Logistic Regression

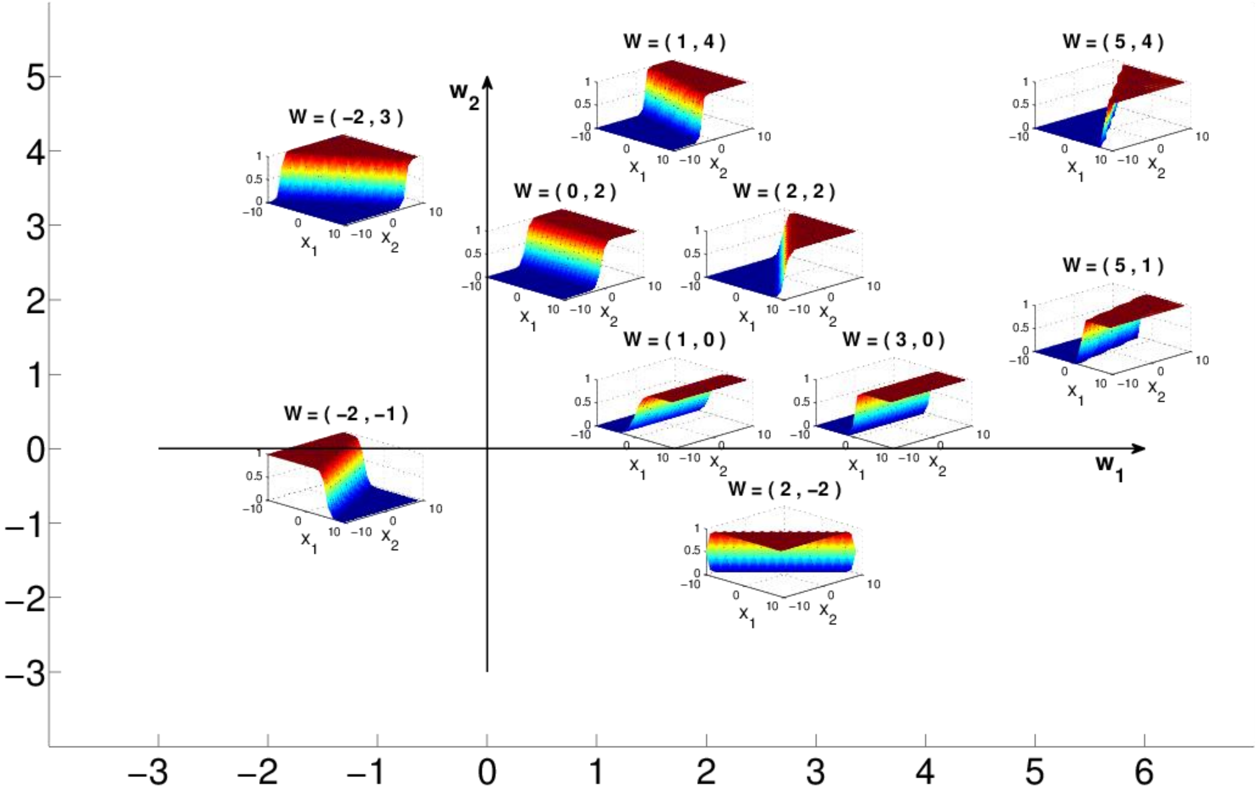
ddebarr@uw.edu

2016-05-26

Agenda

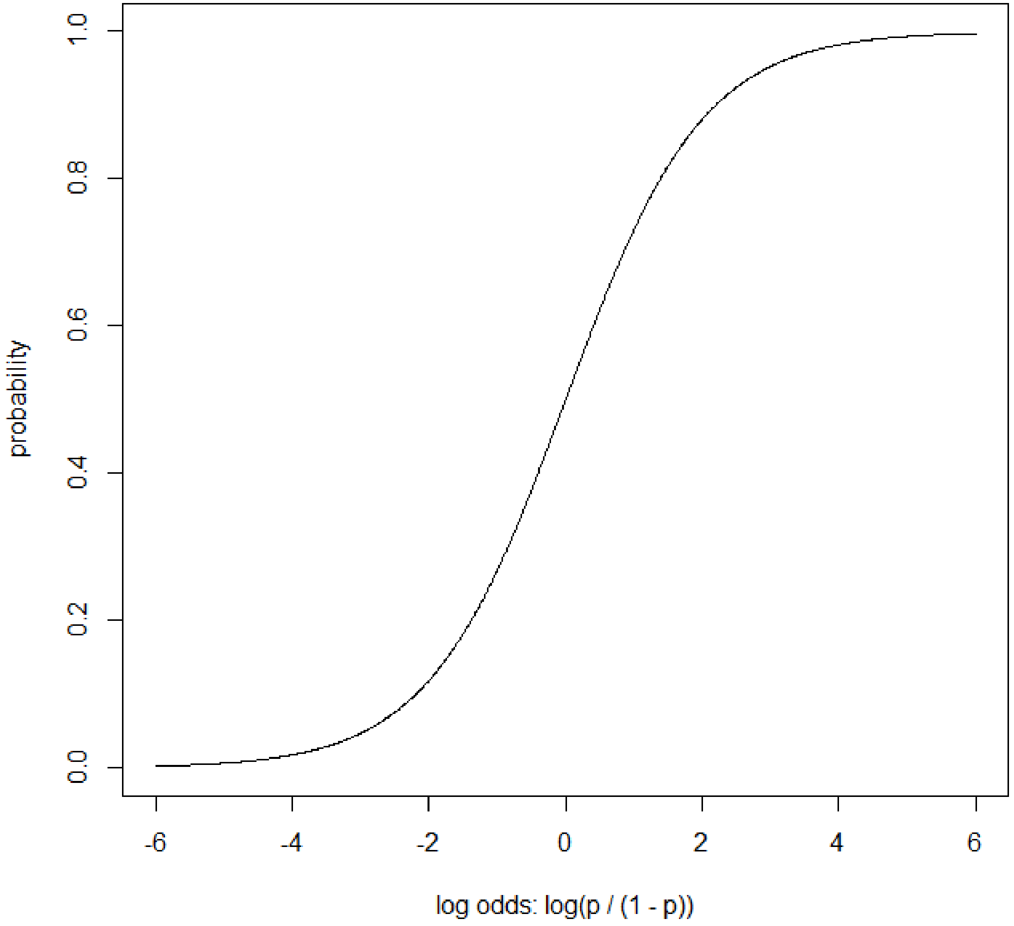
- Model Specification
- Model Fitting
- Bayesian Logistic Regression
- Online Learning and Stochastic Optimization
- Generative versus Discriminative Classifiers

Plots of Sigmoid(w_1*x_1 + w_2*x_2)



$$p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}(y|\mathrm{sigm}(\mathbf{w}^T\mathbf{x}))$$

The Logistic Function



Maps “log odds” to a probability

Logistic Regression Example: The Data

```
> redRate <- 0.2
> grnRate <- 0.6
> bluRate <- 0.8
> set.seed(2^17-1)
> n <- 100000
> red <- as.integer(runif(n) < redRate)
> grn <- as.integer(runif(n) < grnRate)
> blu <- as.integer(runif(n) < bluRate)
> data <- data.frame(y = c(red, grn, blu),
+                    red = c(rep(1, n),
+                            rep(0, n),
+                            rep(0, n)),
+                    grn = c(rep(0, n),
+                            rep(1, n),
+                            rep(0, n)),
+                    blu = c(rep(0, n),
+                            rep(0, n),
+                            rep(1, n)),
+                    redcor = c(sample(c(rep(1, 0.99 * n),
+                                        rep(0, 0.01 * n))),
+                               rep(0, n),
+                               rep(0, n)))
> data <- data[sample(1:nrow(data)),]
```

Logistic Regression Example: Prediction

```
> model1 <- glm(y ~ red + grn, data = data, family = binomial)
> model1$coefficients
(Intercept)         red         grn
  1.3892971  -2.7906595  -0.9839986
> data[1,]
       y red grn blu redcor
164517 1   0   1   0      0
> predict(model1, data[1,], type="response")
 164517
0.59996
> 1 / (1 + exp(-(1.389297 - 2.790660 * data[1,]$red - 0.983999 * data[2,]$grn)))
[1] 0.5999599
```

Logistic Regression Example: The Model

```
> model1 <- glm(y ~ red + grn, data = data, family = binomial)
> summary(model1)

Call:
glm(formula = y ~ red + grn, family = binomial, data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7955  -0.6635   0.6672   1.0108   1.8008

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.389297   0.007913  175.57   <2e-16 ***
red         -2.790660   0.011211 -248.93   <2e-16 ***
grn         -0.983999   0.010212  -96.36   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 414606  on 299999  degrees of freedom
Residual deviance: 333964  on 299997  degrees of freedom
AIC: 333970

Number of Fisher Scoring iterations: 4
```

This model does not have a problem with collinearity

Logistic Regression Example: The Model

```
> model2 <- glm(y ~ red + grn + blu, data = data, family = binomial)
> summary(model2)

Call:
glm(formula = y ~ red + grn + blu, family = binomial, data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7955  -0.6635   0.6672   1.0108   1.8008

Coefficients: (1 not defined because of singularities)
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.389297   0.007913  175.57   <2e-16 ***
red         -2.790660   0.011211 -248.93   <2e-16 ***
grn         -0.983999   0.010212  -96.36   <2e-16 ***
blu                NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 414606  on 299999  degrees of freedom
Residual deviance: 333964  on 299997  degrees of freedom
AIC: 333970

Number of Fisher Scoring iterations: 4
```

This model does not have a problem with collinearity, because our solver recognized the matrix was rank deficient

Logistic Regression Example: Model 2

```
> model3 <- glm(y ~ red + grn + redcor, data = data, family = binomial)
> summary(model3)

Call:
glm(formula = y ~ red + grn + redcor, family = binomial, data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7955  -0.6642   0.6672   1.0108   1.8951

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.389297   0.007913 175.575   <2e-16 ***
red         -3.003543   0.085357 -35.188   <2e-16 ***
grn         -0.983999   0.010212 -96.360   <2e-16 ***
redcor       0.214895   0.085363   2.517   0.0118 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 414606  on 299999  degrees of freedom
Residual deviance: 333958  on 299996  degrees of freedom
AIC: 333966

Number of Fisher Scoring iterations: 4
```

This model has a problem with collinearity: interpretation of the coefficients is problematic

MLE for Logistic Regression

$$\begin{aligned}\mathrm{NLL}(\mathbf{w}) &= -\sum_{i=1}^{N} \log[\mu_i^{\mathbb{I}(y_i=1)} \times (1-\mu_i)^{\mathbb{I}(y_i=0)}] \\ &= -\sum_{i=1}^{N} [y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]\end{aligned}$$

$$NLL(\mathbf{w}) = \sum_{i=1}^{N} \log(1+\exp(-\tilde{y}_i \mathbf{w}^T \mathbf{x}_i))$$

Unfortunately, there is no closed form solution for the MLE!

Gradient and Hessian of the NLL

$$
\begin{aligned}
\mathbf{g} &= \frac{d}{d\mathbf{w}} NLL(\mathbf{w}) = \sum_i (\mu_i - y_i)\mathbf{x}_i = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}) \\
\mathbf{H} &= \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^T = \sum_i (\nabla_{\mathbf{w}} \mu_i)\mathbf{x}_i^T = \sum_i \mu_i(1 - \mu_i)\mathbf{x}_i\mathbf{x}_i^T \\
&= \mathbf{X}^T\mathbf{S}\mathbf{X}
\end{aligned}
$$

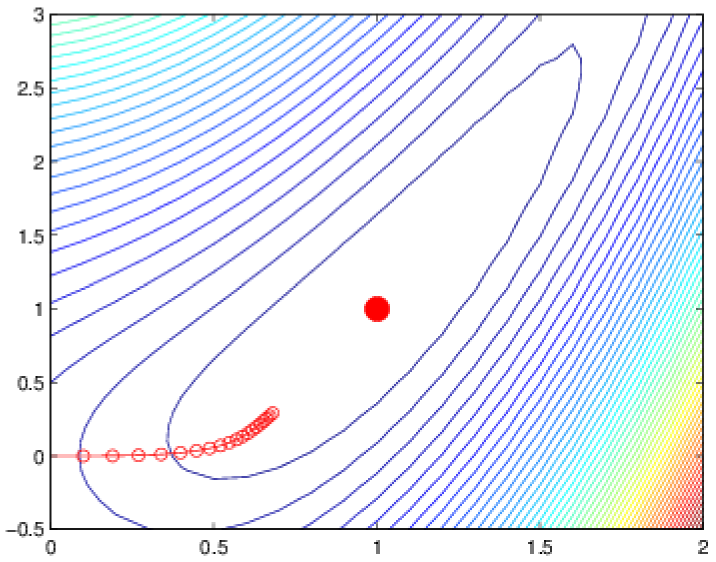
Steepest Descent

- The negative of the gradient is used to update the parameter vector

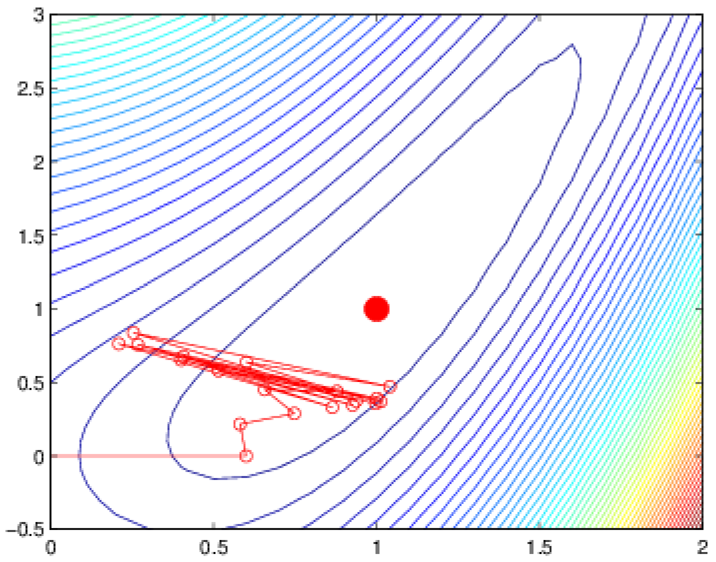
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$$

- Eta is the learning rate, which can be difficult to set
  - Small, constant: takes a long time to converge
  - Large: may not converge at atll

Gradient Descent Example

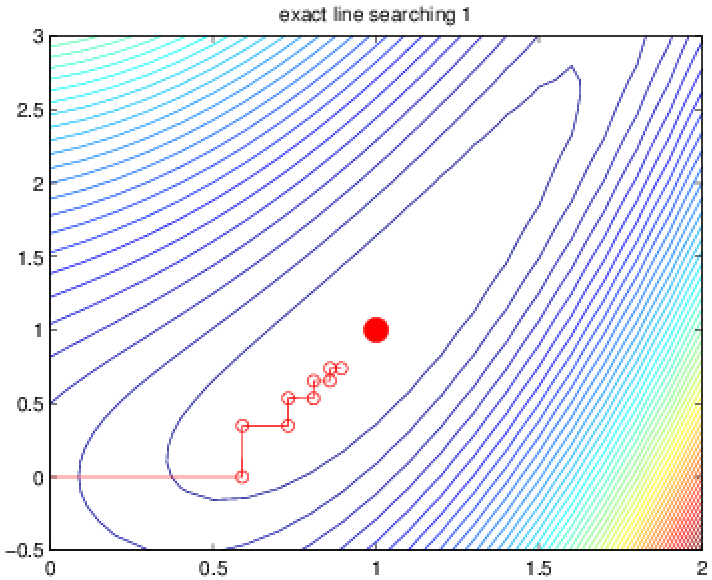


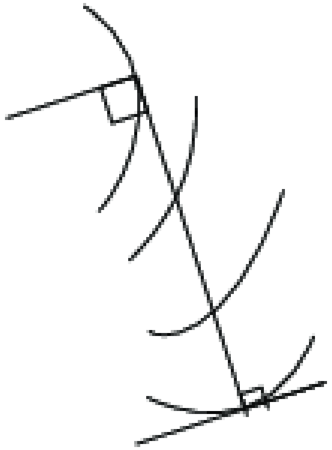
Too small?



Too large!

Steepest Descent Example





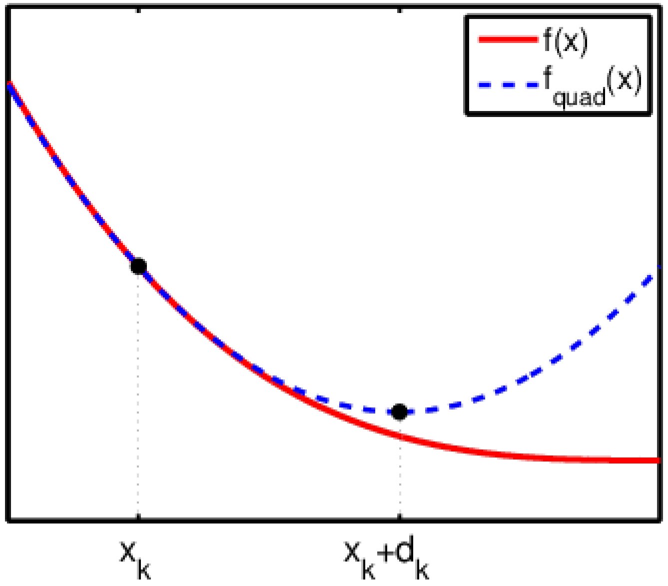
Step size driven by “line search”: the local gradient is perpendicular to the search direction

Newton’s Method

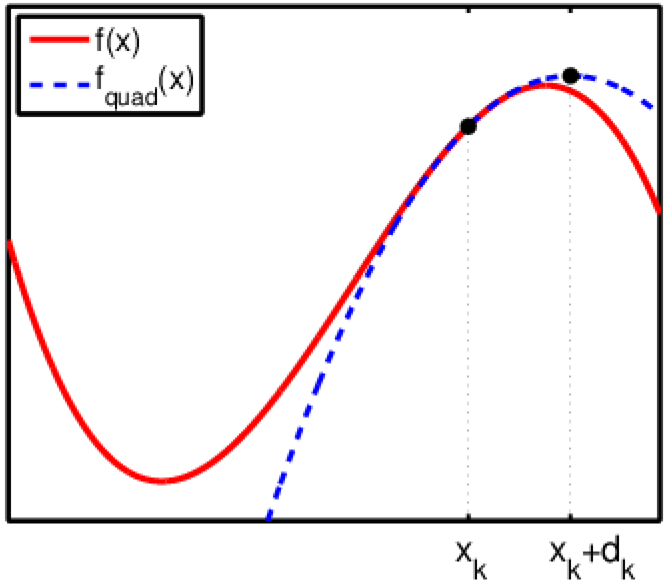
Algorithm 8.1: Newton’s method for minimizing a strictly convex function

Initialize $\boldsymbol{\theta}_0$;
for $k = 1, 2, \ldots$ *until convergence* **do**
  Evaluate $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$;
  Evaluate $\mathbf{H}_k = \nabla^2 f(\boldsymbol{\theta}_k)$;
  Solve $\mathbf{H}_k \mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$;
  Use line search to find stepsize η_k along $\mathbf{d}_k$;
  $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k \mathbf{d}_k$;

Newton’s Method Example



Convex Example



Non-Convex Example

Iteratively Reweighted Least Squares

Algorithm 8.2: Iteratively reweighted least squares (IRLS)

$\mathbf{w} = \mathbf{0}_D$;

$w_0 = \log(\overline{y}/(1-\overline{y}))$;

repeat

$\quad \eta_i = w_0 + \mathbf{w}^T \mathbf{x}_i$;

$\quad \mu_i = \mathrm{sigm}(\eta_i)$;

$\quad s_i = \mu_i(1-\mu_i)$;

$\quad z_i = \eta_i + \frac{y_i - \mu_i}{s_i}$;

$\quad \mathbf{S} = \mathrm{diag}(s_{1:N})$;

$\quad \mathbf{w} = (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}\mathbf{z}$;

until *converged*;

We’re applying Newton’s algorithm to find the MLE

Broyden Fletcher Goldfarb Shanno (BFGS)

- Computing the Hessian matrix explicitly can be expensive
- Rather than computing the Hessian explicitly, we can use an approximation $\mathbf{B}_k \approx \mathbf{H}_k$

$$\mathbf{B}_0 = \mathbf{I}$$

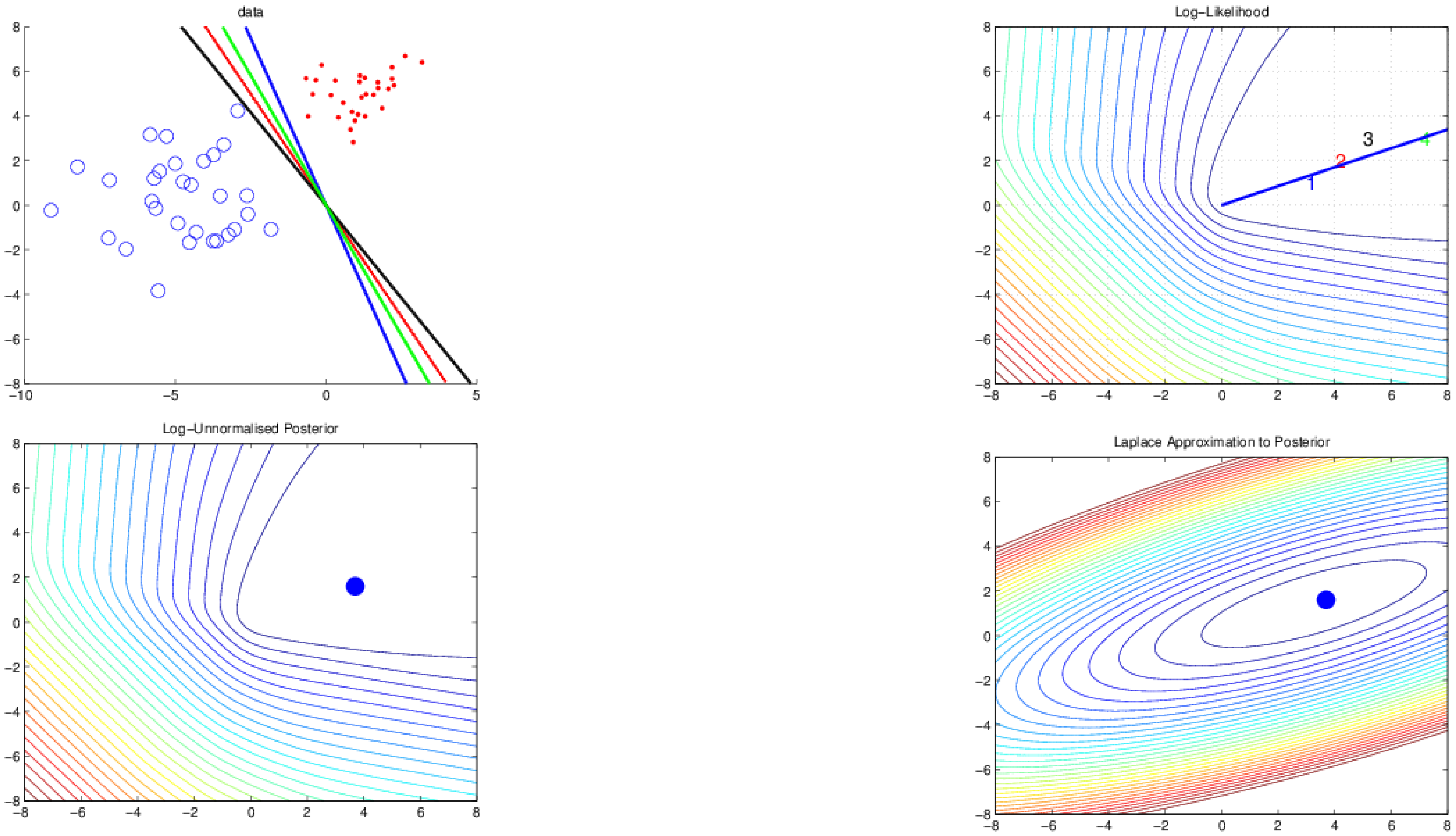
$$\begin{aligned}
\mathbf{B}_{k+1} &= \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{(\mathbf{B}_k \mathbf{s}_k)(\mathbf{B}_k \mathbf{s}_k)^T}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \\
\mathbf{s}_k &= \boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1} \\
\mathbf{y}_k &= \mathbf{g}_k - \mathbf{g}_{k-1}
\end{aligned}$$

We're using a diagonal plus low-rank approximation for the Hessian

Limited Memory BFGS (L-BFGS)

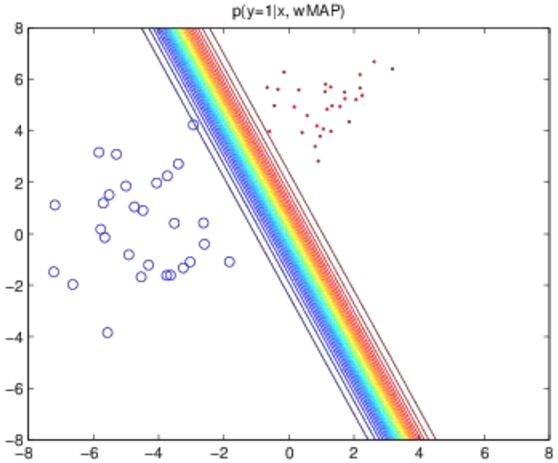
- Storing the Hessian matrix takes $O(D^2)$ space, where D is the number of dimensions
- We can further approximate the Hessian by using only the m most recent $(\boldsymbol{s}_k, \boldsymbol{y}_k)$ pairs, reducing the storage requirement to $O(mD)$

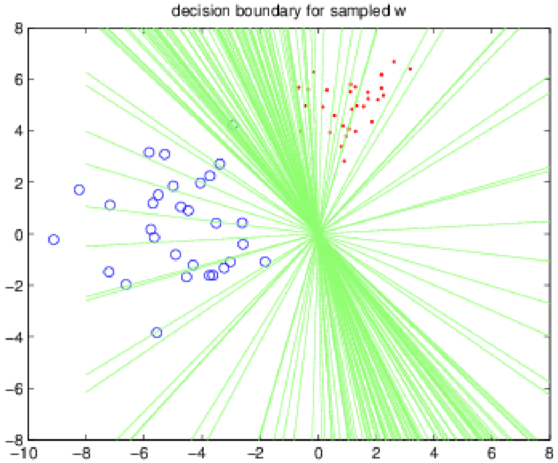
Bayesian Logistic Regression

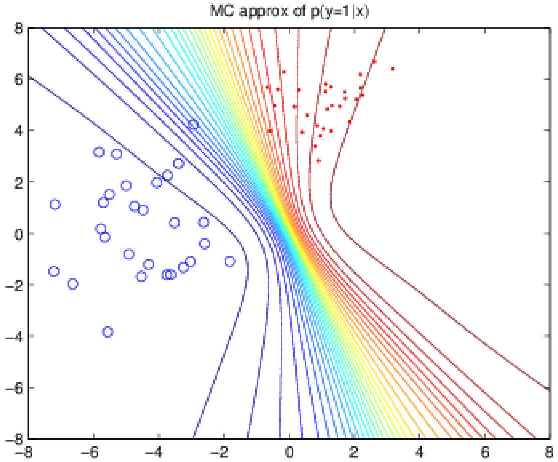


$$
\begin{aligned}
\hat{p}(\boldsymbol{\theta}|\mathcal{D}) &= \frac{1}{Z}\hat{p}(\boldsymbol{\theta}, \mathcal{D}) \propto \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^*, \mathbf{H}^{-1}) \\
Z &= e^{-E(\boldsymbol{\theta}^*)}(2\pi)^{D/2}|\mathbf{H}|^{-\frac{1}{2}}
\end{aligned}
$$

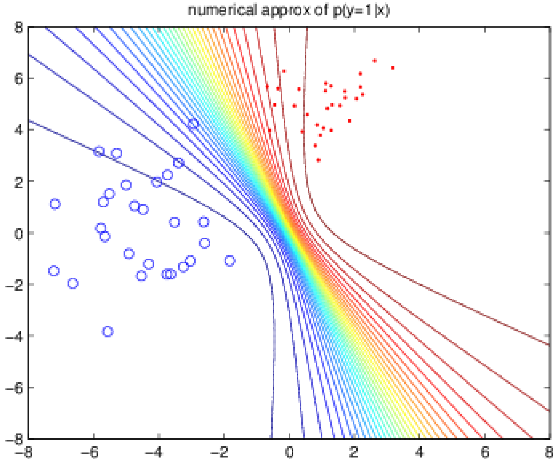
Posterior Predictive Distribution





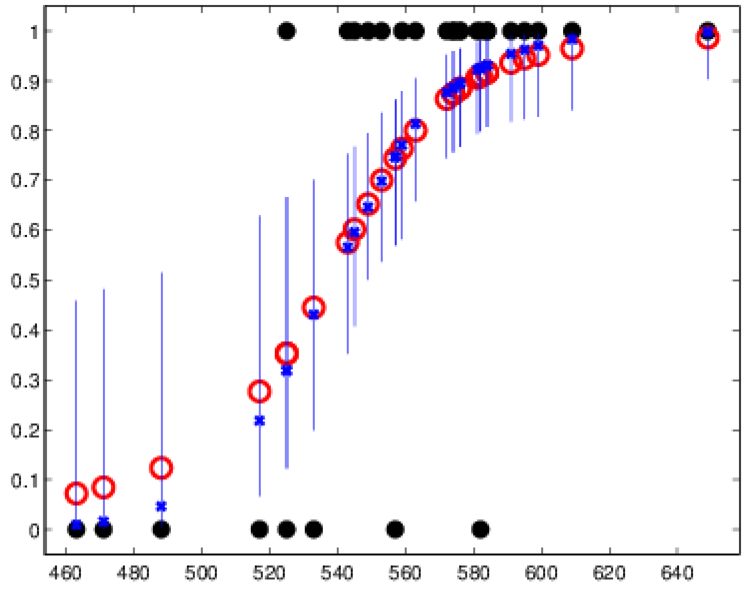


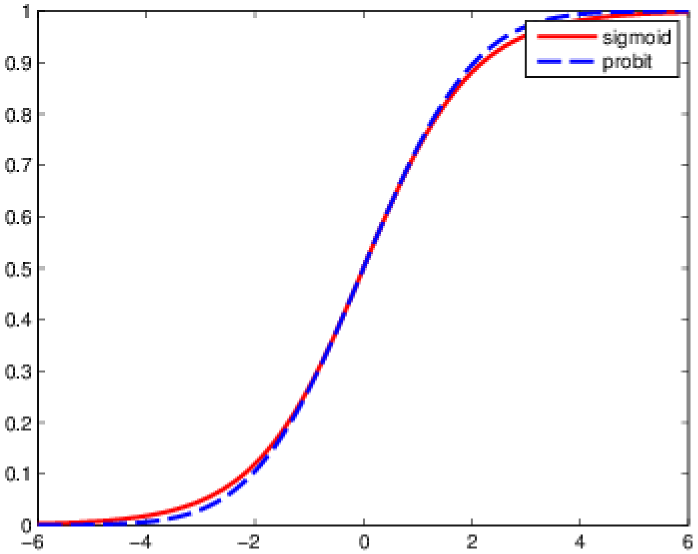
$$p(y=1|\mathbf{x},\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S}\text{sigm}((\mathbf{w}^s)^T\mathbf{x})$$



Probit Approximation

Posterior Predictive Density for SAT Data





includes 95% CI

Online Learning and Regret

- Updating the model as we go
    - Example: user is presented an ad, and either clicks or doesn't click

$$\text{regret}_k \triangleq \frac{1}{k}\sum_{t=1}^{k} f(\boldsymbol{\theta}_t, \mathbf{z}_t) - \min_{\boldsymbol{\theta}^* \in \Theta} \frac{1}{k}\sum_{t=1}^{k} f(\boldsymbol{\theta}_*, \mathbf{z}_t)$$

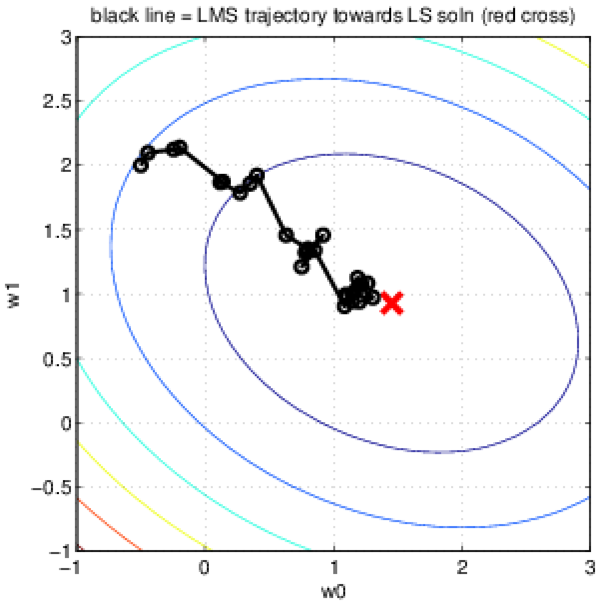
Stochastic Gradient Descent

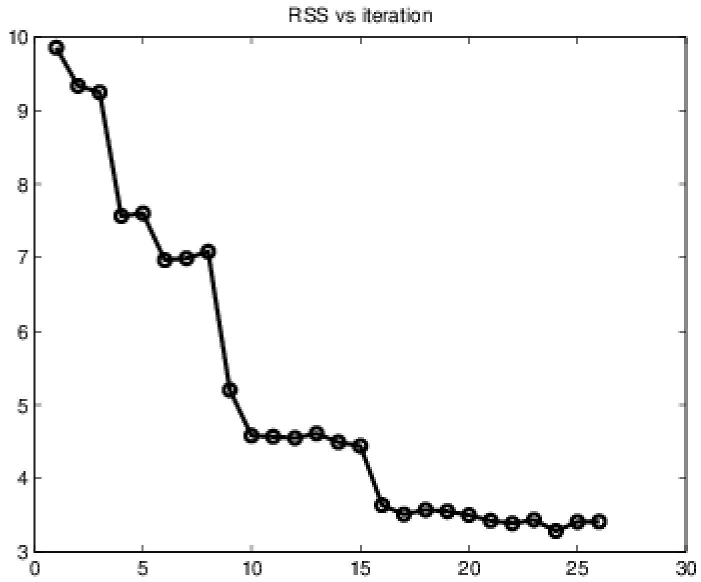
Algorithm 8.3: Stochastic gradient descent

Initialize $\boldsymbol{\theta}$, η;
repeat
  Randomly permute data;
  for $i = 1 : N$ **do**
    $\mathbf{g} = \nabla f(\boldsymbol{\theta}, \mathbf{z}_i)$;
    $\boldsymbol{\theta} \leftarrow \text{proj}_{\Theta}(\boldsymbol{\theta} - \eta \mathbf{g})$;
    Update η;
until *converged*;

The “adagrad” variant uses a per-parameter step size based on the curvature of the loss function

Least Mean Squares Example





$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k(\hat{y}_k - y_k)\mathbf{x}_k$$

known as Widrow-Hoff rule or the delta rule

Stochastic Gradient Descent scales really well; used by Apache Spark

Perceptron

Algorithm 8.4: Perceptron algorithm

Input: linearly separable data set $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, +1\}$ for $i = 1 : N$;
Initialize $\boldsymbol{\theta}_0$;
$k \leftarrow 0$;
repeat
    $k \leftarrow k + 1$;
    $i \leftarrow k \bmod N$;
    if $\hat{y}_i \neq y_i$ **then**
        $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + y_i \mathbf{x}_i$
    else
        no-op
until *converged*;

Converges *if* linearly separable

$$\hat{y}_i = \mathrm{sign}(\boldsymbol{\theta}^T \mathbf{x}_i)$$

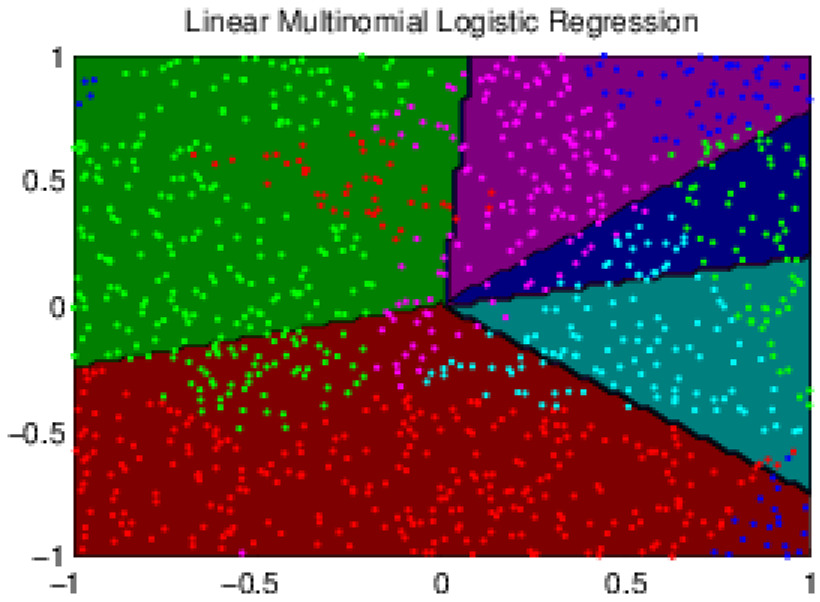
Generative v Discriminative Classifiers

- Easy to fit?
- Fit classes separately?
- Handle missing features easily?
- Can handle unlabeled training data?
- Symmetric in inputs and outputs?
- Can handle feature preprocessing?
- Well-calibrated probabilities?

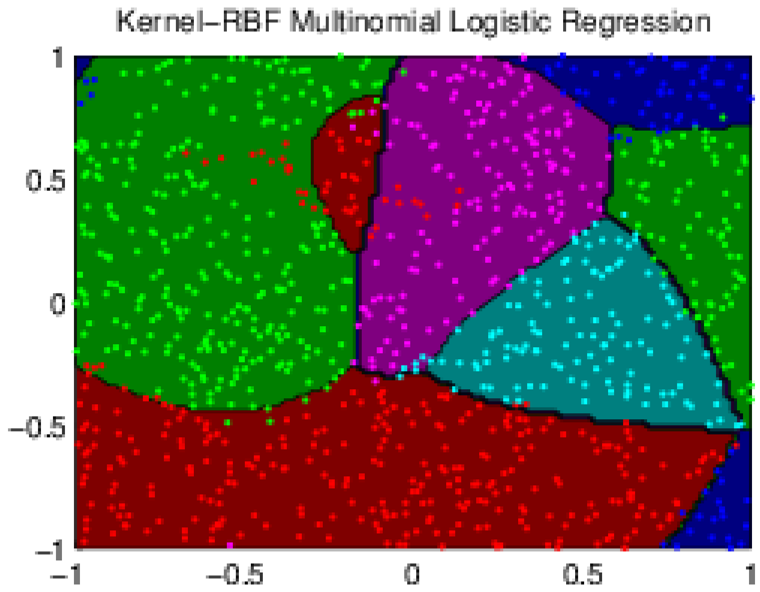
Summary of Models

Model	Classif/regr	Gen/Discr	Param/Non	Section
Discriminant analysis	Classif	Gen	Param	Sec. 4.2.2, 4.2.4
Naive Bayes classifier	Classif	Gen	Param	Sec. 3.5, 3.5.1.2
Tree-augmented Naive Bayes classifier	Classif	Gen	Param	Sec. 10.2.1
Linear regression	Regr	Discrim	Param	Sec. 1.4.5, 7.3, 7.6,
Logistic regression	Classif	Discrim	Param	Sec. 1.4.6, 8.3.4, 8.4.3, 21.8.1.1
Sparse linear/ logistic regression	Both	Discrim	Param	Ch. 13
Mixture of experts	Both	Discrim	Param	Sec. 11.2.4
Multilayer perceptron (MLP)/ Neural network	Both	Discrim	Param	Ch. 16
Conditional random field (CRF)	Classif	Discrim	Param	Sec. 19.6
K nearest neighbor classifier	Classif	Gen	Non	Sec. 1.4.2, 14.7.3
(Infinite) Mixture Discriminant analysis	Classif	Gen	Non	Sec. 14.7.3
Classification and regression trees (CART)	Both	Discrim	Non	Sec. 16.2
Boosted model	Both	Discrim	Non	Sec. 16.4
Sparse kernelized lin/logreg (SKLR)	Both	Discrim	Non	Sec. 14.3.2
Relevance vector machine (RVM)	Both	Discrim	Non	Sec. 14.3.2
Support vector machine (SVM)	Both	Discrim	Non	Sec. 14.5
Gaussian processes (GP)	Both	Discrim	Non	Ch. 15
Smoothing splines	Regr	Discrim	Non	Section 15.4.6

Multinomial Logistic Regression

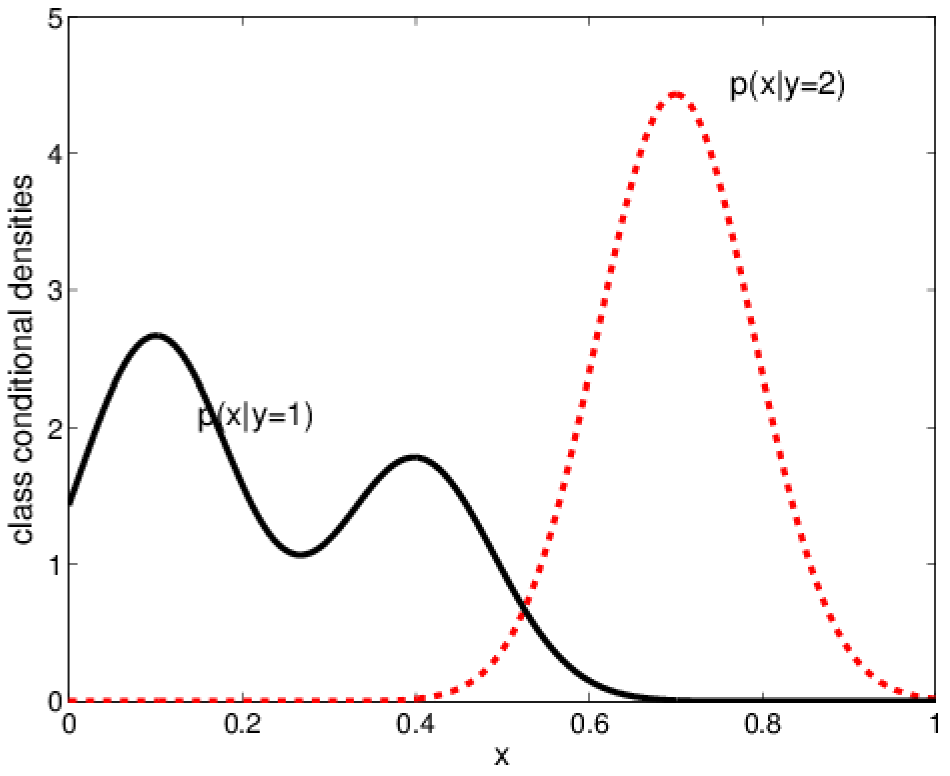


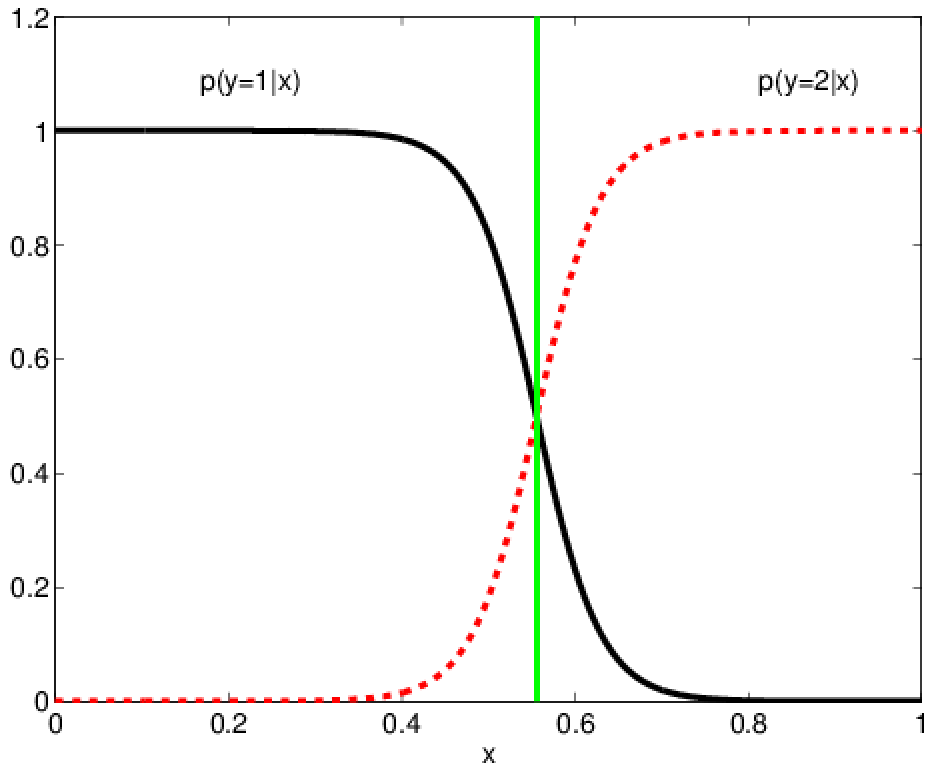
Linear



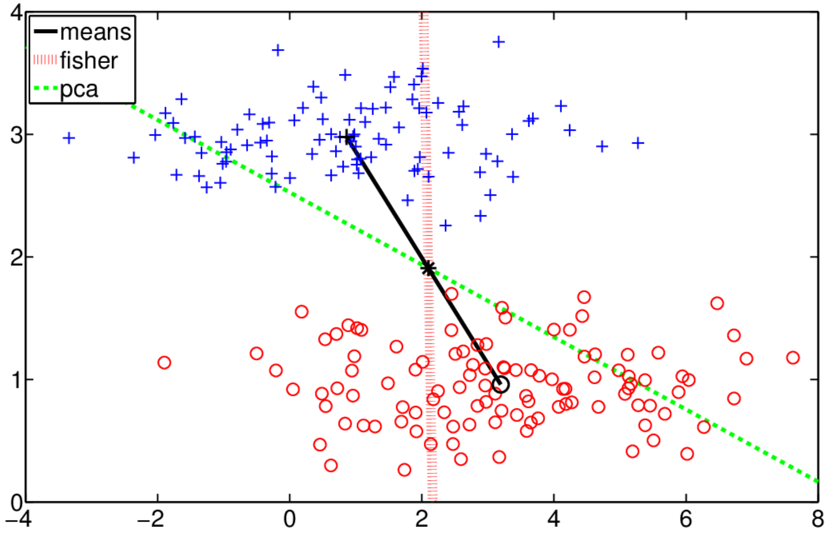
Non-Linear
RBF expansion

Class Conditional Density v Class Posterior





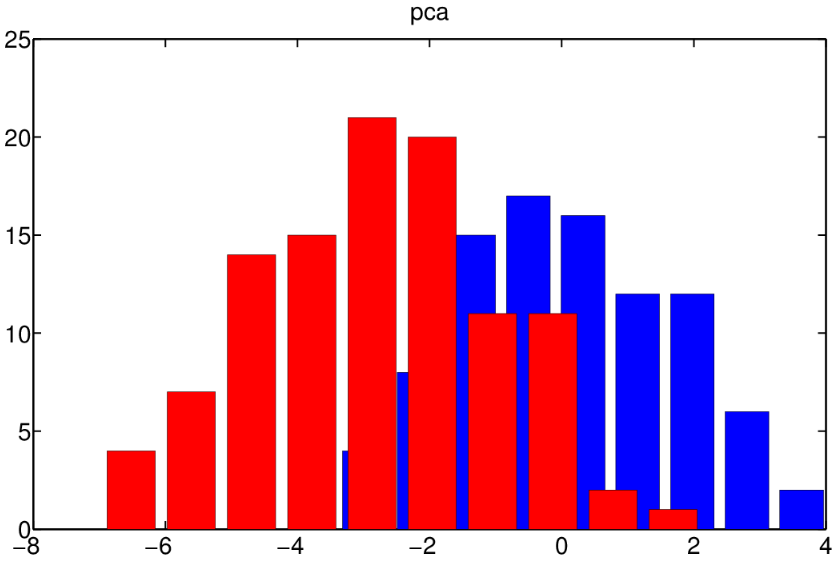
Fisher's Linear Discriminant Analysis Example



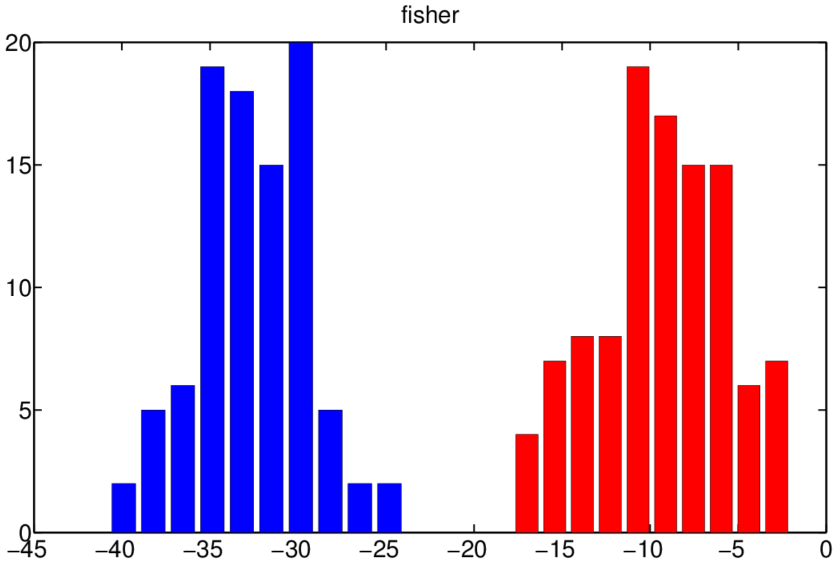
$$\mathbf{S}_W = \sum_{i:y_i=1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T + \sum_{i:y_i=2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T$$

$$\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

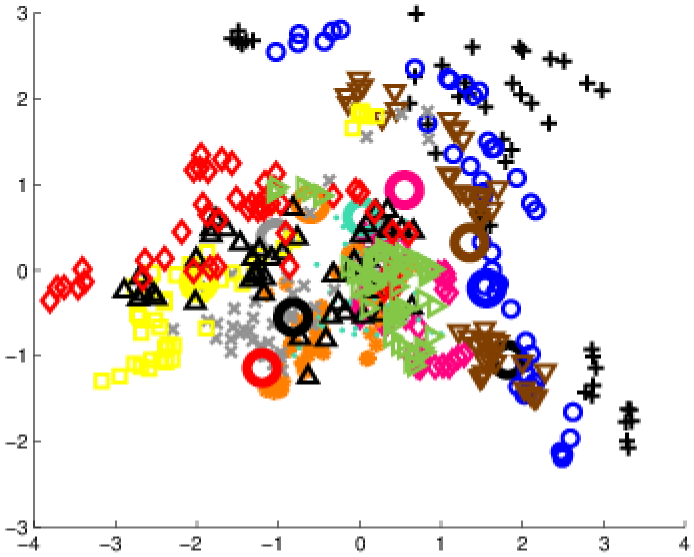
PCA sometimes makes the problem more difficult

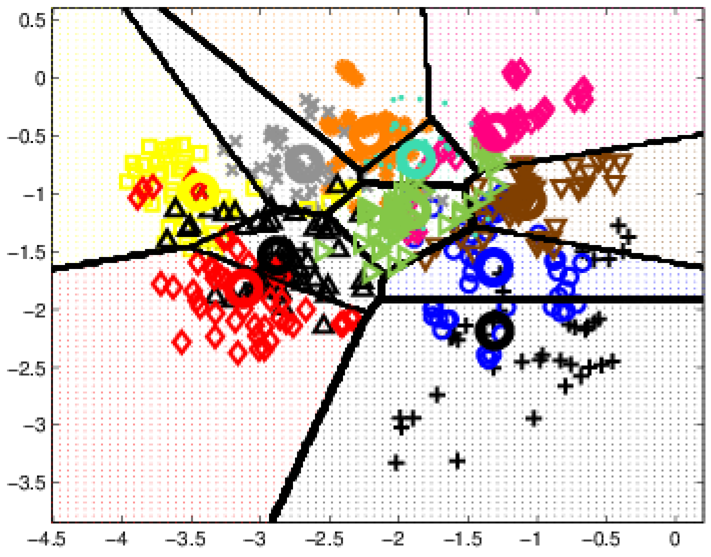


Fisher's LDA projects the data into C – 1 dimensions



PCA versus FLDA Projection of Vowel Data





Projecting high dimensional data down to C – 1 dimensions can be problematic; however, FLDA certainly seems to produce better structure than PCA for this example

Active Learning

Active Learning

- Active learning is a special case of semi-supervised learning, where we are able to obtain labels for some number of unlabeled test examples
- Popular strategies include …
  - Uncertainty sampling
  - Query by committee
  - Balancing exploration and exploitation

… to be continued on June 2nd …