COMP 551 - Applied Machine Learning
Lecture 19 – Midterm Review

William L. Hamilton

Primal Soft SVM problem

- Define slack variables $\xi_i = L_{hin}(w^T x_i, y_i) = \max\{1 - y_i \mathbf{w}^T \mathbf{x}_i, 0\}$

- Solve: $\hat{\mathbf{w}}_{soft} = \mathrm{argmin}_{\mathbf{w},\xi}\ C \sum_{i:1:n} \xi_i + ½||\mathbf{w}||^2$

  s. t. $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$, $i = 1, \ldots, n$

  $\xi_i \geq 0$, $i = 1, \ldots, n$

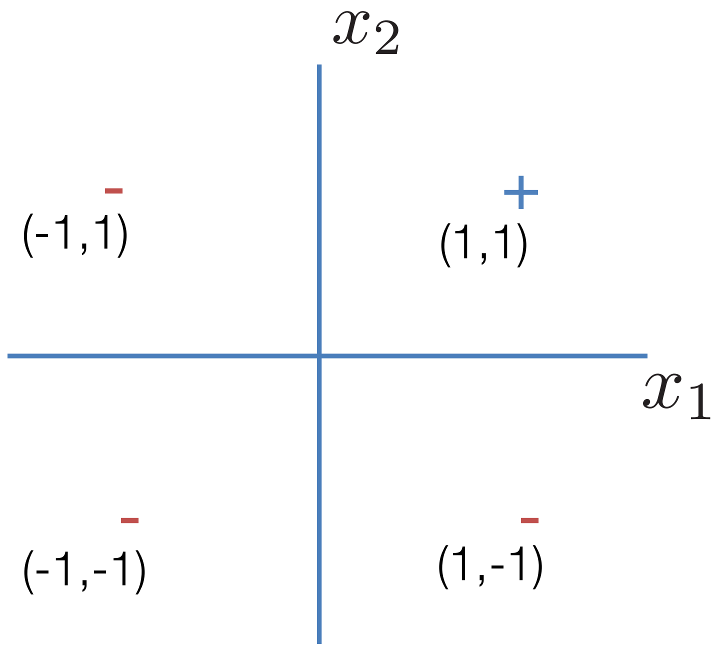
  where $w \in R^m, \xi \in R^n$

Fitting SVMs

- We are given data from a logical AND function.
- We want to fit a hard SVM:

$$\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$$

- The hard SVM decision boundary is the one the maximizes the margin.



Fitting SVMs

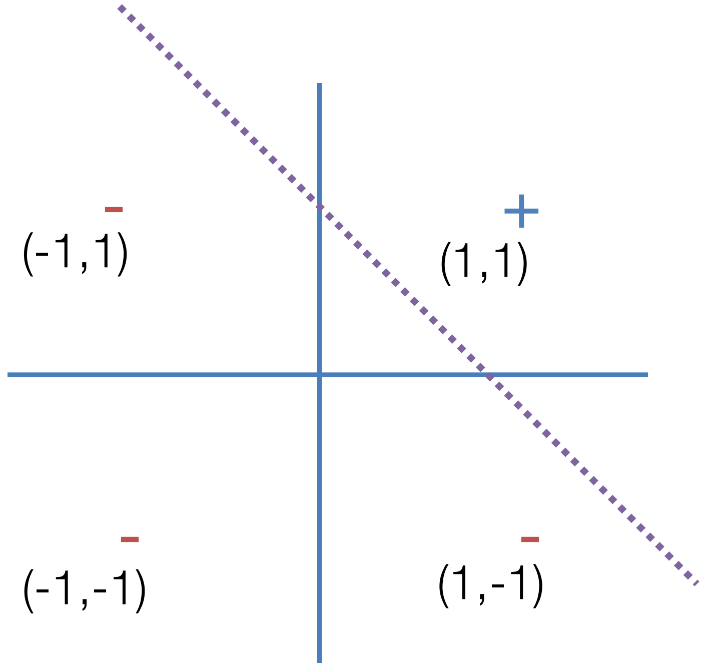
- No need for a convex optimization solver…
- The line that goes through (0,1) and (1,0) seems like a good candidate.
  - In slope-intercept form:

$$x_2 = -x_1 + 1$$

  - But remember that $\mathbf{w}$ in our SVM equations specifies the **normal** to the decision plane, so rearrange:

$$x_1 + x_2 - 1 = 0$$

$$\mathbf{w} = [1, 1], b = -1$$

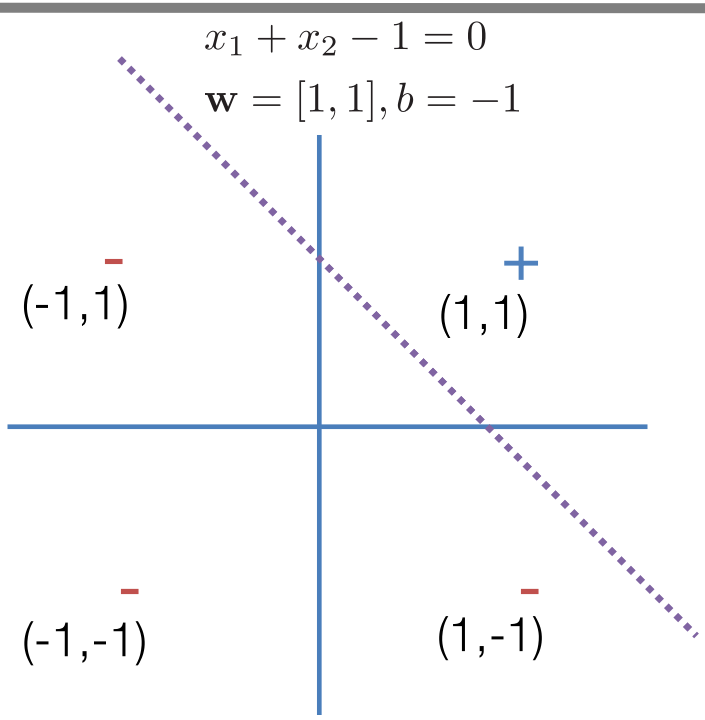


Fitting SVMs

- No need for a convex optimization solver…
- The line that goes through (0,1) and (1,0) seems like a good candidate.
- It classifies all the points correctly:

$$\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$$

$$\text{sign}(1+1-1) = \text{sign}(1) = 1$$
$$\text{sign}(1-1-1) = \text{sign}(-1) = -1$$
$$\text{sign}(-1+1-1) = \text{sign}(-1) = -1$$
$$\text{sign}(-1-1-1) = \text{sign}(-2) = -1$$

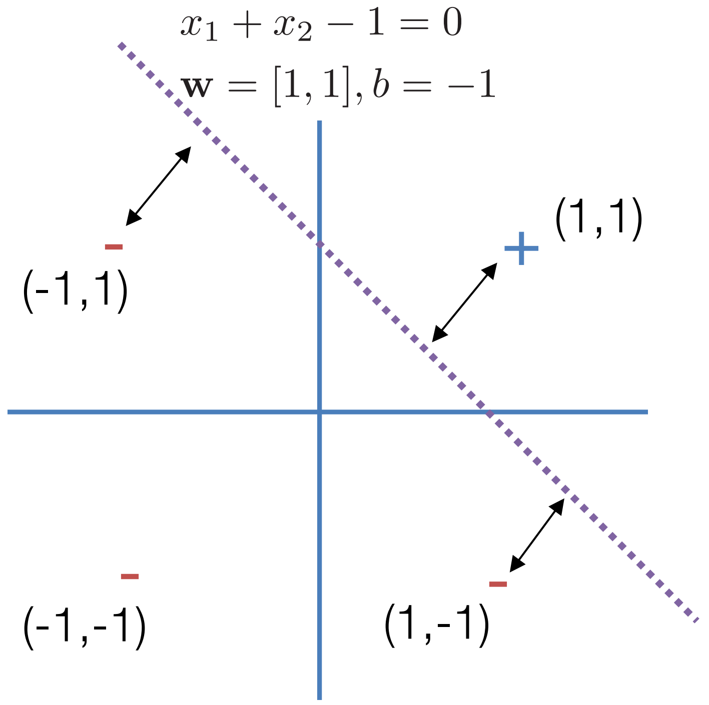


Fitting SVMs

- No need for a convex optimization solver…
- The line that goes through (0,1) and (1,0) seems like a good candidate.
- Now we can calculate the distance between the line and the nearest points (i.e., the candidate support vectors):

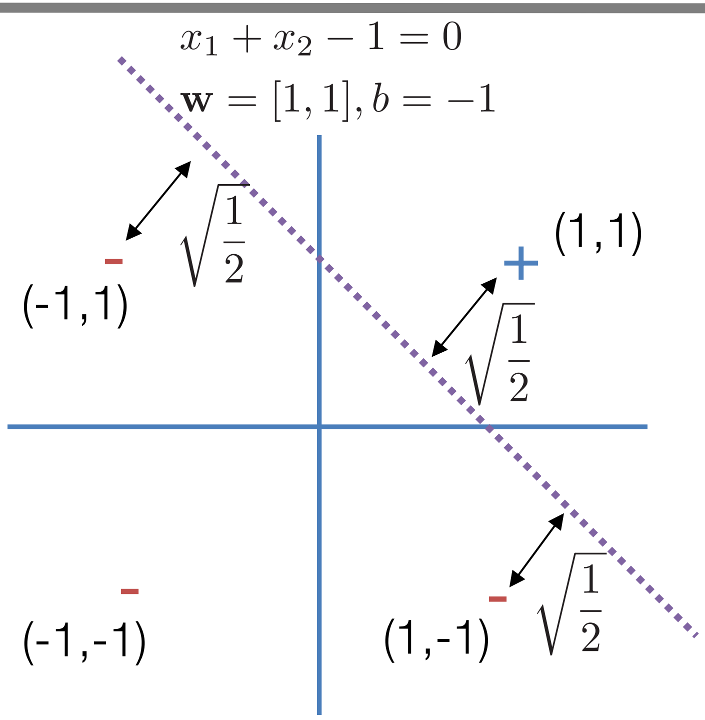
$$d_{\mathbf{w},b}(\mathbf{x}) = \frac{w_1x_1 + w_2x_2 + b}{\sqrt{w_1^2 + w_2^2}}$$

$$d_{\mathbf{w},b}([1,1]) = \sqrt{\frac{1}{2}}, d_{\mathbf{w},b}([1,-1]) = \sqrt{\frac{1}{2}}, d_{\mathbf{w},b}([-1,1]) = \sqrt{\frac{1}{2}}$$



Fitting SVMs

- The distances are all the same!
- So we have indeed found the maximum margin hyperplane because any modification would make it closer to one of the three support vectors!



Fitting SVMs

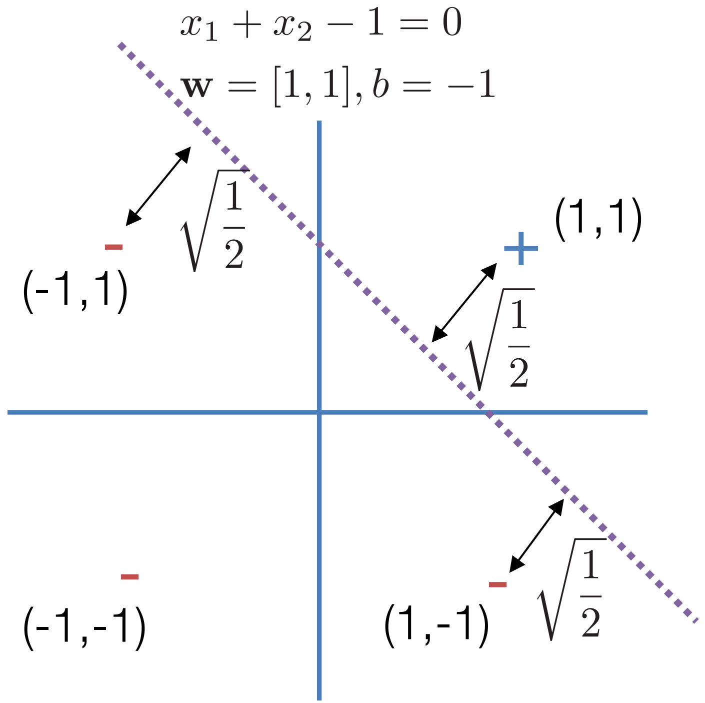
- What is the margin?

$$M = \sqrt{\frac{1}{2}}?$$

or

$$M = 2\sqrt{\frac{1}{2}}?$$

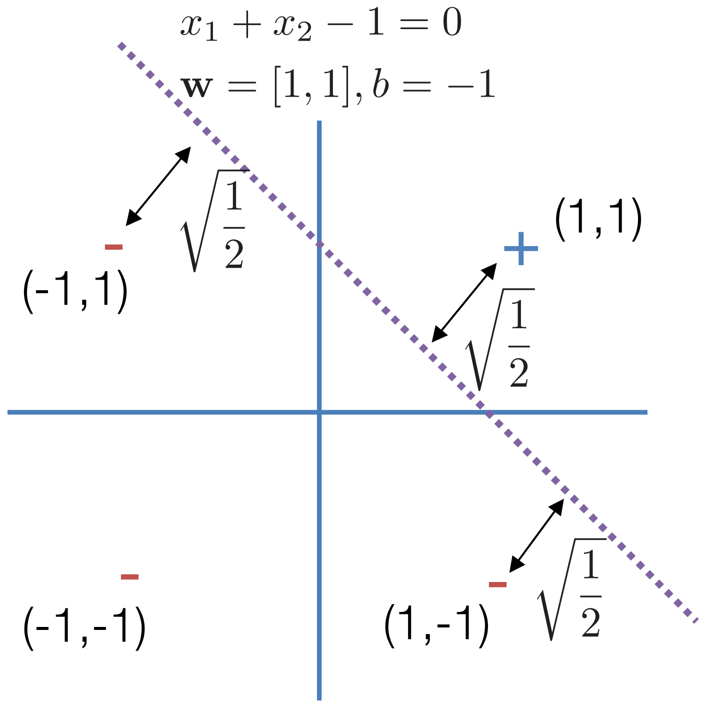
- Different conventions: In the Bishop book M is the distance to the nearest point. In other sources (e.g., Wikipedia, Lecture 10) M is twice the distance to the nearest point.
- We will accept both or be unambiguous.



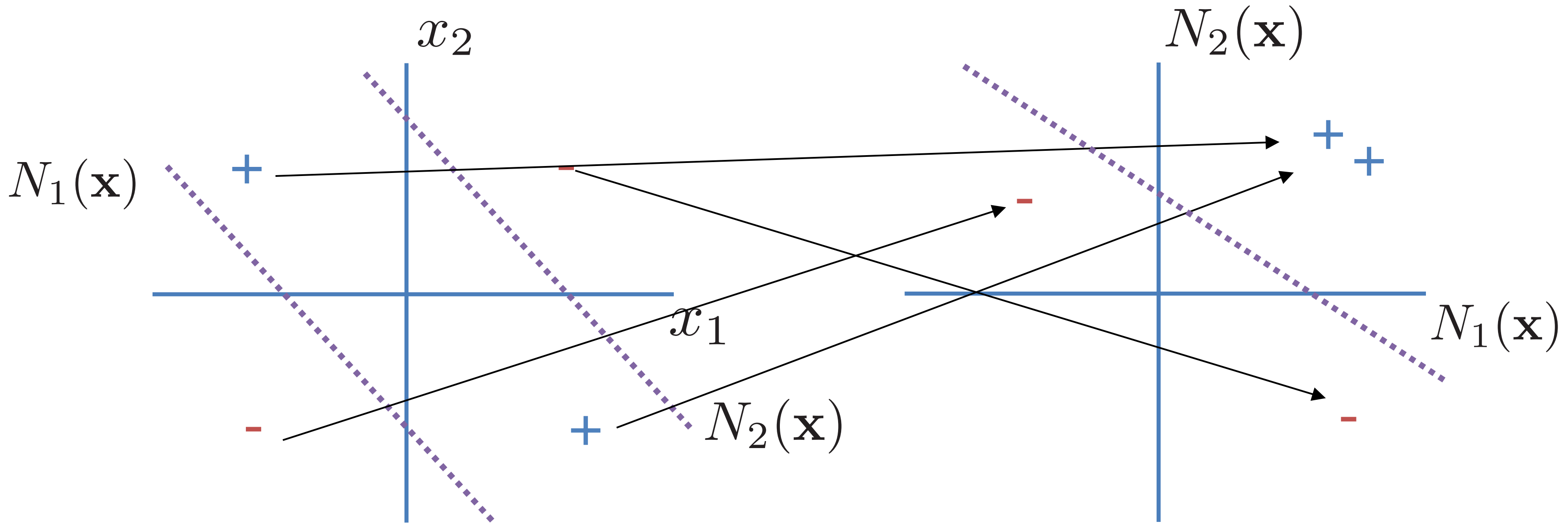
Fitting SVMs

- Note that if we use the definition of the SVM optimization with
  - Minimize $||w||$
    with respect to w
    subject to $y_i \mathbf{w}^T \mathbf{x}_i \geq 1$
  - and $||\mathbf{w}||M = 1$

- Then Bishop's definition is more consistent,
  i.e., $1/||\mathbf{w}|| = \sqrt{\frac{1}{2}}$



Example: A network representing XOR



1) Run two perceptrons (N_1 and N_2) on the original dataset and get the decision boundaries above

2) New dataset defined by the output of N_1 and N_2 is linearly separable!

Elman and Jordan RNNs

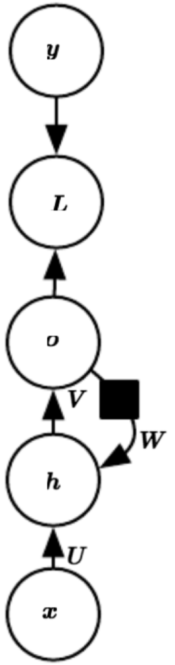
- Q: Which is better?

- Elman RNN:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$
$$\mathbf{o}_t = \phi(\mathbf{V}\mathbf{h}_t + \mathbf{c})$$

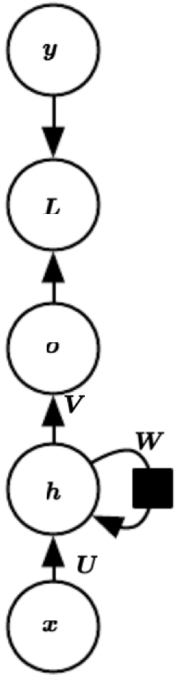
- Jordan RNN:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{o}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$
$$\mathbf{o}_t = \phi(\mathbf{V}\mathbf{h}_t + \mathbf{c})$$



Jordan RNN

vs.



Elman RNN

Generative classification

- Separately model $P(x|y)$ and $P(y)$.
- Use Bayes' rule to estimate $P(y|x)$.

The conditional probability of the target class (i.e., our prediction)

What is the marginal probability of this class? (I.e., ignoring the features, how likely is are we to see class 1?)

$$\boxed{P(y=1|x)} = \frac{\boxed{P(x|y=1)}\boxed{P(y=1)}}{\boxed{P(x)}}$$

How likely are we to see the observed features if the point was from class 1?

What is the marginal probability of the observed features? (This is independent of the class, so we can ignore it)

Generative classification

- If we want probabilities:

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{\sum_{k=1}^{C} P(x|y=k)P(y=k)}$$

- If we want just decisions:

$$P(y=k|x) \propto P(x|y=k)P(y=k)$$

$$\begin{aligned}\text{Prediction} &= \arg\max_k P(x|y=k)P(y=k) \\ &= \arg\max_k \log(P(x|y=k)P(y=k))\end{aligned}$$

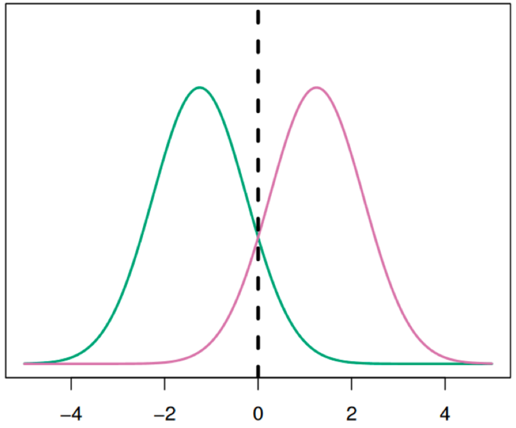
Linear discriminant analysis (LDA)

- LDA makes Gaussian assumptions about P(x|y):

$$\boxed{P(x \mid y)} = \boxed{\frac{e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}}{(2\pi)^{1/2} \mid \Sigma \mid^{1/2}}}$$

Question: What is the probability distribution of features for class y?

Answer: A simple Gaussian/normal distribution!



- In other words, **every class is assumed to be a Gaussian/normally distributed cluster of data points.**

Linear discriminant analysis (LDA)

- LDA makes explicit assumptions about P(x|y):

$$P(x \mid y) = \frac{e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}}{(2\pi)^{1/2} \mid \Sigma \mid^{1/2}}$$

  - Multivariate Gaussian, with mean μ and covariance matrix Σ .
  - Notation: here x is a single instance, represented as a vector.
  - Key assumption of LDA: Both classes have the *same* covariance matrix, Σ.
- Consider the log-odds ratio:

$$\log \frac{P(x|y=1)P(y=1)}{P(x|y=1)P(y=1)} = \underbrace{\log \frac{P(y=1)}{P(y=0)} - \frac{1}{2}\mu_1^\top \Sigma^{-1} \mu_1 + \frac{1}{2}\mu_0^\top \Sigma^{-1} \mu_0}_{w_0} + \underbrace{x^\top \Sigma^{-1}(\mu_1 - \mu_0)}_{x^T w}$$

This is a linear decision boundary! $w_0 + x^T w$

Learning in LDA: Two-class case

- Estimate $P(y)$, μ, Σ, from the training data, then apply log-odds ratio.

Learning in LDA: Two-class case

- Estimate $P(y)$, μ, Σ, from the training data, then apply log-odds ratio.
  - $P(y=0) = N_0 / (N_0 + N_1)$ $P(y=1) = N_1 / (N_0 + N_1)$

    where N_1, N_0 are the # of training samples from classes 1 and 0.

Learning in LDA: Two-class case

- Estimate $P(y)$, μ, Σ, from the training data, then apply log-odds ratio.
  - $P(y=0) = N_0 / (N_0 + N_1)$ $\qquad$ $P(y=1) = N_1 / (N_0 + N_1)$

    where N_1, N_0 are the # of training samples from classes 1 and 0.
  - $\mu_0 = \sum_{i=1:n} I(y_i=0)\, x_i / N_0$ $\qquad$ $\mu_1 = \sum_{i=1:n} I(y_i=1)\, x_i / N_1$

    where $I(x)$ is the indicator function: $I(x)=0$ if $x=0$, $I(x)=1$ if $x=1$.

Learning in LDA: Two-class case

- Estimate $P(y)$, μ, Σ, from the training data, then apply log-odds ratio.
  - $P(y=0) = N_0 / (N_0 + N_1)$ $\quad$ $P(y=1) = N_1 / (N_0 + N_1)$

    where N_1, N_0 are the # of training samples from classes 1 and 0.
  - $\mu_0 = \sum_{i=1:n} I(y_i=0)\, x_i / N_0$ $\quad$ $\mu_1 = \sum_{i=1:n} I(y_i=1)\, x_i / N_1$

    where $I(x)$ is the indicator function: $I(x)=0$ if $x=0$, $I(x)=1$ if $x=1$.
  - $\Sigma = \sum_{k=0:1}\sum_{i=1:n} I(y_i=k)\, (x_i - \mu_k)(x_i - \mu_k)^T / (N_0+N_1-2)$

Learning in LDA: Two-class case

- Estimate $P(y)$, μ, Σ, from the training data, then apply log-odds ratio.
  - $P(y=0) = N_0 / (N_0 + N_1)$ $\quad$ $P(y=1) = N_1 / (N_0 + N_1)$

    where N_1, N_0 are the # of training samples from classes 1 and 0.
  - $\mu_0 = \sum_{i=1:n} I(y_i=0)\, x_i / N_0$ $\quad$ $\mu_1 = \sum_{i=1:n} I(y_i=1)\, x_i / N_1$

    where $I(x)$ is the indicator function: $I(x)=0$ if $x=0$, $I(x)=1$ if $x=1$.
  - $\Sigma = \sum_{k=0:1}\sum_{i=1:n} I(y_i=k)\, (x_i - \mu_k)(x_i - \mu_k)^T / (N_0+N_1-2)$

- **Decision boundary**: Given an input x, classify it as class 1 if the log-odds ratio is >0, classify it as class 0 otherwise.

Gaussian Naïve Bayes

$$P(x_j|y=k) = \frac{1}{\sqrt{2\pi\sigma_{j,k}^2}} e^{-\frac{(x_j-\mu_{j,k})^2}{2\sigma_{j,k}^2}}$$

$$\log(P(\mathbf{x}|y=k)) \propto \sum_{j=1}^{m} -\frac{(x_j-\mu_{j,k})^2}{\sigma_{j,k}^2} - \log(\sigma_{j,k})$$

Gaussian Naïve Bayes

$$\log(\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}) \propto \log(P(y=1)) + \sum_{j=1}^{m} -\frac{(x_j - \mu_{j,1})^2}{\sigma_{j,1}^2} - \log(\sigma_{j,1}) -$$

$$\log(P(y=0)) + \sum_{j=1}^{m} -\frac{(x_j - \mu_{j,0})^2}{\sigma_{j,0}^2} - \log(\sigma_{j,0})$$

Multinomial Naïve Bayes

$$P(\mathbf{x} \mid y = k) = \frac{(\sum_j x_j)!}{\prod_j x_j!} \prod_j {\theta_{j,k}}^{x_j}$$

$$\begin{aligned}\log(P(y = k|\mathbf{x})) &\propto \log(P(y = k)P(\mathbf{x}|y = k)) \\ &\propto \log(P(y = k)) + \sum_{j=1}^{m} x_j \log(\theta_{j,k})\end{aligned}$$

Random forests (Breiman)

- Basic algorithm:
  - Use K bootstrap replicates to train K different trees.
  - At each node, pick m variables at random (use $m<M$, the total number of features).
  - Determine the best test (using normalized information gain).
  - Recurse until the tree reaches maximum depth (no pruning).

- Comments:
  - Each tree has high variance, but the ensemble uses averaging, which reduces variance.
  - Random forests are very competitive in both classification and regression, but still subject to overfitting.

Extremely randomized trees

(Geurts et al., 2005)

- Basic algorithm:
  - Construct K decision trees.
  - Pick m attributes at random (without replacement) and pick a random test involving each attribute.
  - Evaluate all tests (using a normalized information gain metric) and pick the best one for the node.
  - Continue until a desired depth or a desired number of instances (n_{min}) at the leaf is reached.
- <u>Comments</u>:
  - Very reliable method for both classification and regression.
  - The smaller m is, the more randomized the trees are; small m is best, especially with large levels of noise. Small n_{min} means less bias and more variance, but variance is controlled by averaging over trees.