Data Mining 2018
Logistic Regression
Text Classification

Ad Feelders

Universiteit Utrecht

Two types of approaches to classification

In (probabilistic) classification we are interested in the conditional distribution

$$P(Y \mid x),$$

so that, for example, when we observe $X = x$ we can predict the class with the highest probability for that value of X.

There are two basic approaches to modeling $P(Y \mid x)$:

- Generative Models (use Bayes' rule):

$$P(Y = j \mid x) = \frac{P(x \mid Y = j)P(Y = j)}{P(x)} = \frac{P(x \mid Y = j)P(Y = j)}{\sum_{i=1}^{k} P(x \mid Y = i)P(Y = i)}$$

- Discriminative Models: model $P(Y \mid x)$ directly.

Generative Models

Examples of generative classification methods:

- Naive Bayes classifier (discussed in the previous lecture)
- Linear/Quadratic Discriminant Analysis (not discussed)
- ...

Discriminative Models

Discriminative methods only model the *conditional* distribution of Y given X. The probability distribution of X itself is not modeled.

For the binary classification problem:

$$P(Y = 1 \mid X) = f(X, \beta)$$

where $f(X, \beta)$ is some deterministic function of X.

Discriminative Models

Examples of discriminative classification methods:

- Linear probability model
- Logistic regression
- Feed-forward neural networks
- ...

Discriminative Models: linear probability model

Consider the linear regression model

$$\mathbb{E}[Y \mid x] = \beta^\top x \qquad\qquad Y \in \{0, 1\},$$

where

$$\beta^\top x = \sum_{j=0}^{m} \beta_j x_j, \qquad\qquad \text{with } x_0 \equiv 1.$$

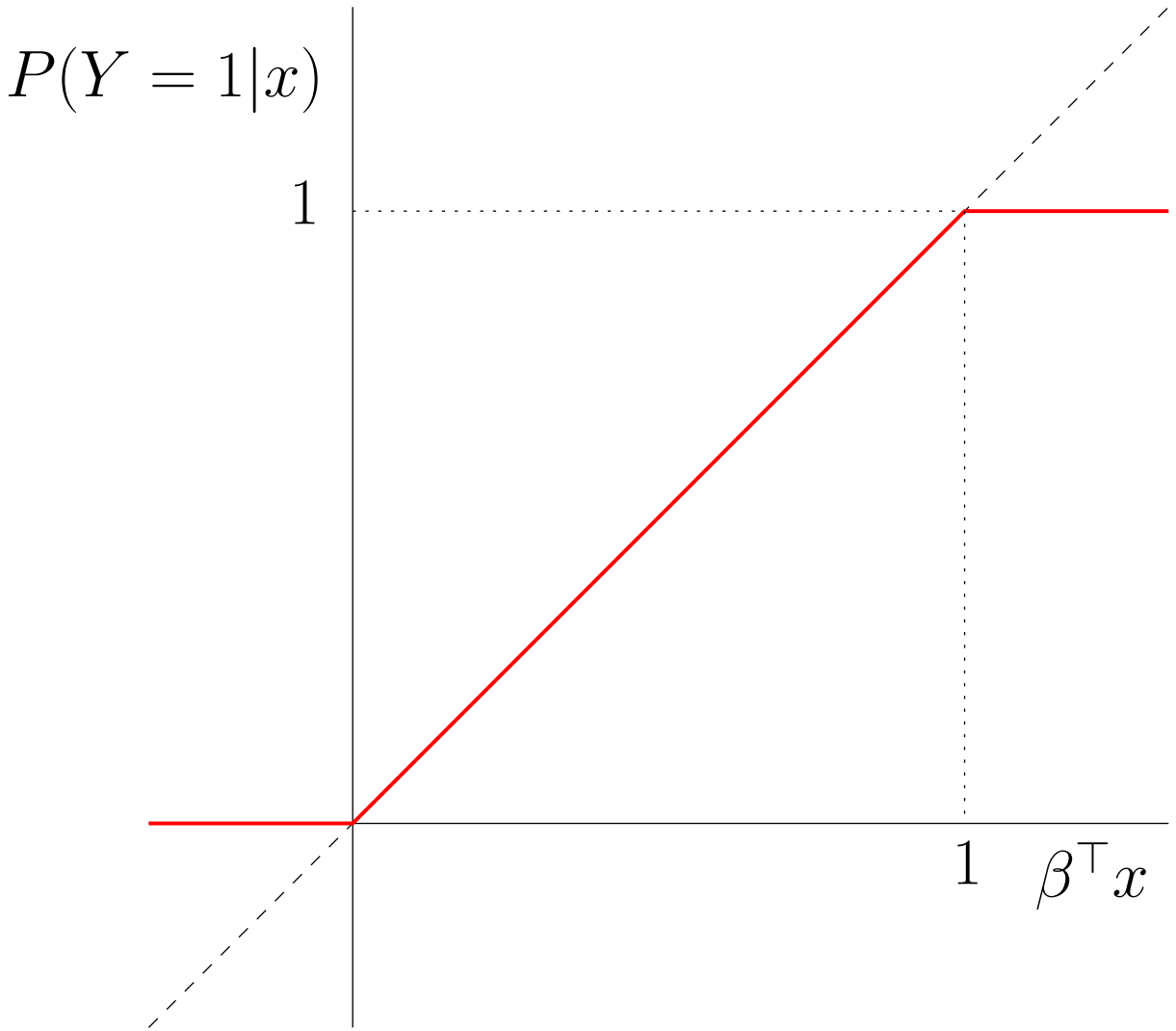
But

$$\begin{aligned} \mathbb{E}[Y \mid x] &= 1 \cdot P(Y = 1 \mid x) + 0 \cdot P(Y = 0 \mid x) \\ &= P(Y = 1 \mid x) \end{aligned}$$

So the model assumes that

$$P(Y = 1 \mid x) = \beta^\top x$$

Linear response function



Logistic regression

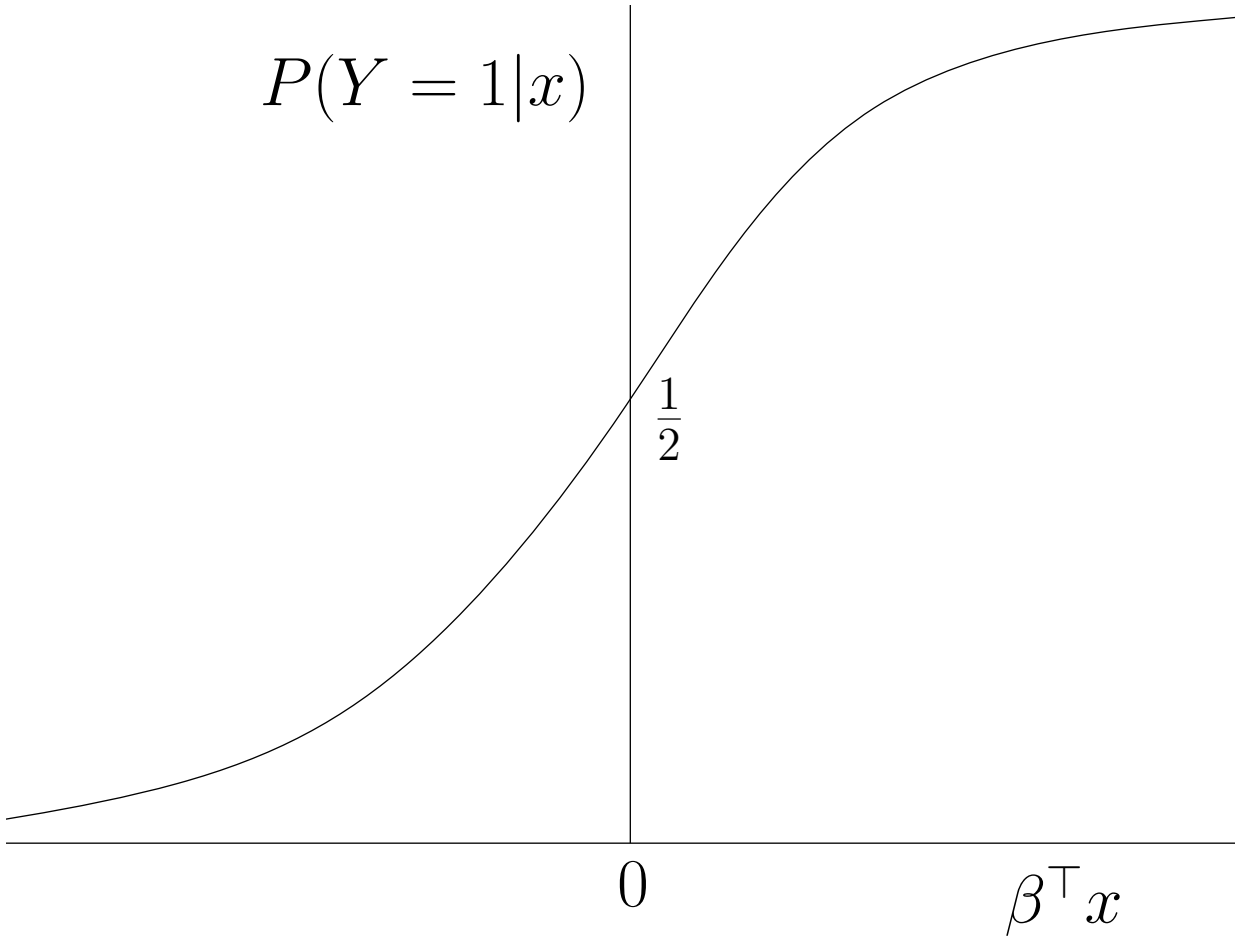
Logistic response function

$$\mathbb{E}[Y \mid x] = P(Y = 1|x) = \frac{e^{\beta^\top x}}{1 + e^{\beta^\top x}}$$

or (divide numerator and denominator by $e^{\beta^\top x}$)

$$P(Y = 1|x) = \frac{1}{1 + e^{-\beta^\top x}} = (1 + e^{-\beta^\top x})^{-1}$$

Logistic Response Function



Linearization: the logit transformation

$$\begin{aligned}
\ln\left\{\frac{P(Y=1|x)}{1-P(Y=1|x)}\right\} &= \ln\left\{\frac{(1+e^{-\beta^\top x})^{-1}}{1-(1+e^{-\beta^\top x})^{-1}}\right\} \\
&= \ln\left\{\frac{1}{(1+e^{-\beta^\top x})-1}\right\} = \ln\left\{\frac{1}{e^{-\beta^\top x}}\right\} \\
&= \ln e^{\beta^\top x} = \beta^\top x
\end{aligned}$$

In the second step, we divided the numerator and the denominator by $(1+e^{-\beta^\top x})^{-1}$. The ratio

$$\frac{P(Y=1|x)}{1-P(Y=1|x)} = \frac{P(Y=1|x)}{P(Y=0|x)}$$

is called the *odds*.

Linear Separation

Assign to class 1 if $P(Y = 1|x) > P(Y = 0|x)$, i.e. if

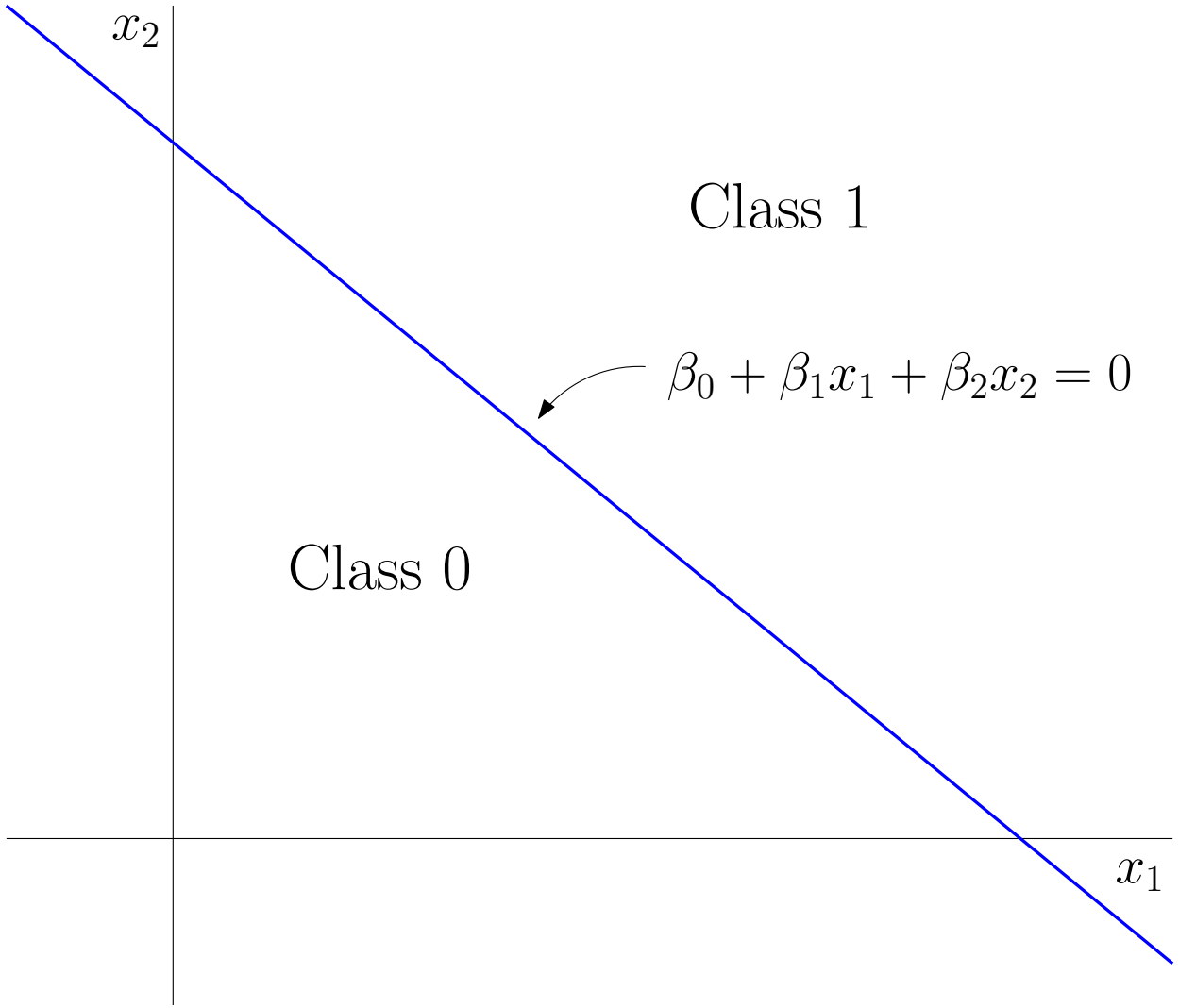
$$\frac{P(Y = 1|x)}{P(Y = 0|x)} > 1$$

This is true if

$$\ln\left\{\frac{P(Y = 1|x)}{P(Y = 0|x)}\right\} > 0$$

So assign to class 1 if $\beta^\top x > 0$, and to class 0 otherwise.

Decision Boundary



Maximum Likelihood Estimation

$Y = 1$ if heads, $Y = 0$ if tails. $p = P(Y = 1)$.
One coin flip

$$P(y) = p^y(1-p)^{1-y}$$

Note that $P(1) = p$, $P(0) = 1 - p$ as required.
Sequence of n independent coin flips

$$P(y_1, y_2, ..., y_n) = \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

which defines the likelihood function when viewed as a function of p.

Maximum Likelihood Estimation

In a sequence of 10 coin flips we observe $y = (1, 0, 1, 1, 0, 1, 1, 1, 1, 0)$.

The corresponding likelihood function is

$$\begin{aligned} P(y|p) &= p \cdot (1-p) \cdot p \cdot p \cdot (1-p) \cdot p \cdot p \cdot p \cdot p \\ &\quad \cdot (1-p) = p^7 (1-p)^3 \end{aligned}$$

The corresponding log-likelihood function is

$$\ln P(y|p) = \ln(p^7 (1-p)^3) = 7 \ln p + 3 \ln(1-p)$$

Computing the maximum

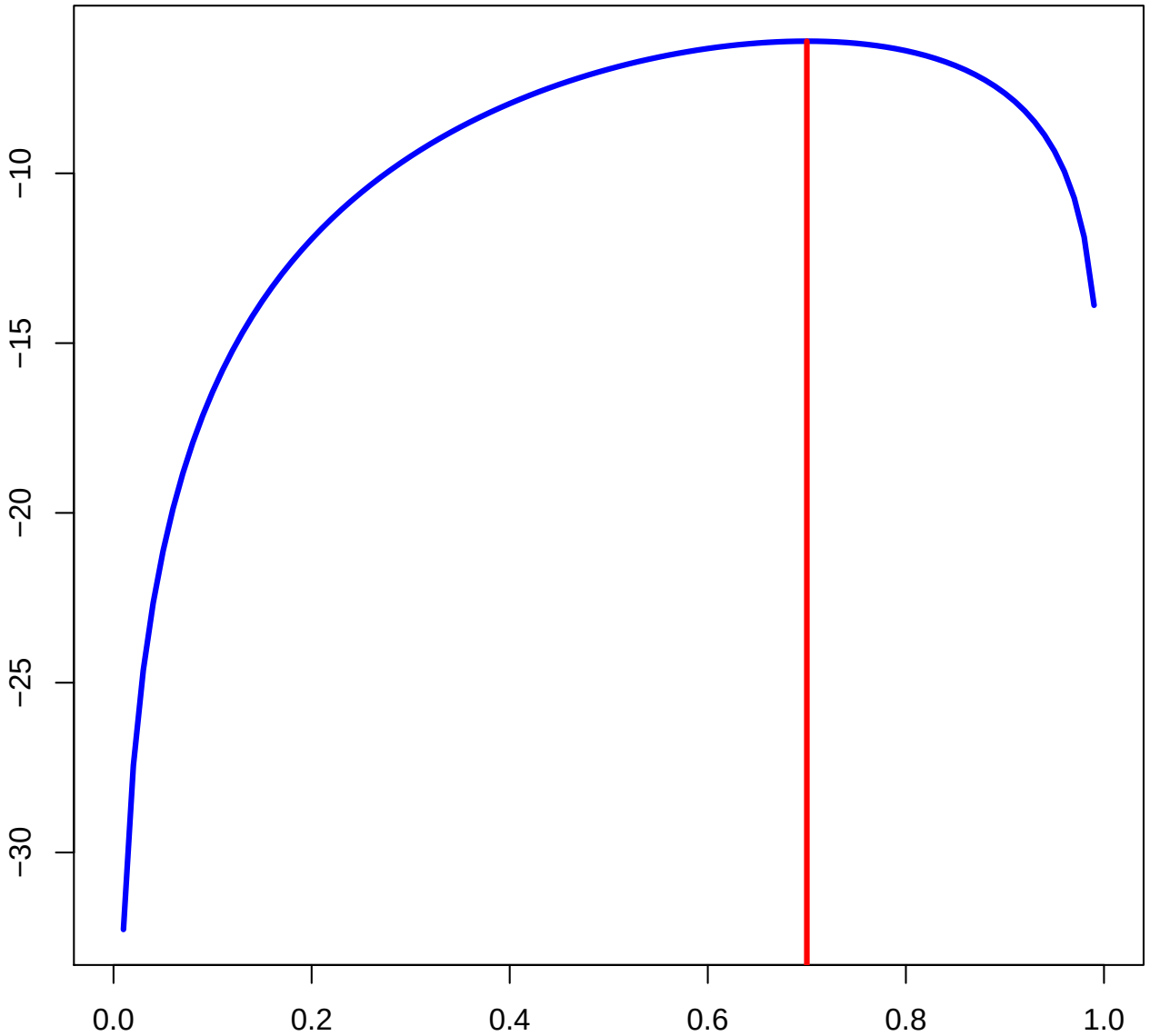
To determine the maximum we take the derivative and equate it to zero

$$\frac{d \ln P(y|p)}{dp} = \frac{7}{p} - \frac{3}{1-p} = 0$$

which yields maximum likelihood estimate $\hat{p} = 0.7$.

This is just the relative frequency of heads in the sample.

Log-likelihood function for $y = (1, 0, 1, 1, 0, 1, 1, 1, 1, 0)$



ML estimation for logistic regression

Logistic regression is a bit like the coin tossing example, except that now the probability of success p_i depends on x_i:

$$\begin{aligned} p_i &= P(Y = 1 \mid x_i) = (1 + e^{-\beta^\top x_i})^{-1} \\ 1 - p_i &= P(Y = 0 \mid x_i) = (1 + e^{\beta^\top x_i})^{-1} \end{aligned}$$

we can represent its probability distribution as follows

$$P(y_i) = p_i^{y_i}(1 - p_i)^{1-y_i} \qquad y_i \in \{0, 1\}; \quad i = 1, \ldots, n$$

ML estimation for logistic regression

Example

i	x_i	y_i	$P(y_i)$
1	8	0	$(1+e^{\beta_0+8\beta_1})^{-1}$
2	12	0	$(1+e^{\beta_0+12\beta_1})^{-1}$
3	15	1	$(1+e^{-\beta_0-15\beta_1})^{-1}$
4	10	1	$(1+e^{-\beta_0-10\beta_1})^{-1}$

LR: likelihood function

Since the y_i observations are independent:

$$P(y|\beta) = \prod_{i=1}^{n} P(y_i) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} \tag{1}$$

Or, taking the natural log:

$$\begin{aligned} \ln P(y|\beta) &= \ln \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} \\ &= \sum_{i=1}^{n} \{y_i \ln p_i + (1-y_i)\ln(1-p_i)\} \end{aligned}$$

LR: log-likelihood function

For the logistic regression model we have

$$\begin{aligned} p_i &= (1 + e^{-\beta^\top x_i})^{-1} \\ 1 - p_i &= (1 + e^{\beta^\top x_i})^{-1}, \end{aligned}$$

so filling in gives

$$\ell(\beta) = \sum_{i=1}^{n} \left\{ y_i \ln \left(\frac{1}{1 + e^{-\beta^\top x_i}} \right) + (1 - y_i) \ln \left(\frac{1}{1 + e^{\beta^\top x_i}} \right) \right\}$$

- Non-linear function of the parameters.
- No closed form solution (no nice formulas for the parameter estimates).
- Likelihood function globally concave so relatively easy optimization problem.

First-order conditions

$$\ell(\beta) = \sum_{i=1}^{n} \{y_i \ln p_i + (1 - y_i) \ln(1 - p_i)\}$$

$$g(\beta_j) = \frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{y_i}{p_i} \cdot \frac{\partial p_i}{\partial \beta_j} + \frac{1 - y_i}{1 - p_i} \cdot \frac{\partial (1 - p_i)}{\partial \beta_j} \tag{1}$$

$$\frac{\partial p_i}{\partial \beta_j} = p_i(1 - p_i)x_{ij} \tag{2}$$

Filling in (2) in equation (1) gives:

$$g(\beta_j) = \sum_{i=1}^{n} (y_i - p_i)x_{ij}$$

First-order conditions

First order condition for maximum:

$$g(\beta_j) = \sum_{i=1}^{n} (y_i - p_i) x_{ij} = 0, \qquad j = 0, \ldots, m.$$

For the intercept β_0 we get

$$\frac{\partial \ell(\beta)}{\partial \beta_0} = \sum_{i=1}^{n} (y_i - p_i) = 0,$$

since $x_{i0} \equiv 1$. So in the optimal solution we have

$$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} p_i$$

The expected number of cases with $Y = 1$ is equal to the observed number of cases with $Y = 1$.

First-order conditions

First order condition for maximum:

$$g(\beta_j) = \sum_{i=1}^{n} (y_i - p_i) x_{ij} = 0$$

Likewise, for β_j we get

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} (y_i - p_i) x_{ij} = 0.$$

So in the optimal solution we have

$$\sum_{i=1}^{n} y_i x_{ij} = \sum_{i=1}^{n} p_i x_{ij}$$

Interpretation?

Optimization by Gradient Ascent

The gradient points in the direction of the *steepest ascent* of the function.

We can perform optimization by the method of gradient ascent as follows. The new estimate of β_j based on processing the i-th observation is:

$$\beta_j^{(t+1)} = \beta_j^{(t)} + \eta \times g(\beta_j)|_{\beta=\beta^{(t)}} = \beta_j^{(t)} + \eta \times (y_i - p_i^{(t)})x_{ij},$$

where

$$p_i^{(t)} = (1 + e^{-\beta^{(t)\top} x_i})^{-1}$$

is the current estimate of $P(Y = 1 \mid x_i)$, that is, using $\beta^{(t)}$.

Optimization by Gradient Ascent

The basic *gradient-ascent algorithm* applied to logistic regression:

1. *choose* an initial value $\beta^{(0)}$ (e.g. at random); $t \leftarrow 0$
2. determine the gradient $\frac{\partial \ell(\beta)}{\partial \beta_j}|_{\beta=\beta^{(t)}}$ of $\ell(\beta)$ at $\beta^{(t)}$ and *update*

$$\beta_j^{(t+1)} = \beta_j^{(t)} + \eta \times \sum_{i=1}^{n} (y_i - p_i^{(t)}) x_{ij}, \qquad j = 0, \ldots, m$$

3. Repeat the previous step until

$$g(\beta_j)|_{\beta=\beta^{(t)}} = 0 \qquad \text{for all } j = 0, \ldots, m$$

and *check* if a *(local) maximum* has been reached.

$\eta > 0$ is the *step size*.

Fitted Response Function

Substitute maximum likelihood estimates into the response function to obtain the *fitted response function*

$$\hat{P}(Y = 1 \mid x) = \frac{e^{\hat{\beta}^\top x}}{1 + e^{\hat{\beta}^\top x}}$$

Example: Programming Assignment

Model the probability of successfully completing a programming assignment.

Explanatory variable: "programming experience".
We find $\hat{\beta}_0 = -3.0597$ and $\hat{\beta}_1 = 0.1615$, so

$$\hat{P}(Y = 1 \mid x) = \frac{e^{-3.0597+0.1615x}}{1 + e^{-3.0597+0.1615x}}$$

14 months of programming experience:

$$\hat{P}(Y = 1 \mid x = 14) = \frac{e^{-3.0597+0.1615(14)}}{1 + e^{-3.0597+0.1615(14)}} \approx 0.31$$

Interpretation

We have

$$\ln\left\{\frac{\hat{P}(Y=1\mid x)}{\hat{P}(Y=0\mid x)}\right\} = -3.0597 + 0.1615x,$$

so with every additional month of programming experience, the log odds increase with 0.1615. The odds are multiplied by $e^{0.1615} \approx 1.175$ so with every additional month of programming experience, the odds increase with 17.5%.

Note that the effect of an increase in x on the probability of success depends on the value of x:

- An increase from 14 to 24 months of programming experience leads to an increase of the probability of success from 0.31 to 0.69.
- An increase from 34 to 44 months of programming experience leads to an increase of the probability of success from 0.92 to 0.98.

Example: Programming Assignment

	month.exp	success	fitted		month.exp	success	fitted
1	14	0	0.310262	16	13	0	0.276802
2	29	0	0.835263	17	9	0	0.167100
3	6	0	0.109996	18	32	1	0.891664
4	25	1	0.726602	19	24	0	0.693379
5	18	1	0.461837	20	13	1	0.276802
6	4	0	0.082130	21	19	0	0.502134
7	18	0	0.461837	22	4	0	0.082130
8	12	0	0.245666	23	28	1	0.811825
9	22	1	0.620812	24	22	1	0.620812
10	6	0	0.109996	25	8	1	0.145815
11	30	1	0.856299				
12	11	0	0.216980				
13	30	1	0.856299				
14	5	0	0.095154				
15	20	1	0.542404				

Allocation Rule

Probability of the classes is equal when

$$-3.0597 + 0.1615x = 0$$

Solving for x we get $x \approx 18.95$.

Allocation Rule:

$x \geq 19$: *predict* $y = 1$
$x < 19$: *predict* $y = 0$

If a person has 19 months or more programming experience, predict success, otherwise predict failure.

Programming Assignment: Confusion Matrix

Cross table of observed and predicted class label:

	0	1
0	11	3
1	3	8

Row: observed, Column: predicted

Error rate: 6/25=0.24

Default (predict majority class): 11/25=0.44

How to in R

```
> prog.logreg <- glm(success ~ month.exp, data=prog.dat, family=binomial)
> summary(prog.logreg)


Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.05970    1.25935  -2.430   0.0151 *
month.exp    0.16149    0.06498   2.485   0.0129 *
---

Number of Fisher Scoring iterations: 4


> table(prog.dat$success, as.numeric(prog.logreg$fitted > 0.5))


    0  1
  0 11  3
  1  3  8
```

Non-binary classes in logistic regression

Recall the logistic regression model assumption for binary class variable $Y \in \{0, 1\}$:

$$P(Y = 1|x) = \frac{\exp(\beta^\top x)}{1 + \exp(\beta^\top x)}$$

from which it follows that

$$P(Y = 0|x) = \frac{1}{1 + \exp(\beta^\top x)}$$

since

$$P(Y = 1|x) + P(Y = 0|x) = 1$$

Non-binary classes in logistic regression

We can generalize this model to non-binary class variable $Y \in \{0, 1, \ldots, K-1\}$ (where K is the number of classes) as follows

$$P(Y = k|x) = \frac{\exp(\beta_k^\top x)}{\sum_{j=0}^{K-1} \exp(\beta_j^\top x)}$$

where we now have a weight vector β_k for each class.

This is called the *multinomial logit model* or multi-class logistic regression.

Multi-class logistic regression

We can arrive at this model in the following steps:

1. Assume that $P(Y = k|x)$ is a function of the linear combination $\beta_k^\top x$.
2. To ensure that the probabilities are non-negative, take the exponential $\exp(\beta_k^\top x)$.
3. To make sure the probabilities sum to 1, we divide $\exp(\beta_k^\top x)$ by $\sum_{j=0}^{K-1} \exp(\beta_j^\top x)$:

$$P(Y = k|x) = \frac{\exp(\beta_k^\top x)}{\sum_{j=0}^{K-1} \exp(\beta_j^\top x)}$$

Identification Restriction

Note that

$$\frac{\exp((\beta_k + d)^\top x)}{\sum_{j=0}^{K-1} \exp((\beta_j + d)^\top x)} = \frac{\exp(d^\top x)}{\exp(d^\top x)} \frac{\exp(\beta_k^\top x)}{\sum_{j=0}^{K-1} \exp(\beta_j^\top x)}$$

so adding a vector d to each of the vectors $\beta_j, j = 0, \ldots, K-1$ would yield the same fitted probabilities.

To identify the model, we put $\beta_0 = 0$.

Verify that binary logistic regression is a special case of the multinomial logit model, with $K = 2$, since $\exp(\beta_0^\top x) = \exp(0) = 1$.

Interpretation

We have

$$\ln\left\{\frac{P(Y=k|x)}{P(Y=\ell|x)}\right\} = (\beta_k - \beta_\ell)^\top x,$$

which allows us to interpret $\beta_{kj} - \beta_{\ell j}$ as follows:

> *for a unit increase in x_j, the log-odds of class k versus class ℓ is expected to change by $\beta_{kj} - \beta_{\ell j}$ units, holding all the other variables constant.*

Since $\beta_0 = 0$, β_{kj} is the effect of x_j on the log-odds of class k relative to class 0:

> *for a unit increase in x_j, the log-odds of class k versus class 0 is expected to change by β_{kj} units, holding all the other variables constant.*

Regularization

- If we have a large number of predictors, even a linear model estimated with maximum likelihood can be prone to overfitting.
- This can be controlled by punishing large (positive or negative) weights. The coefficient estimates are *shrunken* towards zero.
- Add a penalty term for the size of the coefficients to the objective function.
- With LASSO penalty:

$$E(\beta) = -\ell(\beta) + \lambda \sum_{j=1}^{m} |\beta_j|,$$

  where $E(\beta)$ is the new error function that we want to minimize, and $-\ell(\beta)$ is the negative log-likelihood function.
- The value of λ is usually selected by cross-validation.

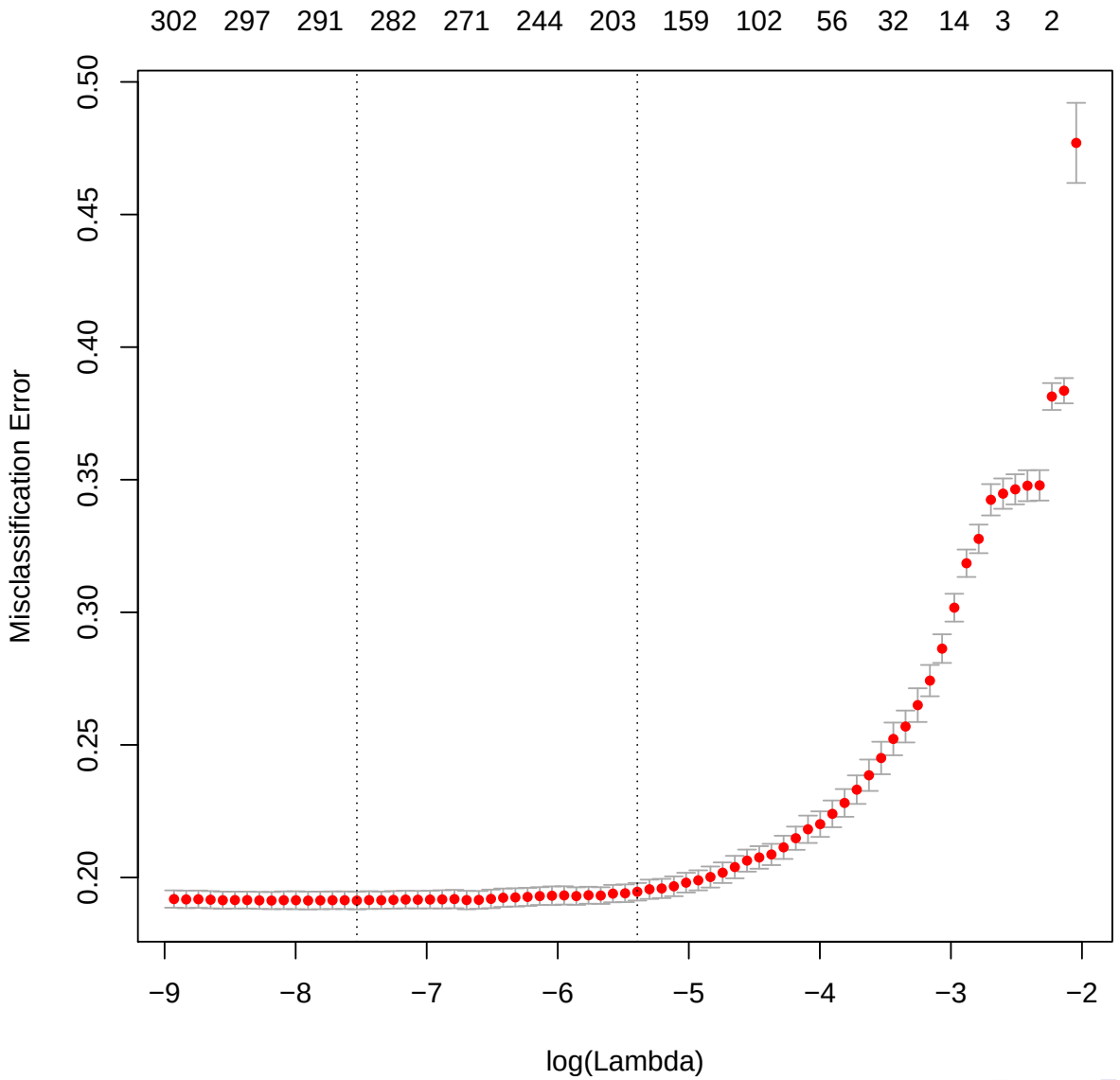
Application of Logistic Regression to Movie Reviews

```
# logistic regression with lasso penalty

> reviews.glmnet <- cv.glmnet(as.matrix(train.dtm),labels[index.train],
                      family="binomial",type.measure="class")
> plot(reviews.glmnet)

> coef(reviews.glmnet,s="lambda.1se")
309 x 1 sparse Matrix of class "dgCMatrix"
                          1
bad           -0.613843496
beautiful      0.378249156
best           0.400765691
better        -0.193594713
boring        -0.904918921
excellent      0.874061528
fun            0.390055537
funny          .
minutes       -0.381871597
perfect        0.757174138
poor          -0.726663951
script        -0.461754268
stupid        -0.555516834
supposed      -0.611473721
terrible      -0.830472064
wonderful      0.697696588
worst         -1.431738320
```

Cross-Validation on lambda



Application of Logistic Regression to Movie Reviews

```
# make predictions on the test set
> reviews.logreg.pred <- predict(reviews.glmnet,
   newx=as.matrix(test.dtm),s="lambda.1se",type="class")
# show confusion matrix
> table(reviews.logreg.pred,labels[-index.train])

reviews.logreg.pred    0    1
                  0 3468  704
                  1 1032 3796
# compute accuracy: about 81% correct
> (3468+3796)/9000
[1] 0.8071111
```

Using tf-idf weights

- Currently, we use term frequency as feature value.
- In information retrieval other term weighting schemes are popular.
- For example:

$$\text{tf-idf}(i,j) = \text{tf}(i,j) \times \text{idf}(i),$$

  where tf-idf(i,j) is the number of times term i occurs in document j, and

$$\text{idf}(i) = \log \frac{N_{doc}}{|\{d : w_i \in d\}|}$$

- There are many variations on this theme.
- Should work well in finding relevant documents to queries, but not obvious why it should work in document classification.

Logistic Regression: using tf-idf weights

```
# construct training document term matrix with tf-idf weights
> train2.dtm <- DocumentTermMatrix(reviews.all[index.train],
                  control=list(weighting=weightTfIdf))

# remove sparse terms
> train2.dtm <- removeSparseTerms(train2.dtm,0.95)

# perform logistic regression with lasso penalty
> reviews2.glmnet <- cv.glmnet(as.matrix(train2.dtm),labels[index.train],
                      family="binomial",type.measure="class")

# compute tf-idf scores on test set: we use the document frequency
# from the training set!
> train3.dtm <- as.matrix(train.dtm)

# convert term frequency counts to binary indicator
> train3.dtm <- matrix(as.numeric(train3.dtm > 0),nrow=16000,ncol=308)
```

Logistic Regression: using tf-idf weights

```
# sum the columns of the training set
> train3.idf <- apply(train3.dtm,2,sum)

# compute idf for each term (column)
> train3.idf <- log2(16000/train3.idf)

# term frequencies on the test set
> test3.dtm <- as.matrix(test.dtm)

# compute tf-idf weights on the test set
> for(i in 1:308){test3.dtm[,i] <- test3.dtm[,i]*train3.idf[i]}
```

Logistic Regression: using tf-idf weights

```
# make predictions on the test set using lambda=lambda.1se
> reviews.logreg2.pred <- predict(reviews2.glmnet,
           newx=test3.dtm,s="lambda.1se",type="class")

# show confusion matrix
> table(reviews.logreg2.pred,labels[-index.train])

reviews.logreg2.pred    0    1
                   0 3641  874
                   1  859 3626

# compute accuracy: still about 81% correct
> (3641+3626)/9000
[1] 0.8074444
```

Including Bigrams

The bigrams in

```
the spy who loved me
```

are:

```
the spy
spy who
who loved
loved me
```

but not for example

```
spy loved
```

The two words need to be next to each other.

Including Bigrams

```
# extract bigrams
> train.dtm2 <- DocumentTermMatrix(reviews.all[index.train],
    control = list(tokenize = BigramTokenizer))
# more than one million bigrams!
> dim(train.dtm2)
[1]   16000 1253863
> train.dtm2 <- removeSparseTerms(train.dtm2,0.99)
# after removing sparse bigrams only 152 left
> dim(train.dtm2)
[1] 16000   152
> train.dat1 <- as.matrix(train.dtm)
> train.dat2 <- as.matrix(train.dtm2)
# combine unigrams and bigrams
> train.dat <- cbind(train.dat1,train.dat2)
> dim(train.dat)
[1] 16000   460
```

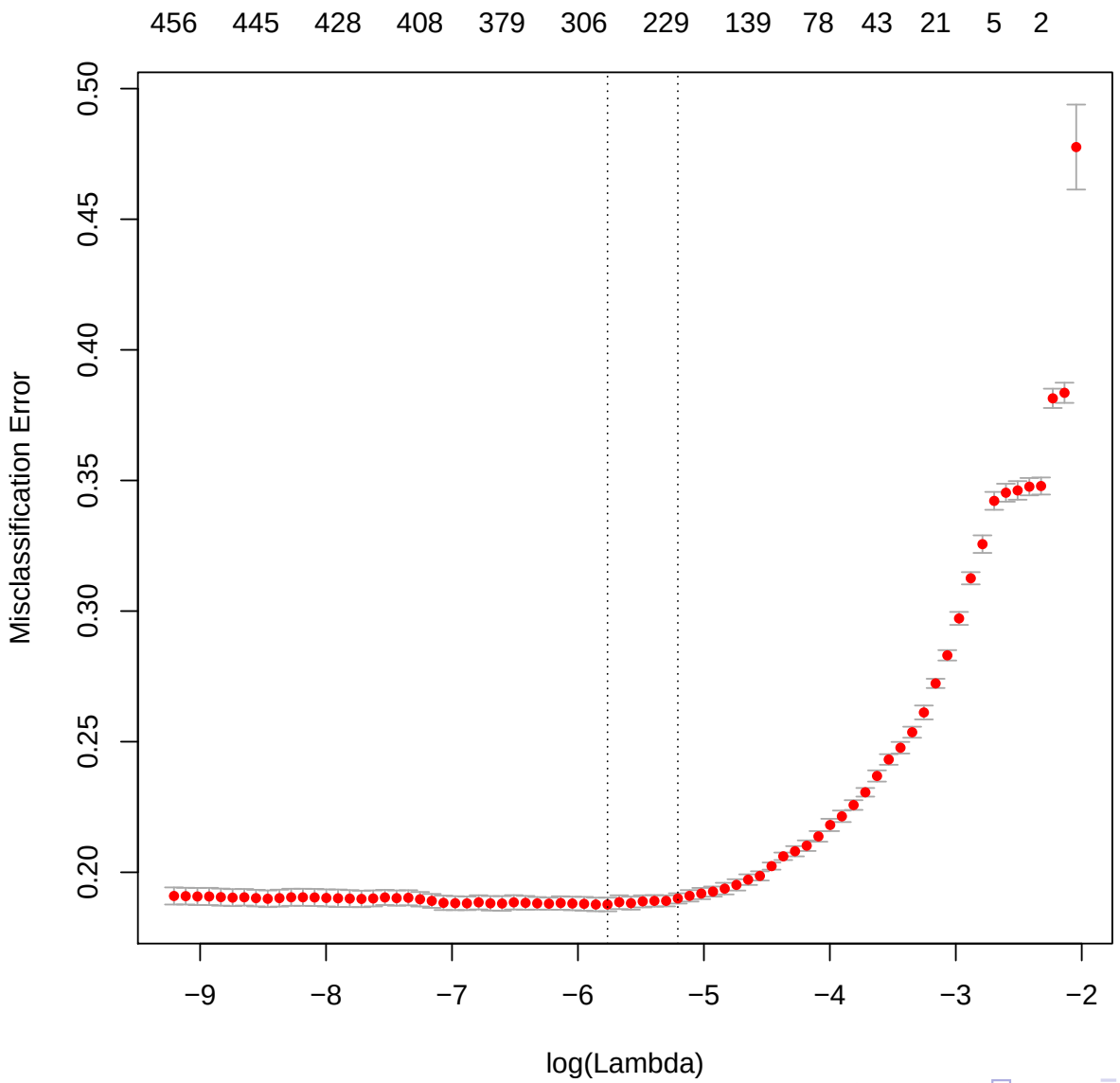
Including Bigrams

```
# fit regularized logistic regression model
# use cross-validation to evaluate different lambda values
> reviews3.glmnet <- cv.glmnet(train.dat,labels[index.train],
    family="binomial",type.measure="class")

# show coefficient estimates for lambda-1se
# (only a selection of the bigram coefficients is shown here)
> coef(reviews3.glmnet,s="lambda.1se")

bad movie             -0.1982858150
black white            0.0165120496
cant believe          -0.1697569250
character development -0.0787603597
dont want              0.1142457065
end movie             -0.0704077038
even though            0.1529414255
felt like             -0.1599889926
first time             0.2509504531
good film              0.2710481124
great movie            0.3211405587
highly recommend       1.1933642095
just plain            -0.5027096999
make sense            -0.3965341401
much better           -0.0149710502
must see               0.9240885363
one best               1.0022471402
one worst             -0.5688509905
read book             -0.2966097639
really good            0.1521121622
worst movie           -0.1227700125
```

Cross-Validation on lambda



Including Bigrams

```
# create document term matrix for the test data,
# using the training dictionary
> test3.dtm <- DocumentTermMatrix(reviews.all[-index.train],
    list(dictionary=dimnames(train.dat)[[2]]))
# convert to ordinary matrix
> test3.dat <- as.matrix(test3.dtm)
# get columns in the same order as on the training set
> test3.dat <- test3.dat[,dimnames(train.dat)[[2]]]
# make predictions using lambda.1se
> reviews.logreg3.pred <- predict(reviews3.glmnet,newx=test3.dat,
    s="lambda.1se",type="class")
> table(reviews.logreg3.pred,labels[-index.train])

reviews.logreg3.pred    0    1
                   0 3436  707
                   1 1064 3793
# accuracy does not improve
> (3436+3793)/9000
[1] 0.8032222
```