”

$$\underline{m_2} - m_1 \equiv \mathbf{w}^\top (\underline{\mathbf{m}_2} - \mathbf{m}_1)$$

Projected mean

Mean

$$\text{where } \mathbf{m}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{x}_n$$

Objective function

- We want to maximize the “distance between classes”

$$\underbrace{m_2 - m_1}_{\text{Projected mean}} \equiv \mathbf{w}^\top (\underbrace{\mathbf{m}_2 - \mathbf{m}_1}_{\text{Mean}}) \quad \text{where } \mathbf{m}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{x}_n$$

- While minimizing the “distance within each class”

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

Objective function

- We want to maximize the “distance between classes”

$$\underbrace{m_2 - m_1}_{\text{Projected mean}} \equiv \mathbf{w}^\top (\underbrace{\mathbf{m}_2 - \mathbf{m}_1}_{\text{Mean}}) \quad \text{where } \mathbf{m}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{x}_n$$

- While minimizing the “distance within each class”

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

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

Derivation of objective

- Numerator: $m_2 - m_1 \equiv \mathbf{w}^\top (\mathbf{m}_2 - \mathbf{m}_1)$
 $\|m_2 - m_1\|^2 = \mathbf{w}^\top \underbrace{(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top}_{= S_B} \mathbf{w}$

Derivation of objective

- Numerator: $m_2 - m_1 \equiv \mathbf{w}^\top (\mathbf{m}_2 - \mathbf{m}_1)$

$$\|m_2 - m_1\|^2 = \mathbf{w}^\top \underbrace{(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top}_{= S_B} \mathbf{w}$$

- Denominator:

- $s_k^2 = \sum_{n \in C_k} (\mathbf{w}^\top \mathbf{x}_n - m_k)^2$

- $= \sum_{n \in C_k} \mathbf{w}^\top (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^\top \mathbf{w}$

- $s_1^2 + s_2^2 = \mathbf{w}^\top \left[\underbrace{\sum_{k=1,2} \sum_{n \in C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^\top}_{= S_W} \right] \mathbf{w}$

Derivation of objective

- Numerator: $m_2 - m_1 \equiv \mathbf{w}^\top (\mathbf{m}_2 - \mathbf{m}_1)$

$$\|m_2 - m_1\|^2 = \mathbf{w}^\top \underbrace{(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top}_{= S_B} \mathbf{w}$$

- Denominator:

- $s_k^2 = \sum_{n \in C_k} (\mathbf{w}^\top \mathbf{x}_n - m_k)^2$

$$= \sum_{n \in C_k} \mathbf{w}^\top (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^\top \mathbf{w}$$

- $s_1^2 + s_2^2 = \mathbf{w}^\top \left[\underbrace{\sum_{k=1,2} \sum_{n \in C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^\top}_{= S_W} \right] \mathbf{w}$

- After definition of terms, we get

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

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

Fisher's Linear Discriminant Analysis: Pros and Cons

Pros:

- Simple and effective approach for classification.
- Can effectively handle correlations between features
- Minimal assumptions about the underlying data distribution.
- Easy to interpret and explain

Cons:

- Only suitable for two-class classification problems
- Can be sensitive to outliers and may produce suboptimal results when the data has noisy features/labels

The Perceptron

- A “generalized linear function”

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

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 .
- This means that we always want:

$$\mathbf{w}^\top \phi(\mathbf{x}^{(n)}) y^{(n)} > 0$$

The Perceptron Criterion

- Only count errors from misclassified points:

$$E_P(\mathbf{w}) = - \sum_{\mathbf{x}^{(n)} \in \mathcal{M}} \mathbf{w}^\top \phi(\mathbf{x}^{(n)}) y^{(n)}$$

– where \mathcal{M} is the set of **misclassified** points.

- Stochastic gradient descent:

– Update the weight vector according to the each misclassified sample (i.e., take gradient per sample):

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

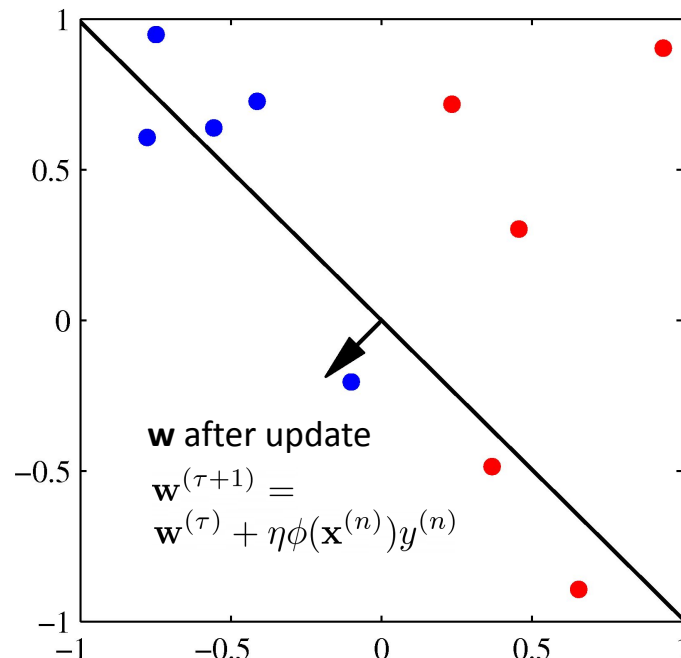
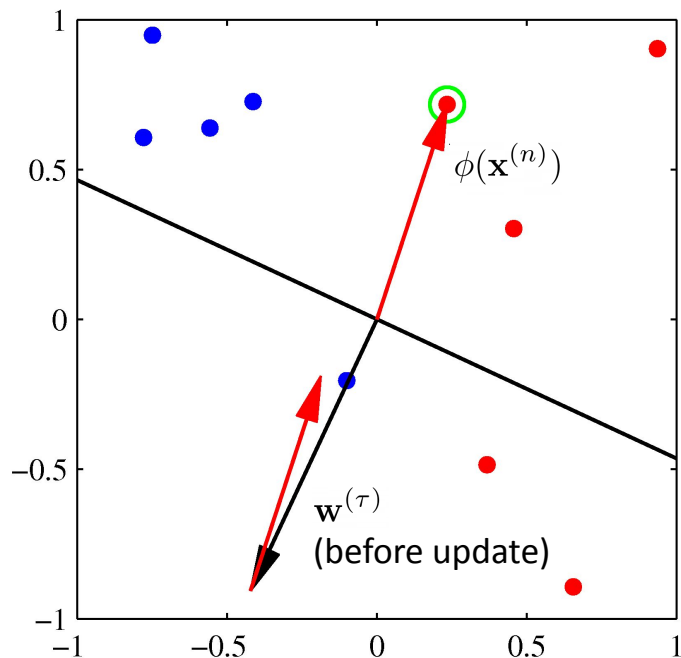
Note: update only for misclassified examples

Perceptron Learning (1)

red: positive ($y = +1$)
blue: negative ($y = -1$)

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

green circle: misclassified sample

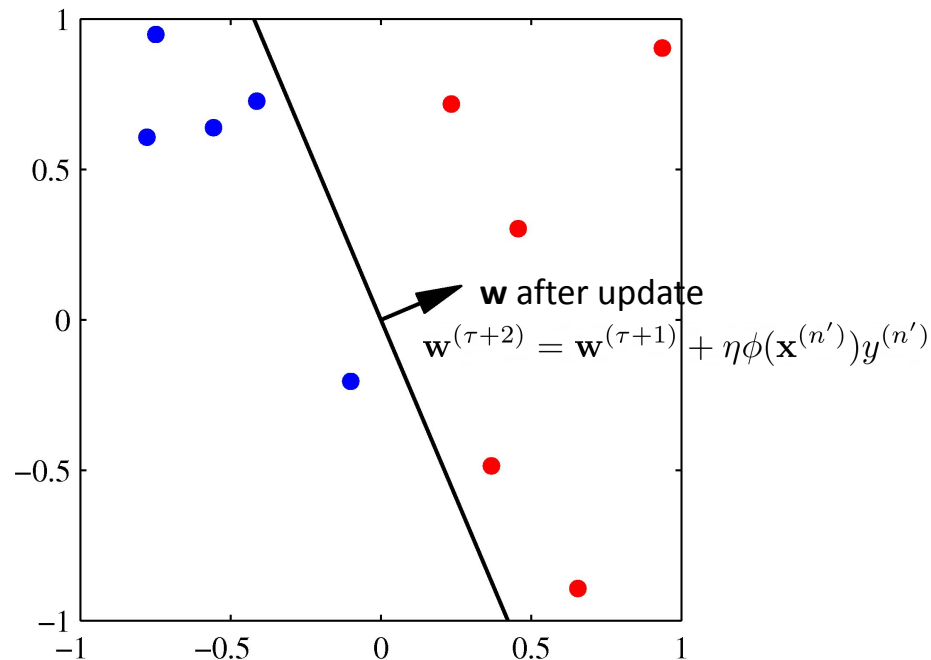
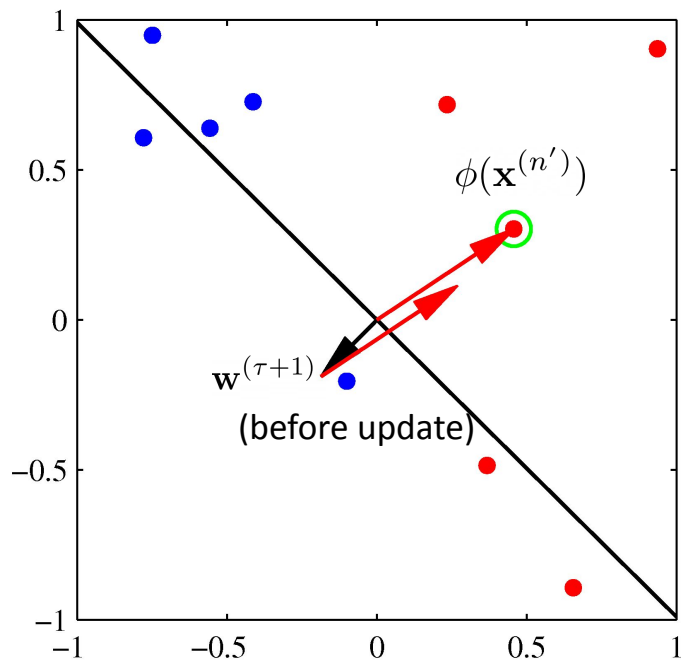


Perceptron Learning (2)

red: positive ($y = +1$)
blue: negative ($y = -1$)

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

green circle: misclassified sample



Perceptron Learning

- Perceptron Convergence Theorem (Block, 1962, and Novikoff, 1962):
 - 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
- Remember to submit your project proposals!

Any feedback (about lecture, slide, homework, project, etc.)?

(via **anonymous** google form: <https://forms.gle/fpYmiBtG9Me5qbP37>)



Change Log of lecture slides:

<https://docs.google.com/document/d/e/2PACX-1vSSIHJklypK7rKFSR1-5GYXyBCEW8UPtpSfCR9AR6M1I7K9ZQEmxfFwaWaW7kLDxusthsF8WlCyZJ-/pub>