13. Linear regression and logistic regression LING 471

Reminders

- Assignment 3 due today at 11:59 PM
- Assignment 4 is out
  - General goal is to use a naive bayes classifier on our IMDB data
  - We will go over naive bayes classifiers on Thursday

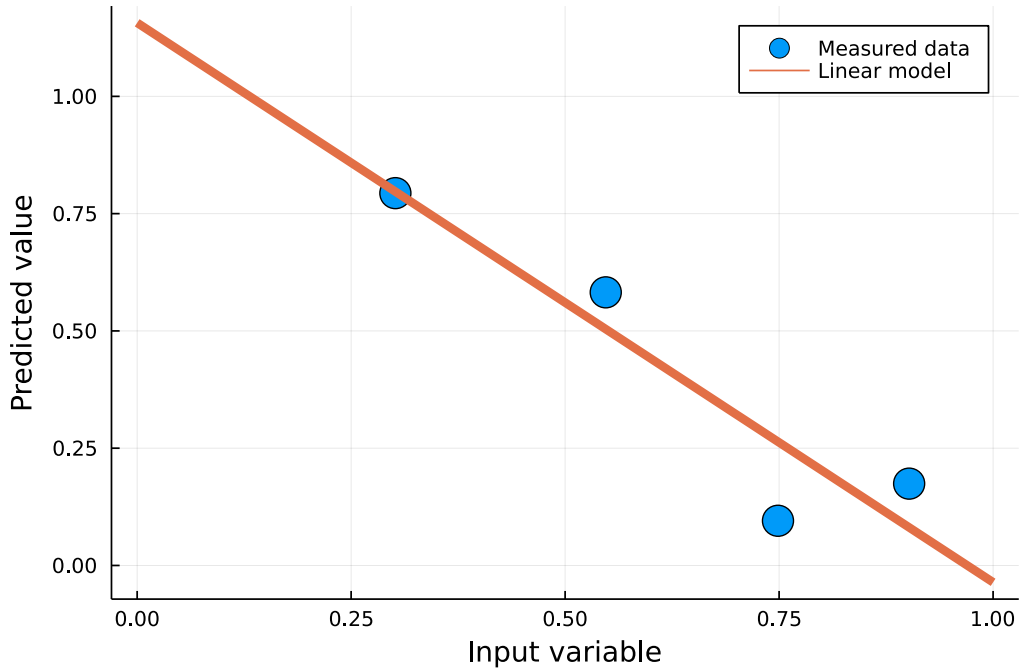
Learning outcomes

- Perform basis expansion to nonlinear functions with "linear" regression
- Perform logistic regression using Python libraries like statsmodels
- Describe the goal of a naive bayes classifier (as used on textual data)

Linear regression

Motivation

- We often want to make numerical predictions about the world
  - For example, does it take longer for a human to process longer words?
- To do so, we need some kind of model
- **Linear** models are generally the simplest useful model you can use



Linear regression overview

- Given some set of points, we want to find the line that best approximates our data
- This process is called linear regression
  - Linear: drawing a line, using linear algebra
  - Regression: well… Basically just a label for doing statistical modeling for a continuous outcome (as opposed to a categorical one)
- Goal: find slope and intercept of this best-fitting line
  - i.e., find m and b in $y = mx + b$

Setting the stage: Data and linear algebra

- We said we could store our data as a table, and we said a matrix is a table
  - Let's then say we have our input data in a matrix/table called X
- Let's then assume that our output/outcome data is stored in a vector called y
- We really want is to find the vector of coefficients b such that

$$Xb = y$$

  - Is this possible?

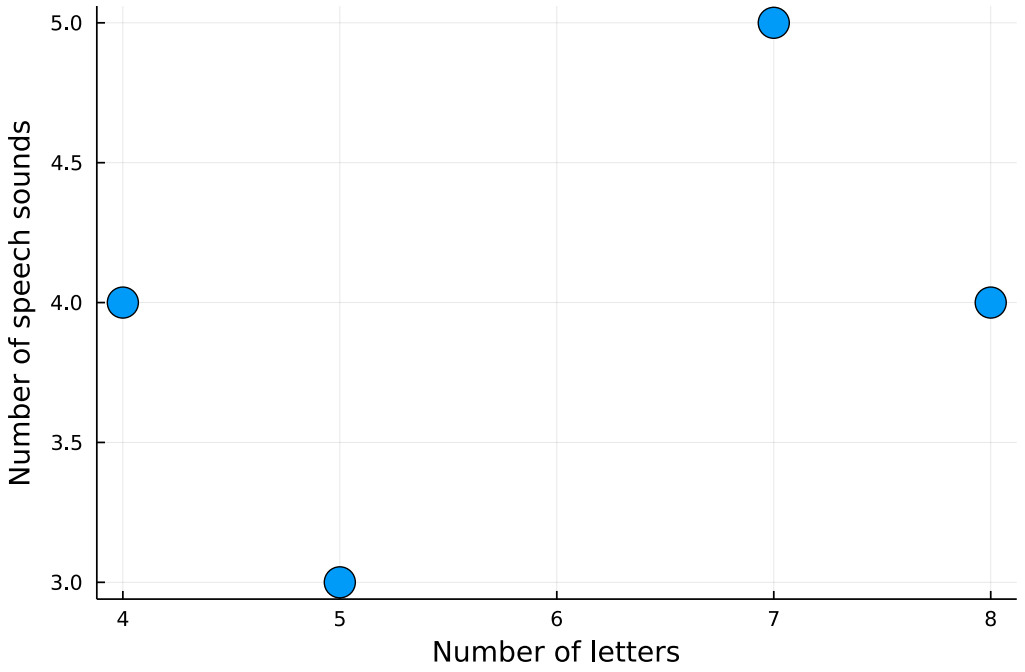
Let's collect some dummy data

Word	Letters	Transcription	Speech sounds
cats	4	kæts	4
thoughts	8	θɑts	4
clapped	7	klæpt	5
sheet	5	ʃit	3

- Let's start with some dummy data
- Let's predict how many speech sounds there are in an English word based on how many letters there are in it

Let's plot our data

- Our original goal was to draw a single line that hit all these points
- It doesn't look like we can do that
  - So, we **can't** do $Xb = y$
- But, we **can** do $Xb \approx y$
  - Or, $Xb = y - e$, where e is whatever the difference is between our true values and the linear predictions ($y - Xb$)



What is "approximately"?

- We wrote $Xb \approx y$, where $\approx$ means "approximately equal"
- What is "approximately" equal?
  - You can define it to be anything…
- Let's have some rigor though and say we want "approximately equal" to mean "as close as possible"
  - Just like how we want our classification accuracy, precision, recall, etc. to be **as good as possible** in our assignments
- So, we want to minimize that e vector

Can we just use the inverse?

- Since we have $Xb = y - e$, can we just multiply by a matrix inverse and be done?
  - Can we do $X^{-1}Xb = X^{-1}(y - e)$ to get $Ib = X^{-1}(y - e)$?
  - It would be convenient, but it is rare that there is actually an inverse for X, so we can't just use the inverse
  - That e vector is also still there…
- However… $X^T X$ **usually** has an inverse, so we can invert that to solve for $\hat{b}$

Solving our approximation

- Let's start from $X^T Xb = X^T y - X^T e$
  - Conveniently, in this instance, $X^T e = 0$ (just trust me…)
- Now, we have $X^T Xb = X^T y$
  - Let's multiply by $(X^T X)^{-1}$ on the left of both sides of the equation
- We have $(X^T X)^{-1}(X^T X)b = (X^T X)^{-1} X^T y$
  - We know that $X^T X$ is a matrix, and a matrix times its inverse is I
- Since I is the matrix equivalent of 1, we are left with

$$b = (X^T X)^{-1} X^T y$$

  - And we are done!

Wait, what?! How does this solve our approximation?

- Hard to explain in satisfying detail…
  - Have to cover projections, orthogonality, column spaces, and null spaces; or multivariate calculus, etc.
- Basically, $(X^T X)^{-1} X^T$ is known as the (left) pseudoinverse of X
  - Often denoted $X^\dagger$ ("X-dagger") or X^+ (presumably, "X-plus")
- The pseudoinverse is like the inverse, except that it minimizes how close $X^+ X$ is to I
  - For our purposes, this is enough, and it minimizes e

Putting our data into matrices

- For our data matrix, we have $X = \begin{bmatrix} 1 & 4 \\ 1 & 8 \\ 1 & 7 \\ 1 & 5 \end{bmatrix}$
- What is that column of 1s?
  - It's a hack to get our intercept (b in $y = mx + b$)
- For our output we have $y = \begin{bmatrix} 4 \\ 4 \\ 5 \\ 3 \end{bmatrix}$

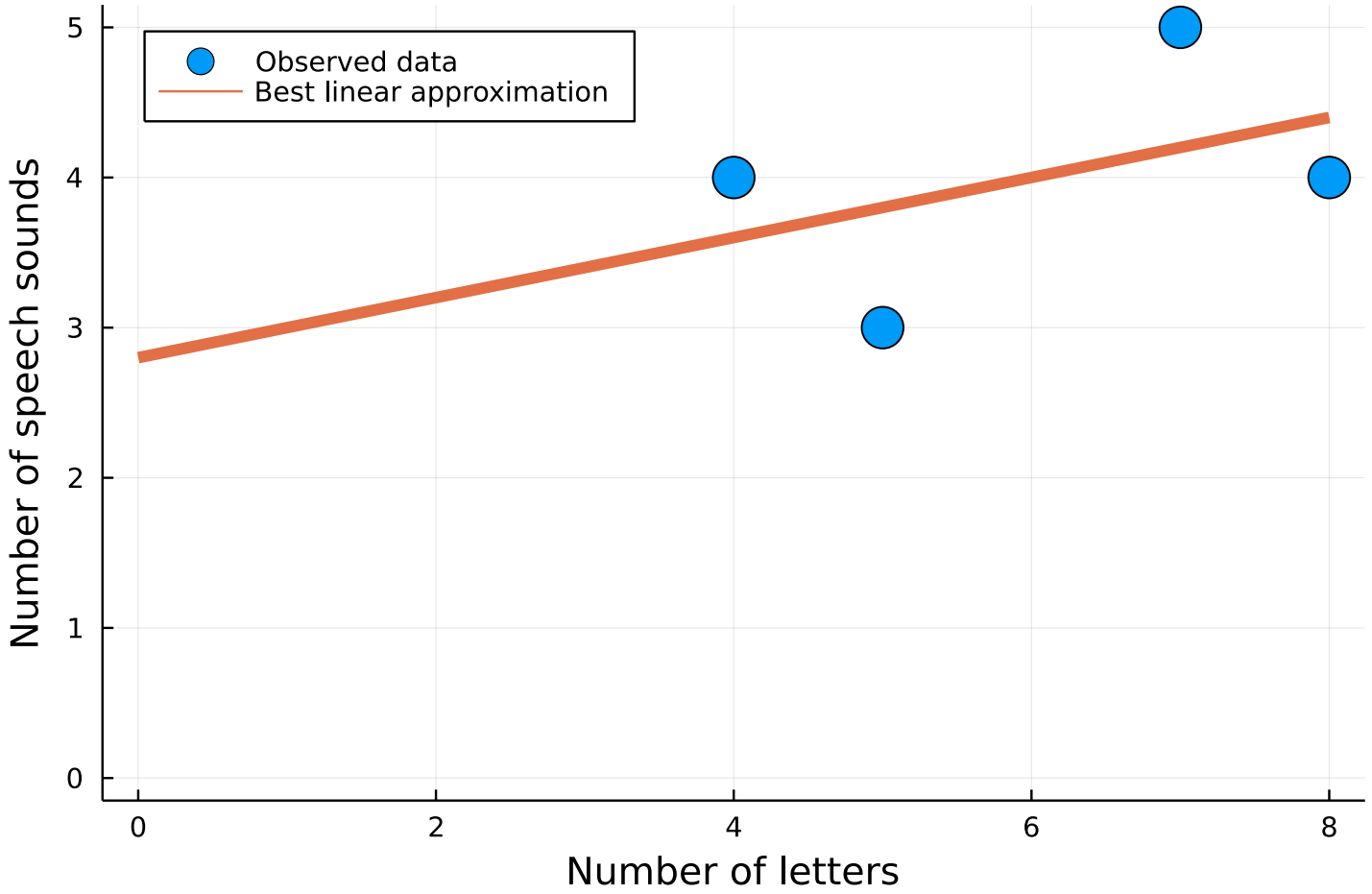
Solving for b

- We said $b = X^{+}y$, and we have X^{+} and y
- Let's put it into NumPy
- Use `numpy.linalg.pinv` to get the pseudoinverse
- For our approximation, we get

$$\psi \approx 0.2\chi + 2.8$$

  - Remember that each number in y and X is represented with a Greek letter

```
X = np.array([[1, 4], [1, 8], [1, 7], [1, 5]])

y = np.array([[4], [4], [5], [3]])

Xplus = np.linalg.pinv(X)

b = Xplus @ y

# b = array([[2.8],
                 [0.2 ]])

# or

np.linalg.lstsq(X, y)
```

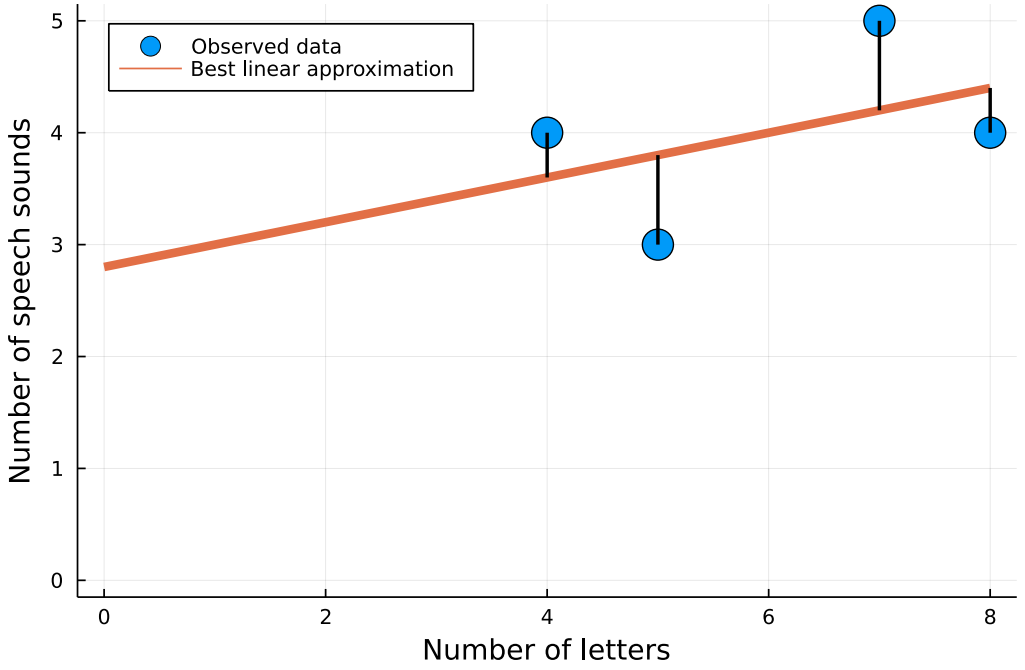


Plotting the results

- Let's plot our line
- This line seems like it satisfies our criteria of minimizing how far it is from each point

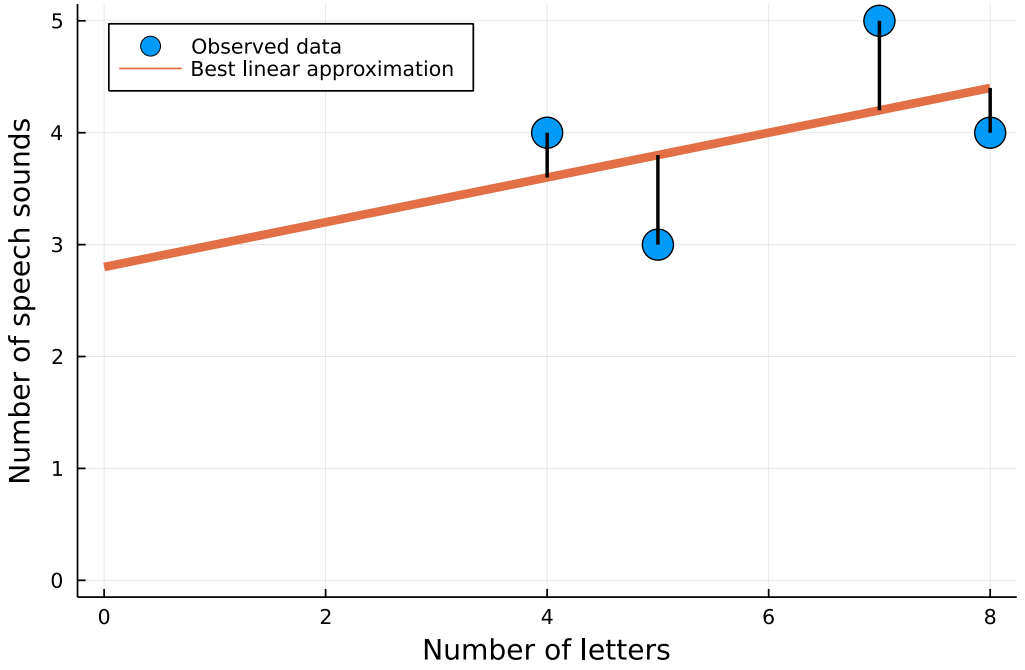
Linear regression and least squares

- When we found b, we minimized the **sum of squared errors** (SSE)
  - $SSE = \sum_i (\psi_i - \hat{\psi}_i)^2$
- That is, we have solved the **linear least squares** problem
  - We computed $\arg\min_b [(y - Xb)^T \cdot (y - Xb)]$
- SSE and similar values are **evaluation metrics** for continuous variables



Least squares and MLE

- Solving the linear least squares problem also happens to also solve the MLE problem
- That is, the slope and intercept could have been **any** numbers
  - What are their likeliest values, given our data?
  - The same numbers as our pseudoinverse solution gave



5 minute brain break

We will do some programming activities after a few more slides…

Using software to do linear regression

Introducing scikit-learn

- Scikit-learn is yet another Python package
- It does a lot of data science/machine learning things
- We can have it do linear regression for us!
- When importing, it is abbreviated as `sklearn`

Using linear regression in scikit-learn

- General workflow
- Set up the model by instantiating it
- Run the `fit()` method the model has, using your data as input
- Inspect the output

```
# assume we have X and y from before

# X does not need the additional column of 1s
this time

from sklearn.linear_model import
LinearRegression

X = X.reshape(-1, 1) # make X a matrix

reg = LinearRegression()

mod = reg.fit(X, y)

print(mod.coef_) # get slope; 0.2

print(mod.intercept_) # get intercept; 2.8

mod.predict(X)
```

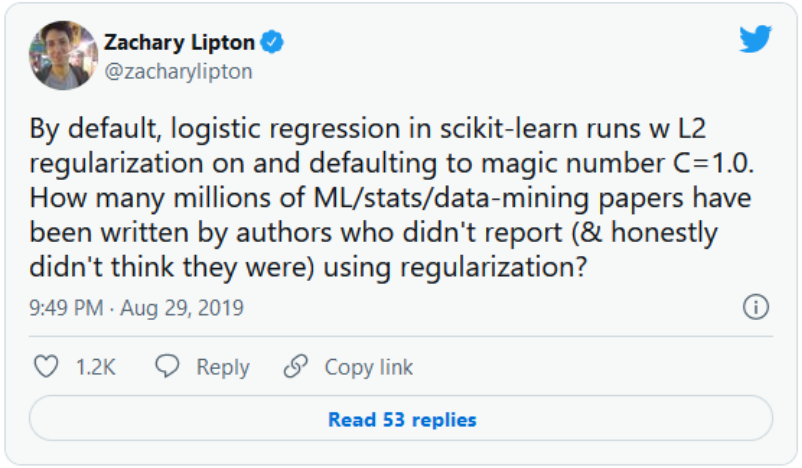
Some pointers about scikit-learn

- It was developed more so by software engineers than statisticians, and it is sort of grafted onto Python
- This has two pitfalls
  1. It is often clunky to use
  2. Some of the defaults and outputs are not ideal (see right)
- Still, it is very powerful and has **a lot** of different machine learning models you can play around with
- We might try another package next time, for experience

Scikit-learn's Defaults are Wrong

Posted on **August 30, 2019** by **W.D.**

This recent Tweet erupted a discussion about how logistic regression in Scikit-learn uses L2 penalization with a lambda of 1 as default options. If you don't care about data science, this sounds like the most incredibly banal thing ever. If you do care about data science, especially from the statistics side of things, well, have fun reading this thread:



https://web.archive.org/web/20220509234715/https://ryxcommar.com/2019/08/30/scikit-learns-defaults-are-wrong/

Some data from an experiment

- We will do programming activities in just a moment
- We are going to perform a linear regression on data that came from a real, published experiment (that I am a co-author on…)
- Participants are played a series of stimuli that are either real words or fake words
- For each stimulus the participant must respond if they heard a real word or fake word
  - We measure how long it takes them to respond in milliseconds (thousandths of a second)
- The response latency is strongly linked to psychological processing of speech (and of written data, if you use text instead of speech)

What are you doing to model?

- Earlier, I suggested that the number of letters in a word might relate to how long it takes to understand it
- So, let's model that!
- Download the "experiment_data.txt" file from the folder for today's activity in GitHub
- The first column is the number of characters in a word
- The second column is how many milliseconds it took a participant to respond to that word

Loading the data

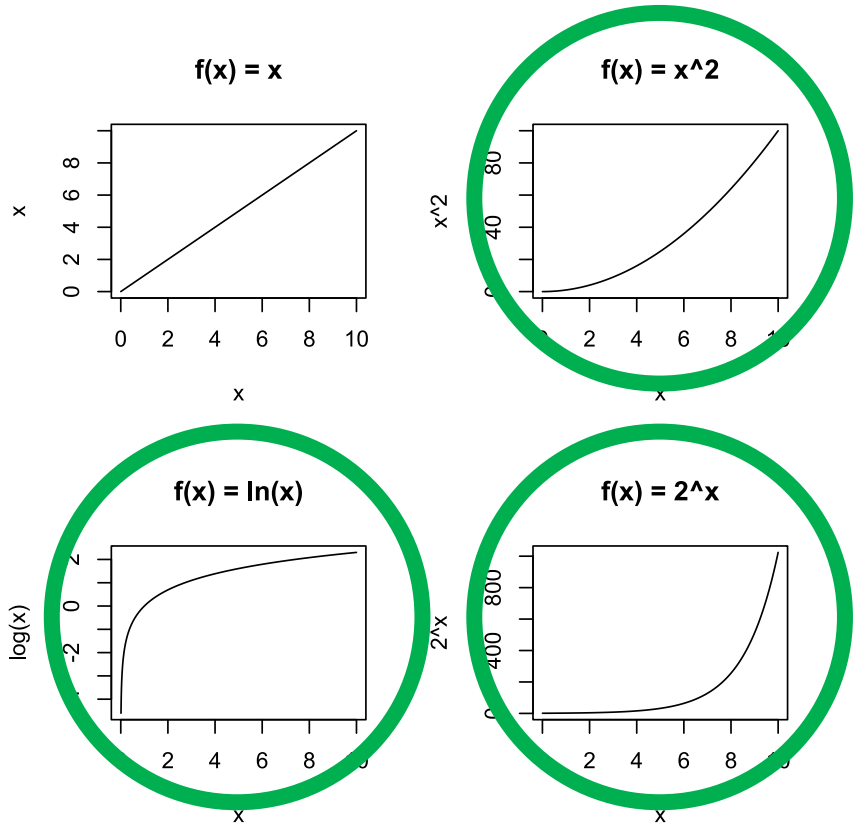
- We can use numpy.loadtxt to load the text file in as a matrix
- We index a NumPy array using [], like with lists
- Each dimension (rows, columns, etc.) is separated by a ","
- If you want to select an entire row, use ":"

```
d = np.loadtxt('experiment_data.txt')

X = d[:,0]

y = d[:,1]

# add the 1s column in a hacky way

X = np.vstack((np.ones(X.shape[0]), X)).T
```

Extending linear regression

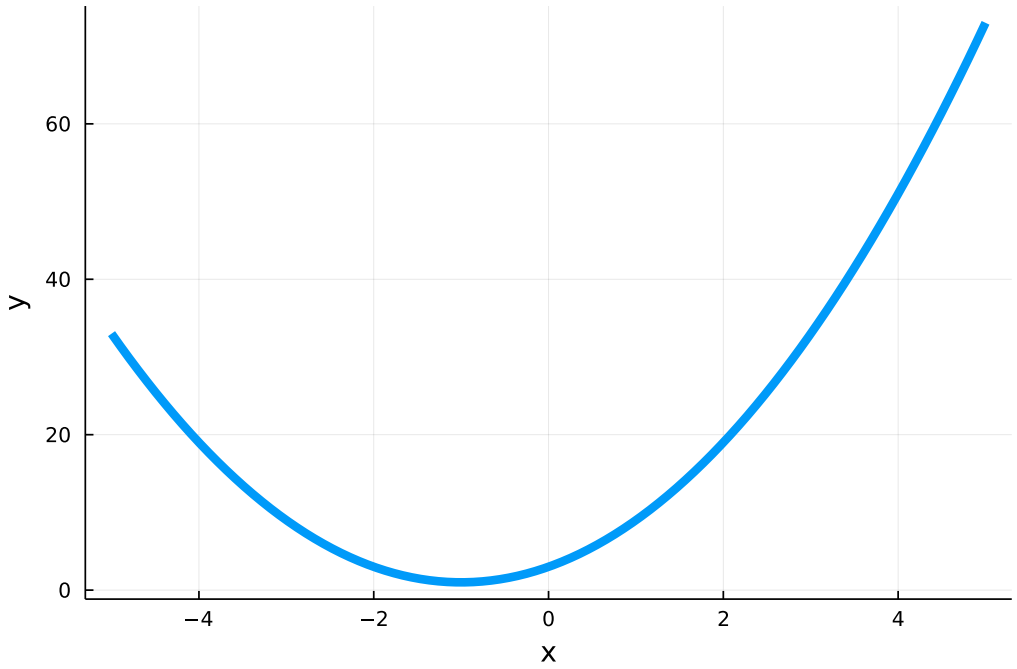
Nonlinear functions

- Many relationships and processes in life are nonlinear
  - I.e., they are not a straight line
  - Lines are often still useful…
- This would seem to be a problem for us and require a tool beyond linear regression
  - But is this actually the case?



Nonlinear function example

- Consider the following parabola: $y = 2x^2 + 4x + 3$
  - This will make a bowl shaped curve
- Is there a clever way for us to use matrix multiplication to convert this to some form of $y = Xb$?
  - (with the understanding that we are approximating and there is an error term)



Thinking about our function

- Let's think about the function we just saw
- It could be written as $y = \beta_1 a + \beta_2 c + \beta_3 d$
  - Where $a = x^2, c = x, d = 1, \beta_1 = 2, \beta_2 = 4, \beta_3 = 3$
  - Doesn't this look like a dot product?
- What would be the result of the following?
  - $[a \quad c \quad d] \cdot \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$

Evaluating our dot product

- If we use the definition of the dot product, we get
  - $[a \quad b \quad c] \cdot \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = a\beta_1 + c\beta_2 + d\beta_3$
- And if we plug in the values we assigned for our variables, we get
  - $[x^2 \quad x \quad 1] \cdot \begin{bmatrix} 2 \\ 4 \\ 3 \end{bmatrix} = x^2 \cdot 2 + x \cdot 4 + 3 \cdot 1 = 2x^2 + 4x + 3$
- So, it looks we **can** model nonlinear functions with linear algebra…

Basis expansion

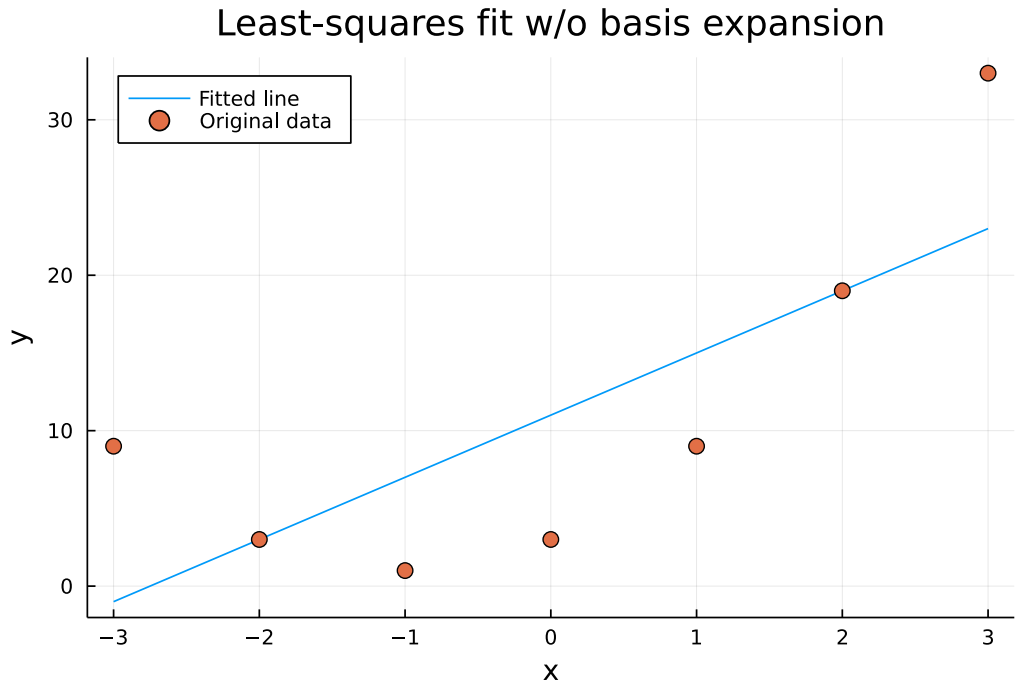
- If we take our input variable, make altered copies of it, and add it to our matrix, we have performed basis expansion
- Given some value for x, we can compute other values like $x^2, 10^x, \sin(x), \log(x)$, etc. and build a more complicated matrix
- What this means is we can model many more functions if we can make them take the form of $Xb = \beta_1 x_1 + \beta_2 \mathrm{x}_2 + \cdots$
  - Each β_i is one of the coefficients stored in the vector b
  - Each x_i is one of the columns (variables) stored in the matrix X

Trying this out

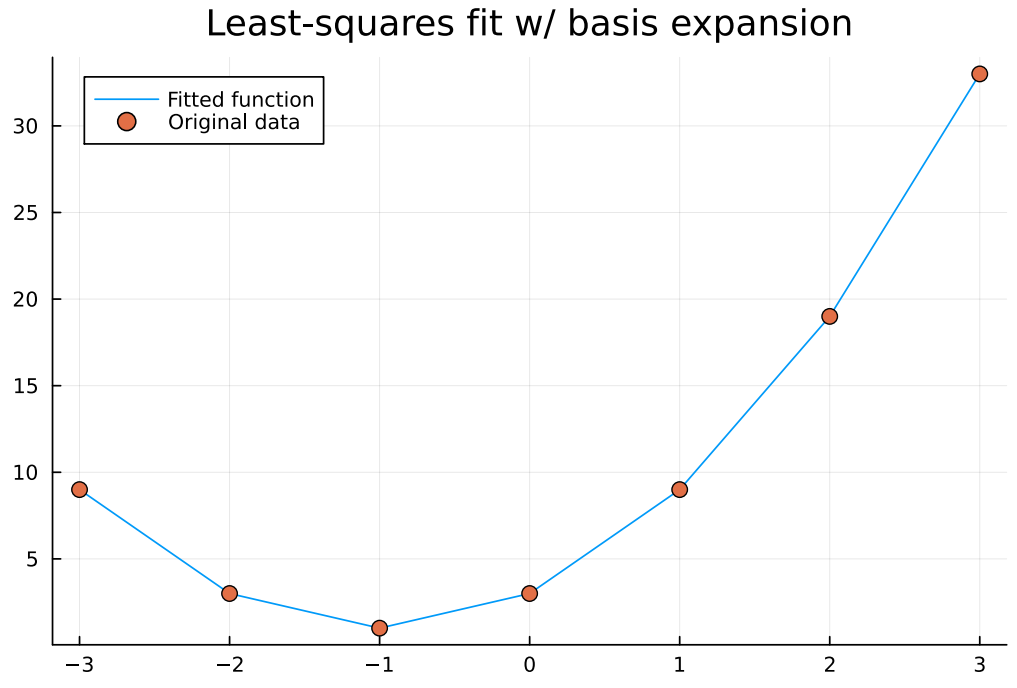
- Let's compare models; we need data to train on/fit to
  - No basis expansion: $X = \begin{bmatrix} 1 & -3 \\ 1 & -2 \\ \vdots & \vdots \\ 1 & 3 \end{bmatrix}, \mathrm{y} = \begin{bmatrix} 9 \\ 3 \\ \vdots \\ 33 \end{bmatrix}$
  - Basis expansion: $X = \begin{bmatrix} 1 & -3 & 9 \\ 1 & -2 & 4 \\ \vdots & \vdots & \vdots \\ 1 & 3 & 9 \end{bmatrix}, \mathrm{y} = \begin{bmatrix} 9 \\ 3 \\ \vdots \\ 33 \end{bmatrix}$

Regression results

NO BASIS EXPANSION



WITH BASIS EXPANSION

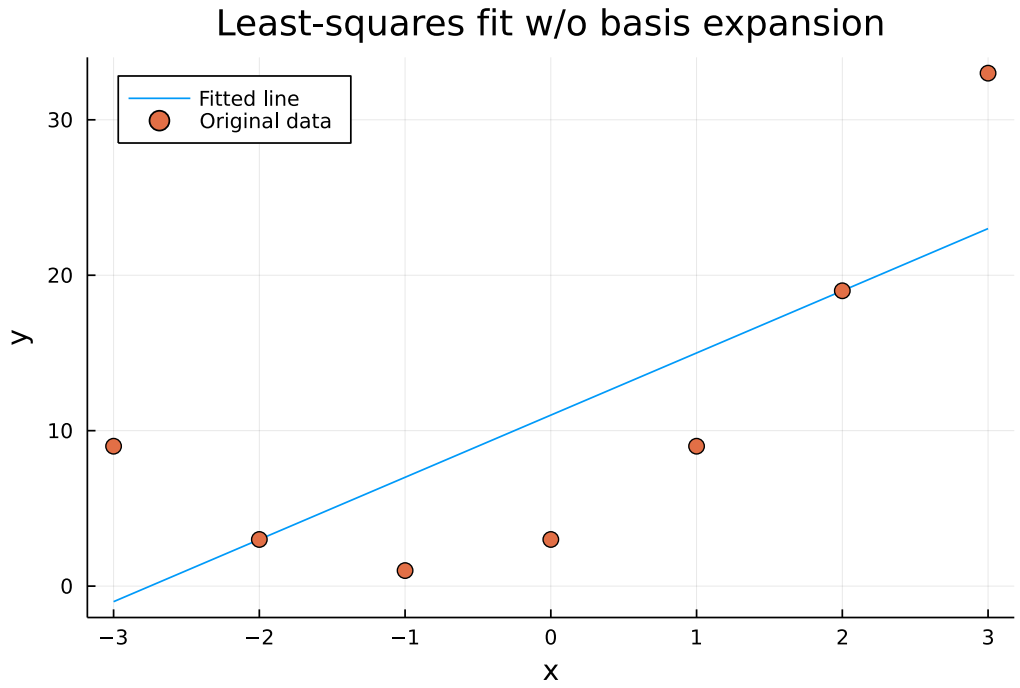


Basis expansions risks

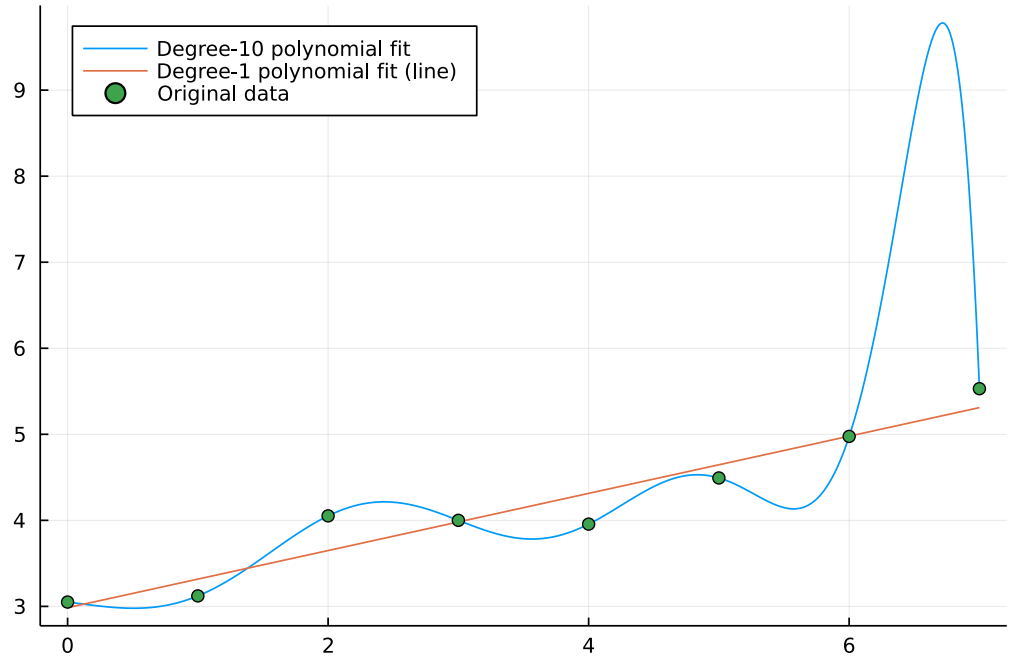
- You may not know the exact relationship and decide to add lots of new variables (maybe up to x^{10} or higher!)
  - This can cause overfitting, where your model assumes too much about a relationship and becomes too sensitive to small changes in the input
- You might also underestimate the complexity of the relationship
  - This can cause underfitting, where your model does not capture enough of the relationship (see our line trying to model the parabola)

Underfitting vs. overfitting

UNDERFITTING



OVERFITTING



Countering overfitting

- We often try to counter overfitting and underfitting at the same time
  - Counter underfitting by fitting more complex models
  - Counter overfitting by forcing model to "justify" complexity
- How to get model to "justify" complexity?
  - Penalize "wiggliness"
  - Penalize high coefficients (so, don't fit something like $100000x^{10}$ unless it's really needed)
  - General process called **regularization**

Programming activity

- Write the function $f(x) = 2\sin(x) + 3(2.0^x) + 3x$ in Python
  - Make sure to use np.sin and not math.sin
- Evaluate the function on the array generated by np.arange(-2, 3) and assign this to the variable y
- Create a matrix X with basis expansion
  - You will need to use np.hstack to join columns
  - You can call, for example, np.sin(np.arange(-2, 3)) to make a column
  - Each column will need to have ".reshape(-1, 1)" called after it
  - This matrix does not need a ones column (but it won't hurt to have it either)
- Use np.linalg.lstsq on your basis expansion matrix to get coefficients and save in the variable b
- See how well Xb matches y
- Repeat the regression with scikit-learn (and use its predict function instead of evaluating Xb)

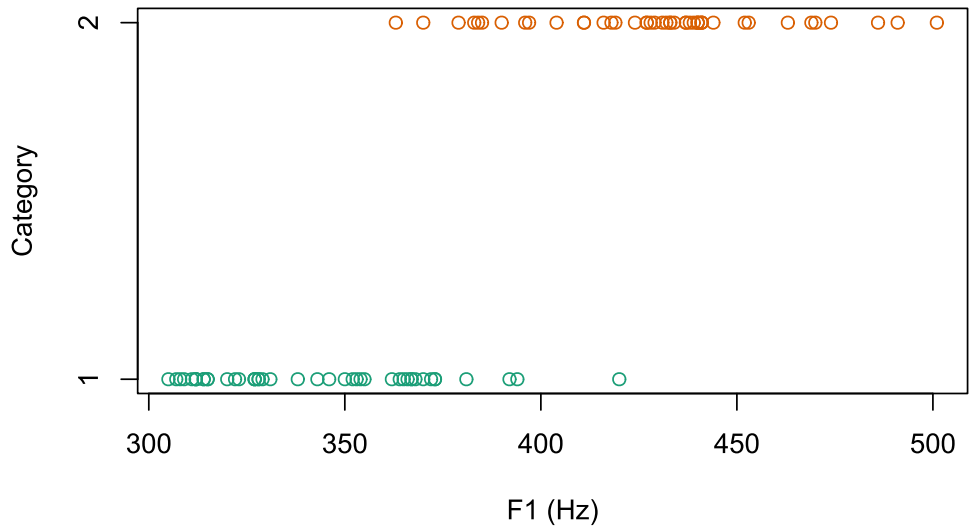
Classification and logistic regression

Classification

- We have been working on classification in our assignments
  - Does the review belong to the "positive"/"good" category?
  - Or does the review belong to the "negative"/"bad" category?
- We have also done some classification in class
  - Is this acoustic measurement more likely to have come from the word *heed*?
  - Or from the word *hid*?
- Can we do classification with linear regression?

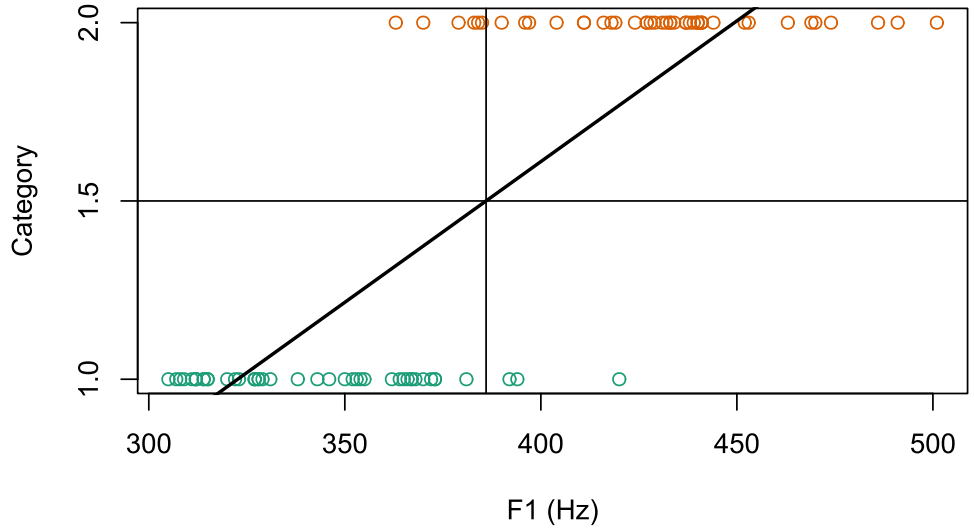
Linear regression as a classifier: Setup

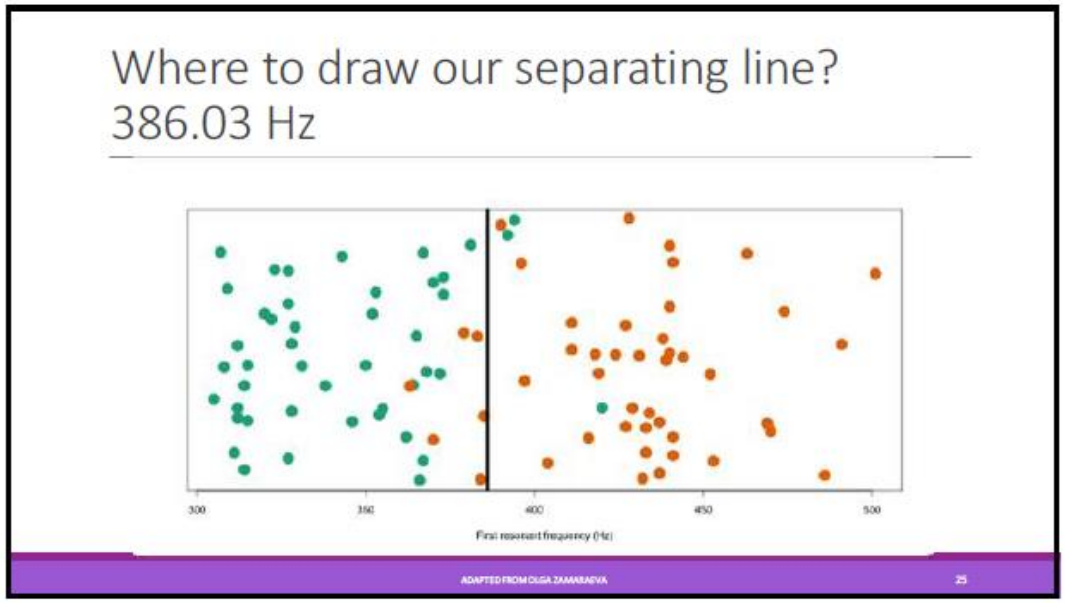
- You certainly *can* try to use linear regression as a classifier
- Let's think back to our vowel data
  - Let's say that *heed* tokens are category 1, *hid* tokens are category 2
- Regress the acoustic measurement against the category



Linear regression as a classifier: Results

- Have to say "1.5" is the 50% mark
  - Occurs at 386.03 Hz
- But 1 and 2 are just categories
  - What is 1.5? Especially if the categories are "cat" and "dog"…
- What if we want a probability?
  - Can't get it here
  - Line extends infinitely





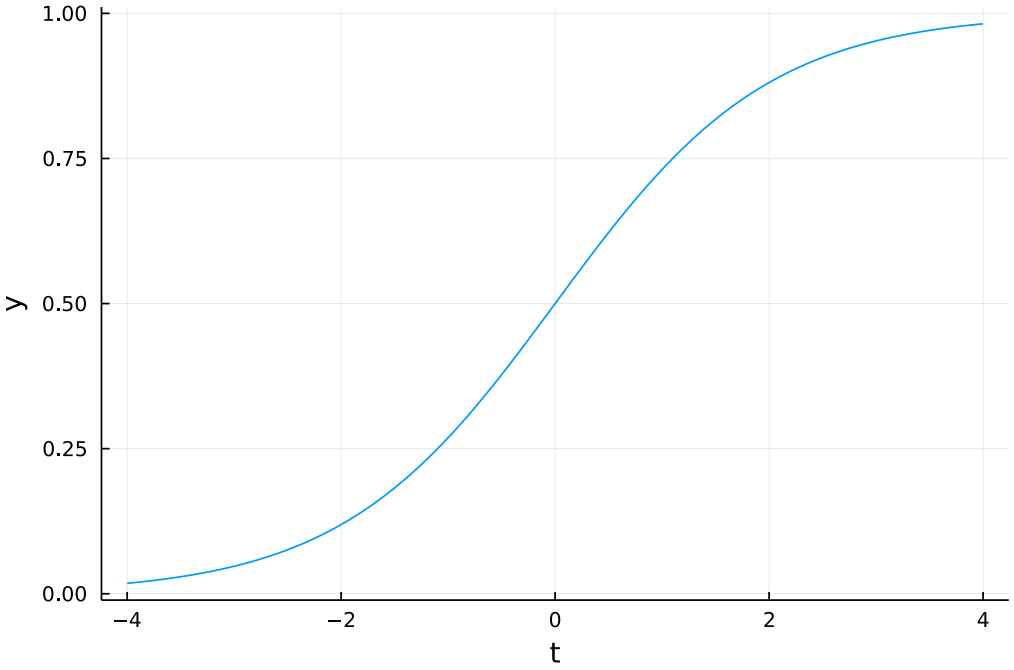
25

Is a better method possible?

- That 386.03 Hz value is familiar…
- It's the same as when we performed linear discriminant analysis!
  - Which gave us probability
- But… Can we instead use regression?
  - Then we don't have to calculate means and standard deviations!

Enter logistic regression

- There is a linear model that gives probability called **logistic regression**
- We start with the logistic **sigmoid** function: $f(t) = \frac{1}{1+e^{-t}}$
  - Sigmoid sometimes denoted with σ
- Notice how it scales between 0 and 1
  - Like a probability
  - Unlike a line from $-\infty$ to ∞



Manipulating the sigmoid function

- We need to somehow transform the sigmoid function to match our linear regression template
  - Luckily, the log (natural log) function helps us with this
- It can be shown (not here!) that $\ln \frac{f(t)}{1-f(t)} = t$
  - We can choose to model t as $Xb = \beta_1 x_1 + \beta_2 x_x$, which is linear
  - This gives us **logits**, which are related to probability
- This means our regression ends up as $p(X) = \sigma(Xb)$
  - Sadly, there is no easy solution to optimize this
  - Instead, we perform a numerical optimization

Fitting logistic regression

- We let software take care of this for us, but basically, we have the following loss function (called **log loss** or **binary cross entropy**):
  - Assuming our categories (y) are 0 or 1
  - $L(y, \hat{y}) = -(y \ln(p) + (1 - y) \log(1 - p))$
- This loss function is a measure of the **entropy** of our probability system
  - We want systems that are the least entropic possible, that is, the least chaotic possible or the **least uncertain** possible
- Simply, we want our model to predict values as close to 0 or 1 as possible

A note on classification

- When we are modeling just two categories, we usually assign one category to a probability of 0 and one category to a probability of 1
- Our model is saying, "there is an x% chance that this item belongs to the category associated with 1"
  - Our highest accuracy comes when we round that probability to either 0 or 1

The statsmodels package

- Provides a lot of machine/statistical learning models to be used for statistics
- statsmodels provides information useful for analyzing the relationship between variables
  - scikit-learn is focused on making models that perform well

Fitting a logistic regression

- We're going to fit a logistic regression model
- Possible to do in scikit-learn, but we're going to use statsmodels this time
- Usually import like `import statsmodels.api as sm`
- Need to use the logit model

```
import statsmodels.api as sm

import pandas as pd

import numpy as np

X = np.loadtxt('h95_ih_iy_adult_male.txt')

iy = np.vstack((X[:,0].T, np.zeros(45))).T

ih = np.vstack((X[:,1], np.ones(45))).T

vowels = np.vstack((iy, ih))

d = pd.DataFrame({'f1': vowels[:,0],
'vowel':vowels[:,1]})

d = sm.add_constant(d)

m = sm.Logit(d.vowel, d[['const', 'f1']]).fit()

m.summary()
```

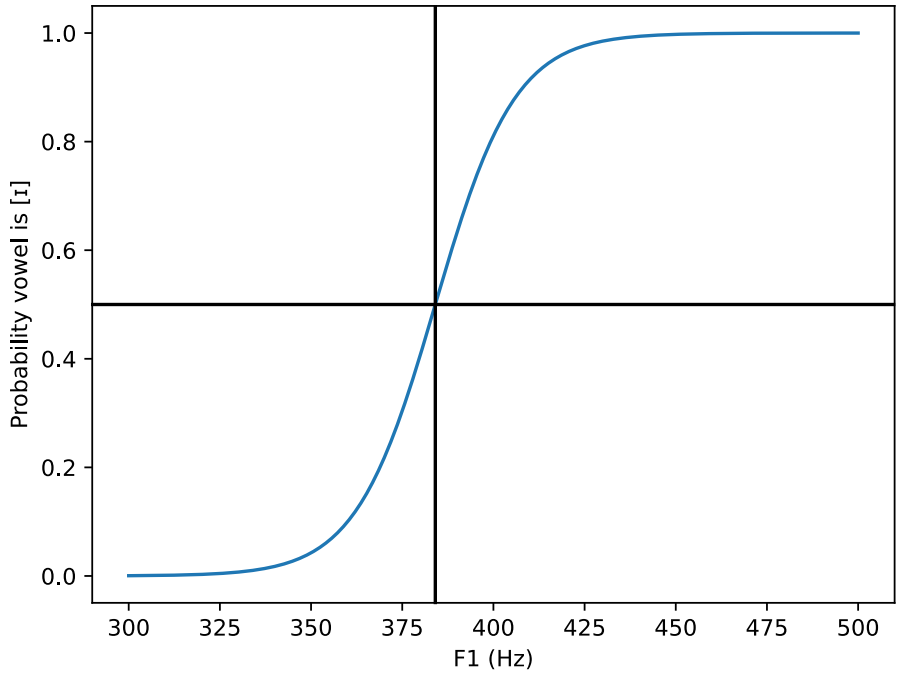
Regression results

- The "coef" column is each β in b
- The other columns are used for statistical analysis
  - But not necessarily for machine learning (maybe for data science)

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  vowel   No. Observations:                   90
Model:                          Logit   Df Residuals:                       88
Method:                           MLE   Df Model:                            1
Date:                Wed, 11 May 2022   Pseudo R-squ.:                  0.7219
Time:                        22:15:56   Log-Likelihood:                -17.351
converged:                       True   LL-Null:                       -62.383
Covariance Type:            nonrobust   LLR p-value:                 2.304e-21
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -35.0865      8.376     -4.189      0.000     -51.503     -18.670
f1             0.0914      0.022      4.164      0.000       0.048       0.134
==============================================================================
"""
```

Where is our 50% cutoff?

- We can determine the 50% probability point from the regression
- It happens to be 384.07 Hz
  - Almost the same as our linear regression
- Accuracy: 91.11%
  - 82 of 90 correct



Programming activity

- Install the statsmodels package
- Load the data from the 5/2 demo in
- Perform the logistic regression as outlined in the previous slides
- Inspect the results

Naive Bayes classifier

Overview

- A naive Bayes classifier has a similar goal as logistic regression
  - Learn how to relate some set of variables to probabilities
- The method is different, however, and more directly involves Bayes theorem
  - For text classification, it relies on associations between words and text categories
  - Logistic regression could model this, though you would need a lot of variables to do this
- We will use our IMDB task as our working example
  - But we will stop short of actually implementing it since that's what your homework is

Problem setup

- We want to model P(good | review)
- By definition, this is $\frac{P(\text{review} \mid \text{good}) \cdot P(\text{good})}{P(\text{review})}$
- What is P(review)?
  - E.g., P("This is a great film!")
  - P(This) * P(is) * P(a) * P(great) * P(film) *P(!)
- How do we get P(great)?

Getting P(word)

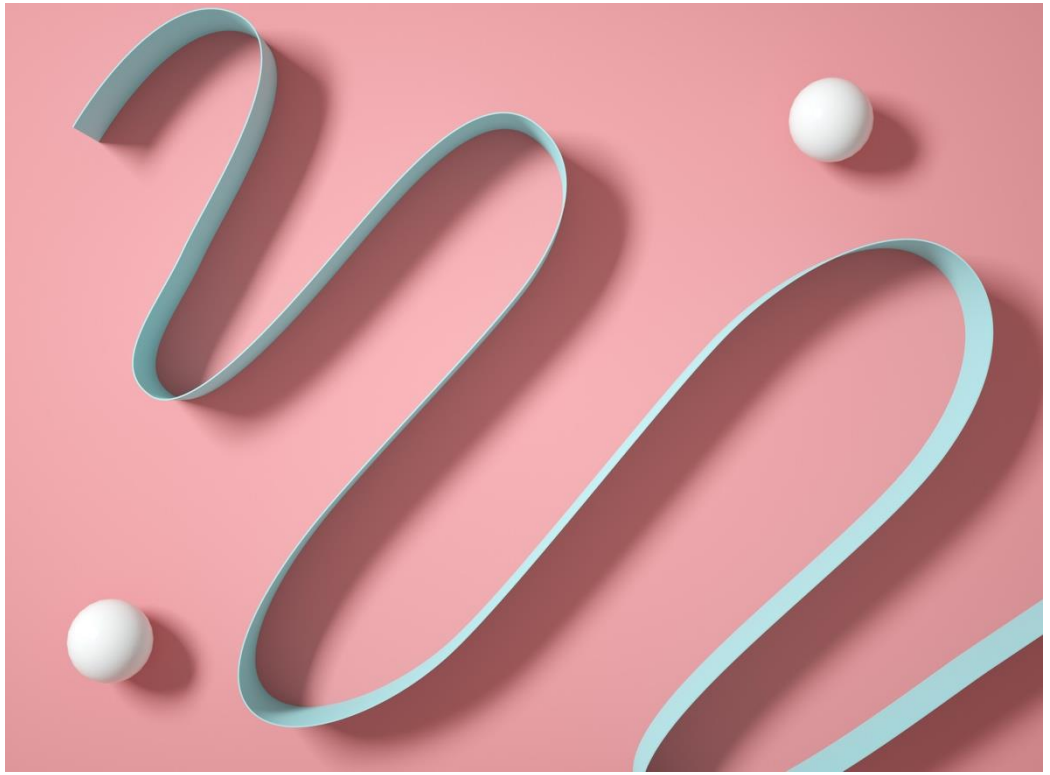
- We can estimate the probability of a word
- General concept: n / N
  - n = number of times word appears in some corpus (body) of text
  - N = total number of words in some corpus of text
- We've already done something similar to this in A2 and A3!
  - We can get more sophisticated though

More sophisticated P(word)

- In your assignment skeleton, there is code to make scikit-learn calculate **tf-idf**
  - Basically, this is a way of calculating probability for important words and discounting unimportant ones
  - E.g., do we really care what P(the) is for our task?
- However, what happens if you encounter a word that wasn't in your corpus?
  - E.g., your corpus was "This is a great film!" and you need P(terrible)

Smoothing

- If we don't account for unseen words, we can end up multiplying by 0, which causes the entire probability to be 0
  - Or $-\infty$ in log space, which quickly ruins everything
- We want to "smooth" our probability estimates
  - Many techniques to do this
  - Code already provided to do this in assignment



General naive bayes learning strategy

- Given some training set
  - Determine the probability of words in the training set relating to each category
    - Like our "good" = "positive" and "bad" = "negative", but looking instead at all words (or all meaningful words)
- When predicting
  - Compare a document/text's set of words to the learned probabilities before to calculate a final probability
    - Performs lots of probability multiplications!

Summary

- Linear regression, especially with basis expansions, allows for a wide variety of models to be fitted
- Logistic regression allows for classification through a generalization of linear models
- Even though linear and logistic regression are no longer state of the art models in data science, they are widely used because of their (relative) simplicity and ease of interpretability
  - In experimental linguistics, they are so common that you might be asked in a review why you aren't using them
  - You will learn more about these if you take a stats class for linguistics
- Naive Bayes classifiers perform a similar task as logistic regression,