Evaluation Metrics

For Classification and Regression

What will be covered today

- What are evaluation metrics?
- Why are there different ones?
- Bias and Variance
- Overfitting vs underfitting
- Classification
  - Accuracy
  - False positives and negatives
  - roc-auc
  - Precision
  - Recall
  - Confusion matrix
  - Fscore
- Regression
  - MAE
  - MSE
  - RMSE
  - RMSLE
  - R
  - R2- coefficient of determination
- Mini Project 3

Goal for today

- To develop a shared vocabulary that empower us to talk about the performance of our models
- To talk about that math of the major methods, when to use each one, and the code for each method

Why can machine learning models perform badly?

- What do you think?
- Write ideas down on whiteboard

There are two ways models can perform badly

- They can have too much
  - Bias
  - Variance
- Let’s talk about these.

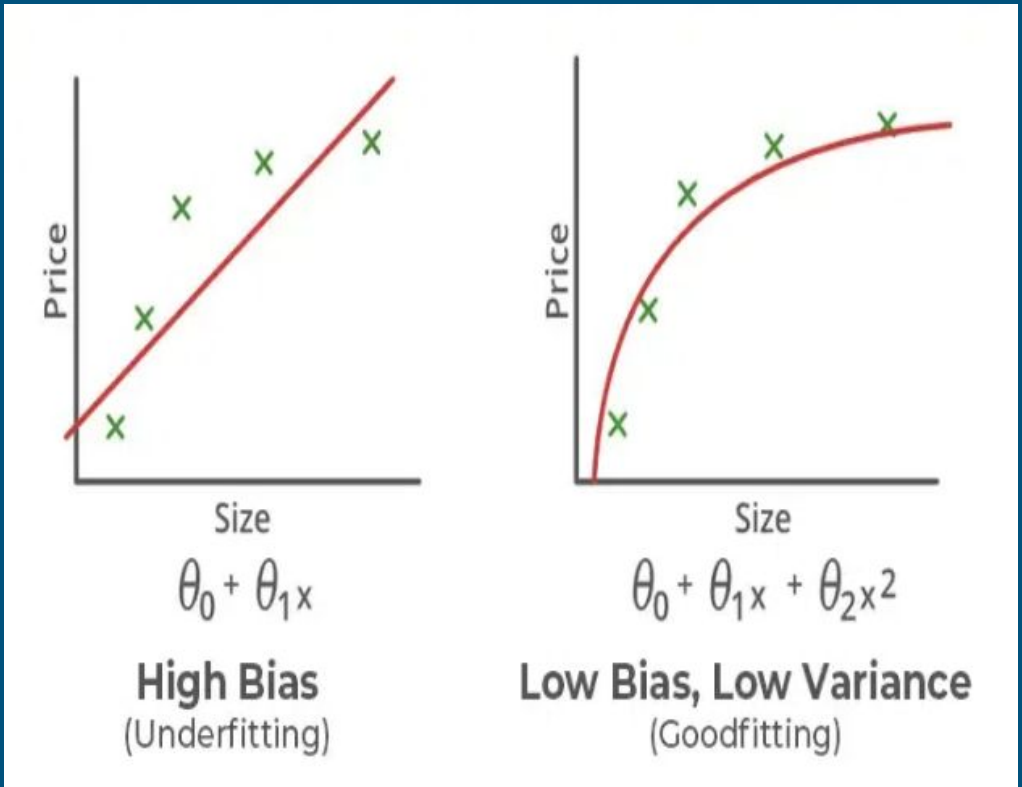
What does it mean for a model to have bias?

- What do you think?
- Write down thoughts on whiteboard

Bias

- Having high bias means the model has assumptions that do not match the data
  - Ex: if you assume it is linear, but the function you're predicting is nonlinear
  - Ex: if you manually remove columns from the training set, including them may actually improve performance
  - Ex: Dataset only includes partial data
    - Only male patients for medical study

Visual Example



How do you know a model is biased?

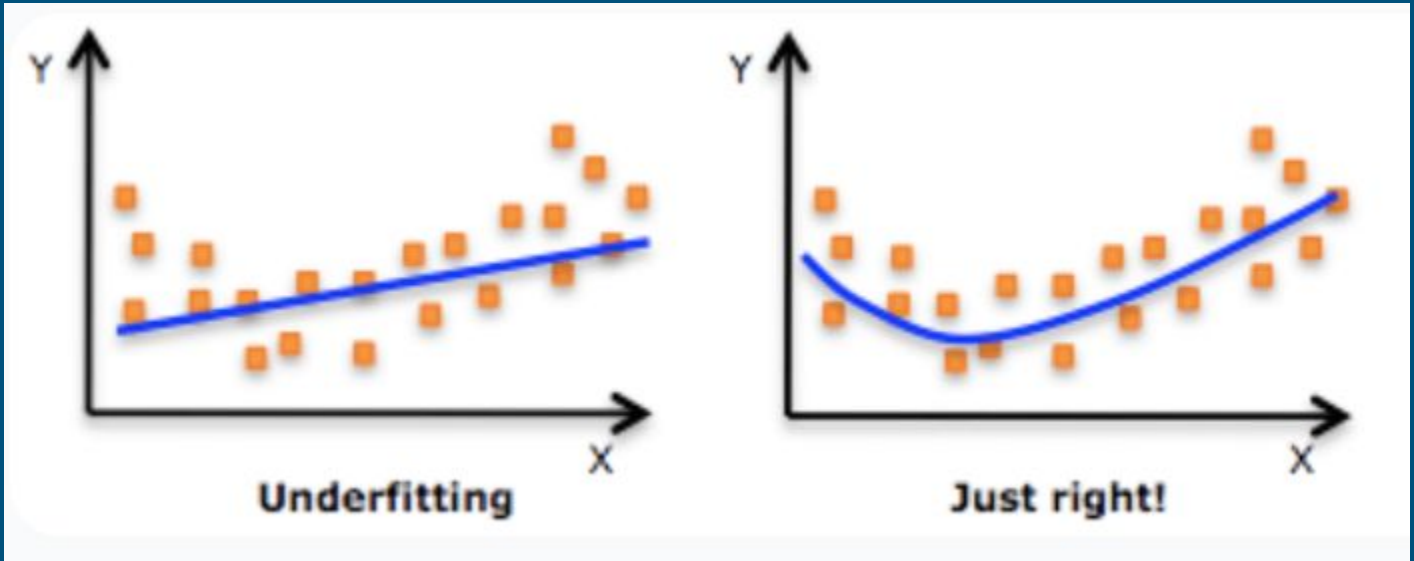
- Any thoughts on how we can mathematically evaluate how biased a model is?
- In other words, how can we tell if a model does not properly understand the pattern in the data?

How do you know a model is biased?

- By seeing how accurate the model is at making predictions from the **training set**
- If the model performs badly, that means it has not understood the pattern that is in the data OR the model is not capable of understanding the pattern

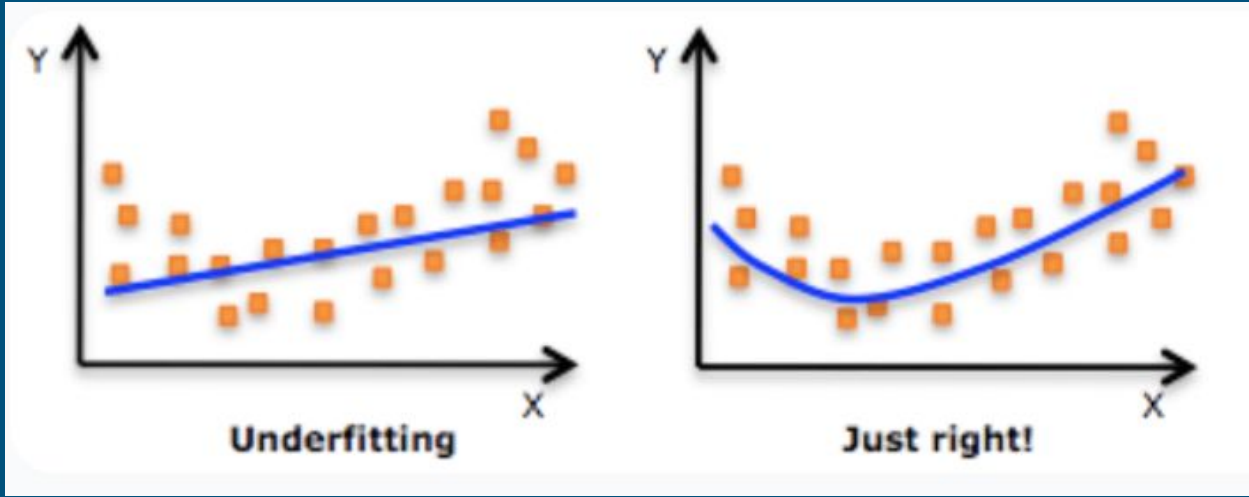
Underfitting

- High bias means the model is underfitting



Underfitting

- The model is too simple
- The model does not understand the underlying pattern



There are two ways models can perform badly

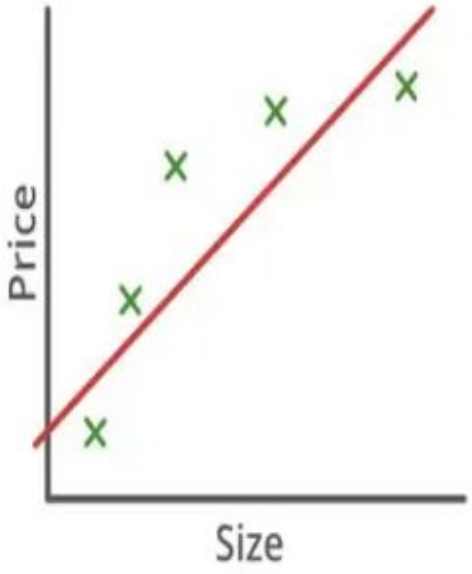
- They can have too much
  - Bias
  - Variance

Variance

- High variance means that the model is sensitive to using different subsets of the data to train.
  - Let’s say you split the training set into 2 buckets, and train model 1 on bucket 1 and model 2 on bucket 2.If the models are otherwise the same but their predictions are very different, then there is high variance.

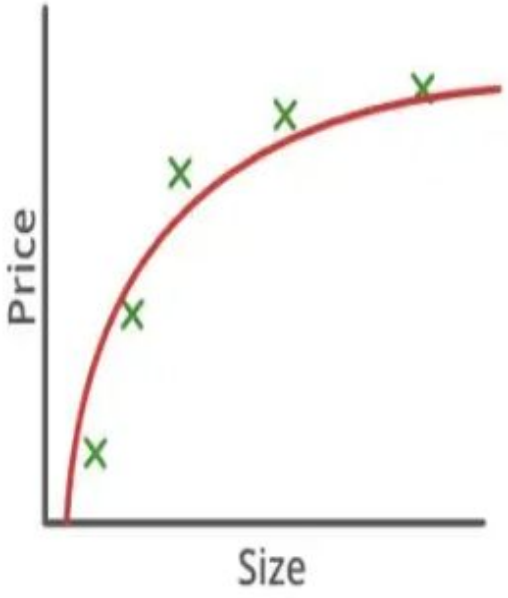
Variance and overfitting

- Models with high variance **overfit** the data
- Model is too sensitive to small changes in the training data



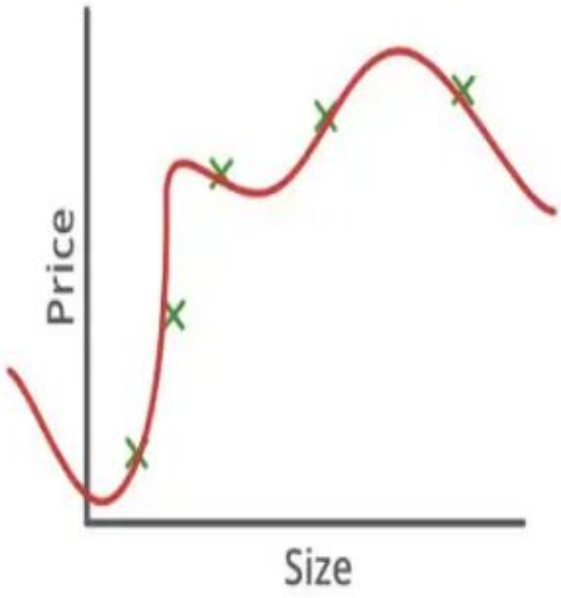
$\theta_0 + \theta_1 x$

High Bias
(Underfitting)



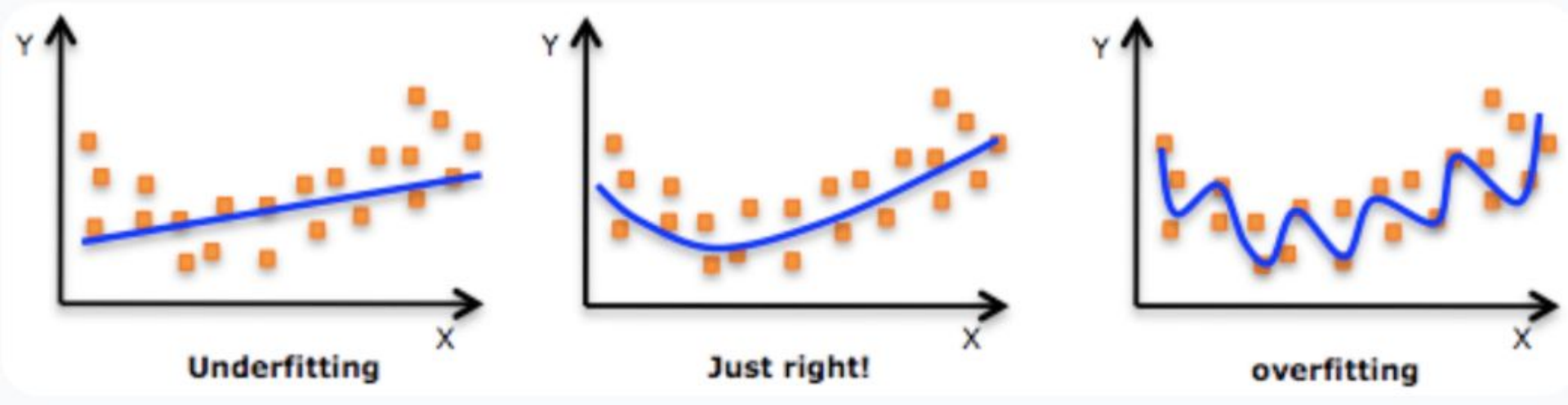
$\theta_0 + \theta_1 x + \theta_2 x^2$

Low Bias, Low Variance
(Goodfitting)



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High Variance
(Overfitting)


Y
X
Underfitting
Y
X
Just right!
Y
X
overfitting

High Variance means Overfitting

How do you know your model is overfitting?

- The model performs well when you test its predictions of the **training set**. BUT
- The model performs badly when you test its predictions on the **testing set**.

Why would your model overfit some data?

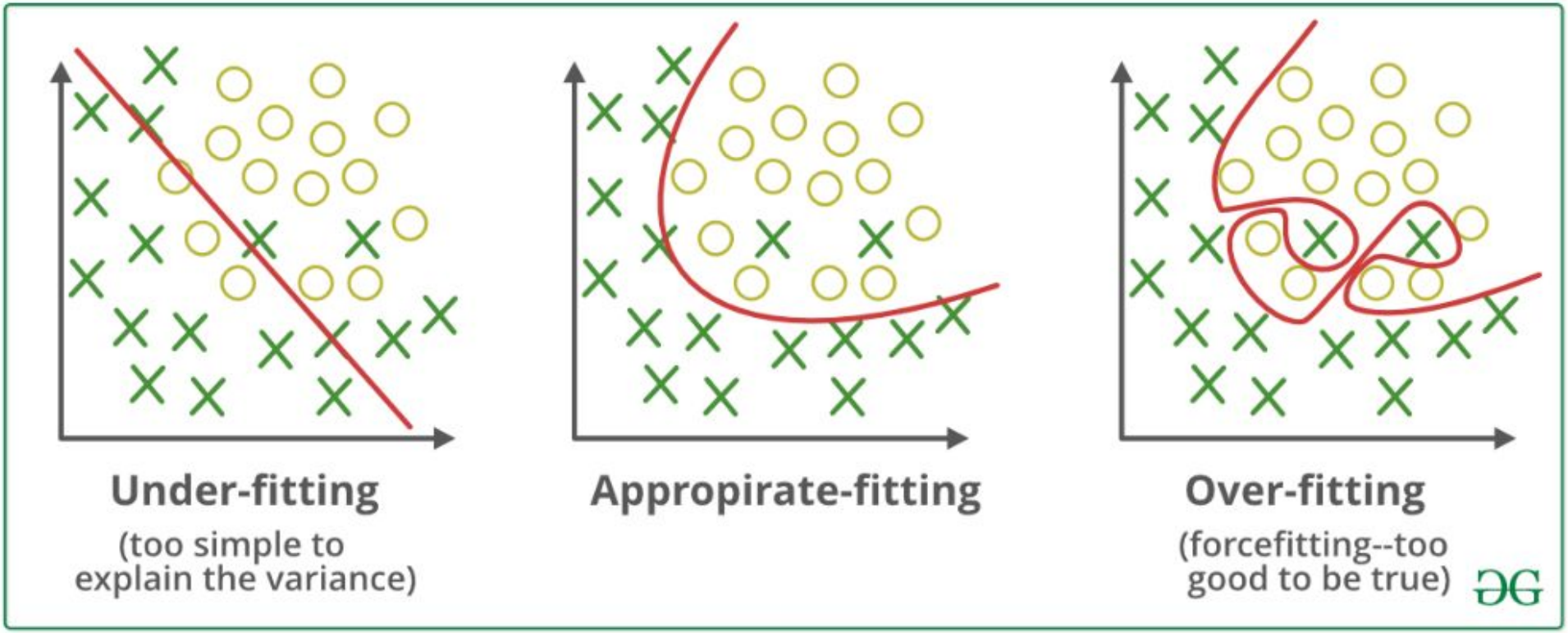
- What do you think?

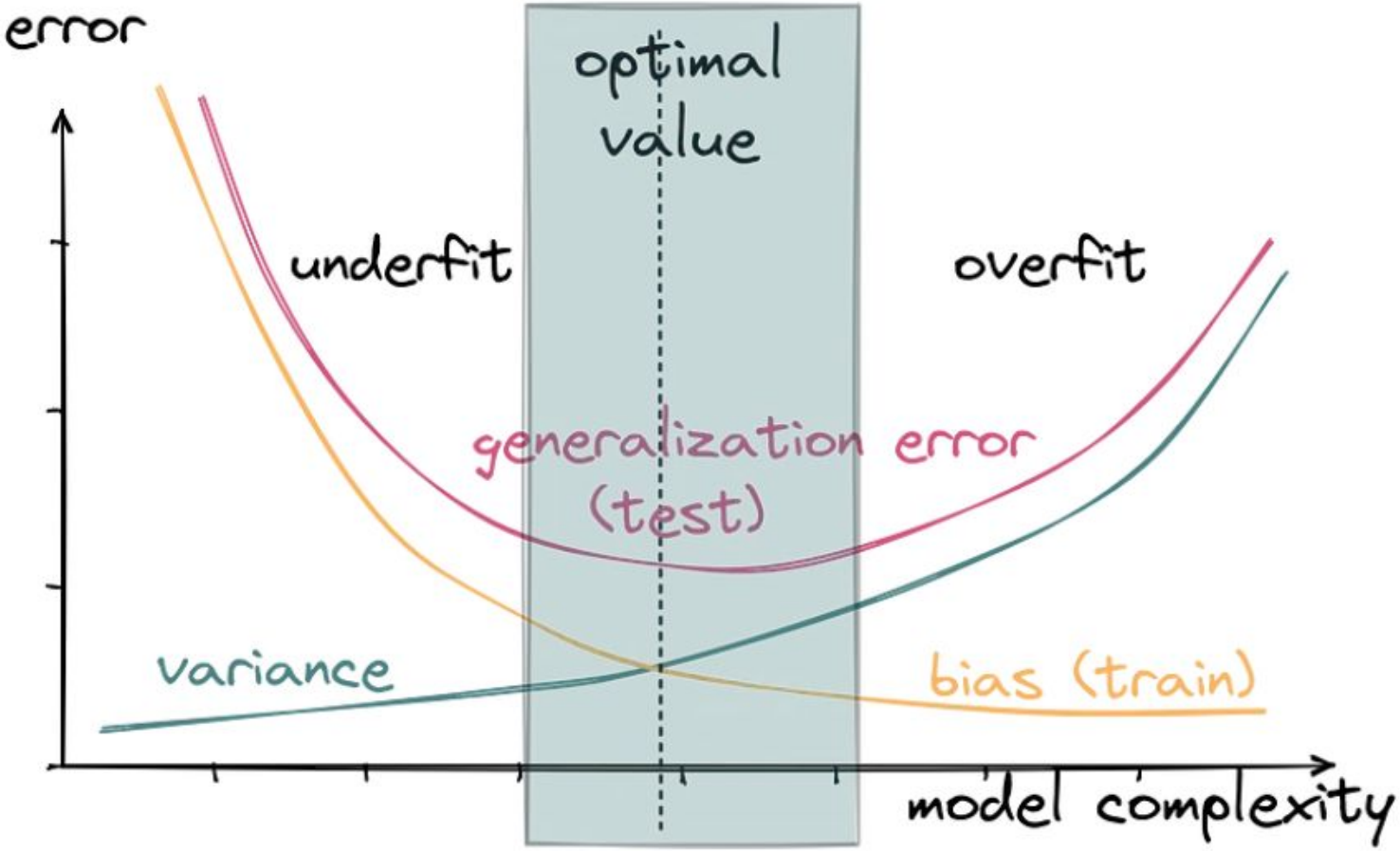
Why would your model overfit some data?

- Spent too much time training on the same bit of data
- The model is too complex
- Model is using too many features
- There is too much noise in the training set and the model learned that noise

How to reduce Variance and Overfitting?

- Get more training data
- Simplify your model
- Make multiple machine learning models and combine their results (average, voting)
  - Called ensemble learning
- Feature selection
  - Mathematical techniques to choose what features to use
- Cross Validation- try splitting your training/testing data in multiple ways to see what combination results in the best model


Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)


error
optimal value
underfit
overfit
generalization error (test)
variance
bias (train)
model complexity
Low Variance
Increasing Variance

Summary

- Models that do not understand the pattern that is in the data are **biased** and **underfit** the data
- Models that depend too much on the training data and does not understand the general pattern has too much **variance** and **overfits** the data.

Thought Experiment

- Neural networks can have as many layers and as many nodes as you want.
- More nodes means the network can understand more complicated models.
- Why not just have all networks have tons of layers and nodes?
- How do you know how many are the right amount?

Thought Experiment

- Data is king for training models
- It can be expensive and time consuming to get more data
- What if we have a small dataset, and I just copy that data a thousand times and used it to train my models? The underlying pattern is still there, but I got a thousand times more data for free!
- What do you think of this?

How to know if our model sucks?

- Must evaluate it!
- This is done with evaluation metrics

What are evaluation metrics?

- Well, they are different ways of evaluating how good your machine learning models are!
- Documentation
- Google sklearn evaluation metrics
- We’ve used accuracy for classification and R squared for regression

Classification

- Your model predicts what “class” or “label” should be applied to a prediction
- Ex:
  - Did they purchase a house or not?
  - Does this image have a picture of a cat, dog, or horse?
  - What disney princess are you?

Accuracy

$$\text{Accuracy} = \frac{\text{No. of correct predictions}}{\text{Total number of input samples}}$$

- Grade between 0-1
- Advantages
  - Easy and intuitive
- Disadvantages
  - We do not know what it got wrong and how. Next slide talks about false positives/negatives

Accuracy disadvantage example

- Let’s say you train a model to predict if someone has lung cancer given a chest scan image.
- Its trained on 1000 data points
  - 999 did not have cancer in the training set
  - 1 did
- If this model just decides to predict that no matter what, no one has cancer, what are the results?

False Positives and False Negatives

- Draw square on whiteboard
- True/False- means it is correct/incorrect prediction
- Positive/Negative- means it is in a class vs not in a class
- Do cancer example
- Ex: True positive means the model correctly predicted the data point was part of the class
- Ex: False positive means the model incorrectly predicted the data point was part of the class
- Ex: False negative means the model correctly predicted the data point was

Why talk about false positive and negatives?

- What do you think?

Measuring Amt of False Positives and Negatives

- Precision- how many of the positive predictions were correct?
- Recall- how many of the negative predictions were correct?

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Cancer example
  - 1000 data points
  - True values: 100 have cancer, 900 do not
  - Predictions: 50 have cancer
- What is Precision Score?

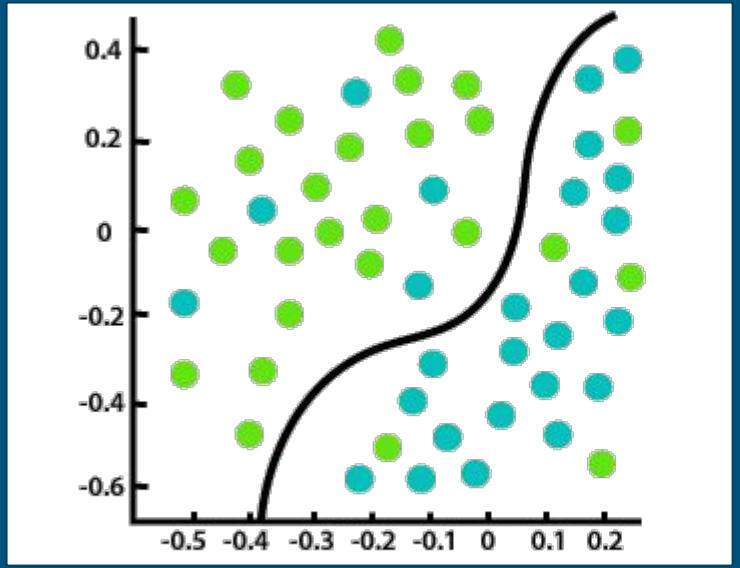
Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

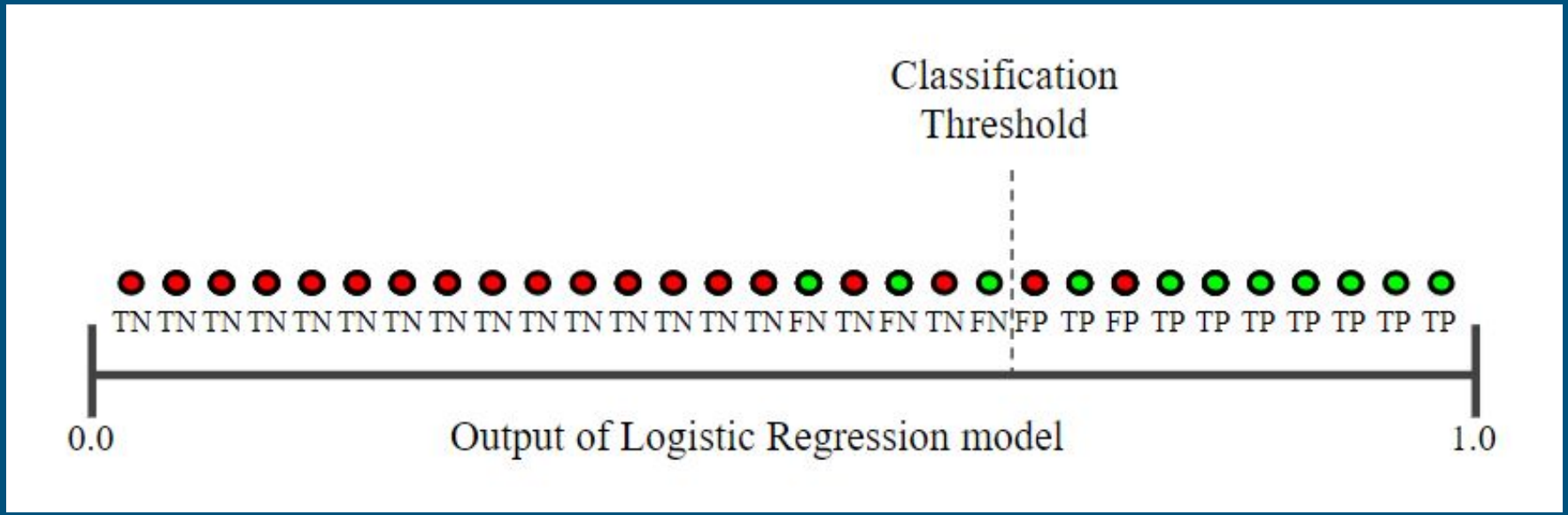
- Cancer example
  - 1000 data points
  - True values: 100 have cancer, 900 do not
  - Predictions: 50 have cancer
- What is recall score?

Tradeoff between precision and recall

- Classification often making a function that splits the data
- Some are obviously in one class or the other
- Some are on the boundary



Looking at position of boundary in classification space



- Draw simpler example on whiteboard and talk about what happens when you shift the decision boundary

Tradeoff between Precision and Recall

- Often, when you train your models, you have to balance between the precision and recall.
- That’s why you should look at both!

Code for precision

```
from sklearn.metrics import precision_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]

precision_score(y_true, y_pred, average="macro")
```

- Parameters:
  - average
    - For binary classification- “binary”
    - For multiple classes- “macro”

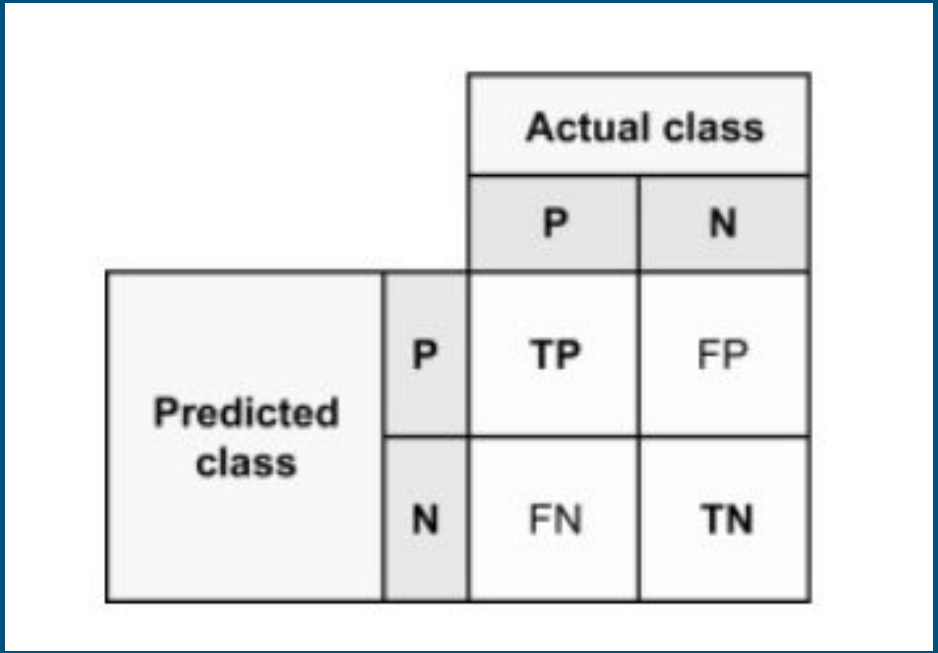
Code for Recall

```
from sklearn.metrics import recall_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]

recall_score(y_true, y_pred, average="macro")
```

- Parameters:
  - average
    - For binary classification- “binary”
    - For multiple classes- “macro”

Confusion Matrix



		Actual class	
		P	N
Predicted class	P	TP	FP
	N	FN	TN

Let’s do example and talk about it

```
from sklearn.metrics import confusion_matrix
y_true = [0, 0, 1, 1, 0, 1]
y_pred = [0, 0, 0, 1, 1, 1]
confusion_matrix(y_true, y_pred)
```

Confusion Matrix Multiclass Example

```
from sklearn.metrics import confusion_matrix
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
```

F1 Score

- Harmonic mean of precision and recall

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score

F1 Score

- Takes both precision and recall into account and weights both evenly
- Let’s do an example on whiteboard then confirm with code- make

```
y_true = [0, 1, 1, 0, 1, 0]
y_pred = [1, 0, 0, 0, 1, 0]
```

F1- score example with code

```
from sklearn.metrics import f1_score
y_true = [0, 1, 1, 0, 1, 0]
y_pred = [1, 0, 0, 0, 1, 0]
print(precision_score(y_true, y_pred, average='binary'))
print(recall_score(y_true, y_pred, average='binary'))
print(f1_score(y_true, y_pred, average='binary'))
```

F1 Score code for multiple classes

```
from sklearn.metrics import f1_score
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
print(precision_score(y_true, y_pred, average='macro'))
print(recall_score(y_true, y_pred, average='macro'))
print(f1_score(y_true, y_pred, average='macro'))
```

- Parameters-
  - Average-
    - “binary” for binary classification
    - “Macro” for multiple classes

F1 Score

- Great way to evaluate classification models.
- It is also good to look at precision and recall too.

Regression Evaluation Metrics

- The goal is for the model to predict a number
- Ex:
  - cost of a home
  - Amt of calories burned
  - Amt of profit

Mean Absolute Error

- Average difference between prediction and true value

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

MAE example- manually and with code

- Do example by hand, and then run code

```
from sklearn.metrics import mean_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_absolute_error(y_true, y_pred)
```

MAE- Pros and Cons

- Pros
  - Intuitive can give you a sense of how far your predictions were
- Cons
  - Hard to compare your model to other models in the same field
  - Talk about thesis comparison

Mean Squared Error (MSE)

- Same as MAE except you square the differences

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2$$

MSE- example by hand and with code

```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
```

MSE- Pros and Cons

- Pros
  - More efficiently trains then MAE (easier to perform derivative)
  - Big errors become BIGGER- models respond by minimizing them
- Cons
  - Units are now squared
- Cons can be addressed!

Root Mean Squared Error (RMSE)

- Same as MSE but you square root it!

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

RMSE example by hand and code

```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred, squared=False)
```

- squared = False means take the square root of the MSE

RMSE Pros and Cons

- Pros
  - All the pros of MSE
- Cons
  - I dont know of any!
  - This is why it is used so often for machine learning
  - Has same units as target var

R Squared- Coefficient of Determination

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

- Represents how much variation in the features is controlled by the target
- $\hat{y}$ is predicted value $\bar{y}$ is average of dataset
- Note- R squared is not you squaring something called R
- Is R squared always positive?
- Talk about scale- what does 0-1 mean?

R squared- code

```
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)
```