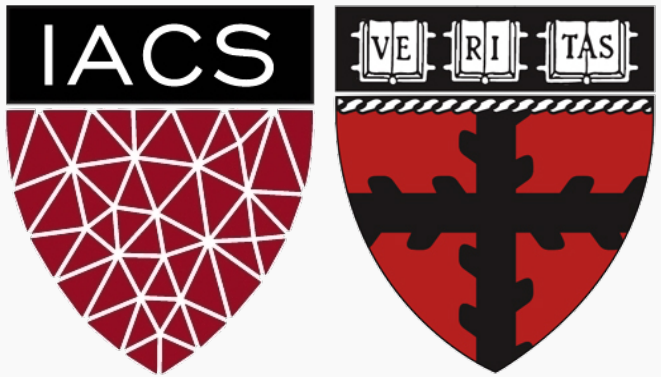
Lecture 11: Logistic Regression Extensions and Evaluating Classification

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Tanner



Announcements

Lecture Outline

- Logistic Regression: a Brief Review
- Classification Boundaries
- Regularization in Logistic Regression
- Multinomial Logistic Regression
- Bayes Theorem and Misclassification Rates
- ROC Curves

Multiple Logistic Regression: the model

Last time we saw the general form of the multiple logistic regression model. Specifically we can define a multiple logistic regression model to predict $P(Y = 1)$ as such:

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

where there are p predictors: $\boldsymbol{X} = (X_1, X_2, \ldots, X_p)$.

Note: statisticians are often lazy and use the notation "log" to mean "ln" (the text does this). We will write $\log_{10}$ if this is what we mean.

Multiple Logistic Regression: Estimation

The model parameters are estimated using ***likelihood theory***. That is the model assumes $Y_i \sim Bern\left(p_i = \frac{1}{1+e^{-(\beta_0+\beta_1 X_{1,i}+\cdots+\beta_p X_{p,i})}}\right)$. Then the ***log-likelihood function*** is maximized to get the $\hat{\beta}$'s:

$$\ln\left[L(\vec{\beta}|Y)\right] = \ln\left[\prod_i (p_i)^{y_i}(1-p_i)^{1-y_i}\right]$$

$$= \ln\left[\prod_i (p_i)^{y_i}(1-p_i)^{1-y_i}\right] = \sum_i \left[y_i \ln(p_i) + (1-y_i)\ln(1-p_i)\right]$$

$$= -\sum_i \left[y_i \ln\left(1+e^{-(\beta_0+\beta_1 X_1+\cdots+\beta_p X_p)}\right) + (1-y_i)\ln\left(1+e^{(\beta_0+\beta_1 X_1+\cdots+\beta_p X_p)}\right)\right]$$

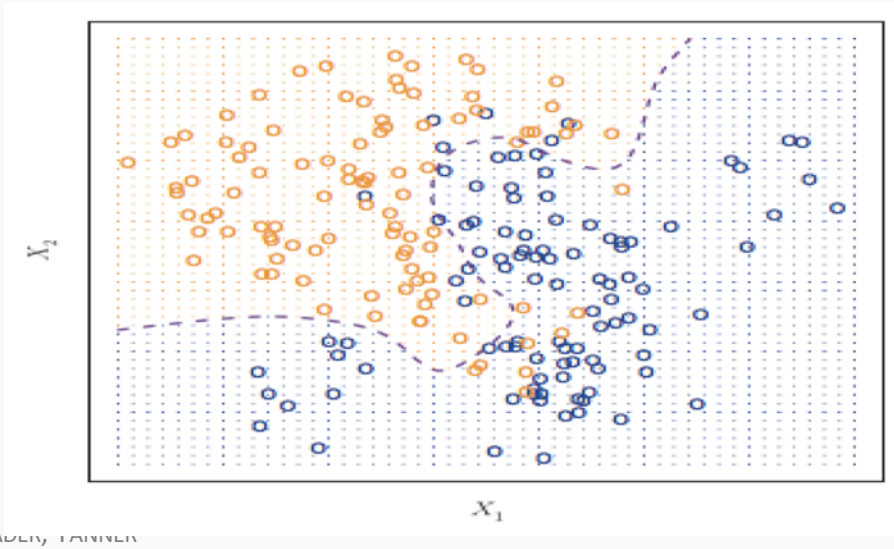
Classification Boundaries

Classification boundaries

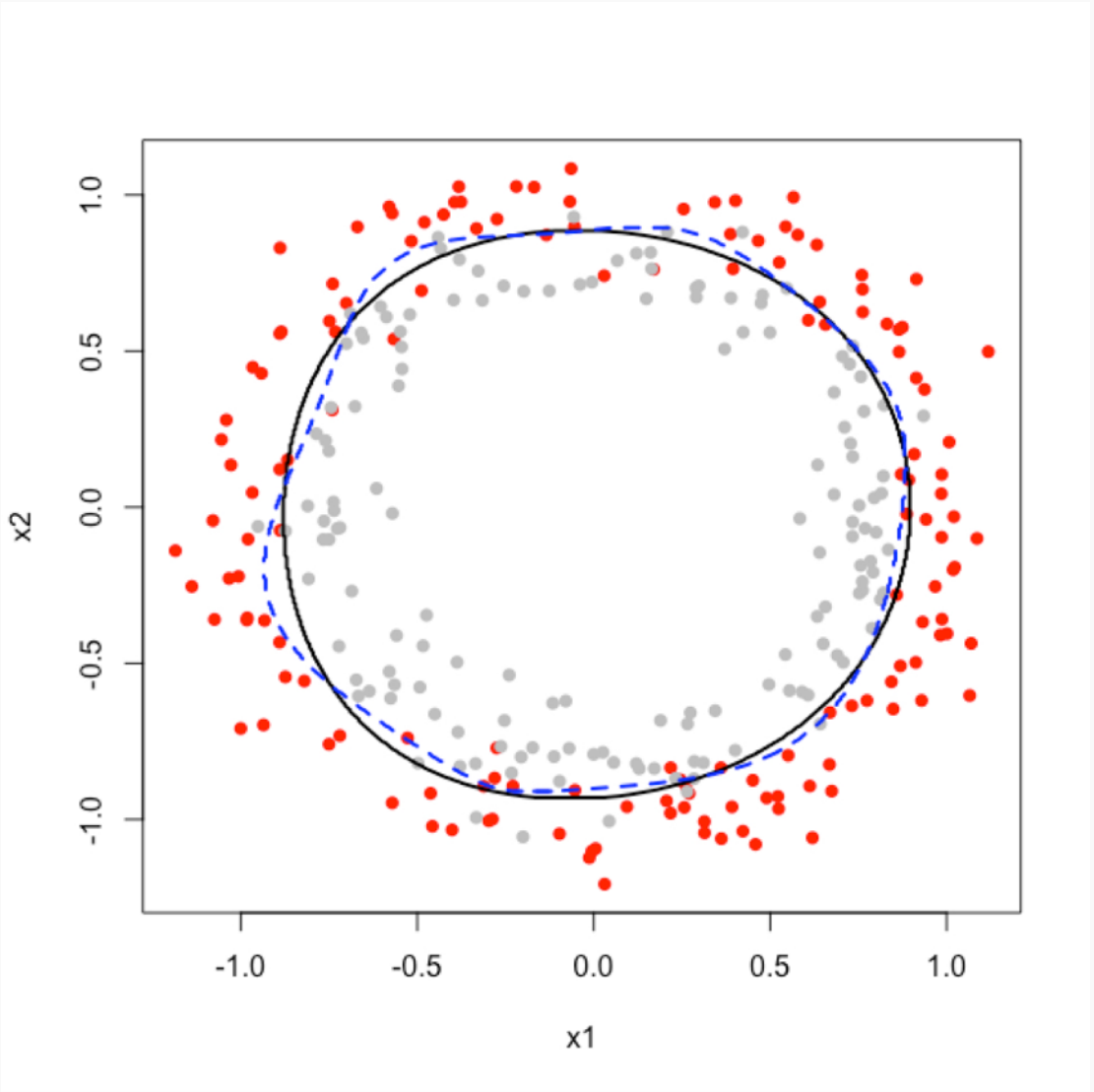
Recall that we could attempt to purely classify each observation based on whether the estimated $P(Y = 1)$ from the model was greater than 0.5.

When dealing with 'well-separated' data, logistic regression can work well in performing classification.

We saw a 2-D plot last time which had two predictors, X_1, X_2 and depicted the classes as different colors. A similar one is shown on the next slide.



2D Classification in Logistic Regression: an Example



2D Classification in Logistic Regression: an Example

Would a logistic regression model perform well in classifying the observations in this example?

What would be a good logistic regression model to classify these points?

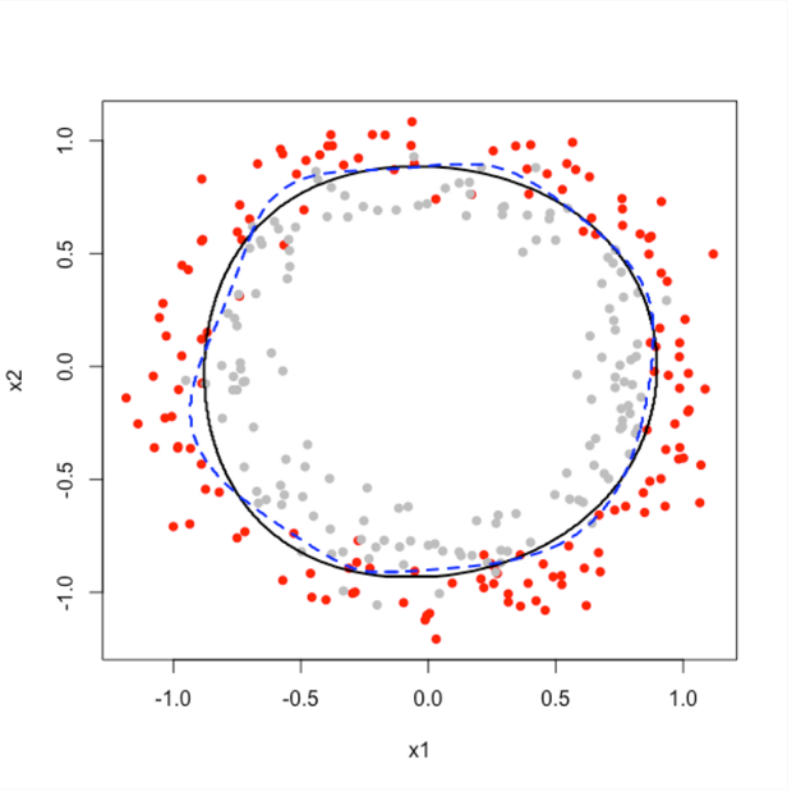
Based on these predictors, two separate logistic regression model were considered that were based on different ordered polynomials of X_1, X_2 and their interactions. The 'circles' represent the boundary for classification.

How can the classification boundary be calculated for a logistic regression?

2D Classification in Logistic Regression: an Example

In this plot, which classification boundary performs better? How can you tell? How would you make this determination in an actual data example?

We could determine the misclassification rates in left out validation or test set(s).



Regularization in Logistic Regression

Review: Regularization in Linear Regression

Based on the Likelihood framework, a loss function can be determined based on the likelihood function.

We saw in linear regression that maximizing the log-likelihood is equivalent to minimizing the sum of squares error:

$$\arg\min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \arg\min \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + ... + \beta_p x_{pi}))^2$$

Review: Regularization in Linear Regression

And a regularization approach was to add a penalty factor to this equation. Which for Ridge Regression becomes:

$$\arg\min \left[\sum_{i=1}^{n} \left(y_i - \left(\beta_0 + \sum_{j=1}^{n} \beta_j x_{ji} \right) \right)^2 + \lambda \sum_{j=1}^{n} \beta_j^2 \right]$$

Note: this penalty *shrinks* the estimates towards zero, and had the analogue of using a Normal prior centered at zero in the Bayesian paradigm.

Loss function in Logistic Regression

A similar approach can be used in logistic regression. Here, maximizing the log-likelihood is equivalent to minimizing the following loss function:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} (y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)) \right]$$

where $p_i = \frac{1}{1-e^{-(\beta_0+\beta_1 X_{1,i}+\cdots+\beta_p x_{p,i})}}$

Why is this a good loss function to minimize? Where does this come from?

The log-likelihood for independent $Y_i \sim \text{Bern}(p_i)$.

Regularization in Logistic Regression

A penalty factor can then be added to this loss function and results in a new loss function that penalizes large values of the parameters:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} (y_i \ln(p_i) + (1-y_i)\ln(1-p_i)) + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

The result is just like in linear regression: shrink the parameter estimates towards zero.

In practice, the intercept is usually not part of the penalty factor.

Note: the sklearn package uses a different tuning parameter: instead of λ they use a constant that is essentially $C = \frac{1}{\lambda}$.

Regularization in Logistic Regression: an Example

Let's see how this plays out in an example in logistic regression.

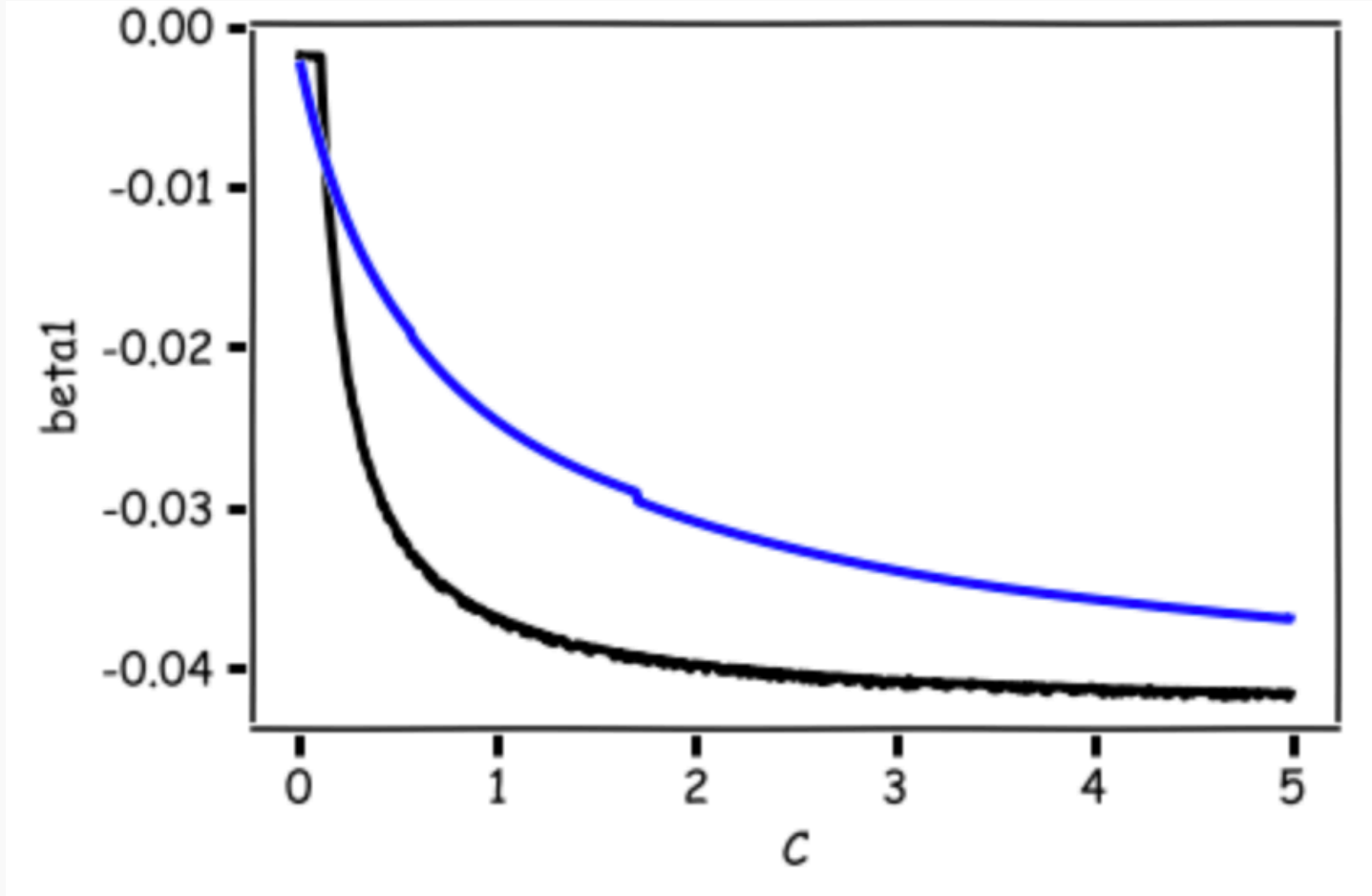
```
beta1_l1 = []
beta1_l2 = []
Cs = []
data_x = df_heart[['MaxHR']]
data_y = df_heart['AHD']

for i in range(1, 500):
    C = i/100
    logitm_l1 = sk.LogisticRegression(C = C, penalty = "l1")
    logitm_l1.fit (data_x, data_y)
    logitm_l2 = sk.LogisticRegression(C = C, penalty = "l2")
    logitm_l2.fit (data_x, data_y)
    beta1_l1.append(logitm_l1.coef_[0])
    beta1_l2.append(logitm_l2.coef_[0])
    Cs.append(C)

plt.plot(Cs, beta1_l1,  color='black', lw=3)
plt.plot(Cs, beta1_l2,  color='blue', lw=3)
plt.xlabel ("C")
plt.ylabel("beta1")
plt.show()
```

Regularization in Logistic Regression: an Example

Let's see how this plays out in an example in logistic regression.



Regularization in Logistic Regression: tuning λ

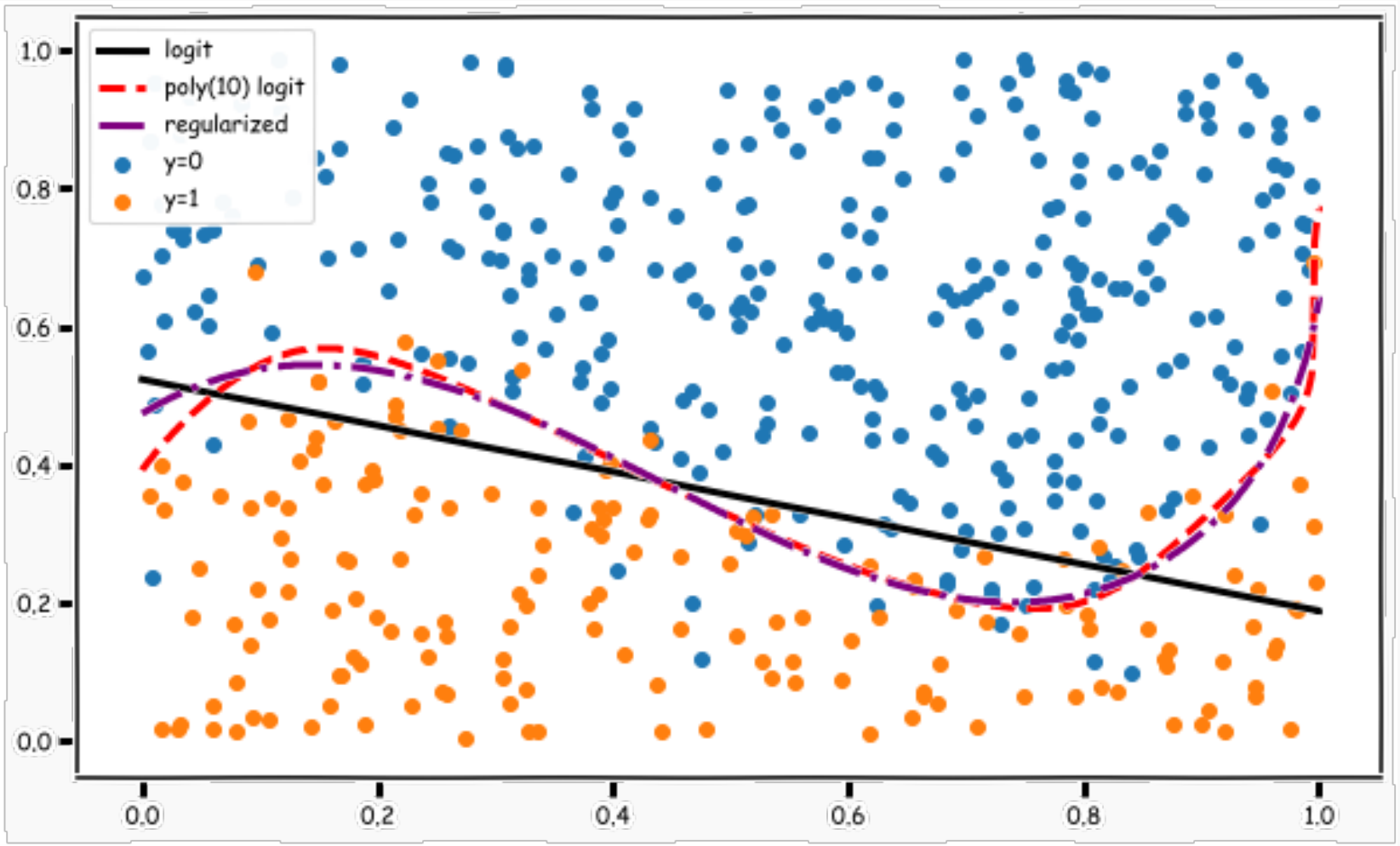
Just like in linear regression, the shrinkage factor must be chosen. How should we go about doing this?

Through building multiple training and test sets (through *k*-fold or random subsets), we can select the best shrinkage factor to mimic out-of-sample prediction.

How could we measure how well each model fits the test sets?

We should measure this based on some loss function!

Regularized Decision Boundaries



Multinomial Logistic Regression

Logistic Regression for predicting more than 2 Classes

There are several extensions to standard logistic regression when the response variable *Y* has more than 2 categories. The two most common are:

1. ordinal logistic regression
2. multinomial logistic regression.

Ordinal logistic regression is used when the categories have a specific hierarchy (like class year: Freshman, Sophomore, Junior, Senior; or a 7-point rating scale from strongly disagree to strongly agree).

Multinomial logistic regression is used when the categories have no inherent order (like eye color: blue, green, brown, hazel, et...).

Multiclass Logistic Regression

There are two common approaches to estimating a nominal (not-ordinal) categorical variable that has more than 2 classes. The first approach sets one of the categories in the response variable as the *reference* group, and then fits separate logistic regression models to predict the other cases based off of the reference group. For example we could attempt to predict a student's concentration:

$$y = \begin{cases} 1 & if \text{ Computer Science (CS)} \\ 2 & if \text{ Statistics} \\ 3 & \text{otherwise} \end{cases}$$

from predictors x_1 number of psets per week and x_2 how much time spent in Lamont Library.

Multiclass Logistic Regression (cont.)

We could select the $y = 3$ case as the reference group (other concentration), and then fit two separate models: a model to predict $y = 1$ (CS) from $y = 3$ (others) and a separate model to predict $y = 2$ (Stat) from $y = 3$ (others).

Ignoring interactions, how many parameters would need to be estimated?

How could these models be used to estimate the probability of an individual falling in each concentration?

One vs. Rest (ovr) Logistic Regression

The default multiclass logistic regression model is called the 'One vs. Rest' approach, which is our second method.

If there are 3 classes, then 3 separate logistic regressions are fit, where the probability of each category is predicted over the rest of the categories combined. So for the concentration example, 3 models would be fit:

- a first model would be fit to predict CS from (Stat and Others) combined.
- a second model would be fit to predict Stat from (CS and Others) combined.
- a third model would be fit to predict Others from (CS and Stat) combined.

An example to predict ECG results from the Heart data to follow…

OVR Logistic Regression in Python

In [293]:

Slide Type

```
# 'Multinomial' Logistic Regression Example

data_x = df_heart[['Sex']]

# 0 = normal; 1 = having ST-T; 2 = hypertrophy
data_y = df_heart['RestECG']

logitm = LogisticRegression(C = 10000000,solver='lbfgs',multi_class='ovr')
logitm.fit(data_x, data_y)

# The coefficients
print('Estimated beta1: \n', logitm.coef_)
print('Estimated beta0: \n', logitm.intercept_)
```

```
Estimated beta1:
 [[-0.04003728]
 [-1.87833213]
 [ 0.1445807 ]]
Estimated beta0:
 [ 0.02061932 -3.44468483 -0.14458062]
```

‘True’ Multinomial Logistic Regression

Another option for multiclass a logistic regression model is the *true* ‘multinomial’ logistic regression model.

One of the classes is chosen as the *baseline group* (think $Y = 0$ group in typicaly logistic regression), and the other $K - 1$ classes are compared to it. Thus a sequence or models are built to predict being in class k from class K:

$$\ln\left(\frac{P(Y=k)}{P(Y=K)}\right) = \beta_{0,k} + \beta_{1,k}X_1 + \cdots + \beta_{p,k}X_p$$

Note: sklearn normalizes the calculations by adding a K^{th} set of estimates in the output.

Multinomial Logistic Regression in Python

In [294]:

Slide Type

```
logitm2 = LogisticRegression(C = 10000000,solver='lbfgs',multi_class='multinomial')
logitm2.fit (data_x, data_y)

# The coefficients
print('Estimated beta1: \n', logitm2.coef_)
print('Estimated beta0: \n', logitm2.intercept_)
```

```
Estimated beta1:
 [[ 0.57896717]
 [-1.25284168]
 [ 0.67387452]]
Estimated beta0:
 [ 0.95944398 -1.83373675  0.87429277]
```

‘multinomial’ vs. ‘ovr’

There are 2 options in sklearn, so which should we use?

‘multinomial’ is slightly more efficient in estimation since there technically are fewer parameters (though sklearn reports extra ones to normalize the calculations to 1) and is more suitable for inferences/group comparisons.

‘ovr’ is often preferred for determining classification: you simply just predict from all 3 separate models (for each individual) and choose the highest probability.

They give VERY similar results in estimated probabilities and classifications.

Classification for more than 2 Categories

When there are more than 2 categories in the response variable, then there is no guarantee that $P(Y = k) \geq 0.5$ for any one category. So any classifier based on logistic regression will instead have to select the group with the largest estimated probability.

The classification boundaries are then much more difficult to determine. We will not get into the algorithm for drawing these in this class.

Softmax

make sure that its a probability (aka beteen 0 and 1), so standardize accordingly; softmax takes care about it

So how do we convert a set of probability estimates from separate models to one set of probability estimates?

The ***softmax*** function is used. That is, the weights are just normalized for each predicted probability. AKA, predict the 3 class probabilities from each model of the 3 models, and just rescale so they add up to 1.

Mathematically that is:

$$P(y = k|\vec{x}) = \frac{e^{\vec{x}^T \hat{\vec{\beta}}_k}}{\sum_{j=1}^{K} e^{\vec{x}^T \hat{\vec{\beta}}_j}}$$

where $\vec{x}$ is the vector of covariates for that observation and $\hat{\vec{\beta}}_k$ are the associated logistic regression coefficient estimates.

Classification for more than 2 Categories in sklearn

In [307]:

Slide Type

```
X=np.arange(0,2)
print("For OVR Logistic Regression:")
print(logitm.predict_proba(X.reshape(-1,1)))
print(logitm.predict(X.reshape(-1,1)))

print("For Multinomial Logistic Regression:")
print(logitm2.predict_proba(X.reshape(-1,1)))
print(logitm2.predict(X.reshape(-1,1)))
```

```
For OVR Logistic Regression:
[[0.5051546  0.03092776 0.46391764]
 [0.49514565 0.00485434 0.50000001]]
[0 2]
For Multinomial Logistic Regression:
[[0.50515267 0.03092856 0.46391877]
 [0.49514568 0.00485429 0.50000003]]
[0 2]
```

Bayes Theorem and Misclassification Rates

Bayes' Theorem

We define conditional probability as:

$$P(B|A) = P(B \; and \; A)/P(A)$$

And using the fact that $P(B \; and \; A) = \; P(A|B)P(B)$ we get the simplest form of Bayes' Theorem:

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

Another version of Bayes' Theorem is found by substituting in the Law of Total Probability (LOTP) into the denominator:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A|B)P(B) + P(A|B^C)P(B^C)}$$

Diagnostic Testing

In the diagnostic testing paradigm, one cares about whether the results of a test (like a classification test) matches truth (the true class that observation belongs to). The simplest version of this is trying to detect disease ($D+$ vs. $D-$) based on a diagnostic test ($T+$ vs. $T-$).

Medical examples of this include various screening tests: breast cancer screening through (i) self-examination and (ii) mammography, prostate cancer screening through (iii) PSA tests, and Colo-rectal cancer through (iv) colonoscopies.

These tests are a little controversial because of poor predictive probability of the tests.

Diagnostic Testing (cont.)

Bayes' theorem can be rewritten for diagnostic tests:

$$P(D+|T+) = \frac{P(T+|D+)P(D+)}{P(T+|D+)P(D+) + P(T+|D-)P(D-)}$$

These probability quantities can then be defined as:

- *Sensitivity*: $P(T+|D+)$
- *Specificity*: $P(T-|D-)$
- *Prevalence*: $P(D+)$
- *Positive Predictive Value*: $P(D+|T+)$
- *Negative Predictive Value*: $P(D-|T-)$

How do positive and negative predictive values relate? Be careful...

Diagnostic Testing

We mentioned that these tests are a little controversial because of their poor predictive probability. When will these tests have poor positive predictive probability?

When the disease is not very prevalent, then the number of 'false positives' will overwhelm the number of true positive. For example, PSA screening for prostate cancer has sensitivity of about 90% and specificity of about 97% for some age groups (men in their fifties), but prevalence is about 0.1%.

What is positive predictive probability for this diagnostic test?

Why do we care?

As data scientists, why do we care about diagnostic testing from the medical world? (hint: it's not just because Kevin is a trained biostatistician!)

Because classification can be thought of as a diagnostic test. Let $Y_i = k$ be the event that observation *i* truly belongs to category *k*, and let $\hat{Y}_i = k$ the event that we correctly predict it to be in class *k*. Then Bayes' rule states that our *Positive Predictive Value* for classification is:

$$P(Y_i = k|\hat{Y}_i = k) = \frac{P(\hat{Y}_i = k|Y_i = k)P(Y_i = k)}{P(\hat{Y}_i = k|Y_i = k)P(Y_i = k) + P(\hat{Y}_i = k|Y_i \neq k)P(Y_i \neq k)}$$

Thus the probability of a predicted outcome truly being in a specific group depends on what? The proportion of observations in that class!

Error in Classification

There are 2 major types of error in classification problems based on a binary outcome. They are:

False positives: incorrectly predicting $\hat{Y} = 1$ when it truly is in $Y = 0$.

False negative: incorrectly predicting $\hat{Y} = 0$ when it truly is in $Y = 1$.

The results of a classification algorithm are often summarized in two ways:

1. a confusion matrix, sometimes called a contingency table
2. a receiver operating characteristics (ROC) curve.

Confusion table

When a classification algorithm (like logistic regression) is used, the results can be summarize in a (2 x 2) table as such:

	Predicted Healthy $(\hat{Y} = 0)$	Predicted AHD $(\hat{Y} = 1)$
Truly Healthy $(Y = 0)$	132	32
Truly AHD $(Y = 1)$	56	83

The table above was a classification based on a logistic regression model to predict heart disease based on 3 predictors: X_1 = Sex, X_2 = Resting HR, and X_3 = the interaction between the two.

What are the false positive and false negative rates for this classifier?

Bayes' Classifier Choice

A classifier's error rates can be tuned to modify this table. How?

The choice of the Bayes' classifier level will modify the characteristics of this table.

If we thought is was more important to predict republicans correctly (lower false positive rate), what could we do for our Bayes' classifier level?

We could classify instead based on:

$$\hat{P}(Y = 1) < \pi$$

and we could choose π to be some level other than 0.5. Let's see what the table looks like if π were or .

Other Confusion tables

predicted mean (from model)

Based on π = 0.4587:

	Predicted Healthy ($\hat{Y} = 0$)	Predicted AHD ($\hat{Y} = 1$)
Truly Healthy ($Y = 0$)	122	42
Truly AHD ($Y = 1$)	47	92

What has improved? What has worsened?

Based on π = 0.75:

	Predicted Healthy ($\hat{Y} = 0$)	Predicted AHD ($\hat{Y} = 1$)
Truly Healthy ($Y = 0$)	156	8
Truly AHD ($Y = 1$)	89	50

303

Which should we choose? Why?

best overall classification rate will be pi = 0.5 (bayes); so changing it will worse overall error rate
but might also gives me what i want (better classification in some areas)

correct:
156+50=206

ROC Curves

ROC Curves

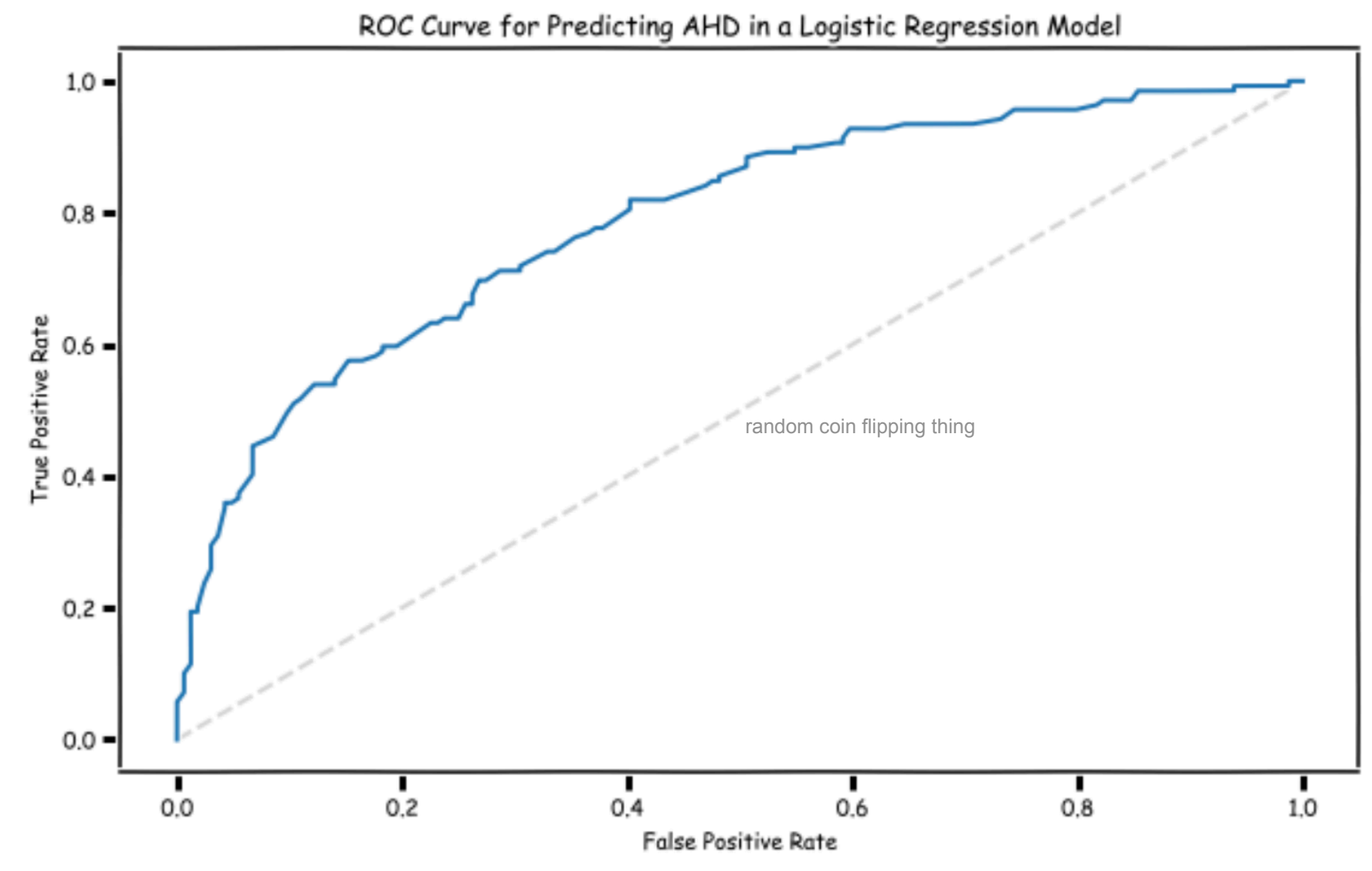
The Radio Operator Characteristics (ROC) curve illustrates the trade-off for all possible thresholds chosen for the two types of error (or correct classification).

The vertical axis displays the true positive rate and the horizontal axis depicts the false positive rate.

What is the shape of an ideal ROC curve?

See next slide for an example.

ROC Curve Example



ROC Curve for measuring classifier preformance

The overall performance of a classifier, calculated over all possible thresholds, is given by the area under the ROC curve (’AUC’).

An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.

What is the worst case scenario for AUC? What is the best case? What is AUC if we independently just flip a coin to perform classification?

This AUC then can be use to compare various approaches to classification: Logistic regression, LDA (to come), *k*-NN, etc...