Reading 6

☆ Article : Logistic Regression

Logistic Regression is a supervised learning algorithm that can be used to classify data into categories, or classes, by predicting the probability that an observation falls into a particular class based on its features.

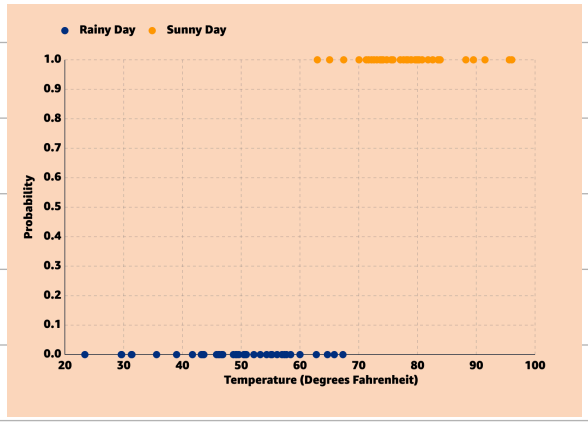
- Logistic Regression is usually used for binary classification.

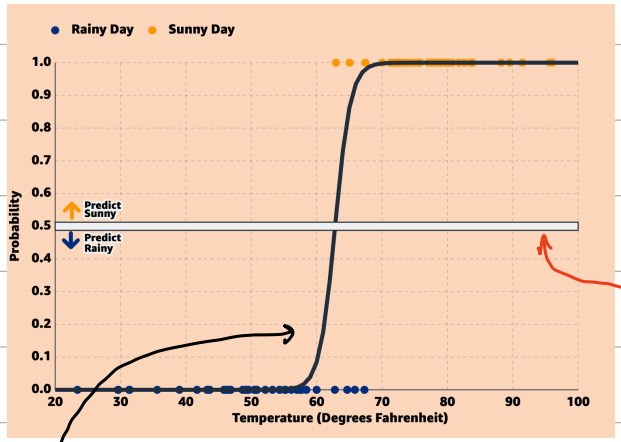
- Typical Setup:
  - Outcome: y (typically 0 or 1)
  - Equation used to estimate the probability that y belongs to a particular category given inputs $X = (x_1, x_2, \ldots, x_K)$:

$$P(y=1 \mid X) = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

where $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_K x_K$

- Ex





you can interpret the black line as the probability of a sunny day given a particular temperature.

This is the decision boundry. It can be changed. For example, you make it higher if you want to be more cautious about predicting rain.

- We use a loss function to determine how well our model fits the data.

- A suitable loss function for logistic regression is called the Log-Loss or binary cross-entropy. This function is:

$$\text{Log-Loss} = \sum_{i=0}^{n} -\left[y_i * \log(p_i) + (1-y_i) * \log(1-p_i)\right]$$

  - where n is the number of samples, indexed by i
  - y_i is the true class for the index i
  - p_i is the model prediction for the index i

- 2 main approaches to finding the coefficients $(\beta_0, \beta_1, \ldots \beta_K)$ that minimize the loss function.

1. Gradient Descent
   - Covered below
2. Maximum Likelihood Estimation (MLE)

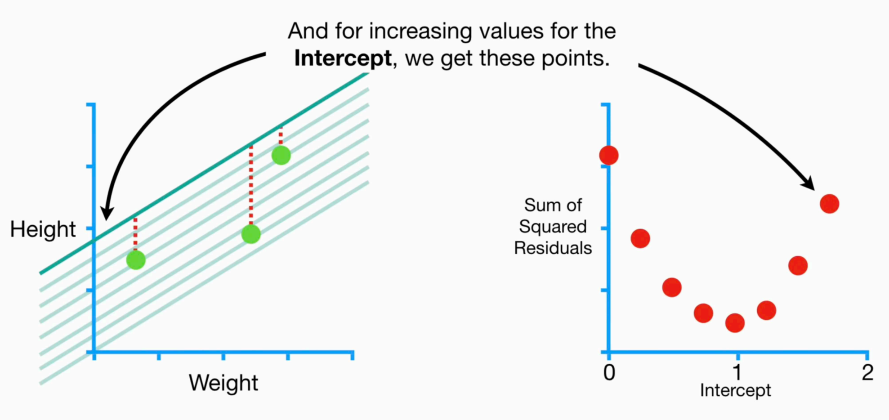
  - Minimizing log-loss function is equivalent to maximizing log-likelihood.
  - Therefore the goal is to find the parameter values that maximize the following:

$$\text{Log-Likelihood} = \sum_{i=0}^{n} \left[y_i * \log(p_i) + (1-y_i) * \log(1-p_i)\right]$$

☆ Video: Gradient Descent

- Gradient Descent can be used to optimize various data science problems.

- For finding an optimal y-intercept for a line, you pick a random y-intercept, calculate its SSE (loss function) and plot it.
  - ↳ You can keep repeating this for different values of the y-intercept:



↳ Gradient descent only does a few calculations when its far away from optimal y-intercept and increases the number of calculations it does as it gets closer.

↳ In other words it starts out by taking big steps and then takes smaller and smaller steps as it approaches the minimum.

↳ Gradient descent would take the derivitive with respect to the intercept in each term of the SSE: each term looks like:

$$(y - \hat{y})^2 = [y - (\text{intercept} + mx)]^2$$

↳ Gradient descent uses these derivatives to find the min, very useful for when you can't solve for when the derivative equals 0.

↳ Gradient descent determines the step-size by multiplying the slope at the current point by a small number called the learning rate.

↳ So in our example the new intercept is the (old intercept) - (step size)

↳ Once again the step size will decrease as the slope decreases

↳ Gradient descent stops when the step-size is very close to 0.

↳ In practice the indication to stop is when step-size = 0.001 or smaller.

↳ Gradient descent also has a max number of steps before giving up.

↳ In practice this threshold is 1,000 steps or greater.

↳ How would you estimate the best intercept AND the slope.

↳ Now your loss function takes a 3D shape, and you need to find the minimum of that.

↳ Remember each term in SSE (Loss function) looks like:

$$(y - \hat{y})^2 = [y - (\text{intercept} + mx)]^2$$

↳ So now you need to deferentiate with respect to intercept AND slope.

↳ Notice this time you computed 2 or more derivatives of the same function:

↳ hence you actually computed a gradient.

↳ hence you used gradient descent.

↳ With multiple variables the condition to stop is that:

1. All the step sizes are at or below the minimum threshold to stop.

OR

2. The the maximum number of steps has been reached.

Gradient Descent Basic Steps:

Step 1: Take the derivative of the Loss Function for each parameter in it.

↳ "Take the gradient of the loss function"

Step 2: Pick random values for the parameters

Step 3: Plug the parameter values into the gradient

Step 4: Calculate the Step Size:

$$\text{Step Size} = \text{Slope} \times \text{Learning Rate}$$

Step 5: Calculate the New Parameters:

$$\text{New Parameter} = \text{Old Parameter} - \text{Step Size}$$

Step 6: Go back to Step 3 and repeat until:

1. Step size(s) are very low
2. The maximum number of steps is reached

- **Note**: This can be computationally expensive for large datasets.

  ↳ To address this Stochastic Gradient Descent uses a randomely selected subset of the data at every step rather than the full dataset.