# Classification

Statistical Learning
James Scott (UT-Austin)

Reference: Introduction to Statistical Learning Chapter 4

# Outline

1. Linear probability model
2. Logistic regression: the basics
3. Interpreting a logit model
4. Estimating a logit model: MLE
5. KNN for classification
6. Multinomial logit
7. Evaluating a classifier: likelihood and deviance
8. Bayes' rule: LDA and Naive Bayes

# Classification

- In classification, the target variable $y$ is membership in a category $\{1, \dots, M\}$.

- Each observation is an observed class $y_i$, together with a vector of features $x_i$.

- The classification problem: given new $x^\star$, what is $y^\star$?

# Linear probability model

We'll start with binary classification (where $y$ is 0 or 1).

Recall the basic form of a supervised learning problem:

$$E(y \mid x) = f(x)$$

Suppose the outcome $y$ is binary (0/1). Then:

$$\begin{aligned} E(y \mid x) &= 0 \cdot P(y = 0 \mid x) + 1 \cdot P(y = 1 \mid x) \\ &= P(y = 1 \mid x) \end{aligned}$$

Conclusion: the expectation is a probability

# Linear probability model

Now suppose we choose $f(x)$ to be a linear function of the features $x_i$:

$$
\begin{aligned}
P(y = 1 \mid x) = f(x) &= x \cdot \beta \\
&= \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j
\end{aligned}
$$

This is called the *linear probability model*: the probability of a "yes" outcome ($y = 1$) is linear in $x_i$.

# LPM: spam classification

Let's revisit our spam classification problem:

- `spamfit.csv`: 3000 e-mails (40% spam) with 9 features.
- `spamtest.csv`: 601 testing e-mails for assessing performance.

First few lines of `spamfit`:

```
  word.freq.remove word.freq.order word.freq.free word.freq.meeting
1                0            0.00           5.35                 0
2                0            0.00           0.00                 0
3                0            0.31           0.63                 0
  word.freq.re word.freq.edu char.freq.semicolon char.freq.exclamation
1            0             0                   0                 0.357
2            0             0                   0                 1.975
3            0             0                   0                 0.055
  capital.run.length.average y
1                      1.971 1
2                     35.461 1
3                      3.509 1
```

# LPM: spam classification

Let's build a linear probability using all the available features for P(spam | x) and examine the fitted coefficients:

```
# Recall: the dot (.) says "use all variables not otherwise named"
lm_spam1 = lm(y ~ ., data=spamfit)
coef(lm_spam1) %>% round(3)
```

```
          (Intercept)           word.freq.remove 
                0.281                      0.311 
     word.freq.order             word.freq.free 
                0.284                      0.097 
   word.freq.meeting               word.freq.re 
               -0.059                     -0.039 
       word.freq.edu        char.freq.semicolon 
               -0.051                     -0.096 
char.freq.exclamation capital.run.length.average 
                0.229                      0.001 
```

# LPM: spam classification

## In-sample performance:

```
phat_train_spam1 = predict(lm_spam1, spamfit)  # predicted probabilities
yhat_train_spam1 = ifelse(phat_train_spam1 > 0.5, 1, 0)
confusion_in = table(y = spamfit$y, yhat = yhat_train_spam1)
confusion_in
```

```
   yhat
y      0    1
  0 1732   68
  1  541  659
```

```
sum(diag(confusion_in))/sum(confusion_in)  # in-sample accuracy
```

```
[1] 0.797
```

# LPM: spam classification

## Out-of-sample performance:

```
phat_test_spam1 = predict(lm_spam1, spamtest)
yhat_test_spam1 = ifelse(phat_test_spam1 > 0.5, 1, 0)
confusion_out = table(y = spamtest$y, yhat = yhat_test_spam1)
confusion_out
```

```
   yhat
y     0   1
  0 372  13
  1  98 118
```

```
sum(diag(confusion_out))/sum(confusion_out)   # out-of-sample accuracy
```

```
[1] 0.8153078
```

# LPM: spam classification

How well are we doing? Note that 60% of the training set isn't spam:

```
table(spamfit$y)
```

```

   0    1
1800 1200
```

Thus “not spam” is the most likely outcome. So a reasonable baseline or “null model” is one that guesses “not spam” for every test-set instance.

# LPM: spam classification

How well does this null model perform on the test set? It's about 64\%, since it gets all the 0's right and all the 1's wrong:

```
table(spamtest$y)
```

```

  0   1 
385 216 
```

```
385/sum(table(spamtest$y))
```

```
[1] 0.640599
```

# LPM: spam classification

Our linear probability model had an 81.5% out-of-sample accuracy rate.

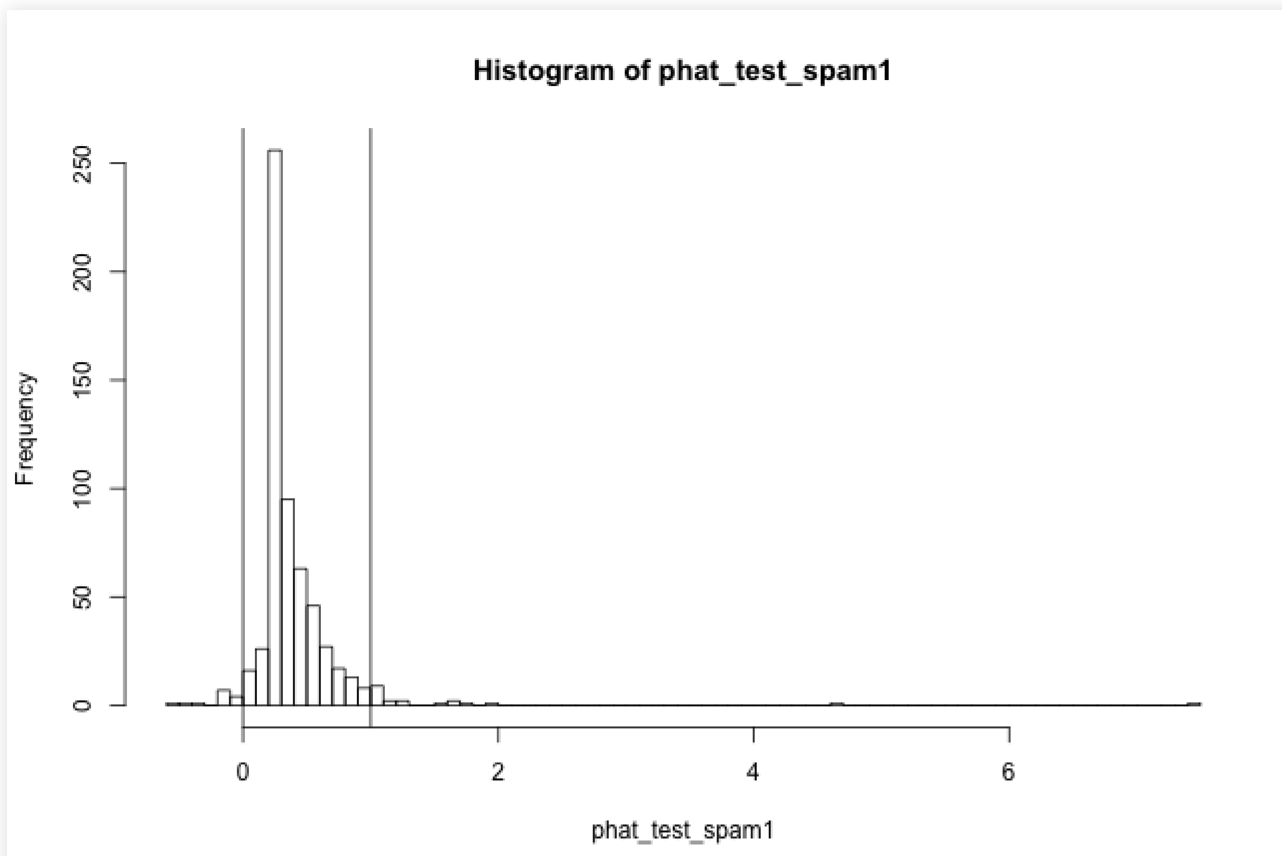
- Its absolute improvement over the null model is $\approx 81.5 - 64.1 = 17.4\%$.
- Its relative improvement, or *lift*, over the null model is $\approx 81.5/64.1 = 1.27$.

Comparing a model to a baseline or "null model" is often an important sanity check, especially in complicated problems.

- The null model might be one that knows nothing about $x$ (as here).
- OR it might be a very simple model.

# LPM: illegal probabilities

The linear probability model has one obvious problem: it can produce fitted probabilities that fall outside (0,1). E.g. here is a histogram of predicted probabilities for the spam test set, where 34/601 predictions (5.6%) have this problem:



# LPM: illegal probabilities

Recall the basic form of the linear probability model:

$$P(y = 1 \mid x) = x \cdot \beta$$

The core of the problem is this:

- the left-hand side needs to be constrained to fall between 0 and 1, by the basic rules of probabilities
- but the right-hand side is unconstrained – it can be any real number.

# Modifying the LPM

A natural fix to this problem is to break our model down into two pieces:

$$P(y = 1 \mid x) = g(x \cdot \beta)$$

The inner piece, $f(x) = x \cdot \beta$, is called the *linear predictor*. It maps features $x_i$ onto real numbers.

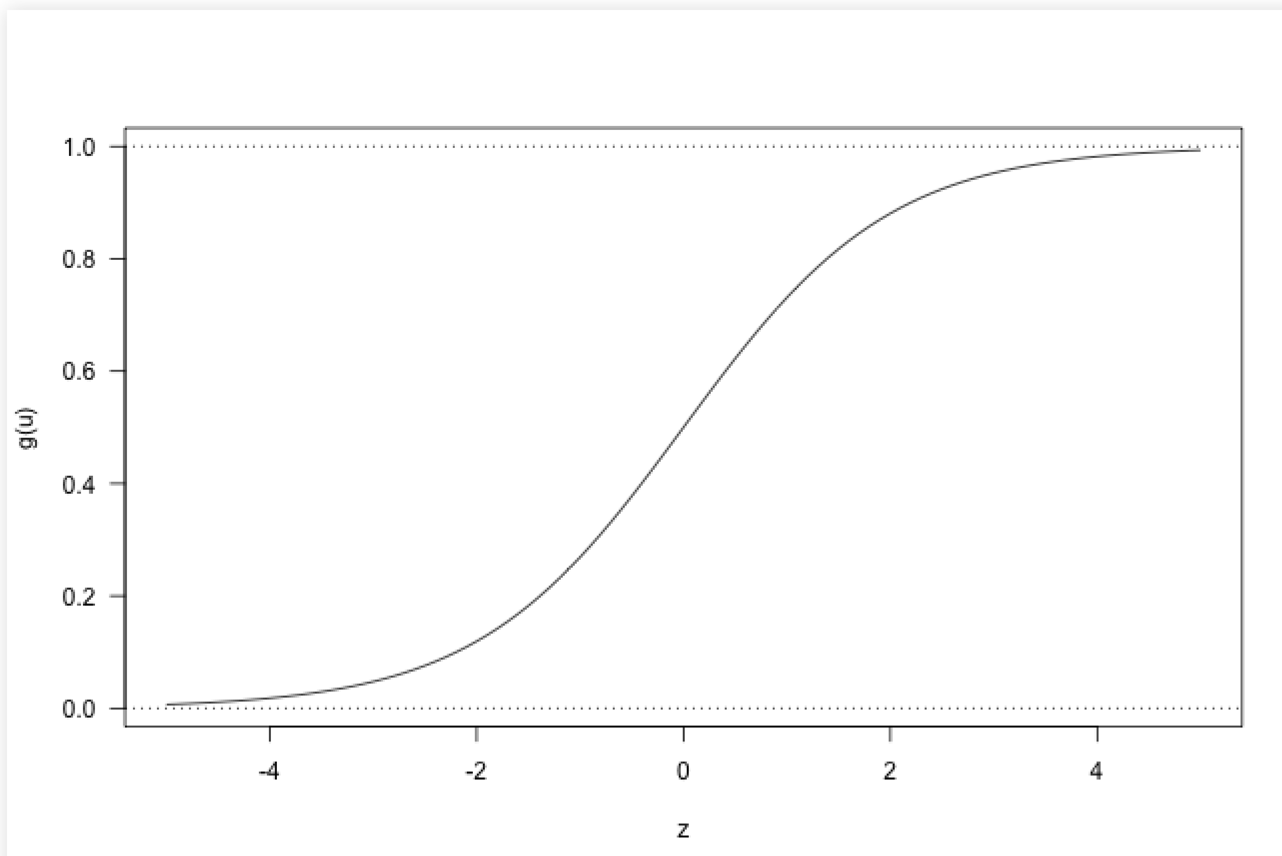
The outer piece, $g(z)$ is called a *link function*.

- It links the linear predictor $z_i \equiv f(x_i) = x_i \cdot \beta$ on the right to the probability on the left.
- It therefore must map real numbers onto the unit interval (0,1).

# Logistic regression

A standard choice is $g(z) = e^z/(1 + e^z)$.

- At $z = 0$, $g(z) = 0.5$.
- When $z \to \infty$, $g(z) \to 1$, and when $z \to \infty$, $g(z) \to 0$.



# Logistic regression

This is called the "logistic" or "logit" link, and it leads to the logistic regression model:

$$P(y = 1 \mid x) = \frac{\exp(x \cdot \beta)}{1 + \exp(x \cdot \beta)}$$

This is a very common choice of link function, for a couple of good reasons. One is interpretability: a little algebra shows that

$$\log\left[\frac{p}{1-p}\right] = x \cdot \beta$$

$$\frac{p}{1-p} = e^{x \cdot \beta}$$

so that it is a log-linear model for the *odds* of a yes outcome.

# Logistic regression is easy in R

```
glm(y ~ x, data=mydata, family=binomial)
```

`glm` stands for "generalized linear model," i.e. a linear model with a link function. The argument `family=binomial` tells R that $y$ is binary and defaults to the logit link.

The response can take several forms:

- `y = 0, 1, 1,...` numeric vector
- `y = FALSE, TRUE, TRUE,...` logical
- `y = 'not spam', 'spam', 'spam',...` factor

Everything else is the same as in linear regression!

# Logistic regression in your inbox

Let's fit a logit model to the spam data.

```
# Recall: the dot (.) says "use all variables not otherwise named"
logit_spam1 = glm(y ~ ., data=spamfit, family='binomial')
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
```

We're warned that some emails are clearly spam or not spam (i.e. p = 0 or p=1 up to floating-point numerical precision.) This `warning` is largely benign and isn't something to worry about.

# Interpeting coefficients in LR

```
coef(logit_spam1)
```

```
            (Intercept)           word.freq.remove
             -2.2381236                  5.7161306
        word.freq.order             word.freq.free
              0.9168758                  1.1018100
      word.freq.meeting               word.freq.re
             -3.9106839                 -0.3800485
         word.freq.edu        char.freq.semicolon
             -1.2632029                 -1.6941404
char.freq.exclamation capital.run.length.average
              2.2251096                  0.3420581
```

Recall our model is

$$\text{Odds} = \frac{p}{1-p} = e^{\beta_0} \cdot e^{\beta_1 x_1} \dots e^{\beta_p x_p}$$

So $e^{\beta_j}$ is an *odds multiplier* or *odds ratio* for for a one-unit increase in feature $x_j$.

# Interpeting coefficients in LR

```
coef(logit_spam1)
```

```
              (Intercept)           word.freq.remove
               -2.2381236                  5.7161306
          word.freq.order             word.freq.free
                0.9168758                  1.1018100
        word.freq.meeting               word.freq.re
               -3.9106839                 -0.3800485
            word.freq.edu        char.freq.semicolon
               -1.2632029                 -1.6941404
     char.freq.exclamation capital.run.length.average
                2.2251096                  0.3420581
```

The $\beta$ for `char.freq.free` is 1.1. So having an extra `free` in an e-mail multiplies odds of spam by $e^{1.1} \approx 3$.

# Interpeting coefficients in LR

```
coef(logit_spam1)
```

```
              (Intercept)          word.freq.remove 
               -2.2381236                 5.7161306 
          word.freq.order            word.freq.free 
                0.9168758                 1.1018100 
        word.freq.meeting              word.freq.re 
               -3.9106839                -0.3800485 
            word.freq.edu       char.freq.semicolon 
               -1.2632029                -1.6941404 
     char.freq.exclamation capital.run.length.average 
                2.2251096                 0.3420581 
```

The $\beta$ for `char.freq.semicolon` is -1.7. So having an extra semicolon in an e-mail multiplies odds of spam by $e^{-1.7} \approx 0.2$. (Down by a factor of five! Note to spammers: use more complex syntax.)

# Interpeting coefficients in LR

```
coef(logit_spam1)
```

```
            (Intercept)          word.freq.remove
             -2.2381236                 5.7161306
        word.freq.order            word.freq.free
              0.9168758                 1.1018100
      word.freq.meeting              word.freq.re
             -3.9106839                -0.3800485
          word.freq.edu       char.freq.semicolon
             -1.2632029                -1.6941404
char.freq.exclamation capital.run.length.average
              2.2251096                 0.3420581
```

The $\beta$ for `word.freq.remove` is 5.7. So having an extra `remove` in an e-mail multiplies odds of spam by $e^{5.7} \approx 300$.

Q: What is the odds multiplier for a coefficient of 0?

# LR for spam: out-of-sample

```
logit_spam = glm(y ~ ., data=spamfit, family='binomial')
phat_test_logit_spam = predict(logit_spam, spamtest, type='response')
yhat_test_logit_spam = ifelse(phat_test_logit_spam > 0.5, 1, 0)
confusion_out_logit = table(y = spamtest$y, yhat = yhat_test_logit_spam)
confusion_out_logit
```

```
   yhat
y     0   1
  0 358  27
  1  51 165
```

We did better!

- Error rate (51+27)/601 $\approx$ 13%, or accuracy of 87%.
- Absolute improvement over LPM: $87 - 81.5 = 6.5\%$.
- Lift over LPM: $87/81.5 \approx 1.07$.

# LR for spam: out-of-sample

We can take a slightly more nuanced look at the performance than simply calculating an overall accuracy/error rate. Three simple metrics you should know about:

- true positive rate (sensivity, recall)
- the false positive rate (specificity)
- the false discovery rate (precision, positive predictive value)

# LR for spam: true positive rate

The *true positive rate* (TPR): among spam e-mails ($y = 1$), how many are correctly flagged as spam ($\hat{y} = 1$)?

```
   yhat
y     0   1
  0 358  27
  1  51 165
```

Here the out-of-sample TPR is $165/(51 + 165) \approx 0.76$.

Synonyms for the TPR: sensitivity, recall.

# LR for spam: false positive rate

The *false positive rate* (FPR): among non-spam e-mails ($y = 0$), how many are wrongly flagged as spam ($\hat{y} = 1$)?

```
    yhat
y      0   1
  0 358  27
  1  51 165
```

Here the out-of-sample FPR is $27/(27 + 358) \approx 0.07$.

Synonyms: *specificity* is the opposite of FPR, but conveys same information:

$$\text{Specificity} = 1 - \text{FPR}$$

So this procedure had a 93% out-of-sample specificity.

# LR for spam: false discovery rate

The *false discovery rate* (FDR): among e-mails flagged as spam ($\hat{y} = 1$), how many were actually not spam ($y = 0$)?

```
    yhat
y      0   1
  0 358  27
  1  51 165
```

Here the out-of-sample FDR is $27/(27 + 165) \approx 0.14$.

Synonyms: The *precision/positive predictive value* is the opposite of FDR, but convey same information:

$$\text{Precision} = \text{Positive Predictive Value} = 1 - \text{FDR}$$

So this procedure had a 86% precision. Among flagged spam e-mails, 86% were actually spam.

# Who uses these terms?

All these synonyms for the same error rates can be a pain! But their usage tends to be field-dependent.

- FPR, FNR, FDR: statistics, machine learning
- Sensivity, specificity, positive predictive value: medicine, epidemiology, and public health
- Precision and recall: database and search engine design, machine learning, computational linguistics

Solution: always go back to the confusion matrix! It tells the whole story. Ironically, the confusion matrix *avoids confusion* over terminology.

# Estimating a logit model

A logistic regression model is fit by the principle of maximum likelihood: *choose the parameters so that the observed data looks as likely as possible.*

In LR, each outcome $y_i$ is binary. By assumption:

$$P(y_i = 1 \mid x_i) = \frac{e^{x \cdot \beta}}{1 + e^{x \cdot \beta}}$$

$$P(y_i = 0 \mid x_i) = 1 - \frac{e^{x \cdot \beta}}{1 + e^{x \cdot \beta}} = \frac{1}{1 + e^{x \cdot \beta}}$$

# Estimating a logit model

Recall that the likelihood function is the probability of the observed data as a function of the parameters. So we can write the likelihood contribution for observation $i$ as:

$$L_i(\beta) = \left(\frac{e^{x\cdot\beta}}{1+e^{x\cdot\beta}}\right)^{y_i} \cdot \left(\frac{1}{1+e^{x\cdot\beta}}\right)^{1-y_i}$$

If $y_i = 1$, the second term gets zeroed out. Similarly, if $y_i = 0$, the first term gets zeroed out.

# Estimating a logit model

The overall likelihood is then

$$L(\beta) = \prod_{i=1}^{N} L_i(\beta)$$

or on a log scale, to avoid numerical underflow:

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{N} \log L_i(\beta) \\ &= \sum_{i=1}^{N} \left[ y_i \cdot x_i \cdot \beta - \log(1 + e^{x \cdot \beta}) \right] \end{aligned}$$

This quantity can be maximized as a function of $\beta$ using an iterative numerical routine (typically Newton's method, sometimes gradient ascent or BFGS). Details for another course (feel free to ask me)!

# KNN for classification

Another approach to classification: back to K-nearest-neighbors.

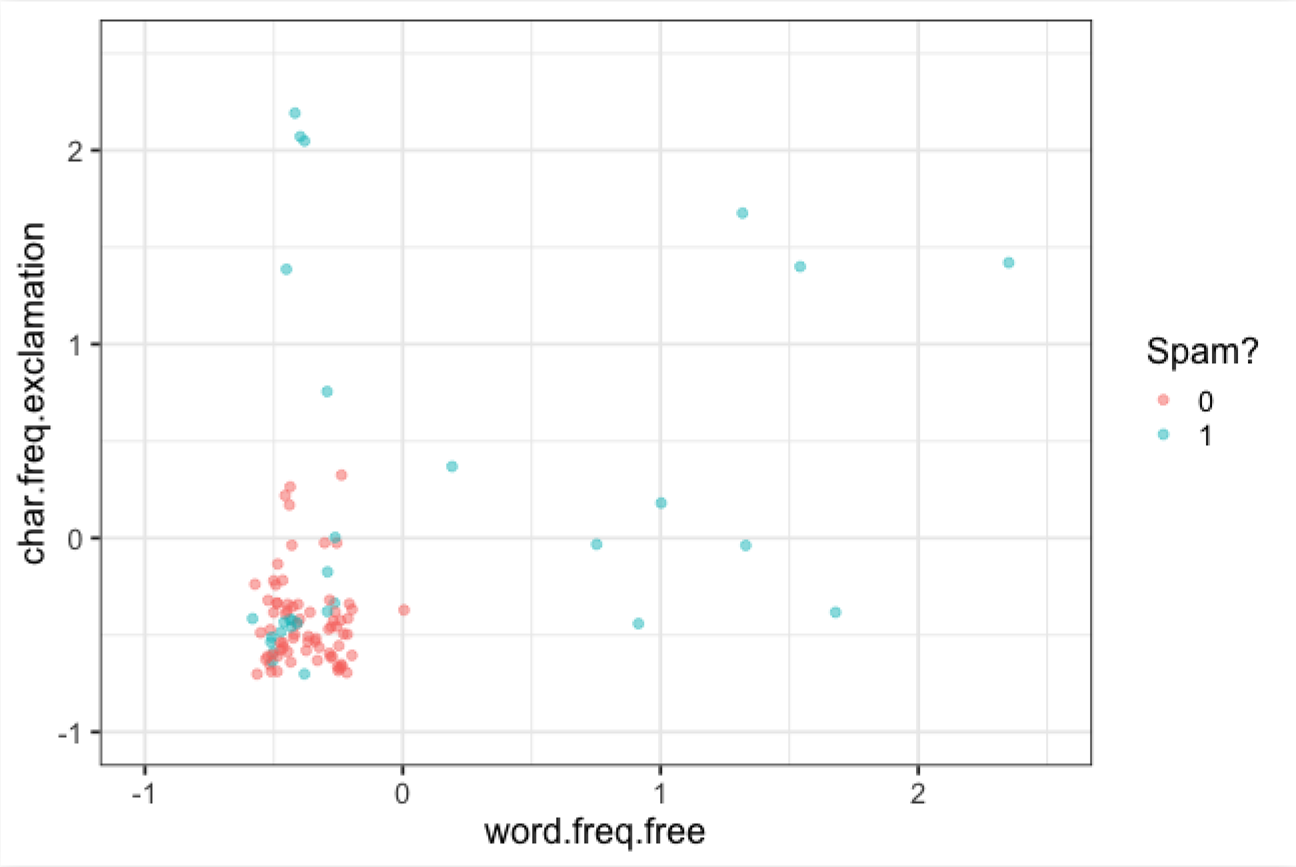
Super intuitive: what is the most common class around $x^\star$?

- Nearness measured using some metric, typically Euclidean distance:

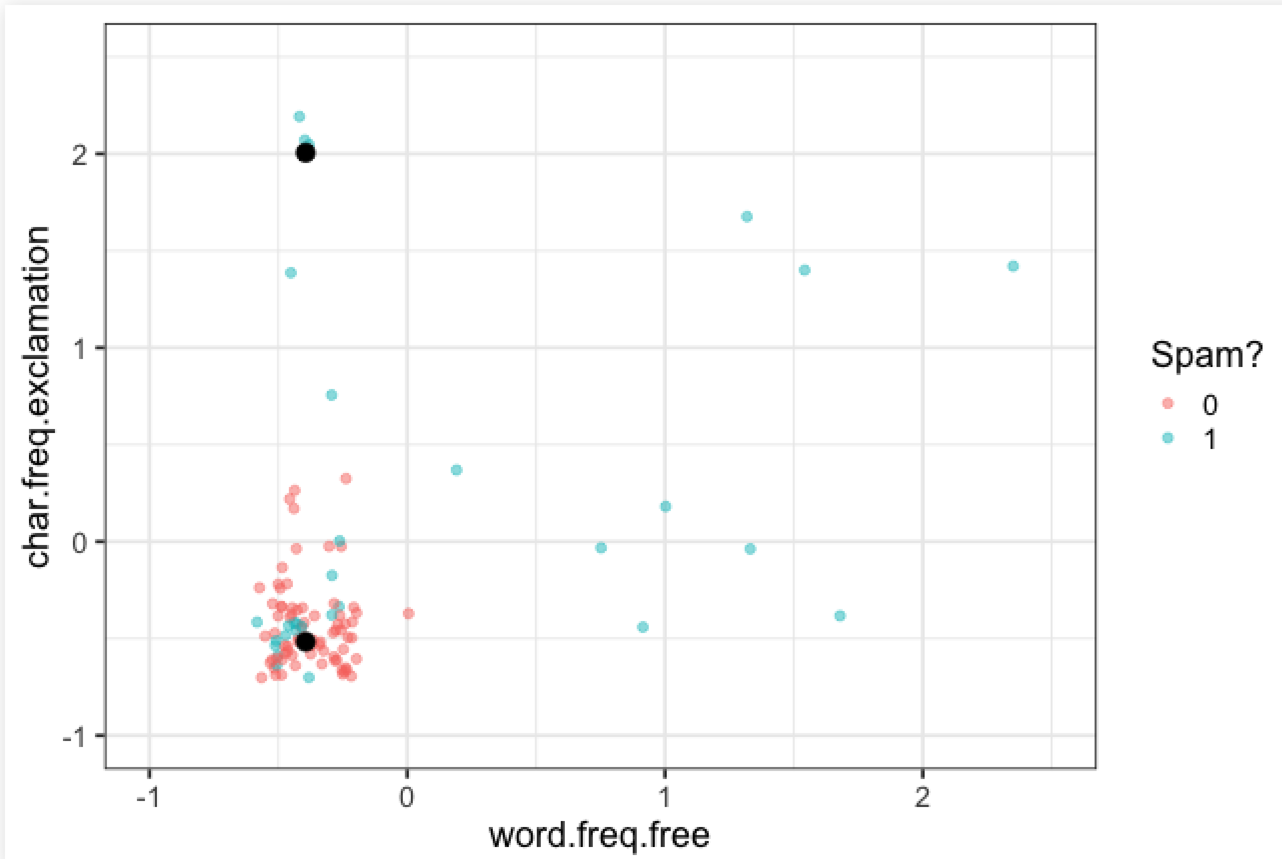
$$d(x, x') = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}$$

- Remember the importance of scaling your feature variables here! Typically we use distances scaled by $\mathrm{sd}(x_j)$ rather than raw distances.

# KNN for classification



# KNN for classification



# KNN for classification

In-class example: classifying glass shards for a recycling center

6 classes:

- WinF: float glass window
- WinNF: non-float window
- Veh: vehicle window
- Con: container (bottles)
- Tabl: tableware
- Head: vehicle headlamp

See `glass.R` on the class website!

# Limitations of KNN for classification

Nearest-neighbor classification is simple, but limited.

- There is no good way to choose K. Train/test splits work, but they are unstable: different data $\longrightarrow$ different K (perhaps *very* different).
- The classification can be very sensitive to K.
- All you get is a classification, with only rough probabilities. E.g. with $k = 5$, all probability estimates are multiple of 20%. Without accurate probabilities, it is hard to assess misclassification risk.
- But the basic idea is the same as in logistic regression: Observations with similar $x$'s should be classified similarly.

# Multinomial logistic regression

In logistic regression, we get binary class probabilities.

In multi-class problems, the response is one of $K$ categories. We'll encode this as $y_i = [0, 0, 1, \ldots, 0]$ where $y_{ik} = 1$ if response $i$ is in class $k \in \{1, \ldots, K\}$.

In multinomial logistic regression (MLR), we fit a model for

$$E(y_{ik} \mid x_i) = P(y_{ik} = 1 \mid x_i) = g(x_i \cdot \beta_k)$$

That is, we fit regression coefficients for *each class*.

# Multinomial logistic regression

In the MLR model, we construct this by analogy with the sigmoid link function (from binary LR) as follows:

$$\hat{p}_{ik} = P(y_{ik} = 1 \mid x_i) = \frac{e^{x_i \cdot \beta_k}}{\sum_{l=1}^{K} e^{x_i \cdot \beta_l}}$$

I like to think of this as each class vying to predict the outcome for $x_i$ as its own, via a "rate and normalize" procedure:

- each class "rates" $x_i$ as $e^{x_i \cdot \beta_k}$. The closer $x_i$ is to the class-specific regression coefficient $\beta_k$, the bigger this rating is.
- Ratings $\rightarrow$ probs: divide by the sum of the ratings across classes.
- This is often called the "softmax" function.

# Multinomial logit: glass example

```
library(nnet)
n = nrow(fgl); n_train = 180
train_ind = sample.int(214, n_train, replace=FALSE)
ml1 = multinom(type ~ RI + Mg, data=fgl[train_ind,])
```

```
# weights:  24 (15 variable)
initial  value 322.516704
iter  10 value 225.637114
iter  20 value 199.130574
iter  30 value 198.899834
final  value 198.899720
converged
```

```
coef(ml1)
```

```
      (Intercept)         RI          Mg
WinNF    5.459371 -0.2027355 -1.59091718
Veh     -1.016222 -0.1188378 -0.09264681
Con      7.444248 -0.4911691 -3.28404976
Tabl     6.571685 -0.5274920 -2.90523982
Head     8.190010 -0.6817354 -3.34953951
```

# Multinomial logit: glass example

Fitted class probabilities for the first five test-set examples:

```
predict(ml1, fgl[-train_ind,], type='probs') %>%
  head(5) %>%
  round(3)
```

```
    WinF WinNF   Veh   Con  Tabl  Head
1  0.788 0.080 0.132 0.000 0.000 0.000
18 0.719 0.166 0.114 0.001 0.001 0.000
19 0.575 0.285 0.129 0.003 0.004 0.004
26 0.443 0.401 0.121 0.008 0.013 0.014
27 0.438 0.411 0.116 0.008 0.013 0.014
```

# Multinomial logit: glass example

## How did we do?

```
y_test = fgl[-train_ind,'type']
yhat_test = predict(ml1, fgl[-train_ind,], type='class')
conf_mat = table(y_test, yhat_test)
conf_mat
```

```
       yhat_test
y_test  WinF WinNF Veh Con Tabl Head
  WinF     8     2   0   0    0    0
  WinNF    4     9   0   0    0    1
  Veh      1     1   0   0    0    0
  Con      0     1   0   0    0    0
  Tabl     0     1   0   0    0    1
  Head     0     0   0   0    0    5
```

```
sum(diag(conf_mat))/(n-n_train)
```

```
[1] 0.6470588
```

# Evaluating a classifier: deviance

In making decisions, both costs and probabilities matter. E.g. if $P(y = 1 \mid x) = 0.3$, how would you respond differently if:

- $x$ is word content of an e-mail and $y$ is spam status?
- $x$ is mammogram result and $y$ is breast cancer status?
- $x$ is DNA test and $y$ is guilty/not guilty?

Different kinds of errors may have different costs. Thus it helps to de-couple two tasks: *modeling probabilities accurately* and *making decisions*.

This requires that we evaluate the performance of a classifier in terms its *predicted probabilities*, not its *decisions about class labels*.

# Evaluating a classifier: likelihood

The natural way to do us is by calculating the *likelihood* for our model's predicted probabilities. Suppose that our classifier produces predicted probabilities $\hat{p}_{ik}$ for each response $i$ and class $k$. Then the likelihood is

$$\begin{aligned}\text{Like} &= \prod_{i=1}^{n} \prod_{l=1}^{K} \hat{p}_{il}^{y_{il}} \\ &= \prod_{i=1}^{n} \hat{p}_{i,k_i}\end{aligned}$$

where $k_i$ is the observed class label for case $i$.

To get from the first to the second lines, notice that $y_{il} = 1$ for $l = k_i$, and zero otherwise.

# Evaluating a classifier: log likelihood

On a log scale, this becomes

$$\text{loglike} = \sum_{i=1}^{n} \log \hat{p}_{i,k_i}$$

In words: we sum up our model's predicted log probabilities for the outcomes $y_{i,k_i}$ that actually happened.

As with everything in statistical learning: we can calculate an in-sample or a out-of-sample log likelihood, and the out-of-sample is more important!

Q: what's the largest possible log likelihood for a classifier?

# Evaluating a classifier: deviance

Sometimes we quote a model's *deviance* instead of its log likelihood. The relationship is simple:

$$\text{deviance} = -2 \cdot \text{loglike}$$

Log likelihood measures *fit* (which we want to maximize), deviance measures *misfit* (which we want to minimize).

So the negative sign makes sense. But why the factor of 2? *Because of the analogy because least squares and the normal distribution.*

# Evaluating a classifier: deviance

Remember back to an ordinary regression problem with normally distributed errors, $y_i \sim N(f(x_i), \sigma^2)$:

$$\text{Like} = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2}(y_i - f(x_i))^2 \right\}$$

On a log scale, up to a constant not involving $f(x)$, this becomes:

$$\text{loglike} \propto -\frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i))^2 = -\text{RSS}/2$$

where RSS = residual sums of squares.

Deviance generalizes the notion of "residual sums of squares" to non-Gaussian models.

# Bayes' Rule for classification

Recall Bayes' rule:

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}$$

You might remember that each of these terms has a name:

- $P(A)$: the prior probability
- $P(A \mid B)$ : the posterior probability
- $P(B \mid A)$: the likelihood
- $P(B)$: the marginal (total/overall) probability

In classification, “A” is a class label and “B” is a set of features.

# Bayes' Rule for classification

Bayes's rule:

$$P(y = k \mid x) = \frac{P(y = k) \cdot P(x \mid y = k)}{P(x)}$$

$P(y = k)$ is the prior probability for class $k$. We usually get this from the raw class frequencies in the training data. For example:

```
table(fgl[train_ind,]$type) %>% prop.table
```

```

     WinF      WinNF        Veh        Con       Tabl       Head 
0.33333333 0.34444444 0.08333333 0.06666667 0.03888889 0.13333333 
```

# Bayes' Rule for classification

Bayes's rule:

$$P(y = k \mid x) = \frac{P(y = k) \cdot P(x \mid y = k)}{P(x)}$$

$P(x)$ is the marginal probability of observing feature vector $x$. Notice it doesn't depend on $k$! It's the same number for all classes.

Thus we usually write the posterior probabilities up to this constant of proportionality, without bothering to compute it:

$$P(y = k \mid x) \propto P(y = k) \cdot P(x \mid y = k)$$

(Note: often we do the actual computations on a log scale instead.)

# Bayes' Rule for classification

Bayes's rule:

$$P(y = k \mid x) = \frac{P(y = k) \cdot P(x \mid y = k)}{P(x)}$$

The hard part is estimating the likelihood $P(x \mid y = k)$. In words: how likely is it that we would have observed feature vector $x$ if the true class label were $k$?

This is like regression in reverse! See `congress109_bayes.r` for a teaser example.

# Naive Bayes

Recall that $x = (x_1, x_2, \ldots, x_p)$ is a vector of $p$ features. Our first strategy for estimating $P(x \mid y = k)$ is called "Naive Bayes."

It's "naive" because we make the simplifying assumption that *every feature* $x_j$ *is independent* of all other features:

$$
\begin{aligned}
P(x \mid y = k) &= P(x_1, x_2, \ldots, x_p \mid y = k) \\
&= \prod_{j=1}^{p} P(x_j \mid y = k) \quad \text{(independence)}
\end{aligned}
$$

This simplifies the requirements of the problem: *just calculate the marginal distribution of the features,* i.e. $P(x_j \mid y = k)$ for all features $j$ and classes $k$.

# Naive Bayes: a small example

In `congress109.csv` we have data on all speeches given on the floor of the U.S. Congress during the 109th Congressional Session (January 3, 2005 to January 3, 2007).

Every row is a set of *phrase counts* associated with a single representative's speeches across the whole session. $X_{ij}$ = number of times that rep $i$ utter phrase $j$ during a speech.

The target variable $y \in \mathrm{R}, \mathrm{D}$ is the party affiliation of the representative.

# Naive Bayes: a small example

```r
# read in data
congress109 = read.csv("../data/congress109.csv", header=TRUE, row.names=1)
congress109members = read.csv("../data/congress109members.csv",
header=TRUE, row.names=1)
```

## Focus on a few key phrases and a few famous pols:

```r
X_small = dplyr::select(congress109, minimum.wage, war.terror, tax.relief,
hurricane.katrina)
X_small[c('John McCain', 'Mike Pence', 'John Kerry', 'Edward Kennedy'),]
```

```
               minimum.wage war.terror tax.relief hurricane.katrina
John McCain               0         27          0                14
Mike Pence                0         12          1                11
John Kerry               12         16         13                23
Edward Kennedy          260          8          1                53
```

# Naive Bayes: a small example

Let's look at these counts summed across all members in each party:

```
y = congress109members$party

# Sum phrase counts by party
R_rows = which(y == 'R')
D_rows = which(y == 'D')
colSums(X_small[R_rows,])
```

```
     minimum.wage          war.terror         tax.relief hurricane.katrina
              294                 604                497               717
```

```
colSums(X_small[D_rows,])
```

```
     minimum.wage          war.terror         tax.relief hurricane.katrina
              767                 237                176              1295
```

So we get the sense that some phrases are “more Republican” and some “more Democrat.”

# Naive Bayes: a small example

To make this precise, let's build a simplified "bag of phrases" model for a Congressional speech:

- Imagine that every phrase uttered in a speech is a random sample from a "bag of phrases," where each phrase has its own probability. (*This is the Naive Bayes assumption of independence*.)
- Here the bag consists of just four phrases: "minimum wage", "war on terror", "tax relief," and "hurricane katrina".
- Each class (R or D) has its own probability vector associated with the phrases in the bag.

# Naive Bayes: a small example

We can estimate these probability vectors for each class from the phrase counts in the training data. For Republicans:

```
probhat_R = colSums(X_small[R_rows,])
probhat_R = probhat_R/sum(probhat_R)
probhat_R
```

```
     minimum.wage          war.terror         tax.relief hurricane.katrina
        0.1392045           0.2859848          0.2353220         0.3394886
```

And for Democrats:

```
probhat_D = colSums(X_small[D_rows,])
probhat_D = probhat_D/sum(probhat_D)
probhat_D
```

```
     minimum.wage          war.terror         tax.relief hurricane.katrina
       0.30989899          0.09575758         0.07111111        0.52323232
```

# Naive Bayes: a small example

Let's now look at some particular member of Congress and try to build the “likelihood” for his or her phrase counts

```
X_small['Sheila Jackson-Lee',]
```

```
                   minimum.wage war.terror tax.relief hurricane.katrina
Sheila Jackson-Lee           11         15          3                66
```

Are Sheila Jackon-Lee's phrase counts $x = (11, 15, 3, 66)$ more likely under the Republican or Democrat probability vector?

# Naive Bayes: a small example

Recall the Republican vector:

```
    minimum.wage       war.terror       tax.relief hurricane.katrina
       0.1392045        0.2859848        0.2353220         0.3394886
```

Under this probability vector:

$$
\begin{aligned}
P(x \mid y = \mathrm{R}) &= P(x_1 = 11 \mid y = \mathrm{R}) \\
&\times P(x_2 = 15 \mid y = \mathrm{R}) \\
&\times P(x_3 = 3 \mid y = \mathrm{R}) \\
&\times P(x_4 = 66 \mid y = \mathrm{R}) \\
&= (0.1392)^{11} \cdot (0.2860)^{15} \cdot (0.2353)^{3} \cdot (0.3395)^{66} \\
&= 3.765 \times 10^{-51}
\end{aligned}
$$

# Naive Bayes: a small example

Now recall the Democratic vector:

```
     minimum.wage         war.terror         tax.relief hurricane.katrina
       0.30989899         0.09575758         0.07111111        0.52323232
```

Under this probability vector:

$$
\begin{aligned}
P(x \mid y = \mathrm{D}) &= P(x_1 = 11 \mid y = \mathrm{D}) \\
&\times P(x_2 = 15 \mid y = \mathrm{D}) \\
&\times P(x_3 = 3 \mid y = \mathrm{D}) \\
&\times P(x_4 = 66 \mid y = \mathrm{D}) \\
&= (0.3099)^{11} \cdot (0.0958)^{15} \cdot (0.0711)^{3} \cdot (0.5232)^{66} \\
&= 1.293 \times 10^{-43}
\end{aligned}
$$

# Naive Bayes: a small example

Because these numbers are so tiny, it's much safer to work on a log scale:

$$\log P(x \mid y = k) = \sum_{j=1}^{p} x_j \log p_j^{(k)}$$

where $p_j^{(k)}$ is the jth entry in the probability vector for class $k$.

```
x_try = X_small['Sheila Jackson-Lee',]
sum(x_try * log(probhat_R))
```

```
[1] -116.1083
```

```
sum(x_try * log(probhat_D))
```

```
[1] -98.75633
```

# Naive Bayes: a small example

Let's use Bayes' rule (posterior $\propto$ prior times likelihood) to put this together with our prior, estimated using the empirical class frequencies:

```
table(y) %>% prop.table
```

```
y
          D           I           R 
0.457466919 0.003780718 0.538752363 
```

So:

$$P(R \mid x) \propto 0.539 \cdot (3.765 \times 10^{-51})$$

and

$$P(D \mid x) \propto 0.457 \cdot (1.293 \times 10^{-43})$$

# Naive Bayes: a small example

- Turn this into a set of probabilities by normalizing, i.e. dividing by the sum across all classes:

$$
\begin{aligned}
P(D \mid x) &= \frac{0.457 \cdot (1.293 \times 10^{-43})}{0.457 \cdot (1.293 \times 10^{-43} + 0.539 \cdot (3.765 \times 10^{-51})} \\
&\approx 1
\end{aligned}
$$

- So:

1. Sheila Jackson-Lee is probably a Democrat, according to our model.
2. The data completely overwhelm the prior! This is often the case in Naive Bayes models.

# Naive Bayes: a bigger example

Turn to `congress109_bayes.R` to see a larger example of Naive Bayes classification, where we fit our model with all 1000 phrase counts.

# Naive Bayes: summary

- Works by directly modeling $P(x \mid y)$, versus $P(y \mid x)$ as in logit.
- Simple and easy to compute, and therefore scalable to very large data sets and classification problems.
- Works even more with feature variables $P$ than observations $N$.
- Often too simple: the "naive" assumption of independence really is a drastic simplification.
- The resulting probabilities are useful for classification purposes, but often not believeable as probabilities.
- Most useful when the features $x$ are categorical variables (like phrase counts!) Very common in text analysis.

# Linear discriminant analysis

Linear discriminant analysis (LDA) has a similar motivation to Naive Bayes.

$$P(y = k \mid x) \propto p(y = k) \cdot p(x \mid y = k)$$

There are two key differences:

- LDA relaxes the assumption of independence.
- We explicitly model the multivariate *joint distibution* for vector $x$ as a multivariate normal distribution:
  $p(x \mid y = k) = (2\pi)^{-p/2} \cdot |\Sigma_k|^{-1/2} \exp\left\{(x - \mu_k)'\Sigma_k^{-1}(x - \mu_k)\right\}$
  Written more concisely: $(x \mid y = k) \sim N(\mu_k, \Sigma_k)$, where $(\mu_k, \Sigma_k)$ are the mean vector and covariance matrix for class $k$.
- See `glass_LDA.R`