Generative Models

Based on ISL Book and Lecture Notes from Dr. Piyush Rai's ML course at IITK

Bayes theorem for classification

- Thomas Bayes was a famous mathematician whose name represents a big subfield of statistical and probabilistic modeling.
- Here we focus on a simple result, known as Bayes theorem

$$\Pr(Y = k|X = x) = \frac{\Pr(X = x|Y = k) \cdot \Pr(Y = k)}{\Pr(X = x)}$$

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}, \quad \text{where}$$

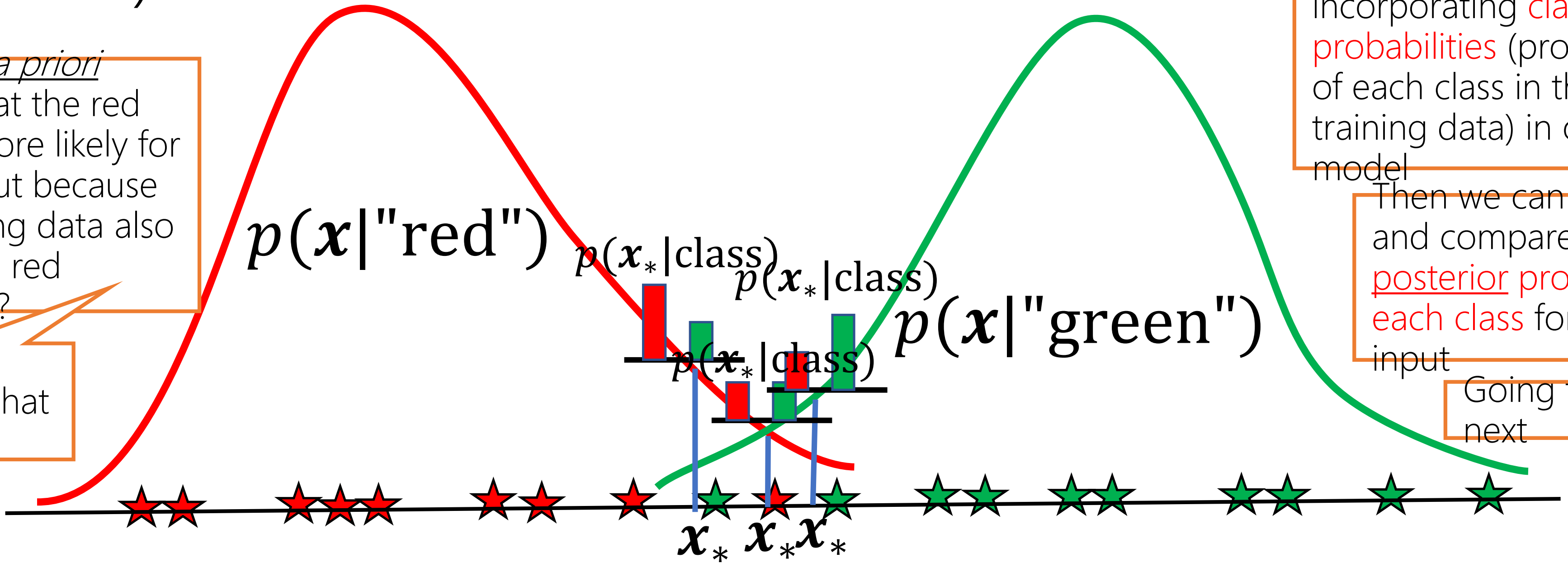
- $f_k(x) = \Pr(X = x|Y = k)$ is the *density* for X in class k. Here we will use normal densities for these, separately in each class.
- $\pi_k = \Pr(Y = k)$ is the marginal or *prior* probability for class k.

Generative Classification: A Basic Idea

- Learn the probability distribution of inputs from each class (“class-conditional”)

What if I *a priori* expect that the red class is more likely for a test input because the training data also had more red examples?

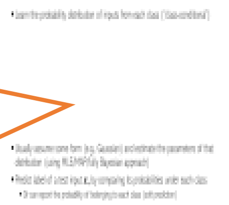
Can I incorporate that knowledge?


$p(\boldsymbol{x}|\text{"red"})$
$p(\boldsymbol{x}_*|\text{class})$
$p(\boldsymbol{x}_*|\text{class})$
$p(\boldsymbol{x}_*|\text{class})$
$p(\boldsymbol{x}|\text{"green"})$
$\boldsymbol{x}_*$
$\boldsymbol{x}_*$
$\boldsymbol{x}_*$

Yes. We can do it by incorporating class prior probabilities (proportion of each class in the training data) in our model

Then we can compute and compare the class posterior probabilities of each class for the test input

Going to talk about this next



- Usually assume some form (e.g., Gaussian) and estimate the parameters of that distribution (using MLE/MAP/fully Bayesian approach)
- Predict label of a test input $\boldsymbol{x}_*$ by comparing its probabilities under

Generative Models for Classification

- Model the distribution of the predictors X separately in each of the response classes (for each value of Y)
  - Alternative to modeling p(Y|X)
  - We then use Bayes' theorem to flip these around into estimates for Pr(Y = k|X = x)
- When the distribution of X within each class is assumed to be normal, it turns out that the model is very similar in form to logistic regression.
- Why use generative models?
  - Logistic Regression is unstable when there is substantial separation between the two classes
  - Generative methods considered here do not suffer from this problem.
  - If the distribution of the predictors X is approximately normal in each of the classes and the sample size is small, then the approaches in this section may be more accurate than logistic regression.

Generative Classification: More Generally..

Roughly speaking, what' s the fraction of each class in the training data

- Consider a classification problem with $K \geq 2$ classes
- The class prior probability of each class $k \in \{1,2,\ldots,K\}$ is $p(y=k)$
- Can use Bayes rule to compute class posterior probability for a test input $\boldsymbol{x}_*$

Class prior distribution for class k

Class-conditional distribution of inputs from class k

$$p(y_* = k|\boldsymbol{x}_*, \theta) = \frac{p(\boldsymbol{x}_*, y_* = k|\theta)}{p(\boldsymbol{x}_*|\theta)} = \frac{p(y_* = k|\theta)p(\boldsymbol{x}_*|y_* = k, \theta)}{p(\boldsymbol{x}_*|\theta)}$$

This is just the marginal distribution of the joint distribution in the numerator (summed over all K values of y_*)

θ collectively denotes the parameters the joint distribution of inputs and labels depends on

Setting $p(y_* = k|\theta) = 1/K$ will give us the approach that predicts by comparing the probabilities $p(\boldsymbol{x}_*|y_* = k, \theta)$ of $\boldsymbol{x}_*$ under each of the classes

- We will first estimate the parameters of class prior and class-conditional distributions. Once estimated, we can use the above rule to predict the label for any test input

Estimating Class Priors

Note: Can also do MAP estimation using a Dirichlet prior on π (this is akin to using Beta prior for doing MAP estimation for the bias of a coin). May try this as an exercise

- Estimating class priors $p(y = k)$ is usually straightforward in gen. classification
- Roughly speaking, it is the proportion of training examples from each class
  - Note: The above is true only when doing MLE (as we will see shortly)
  - If estimating class priors using MAP/fully Bayesian, they will be a "smooth version" of the proportions (because of the effect of regularization)

$\pi_k = p(y = k)$

These probabilities sum to 1: $\sum_{k=1}^{K} \pi_k = 1$

Generalization of Bernoulli

$$p(y|\boldsymbol{\pi}) = \text{multinoulli}(y|\pi_1, \pi_2, \ldots, \pi_K) = \prod_{k=1}^{K} \pi_k^{\mathbb{I}[y=k]}$$

- The class prior distribution is assumed to be a discrete distribution (multinoulli)

$$\boldsymbol{\pi}_{MLE} = \arg\max_{\boldsymbol{\pi}} \sum_{n=1}^{N} \log p(y_n|\boldsymbol{\pi})$$

Subject to constraint $\sum_{k=1}^{K} \pi_k = 1$

Can use Lagrange based opt. (note that we have an equality constraint)

Exercise: Verify that the MLE solution will be $p(y = k) = \pi_k = N_k/N$ where $N_k = \sum_{n=1}^{N} \mathbb{I}[y = k]$ (the frac. of class k examples)

Estimating Class-Conditionals

To be estimated using inputs from class k

- Can assume an appropriate distribution $p(\boldsymbol{x}|y=k,\theta)$ for inputs of each class

- If $\boldsymbol{x}$ is D-dim, it will be a D-dim. distribution. C various factors
  - Nature of input features, e.g.,
    - If $\boldsymbol{x} \in \mathbb{R}^D$, can use a D-dim Gaussian $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
    - If $\boldsymbol{x} \in \{0,1\}^D$, can use D Bernoullis (one for each feature)
    - Can also choose more flexible/complex distributions if possible to estimate
  - Amount of training data available
    - With little data from a class, difficult to estimate the params of its class-cond. distribution

Some workarounds: Use strong regularization, or a simple form of the class-conditional (e.g., use a spherical/diagonal rather than a full covariance if the class-cond is Gaussian), or assume features are independent given class ("naïve Bayes" assumption)

A big issue especially if the number of features (D) is very large

- Once decided the form of class-cond, estimate $\boldsymbol{\theta}$ via MLE/MAP/Bayesian infer.
  - This essentially is a density estimation problem for the class-cond.

Linear Discriminant Analysis (p=1)

The Gaussian density has the form

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2}$$

Here μ_k is the mean, and σ_k^2 the variance (in class k). We will assume that all the $\sigma_k = \sigma$ are the same.

Plugging this into Bayes formula, we get a rather complex expression for $p_k(x) = \Pr(Y = k|X = x)$:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_l}{\sigma}\right)^2}}$$

Happily, there are simplifications and cancellations.

Discriminant Functions (p=1)

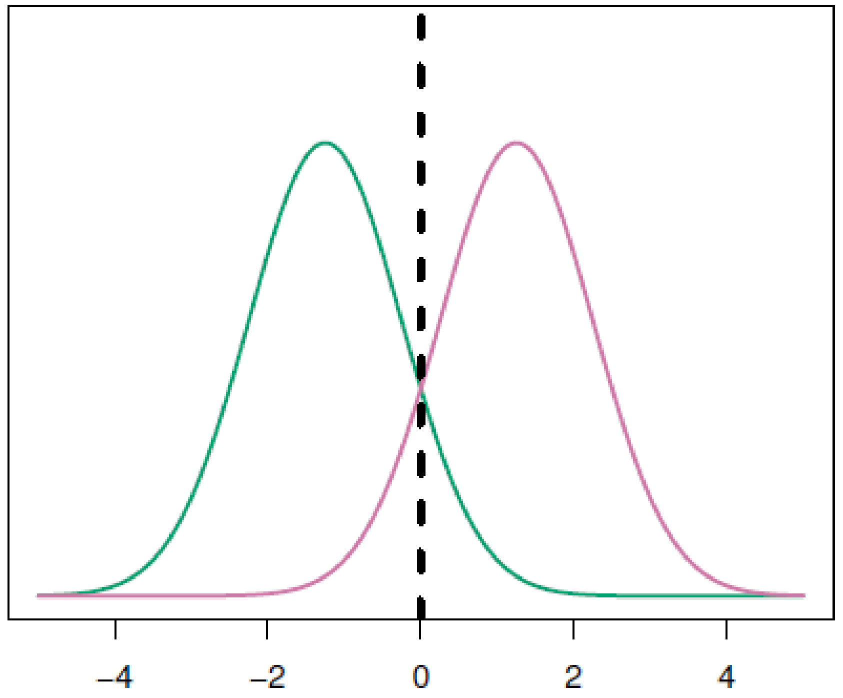
- To classify at the value X = x, we need to see which of the $p_k(x)$ is largest.
- Taking logs, and discarding terms that do not depend on k, we see that this is equivalent to assigning x to the class with the largest discriminant score:

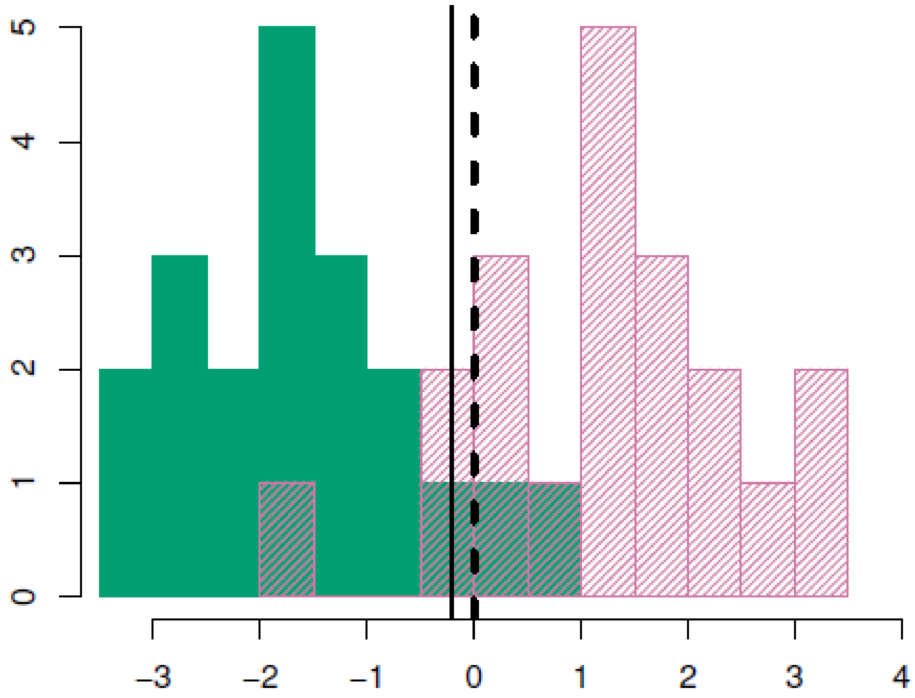
$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

Note that $\delta_k(x)$ is a *linear* function of x.

If there are $K = 2$ classes and $\pi_1 = \pi_2 = 0.5$, then one can see that the *decision boundary* is at

$$x = \frac{\mu_1 + \mu_2}{2}.$$





Example with $\mu_1 = -1.5$, $\mu_2 = 1.5$, $\pi_1 = \pi_2 = 0.5$, and $\sigma^2 = 1$.

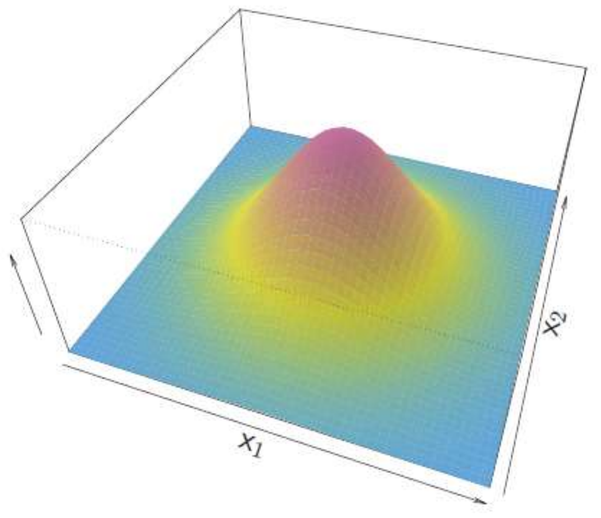
Typically we don't know these parameters; we just have the training data. In that case we simply estimate the parameters and plug them into the rule.

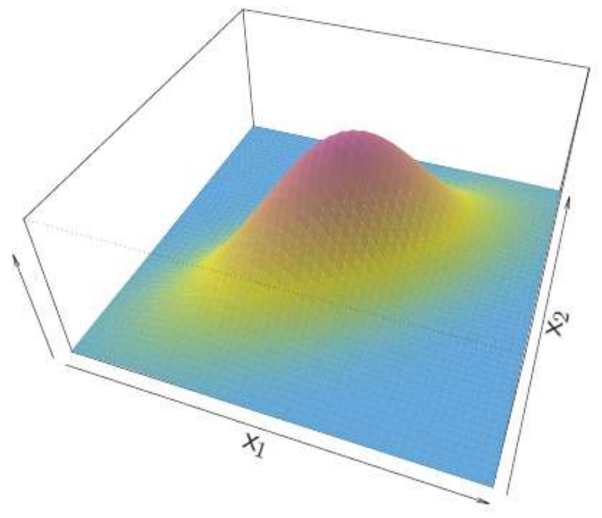
Estimating the parameters

$$
\begin{aligned}
\hat{\pi}_k &= \frac{n_k}{n} \\
\hat{\mu}_k &= \frac{1}{n_k} \sum_{i:\, y_i = k} x_i \\
\hat{\sigma}^2 &= \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:\, y_i = k} (x_i - \hat{\mu}_k)^2 \\
&= \sum_{k=1}^{K} \frac{n_k - 1}{n - K} \cdot \hat{\sigma}_k^2
\end{aligned}
$$

where $\hat{\sigma}_k^2 = \frac{1}{n_k - 1} \sum_{i:\, y_i = k} (x_i - \hat{\mu}_k)^2$ is the usual formula for the estimated variance in the kth class.

Linear Discriminant Analysis (p>1)





$$\text{Density: } f(x) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)}$$

$$\text{Discriminant function: } \delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log \pi_k$$

Despite its complex form,
$\delta_k(x) = c_{k0} + c_{k1}x_1 + c_{k2}x_2 + \ldots + c_{kp}x_p$ — a linear function.

A Closer Look at the Linear Case

- For the linear case (when $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}, \forall k$), the class posterior probability

$$p(y = k|\mathbf{x}, \theta) \propto \pi_k \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right]$$

- Expanding further, we can write the above as

$$p(y = k|\mathbf{x}, \theta) \propto \exp\left[\boldsymbol{\mu}_k^\top \Sigma^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_k^\top \Sigma^{-1}\boldsymbol{\mu}_k + \log \pi_k\right] \exp\left[\boldsymbol{x}^\top \Sigma^{-1}\mathbf{x}\right]$$

- Therefore, the above class posterior probability can be written as

$$p(y = k|\mathbf{x}, \theta) = \frac{\exp\left[\mathbf{w}_k^\top \mathbf{x} + b_k\right]}{\sum_{k=1}^K \exp\left[\mathbf{w}_k^\top \mathbf{x} + b_k\right]}$$

$$\boldsymbol{w}_k = \Sigma^{-1}\boldsymbol{\mu}_k \qquad b_k = -\tfrac{1}{2}\boldsymbol{\mu}_k^\top \Sigma^{-1}\boldsymbol{\mu}_k + \log \pi_k$$

If all Gaussians class-cond have the same covariance matrix (basically, of all classes are assumed to have the same shape)

- The above has *exactly* the same form as softmax classification(thus softmax is a special case of a generative classification model with Gaussian class-conditionals)

A Very Special Case: LwP Revisited

- Note the prediction rule when $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}, \forall k$

$$\begin{aligned}\hat{y} = \arg\max_k p(y=k|\mathbf{x}) &= \arg\max_k \quad \pi_k \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right] \\ &= \arg\max_k \quad \log \pi_k - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\end{aligned}$$

- Also assume all classes to have equal no. of training examples, i.e., $\pi_k = 1/K$. Then

$$\hat{y} = \arg\min_k \quad (\mathbf{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)$$

The Mahalanobis distance matrix = $\boldsymbol{\Sigma}^{-1}$

- Equivalent to assigning $\boldsymbol{x}$ to the "closest" class in terms of a Mahalanobis distance

- If we further assume $\boldsymbol{\Sigma} = \boldsymbol{I}_D$ then the above is exactly the LwP rule

Example

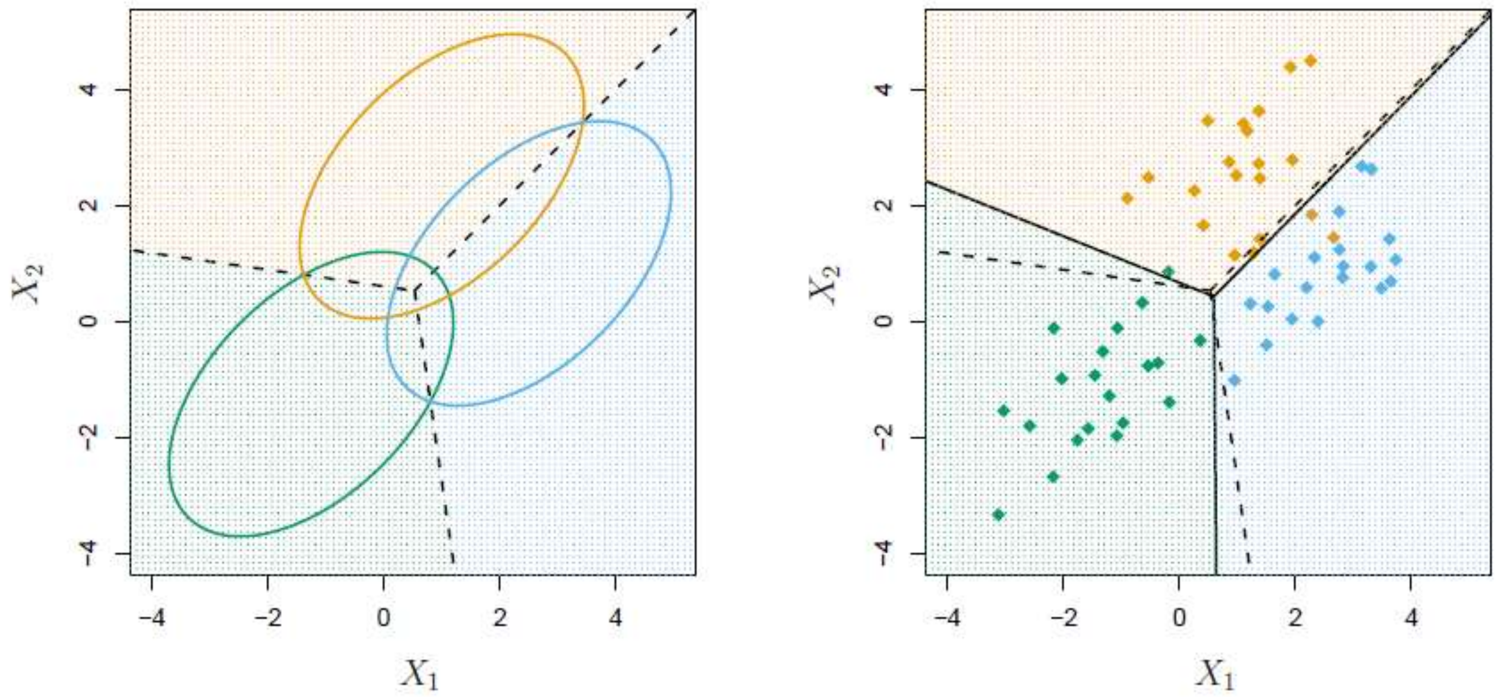
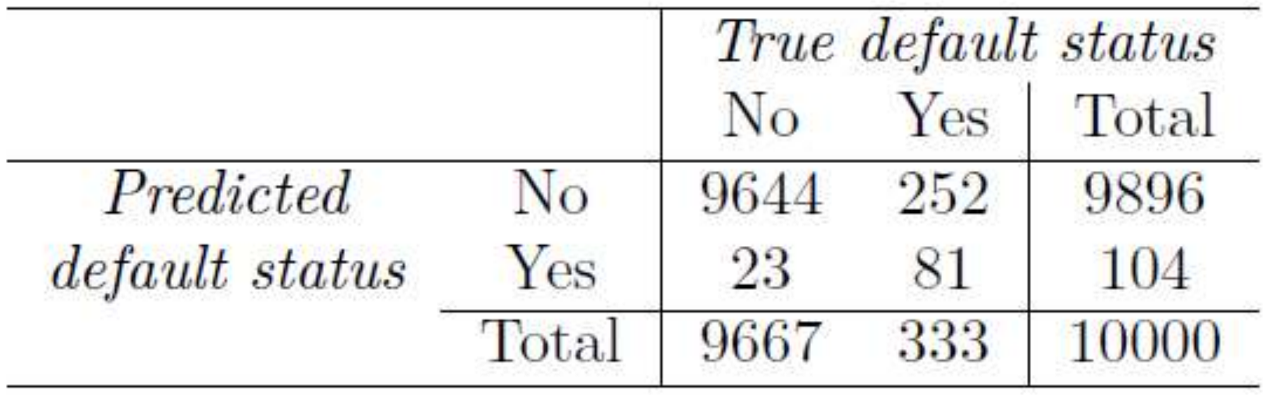


FIGURE 4.6. *An example with three classes. The observations from each class are drawn from a multivariate Gaussian distribution with $p = 2$, with a class-specific mean vector and a common covariance matrix.* Left: *Ellipses that contain 95 % of the probability for each of the three classes are shown. The dashed lines are the Bayes decision boundaries.* Right: *20 observations were generated from each class, and the corresponding LDA decision boundaries are indicated using solid black lines. The Bayes decision boundaries are once again shown as dashed lines.*

Credit Card Default using LDA



		True default status		
		No	Yes	Total
Predicted	No	9644	252	9896
default status	Yes	23	81	104
	Total	9667	333	10000

- The above table is called a confusion matrix
- Training error rate: 2.75%
- Only 3.33% of individuals in the training dataset defaulted
  - If we classified to the prior always to class No in this case, we would make 333=10000 errors, or only 3.33%.
  - Of the true No's, we make 23=9667 = 0:2% errors; of the true Yes's, we make 252=333 = 75:7% errors!

Types of errors

False positive rate: The fraction of negative examples that are classified as positive — 0.2% in example.

False negative rate: The fraction of positive examples that are classified as negative — 75.7% in example.

We produced this table by classifying to class `Yes` if

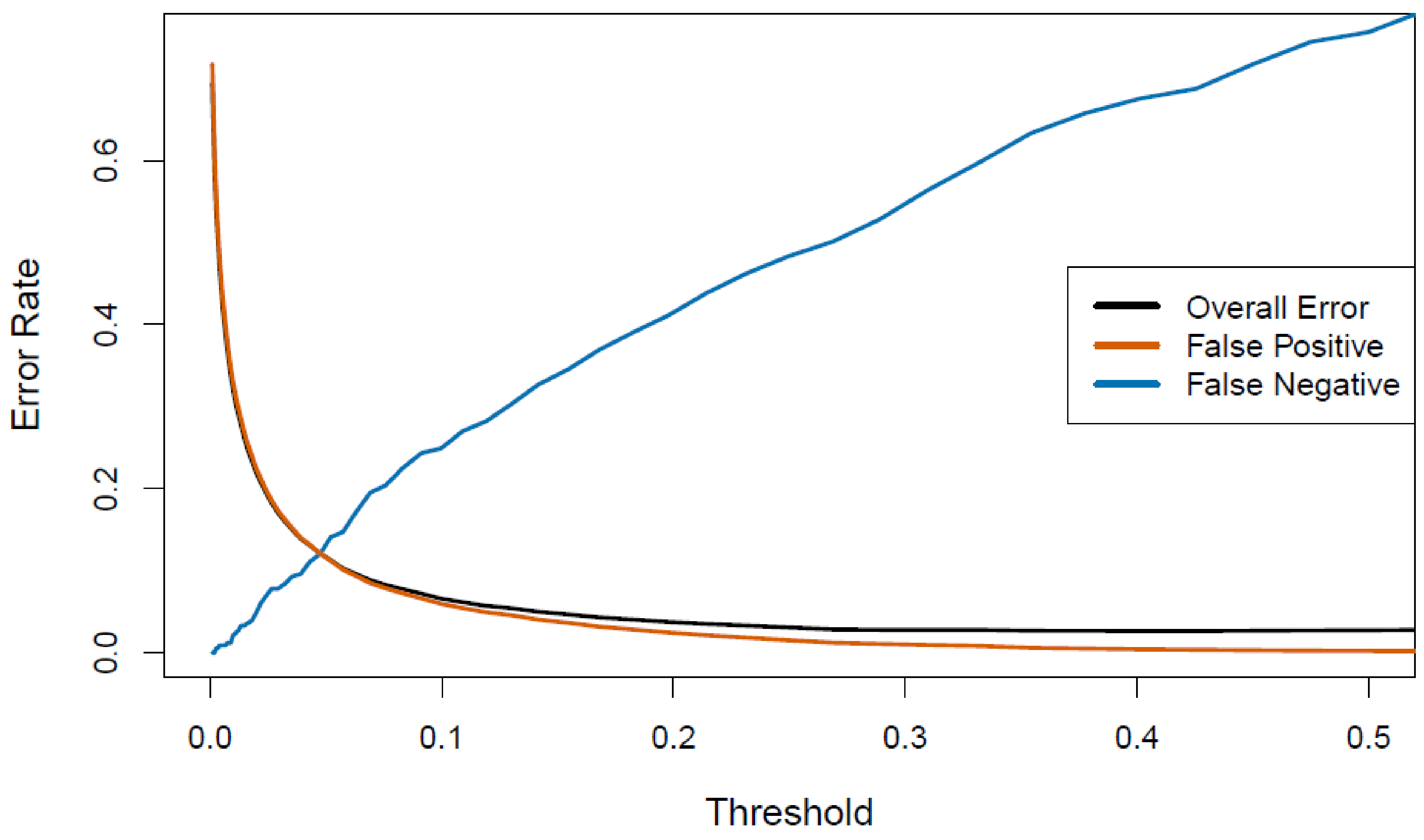
$$\widehat{\Pr}(\texttt{Default} = \texttt{Yes}|\texttt{Balance}, \texttt{Student}) \geq 0.5$$

We can change the two error rates by changing the threshold from 0.5 to some other value in $[0, 1]$:

$$\widehat{\Pr}(\texttt{Default} = \texttt{Yes}|\texttt{Balance}, \texttt{Student}) \geq \textit{threshold},$$

and vary *threshold*.

Varying the *threshold*



Class-specific performance is also important in medicine and biology,
where the terms *sensitivity* and *specificity* characterize the performance of sensitivity specificity a classifier or screening test.

In this case the sensitivity is the percentage of true defaulters that are identified; it equals 24.3 %.

The specificity is the percentage of non-defaulters that are correctly identified; it equals (1 − 23/9667) = 99.8 %.

In order to reduce the false negative rate, we may want to reduce the threshold to 0.1 or less.

A ROC curve for the LDA classifier on the Default data.

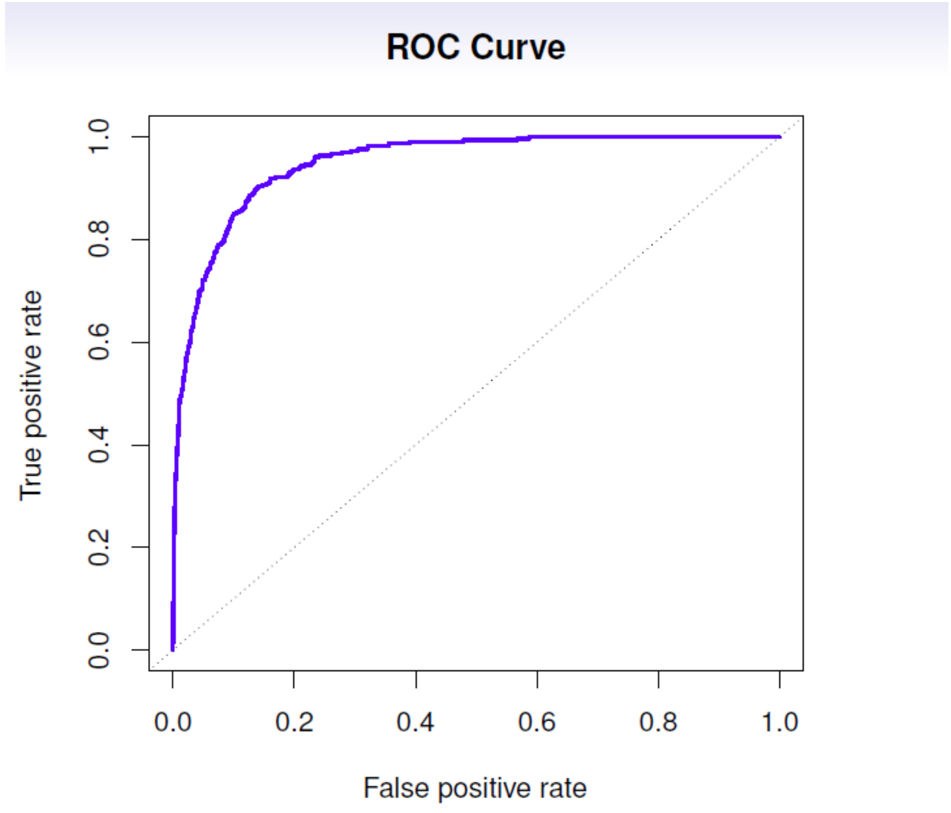
It traces out two types of error as we vary the threshold value for the posterior probability of default. The actual thresholds are not shown.

The true positive rate is the sensitivity: the fraction of defaulters that are correctly identified, using a given threshold value.

The false positive rate is 1-specificity: the fraction of non-defaulters that we classify incorrectly as defaulters, using that same threshold value.

The ideal ROC curve hugs the top left corner, indicating a high true positive rate and a low false positive rate.

The dotted line represents the “no information” classifier; this is what we would expect if student status and credit card balance are not associated with probability of default.



Sometimes we use the AUC or area under the curve to summarize the overall performance. Higher AUC is good.

		True class		
		− or Null	+ or Non-null	Total
Predicted class	− or Null	True Neg. (TN)	False Neg. (FN)	N*
	+ or Non-null	False Pos. (FP)	True Pos. (TP)	P*
	Total	N	P	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Name	Definition	Synonyms
False Pos. rate	FP/N	Type I error, 1−Specificity
True Pos. rate	TP/P	1−Type II error, power, sensitivity, recall
Pos. Pred. value	TP/P*	Precision, 1−false discovery proportion
Neg. Pred. value	TN/N*	

TABLE 4.7. *Important measures for classification and diagnostic testing, derived from quantities in Table 4.6.*

Other forms of Discriminant Analysis

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

When $f_k(x)$ are Gaussian densities, with the same covariance matrix $\mathbf{\Sigma}$ in each class, this leads to linear discriminant analysis. By altering the forms for $f_k(x)$, we get different classifiers.

- With Gaussians but different $\mathbf{\Sigma}_k$ in each class, we get *quadratic discriminant analysis*.
- With $f_k(x) = \prod_{j=1}^{p} f_{jk}(x_j)$ (conditional independence model) in each class we get *naive Bayes*. For Gaussian this means the $\mathbf{\Sigma}_k$ are diagonal.
- Many other forms, by proposing specific density models for $f_k(x)$, including nonparametric approaches.

Gen. Classifn. using Gaussian Class-conditionals

- The generative classification model $p(y = k|\boldsymbol{x}) = \frac{p(y=k)p(\boldsymbol{x}|}{p(\boldsymbol{x}|\theta)}$
- Assume each class-conditional $p(\boldsymbol{x}|y = k)$ to be a Gaussian

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}_k|}}\exp[-(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)]$$

A benefit of modeling each class by a distribution (recall that LwP had issues)

Since the Gaussian's covariance models its shape, we can learn the shape of each class ☺

- Class prior is multinoulli (we already saw): $p(y = k) = \pi_k, \pi_k \in (0,1), \sum_{k=1}^{K}\pi_k = 1$
- Let's denote the parameters of the model collectively by $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$
  - Can estimate these using MLE/MAP/Bayesian inference
  - Already saw the MLE solution for $\boldsymbol{\pi}: \pi_k = N_k/N$ (can also do MA
  - MLE solution for $\boldsymbol{\mu}_k = \frac{1}{}\sum \boldsymbol{x}_n$, $\boldsymbol{\Sigma}_k = \frac{1}{}\sum (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top$

Can also do MAP estimation for $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ using a Gaussian prior on $\boldsymbol{\mu}_k$ and inverse Wishart prior on $\boldsymbol{\Sigma}_k$

Exercise: Try to derive this. I will provide a separate note containing the derivation

- st wil

Can predict the most likely class for the test input $\boldsymbol{x}_*$ by comparing these probabilities for all values of k

$$p(y_* = k|\mathbf{x}_*, \theta) = \frac{\pi_k|\boldsymbol{\Sigma}_k|^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}_* - \boldsymbol{\mu}_k)^\top\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_* - \boldsymbol{\mu}_k)\right]}{\sum_{k=1}^{K}\pi_k|\boldsymbol{\Sigma}_k|^{-1/2}\exp\left[-\frac{1}{2}(\mathbf{x}_* - \boldsymbol{\mu}_k)^\top\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_* - \boldsymbol{\mu}_k)\right]}$$

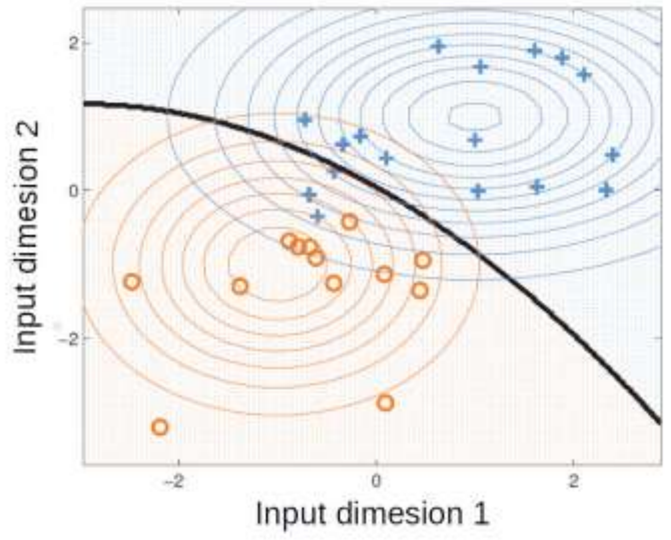
Note that the exponent has a Mahalanobis distance like term. Also, accounts for the fraction of training examples in class k

Decision Boundary with Gaussian Class-Conditional

- As we saw, the prediction rule when using Gaussian class-conditional

$$p(y = k|\mathbf{x}, \theta) = \frac{\pi_k |\boldsymbol{\Sigma}_k|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right]}{\sum_{k=1}^{K} \pi_k |\boldsymbol{\Sigma}_k|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right]}$$

- The decision boundary between any pair of classes will be a quadratic curve



Reason: For any two classes k and k' at the decision boundary, we will have $p(y = k|x, \theta) = p(y = k'|x, \theta)$. Comparing their logs and ignoring terms that don' t cont

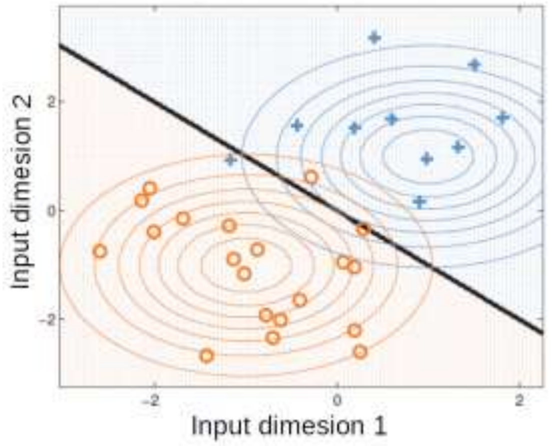
$$(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - (\mathbf{x} - \boldsymbol{\mu}_{k'})^\top \boldsymbol{\Sigma}_{k'}^{-1}(\mathbf{x} - \boldsymbol{\mu}_{k'}) = 0$$

Decision boundary contains all inputs $\boldsymbol{x}$ that satisfy the above

This is a quadratic function of $\boldsymbol{x}$ (this model is sometimes

Decision Boundary with Gaussian Class-Conditional

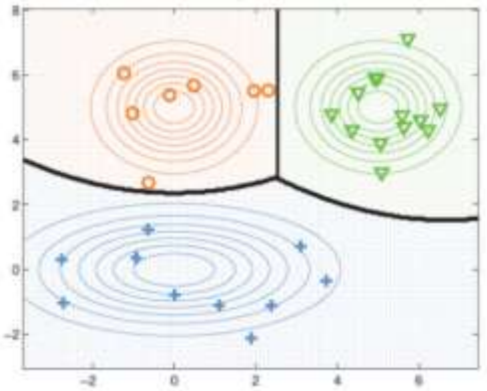
- Assume all classes are modeled using the same covariance matrix $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}, \forall k$
- In this case, the decision boundary b/w any pair of classes will be linear

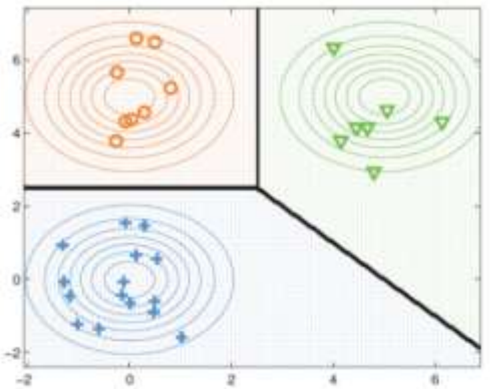


Reason: Again using $p(y = k|x, \theta) = p(y = k'|x, \theta)$, comparing their logs and ignoring terms that don' t contain $\boldsymbol{x}$, we have

$$(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - (\mathbf{x} - \boldsymbol{\mu}_{k'})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_{k'}) = 0$$

Quadratic terms of $\boldsymbol{x}$ will cancel out; only linear terms will remain; hence decision boundary will be a linear function of $\boldsymbol{x}$ (Exercise: Verify that we can indeed write the decision boundary between this pair of classes as $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ where $\boldsymbol{w}$ and b depend on $\boldsymbol{\mu}_k, \boldsymbol{\mu}_{k'}$ and $\boldsymbol{\Sigma}$)





If we assume the covariance matrices of the assumed Gaussian class-conditionals for any pair of classes to be equal, then the learned separation boundary b/w this pair of classes will be linear; otherwise, quadratic as shown in the figure on left

Naive Bayes

Assumes features are independent in each class.
Useful when p is large, and so multivariate methods like QDA and even LDA break down.

- Gaussian naive Bayes assumes each $\mathbf{\Sigma}_k$ is diagonal:

$$\begin{aligned} \delta_k(x) &\propto \log\left[\pi_k \prod_{j=1}^{p} f_{kj}(x_j)\right] \\ &= -\frac{1}{2}\sum_{j=1}^{p}\left[\frac{(x_j-\mu_{kj})^2}{\sigma_{kj}^2} + \log\sigma_{kj}^2\right] + \log\pi_k \end{aligned}$$

- can use for *mixed* feature vectors (qualitative and quantitative). If X_j is qualitative, replace $f_{kj}(x_j)$ with probability mass function (histogram) over discrete categories.

Despite strong assumptions, naive Bayes often produces good classification results.

Logistic Regression versus LDA

For a two-class problem, one can show that for LDA

$$\log\left(\frac{p_1(x)}{1-p_1(x)}\right) = \log\left(\frac{p_1(x)}{p_2(x)}\right) = c_0 + c_1 x_1 + \ldots + c_p x_p$$

So it has the same form as logistic regression.

The difference is in how the parameters are estimated.

- Logistic regression uses the conditional likelihood based on $\Pr(Y|X)$ (known as *discriminative learning*).
- LDA uses the full likelihood based on $\Pr(X, Y)$ (known as *generative learning*).
- Despite these differences, in practice the results are often very similar.

Footnote: logistic regression can also fit quadratic boundaries like QDA, by explicitly including quadratic terms in the model.

Generative Classification: Some Comments

- A simple but powerful approach to probabilistic classification
- Especially easy to learn if class-conditionals are simple
  - E.g., Gaussian with diagonal covariances ⇒ Gaussian naïve Bayes
  - Another popular model is multinomial naïve Bayes (widely used for document classification)
  - The naïve Bayes assumption: features are conditiona

$$p(\boldsymbol{x}|y=k)=\prod_{d=1}^{D} p(x_d|y=k)$$

Benefit: Instead of estimating a D-dim distribution which may be hard (if we don' t have enough data), we will estimate D one-dim distributions (much simpler task)

- Can choose the f nditionals $p(\boldsymbol{x}|y=k)$ based on the type of inputs $\boldsymbol{x}$

Will see such methods later

- Can handle missing data (e.g., if some part of the input $\boldsymbol{x}$ is missing) or missing labels

Will see such methods later

Comparing classifiers: LDA, QDA, Naïve Bayes, Logistic Regression

- Comparison using: $\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right)$

- LDA, like logistic regression, assumes that the log odds of the posterior probabilities is linear in x.

$$a_k + \sum_{j=1}^{p} b_{kj} x_j, \qquad a_k = \log\left(\frac{\pi_k}{\pi_K}\right) - \frac{1}{2}(\mu_k + \mu_K)^T \mathbf{\Sigma}^{-1}(\mu_k - \mu_K)$$

b_{kj} j^{th} component of $\mathbf{\Sigma}^{-1}(\mu_k - \mu_K)$

For QDA: $$\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right) = a_k + \sum_{j=1}^{p} b_{kj} x_j + \sum_{j=1}^{p}\sum_{l=1}^{p} c_{kjl} x_j x_l,$$

Naïve Bayes: $$a_k + \sum_{j=1}^{p} g_{kj}(x_j),$$

Observations

- LDA is a special case of QDA
- Any classifier with a linear decision boundary is a special case of naïve Bayes
    - In particular, this means that LDA is a special case of naive Bayes
- Model density in the naive Bayes using a one-dimensional Gaussian distribution
    - In this case, naive Bayes is actually a special case of LDA
    - Covariance matrix restricted to be a diagonal matrix
- Neither QDA nor naive Bayes is a special case of the other.
    - Naïve Bayes can produce a more flexible fit, since any choice can be made for gkj(xj).
    - Naïve Bayes restricted to a purely *additive* fit
    - QDA includes multiplicative terms of the attributes
    - QDA has the potential to be more accurate in settings where interactions among the predictors are important in discriminating between classes

Discriminative vs Generative

Proponents of discriminative models: Why bother modeling $\boldsymbol{x}$ if $\boldsymbol{y}$ is what you care about? Just model $\boldsymbol{y}$ directly instead of working hard to model x by learning the class-conditional

- Recall that discriminative approaches model $\boldsymbol{p(y|x)}$ (
- Generative approaches model $\boldsymbol{p(y|x)}$ via $\boldsymbol{p(x,y)}$
- Number of parameters: Discriminative models have fewer parameters to be learned
  - Just the weight vector/matrix $\boldsymbol{w}/\boldsymbol{W}$ in case of logistic/softmax classification
- Ease of parameter estimation: Debatable as to which one is easier
  - For "simple" class-conditionals, easier for gen. classifn model (often closed-form solution)
  - Parameter estimation for discriminative models (logistic/softmax) usually requires iterative methods(although objective functions usually have global optima)
- Dealing with missing features: Generative models can handle this easily
  - E.g., by integrating out the missing features while estimating the parameters)
- Inputs with features having mixed types: Generative model can handle

Discriminative vs Generative (Contd)

- Leveraging unlabeled data: Generative models can handle this easily by treating the missing labels are latent variables and are ideal for Semi-supervised Learning. Discriminative models can't do it easily
- Adding data from new classes: Discriminative model will need to be re-trained on all classes all over again. Generative model will just require estimating the class-cond of newly added classes
- Have lots of labeled training data? Discriminative models usually work very well
- Final Verdict? Despite generative classification having some clear advantages, both methods can be quite powerful (the actual choice may be dictated by the problem)
  - Important to be aware of their strengths/weaknesses, and also the connections between these
- Possibility of a Hybrid Design? Yes, Generative and Disc. models can be combined, e.g.,
  - "Principled Hybrids of Generative and Discriminative Models" (Lassere et al, 2006)

Generative Models for Regression

- Yes, we can even model regression problems using a generative approach
- Note that the output y is not longer discrete (so no notion of a class-conditional)
- However, the basic rule of recovering a conditional from joint would still apply

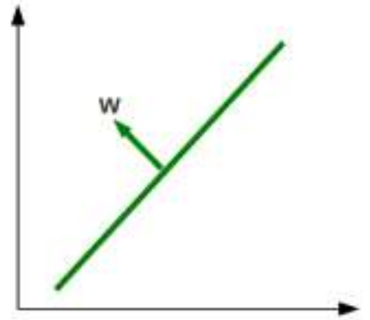
$$p(y|\boldsymbol{x},\theta) = \frac{p(\boldsymbol{x},y|\theta)}{p(\boldsymbol{x}|\theta)}$$

- Thus we can model the joint distribution $p(\boldsymbol{x},y|\theta)$ of features $\boldsymbol{x}$ and outputs $y \in \mathbb{R}$
  - If features are real-valued the we can model $p(\boldsymbol{x},y|\theta)$ using a $(D+1)$-dim Gaussian
  - From this $(D+1)$-dim Gaussian, we can get $p(y|\boldsymbol{x},\theta)$using Gaussian conditioning formula

Hyperplane Based classifiers

Hyperplane

- Separates a D-dimensional space into two **half-spaces** (positive and negative)
- Defined by a normal vector $\boldsymbol{w} \in \mathbb{R}^D$ (pointing towards positive half-space)



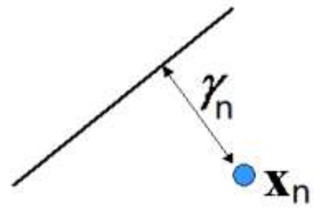
$b > 0$ means moving $\boldsymbol{w}^\top \boldsymbol{x} = 0$ along the direction of w; $b < 0$ means in opp. dir.

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$$

- Equation of the hyperplane: $\boldsymbol{w}^\top \boldsymbol{x} = 0$
- Assumption: The hyperplane passes through origin. If not, add a bias term b
- Distance t rplan $= 0$

Can be positive or negative

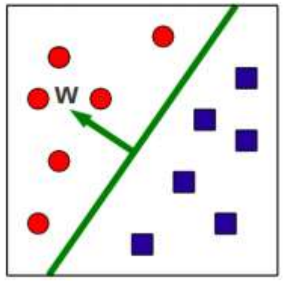
$$\gamma_n = \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{||\boldsymbol{w}||}$$



Hyperplane based (binary) classification

- Basic idea: Learn to separate two classes by a hyperplane $\boldsymbol{w}^\top \boldsymbol{x} +$

For multi-class classification with hyperplanes, there will be multiple hyperplanes (e.g., one for each pair of classes); more on this later



Prediction Rule

$$y_* = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_* + b)$$

- The hyperplane may be "implied" by the model, or learned directly
  - Implied: Prototype-based classification, nearest neighbors, generative classification, etc
  - Directly learned: Logistic regression, Perceptron, Support Vector Machine (SVM), etc
- The "direct" approach defines a model with params $\boldsymbol{w}$ (and optionally a bias param b)
  - The parameters are learned by optimizing a classification loss function (will soon see examples)
  - These are also discriminative approaches – $\boldsymbol{x}$ is not modeled but treated as fixed

Interlude: Loss Functions for Classification

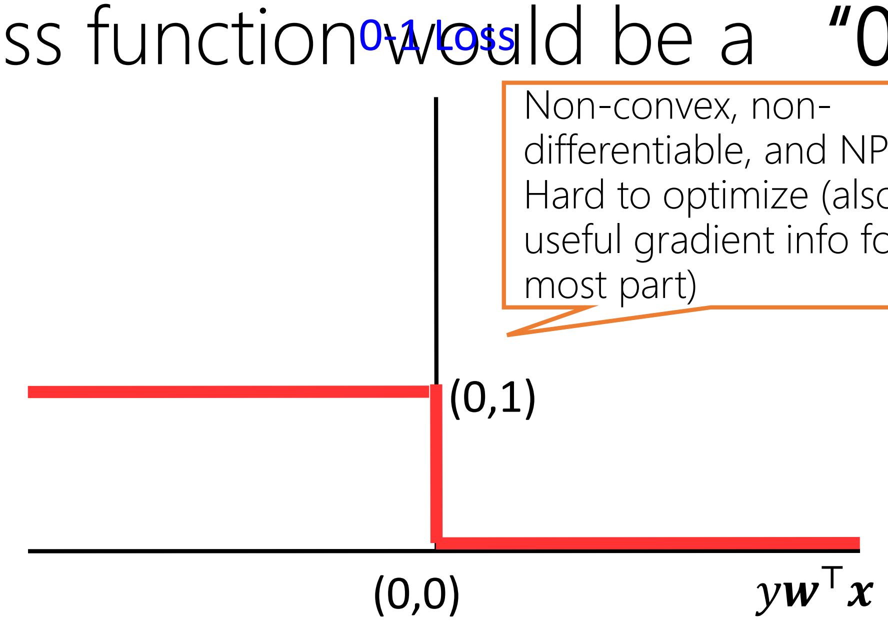
- In regression (assuming linear model $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$), some common loss fn

$$\ell(y, \hat{y}) = (y - \hat{y})^2 \qquad\qquad \ell(y, \hat{y}) = |y - \hat{y}|$$

- These measure the difference between the true output and model's prediction
- What about loss functions for <u>classification</u> where $\hat{y} = \textbf{sign}(\boldsymbol{w}^\top \boldsymbol{x})$?
- Perhaps the most natural classification loss function would be a "0-1 Loss"
  - Loss = 1 if $\hat{y} \neq y$ and Loss = 0 if $\hat{y} = y$.
  - Assuming labels as +1/-1, it means

$$\ell(y, \hat{y}) = \begin{cases} 1 & \text{if } y\boldsymbol{w}^\top \boldsymbol{x} < 0 \\ 0 & \text{if } y\boldsymbol{w}^\top \boldsymbol{x} \geq 0 \end{cases}$$

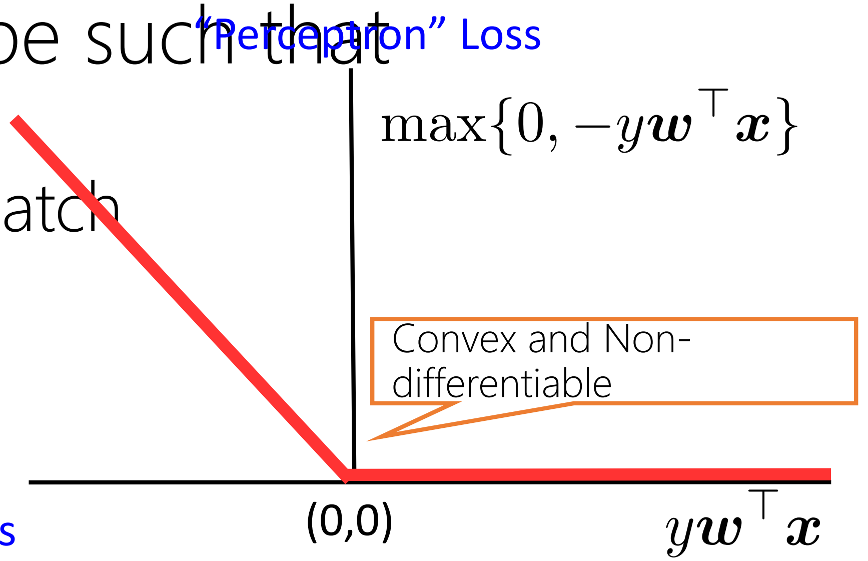
Same as $\mathbb{I}[y\boldsymbol{w}^\top \boldsymbol{x} < 0]$ or $\mathbb{I}[\text{sign}(\boldsymbol{w}^\top \boldsymbol{x}) \neq y]$

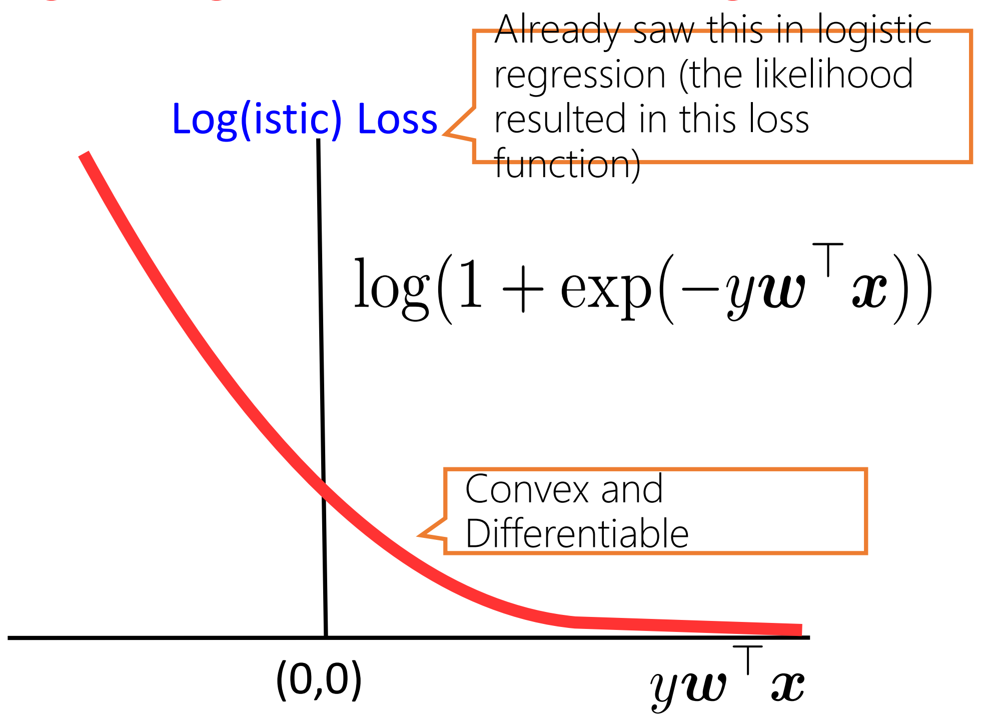


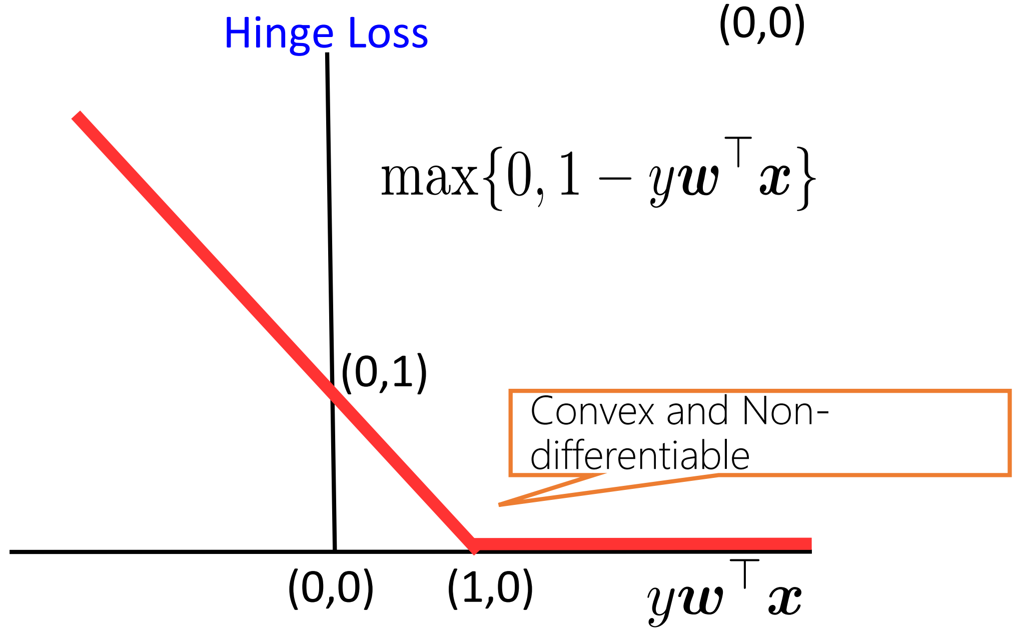
Non-convex, non-differentiable, and NP-Hard to optimize (also no useful gradient info for the most part)

Interlude: Loss Functions for Classification

- An ideal loss function for classification should be such that
  - Loss is small/zero if y and $\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x})$ match
  - Loss is large/non-zero if y and $\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x})$ do not match
  - Large positive $y\boldsymbol{w}^\top\boldsymbol{x} \Rightarrow$ small/zero loss
  - Large negative $y\boldsymbol{w}^\top\boldsymbol{x} \Rightarrow$ large/non-zero loss

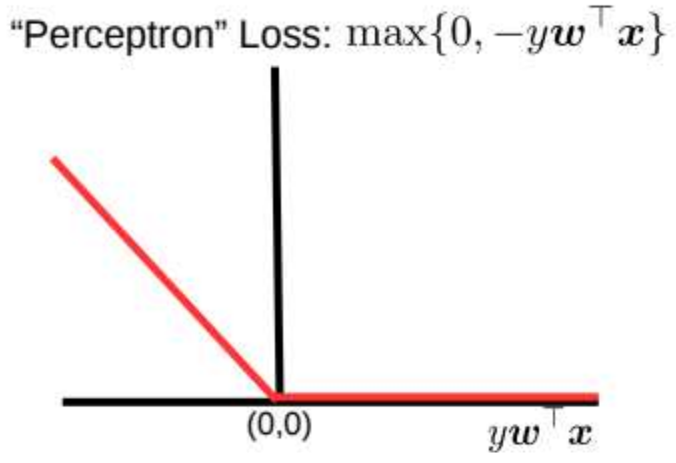






Learning by Optimizing Perceptron Loss

- Let' s ignore the bias term $\boldsymbol{b}$ for now. So the hyperplane is simply $\boldsymbol{w}^\top \boldsymbol{x} = 0$
- The Perceptron loss function: $L(w) = \sum_{n=1}^{N} \max\{0, -y_n \boldsymbol{w}^\top \boldsymbol{x}_n\}$. Let' do SG



Subgradients w.r.t. $\boldsymbol{w}$

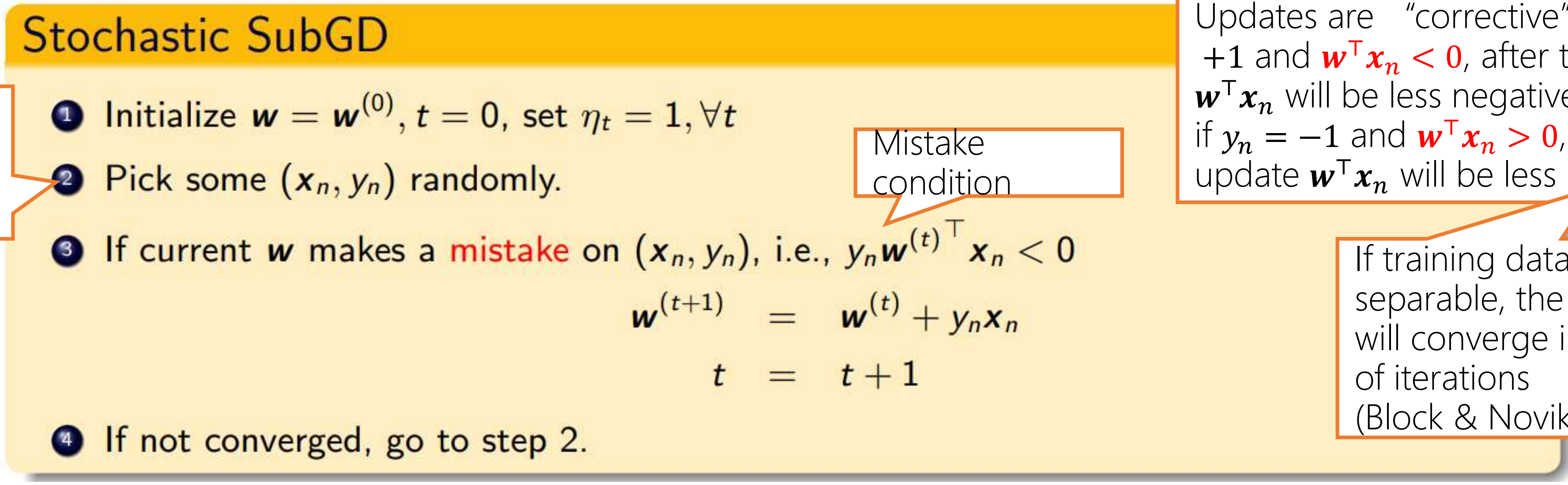
One randomly chosen example in each iteration

$$\boldsymbol{g}_n = \begin{cases} 0, & \text{for } y_n \boldsymbol{w}^\top \boldsymbol{x}_n > 0 \\ -y_n \boldsymbol{x}_n & \text{for } y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0 \\ k y_n \boldsymbol{x}_n & \text{for } y_n \boldsymbol{w}^\top \boldsymbol{x}_n = 0 \quad (\text{where } k \in [-1, 0]) \end{cases}$$

- If we use $k = 0$ then $\boldsymbol{g}_n = 0$ for $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \geq 0$, and $\boldsymbol{g}_n = -y_n \boldsymbol{x}_n$ for $y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0$
- Non-zero gradients only when the model makes a mistake on current

The Perceptron Algorithm

- Stochastic Sub-grad desc on Perceptron loss is also known as the Perceptron algorithm



Stochastic SubGD

```
1. Initialize $\boldsymbol{w} = \boldsymbol{w}^{(0)}$, $t = 0$, set $\eta_t = 1, \forall t$
2. Pick some $(\boldsymbol{x}_n, y_n)$ randomly.
3. If current $\boldsymbol{w}$ makes a mistake on $(\boldsymbol{x}_n, y_n)$, i.e., $y_n \boldsymbol{w}^{(t)\top} \boldsymbol{x}_n < 0$
       $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + y_n \boldsymbol{x}_n$
       $t = t + 1$
4. If not converged, go to step 2.
```

Note: An example may get chosen several times during the entire run

Mistake condition

Updates are "corrective": If $y\_n = +1$ and $\boldsymbol{w}^\top \boldsymbol{x}_n < 0$, after the update $\boldsymbol{w}^\top \boldsymbol{x}_n$ will be less negative. Likewise, if $y_n = -1$ and $\boldsymbol{w}^\top \boldsymbol{x}_n > 0$, after the update $\boldsymbol{w}^\top \boldsymbol{x}_n$ will be less positive

If training data is linearly separable, the Perceptron algo will converge in a finite number of iterations (Block & Novikoff theorem)

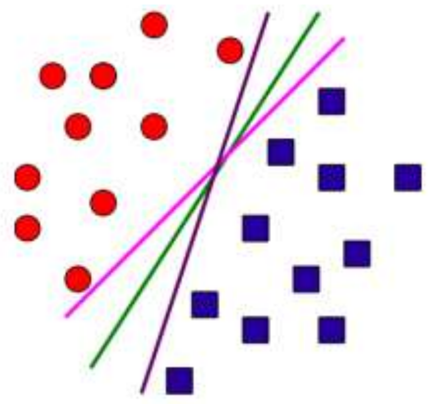
- An example of an online learning algorithm (processes one training ex. at a time)
- Assuming $\boldsymbol{w}^{(0)} = \boldsymbol{0}$, easy to see that the final $\boldsymbol{w}$ has the form $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$

Meaning of α_n may be different

  - α_n is total number of mistakes made by the algorithm on example $(\boldsymbol{x}_n, y_n)$

Perceptron and (lack of) Margins

- Perceptron would learn a hyperplane (of many possible) that separates the classes



Basically, it will learn the hyperplane which corresponds to the $\boldsymbol{w}$ that minimizes the Perceptron loss

Kind of an "unsafe" situation to have – ideally would like it to be reasonably away from closest training examples from either class

- Doesn't guarantee any "margin" around the hyperplane
  - The hyperplane can get arbitrarily close to some training e ide
  - This may not be good for generalization performance

$\gamma > 0$ is some pre-specified margin

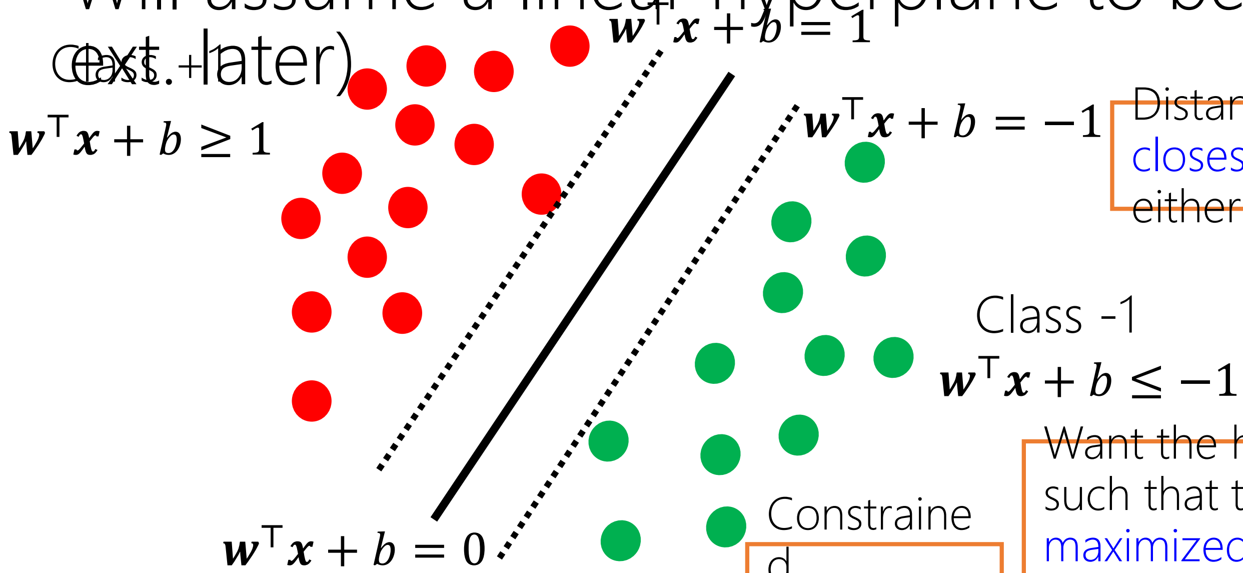
- Can artificially introduce margin by changing the mistake condition to $y_n \boldsymbol{w}^\top \boldsymbol{x}_n < \gamma$
- Support Vector Machine (SVM) does it directly by learning the max. margin

Support Vector Machines

Support Vector Machine (SVM)

SVM originally proposed by Vapnik and colleagues in early 90s

- Hyperplane based classifier. Ensures a large margin around the hyperplane
- Will assume a linear hyperplane to be of the form $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ (nonlinear ext. later)


Class +1
$\boldsymbol{w}^\top \boldsymbol{x} + b \geq 1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = 1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = -1$
Class -1
$\boldsymbol{w}^\top \boldsymbol{x} + b \leq -1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 \quad \text{if} \quad y_n = +1$$

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 \quad \text{if} \quad y_n = -1$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad \forall n$$

Distance from the closest point (on either side)

"Margin" of the hyper[plane]

Distance of an input $\boldsymbol{x}_n$ from the h.p.

$$\gamma = \min_{1 \leq n \leq N} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_n + b|}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||}$$

$$\textbf{Total margin} = \frac{2}{||\boldsymbol{w}||}$$

Want the hyperplane $(\boldsymbol{w}, b)$ such that this margin is maximized (max-margin hyperplane) and

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad \forall n$$

Constrained optimization problem

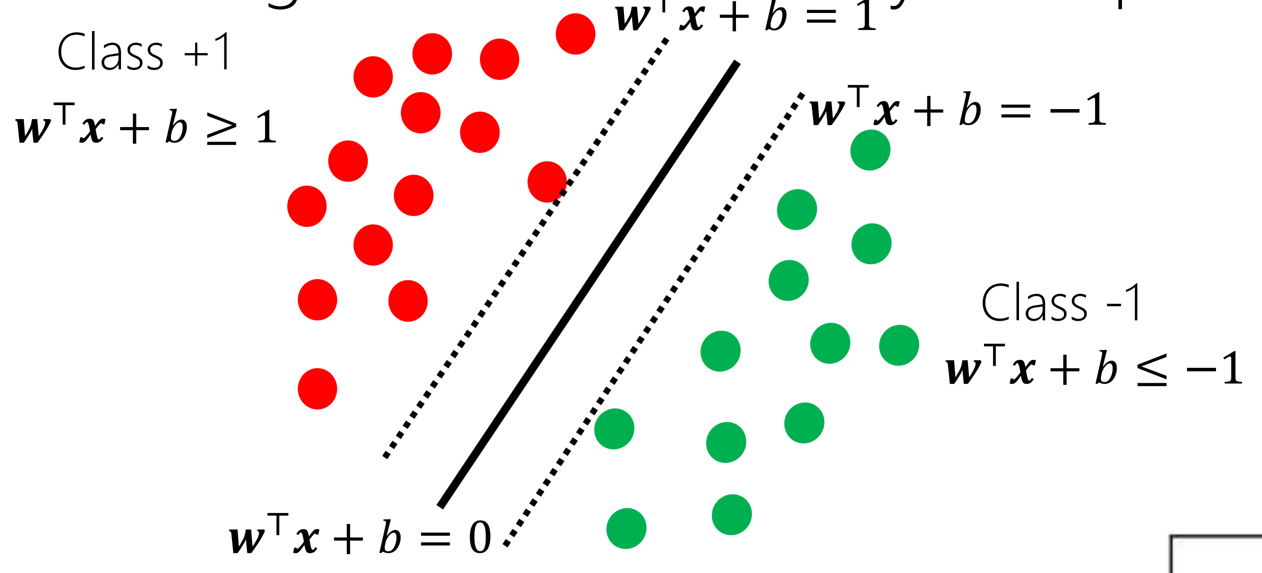
The 1/-1 in supp. h.p. equations is arbitrary; can replace by any scalar m/-m and solution won't change, except a simple scaling of $\boldsymbol{w}$

$w^T x + b = 1$

- Two other "supporting" hyperplanes defining a "no man's land"
  - Ensure that zero training examples fall in this region (will relax later)

Hard-Margin SVM

- Hard-Margin: Every training example must fulfil margin condition $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$
- Meaning: Must not have any example in the no-man's land



Class +1
$\boldsymbol{w}^\top \boldsymbol{x} + b \geq 1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = 1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = -1$
Class -1
$\boldsymbol{w}^\top \boldsymbol{x} + b \leq -1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$

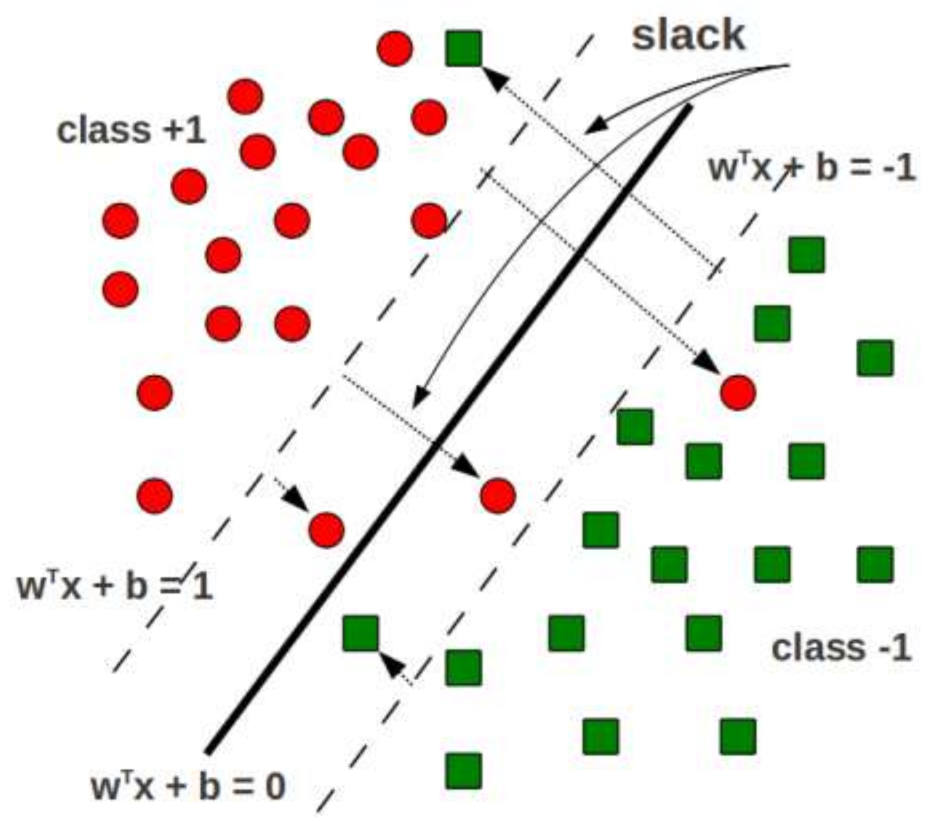
- Also want to maximize margin $2\gamma = \frac{2}{\|\boldsymbol{w}\|}$
- Equivalent to <u>minimizing</u> $\|\boldsymbol{w}\|^2$ or $\frac{\|\boldsymbol{w}\|^2}{2}$
- The objective func. for hard-margin SV

Constrained optimization problem with N inequality constraints. Objective and constraints both are convex

$$\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w}, b) = \frac{||\boldsymbol{w}||^2}{2}$$

$$\text{subject to} \quad y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

Soft-Margin SVM (More Commonly Used)



- Allow some training examples to fall within the no-man' s land (margin region)
- Even okay for some training examples to fall totally on the wrong side of h.p.
- Extent of "violation" by a training input $(\boldsymbol{x}_n, y_n)$ is known as slack $\xi_n \geq 0$
- $\xi_n > 1$ means totally on the wrong side

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 - \xi_n \quad \text{if} \quad y_n = +1$$

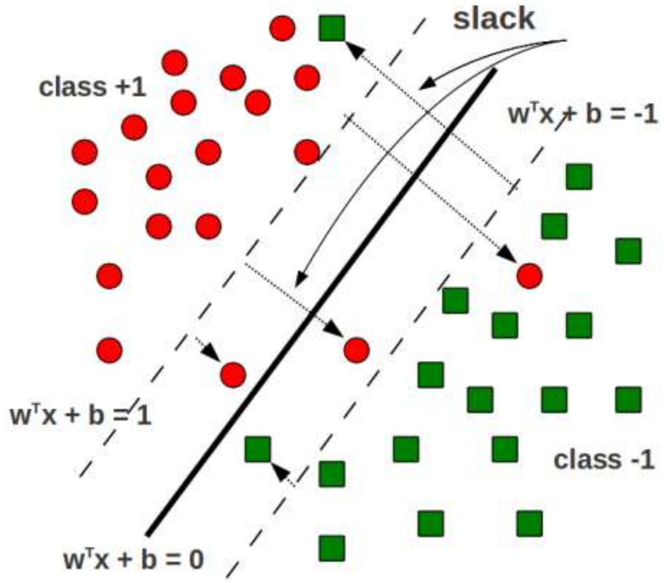
$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 + \xi_n \quad \text{if} \quad y_n = -1$$

Soft-margin constraint $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \quad \forall n$

Soft-Margin SVM (Contd)

- Goal: Still want to maximize the margin such that
  - Soft-margin constraints $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$ are satisfied for all training ex.
  - Do not have too many margin violations (sum of slacks $\sum_{n=1}^{N} \xi_n$ should

Sum of slacks is like the training error



- The objective func. for soft-margin SVM

Inversely prop. to margin

Trade-off hyperparam

training error

Constrained optimization problem with $2N$ inequality constraints. Objective and constraints both are convex

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad f(\boldsymbol{w},b,\boldsymbol{\xi}) = \frac{||\boldsymbol{w}||^2}{2} + C\sum_{n=1}^{N} \xi_n$$

$$\text{subject to} \quad y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N$$

- Hyperparameter C controls the trade off between large margin and small training error (need to tune)
  - Large C: small training error but also small margin (bad)
  - Small C: large margin but large training error (bad)

Solving the SVM Problem

Solving Hard-Margin SVM

- The hard-margin SVM optimization problem is

$$\begin{aligned}&\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w},b) = \frac{||\boldsymbol{w}||^2}{2}\\ &\text{subject to} \quad 1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \le 0, \qquad n = 1,\ldots,N\end{aligned}$$

- A constrained optimization problem. One option is to solve using Lagrange' s method
- Introduce Lagrange multipliers α_n $(n = 1, \ldots, N)$, one for each constraint, and solve

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha}\geq 0} \ \mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha}) = \frac{||\boldsymbol{w}||^2}{2} + \sum_{n=1}^{N} \alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)\}$$

- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$ denotes the vector of Lagrange multipliers
- It is easier (and helpful; we will soon see why) to solve the dual: min and

Solving Hard-Margin SVM

- The dual problem (min then max) is

$$\boxed{\max_{\boldsymbol{\alpha}\geq 0}\ \min_{\boldsymbol{w},b}\ \mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha}) = \frac{\boldsymbol{w}^\top\boldsymbol{w}}{2} + \sum_{n=1}^{N}\alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)\}}$$

Note: if we ignore the bias term b then we don't need to handle the constraint $\sum_{n=1}^{N}\alpha_n y_n = 0$ (problem becomes a bit more easy to solve)

Otherwise, the α_n's are coupled and some opt. techniques such as co-ordinate ascent can't easily be applied

- Take (partial) derivatives of $\mathcal{L}$ w.r.t. $\boldsymbol{w}$ and b and setting them to zero gives (verify)

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} = 0 \Rightarrow \boxed{\boldsymbol{w} = \sum_{n=1}^{N}\alpha_n y_n \boldsymbol{x}_n} \qquad \frac{\partial\mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N}\alpha_n y_n = 0$$

α_n tells us how important training example $(\boldsymbol{x}_n, y_n)$ is

- The solution $\boldsymbol{w}$ is simply a weighted sum of all the training inputs

This is also a "quadratic program" (QP) – a quadratic function of the variables $\boldsymbol{\alpha}$

$$\boxed{\max_{\boldsymbol{\alpha}\geq 0}\ \mathcal{L}_D(\boldsymbol{\alpha}) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n y_m y_n(\boldsymbol{x}_m^T\boldsymbol{x}_n)}$$

Note that inputs appear only as pairwise dot products. This will be useful later on when we make SVM nonlinear using kernel methods

Maximizing a concave function (or minimizing a convex function) s.t. $\boldsymbol{\alpha} \geq \mathbf{0}$ and $\sum_{n=1}^{N}\alpha_n y_n = 0$. Many methods to solve it.

$$\boxed{\max_{\boldsymbol{\alpha}\geq 0}\ \mathcal{L}_D(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{G}\boldsymbol{\alpha}}$$

G is an $N \times N$ p.s.d. matrix, also called the Gram Matrix, $G_{nm} = y_n y_m \boldsymbol{x}_n{}^\top\boldsymbol{x}_m$, and **1** is a vector of all 1s

(Note: For various SVM solvers, can see "Support Vector Machine Solvers" by Bottou and Lin)

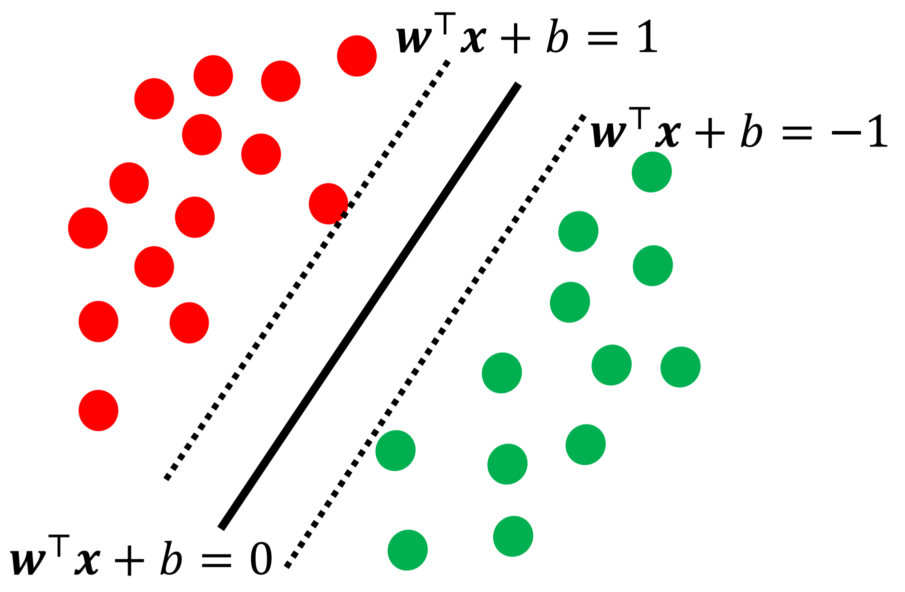
Solving Hard-Margin SVM

- One we have the $\boldsymbol{\alpha_n}$' s by solving the dual, we can get $\boldsymbol{w}$ and $\boldsymbol{b}$ as

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n \quad \text{(we already saw this)}$$

$$b = -\frac{1}{2}\left(\min_{n:y_n=+1} \mathbf{w}^T\mathbf{x}_n + \max_{n:y_n=-1} \mathbf{w}^T\mathbf{x}_n\right) \quad \text{(exercise)}$$

- A nice property: Most $\boldsymbol{\alpha_n}$' s in the solution will be zero (sparse solution)


$\boldsymbol{w}^\top\boldsymbol{x} + b = 1$
$\boldsymbol{w}^\top\boldsymbol{x} + b = -1$
$\boldsymbol{w}^\top\boldsymbol{x} + b = 0$

- Reason: KKT conditions
- For the optimal $\boldsymbol{\alpha_n}$' s, we must have
- Thus $\boldsymbol{\alpha_n}$ nonzero only if $y_n(\boldsymbol{w}^\top\boldsymbol{x}_n + b) = 1$, i.e., the training example lies on the boundary

$$\alpha_n\{1 - y_n(\boldsymbol{w}^\top\boldsymbol{x}_n + b)\} = 0$$

- These examples are called support vectors

Solving Soft-Margin SVM

- Recall the soft-margin SVM optimization problem

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad f(\boldsymbol{w},b,\boldsymbol{\xi}) = \frac{||\boldsymbol{w}||^2}{2} + C\sum_{n=1}^{N}\xi_n \\ &\text{subject to} \quad 1 \le y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) + \xi_n, \quad -\xi_n \le 0 \qquad n = 1,\dots,N \end{aligned}$$

- Here $\boldsymbol{\xi} = [\xi_1, \xi_2, \dots, \xi_N]$ is the vector of slack variables
- Introduce Lagrange multipliers α_n, β_n for each constraint and solve L

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \max_{\alpha \ge 0, \beta \ge 0} \mathcal{L}(\boldsymbol{w},b,\xi,\alpha,\beta) = \frac{||\boldsymbol{w}||^2}{2} + +C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - \xi_n\} - \sum_{n=1}^{N}\beta_n\xi_n$$

- The terms in red color above were not present in the hard-margin SVM
- Two set of dual variables $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_N]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \dots, \beta_N]$
- Will eliminate the primal var $\boldsymbol{w}$, b, $\boldsymbol{\xi}$ to get dual problem containing the dual

Solving Soft-Margin SVM

Note: if we ignore the bias term b then we don't need to handle the constraint $\sum_{n=1}^{N} \alpha_n y_n = 0$ (problem becomes a bit more easy to solve)

Otherwise, the α_n's are coupled and some opt. techniques such as co-ordinate aspect can't easily applied

- The Lagrangian problem to solve

$$\boxed{\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}\geq 0} \mathcal{L}(\boldsymbol{w},b,\xi,\alpha,\beta) = \frac{||\boldsymbol{w}||^2}{2} + +C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - \xi_n\} - \sum_{n=1}^{N}\beta_n\xi_n}$$

- Take (partial) derivatives of $\mathcal{L}$ w.r.t. $\boldsymbol{w}, b$, and ξ_n and setting to zero gives

Weighted sum of training inputs

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} = 0 \Rightarrow \boxed{\boldsymbol{w} = \sum_{n=1}^{N}\alpha_n y_n \boldsymbol{x}_n}, \quad \frac{\partial\mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N}\alpha_n y_n = 0, \quad \frac{\partial\mathcal{L}}{\partial\xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0$, we have $\alpha_n \leq C$ (for hard-margin, $\alpha_n \geq 0$)
- Substituting these in the Lagrangian $\mathcal{L}$ gives the Dual problem

The dual variables β don't appear in the dual problem!

Given $\boldsymbol{\alpha}$, $\boldsymbol{w}$ and b can be found just like the hard-margin SVM case

$$\boxed{\max_{\boldsymbol{\alpha}\leq C,\boldsymbol{\beta}\geq 0} \mathcal{L}_D(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n y_m y_n(\boldsymbol{x}_m^T\boldsymbol{x}_n) \quad \text{s.t.} \quad \sum_{n=1}^{N}\alpha_n y_n = 0}$$

Maximizing a concave function (or minimizing a convex function) s.t. $\boldsymbol{\alpha} \leq \boldsymbol{C}$ and $\sum_{n=1}^{N}\alpha_n y_n = 0$. Many methods to solve it

$$\boxed{\max_{\boldsymbol{\alpha}\leq C} \mathcal{L}_D(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{G}\boldsymbol{\alpha}}$$

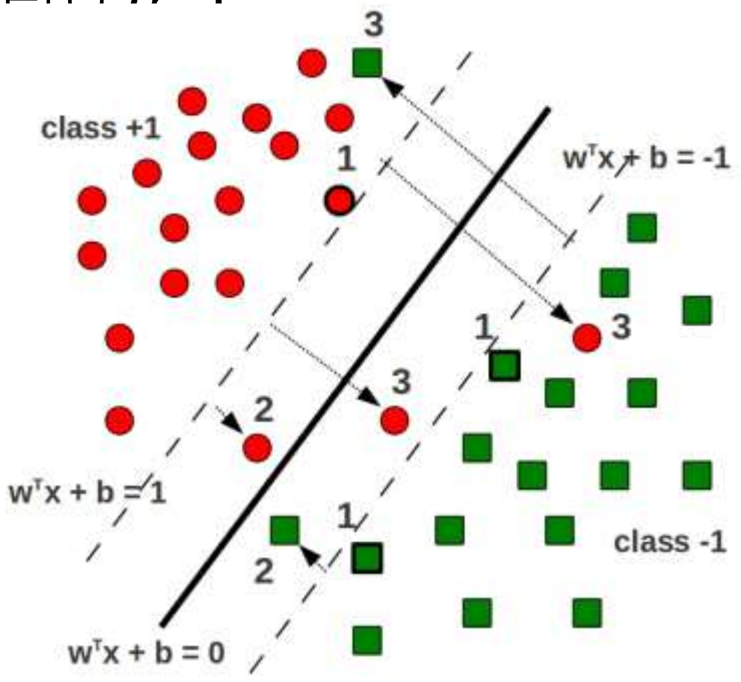
In the solution, $\boldsymbol{\alpha}$ will still be sparse just like the hard-margin SVM case. Nonzero α_n correspond to the support vectors

(Note: For various SVM solvers, can see "Support Vector Machine Solvers" by Bottou and Lin)

Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - All lied on the supporting hyperplanes $\boldsymbol{w}^{\top}\boldsymbol{x}_n + b = 1$ and $\boldsymbol{w}^{\top}\boldsymbol{x}_n + b = -1$

- The soft-margin SVM solution has <u>three</u> types of support vectors (with nonzero α)



1. Lying on the supporting hyperplanes
2. Lying within the margin region but still on the correct side of the hyperplane
3. Lying on the wrong side of the hyperplane (misclassified training examples)

SVMs via Dual Formulation: Some Comments

- Recall the final dual objectives for hard-margin and soft-margin SVM

Hard-Margin SVM: $\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{G}\boldsymbol{\alpha}$

Soft-Margin SVM: $\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{G}\boldsymbol{\alpha}$

Note: Both these ignore the bias term b otherwise will need another constraint $\sum_{n=1}^{N} \alpha_n y_n = 0$

- The dual formulation is nice due to two primary reasons
  - Allows conveniently handling the margin based constraint (via Lagrangians)
  - Allows learning nonlinear separators by replacing inner products in $G_{nm} = y_n y_m \boldsymbol{x}_n{}^\top \boldsymbol{x}_m$ by general kernel-based similarities (more on this when we talk about kernels)
- However, dual formulation can be expensive if N is large (esp. compared to D)
  - Need to solve for N variables $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_N]$
  - Need to pre-compute and store $N \times N$ gram matrix G
- Lot of work on speeding up SVM in these settings (e.g., can use co-ord.

Solving for SVM in the Primal

- Maximizing margin subject to constraints led to the soft-margin formulation of SVM

$$\begin{aligned}&\arg\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{||\boldsymbol{w}||^2}{2} + C\sum_{n=1}^{N}\xi_n \\ &\text{subject to} \quad y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \qquad n = 1, \ldots, N\end{aligned}$$

- Note that slack ξ_n is the same as $\max\{0, 1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)\}$, i.e., hinge loss for $(\boldsymbol{x}_n, y_n)$
- Thus the above i egularized hinge loss

$$\mathcal{L}(\boldsymbol{w}, b) = \sum_{n=1}^{N} \max\{0, 1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)\} + \frac{\lambda}{2}\boldsymbol{w}^\top \boldsymbol{w}$$

- Sum of slacks is like sum of hinge losses, C and λ play similar roles
- Can learn $(\boldsymbol{w}, b)$ directly by minimizing $\mathcal{L}(\boldsymbol{w}, b)$ using (stochastic)(sub)grad

SVM: Summary

- A hugely (perhaps the most!) popular classification algorithm
- Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)
- Some popular ones: libSVM, LIBLINEAR, sklearn also provides SVM
- Lots of work on scaling up SVMs* (both large $\boldsymbol{N}$ and large $\boldsymbol{D}$)
- Extensions beyond binary classification (e.g., multiclass, structured outputs)
- Can even be used for regression problems (Support Vector Regression)
- Nonlinear extensions possible via kernels

* See: “Support Vector Machine Solvers” by Bottou and Lin