Regression Methods

Linear Regression and its Extensions

An AI Commons initiative

Lecture overview

1. Modular Approach to Machine Learning
   - **Model:** defining the input/output relationship
   - **Loss:** how to measure how good a prediction is
   - **Regularization:** [covered next week :)]
   - **Gradient Descent:** how a model learns
2. Expansions of Linear Regression
   - Multivariate Linear Regression
   - Polynomial Regression
   - Logistic Regression

Models

defining the input/output relationship

What is a model?

quick definition: a model is a function that maps input features to a target value. In this lesson, we define a model based on its weights / parameters that scale different input dimensions.

$$f(\mathbf{x}) = y$$
$$y \approx t$$

- **t** is our measured (real-world) target value
- **y** is our prediction (which we want to approximate t)
- bold **x** means **x** is a vector, so we allow for multiple input features!

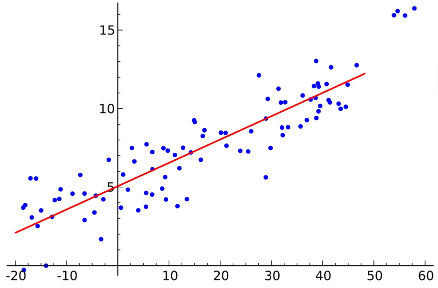
Models in machine learning

In machine learning, **we use training data (data we have) to 'teach'** a model to make predictions on **new data.**

Training is also called **fitting a model** to our training data.

Parameters (or weights) are the values that are learned during training.

Ex. Simple Linear Regression



$$y = mx + b$$

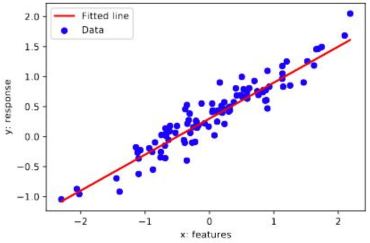
What is linear regression?

Linear regression is a type of model

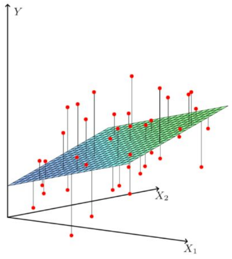
- in the single input variable case, **it fits a line**

What does linear mean?

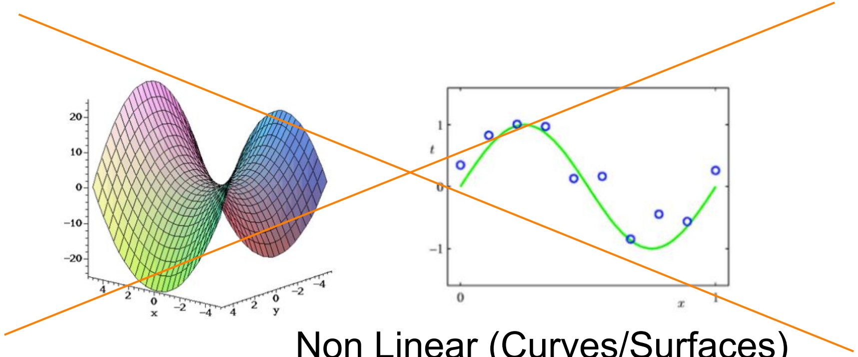
the relationship between input features and output target (t) is **a line or a plane** (not curved)



Linear regression



multivariate regression (here 2 variables)
$y=a_1x_1+a_2x_2$



Non Linear (Curves/Surfaces)

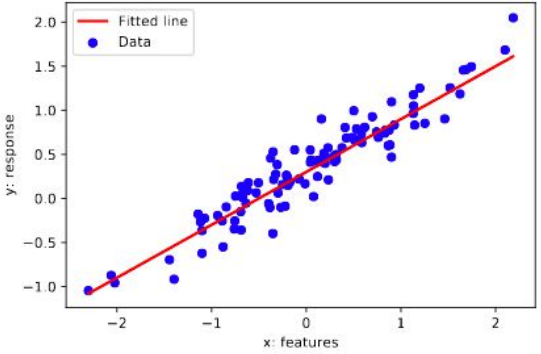
What is linear regression?

Linear regression is a type of model

- in the single input variable case, **it fits a line**

How do we define a line?

$$y = mx + b$$

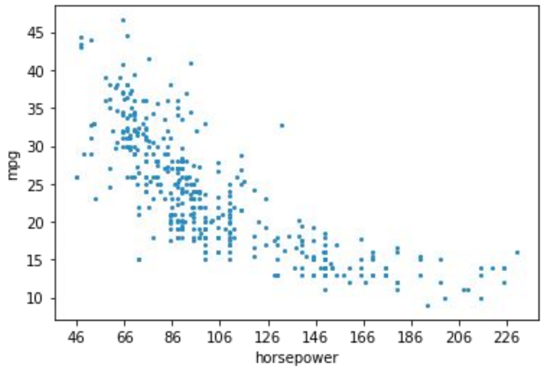


m and b are the parameters of our model

Example

We want have a dataset that contains information about cars.

- focus on two columns
  - Horsepower (engine power)
  - MPG (fuel efficiency)

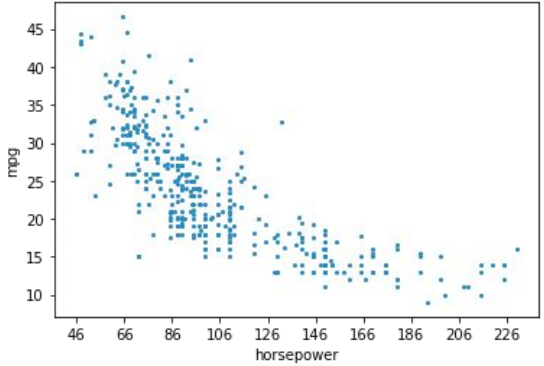


	horsepower	mpg
0	130	18.0
1	165	15.0
2	150	18.0
3	150	16.0
4	140	17.0
5	198	15.0
6	220	14.0
7	215	14.0
8	225	14.0
9	190	15.0

Example

We want to train a linear model that takes horsepower as input and outputs a fuel efficiency prediction.

- Assume the relationship between fuel efficiency and horsepower is linear → a line!



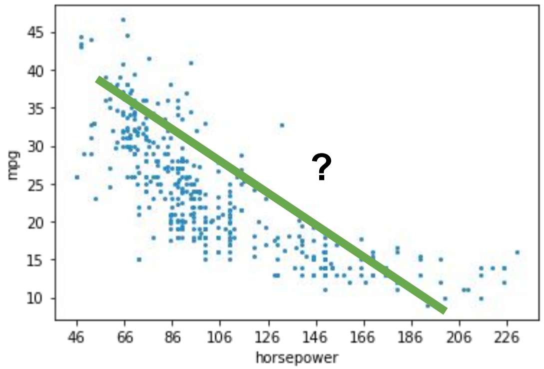
	horsepower	mpg
0	130	18.0
1	165	15.0
2	150	18.0
3	150	16.0
4	140	17.0
5	198	15.0
6	220	14.0
7	215	14.0
8	225	14.0
9	190	15.0

Example

We want to train a linear model that takes horsepower as input and outputs a fuel efficiency prediction.

- Our goal is to find the **m and b** of the 'line of best fit'

$$y = mx + b$$

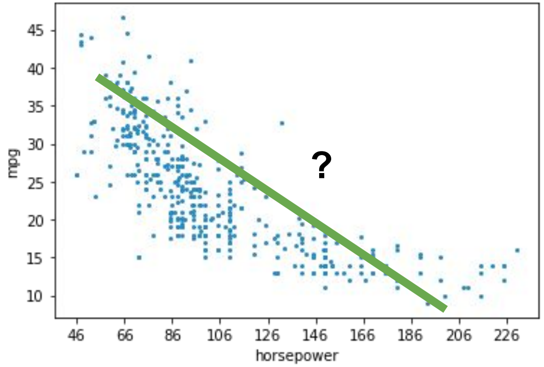


Example

We want to train a linear model that takes horsepower as input and outputs a fuel efficiency prediction.

- Our goal is to find the **m and b** of the 'line of best fit'

$$y = mx + b$$



How do we know what the best fit is?

→ we need a method for comparing lines

Loss Functions

how to measure 'wrongness'

Loss Functions

quick definition: A loss function is any function that takes as input, a labeled example from our dataset (x and t) and our model's prediction (y) and returns a value of how wrong our model's prediction is for the given example → smaller loss values implies more correct.

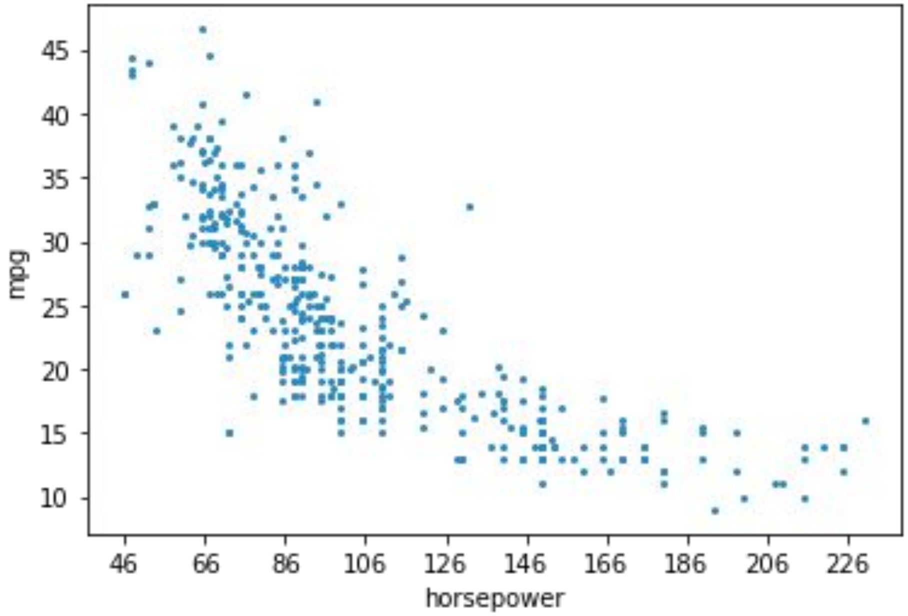
Loss Functions

quick definition: A loss function is any function that takes as input, a labeled example from our dataset (x and t) and our model's prediction (y) and returns a value of how wrong our model's prediction is for the given example → smaller loss values implies more correct.

There is no one (1) 'best/correct' loss function! Just like in model selection, you choose a loss based on the problem at hand.

Loss Functions

What makes sense for our car problem?



Loss Functions

Learn ai

What makes sense for our car problem?

Squared Error Loss

$$(y_i - t_i)^2$$

- very common loss choice for regression
- y - t is called the **residual**
- since its **squared**
    - penalizes over predictions as well as under predictions
    - penalizes more incorrect predictions more severely

AI

Loss Functions

A cost function is the average loss over our entire training set.

Squared Error Loss

$$(y_i - t_i)^2$$

Mean Squared Error Cost

$$MSE = \frac{1}{n}\sum_{i=1}^{N}(y_i - t_i)^2$$

We compare lines by comparing their cost.

- There are an infinite number of lines.
  (both m and b can be any real number)
- **How do we find the best?**

Gradient Descent

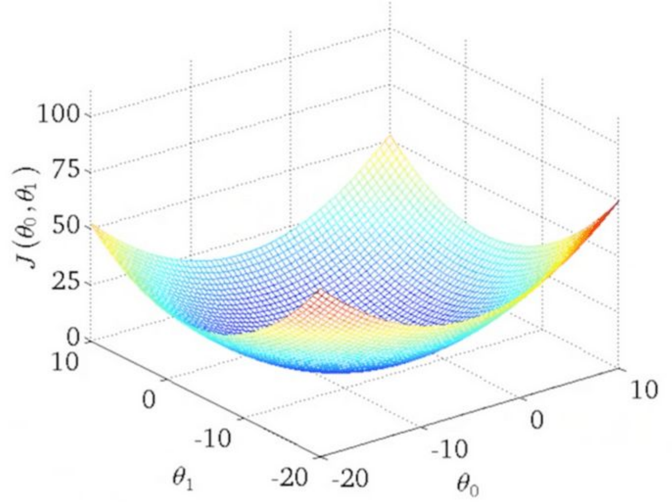
how a model learns the best parameters

Gradient descent

How do we know that we've found the best line?

Gradient descent

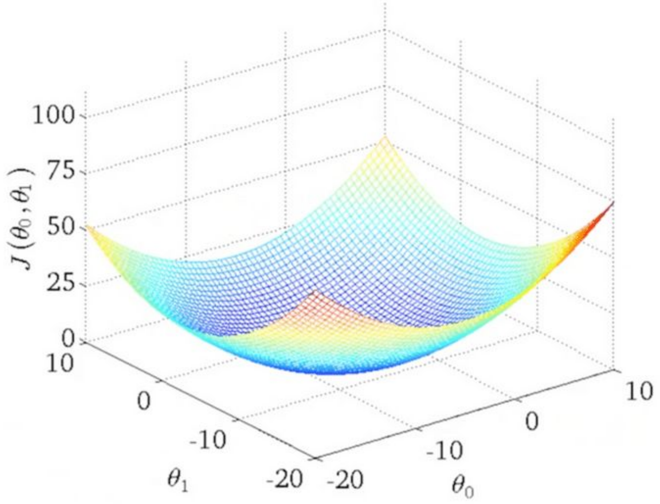
How do we minimize cost?



Gradient descent

Thought experiment:

1. You are on some hill, you cannot see anything around you. And you want to go downhill to find a nice valley.
2. You can feel the slope of the ground underneath your feet, so you know which way leads you down.
3. How do you find a nice valley.
   → **just take steps downward!**



Gradient descent

quick definition: Gradient descent is an iterative method for finding an optimum set of parameters relative to a model, loss function, and training dataset. Gradient descent uses the gradient of given cost function (average of loss) to minimize it over the training set.

$$m_{i+1} = m_i - \alpha \nabla_m J(m_i, b_i)$$
$$b_{i+1} = b_i - \alpha \nabla_b J(m_i, b_i)$$

Gradient descent

Learn ai

math definition: The gradient of a function is the vector of partial derivatives of its inputs.

Translation: the gradient of a function at a point is the **direction of steepest <u>ascent</u>** on the function at that point.

The gradient is the feeling underneath your feet which tells you the slope of the hill you are standing on.

- 1 * gradient is the **direction of steepest <u>descent</u>**

AI

Gradient descent

The gradient is the feeling underneath your feet which tells you the slope of the hill you are standing on.

- \- 1 * gradient is the **direction of steepest <u>descent</u>**

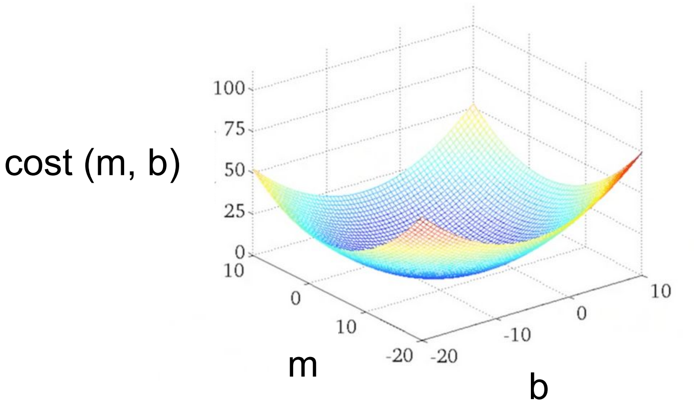
Car example: ***How do we update our parameters (m and b) to decrease the cost?***

$$m_{i+1} = m_i - \alpha \nabla_m J(m_i, b_i)$$
$$b_{i+1} = b_i - \alpha \nabla_b J(m_i, b_i)$$

Example

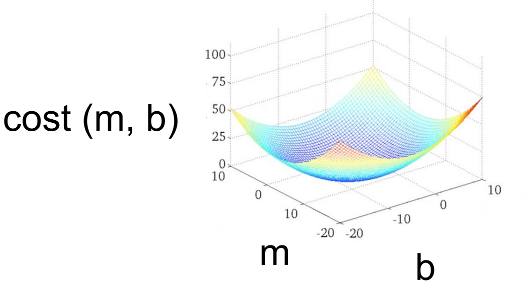
In our car example the cost surface we are descending is a surface in 3D, since we have two weights (m and b). A point on this surface looks like: (m, b, cost(m, b)).

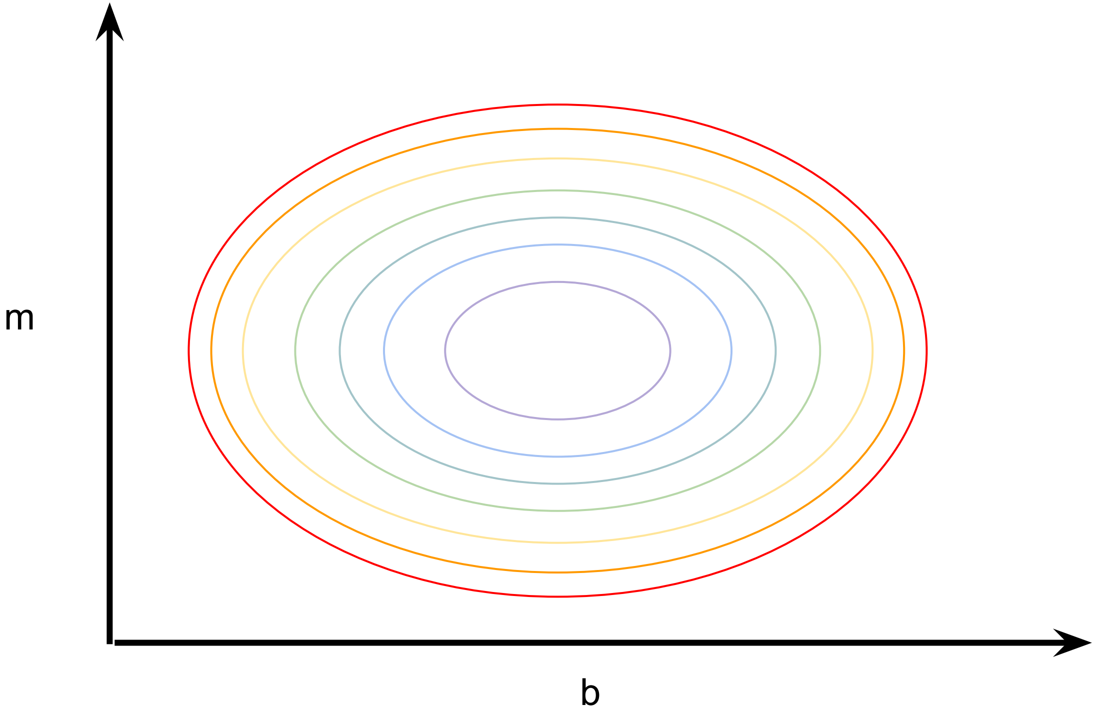
In other words, **each point on this surface represents a line in our data space with its respective cost over our training dataset.**



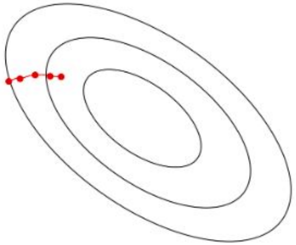
Gradient descent

$$m_{i+1} = m_i - \alpha \nabla_m J(m_i, b_i)$$
$$b_{i+1} = b_i - \alpha \nabla_b J(m_i, b_i)$$

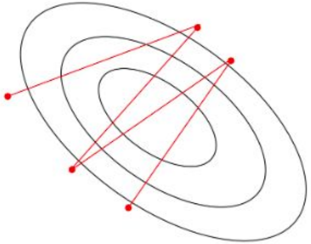




Learning rate



α too small:
slow progress



α too large:
oscillations

Learning rate (alpha) can be thought of as the size of the step each iteration of gradient descent takes.

$$m_{i+1} = m_i - \alpha \nabla_m J(m_i, b_i)$$
$$b_{i+1} = b_i - \alpha \nabla_b J(m_i, b_i)$$

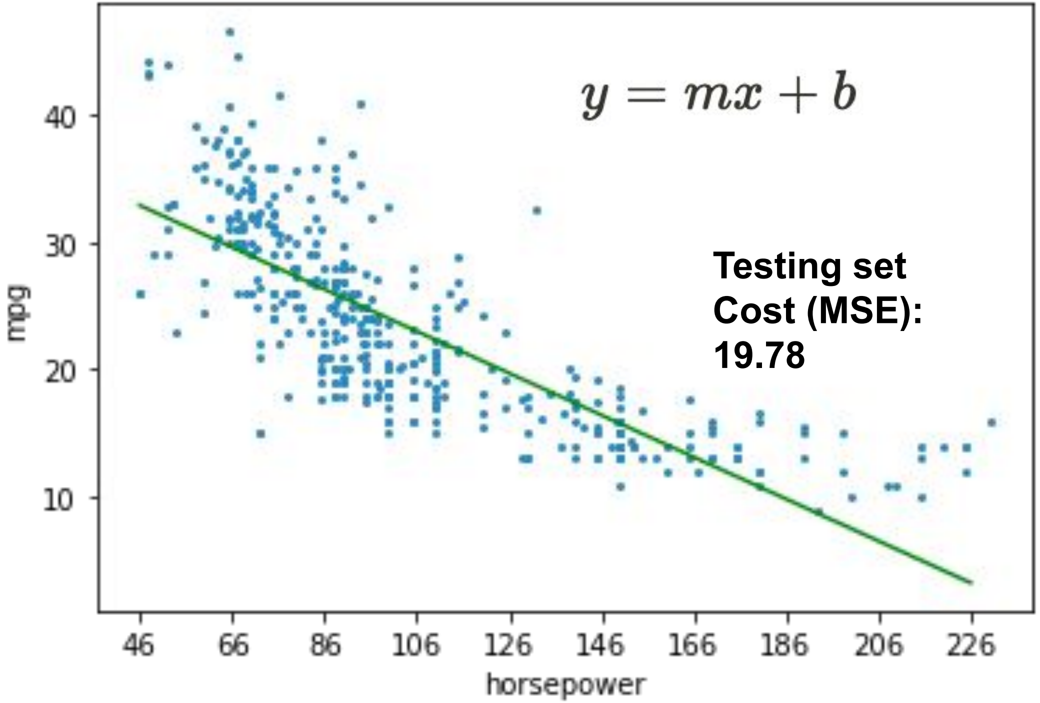
Example

In sklearn, we can fit a linear regression model quite easily!

```
mpg_df['horsepower^2'] = mpg_df['horsepower'] ** 2


X = mpg_df['horsepower'].to_numpy()
y = mpg_df['mpg'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

regression_model = LinearRegression()
regression_model.fit(X_train.reshape(-1,1), y_train)

print('Mean squared error: %.2f'
      % mean_squared_error(regression_model.predict(X_test.reshape(-1,1)), y_test))
```

Example

In sklearn, we can fit a linear regression model quite easily!



Expanding on Linear Regression

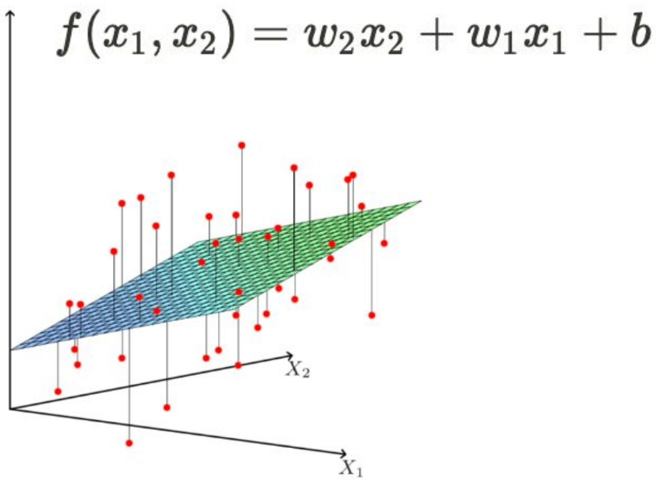
Beyond the line of best fit

1. Multiple linear Regression
2. Polynomial Regression
3. Logistic Regression

Multiple linear regression

We can use more than one input feature!

Instead of a line, we fit a plane.



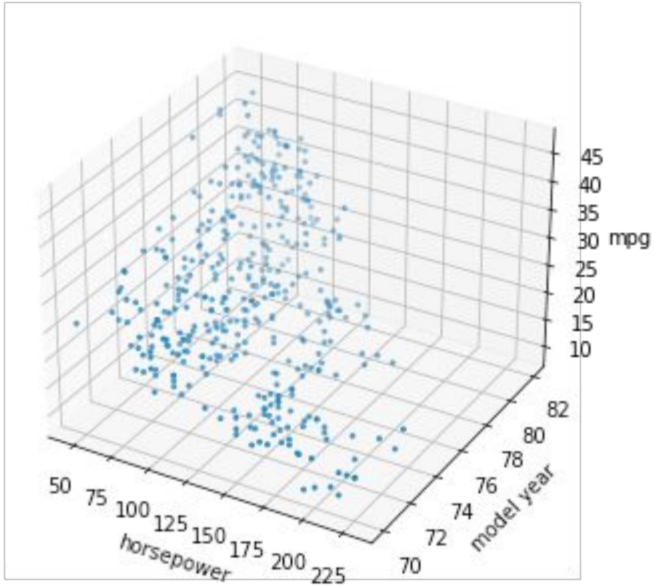
Only change is that we have additional parameters to learn.

For D input features:

$$f(\mathbf{x}) = \sum_{d=1}^{D} w_d x_d + b$$

Example

For our car example, let's include a new input column called 'model year' (the year of the car's release).



horsepower	model year	mpg
92.0	76	25.0
88.0	73	18.0
68.0	81	34.1
112.0	73	19.0
170.0	75	16.0
90.0	79	28.4
85.0	81	17.6
85.0	70	21.0
80.0	81	28.1
190.0	70	15.0

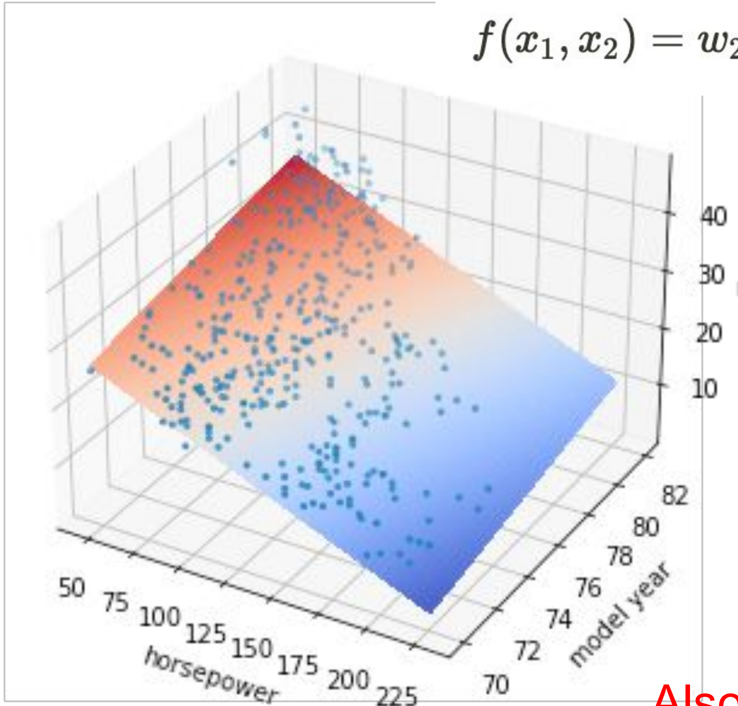
Example

To fit a plane to our points:

```
X = mpg_df[['horsepower', 'model year']]
y = mpg_df['mpg']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

regression_model = LinearRegression()
regression_model.fit(X_train, y_train)

print('Mean squared error: %.2f'
      % mean_squared_error(regression_model.predict(X_test), y_test))
```

Example

$$f(x_1, x_2) = w_2 x_2 + w_1 x_1 + b$$



Testing set Cost (MSE): 15.15

Notice, our plane obtains a better cost than our line (19.78).

What are some conclusions to draw from this?

Also note: no matter how many input features we add, our prediction remains uncurvy!

Polynomial regression

We can fit a curve!

- not every relationship in the real-world is linear, **in fact few are!**

With M being the max degree of our polynomial:

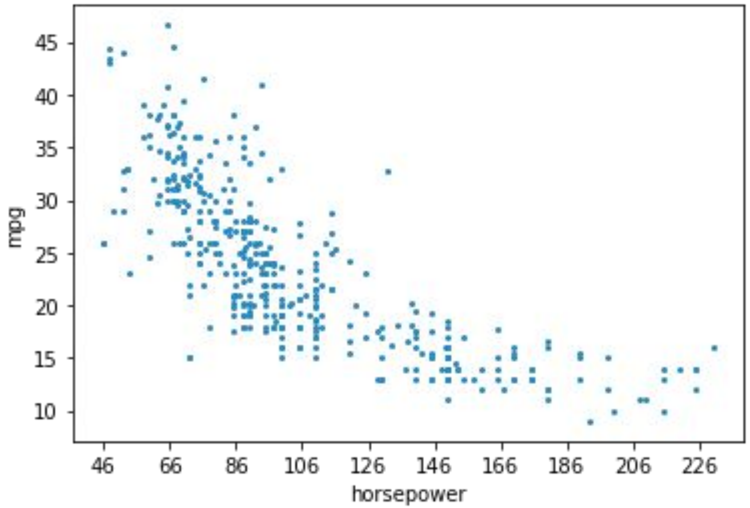
$$f(x) = \sum_{p=0}^{M} w_p x^p$$

We fit a curve the exact same way we fit a line.

Example

Look back at our 1D car example once more:

- not every relationship in the real-world is linear, **in fact few are!**



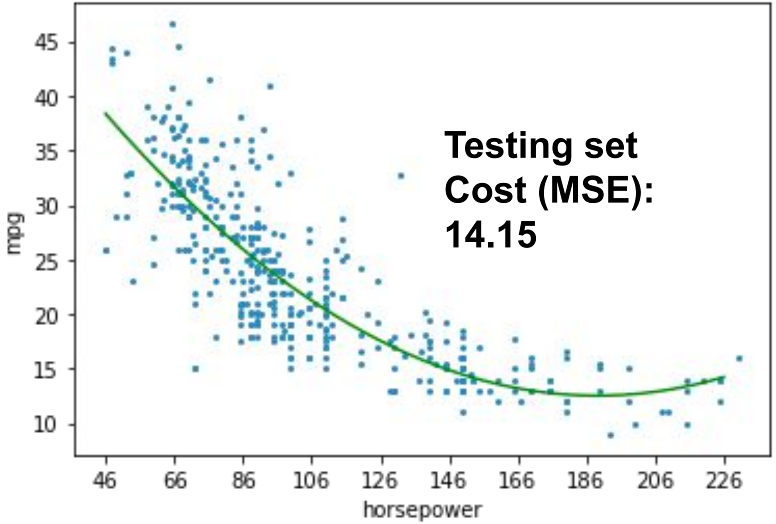
Let's try to fit a quadratic.

$$f(x) = w_2 x^2 + w_1 x + b$$

Example

Look back at our 1D car example once more:

- not every relationship in the real-world is linear, **in fact few are!**



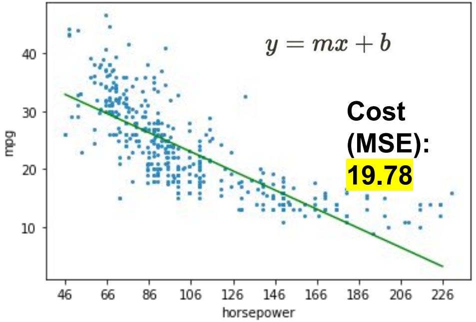
Let's try to fit a quadratic.

$$f(x) = w_2 x^2 + w_1 x + b$$

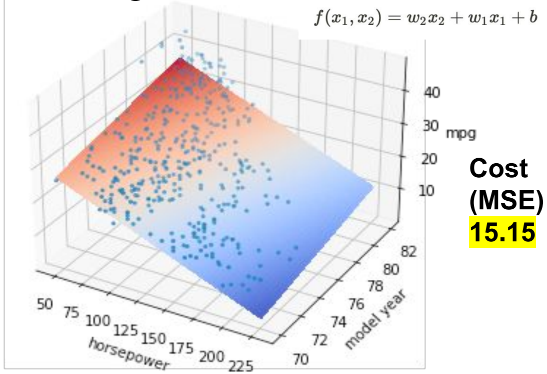
w2 = 0.0013
w1 = -0.48
b = 58

What's the lesson here?

Simple Linear Regression

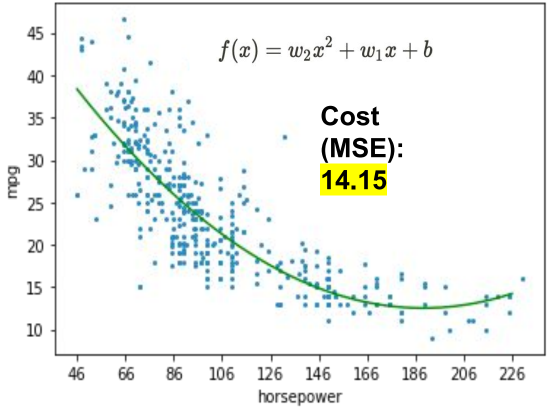


Multivariate Linear Regression



Cost (MSE): 15.15

Polynomial Regression



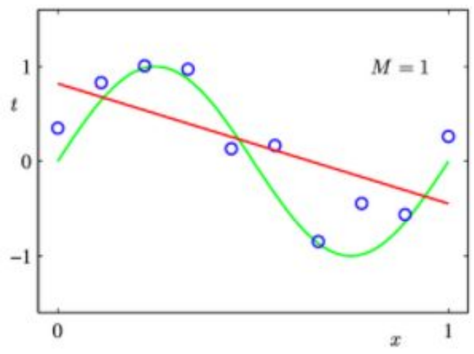
Two strategies for improving model performance:

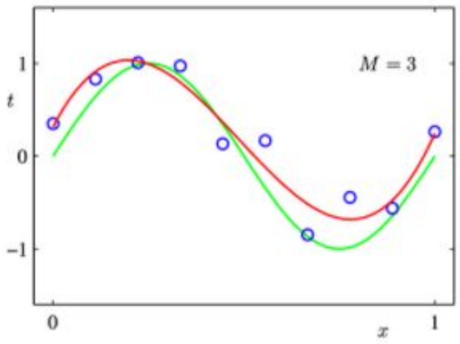
1. Give your model **more data** (ex: multivariate)
2. Give your model **more expressivity** (ex: polynomial)

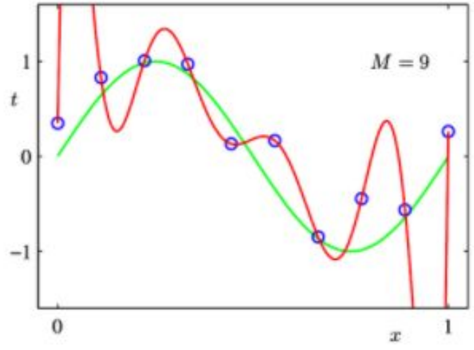
Expressivity

The expressivity of a model is the range of functions that it can approximate.

$$f(x) = \sum_{p=0}^{M} w_p x^p$$

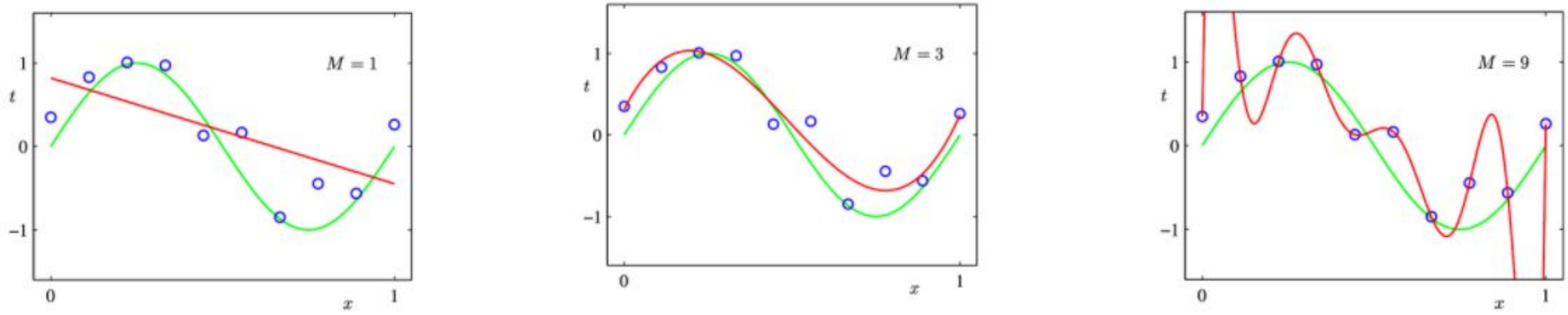






Expressivity

$$f(x) = \sum_{p=0}^{M} w_p x^p$$



[more on over/underfitting and how to address these issues next week :)]

Logistic Regression

Predicting Discrete Targets

Continuous vs. discrete target

- Statistical Models vary in the **datatype** of their **predictions**
- **Regression** is where the prediction **is a real number (continuous variable)**

  Ex. Height in cm, Weight in lbs, Price in $
- **Classification** is where the prediction **is for a class (discrete values)**

  Ex. Dog or cat, dead or alive, gender

Animal Weight Model

Regression

Prediction:

5 kgs

Animal Type Model

Classification

Prediction:

Cat

How can we predict discrete values?

Let’s say we are trying to predict whether a given beverage is **coffee or tea,** given its **caffeine content.**

What is/are our input feature(s)?

What is our target? What values can it be?

Does linear regression work?

Let's use linear regression to fit a line to the data. We will then choose a **threshold value** on that line above which examples are classified as **positive (coffee)** and below which examples are classified as **negative (tea)**.

Does linear regression work?

Let's use linear regression to fit a line to the data. We will then choose a **threshold value** on that line above which examples are classified as **positive (coffee)** and below which examples are classified as **negative (tea)**.

PROBLEM: highly confident correct predictions are punished!

SOLUTION: *squash* our output to exist only between 0 and 1.

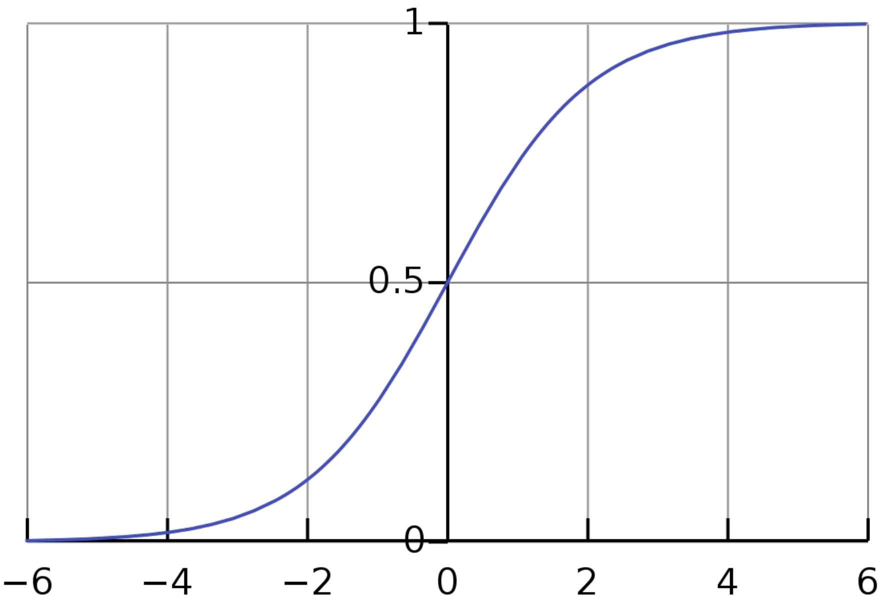
The logistic (sigmoid) function

Learn ai

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

maps real values onto the interval (0,1)

- Limit as t $\rightarrow$ +∞ = 1
- Limit as t $\rightarrow$ - ∞ = 0



AI

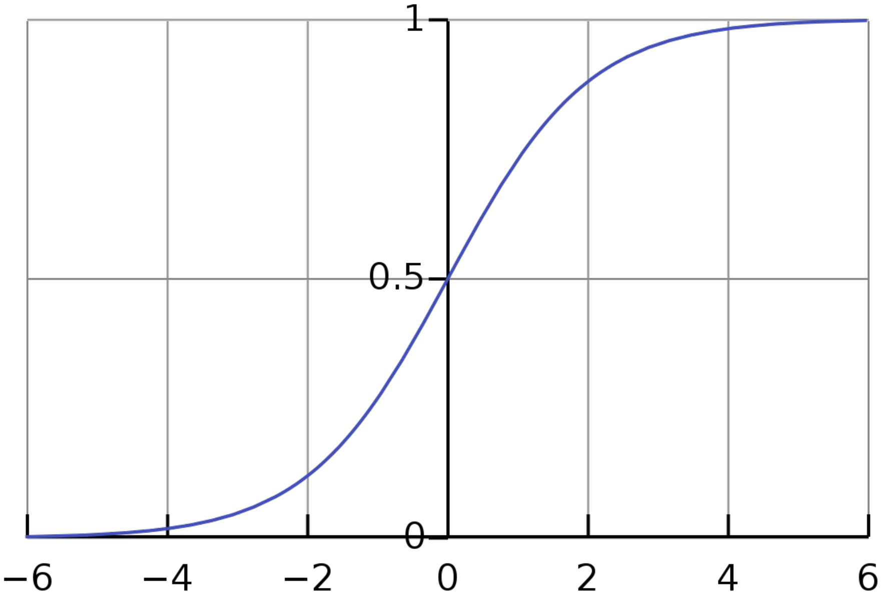
Logistic regression

In linear regression we assume our data is best fit by **a line**.

Logistic regression is a different type of model
→ we assume our data is best fit by a **sigmoid curve**.

$$f(x) = \sigma(mx + b)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

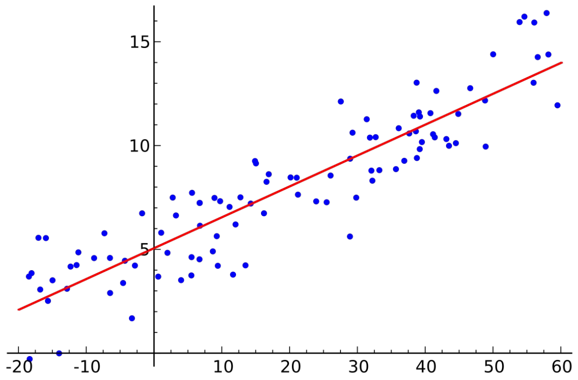


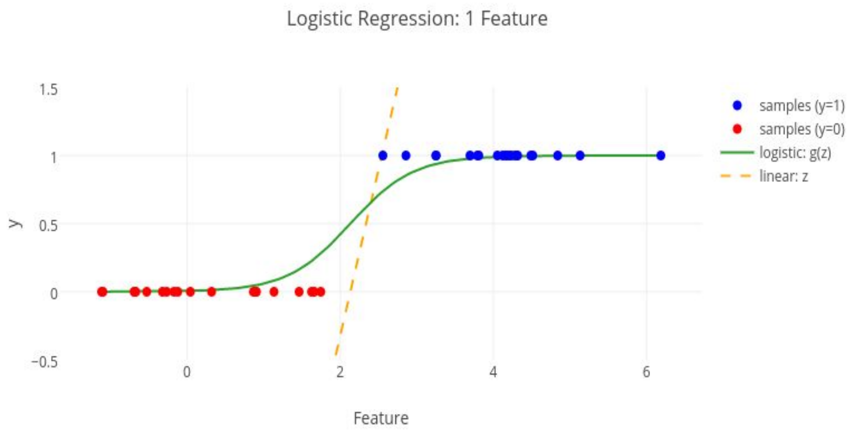
Logistic regression is for classification

Learn ai

With **linear** regression,
we try to fit a **line** to some data.

For **logistic** regression,
we fit a **logistic** function to some data





AI

Cross-entropy (log) loss

With **linear** regression, we used **squared error** loss.

$$SE = (y_i - t_i)^2$$

With **logistic** regression, we used **cross-entropy (log)** loss.

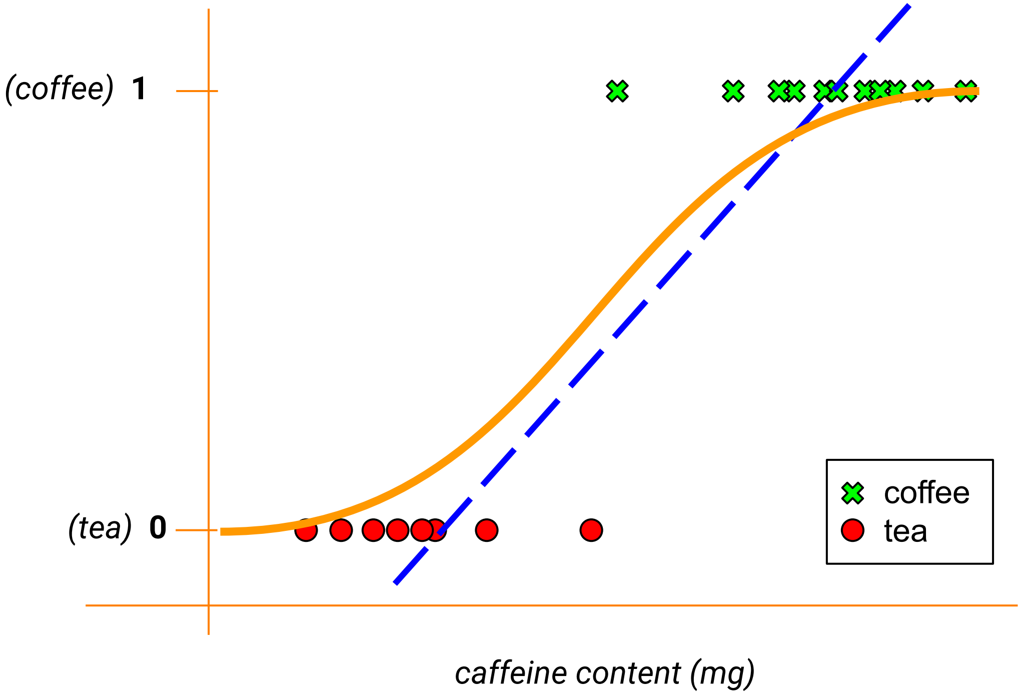
$$CE = -t \log y - (1 - t) \log(1 - y)$$

Why?

Binary classification example

Learn ai

- We want to train a model to classify a beverage as either **coffee** or **tea,** based solely on caffeine content.
- We randomly sample 20 beverages, measure their caffeine content, and label each as **coffee (1)** or **tea (0)**.
- We tried to fit a line to this data. This caused issues.
- Now we fit a logistic curve (logistic regression) to this data. This works better.

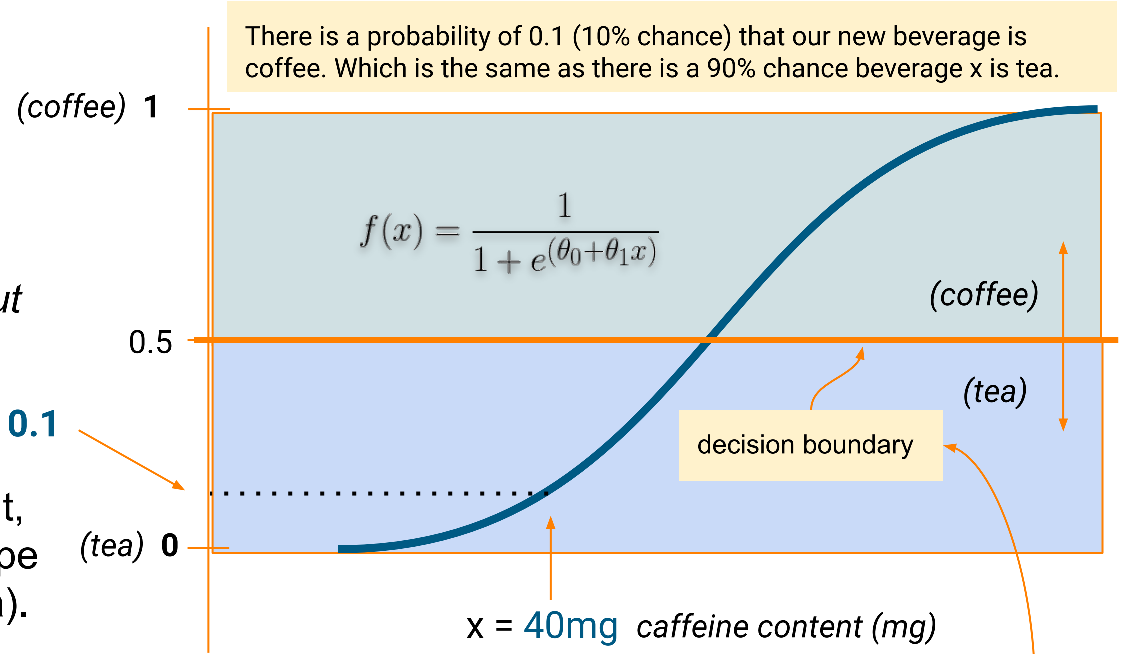


Which looks like it fits the data more? Why?

AI

Binary classification interpretation

- We have a trained logistic model on our beverage data, *f(x)*.
- *How do we interpret the output of our model?*
- Example: we have a new caffeine content measurement, *x*, and want to predict what type of beverage it is (coffee or tea).



We have: $f : \mathbb{R} \rightarrow (0, 1)$ (continuous output)
We want: $f : \mathbb{R} \rightarrow \{0, 1\}$ (discrete output)

```
if f(x) >= 0.5 then drink=coffee
else (f(x) < 0.5)then not coffee(tea)
```

Multiple logistic regression

When we have multiple input features $x_1, \ldots, x_n$ our logistic regression function looks like:

$$f(\mathbf{x}) = \sigma(b + \sum_{i=1}^{N} w_i x_i)$$

(very similar to multiple linear regression)

References

https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec02.pdf

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

https://archive.ics.uci.edu/ml/datasets/Auto+MPG

https://upload.wikimedia.org/wikipedia/commons/thumb/8/88/Logistic-curve.svg/1200px-Logistic-curve.svg.png

Pattern Recognition and Machine Learning, Christopher Bishop

Contributors

Elias Williams

Addison Weatherhead

Isha Sharma

Kevin Zhu

Gilles Fayad