20211025 Linear Regression continued Also continued 20211027

Wednesday, October 20, 2021 3:45 PM

-

Linear Regression

Boston University CS 506 - Lance Galletti

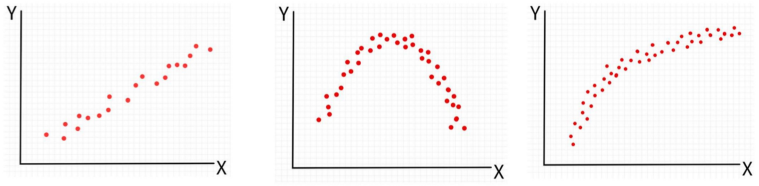
Challenge for those who have LR experience

- Find the data.csv file in the regression folder of our course repo
- Challenge:
  - Every day my alarm goes off at seemingly random times...
  - I've recorded the times for the past year of so (1 - 355 days)
  - Today is day 356
  - Can you predict when my alarm will ring?

- If you have done linear regression before, than try the challenge
  - This dataset is on the repo
  - It will ring during class time FOR SURE
  - Try to run your linear regression model, you will have to do some feature tuning and feature extraction
  - Raise hand if you have a prediction

Motivation

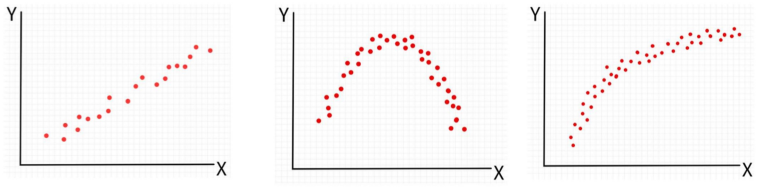
Given **n** samples / data points ($\mathbf{y_i}$, $\mathbf{x_i}$)



- We are moving away from classification
- We are now wanting to predict a continuum of possible values
- X and Y are continuous
- How does Y change as a function of X
- Any new point X, we apply learned function h and get an estimate for Y

Motivation

Understand/explain how **y** varies as a function of **x** (i.e. find a function **y = h(x)** that best fits our data)



Motivation

Suppose we are given a curve **y = h(x)**, how can we evaluate whether it is a good fit to our data?

Compare $\mathbf{h(x_i)}$ to $\mathbf{y_i}$ for all **i**.

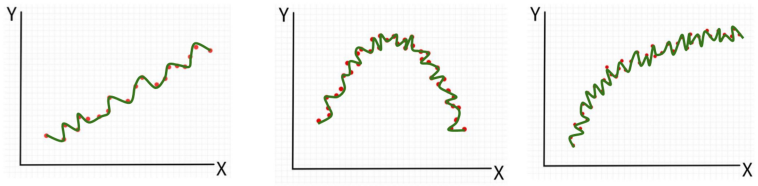
Goal: For a given distance function **d**, find **h** where **L** is smallest.

$$L(h) = \sum_i d(h(x_i), y_i)$$

- If I gave you a function h, how would you know it's a good prediction or not?
- For a particular distance function, can take the sum of all the comparisons

Motivation

Should **h** be the curve that goes through the most samples? I.e. do we want $\mathbf{h(x_i) = y_i}$ for the maximum number of **i**?



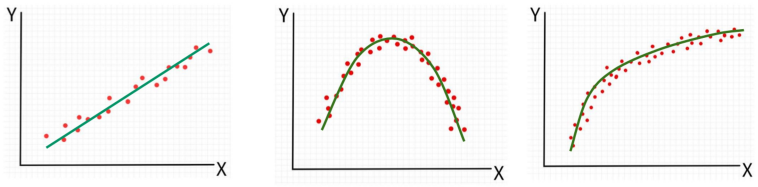
h may be too complex
overfitting - may not perform well on unseen data

- Do we want the curve to go through every point? No
  - You may overfit
  - Also over-complicated
  - We want to be able to discover h ourselves without an oracle giving us the equation
- Our cost function still stands
- It's not about finding the right distance function
- It's about constraining the functions that we want to look at
- Infinity of infinity of functions
  - Iterate through that many functions set us up for failure
- Create some constraints for what would be good candidates

for us

Motivation

The following curves seem the most intuitive "best fit" to our samples. How can we define this best fit mathematically? Is it just about finding the right distance function?



Motivation

Another way to define this problem is in terms of probability.

Define **P(Y | h)** as the probability of observing **Y** given that it was sampled from **h**.

Goal: Find **h** that maximizes the probability of having observed our data.

- We just want to find the distribution h that maximizes the probability of having observed Y

Motivation

To sum up we can either:

1. Minimize

$$L(h) = \sum_i d(h(x_i), y_i)$$

1. Maximize

Motivation

To sum up we can either:

1. Minimize

$$L(h) = \sum_i d(h(x_i), y_i)$$

1. Maximize

L(h) = P(Y | h)

- Need to parameterize h in both cases
- We need a smaller window into the space of both functions

Getting Started

Do we have enough to get started?

Seems like there are too many possible **h** and our problem statements are still too vague to effectively find solutions.

What can we do to constrain the problem?

Let's make some assumptions!

- We make some assumptions

Assumptions

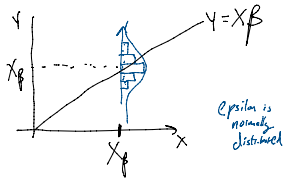
Let's start by assuming our data was generated by a **linear function** plus some **noise**:

$$\vec{y} = h_\beta(\mathrm{X}) + \vec{\epsilon}$$

Where **h** is linear in a parameter $\boldsymbol{\beta}$.
Which functions below are linear in $\boldsymbol{\beta}$?

$h(x) = \beta_1 x$	✔
$h(x) = \beta_0 + \beta_1 x$	✔
$h(x) = \beta_0 + \beta_1 x + \beta_2 x^2$	✔
$h(x) = \beta_1 \log(x) + \beta_2 x^2$	✔
$h(x) = \beta_0 + \beta_1 x + \beta_1^2 x$	✗

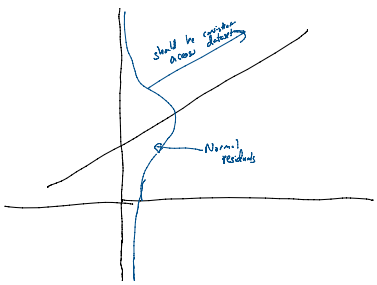
- In linear regression, we assume the data is LINEAR
- What does it mean to be linear in parameter beta
- We don't care about x, we just want the beta terms to be linear
- Green checks are linear in Beta
  - Does not need to be linear in x
- Red x are not linear in Beta
- Finding H is the same as finding all the Beta coefficients
- We expect that y is a function of x plus noise
  - With the same value of X, the value of Y may vary



Assumptions

1. The relation between **x** (independent variable) and **y** (dependent variable) is linear in a parameter $\boldsymbol{\beta}$.
2. ϵ_i are independent, identically distributed random variables following a $\mathbf{N(0, \sigma^2)}$ distribution. (Note: $\boldsymbol{\sigma}$ is constant)

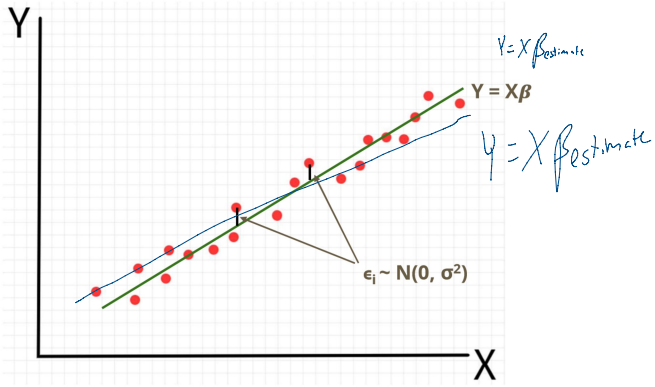
-



Assumptions

Y

Assumptions



- Y = XB generated the data
- There will always be noise
- We might never observe a point on the line
- We would like to get an estimate for Y = XB
- There should be an updated version of this slide
- We want on average to fall on the line

Goal

Given these assumptions, let's try to solve the max and min problems we defined earlier!

Q: What does solving these mean?

A: Finding $\boldsymbol{\beta}$ is equivalent to finding **h**

Least Squares

$$\begin{aligned}\beta_{LS} &= \arg\min_{\beta} \sum_i d(h_\beta(x_i), y_i) \\ &= \arg\min_{\beta} \|\vec{y} - h_\beta(\mathrm{X})\|_2^2 \\ &= \arg\min_{\beta} \|\vec{y} - \beta\mathrm{X}\|_2^2\end{aligned}$$

- Let's choose Euclidean distance and square it
- This is matrix notation

Least Squares

$$\begin{aligned}\frac{\partial}{\partial\beta} &= 0 \\ \frac{\partial}{\partial\beta}(y - \beta\mathrm{X})^T(y - \beta\mathrm{X}) &= 0 \\ \frac{\partial}{\partial\beta}(y^Ty - y^TX\beta - \beta^TX^Ty - \beta^TX^TX\beta) &= 0 \\ \frac{\partial}{\partial\beta}(y^Ty - 2\beta^TX^Ty - \beta^TX^TX\beta) &= 0 \\ -2X^Ty - X^TX\beta &= 0 \\ X^TX\beta &= X^Ty \\ \boxed{\beta_{LS} = (X^TX)^{-1}X^Ty}\end{aligned}$$

- Matrix version
  - Needed for more that than two variables
  - We don't need to get too deep into this math in this class
- It's like magic!
  - Squared equation gets you an estimate of Beta
- Beta least square parameter that you can get directly from your data

Maximum Likelihood

Since $\epsilon \sim N(0, \sigma^2)$ and $Y = X\beta + \epsilon$ then $Y \sim N(X\beta, \sigma^2)$.

$$\begin{aligned}\beta_{MLE} &= \arg\max_{\beta} \frac{1}{\sqrt{(2\pi)^n \sigma^n}} \exp(-\frac{\|y - X\beta\|_2^2}{2\sigma^2}) \\ &= \arg\max_{\beta} \exp(-\|y - X\beta\|_2^2) \\ &= \arg\max_{\beta} -\|y - X\beta\|_2^2 \\ &= \arg\min_{\beta} \|y - X\beta\|_2^2 \\ &= \beta_{LS} = (X^T X)^{-1} X^T y\end{aligned}$$

- This is the other approach
- Maximum Likelihood Estimation
- We know how Y is distributed
  - We want to find h that maximizes this probability
- Evaluate the PDF of the normal distribution
- Maximizing the exponent
- Now we got the same equation
  - We had two approaches to this problem, converged on the same solution
- Get same estimate for Beta using this approach as well

An Unbiased Estimator

β_{LS} is an unbiased estimator of the true β. That is $E[\beta_{LS}]=\beta$.

$$\begin{aligned}E[\beta_{LS}] &= E[(X^T X)^{-1} X^T y] \\ &= (X^T X)^{-1} X^T E[y] \\ &= (X^T X)^{-1} X^T E[X\beta + \epsilon] \\ &= (X^T X)^{-1} X^T X\beta + E[\epsilon] \\ &= \beta\end{aligned}$$

- Beta least squares is an unbiased estimator
  - On average, you get the true Beta
- How do we show this is the case?
  - Derivation
- Guarantee provided you are following the assumptions
  - If your data is indeed generated from this function…

- Expectation of beta least squares
- Random variable
  - y is random, X is not random
- Expectation of epsilon is 0
- On average, we expect to get the true result
-

Demo

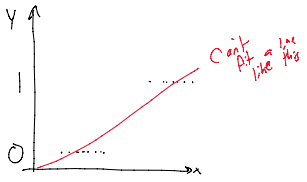
Logistic Regression

So far y_i was a continuous variable. What if y_i is categorical?

Assume we have **2 classes**.

Even if we can make these classes numerical (i.e. translate labels such as "yes"/"no" into 1 / 0), these numbers don't have a mathematical meaning in the context of linear models and what we learn will be as arbitrary as the numerical labels we assigned (i.e. using "yes" =2/"no"=7 instead of "yes"=1/"no"=0 might "fit" a better model...).

Maybe we can use the probability of belonging to a given class as a proxy for how confidently we can classify a given point? Maybe we can fit a linear model to the probability of being in a given class!



- What if Y is not linear?
  - Is there something kind of like linear regression?
- What is we predict a probability of belonging to a class
- Two classes: yes/no; 0/1; 2/7
  - Numerical values shouldn't mean anything
- A linear model will output values between negative and positive infinity, which doesn't make sense for probability
  - Doesn't make sense to use a linear model to something that is constrained to that interval

Logistic Regression

So the output of our regression model could be a probability. But how can we enforce that $X\boldsymbol{\beta}_{LS}$ from our model is always constrained to [0,1]? i.e. how can we learn a $\boldsymbol{\beta}_{LS}$ such that $0 \leq X\boldsymbol{\beta}_{LS} \leq 1$ even for unseen X?

Instead define the odds = p / 1 - p where p = P(Y = class 1 | X)

Now the range of $X\boldsymbol{\beta}_{LS}$ is [0, ∞)

But again how can we enforce that the $X\boldsymbol{\beta}_{LS}$ are constrained to [0, ∞)? We need (-∞, ∞) - but how?

Let's take the log! This is also convenient numerically because in the previous odds format, tiny variations in p have large effects on the odds!

- We need to define a function of Y that lets its model it with a linear model
  - Probability won't stay within the desired range
- Log of odds is something that can capture the whole real number line
- How does this function of Y fluctuate with the fluctuations of X

Logistic Regression

Our goal is to fit a linear model to the log-odds of being in one of our classes (in the 2-class case) i.e.

$$\log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) = \alpha + \beta X$$

- Log of the odds is a linear assumption of X is our assumption
- We will try to find an estimate for Beta based on this assumption
- This is what we will apply our linear model to
- This is natural log

Logistic Regression

Suppose we have such a model. How do we recover the P(Y=1 |X)?

$$\log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) = \alpha + \beta X$$

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

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

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

$$P(Y=1|X) = \frac{e^{\alpha+\beta X}}{1+e^{\alpha+\beta X}}$$

The function we apply to our probability to obtain the log odds is called the **logit** function. The function used to retrieve our probability from the log odds is called **logit^{-1}**

- Derivation of how to get the probability back from the log odds operation
  - logit inverse to get the probability
  - Then we use the probability as a proxy for what class you should be in
- Need to undo this log-odds function to get this probability
- This is ln (natural log)

Logistic Regression

How do we learn our model? I.e. the α and $\boldsymbol{\beta}$ parameters.

We know:

$$P(y_i = 1|x_i) = \begin{cases} logit^{-1}(\alpha + \beta x_i) \text{ if } y_i = 1 \\ 1 - logit^{-1}(\alpha + \beta x_i) \text{ if } y_i = 0 \end{cases}$$

$$= (logit^{-1}(\alpha + \beta x_i))^{y_i}(1 - logit^{-1}(\alpha + \beta x_i))^{1-y_i}$$

- We can transform this into a single equation by using yi and 1-yi as the exponents

Logistic Regression

So we can define

$$L(\alpha, \beta) = \prod_i (logit^{-1}(\alpha + \beta x_i))^{y_i} (1 - logit^{-1}(\alpha + \beta x_i))^{1-y_i}$$

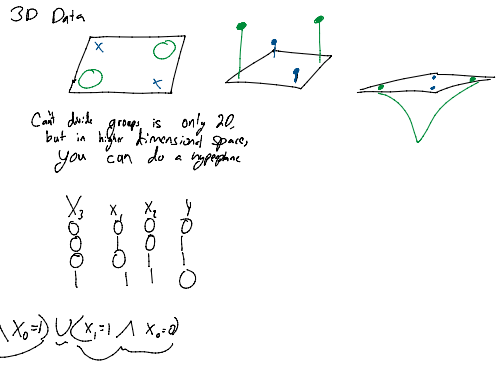
And try to maximize this quantity!

Unfortunately, there is no closed form solution here and we need to use numerical approximation methods to solve this optimization problem

- Cost function
- Goal was just to maximize the probability of seeing the data that we saw
- Gradient descent: we will discuss soon
- No close form solution like we saw with linear regression
- We will need to approximation this maximum with iterative techniques

Demo

- You could try to implement this for the midterm
- We need to engineer features to be able to use lines/curves to separate the points



use and, or, etc operators to be able to linearly separate these features
Find relevant features for the problem you are trying to solve

Evaluating Our Regression Model

Some Notation:

$\mathbf{y_i}$ is the "true" value from our data set (i.e. $\mathbf{x_i}\boldsymbol{\beta} + \boldsymbol{\epsilon_i}$)

$\hat{\mathbf{y}}_\mathbf{i}$ is the estimate of y_i from our model (i.e. $\mathbf{x_i}\boldsymbol{\beta}_{\mathbf{LS}}$)

$\bar{\mathbf{y}}$ is the sample mean all $\mathbf{y_i}$

$\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i}$ are the estimates of $\boldsymbol{\epsilon_i}$ and are referred to as residuals

---

- How do we know if our model means anything?
- We are making some assumptions on how our data was generated, these assumptions need to be validated

Evaluating Our Regression Model

$$TSS = \sum_i^n (y_i - \bar{y})^2 \qquad R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

$$RSS = \sum_i^n (y_i - \hat{y}_i)^2$$

$$ESS = \sum_i^n (\hat{y}_i - \bar{y})^2$$

R^2 measures the fraction of variance that is explained by $\hat{y}$

- Total sum of squares
- R-squared: how good of a linear fit our model is to the data
- How much variance are we actually explaining with our model
- What is a good value of R-square?
  - If ESS and TSS are close, that's better
  - R-squared close to 1 is good
  - Bad value would be 0

Exercise

Assume for simplicity that $\hat{y}_i = \beta_0 + \beta_1 x_i$
Since β_0 and β_1 are least squares estimates, we know they minimize

$$\sum_i (y_i - \hat{y}_i)^2$$

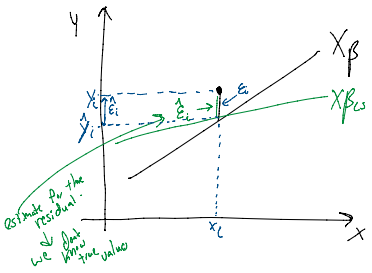
By taking derivatives of the above with respect to β_0 and β_1 we discover that

$$\sum_i (y_i - \hat{y}_i) = 0 \text{ and } \sum_i (y_i - \hat{y}_i) x_i = 0$$

Show that TSS = ESS + RSS

$$\begin{aligned}
\text{TSS} &= \sum_i (y_i - \bar{y})^2 \\
&= \sum_i (y_i - \hat{y}_i + \hat{y}_i - \bar{y})^2 \\
&= \sum_i (y_i - \hat{y}_i)^2 + \sum_i (\hat{y}_i - \bar{y})^2 + 2\sum_i (y_i - \hat{y}_i)(\hat{y}_i - \bar{y}) \\
&= \text{ESS} + \text{RSS} + 2\sum_i (y_i - \hat{y}_i)(\hat{y}_i - \bar{y}).
\end{aligned}$$

$$\begin{aligned}
\sum_i (y_i - \hat{y}_i)(\hat{y}_i - \bar{y}) &= \sum_i (y_i - \hat{y}_i)\hat{y}_i - \bar{y}\sum_i (y_i - \hat{y}_i) \\
&= \hat{\beta}_0 \sum_i (y_i - \hat{y}_i) + \hat{\beta}_1 \sum_i (y_i - \hat{y}_i) x_i - \bar{y}\sum_i (y_i - \hat{y}_i)
\end{aligned}$$



- Try to show this for next time
  - Someone is welcome to do this on the board next time
  - Fun problem
  - It's on the slides, so try not to cheat…
- Wednesday:

Evaluating our Regression Model

Each parameter of an independent variable **x** has an associated confidence interval

If the parameter / coefficient is not significantly distinguishable from 0 then we cannot assume that there is a significant linear relationship between that independent variable and the observations **y** (i.e. if the interval includes 0)

- Just because the coefficient is close to 0, you cannot assume there is no linear relationship

Confidence Intervals

How do we build a confidence interval?

Assume $Y_i \sim N(5, 25)$, for $1 \le i \le 100$ and $y_i = \mu + \epsilon$ where $\epsilon \sim N(0, 25)$. Then the Least Squares estimator of μ (μ_{LS}) is

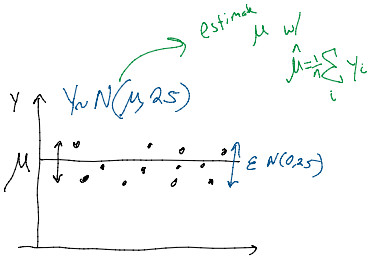
the sample mean $\bar{y}$

What is the 95% confidence interval for μ_{LS}?

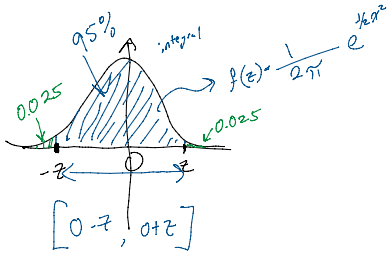
$SE(\mu_{LS}) = \sigma_\epsilon / \sqrt{n}$
$= 5 / \sqrt{100}$
$= .5$

$CI_{.95} = [\bar{y} - 1.96 \times SE(\mu_{LS}), \bar{y} + 1.96 \times SE(\mu_{LS})]$
$= [\bar{y} - 1.96 \times .5, \bar{y} + 1.96 \times .5]$

Z-value for 95% Confidence Interval



- How do we build confidence intervals
- Yi is a normal random variable of mean 5 and standard deviation 25. We have 100 observations of Y.
- Epsilon is a normal variable with mean 0 and sigma of 25.
- Estimate mu with the sample mean (mu-hat)
- What is the 95% confidence interval?
- z-value for the 95% confidence interval and SE



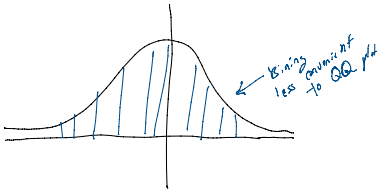
Z-values

These are the number of standard deviations from the mean of a N(0,1) distribution required in order to contain a specific % of values were you to sample a large number of times.

To find the .95 z-value (the number of standard deviations from the mean that contains 95% of values) you need to solve:

$$\int_{-z}^{z} \frac{1}{2\pi} e^{-\frac{1}{2}x^2} dx = .95$$

- z-value: the number of standard deviations you need to go away from the mean of N(0, 1) to contain a % of



QQ plot

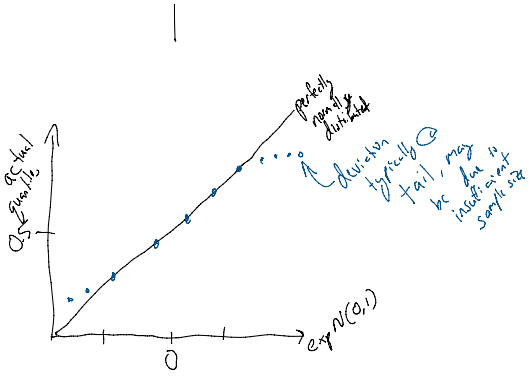
QQ plot

We need to check our assumption that our residuals / noise estimates are normally distributed.

How do can you check that a variable follows a specific distribution?

Need to check that our variable is **distributed** in the same way that a variable following our target distribution would be.

Plot the quantile of your target distribution against the quantiles of your data/ variable! If they match then your data probably comes from that distribution.



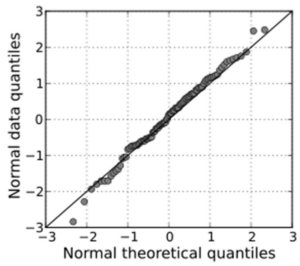
- We assumed the noise was normally distributed
- Expected quantiles vs actual quantiles observed in your data

QQ plot

Quantiles are the values for which a particular % of values are contained below it.

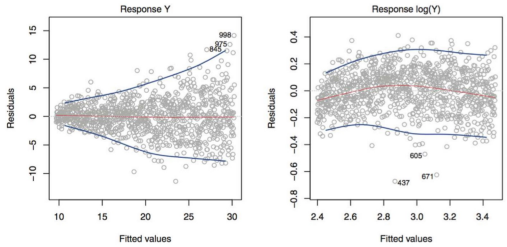
For example the 50% quantile of a N(0,1) distribution is 0 since 50% of samples would be contained below 0 were you to sample a large number of times.

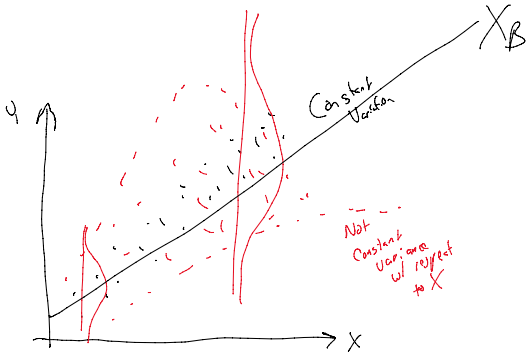
QQ plot



Constant Variance

One of our assumptions was that our noise had constant variance. How can we verify this?
We can plot our fitted values against our residuals (noise estimates)





- We can't get the true errors, but we have the residuals
- Left plot is not constant variation
- Right plot is
- You could transform Y
  - Try different transformations
- Square root transforms, and other similar ones

Extending our Linear Model

Changing the assumptions we made can drastically change the problem we are solving. A few ways to extend the linear model:

1. Non-constant variance - used in WLS (weighted least squares)
2. Distribution of error is not Normal - used in GLM (generalized linear models)

- You could verify this experimentally.
- You could use an exponential or something like that
- Extend both linear and logistic regression models we've seen
- Weighted least squares
- You could try different distributions around Y
- Generalized linear models
- If your parameters are not significantly far from 0…
- If your QQ-plot doesn't follow a linear distribution
    - Maybe a linear model isn't the best because you can't validate that assumption

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\linear-regression.py

```python
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from sklearn import datasets
import statsmodels.api as sm

import seaborn as sns; sns.set()

# We generate our data
# Then apply our transfrom of X and X transpose to the data
# If you think X2 may be a term, you need to include it in the terms you try

# plot line
# We know what the parameters are Beta0 is 1, Beta1 is 0.5
line = np.array([1, 0.5])
xlin = -10.0 + 20.0 * np.random.random(100)
ylin = line[0]+(line[1]*xlin)+np.random.randn(100)
plt.plot(xlin,ylin,'ro',markersize=4)
plt.show()

# plot quadratic
# There are ways to explore your data so you can decide what parameters to try
quad = np.array([1, 3, 0.5])
xquad = -10.0 + 20.0 * np.random.random(100)
# Calling it xquad is a little misleading
# How did you know if was X2? That's where being a data scientist comes in
yquad = quad[0]+(quad[1]*xquad)+(quad[2]*xquad*xquad)+np.random.randn(100)
plt.plot(xquad,yquad,'ro',markersize=4)
plt.show()

# plot log
# You can see that shape is clearly logarithmic so you need to include a log term
# If we get negative points, then our model would bbe
```

```
34 clearly wrong and it wasn't actually log
35 # If you repeat this many times, you expect on average
    you will get the true values you had generated
36 log = np.array([1, 4])
37 xlog = 10.0 * np.random.random(100)
38 ylog = log[0]+log[1]*np.log(xlog)+np.random.randn(100)
39 plt.plot(xlog,ylog,'ro',markersize=4)
40 plt.show()
41
42 # plot line through first plot
43 line = np.array([1, 0.5])
44 xlin = -10.0 + 20.0 * np.random.random(100)
45 ylin = line[0]+(line[1]*xlin)+np.random.randn(100)
46 plt.plot(xlin,ylin,'ro',markersize=4)
47 plt.plot(xlin,line[0]+line[1]*xlin,'b-')
48 plt.text(-9,3,r'$y = \beta_0 + \beta_1x$',size=20)
49 plt.show()
50
51 # Least square estimate through first plot
52 m = np.shape(xlin)[0]
53 X = np.array([np.ones(m),xlin]).T
54 beta = np.linalg.inv(X.T @ X) @ X.T @ ylin
55
56 xplot = np.linspace(-10,10,50)
57 yestplot = beta[0]+beta[1]*xplot
58 plt.plot(xplot,yestplot,'b-',lw=2)
59 plt.plot(xlin,ylin,'ro',markersize=4)
60 plt.show()
61 print(beta)
62
63 # Least square estimate through second plot
64 m = np.shape(xquad)[0]
65 X = np.array([np.ones(m),xquad,xquad**2]).T
66 beta = np.linalg.inv(X.T @ X) @ X.T @ yquad
67
68 xplot = np.linspace(-10,10,50)
69 yestplot = beta[0]+beta[1]*xplot+beta[2]*xplot**2
70 plt.plot(xplot,yestplot,'b-',lw=2)
71 plt.plot(xquad,yquad,'ro',markersize=4)
72 plt.show()
73 print(beta)
```

```
 74
 75 # Least square estimate through third plot
 76 m = np.shape(xlog)[0]
 77 X = np.array([np.ones(m),np.log(xlog)]).T
 78 beta = np.linalg.inv(X.T @ X) @ X.T @ ylog
 79
 80 xplot = np.linspace(-10,10,50)
 81 yestplot = beta[0]+beta[1]*np.log(xplot)
 82 plt.plot(xplot,yestplot,'b-',lw=2)
 83 plt.plot(xlog,ylog,'ro',markersize=4)
 84 plt.show()
 85 print(beta)
 86
 87 # using libraries
 88 # There are libraries to do this for you
 89 # We can plot this in 3D (pycharm showed it flat, but
     there may be a way to manipulate it)
 90 X, y = datasets.make_regression(n_samples=100,
    n_features=2, n_informative=5, noise=30, random_state
    =1)
 91 X = sm.add_constant(X)
 92 ax = plt.axes(projection='3d')
 93 ax.scatter3D(X.T[1], X.T[2], y, cmap='ro')
 94 plt.show()
 95
 96 model = sm.OLS(y, X)
 97 results = model.fit()
 98
 99 ax = plt.axes(projection='3d')
100 ax.scatter3D(X.T[1], X.T[2], y)
101
102 x1, x2 = np.meshgrid(np.arange(min(X.T[1]), max(X.T[1
    ]), .5), np.arange(min(X.T[2]), max(X.T[2]), .5))
103 exog = pd.core.frame.DataFrame({'x0': np.ones(len(x1.
    ravel())).ravel(), 'x1': x1.ravel(), 'x2':x2.ravel
    ()})
104 print(exog)
105 out = results.predict(exog=exog)
106 ax.plot_surface(x1, x2, out.values.reshape(x1.shape
    ), color='Green', alpha=.2)
107 plt.show()
```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\linear-regression.py

```
108
109 # Evaluating the Modal
110 print(results.summary())
111
```



```
./linear-regression.py
[1.01858886 0.51037431]
[1.24622978 3.00077925 0.49682696]s\GitHub\CS506-Fall2021\10-regression\linear-regression.py:70:
RuntimeWarning: invalid value encountered in log
  yestplot = beta[0]+beta[1]*np.log(xplot)
[0.94582328 4.0255589 ]
    x0       x1       x2
0   1.0 -2.434838 -1.857982
1   1.0 -1.934838 -1.857982
2   1.0 -1.434838 -1.857982
3   1.0 -0.934838 -1.857982
4   1.0 -0.434838 -1.857982
..  ...      ...      ...
85  1.0  0.065162  2.142018
86  1.0  0.565162  2.142018
87  1.0  1.065162  2.142018
88  1.0  1.565162  2.142018
89  1.0  2.065162  2.142018
```

[90 rows x 3 columns]

OLS Regression Results

Dep. Variable:	y	R-squared:	0.840
Model:	OLS	Adj. R-squared:	0.836
Method:	Least Squares	F-statistic:	254.1
Date:	Wed, 20 Oct 2021	Prob (F-statistic):	2.72e-39
Time:	17:35:38	Log-Likelihood:	-482.37
No. Observations:	100	AIC:	970.7
Df Residuals:	97	BIC:	978.5
Df Model:	2		
Covariance Type:	nonrobust		

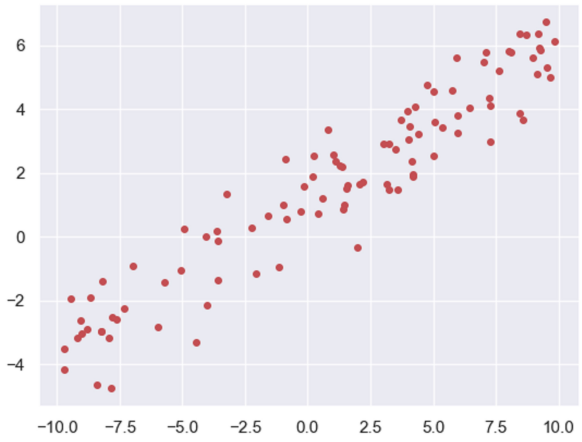
	coef	std err	t	P>|t|	[0.025	0.975]
const	2.1912	3.162	0.693	0.490	-4.085	8.467
x1	29.3912	3.274	8.977	0.000	22.893	35.889
x2	78.1391	3.594	21.741	0.000	71.006	85.272

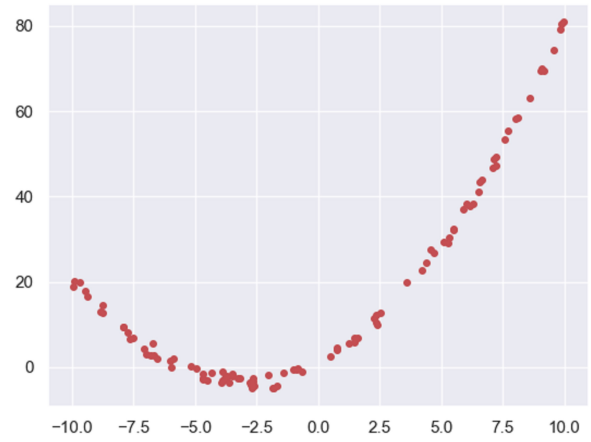
Omnibus:	1.279	Durbin-Watson:	1.824
Prob(Omnibus):	0.527	Jarque-Bera (JB):	1.065
Skew:	0.253	Prob(JB):	0.587
Kurtosis:	2.999	Cond. No.	1.38

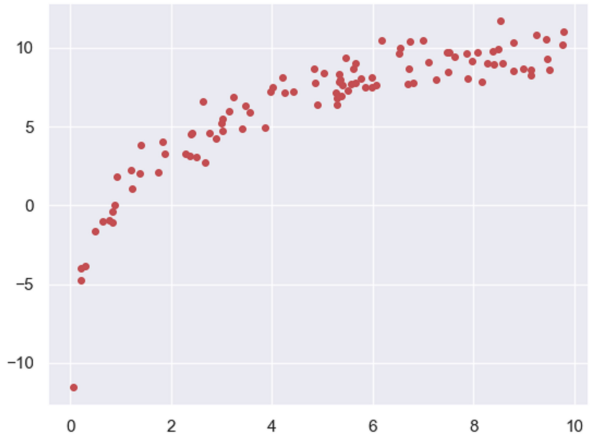
Notes:

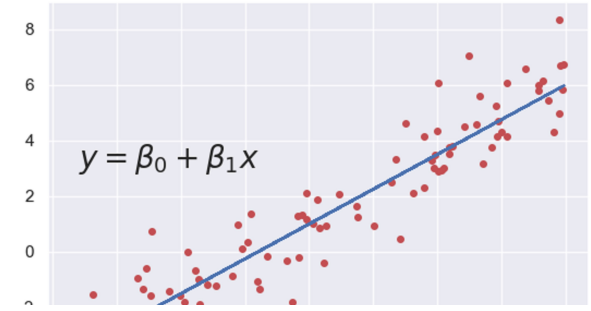
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Process finished with exit code 0









File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\logistic-regression.py

```
 1 import numpy as np
 2 import pandas as pd
 3 from mpl_toolkits.mplot3d import Axes3D
 4 import matplotlib.pyplot as plt
 5 from sklearn import datasets
 6 import statsmodels.api as sm
 7 import seaborn as sns; sns.set()
 8
 9 # Logisitc regression is a little more complicated
10
11 df = pd.read_csv('ats-admissions.csv')
12 print(df.head(10))
13
14 print()
15
```

```
print(df.head(10))

print()

# Describe dataset using pandas library:
# Get average GRE
# Most students get rejected
# Quantile for 50% is 0, so at least 50% get's rejected
print("DESCRIBE")
print(df.describe())
# If you see an increase in GRE by 1 what is the increase in log-odds... need to apply logit-inverse to get probability
# GPA looks more important than GRE, BUT GPA range is less
# Resolve some issues of scale through normalization
# The way you interpret these coefficients is different than it would be in lienar regression


# Can see histogram distributions

df.hist()
plt.show()

df['intercept'] = 1.0
train_cols = df.columns[1:]
train_cols
logit = sm.Logit(df['admit'], df[train_cols])
```

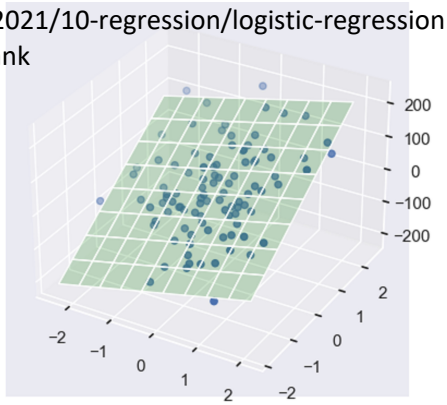

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\logistic-regression.py

```
# fit the model
result = logit.fit()

print()
print("SUMMARY")
print(result.summary())

```

/GitHub/CS506-Fall2021/10-regression/logistic-regression.py



	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4
5	1	760	3.00	2
6	1	560	2.98	1
7	0	400	3.08	2
8	1	540	3.39	3
9	0	700	3.92	2

DESCRIBE

	admit	gre	gpa	rank
count	400.000000	400.000000	400.000000	400.00000
mean	0.317500	587.700000	3.389900	2.48500
std	0.466087	115.516536	0.380567	0.94446
min	0.000000	220.000000	2.260000	1.00000
25%	0.000000	520.000000	3.130000	2.00000
50%	0.000000	580.000000	3.395000	2.00000
75%	1.000000	660.000000	3.670000	3.00000
max	1.000000	800.000000	4.000000	4.00000

Optimization terminated successfully.
Current function value: 0.574302
Iterations 6

SUMMARY

Logit Regression Results

Dep. Variable:	admit	No. Observations:	400
Model:	Logit	Df Residuals:	396
Method:	MLE	Df Model:	3
Date:	Mon, 25 Oct 2021	Pseudo R-squ.:	0.08107
Time:	17:44:35	Log-Likelihood:	-229.72
converged:	True	LL-Null:	-249.99
Covariance Type:	nonrobust	LLR p-value:	8.207e-09

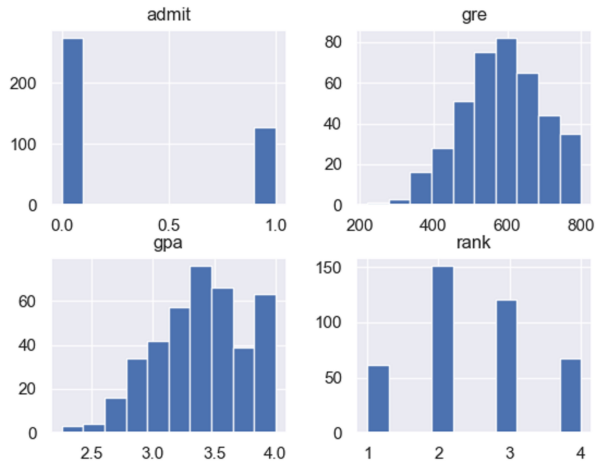
	coef	std err	z	P>|z|	[0.025	0.975]
gre	0.0023	0.001	2.101	0.036	0.000	0.004
gpa	0.7770	0.327	2.373	0.018	0.135	1.419
rank	-0.5600	0.127	-4.405	0.000	-0.809	-0.311

intercept -3.4495 1.133 -3.045 0.002 -5.670 -1.229

==========

==========

Process finished with exit code 0



```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
import sklearn.datasets as datasets
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures


# Here we are just generating data and seeing how model performs:
# Just want to generate a line that separates our points:
# Want to separate the two classes
# Coloring indicates confidence (shades of red and shade of blue)
# The boundary is a line
# Changing how the log odds change as a function of X
# Close to a decision boundary, we are not very confident
# Far from a decision bounday, we are more confident
# But maybe you shouldn't be confident in ranges of data you haven't seen...
# Epoch: every iteration of logistic regression

# LINE
def generate_line_data():
    centers = [[0, 0]]
    t, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=1, random_state=0)
    # create some space between the classes
    X = np.array(list(filter(lambda x : x[0] - x[1] < -.5 or x[0] - x[1] > .5, t)))
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y

# AND
def generate_and_data():
    X = np.array([
        [0,0],
```

```
34          [0,1],
35          [1,0],
36          [1,1]])
37      Y = np.array([x[0] and x[1] for x in X])
38      return X, Y
39
40  # OR
41  def generate_or_data():
42      X = np.array([
43          [0,0],
44          [0,1],
45          [1,0],
46          [1,1]])
47      Y = np.array([x[0] or x[1] for x in X])
48      return X, Y
49
50  # XOR: one needs to be true, but NOT BOTH
51  # XOR
52  def generate_xor_data():
53      X = np.array([
54          [0,0],
55          [0,1],
56          [1,0],
57          [1,1]])
58      Y = np.array([x[0]^x[1] for x in X])
59      return X, Y
60
61  # Do only one of the following lines:
62  X, Y = generate_line_data()
63  # X, Y = generate_xor_data()
64
65  cs = np.array([x for x in '
    bgrcmykbgrcmykbgrcmykbgrcmyk'])
66  cs = np.hstack([cs] * 20)
67
68  plt.scatter(X[:,0],X[:,1],color=cs[Y].tolist(), s=50,
    alpha=0.8)
69  plt.show()
70
71  # for xor to be fit
72  # Put in an interaction term:
```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\logistic-regression-2.py

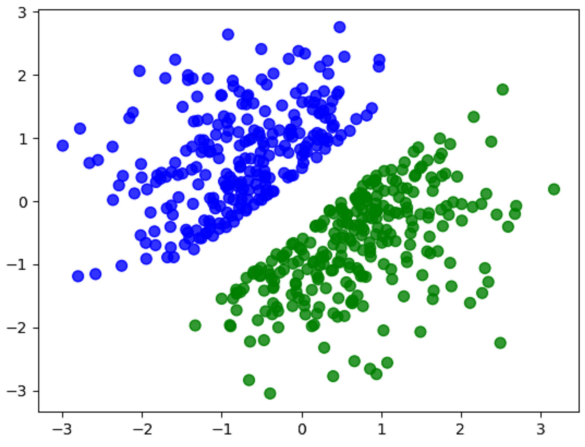
```
 73 # Bend the space in your head
 74 # poly = PolynomialFeatures(interaction_only=True)
 75 # lr = LogisticRegression(penalty='none', verbose=2,
    solver='sag')
 76 # model = make_pipeline(poly, lr).fit(X, Y)
 77
 78 model = LogisticRegression(penalty='none', verbose=2
    , solver='sag').fit(X, Y)
 79
 80 # create a mesh to plot in
 81 h = .02  # step size in the mesh
 82 x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
 83 y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
 84 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 85                      np.arange(y_min, y_max, h))
 86 meshData = np.c_[xx.ravel(), yy.ravel()]
 87
 88 fig, ax = plt.subplots()
 89 A = model.predict_proba(meshData)[:, 1].reshape(xx.
    shape)
 90 Z = model.predict(meshData).reshape(xx.shape)
 91 ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
 92 ax.axis('off')
 93
 94 # Plot also the training points
 95 T = model.predict(X)
 96 T = T.reshape(X[:,0].shape)
 97 ax.scatter(X[:, 0], X[:, 1], color=cs[Y].tolist(), s=
    50, alpha=0.9)
 98 plt.show()
 99
100 # You can do completely different parameters
```

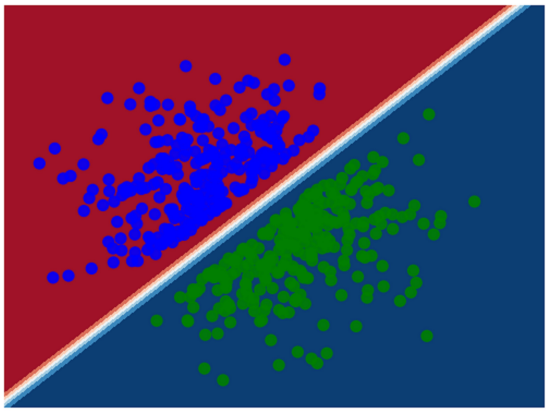

\anaconda3\envs\10-regression\python.exe
/Documents/GitHub/CS506-Fall2021/10-regression/logistic-regression-2.py
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
Epoch 1, change: 1.00000000
Epoch 2, change: 0.14896853
Epoch 3, change: 0.06579575
Epoch 4, change: 0.03147696
Epoch 5, change: 0.02061730
Epoch 6, change: 0.01145218
Epoch 7, change: 0.00575004
Epoch 8, change: 0.00586071
Epoch 9, change: 0.00527536
Epoch 10, change: 0.00505211
Epoch 11, change: 0.00484836
Epoch 12, change: 0.00465001
Epoch 13, change: 0.00447972
Epoch 14, change: 0.00430780

```
Epoch 15, change: 0.00412875
Epoch 16, change: 0.00402650
Epoch 17, change: 0.00391667
Epoch 18, change: 0.00377249
Epoch 19, change: 0.00366537
Epoch 20, change: 0.00357392
Epoch 21, change: 0.00349561
Epoch 22, change: 0.00341727
Epoch 23, change: 0.00331606
Epoch 24, change: 0.00323060
Epoch 25, change: 0.00316173
Epoch 26, change: 0.00310670
Epoch 27, change: 0.00303216
Epoch 28, change: 0.00295369
Epoch 29, change: 0.00289938
Epoch 30, change: 0.00283377
Epoch 31, change: 0.00277144
Epoch 32, change: 0.00272848
Epoch 33, change: 0.00267779
Epoch 34, change: 0.00261813
Epoch 35, change: 0.00258578
Epoch 36, change: 0.00253607
Epoch 37, change: 0.00249648
Epoch 38, change: 0.00244881
Epoch 39, change: 0.00240460
Epoch 40, change: 0.00236050
Epoch 41, change: 0.00232400
Epoch 42, change: 0.00228136
Epoch 43, change: 0.00224300
Epoch 44, change: 0.00220994
Epoch 45, change: 0.00217308
Epoch 46, change: 0.00213779
Epoch 47, change: 0.00211281
Epoch 48, change: 0.00208232
Epoch 49, change: 0.00205578
Epoch 50, change: 0.00202420
Epoch 51, change: 0.00198854
Epoch 52, change: 0.00195555
Epoch 53, change: 0.00193034
Epoch 54, change: 0.00190210
Epoch 55, change: 0.00188346
\anaconda3\envs\10-regression\lib\site-packages\sklearn\linear_model
\_sag.py:328: ConvergenceWarning: The max_iter was reached which
means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
Epoch 56, change: 0.00185008
Epoch 57, change: 0.00182315
Epoch 58, change: 0.00179910
Epoch 59, change: 0.00177932
Epoch 60, change: 0.00175837
Epoch 61, change: 0.00173586
Epoch 62, change: 0.00171463
```

Epoch 63, change: 0.00169269
Epoch 64, change: 0.00167314
Epoch 65, change: 0.00164708
Epoch 66, change: 0.00163033
Epoch 67, change: 0.00160951
Epoch 68, change: 0.00159285
Epoch 69, change: 0.00157236
Epoch 70, change: 0.00155575
Epoch 71, change: 0.00153588
Epoch 72, change: 0.00152241
Epoch 73, change: 0.00150488
Epoch 74, change: 0.00148661
Epoch 75, change: 0.00146849
Epoch 76, change: 0.00145317
Epoch 77, change: 0.00144007
Epoch 78, change: 0.00142373
Epoch 79, change: 0.00140691
Epoch 80, change: 0.00139099
Epoch 81, change: 0.00137831
Epoch 82, change: 0.00136370
Epoch 83, change: 0.00135124
Epoch 84, change: 0.00133897
Epoch 85, change: 0.00132015
Epoch 86, change: 0.00130561
Epoch 87, change: 0.00129553
Epoch 88, change: 0.00128364
Epoch 89, change: 0.00127005
Epoch 90, change: 0.00125509
Epoch 91, change: 0.00124610
Epoch 92, change: 0.00123302
Epoch 93, change: 0.00121997
Epoch 94, change: 0.00120799
Epoch 95, change: 0.00119791
Epoch 96, change: 0.00118784
Epoch 97, change: 0.00117598
Epoch 98, change: 0.00116779
Epoch 99, change: 0.00115456
Epoch 100, change: 0.00114649
max_iter reached after 0 seconds

Process finished with exit code 0


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3



```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
import sklearn.datasets as datasets
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures


# Here we are just generating data and seeing how model performs:
# Just want to generate a line that separates our points:
# Want to separate the two classes
# Coloring indicates confidence (shades of red and shade of blue)
# The boundary is a line
# Changing how the log odds change as a function of X
# Close to a decision boundary, we are not very confident
# Far from a decision bounday, we are more confident
# But maybe you shouldn't be confident in ranges of data you haven't seen...
# Epoch: every iteration of logistic regression

# LINE
def generate_line_data():
    centers = [[0, 0]]
    t, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=1, random_state=0)
    # create some space between the classes
    X = np.array(list(filter(lambda x : x[0] - x[1] < -.5 or x[0] - x[1] > .5, t)))
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y

# AND
def generate_and_data():
    X = np.array([
        [0,0],
```

File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\logistic-regression-2.py

```
34          [0,1],
35          [1,0],
36          [1,1]])
37      Y = np.array([x[0] and x[1] for x in X])
38      return X, Y
39
40  # OR
41  def generate_or_data():
42      X = np.array([
43          [0,0],
44          [0,1],
45          [1,0],
46          [1,1]])
47      Y = np.array([x[0] or x[1] for x in X])
48      return X, Y
49
50  # XOR: one needs to be true, but NOT BOTH
51  # XOR
52  def generate_xor_data():
53      X = np.array([
54          [0,0],
55          [0,1],
56          [1,0],
57          [1,1]])
58      Y = np.array([x[0]^x[1] for x in X])
59      return X, Y
60
61  # Do only one of the following lines:
62  # X, Y = generate_line_data()
63  X, Y = generate_xor_data()
64
65  cs = np.array([x for x in '
    bgrcmykbgrcmykbgrcmykbgrcmyk'])
66  cs = np.hstack([cs] * 20)
67
68  plt.scatter(X[:,0],X[:,1],color=cs[Y].tolist(), s=50,
    alpha=0.8)
69  plt.show()
70
71  # for xor to be fit
72  # Put in an interaction term:
```


File - D:\Users\Wolfs\Documents\GitHub\CS506-Fall2021\10-regression\logistic-regression-2.py

```
73  # Bend the space in your head
74  poly = PolynomialFeatures(interaction_only=True)
75  lr = LogisticRegression(penalty='none', verbose=2,
    solver='sag')
76  model = make_pipeline(poly, lr).fit(X, Y)
77
78  # model = LogisticRegression(penalty='none', verbose=
    2, solver='sag').fit(X, Y)
79
80  # create a mesh to plot in
81  h = .02  # step size in the mesh
82  x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
83  y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
84  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
85                       np.arange(y_min, y_max, h))
86  meshData = np.c_[xx.ravel(), yy.ravel()]
87
88  fig, ax = plt.subplots()
89  A = model.predict_proba(meshData)[:, 1].reshape(xx.
    shape)
90  Z = model.predict(meshData).reshape(xx.shape)
91  ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
92  ax.axis('off')
93
```

```
shape)
Z = model.predict(meshData).reshape(xx.shape)
ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
ax.axis('off')

# Plot also the training points
T = model.predict(X)
T = T.reshape(X[:,0].shape)
ax.scatter(X[:, 0], X[:, 1], color=cs[Y].tolist(), s=
50, alpha=0.9)
plt.show()

# You can do completely different parameters
```


\anaconda3\envs\10-regression\python.exe
/Documents/GitHub/CS506-Fall2021/10-regression/logistic-regression-2.py
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
Epoch 1, change: 1.00000000
Epoch 2, change: 1.39567069
Epoch 3, change: 0.67453046
Epoch 4, change: 0.30697204
Epoch 5, change: 0.20211083
Epoch 6, change: 0.14332801
Epoch 7, change: 0.10081086
Epoch 8, change: 0.09131291
Epoch 9, change: 0.09343216
Epoch 10, change: 0.07381825
Epoch 11, change: 0.06801616
Epoch 12, change: 0.06379240
Epoch 13, change: 0.05594364
Epoch 14, change: 0.05079137
Epoch 15, change: 0.04711705
Epoch 16, change: 0.04497621
Epoch 17, change: 0.04239083
Epoch 18, change: 0.03835497
Epoch 19, change: 0.03619255
Epoch 20, change: 0.03406630
Epoch 21, change: 0.03198480
Epoch 22, change: 0.03072087
Epoch 23, change: 0.02851905
Epoch 24, change: 0.02767053
Epoch 25, change: 0.02503580
Epoch 26, change: 0.02474171
Epoch 27, change: 0.02405730
Epoch 28, change: 0.02293391
Epoch 29, change: 0.02208032
Epoch 30, change: 0.02051730

Epoch 31, change: 0.01987694
Epoch 32, change: 0.01944321
Epoch 33, change: 0.01907238
Epoch 34, change: 0.01740106
Epoch 35, change: 0.01682955
Epoch 36, change: 0.01648007
Epoch 37, change: 0.01621288
Epoch 38, change: 0.01595421
Epoch 39, change: 0.01487491
Epoch 40, change: 0.01387118
Epoch 41, change: 0.01369680
Epoch 42, change: 0.01339562
Epoch 43, change: 0.01283312
Epoch 44, change: 0.01254500
Epoch 45, change: 0.01202148
Epoch 46, change: 0.01183312
Epoch 47, change: 0.01154774
Epoch 48, change: 0.01115757
Epoch 49, change: 0.01082604
Epoch 50, change: 0.01054332
Epoch 51, change: 0.01028367
Epoch 52, change: 0.01009903
Epoch 53, change: 0.00969182
Epoch 54, change: 0.00954914
Epoch 55, change: 0.00942291
Epoch 56, change: 0.00922072
Epoch 57, change: 0.00905023
Epoch 58, change: 0.00878672
Epoch 59, change: 0.00846779
Epoch 60, change: 0.00827361
Epoch 61, change: 0.00806477
Epoch 62, change: 0.00788909
Epoch 63, change: 0.00775723
Epoch 64, change: 0.00763932
Epoch 65, change: 0.00740807
Epoch 66, change: 0.00735359
Epoch 67, change: 0.00729991
Epoch 68, change: 0.00724701
Epoch 69, change: 0.00719487
Epoch 70, change: 0.00668870
Epoch 71, change: 0.00651733
Epoch 72, change: 0.00645970
Epoch 73, change: 0.00637427
Epoch 74, change: 0.00618700
Epoch 75, change: 0.00610185
Epoch 76, change: 0.00602377
Epoch 77, change: 0.00582693
Epoch 78, change: 0.00575623
Epoch 79, change: 0.00567597
Epoch 80, change: 0.00556168
Epoch 81, change: 0.00548295
Epoch 82, change: 0.00537470
Epoch 83, change: 0.00528865
Epoch 84, change: 0.00523061

```
Epoch 85, change: 0.00514793
Epoch 86, change: 0.00508684
Epoch 87, change: 0.00503992
Epoch 88, change: 0.00497590
Epoch 89, change: 0.00488319
Epoch 90, change: 0.00478053
Epoch 91, change: 0.00475021
Epoch 92, change: 0.00463861
Epoch 93, change: 0.00456175
Epoch 94, change: 0.00449247
Epoch 95, change: 0.00440878
Epoch 96, change: 0.00436213
Epoch 97, change: 0.00432381
Epoch 98, change: 0.00423762
Epoch 99, change: 0.00419076
Epoch 100, change: 0.00417327
max_iter reached after 0 seconds
\anaconda3\envs\10-regression\lib\site-packages\sklearn\linear_model
\_sag.py:328: ConvergenceWarning: The max_iter was reached which
means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished

Process finished with exit code 0
```