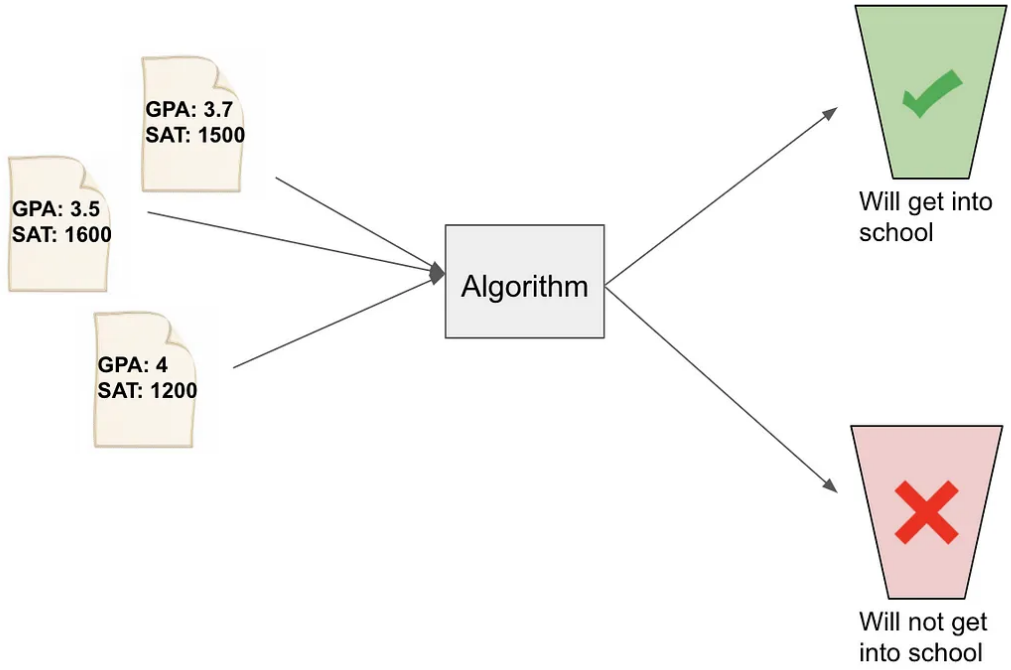
GETTING STARTED

# Binary classification and logistic regression for beginners

How to solve binary classification with gradient ascent

Lily Chen · Follow
Published in Towards Data Science
7 min read · Dec 2, 2020

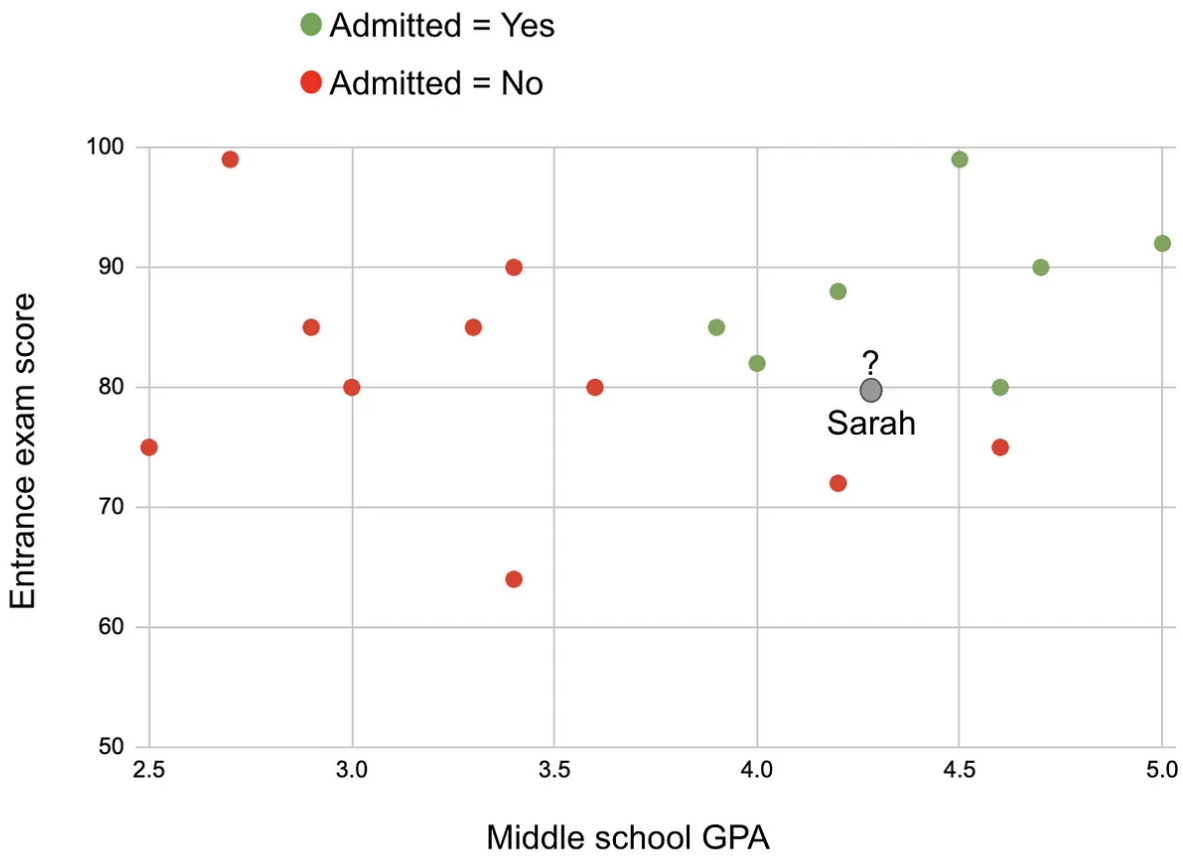





Binary classification (Image created by me)

Let's say you have a dataset where each data point is comprised of a middle school GPA, an entrance exam score, and whether that student is admitted to her town's magnet high school.

Given a new pair of `(GPA, exam score)` from Sarah, how can you predict whether Sarah will be admitted? Sarah's GPA is 4.3 and her exam score is 79.




We need to classify Sarah as "yes" or "no" for admission

This is a binary classification problem because we're predicting an outcome that can only be one of two values: "yes" or "no".

**The algorithm for solving binary classification is logistic regression.**

Before we delve into logistic regression, this article assumes an understanding of linear regression. This article also assumes familiarity with how gradient descent works in linear regression. Need a refresher? Read this:

**Linear regression and gradient descent for absolute beginners**

A simple explanation and implementation of gradient descent

towardsdatascience.com

Another way of asking *"will Sarah be admitted to magnet school"* is:

*"What is the probability of Sarah being admitted given her GPA and entrance exam score?"*

The mathematical way of representing this question is:

$$P(y = 1 \mid x; \theta)$$

"Probability of y equaling to 1 given x parameterized by theta"

This equation reads *"probability of y equaling to 1 given x parameterized by theta"*.

`y = 1` means "admitted". Conversely, `y = 0` means "not admitted".

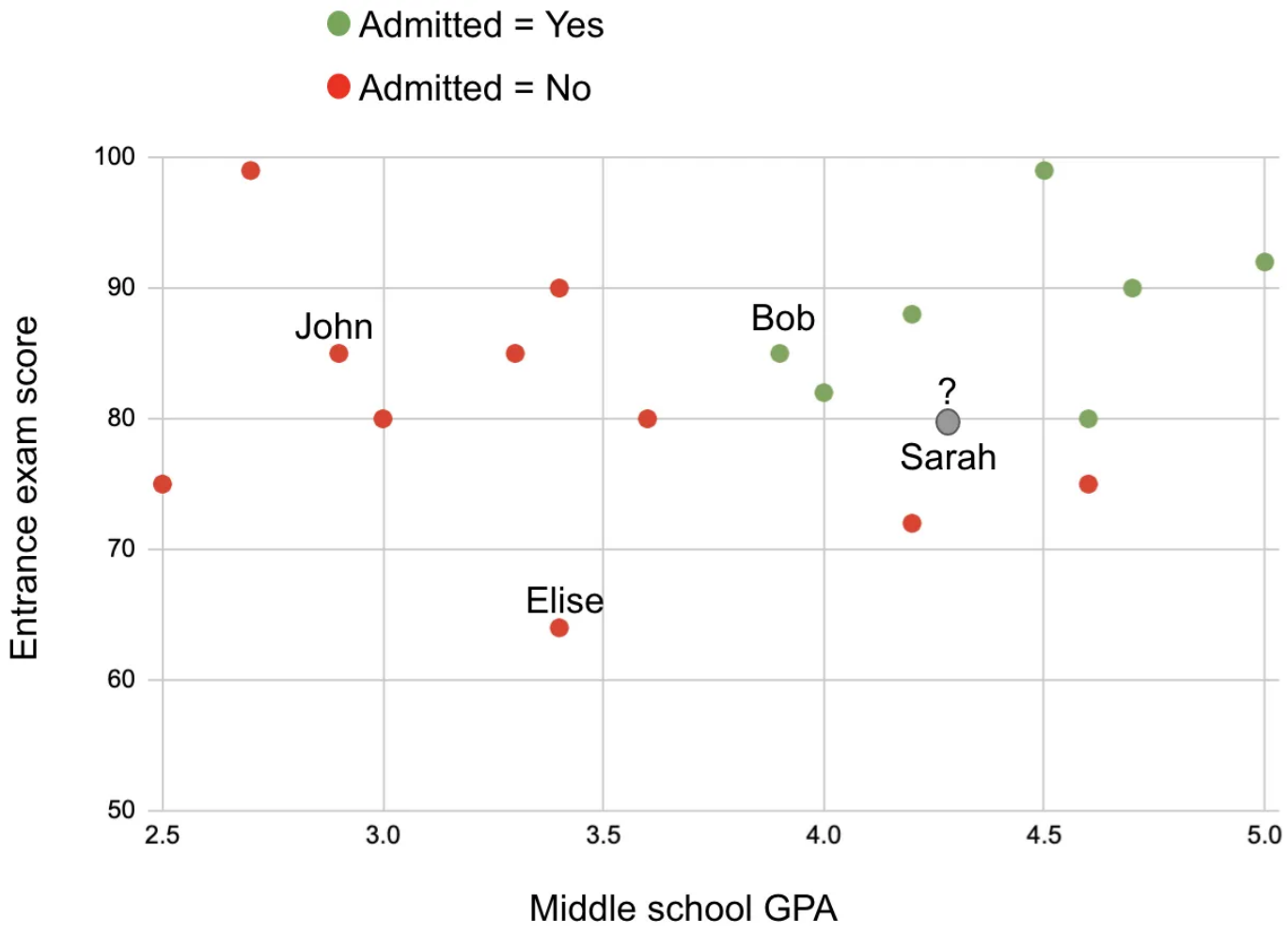
`x` is the set of features, which in this case, are GPA and entrance exam score.

`θ` is the parameters that describes how much GPA/exam score affect probability. Remember in linear regression, `θ` is the vector [y-intercept, slope] and the slope `m` of a line (y = mx + b) describes how much the variable `x` affects `y`.

Logistic regression is about finding this probability, i.e. P(y=1 | x; θ).

### The logic behind logistic regression

We don't know Sarah's admission status; but we do know the admission status of 17 other students.



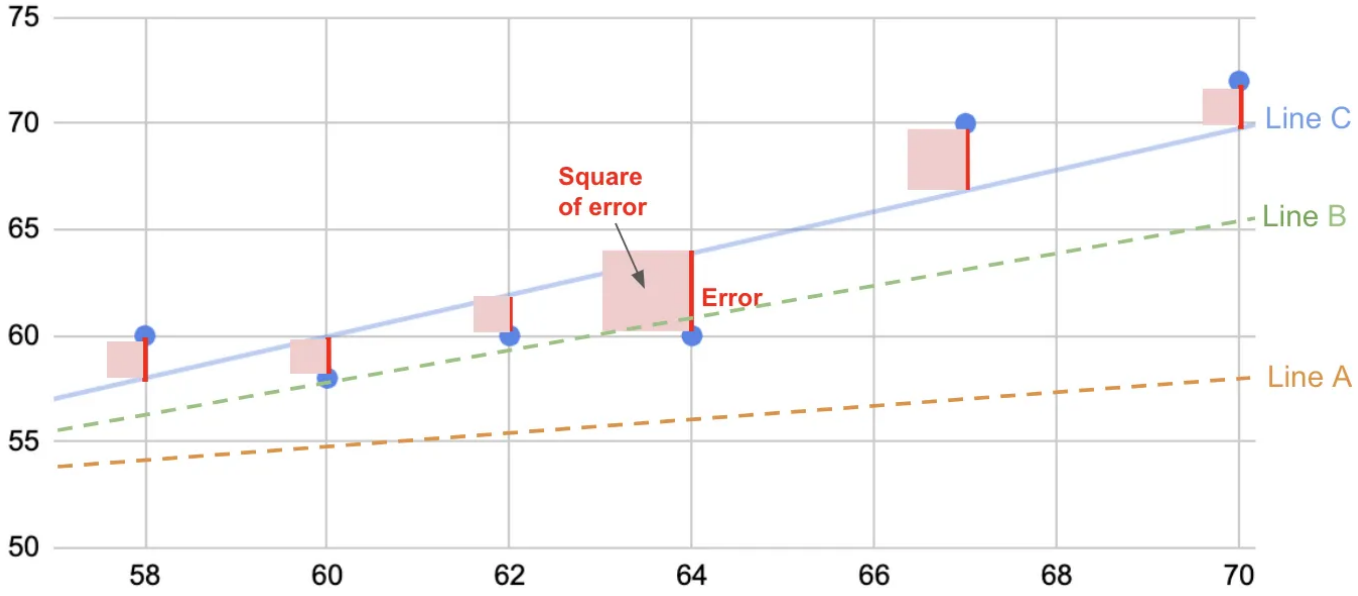

For instance, as the chart shows, we know that John is not admitted, Elise is not either, and Bob is. We also know the score and GPA for all of them.

The probability of John not being admitted is some number between 0 and 1. The probability of Bob being admitted is also somewhere between 0 and 1.

We want our model to maximize $P(y=0 \mid x; \theta)$ for John, and $P(y=1 \mid x; \theta)$ for Bob, and $P(y=0 \mid x; \theta)$ for Elise, etc. In logistic regression, we want to maximize the probability of all the data points given.

### Visualizing Logistic Regression

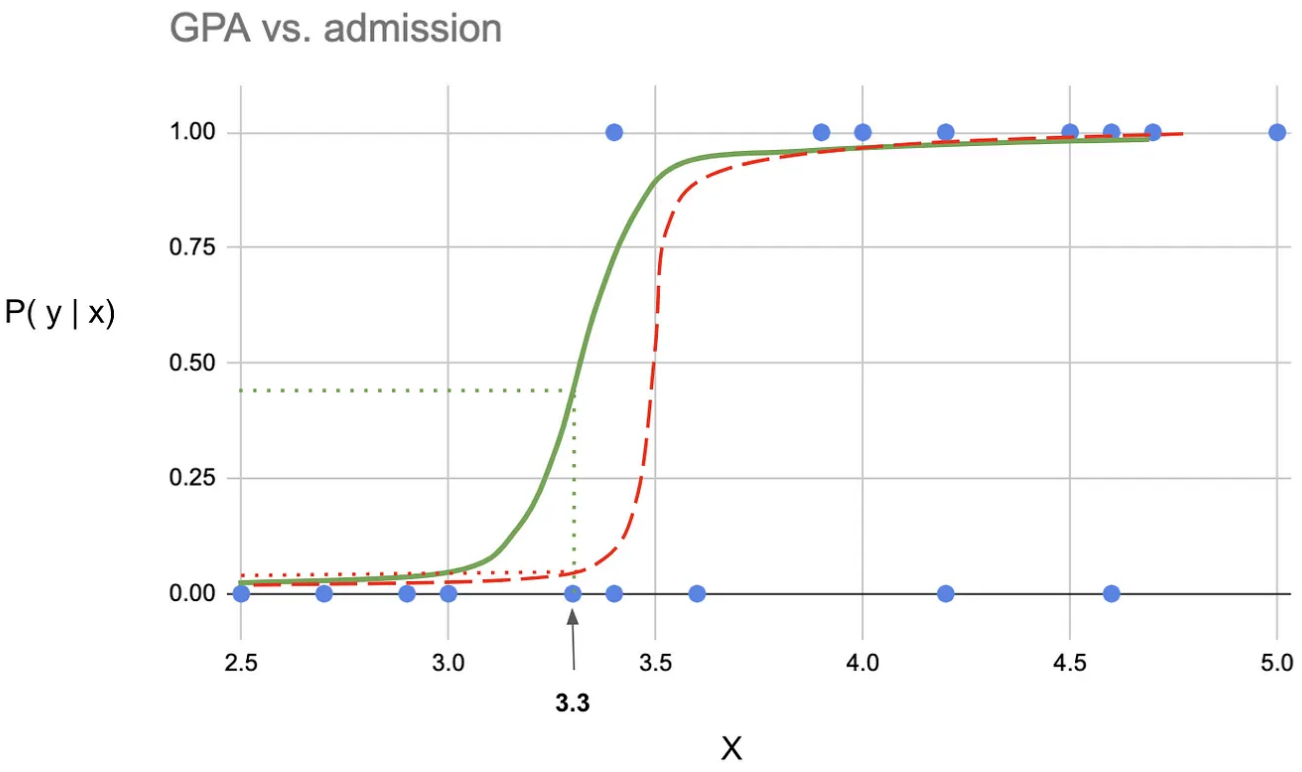
In linear regression and gradient descent, your goal is to arrive at the line of best fit by tweaking the slope and y-intercept little by little with each iteration. The line of best fit limits the sum of square of errors.




Linear regression is about finding line of least sum of squared errors

Obviously, finding the least square line makes less sense when you're doing classification.

To visualize logistic regression, let's start in 2D first, when you only have 1 feature instead of 2. In our case, let's only look at GPA.



The x-axis is the GPA. The y-axis is the probability that a student gets admitted given her GPA.

Since it's a binary classification, all the data points given have a y-value of either 0 or 1.

Instead of finding the least square regression line, you want to find a sigmoid function that best fit the dataset.

Which is a better fit? Red line or green line?

To answer this question, find where P(y | x) land for each GPA. For all your GPA values, you want P(y | x) to be as close as possible to the observed value of y (either 0 or 1). For instance, we know John is not admitted and his GPA is 2.7, so we want P(y | 2.7) to be close to 0. Similarly, Bob is admitted and his GPA is 3.8, so we want P(y | 3.8) to be close to 1. The exact math to compute P(y | x) will be discussed momentarily.

> Logistic regression is about finding a sigmoid function `h(x)` that maximizes the probability of your observed values in the dataset.

**Logistic regression algorithm**

Onto the math itself!

If you remember from statistics, the probability of eventA AND eventB occurring is equal to the probability of eventA times the probability of eventB.

P(A and B) = P(A) * P(B).

In logistic regression, we want to maximize probability for all of the observed values. Mathematically, the number we're trying to maximize can be written as:

$$L(\theta) = \Pi_{i=1}^{m} P(y^i | x^i; \theta)$$

Product of all probability of dataset. Pi means “product”.

`L(θ)` is what we want to maximize. In machine learning term, `L(θ)` is called “maximum likelihood estimation” or MLE.

$$P(y=1 \mid x; \theta) = h(x)$$

$$P(y=0 \mid x; \theta) = 1 - h(x)$$

$$P(y \mid x; \theta) = h(x)^y \cdot (1 - h(x))^{1-y}$$

The third function is a combination of the first two. If you plug in y = 0 or y = 1 into the third function, you get one of the first two.

In linear regression, h(x) takes the form `h(x) = mx + b` , which can be further written as such:

$$h(x) = mx + b$$

$$h(x) = [b\ m] \cdot \begin{bmatrix} 1 \\ x \end{bmatrix}$$

$$h(x) = [\theta_0\ \theta_1] \cdot \begin{bmatrix} 1 \\ x \end{bmatrix}$$

In logistic regression we use sigmoid function instead of a line.

$$h(x) = \frac{1}{1 + e^{-\theta^T X}}$$

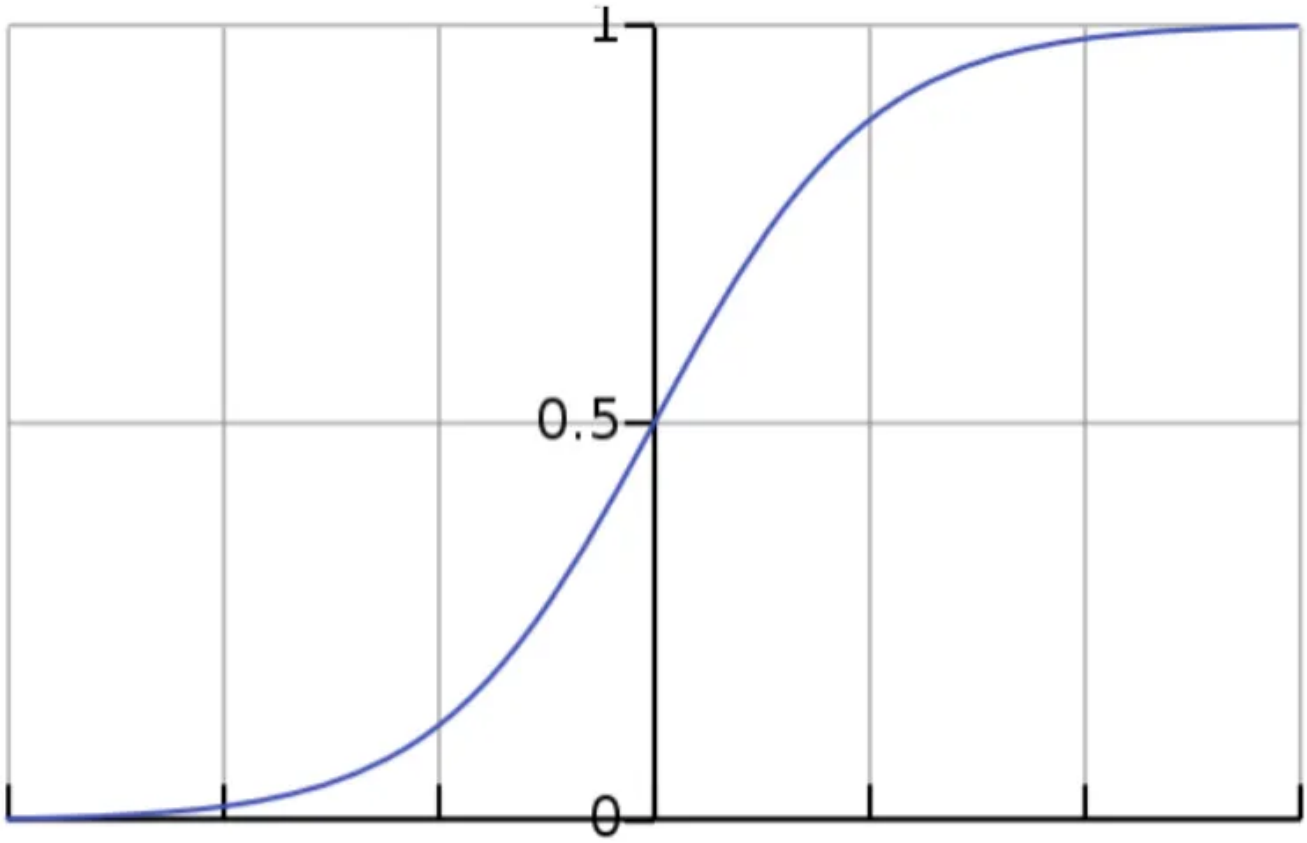
Sigmoid function for logistic regression

Taken together, this is the equation for P( y | x; θ)

$$P(y|x;\theta) = (\frac{1}{1 + e^{-\theta^T x}})^y \cdot (1 - \frac{1}{1 + e^{-\theta^T x}})^{1-y}$$

This is how you compute P(y | x) for all the datapoint. Remember, y is either 0 or 1.

The last step to logistic regression is finding good value for theta.




https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg

The way we go about finding the parameters in theta is similar to what we do in linear regression to find the line of best fit.

In linear regression, we adjust the y-intercept and slope through multiple iterations to arrive at the least square regression line.

In logistic regression, instead of minimizing the sum of squared errors (as in linear regression), we'll adjust the parameters of theta to maximize L(θ).

$$L(\theta) = \Pi_{i=1}^{m} P(y^i | x^i; \theta)$$

Product of all probability of dataset. Pi means "product".

From L(θ), we derive log(L(θ)), or l(θ).

$$L(\theta) = \Pi_{i=1}^{m} P(y^i | x^i; \theta)$$
$$L(\theta) = \Pi_{i=1}^{m} h(x^i)^{y^i} \cdot (1 - h(x^i))^{(1 - y^i)}$$
$$\ell(\theta) = log(L(\theta))$$
$$\ell(\theta) = \sum_{i=1}^{m} y^i log(h(x^i)) + (1 - y^i) log(1 - h(x^i))$$

Deriving l(θ) from L(θ)

The last equation for l(θ) is actually what the logistic regression algorithm maximizes. We take log of L(θ) purely to make the algorithm computationally easier.

Because we're trying to **maximize** a number here, the algorithm we'll use is called gradient **ascent**. This is in contrast to gradient *descent* used in linear regression where we're trying to *minimize* the sum of squared errors.

Note: you can also use gradient descent in logistic regression. After all, maximizing likelihood is the same as minimizing the negative of maximum likelihood.

To get the gradient ascent formula, we take the partial derivative of l(θ) with respect to theta.

$$new\ \theta_j = old\ \theta_j + \alpha \cdot \frac{\partial}{\partial \theta_j} \ell(\theta)$$
$$new\ \theta_j = old\ \theta_j + \alpha \cdot \sum_{i=1}^{m} (y^i - h(x)^i) \cdot x_j$$

Computation steps found in this article

If you were doing gradient descent instead, you take the partial derivative of negative $l(\theta)$ to arrive at the formula.

I've implemented logistic regression with gradient ascent in the gist show below.

```python
import numpy as np
import math

# example dataset:
# dataset = [
#     [4, 82, 1],
#     [3, 80, 0],
#     [2.5, 75, 0],
#     [3.4, 90, 1],
#     [4.2, 88, 1],
#     [5, 92, 1],
#     [2.7, 99, 0],
#     [3.3, 85, 0],
#     [4.2, 72, 0],
#     [3.6, 80, 0],
#     [2.9, 85, 0],
#     [3.9, 85, 1],
#     [4.5, 99, 1],
#     [4.7, 90, 1],
#     [4.6, 80, 1],
#     [4.6, 75, 0],
#     [3.4, 64, 0],
# ]

# each inner array (datapoint) represents one student.
# datapoint[0] = GPA
# datapoint[1] = exam score
# datapoint[2] = whether student was admitted. 1 = yes. 0 = no.

class LogisticRegression:
    def __init__(self, dataset):
        self.dataset = dataset
        self.alpha = 0.002  # alpha is "learning rate"

    # get new theta according to gradient ascent formula
    def get_theta(self, theta):
        num_params = len(self.dataset[0])
        new_gradients = [0] * num_params
        m = len(self.dataset)
        for i in range(0, len(self.dataset)):
            predicted = self.get_prediction(theta, self.dataset[i])
            actual = self.dataset[i][-1]
            for j in range(0, num_params):
                x_j = 1 if j == 0 else self.dataset[i][j - 1]
                new_gradients[j] += (actual - predicted) * x_j
```

```
                # new_gradients[j] += (predicted - actual) * x_j

        new_theta = [0] * num_params
        for j in range(0, num_params):
            new_theta[j] = theta[j] + self.alpha * (1/m) * new_gradients[j]

        return new_theta

    # uses sigmoid function. Outputs a number from 0 to 1, which denotes probability
    def get_prediction(self, theta, data_point):
        inner_val = self.matrix_dot_product(theta, data_point)
        probability = 1 / (1 + math.exp(- 1 * inner_val))
        return probability

    def matrix_dot_product(self, theta, data_point):
        values = [0]*len(data_point)
        for i in range(0, len(values)):
            values[i] = 1 if i == 0 else data_point[i-1]

        result = np.dot(theta, values)
        return result

    def calc_max_likelihood_estimate(self, theta):
        sum = 0
        for i in range(0, len(self.dataset)):
            predicted = self.get_prediction(theta, self.dataset[i])
            actual = self.dataset[i][-1]
            increment = actual * math.log(predicted) + (1 - actual) * math.log(1 - predict
            sum += increment

        return sum

    def iterate(self):
        num_iteration = 0
        current_probability = None
        # current_theta = [-1] * len(self.dataset[0])  # initialize to 0
        current_theta = [-109.99, 10.655, 0.821]

        while num_iteration < 1000:
            if num_iteration % 10 == 0:
                print('current iteration: ', num_iteration)
                print('current probability: ', current_probability)
                print('current theta: ', current_theta)
            new_probability = self.calc_max_likelihood_estimate(current_theta)
```

```
            current_probability = new_probability
            new_theta = self.get_theta(current_theta)
            current_theta = new_theta
            num_iteration += 1

        print(f'After {num_iteration}, total probability is {current_probability}. Theta i
```

logistic_regression.py hosted with ❤️ by GitHub view raw

Through a series of trial and error tweaking the learning rate alpha and initialized values for theta, I found the parameters [-109.99, 10.655, 0.821] to be a good fit for the model.

Using these parameters, the probability of Sarah being admitted is:

$$P = \frac{1}{1 + exp(-[-109.99\ 10.655\ 0.821] \cdot \begin{bmatrix} 1 \\ 4.3 \\ 79 \end{bmatrix})}$$

(Remember Sarah's GPA is 4.3 and her exam score is 79).

P = 0.665. Her chances aren't great, but she has a decent shot. She's more likely than not to be admitted. Thus, we'll classify her as "admitted."

This article talks about binary classification. In my next article, I will write about multiclass classification. Stay tuned!

You can find me on LinkedIn https://www.linkedin.com/in/yilingchen405/

Logistic Regression | Data Science | Machine Learning | Editors Pick | Getting Started

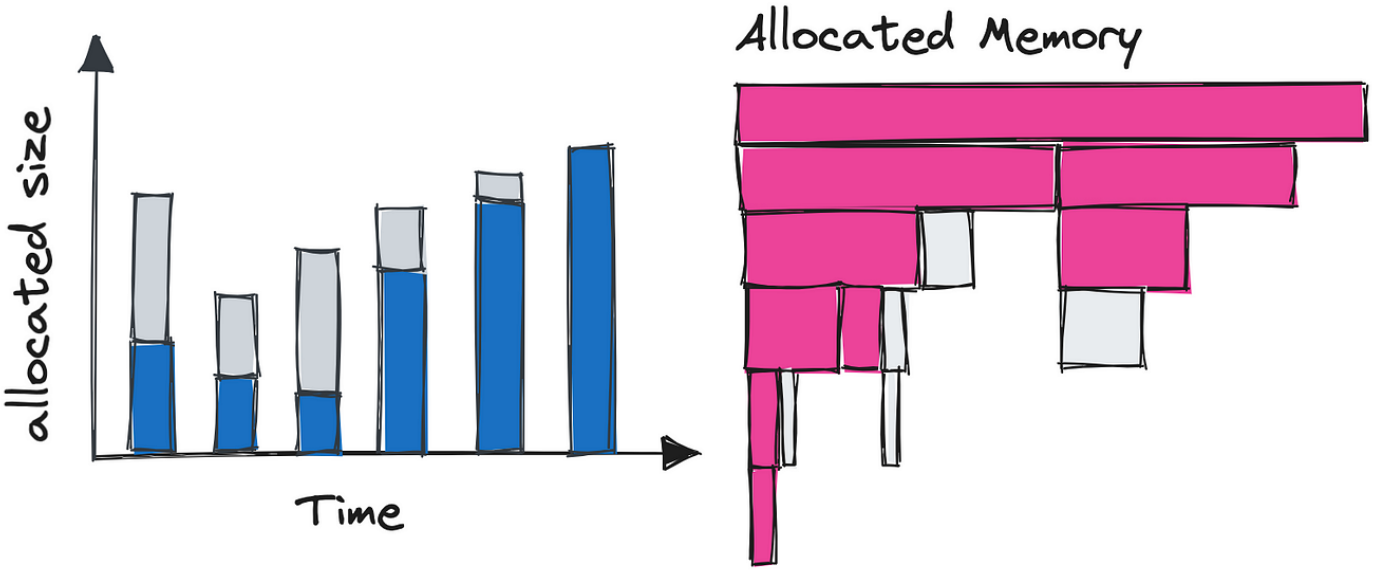
Follow

## Written by Lily Chen

7.2K Followers · Writer for Towards Data Science

Senior software engineer at Datadog. I write about tech and life. Portfolio: https://lilychencodes.com/

### More from Lily Chen and Towards Data Science

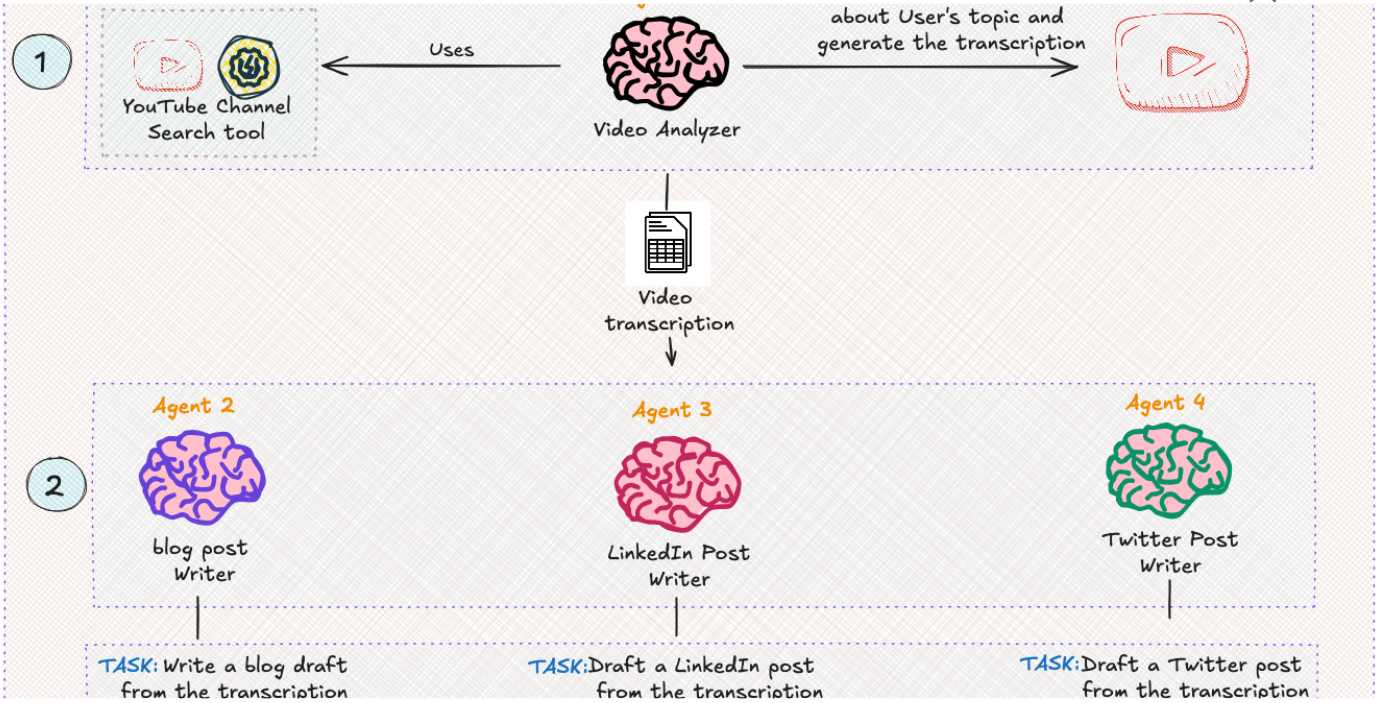



 Lily Chen in Performance engineering for the ordinary Barbie

## Intro to Memory Profiling & Chrome DevTools Memory Tab explained

I think the Homer hiding gif best captures my immediate reaction when I saw the Memory tab for the first time.

Oct 11, 2023 241 2




Zoumana Keita in Towards Data Science

# AI Agents—From Concepts to Practical Implementation in Python

This will change the way you think about AI and its capabilities

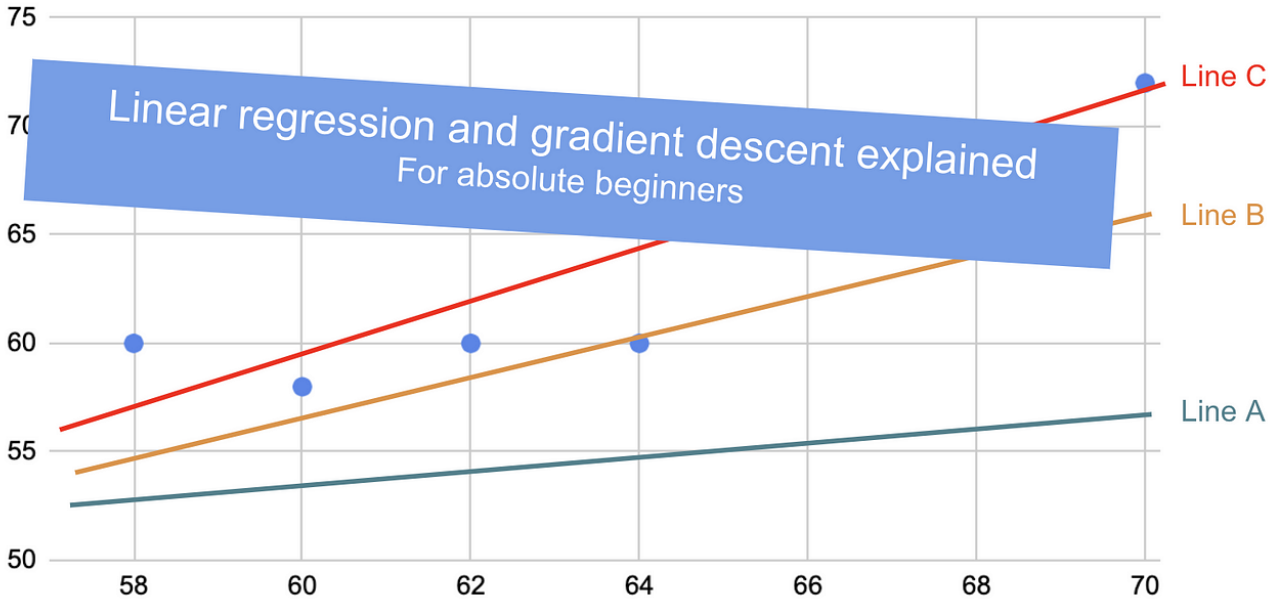
Aug 12 1K 13


Ahmed Besbes in Towards Data Science

## What Nobody Tells You About RAGs

A deep dive into why RAG doesn't always work as expected: an overview of the business value, the data, and the technology behind it.

Aug 22 1.4K 20




Lily Chen in Towards Data Science

## Linear regression and gradient descent for absolute beginners

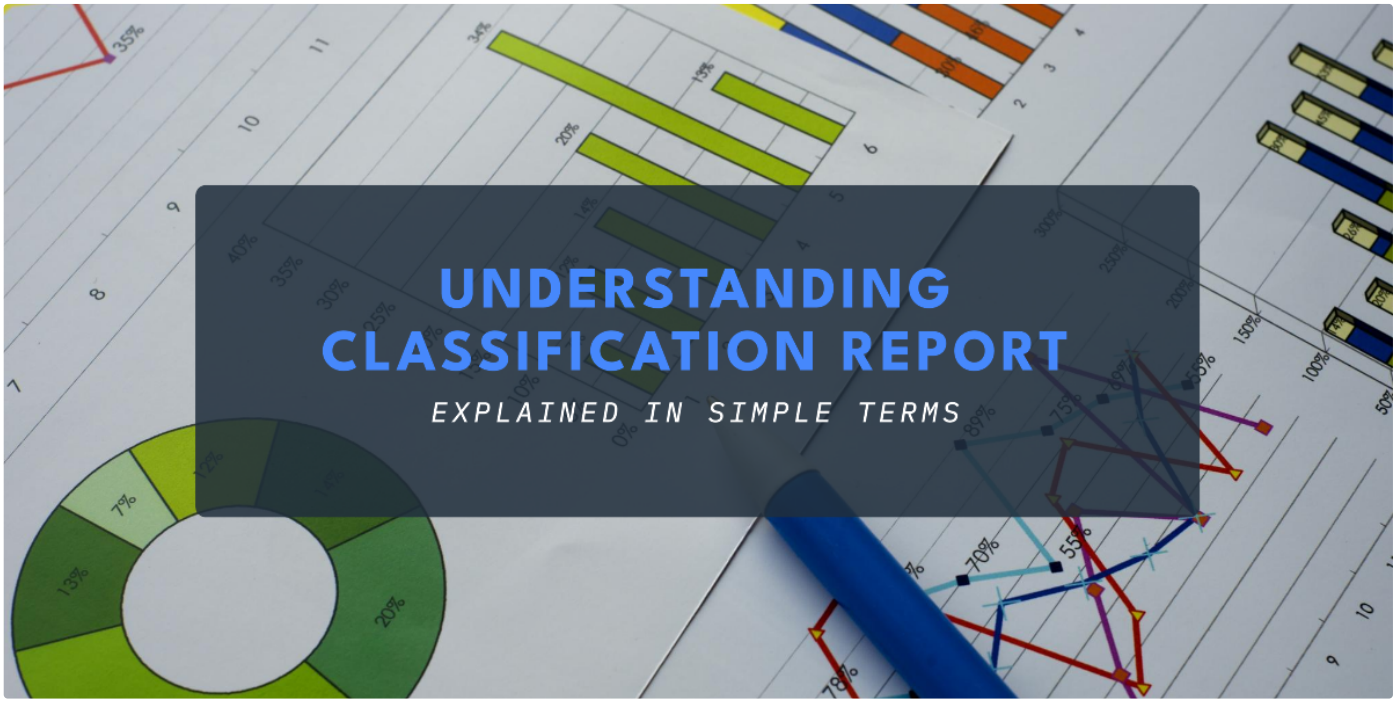
A simple explanation and implementation of gradient descent

Nov 26, 2020 631 4

See all from Lily Chen

See all from Towards Data Science

## Recommended from Medium

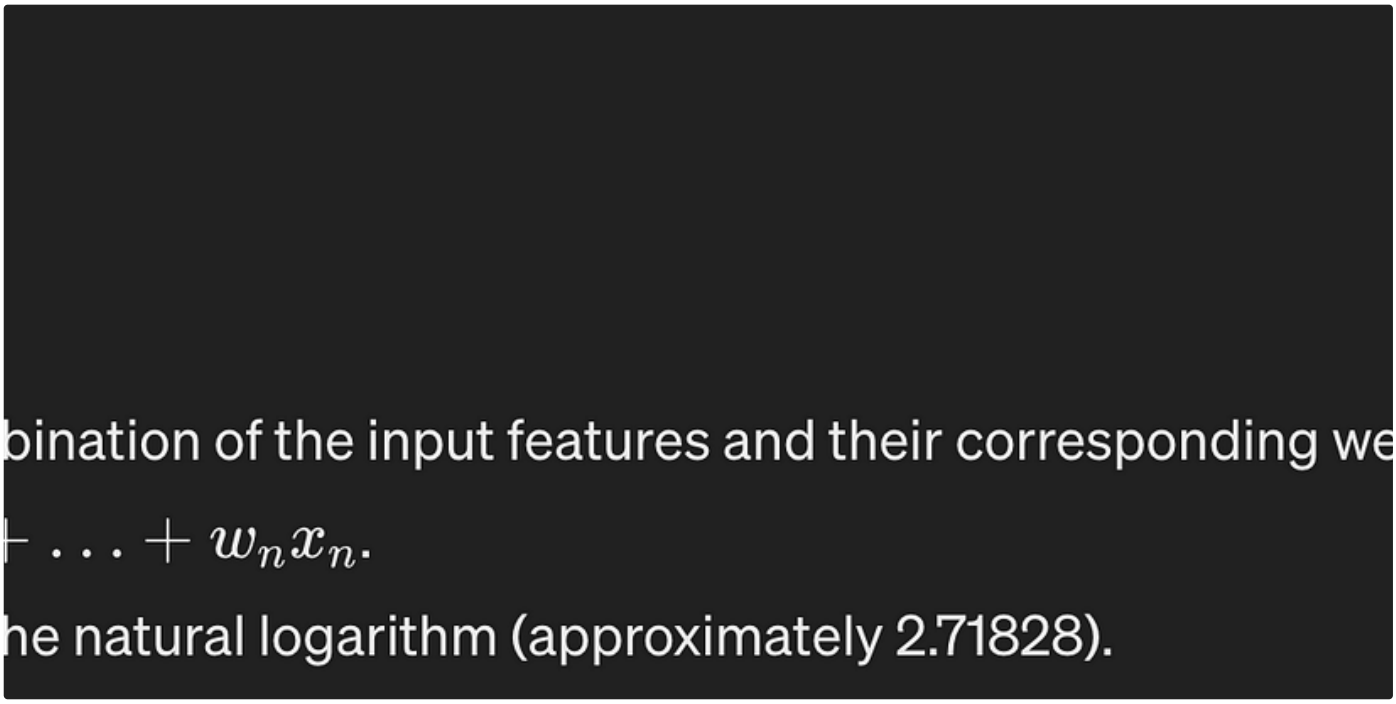



Chanaka Prasanna

### Classification Report Explained—Precision, Recall, Accuracy, Macro average, and Weighted Average

Why do we need a classification report?

Apr 21 110 3



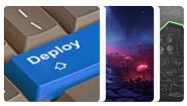

Avicsebooks

## ML Part 4: Linear Classification

What is logistic regression?

Apr 16 5

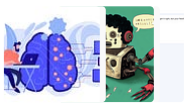
### Lists




**Predictive Modeling w/ Python**
20 stories · 1488 saves

**Practical Guides to Machine Learning**
10 stories · 1819 saves

**Natural Language Processing**
1675 stories · 1252 saves



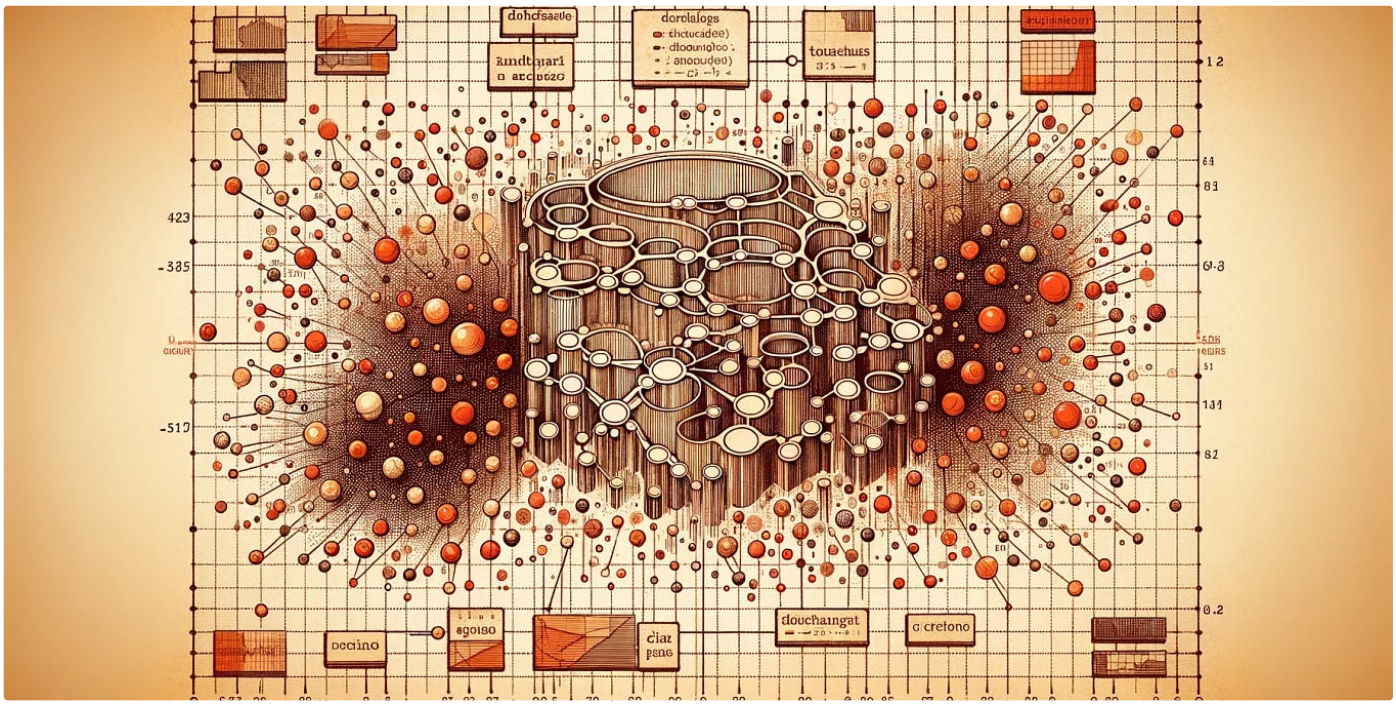
**The New Chatbots: ChatGPT, Bard, and Beyond**
12 stories · 454 saves

Irina (Xinli) Yu, Ph.D.

# Understanding Hypothesis Testing: T-Test, Z-Test, Chi-Square Test, and ANOVA

Hypothesis testing is a critical component of statistical analysis, allowing researchers to make inferences about populations based on...



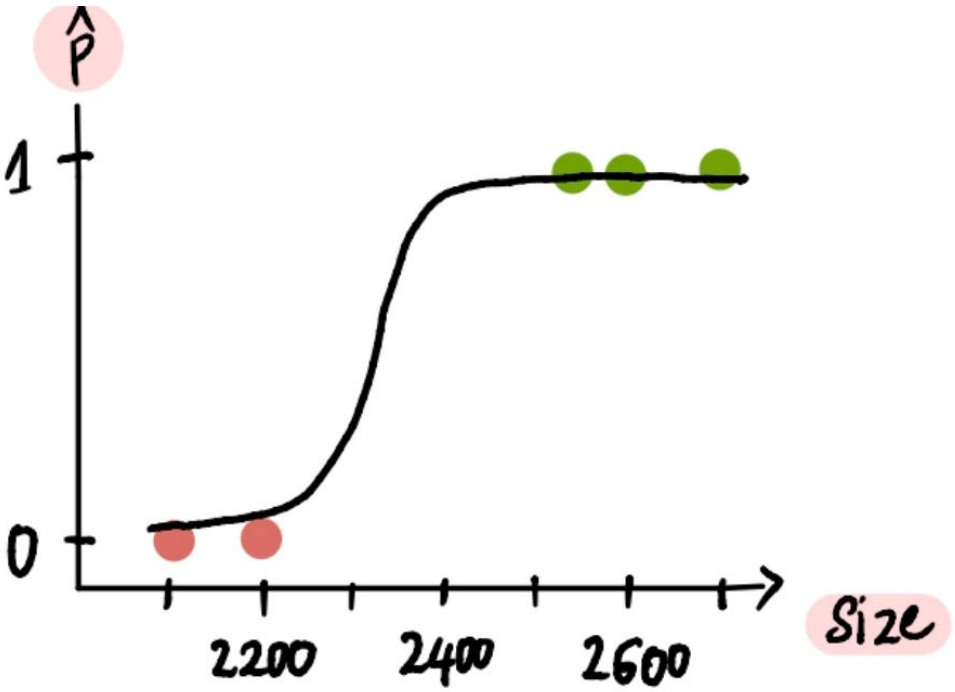


Cristian Leo in Towards Data Science

## The Math Behind K-Means Clustering

Why is K-Means the most popular algorithm in Unsupervised Learning? Let's dive into its math, and build it from scratch.

Feb 13 777 4

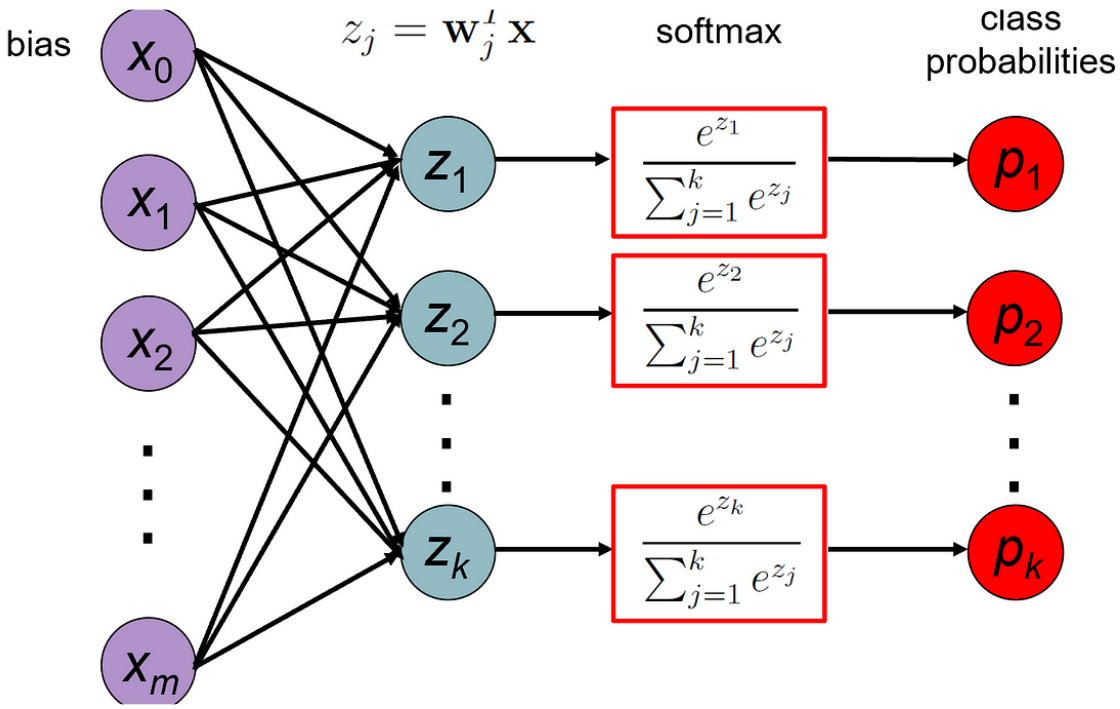



Shreya Rao in Towards Data Science

## Back to Basics, Part Tres: Logistic Regression

An illustrated guide to everything you need to know about Logistic Regression

Mar 2, 2023 638 10




Dr. Roi Yehoshua in Towards Data Science

## Deep Dive into Softmax Regression

Understand the math behind softmax regression and how to use it to solve an image classification task

May 25, 2023 300 2