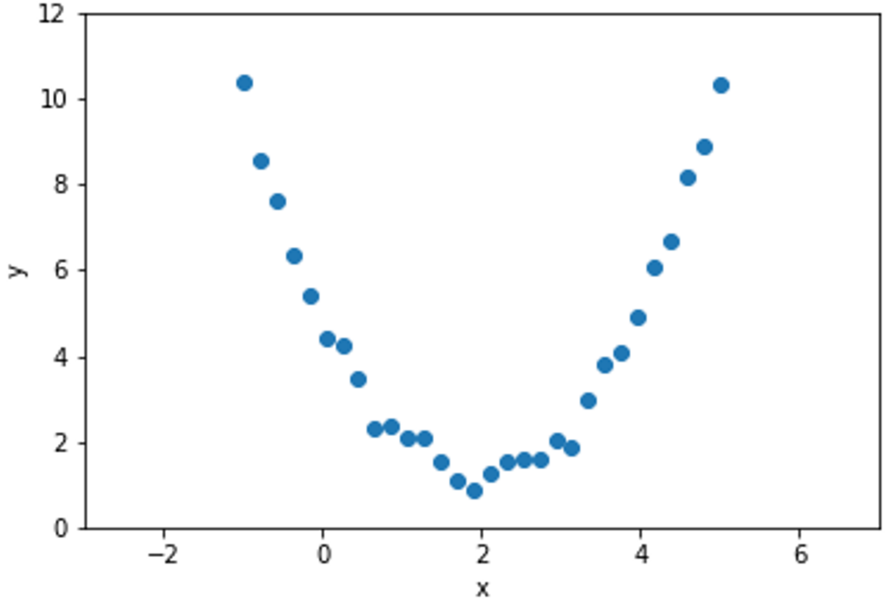
LECTURE 14

Feature Engineering

Transforming numerical features and encoding categorical data in order to build more sophisticated models

Motivating Feature Engineering

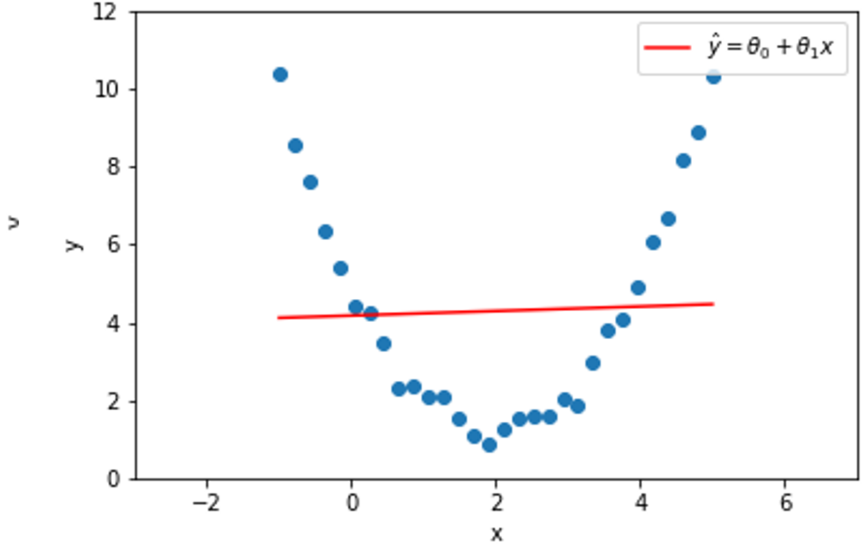
Can we fit a model to this data?



Can we fit a model to this data?

Simple Linear Regression?

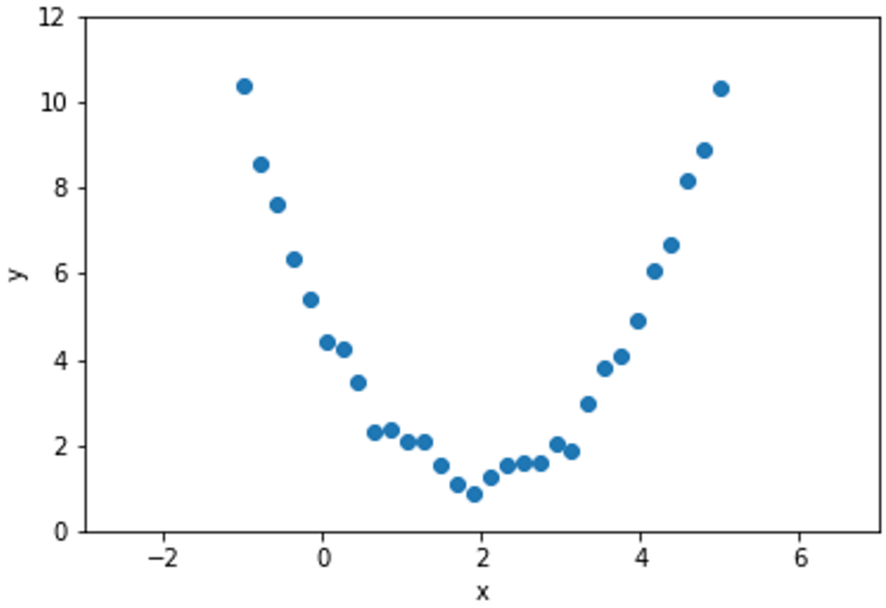
No, because the data is fundamentally nonlinear



Can we fit a model to this data?

Multiple Linear Regression?

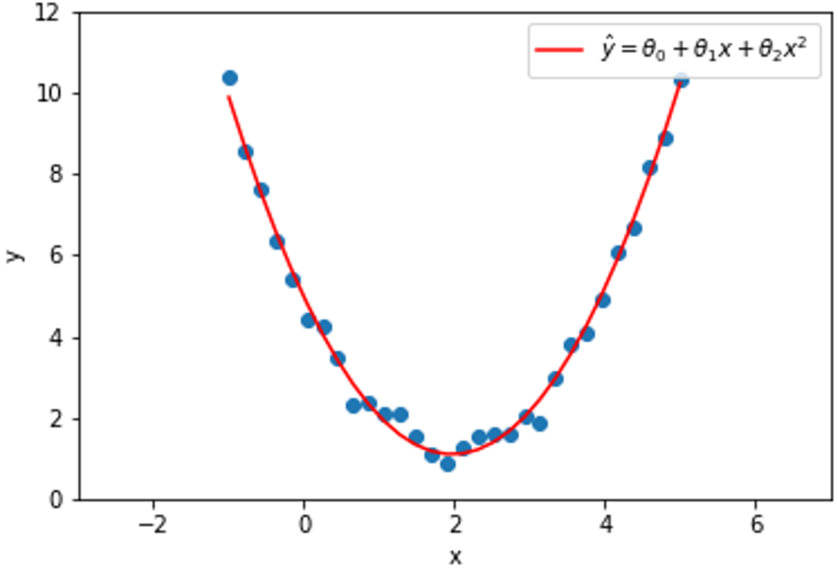
No, because there are no other features to add



Can we fit a model to this data?

Idea: *Create* an extra feature to use in the model. What feature should we add?

Since the data looks like a parabola, let's add a quadratic feature



What is a feature?

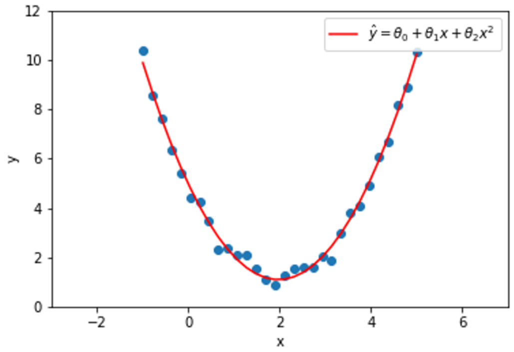
A feature is an input to our model

- So far, we have just used the raw data as features

We can also create new features to use an inputs to our model

- The process of creating new features is called **feature engineering**

Example: Quadratic model



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

Features: x, x^2

Parameters: θ_0, θ_1, θ_2

Can we really just create our own features?

Yes! (with some restrictions)

We can create any feature we want as long we can write the model in the form

$$\boxed{\hat{y} = x^T\theta}$$

This is a **linear combination of the features**. The features <u>*cannot*</u> depend on the parameters of the model!

Example: Adding a Quadratic Feature

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 = \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = x^T\theta$$

Agenda

We can choose/create **x** any way we like as long as our model follows the form $\boxed{\hat{y} = x^T\theta}$

The rest of the lecture will discuss different techniques we can use to create **x**:

- How can we create features from **quantitative data**?
- How can we create features from **categorical data**?
- How can we create features from **text data**?

Intro to Scikit-Learn

Feature Engineering: Quantitative Data

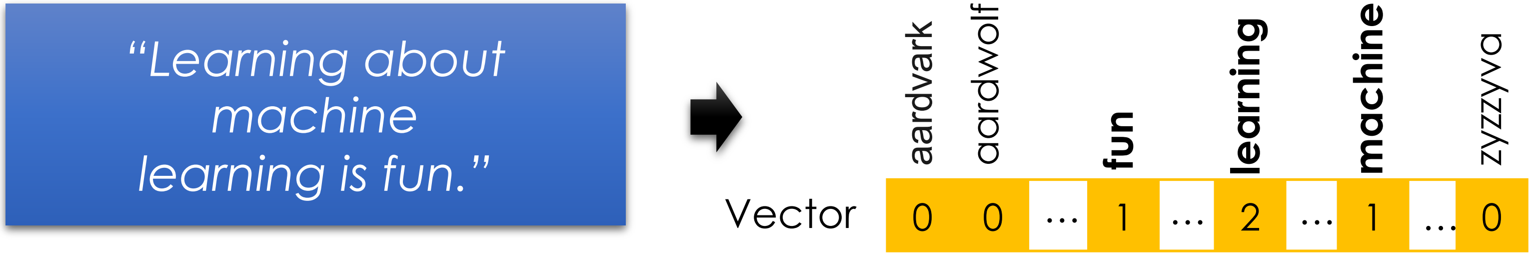
Feature Engineering: Categorical Data

Feature Engineering: Imputation

Feature Engineering: Text Data

Bag-of-words Encoding

- Generalization of one-hot-encoding for a string of text:



- Encode text as a long vector of word counts (Issues?)
  - Typically high dimensional (millions of columns) and very sparse
  - Word order information is lost… (is this an issue?)
  - What happens when you see a word not in the dictionary?

- A **bag** is another term for a multiset: *an unordered collection which may contain multiple instances of each element.*

- **Stop words:** words that do not contain significant information
  - Examples: the, in, at, or, on, a, an, and …
  - Typically removed

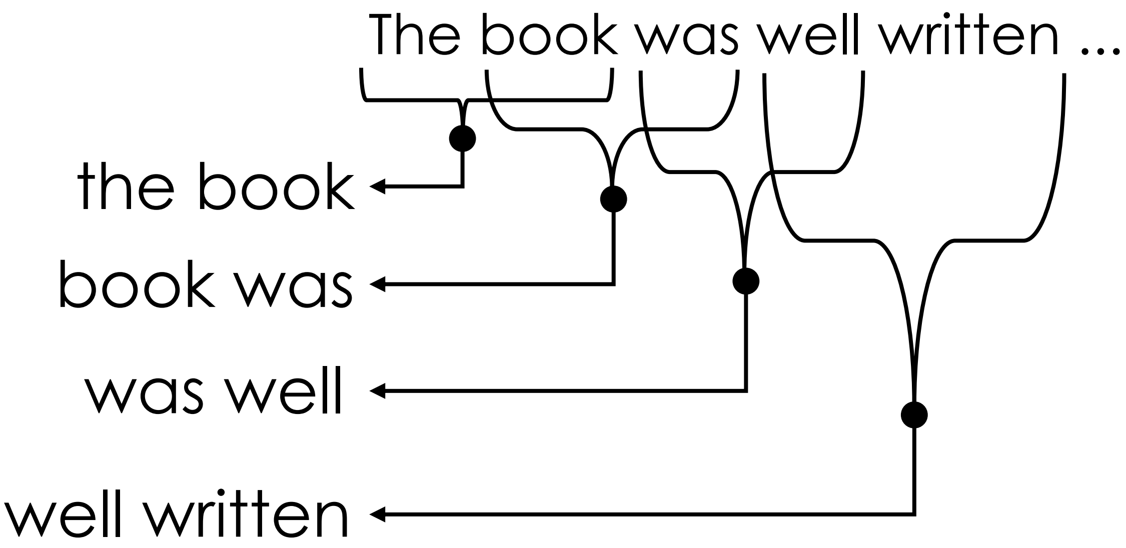
N-Gram Encoding

- Sometimes word order matters:

*The book was **<u>not</u>** well written but I did enjoy it.*

*The book was well written but I did **<u>not</u>** enjoy it.*

- How do we capture word order in a "vector" model?
  - N-Gram: *"Bag-of- sequences-of-words"*



2-Gram Encoding

	aardvark is	apple shines		**the book**		**book was**		**was well**		**well written**		zebra ran
Vector	0	0	...	1	...	1	...	1	...	1	...	0

N-Gram Encoding

- Sometimes word order matters:

*The book was **not** well written but I did enjoy it.* ➡ *The book was well written but I did **not** enjoy it.*

- How do we capture word order in a "vector" model?
  - N-Gram: *"Bag-of- sequences-of-words"*

- Issues:
  - Can be very sparse (many combinations occur only once)
  - Many combinations will only occur at prediction time

Feature Functions

Expressing Feature Engineering Mathematically

We encapsulate the feature engineering process as a function ϕ that transforms the raw data **x** into the features ultimately used for the model

We call ϕ a **feature function** because it maps data vectors to feature vectors

Example: Intercept term in Simple Linear Regression

$$\hat{y} = \theta_0 + \theta_1 x = \begin{bmatrix} 1 \\ x \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \phi(x)^T \theta$$

$$x \longrightarrow \phi(x) \longrightarrow \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Expressing Feature Engineering Mathematically

Feature functions can be arbitrarily complex (i.e. they can encompass multiple feature engineering operations)

Example: MPG dataset

```
def nonlinear_design_matrix(df):
    X = basic_design_matrix(df)
    # Compute nonlinear transformations for each quantitative feature
    for col in X.columns:
        X[f'{col}^2'] = X[f'{col}'] ** 2
        X[f'{col}^3'] = X[f'{col}'] ** 3
        X[f'log({col})'] = np.log(X[f'{col}'])
        X[f'sin({col})'] = np.sin(X[f'{col}'])
    return X
```

This feature function takes in a DataFrame and outputs another DataFrame with 5 times the number of features

Revisiting the Design Matrix

The feature function ϕ transforms *one* data vector into *one* feature vector

We have multiple data vectors $x_1, x_2, \ldots, x_n$ so we need to apply ϕ to each data vector

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \longrightarrow \phi \longrightarrow \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_n) \end{bmatrix} = \Phi$$

In the previous lecture, the **X** matrix was called the design matrix

After applying the feature functions, the design matrix is called $\boldsymbol{\Phi}$

Mathematical Implications of Feature Engineering

Revisiting the Normal Equations

Recall that fitting an OLS model requires solving the normal equations

$$(X^T X)\theta = X^T y$$

which can be updated using the new design matrix to

$$(\Phi^T \Phi)\theta = \Phi^T y$$

Since feature engineering changes the $\boldsymbol{\Phi}$ matrix, feature engineering can influence the solution to the normal equations.

Linearly dependent features

Recall the solution to the normal equations: $\theta^* = (\Phi^T\Phi)^{-1}\Phi^T y$

- This solution exists only when $(\Phi^T\Phi)^{-1}$ exists
- $(\Phi^T\Phi)^{-1}$ exists only when $\boldsymbol{\Phi}$ is full rank (i.e. all columns are linearly independent)
- If $\boldsymbol{\Phi}$ is not full rank then there are infinite solutions to the normal equations
  - This is bad because model parameters are unstable

A simple example of linearly dependent features is having height in both inches and cm

This issue of linear dependence is a problem for one of the feature engineering techniques we saw earlier in the lecture. Can you identify which one?

One-Hot Encoding and Linear Dependence

bias	...	co
1	...	Apple
1	...	Samsung
1	...	Apple

bias	...	co_appl	co_sam
1	...	1	0
1	...	0	1
1	...	1	0

Notice that `co_appl + co_sam = bias`! This means the columns are linearly dependent

- **Solution:** Drop one of the one-hot encoded columns per variable

Too Many Features

If you add too many features, the normal equations will have infinite solutions

The normal equations can be thought of as a system of equations with N equations and P unknown quantities to solve for

- N: # of data points
- P: # of parameters

If P > N, you have more unknowns than equations so there can be no unique solution

Additionally, too many features can cause overfitting, which will be covered in future lectures

Summary

Feature Engineering Summary

- Feature engineering is the process of creating new useful features from your data to build more sophisticated models
- Feature engineering allows you to utilize non-numerical data
  - One-hot encoding is a widely used technique
- Need to be careful in choosing how many and which features to create
  - Linearly dependent features
  - Too many features
- Feature engineering is as much an art as it is a skill
  - Neural networks try to automatically do feature engineering