Mostly Classification: Review 1

Siva Balakrishnan
Data Mining: 36-462/36-662

March 5th, 2019

Announcements

- Midterm exam is on Thursday, in class, in this room.
- Material up to end of Lecture 12. (Not including linear algebra review.)
- 1 double-sided page of notes allowed. *Handwritten by you.*
- No calculators or computers.
- No programming required. Only:
    - Understanding a snippet of code.
    - Being able to describe an algorithm in english.

Supervised learning

We measure a bunch of covariates $\overset{\text{features}}{\underline{X_1, \dots, X_p}}$, along with a response Y. We want to estimate Y when given new values of $X_1, \dots, X_p$.

{ Prediction focuses on a guess $\widehat{Y} = \boxed{\widehat{f}}(X)$ for Y based on X. } .

Inference (previous/other statistics classes) focuses on questions about the structure of f.

Interpretability focuses on the structure of $\widehat{f}$, rather than f.

Classification

For classification, we get predictors $X = (X_1, \ldots, X_p)$ and outcome Y, importantly Y *is discrete.* We usually notate $Y \in \underline{\{1, \ldots, K\}}$ for a K class problem.

$$\sum_i (y_i - \hat{f}(x_i))^2$$

Our classification rule $\widehat{f}(X)$ then gives values in $\underline{\{1, \ldots, K\}}$.

We measure the quality of a classification with a Loss function $\mathcal{L}(y, \widehat{f}(x))$ (as in regression). However, since $\widehat{f}(x), y \in \{1, \ldots, K\}$, $\mathcal{L}(y, \widehat{f}(x))$ only takes K^2 values, and can be represented by a $K \times K$ matrix.

	Actually cancer	Actually healthy
Predict cancer	0	1
Predict healthy	1	0

Classification

We often talk about 0-1 loss, meaning that all misclassifications have a loss of 1.

$$\boxed{\mathcal{L}(y, \widehat{f}(x)) = \mathbb{I}(y \neq \widehat{f}(x)).}$$

This is equivalent to just counting misclassifications.

Minimizing 0-1 loss gives the ideal classifier (you should understand the steps here...),

$$\widehat{f}(x) = \underset{k=1,\ldots,K}{\operatorname{argmax}} \underbrace{\mathbb{P}(Y = k|X = x)}$$

$$= \underset{k=1,\ldots,K}{\operatorname{argmax}} \underline{\mathbb{P}(X = x|Y = k) \cdot \pi_k},$$

just pick most likely label.

prior probability.

Classification

Minimizing 0-1 loss gives the ideal classifier,

$$\hat{f}(x) = \underset{k=1,\dots,K}{\operatorname{argmax}}\, \mathbb{P}(Y=k|X=x)$$

$$= \underset{k=1,\dots,K}{\operatorname{argmax}}\, \mathbb{P}(X=x|Y=k)\cdot \pi_k,$$

This is called the Bayes classifier, and its error rate is called the Bayes error. In the binary case, the Bayes error is:

$$\mathbb{E}\left[1-\max\right]$$

$$\mathcal{E} = \mathbb{E}[\min\{\mathbb{P}(Y=1|X=x), 1-\mathbb{P}(Y=1|X=x)\}].$$

It is the best error rate that is possible, even if one had complete information about the process. (Similar to irreducible error in regression.)

$$\hat{\mathbb{P}}(y=k) = \frac{\#\, y=k}{n}$$

Most of our classifiers can be thought of as estimating either $\mathbb{P}(Y=k|X=x)$ or $\mathbb{P}(X=x|Y=k)\times\pi_k$ in an attempt to approximate this rule.

$$\mathbb{P}(Y=k)$$

Logistic regression

$$\mathbb{P}(Y=1|X=x) = \frac{\exp(\beta^T x)}{1+\exp(\beta^T x)}.$$

Models $\mathbb{P}(Y = k|X = x)$ directly, rather than $\mathbb{P}(X = x|Y = k)$, π_k. like LDA. Focus on the case where $K = 2$.

log(odds) — linear.

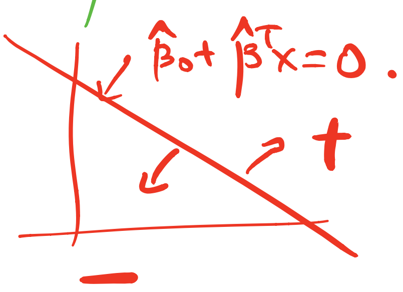
$$\log\left(\frac{\mathbb{P}(Y=1|X=x)}{\mathbb{P}(Y=0|X=x)}\right) = \widehat{\beta}_1 x_1 + \cdots + \widehat{\beta}_p x_p = x^T \widehat{\beta}$$

$e^{\widehat{\beta}_1}$ can be interpreted as the multiplicative change in our model of the odds of $Y = 1$ over $Y = 0$ when x_1 increases by 1.

How do we estimate $\widehat{\beta}$? → Use max likelihood

$$\hat{\beta} = \arg\max_{\beta} \underbrace{\sum_{i=1}^{n} \left[(x_i^T \beta) y_i - \log(1+\beta^T x_i)\right]}_{\mathcal{LL}(\beta)}$$

Logistic regression



The actual classifications are given by checking

$$\underbrace{\hat\beta_0 + x^T\hat\beta > 0}_{\text{label } +1} \text{ vs. } \underbrace{\hat\beta_0 + x^T\hat\beta \leq 0}_{\text{label } 0.}$$

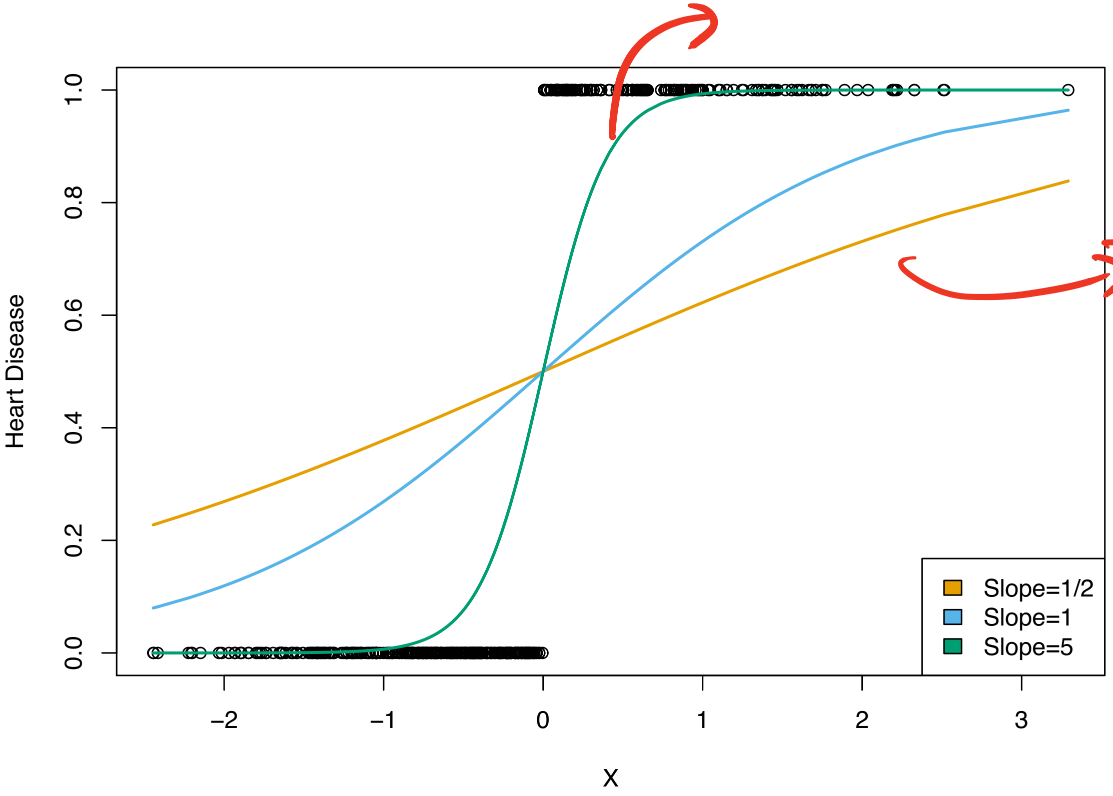
or equivalently

do what's Bayes classifier would.

$$\left\{ \underline{\mathbb{P}(Y=1|X=x) > 0.5} \text{ vs. } \underline{\mathbb{P}(Y=1|X=x) \leq 0.5} \right.$$

Gives a linear decision boundary (like LDA) but is less dependent on assumptions.

- LDA will do better if the Gaussian model is reasonable. It uses information from all the data.
- Logistic will do better if the LDA assumptions are badly wrong. It focuses on data near the boundary.

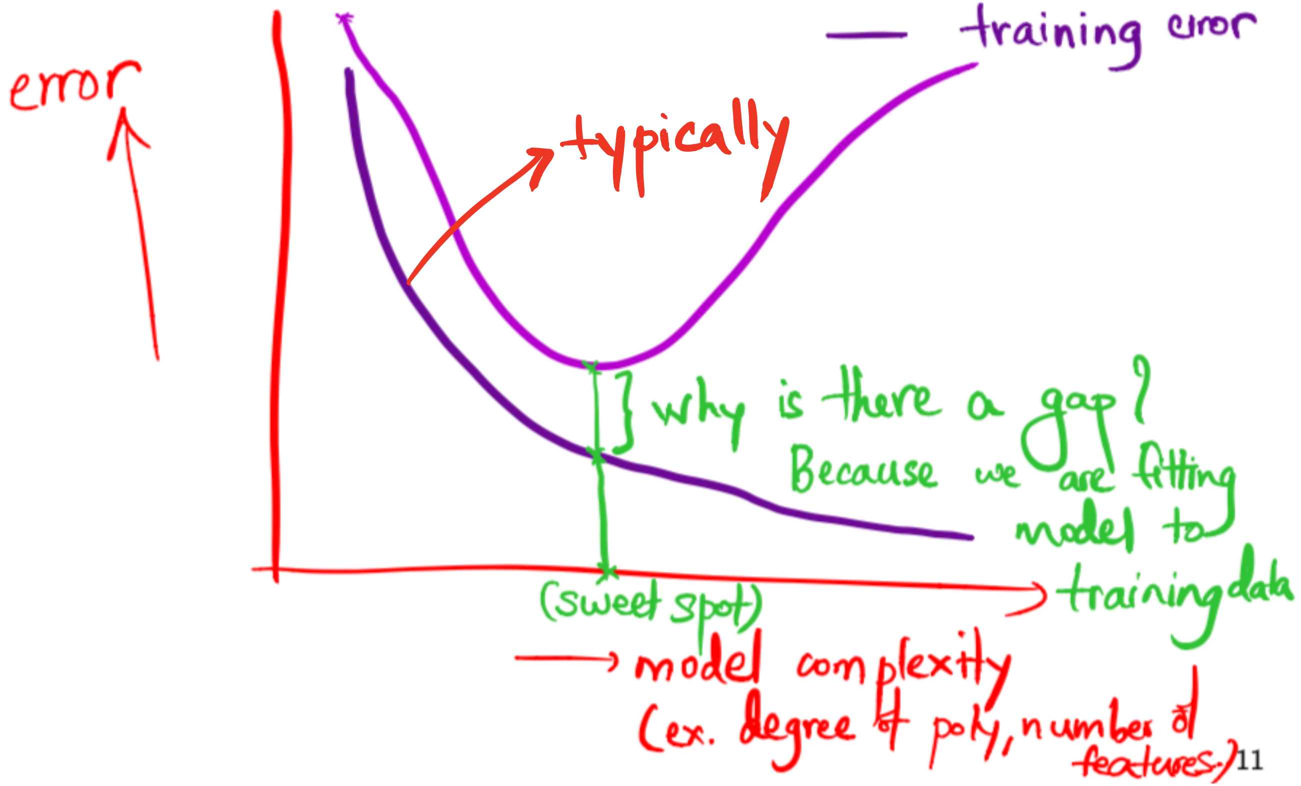


there is a perfect linear classifier

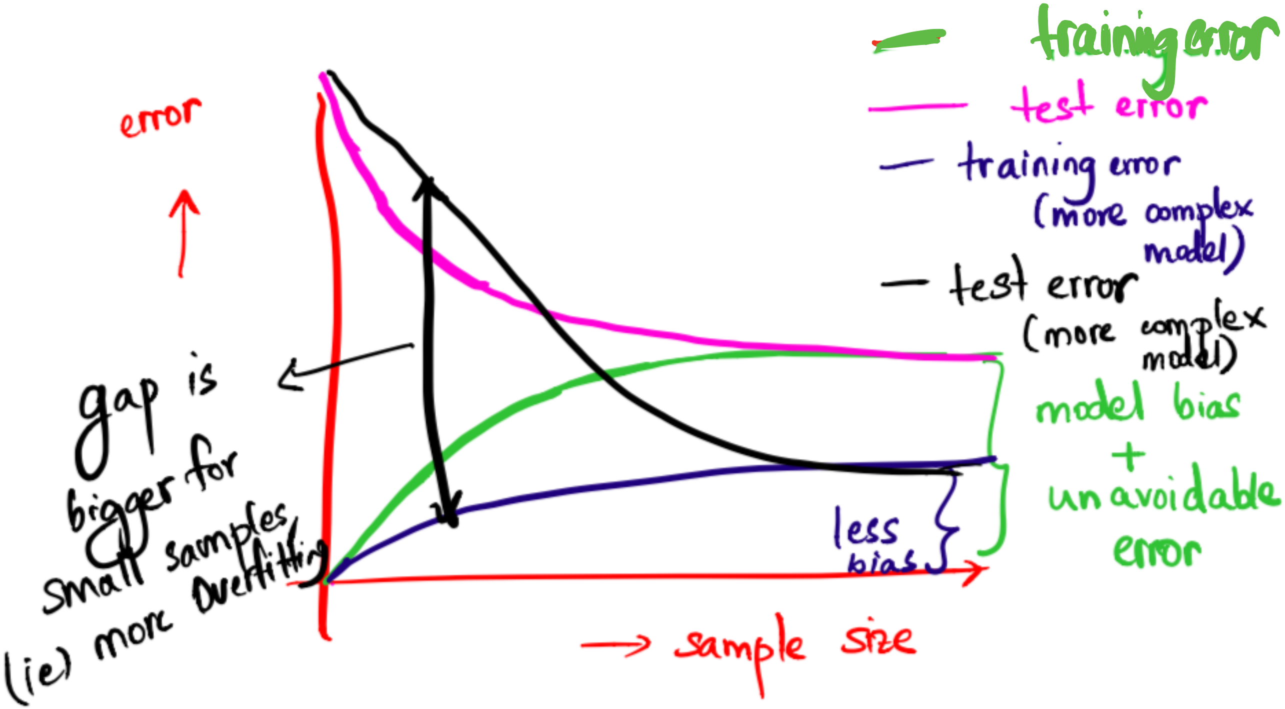
Perfectly separated groups make slopes grow toward infinity! This will make R give errors. Our classifications will be overly-confident. In general, good idea to regularize.

Model Complexity Heuristics

There are two pictures to remember:



Model Complexity Heuristics Contd.



Regularization gives us one way to control model complexity in models with coefficients.

Regularized logistic regression

We can regularize logistic regression (in the same we would linear regression) by penalizing large coefficients. Penalizing coefficients will typically increase bias (the coefficients are biased towards 0) but can potentially reduce variance much more (at least for a well-chosen penalty parameter).

→ high λ increases bias & decreases variance

- Ridge penalty: Reduce variance by shrinking.

$$\hat{\beta} = \arg\max_{\beta} \ell\ell(\beta) - \lambda \underline{\sum_{j=1}^{p} \beta_j^2}$$

- Lasso penalty: Reduce variance by shrinking, and also obtain sparse $\widehat{\beta}$.

$$\hat{\beta} = \arg\max_{\beta} \ell\ell(\beta) - \lambda \sum_{j=1}^{p} |\beta_j|.$$

Linear Discriminant Analysis (LDA)

We model $\mathbb{P}(X = x|Y = k)$ by a multivariate Gaussian with shared covariance between classes. → $N(\mu_k, \Sigma)$.

Leads to decision rule:

$$\widehat{f}^{\text{LDA}}(x) = \underset{j=1,\dots K}{\text{argmax}} \; \widehat{\delta}_j(x)$$

Binary

$\mathbb{P}(X=x|Y=1)\pi_1$

$> \mathbb{P}(X=x|Y=0) \times \pi_0.$

where

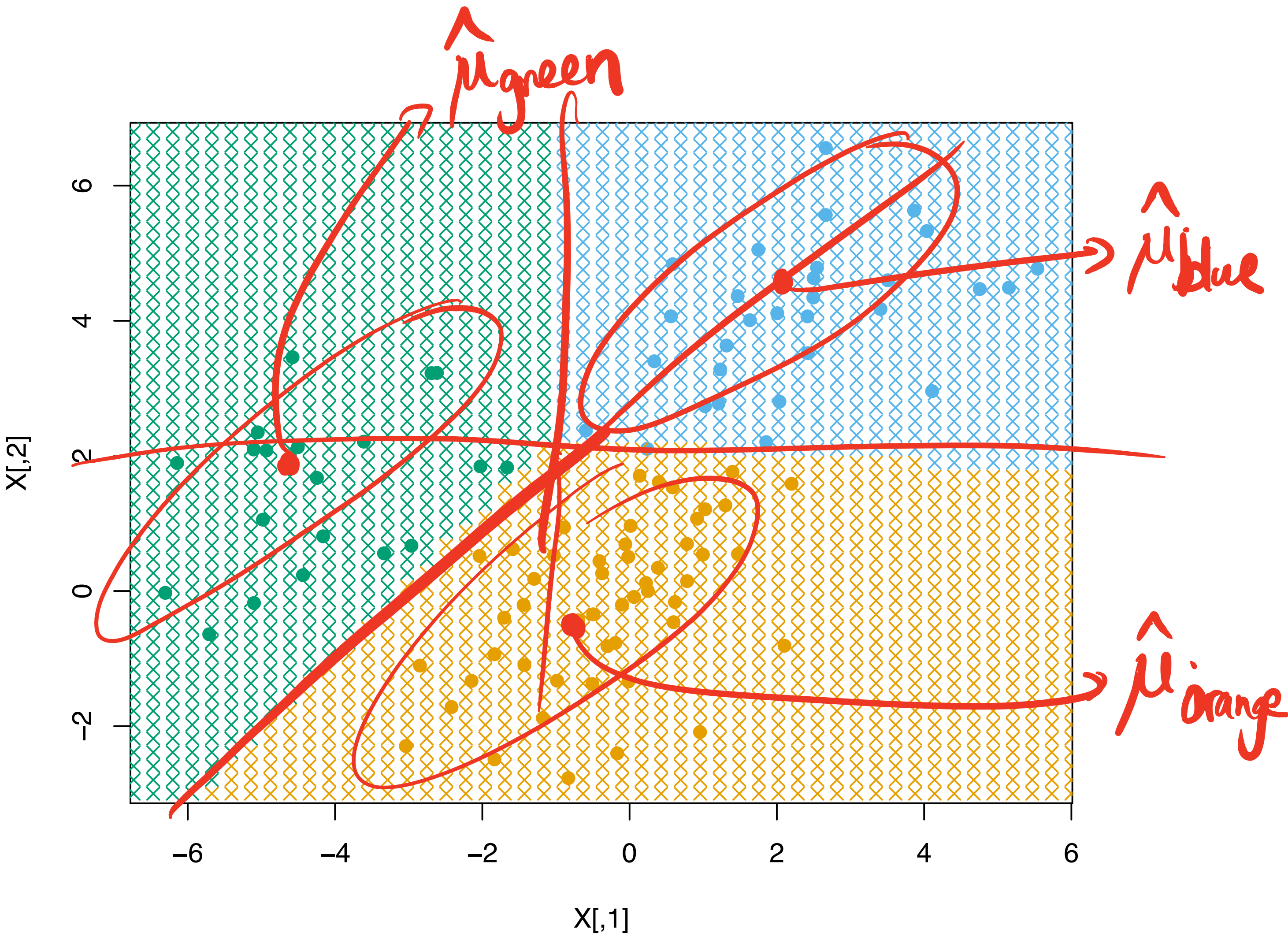
$$\delta_j(x) = x^T \Sigma^{-1} \mu_j - \frac{1}{2}\mu_j^T \Sigma^{-1} \mu_j + \log \pi_j$$

$x^T \Sigma^{-1} x.$

↪ $\delta_1(x) > \delta_0(x).$

and

- $\widehat{\pi}_j = n_j/n$, the proportion of observations in class j
- $\widehat{\mu}_j = \frac{1}{n_j}\sum_{y_i=j} x_i$, the centroid of class j
- $\widehat{\Sigma} = \frac{1}{n-K}\sum_{j=1}^K \sum_{y_i=j}(x_i - \widehat{\mu}_j)(x_i - \widehat{\mu}_j)^T$, the pooled sample covariance matrix


$\hat{\mu}_{green}$
$\hat{\mu}_{blue}$
$\hat{\mu}_{orange}$
X[,2]
X[,1]
−6
−4
−2
0
2
4
6
−2
0
2
4
6

LDA

Performs well on many data sets, particularly those where the classes can be thought of as "clumps."

Rule also provides $K-1$ dimensional transformation that separates the classes well and can still represent $\widehat{f}^{\mathrm{LDA}}(x)$ perfectly.

Extensions:

- Quadratic discriminant analysis: using the same normal model, we now allow each class j to have its own covariance matrix Σ_j. This leads to quadratic decision boundaries. (More variance, less bias)

Σ ↓ only est!

- Diagonal Covariance LDA: We assume that the covariance matrix is diagonal and only estimate its diagonal entries. Similar to Naïve Bayes assumption.

Σ_{ii} ⟷ features are independent.

Naive Bayes

Naive Bayes takes an alternative approach to modeling $P(X = x|Y = k)$.

Naive Bayes models the feature distributions as independent within each class.

Naive Bayes assumption
cond. on Y features
are ind.

$$
\begin{aligned}
f_k(x) &= \underline{P(X = x|Y = k)} \\
&= \underline{P(X_1 = x_1|Y = k)P(X_2 = x_2|Y = k)\cdots P(X_p = x_p|Y} = k) \\
&= \prod_{j=1}^{p} \underline{P(X_j = x_j|Y = k)}
\end{aligned}
$$

→ just model univariate
densities.

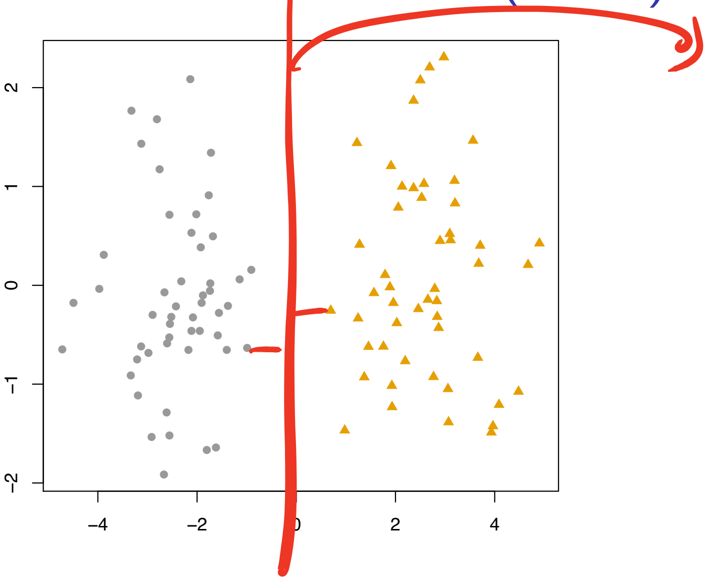
- Individual features can be modeled flexibly.
- Works well when we have many features to model.
- Cannot capture dependence within class (unlike LDA).

$$\mathbb{P}(X=x \mid Y=1) \sim N\left(\mu_1, \begin{bmatrix} \sigma_{11} & & 0 \\ & \sigma_{22} & \\ 0 & & \ddots \end{bmatrix}\right)$$

$$\mathbb{P}(X_1 \mid Y=1) \sim N(\mu_{11}, \sigma_{11}^2)$$

$$\mathbb{P}(X_2 \mid Y=1) \sim N(\mu_{22}, \sigma_{22}^2).$$

Support vector machine (SVM)



large "margin"

Tries to find good separating lines between groups of points, in a way that minimizes incursions across the boundary.

It does this by creating a "soft margin" around the boundary. Points have a budget for allowed crossings, which is tuned by cross-validation.

Hard-Margin SVM

$\beta_0 + \beta^T x \geq 0.$

$\to \alpha\beta_0 + (\alpha\beta)^T x \geq 0.$

↓

same line.

Maximize M

subject to $\sum_{j=1}^{p} \beta_j^2 = 1,$

$y_i(\beta_0 + x_i^T\beta) \geq M.$

If I give you z.

$\underbrace{|\beta_0 + \beta^T z|}$

↓

dist of z to line.

↓

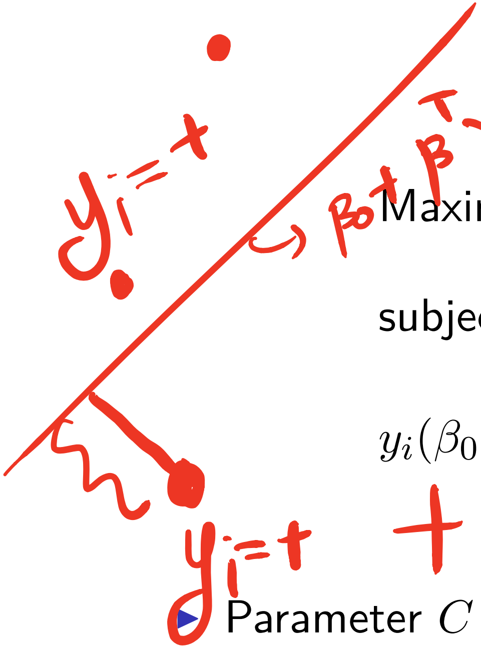
only true if $\sum \beta_j^2 = 1$

If x_i has label $y_i = +1$.

if label is -1 then

$\beta_0 + \beta^T x_i < 0.$

Soft-Margin SVM



Maximize M

subject to $\displaystyle\sum_{j=1}^{p} \beta_j^2 = 1, \boxed{\varepsilon_i} \ge 0, \sum_{i=1}^{n} \varepsilon_i \le \boxed{C}$

$$y_i(\beta_0 + x_i^T \beta) \ge \underline{M(1 - \varepsilon_i)}$$

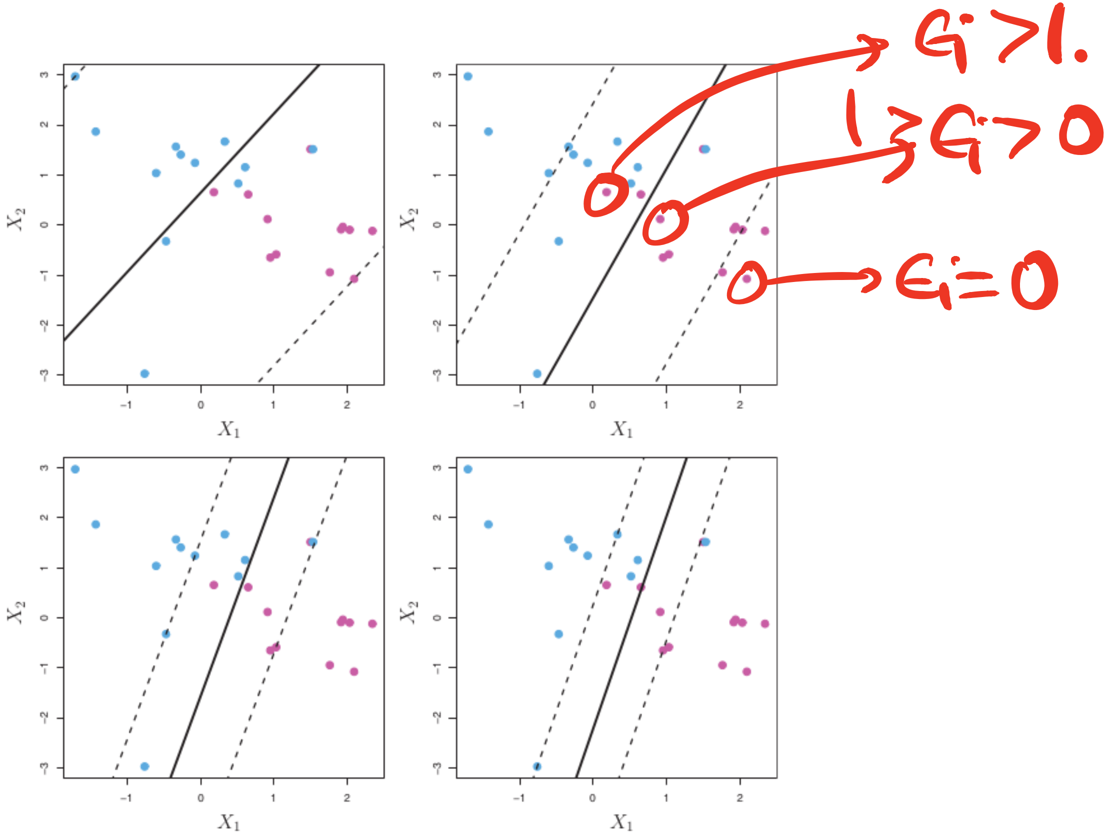
introduce slack

tuning param

$\varepsilon_i > 0$ then x_i pt can violate margin.

- Parameter C determines "softness" of the margin. Small C makes it harder to cross. In particular, no more than C observations cross because. . .

- Variable ε_i encodes point location: $\varepsilon_i = 0$ outside margin, $\varepsilon_i > 0$ inside margin, $\varepsilon_i > 1$ across boundary.



Tuning the budget shifts between bias (ignoring small features at the boundary) and variance (letting small features at the boundary dramatically shift the line).

Kernels

- SVMs give us a way to obtain a linear classifier with a large margin. Suppose we want a non-linear classifier.
- The usual answer is to use feature expansions, i.e. we take our features and concatenate new features which are combinations of existing features.

$$\Phi((\mathrm{balance}, \mathrm{income})) = (\underline{\mathrm{b}}, \ \underline{\mathrm{i}}, \ \underline{\mathrm{b} \times \mathrm{i}}, \underline{\mathrm{b}^2}, \underline{\mathrm{i}^2}).$$

- A linear classifier in the expanded feature space is a non-linear classifier in the original space.
- Can be computationally very annoying – we have to create, store and manipulate these much (much) larger feature vectors.

Kernels

- If the optimal hyper-plane was a linear combination of our data-points (it always is):

$$\widehat{\beta} = \sum_{i=1}^{n} \alpha_i x_i,$$

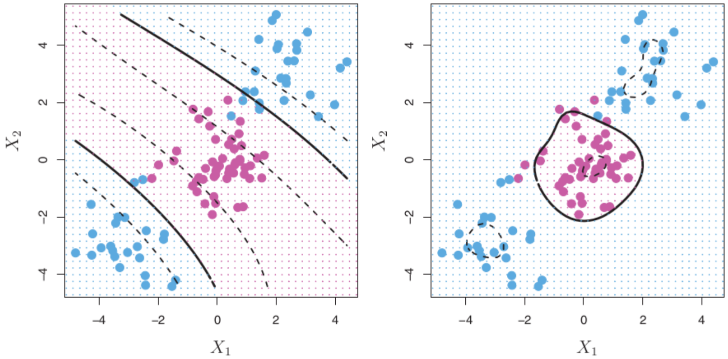
  then SVMs could be written only in terms of inner products $x_i^T x_j$ for the training data (and of course the labels).
- To obtain the SVM classifier (after feature expansion) we do not need to store the big feature vectors, we just need to be able to compute their inner products quickly, i.e. we need some way of computing $\underbrace{\Phi(x)^T \Phi(x')}_{\hookrightarrow\ \underline{K(x,x')}}$ for pairs of training examples.

Kernels

- For many interesting, non-linear kernels, we can compute $\Phi(x)^T\Phi(x')$ very easily using a kernel function:
$$K(x,x') = \Phi(x)^T\Phi(x').$$
- For example, suppose our original data is 2 dimensional, if we choose a quadratic feature map (so we can learn quadratic decision boundaries):
$$\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2).$$
Instead of computing $\Phi(x)^T\Phi(x')$ by this feature expansion we can see that this just corresponds to:
$$K(x,x') = (1 + x^Tx')^2.$$
- For higher-order polynomials we use the polynomial kernel:
$$K(x,x') = (1 + x^Tx')^p.$$
Another popular kernel is the Radial Basis Function kernel:
$$K(x,x') = \exp(-\gamma\|x - x'\|_2^2).$$

Kernels Main Points

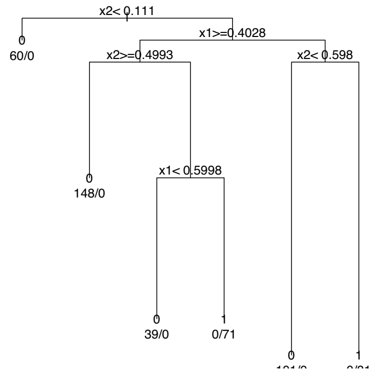
- We can make linear classifiers non-linear by feature expansion.
- Many classifiers only need inner products between the training examples.
- We can often compute inner-products between the feature expanded training examples *directly* using kernels.
- This gives us a way to quickly "non-linearize" (kernelize) classifiers without having to carefully craft feature expansions.

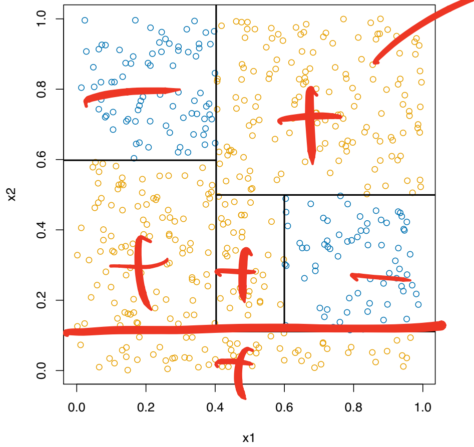


Trees

Again, we wish to predict $y \in \{1, \ldots, K\}$ from $x \in \mathbb{R}^p$.
Classification trees divide the feature space $\mathbb{R}^p$ into rectangles R_j.

For each R_j, we compute the proportion of points from each class, and a single estimate c_j corresponding to the most popular class.





if regression used average

$$\min_{j} \min_{\text{split loc.}} n_1 \, \text{error}(R_1) + \text{error}(R_2) n_2$$

CART

The trees give us a recipe for mapping points to the corresponding rectangle/estimate. We just progress down the tree.

To fit the tree, we split greedily at each step, ignoring the future. We grow the tree very large, and then prune back to a reasonable size with cross-validation.

How did we decide where to split? Purity...

- Very interpretable and easy to explain
- Flexible and adapt to a wide variety of signal shapes
- Invariant to variable transformations
- Able to capture interactions
- Unfortunately, high variance and thus somewhat poor predictors

Ensemble Classifiers

- Construct many different classifiers $\{\underline{\widehat{f}_1}, \ldots, \underline{\widehat{f}_b}\}$. Combine their predictions by taking a (weighted) majority vote:

$$\widehat{f}(x) = \text{sign}(\underline{\sum_{t=1}^{b} \boxed{\alpha_t} \boxed{\widehat{f}_t(x)}}),$$

for some weights α_t.

- Why do we call this voting?

Boosting Key Ideas

- Want to force each classifier to bring something new (and useful) to the ensemble.

- How much should we value each new classifier?

The AdaBoost Algorithm

Initialize: Weights $w_i = 1/n$

For $b = 1, \ldots, B$**:**

1. Fit classification tree $\underline{\widehat{f}^b}$ to the training data with weights $w_1, \ldots, w_n$.
2. Compute weighted misclassification error:

$$e_b = \frac{\sum_{i=1}^n w_i \mathbb{I}\{y_i \neq \widehat{f}^b(x_i)\}}{\sum_{i=1}^n w_i}$$

3. Define $\alpha_b = \log \frac{1-e_b}{e_b}$
4. Update the training data weights:

$$w_i \leftarrow \underline{\underline{w_i}} \cdot \exp\left(\boxed{\alpha_b}\underline{\underline{\mathbb{I}\{y_i \neq \widehat{f}^b(x_i)\}}}\right)$$

Result: $\widehat{f}(x) = \text{sign}\left(\sum_{b=1}^B \alpha_b \widehat{f}^b(x)\right)$

Machine Learning – the Loss Minimization Viewpoint

In some sense, given training data a natural strategy is to select a classifier which makes fewest mistakes on the training set, i.e.:

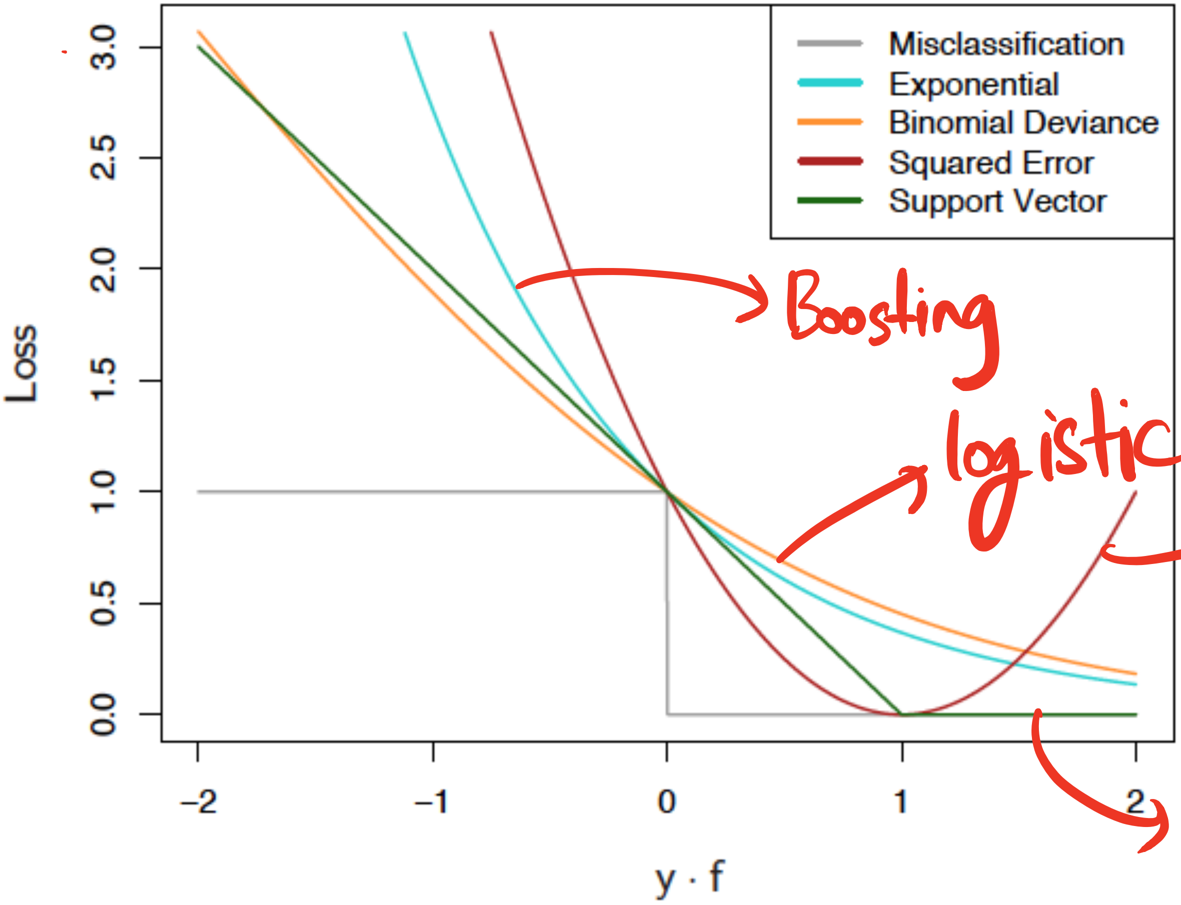
$$\mathcal{L}(\widehat{f}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i \neq \underline{\widehat{f}}(x_i)).$$

Many classification algorithms can be viewed as attempting to minimize an upper bound on the 0/1 loss. This is difficult to do – the 0/1 loss is not "nice" and is computationally difficult to minimize.

One way to deal with this is to minimize a proxy for the 0/1 loss.

This viewpoint unifies many of the methods we have seen so far.

Machine Learning – the Loss Minimization Viewpoint



Boosting

logistic regression

regression

SVMs.

Bagging

Bagging reduces the variance of an estimator. Many bootstrap data sets are generated, and the estimator is fit on each one, yielding many different versions of $\widehat{f}$ (called $\widehat{f}^{*b}$).

When a new data point is observed, every $\widehat{f}^{*b}$ makes a prediction, and those predictions are combined into a lower-variance prediction.

In our best version of this, each bootstrap tree produces a set of class probability estimates. These are averaged across the trees, and we choose the class with the highest average.

We can estimate variable importance by seeing which variables most help their corresponding bootstrap trees. We can estimate test error by looking at the “out of bag” samples — those that are not drawn in a particular bootstrap realization.

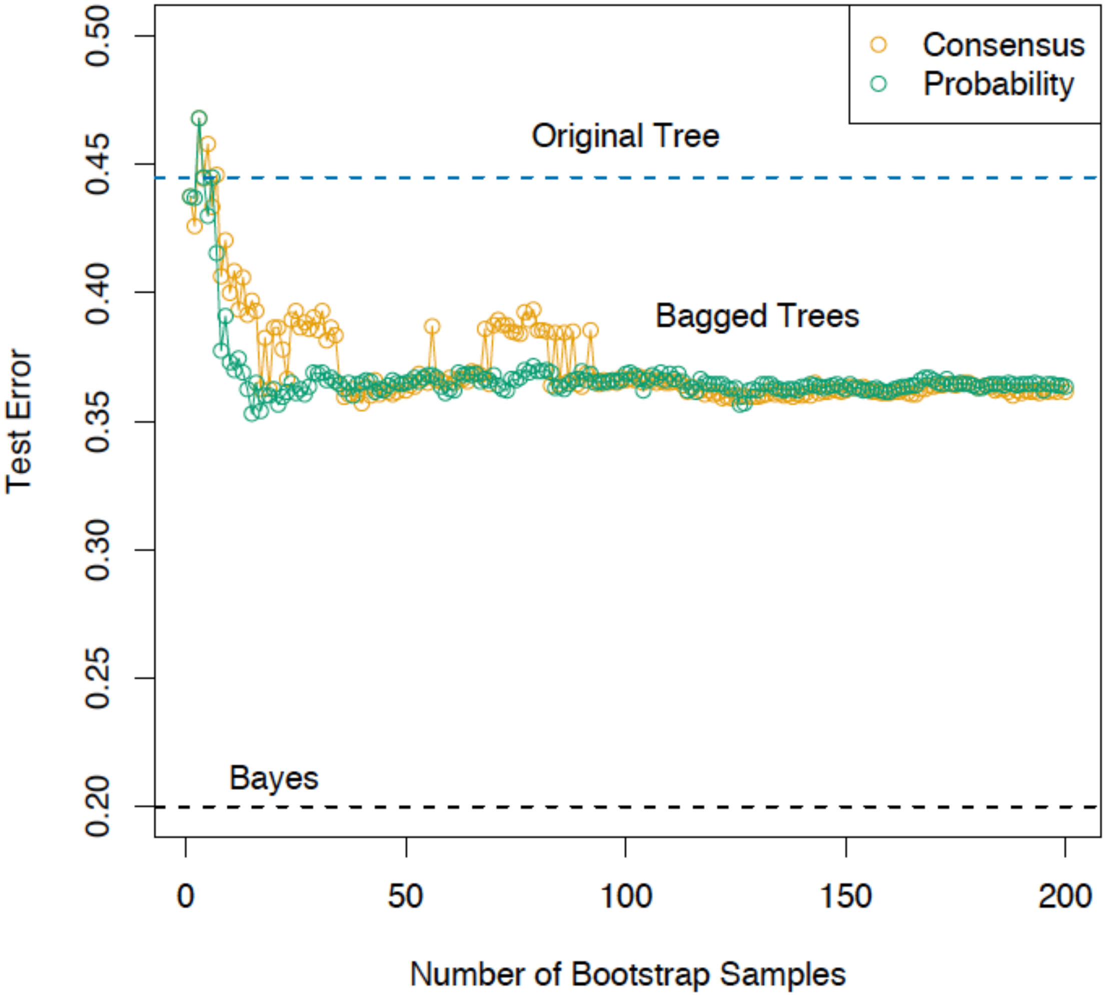
Bagging, a picture

Bagging performance, Random forests

Bagging works well if: the predictors being combined are close to independent.

Random forests are just bagged trees, with one twist. At each split, only splits are only allowed on a small subset of m predictors, rather than all p. m is often $\sqrt{p}$.

This makes the trees more independent, which improves bagging. It also stabilizes the behavior of correlated variables, since they will not all appear together.


Consensus
Probability
Original Tree
Bagged Trees
Bayes
Test Error
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0
50
100
150
200
Number of Bootstrap Samples

Evaluating Classifiers

There are two things we typically can choose to evaluate:

- How good are the labels produced by the classifier?
- How good are the probability estimates produced?

To evaluate labels – missclassification error, and others that can be more useful depending on application if we care about some errors more than others:

- Sensitivity
- Specificity
- Postive Predictive Value
- Negative Predictive Value

We can tune these errors typically by trading them off. For instance, if we really wanted to reduce False Positives we could...

To evaluate probabilities we use calibration plots, or the ROC curve (and the associated AUC measure).

Remember!

Exam on Thursday at the usual class time. Don't be late!

The exam is in this room.

Bring something to write with and your note sheets