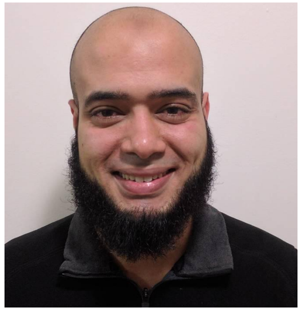
Machine Learning
Model Selection

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Modeling So Far

- We've studied several machine learning algorithms based on linear regression:
- Multivariate linear regression
    - This has a critical hyperparameter: learning rate (α)
- Polynomial linear regression
    - This also involved a hyperparameter: the polynomial degree
- There are **many** other algorithms available, such as:
    - Neural Network, Support Vector Machines (SVM), and Random Forest, among others
- These algorithms can be used to build models for various common tasks
- However, the question is: How can we select the best model among them?
    - The answer? We use a process called "Model Selection".
    - The best model is the one that **generalizes** well to unseen data

Generalization

- What is the purpose of ML? Mainly **generalization**
  - We **train a model** on a dataset to learn the patterns/**associations** within it
  - Once the model is trained, it's **released** in the wild to work on new, **unseen** data!
- For example, Tesla uses videos from the US, training their vehicles to detect pedestrians on the streets
  - When Tesla vehicles are released in a different country, such as South Africa, they will face new, unseen data
- The key ML question here:
- Will the learned model work on the **new, unseen** data?

Generalization

- **Generalization** refers to your model's ability to work properly to **unseen data**, drawn from the **same distribution** as the one used to create the model
- If your model **doesn't work** on the training data, it is useless!
- The question is whether a model that performs well on training data will also perform well on new, unseen data
    - Did it learn the associations? Or **memorized** the examples?
- For example, if 5 different models all score 98% accuracy on the training data, how do we decide **which one to select**?
- Do you think we can count only on a single training dataset? Any thoughts?!

The Same Distribution!

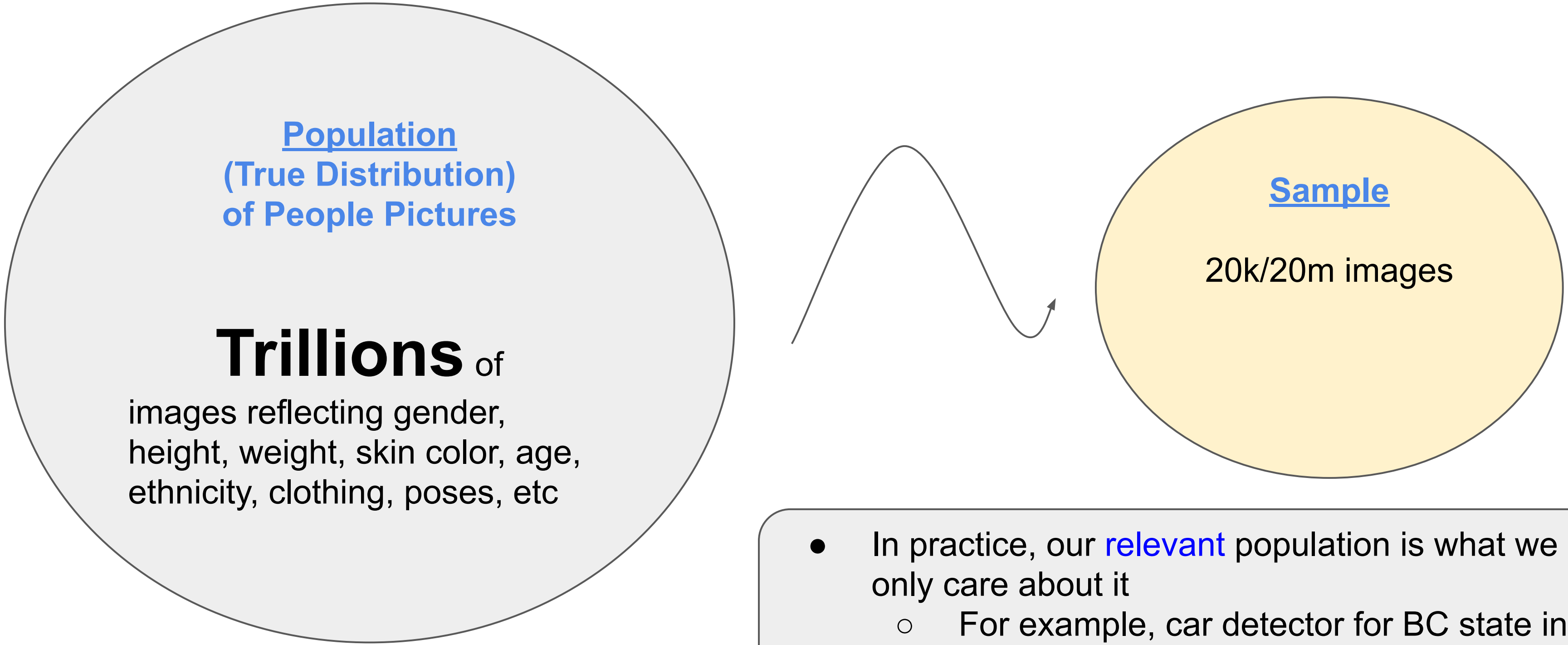
- Imagine that we trained a people detection model on images of white adult males. However, in reality we test on images from black female children.
- Will the model be able to generalize to this new, unseen data?

No! The data comes from two very different distributions!

List all the differences?

Img src1, src2

Population vs Sample



Population
(True Distribution)
of People Pictures

Trillions of
images reflecting gender, height, weight, skin color, age, ethnicity, clothing, poses, etc

Sample

20k/20m images

- In practice, our relevant population is what we only care about it
    - For example, car detector for BC state in canada

Population vs Sample

We train on a sample, due to availability/resources

20k/20m images

In reality, when the model is deployed, it will be tested on a much more diverse population, for which the **true distribution is unknown**

Will the model work well?
If yes, then the model achieves **generalization**

Trillions of images reflecting gender, height, weight, skin color, age, ethnicity, clothing, poses, etc

I.I.D Sample

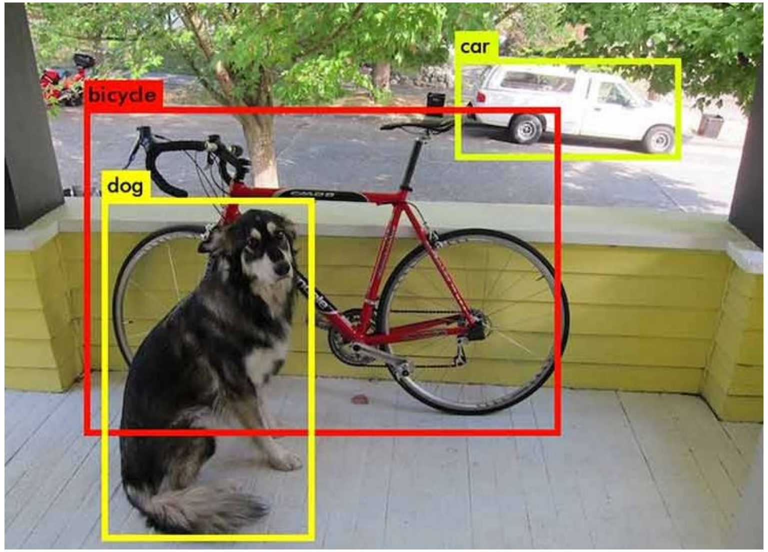
- IID sample: independent and identically distributed
- Imagine we toss a coin 8 times (HHTHTHHT)
- **Independent**: each toss has nothing to do with the previous one
  - samples are not influenced by each other
- **Identically distributed**: each toss has the same probability of being head
  - Samples are generated independently of the others (no relationship/correlation)

I.I.D Sample

- Imagine we trained on a sample that has mean of 10 and variance of 2
- However, in testing on another sample with mean of 70 and variance of 4
  - Do you think the model will work? No
  - Statistical properties of the data should be **consistent** across all samples
- The moral of that in ML: your dataset need to be a **real representative**!
  - The bigger the gap between sample and population, the higher the error!
- Tip: IID is not satisfied in temporal (time-series) data or spatial data
  - Use temporal/spatial specialized models
- In practice, IID could be violated to some extent. However, with enough samples and diversity, it is ok that some samples violates it
  - Shuffling in gradient descent is critical
- IID plays critical role in simplifying the mathematics of some models

Question!

- In object detection task, we draw a box around each object of interest
  - Here 3 objects are detected
- Assume our dataset has 20 classes
- In the dataset, the triple of dog, bicycle and car appears only/always **together**
  - E.g. no car alone
- Any notes?



IID in CV (Future note)

- In **semantic segmentation**, the IID assumption implies that each pixel or patch in an image is **independent** of others and follows the same distribution. This assumption allows the model to learn to assign semantic labels to each pixel based on **local information** without considering the global context
  - This is a hidden important assumption in current deep learning models
- Consecutive frames in a **video** or images in a **panorama** violates IID. In videos:
  - We can use the whole clip as an example or divide to sub-clips
  - Or we can use the frames as examples but make sure enough videos creates diversity
    - Batches should have a good shuffle of the data

Toward Generalization

- The standard approach is to **divide** your **collected data** into **3** datasets!
- **Training dataset**
  - Used to train your different models (**fit parameters**)
- **Validation (dev) dataset**
  - Used to **tune the hyperparameters** of the trained models
    - For example, how can the best learning rate for a trained model be derived?
  - Uses the performance metrics (e.g. accuracy) to choose the **BEST** model
- **Test dataset**
  - **Evaluate the BEST** model to make sure it really generalizes (*unbiased* performance)
  - Keep the test dataset separate from the other two datasets as much as possible - only use it to evaluate the final model!
- *This is an **empirical** approach. The **theoretical** statistical approaches (e.g. generalized bounds) are limited in practice*

Will the model generalize?

- If a model performs well on the test set, will it generalize?
- It's a **good indicator**, but this is no guarantee. Consider:
- 1) Is the training dataset representative and unbiased? (wide diversity)
    - Do you target male images? Females? Black? White? Asian? *Ok, Collect data for all!*
    - Consider conditions: A) iid B) stationary
        - Stationary means it doesn't change over time
        - A non-stationary example: collecting house prices, but they keep changing due to inflation
- 2) Is the test set large enough? Is it drawn from the same distribution as the training set?
- 3) Are the appropriate performance metrics being used?
- 4) Is there any misuse or cheating in the evaluation process?
    - Some people try to get insights from the test set to influence the hyperparameters/models

Question!

- Which of the following types of data are **non-stationary**?
    - Non-stationary means that they change over time)
- 1) Weather data
- 2) Rainfall measurements
- 3) Temperature readings
- 4) Stock prices
- 5) Interest rates

- All of the above types of data are non-stationary!
- This type of data is called **Time Series data** and requires careful handling

Train-Val-Test Split Percentages

- What are good percentages for **splitting the data**? No optimal answer
  - Training data should be as large as possible
  - The validation and test sets should be **representative** enough to be **good indicators** of the model's performance (similar size)
  - Other factors: how many hyperparameters to finetune? More ⇒ Larger validation set
- Some guidelines to give you a sense of that:
- Small to medium datasets (e.g. a few hundred to thousands of examples):
  - 80% train, 10% val and 10% test
  - 70% train, 15% val and 15% test
- Large datasets (e.g. millions of examples):
  - 98% train, 1% val and 1% test (1% of 10 million images is 100k examples)
  - 96% train, 2% val and 2% test
- Coding tip: always **shuffle** your data before splitting for **mini-batch** processing

Experimentations Example

Model	Hyperparameters	Train%	Val %
Polynomial Regression	Config #1	23%	
Polynomial Regression	Config #2	85%	15%
Polynomial Regression	Config #3	82%	83%
Neural Network	Config #1	92%	70%
Neural Network	Config #2	84%	85%
XGBoost	Config #1	40%	
XGBoost	Config #2	75%	65%
XGBoost	Config #3	79%	65%

Test %
80%
79%

Question!

- One of the Computer vision teams working at **Apple** compared the performance of 2 models for an ML task
  - Model A had a performance of 87% on the validation and test sets
  - Model B had a performance of 85% on the validation and test sets
- The team decided to use and deploy **model B**. Can you guess some possible reasons?

- Model A requires a lot of memory and processing while **model B is light**
- The performance gap is only 2%
- As the model will be used by their Apple phones, it is wise to deploy a lighter models on Mobiles
  - A lighter model won't consume as much battery or cause overheating

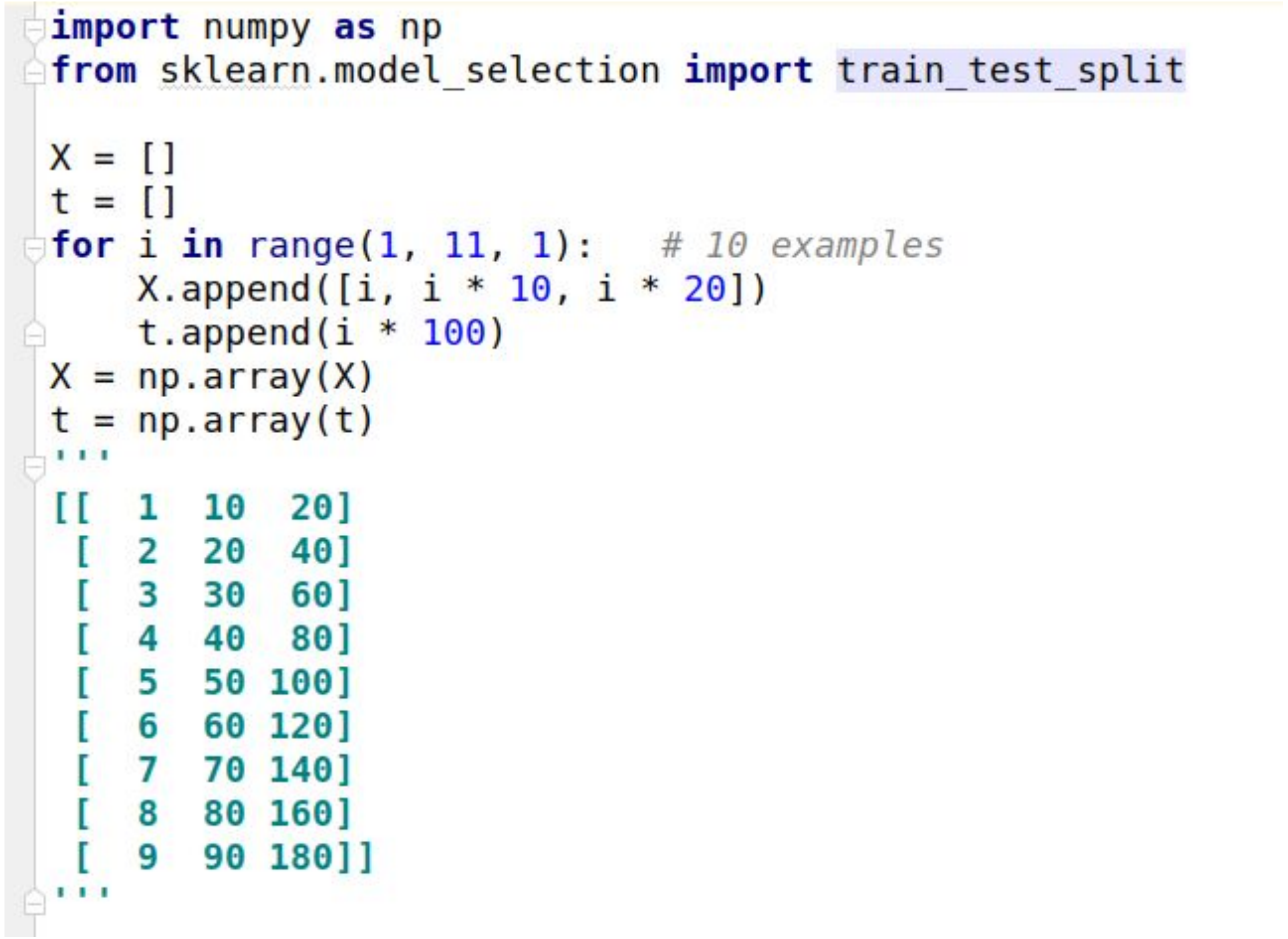
Question!

- PetsSmart shop has 500 animals of 53 different categories (cat, dog, etc) and they want to build an ML model for these animals. An ML engineer took 100 images for each animal, resulting in 50,000 images. These images were then shuffled and split into train/validation/test sets.
- There is a hidden mistake in the **data split**. What is it?

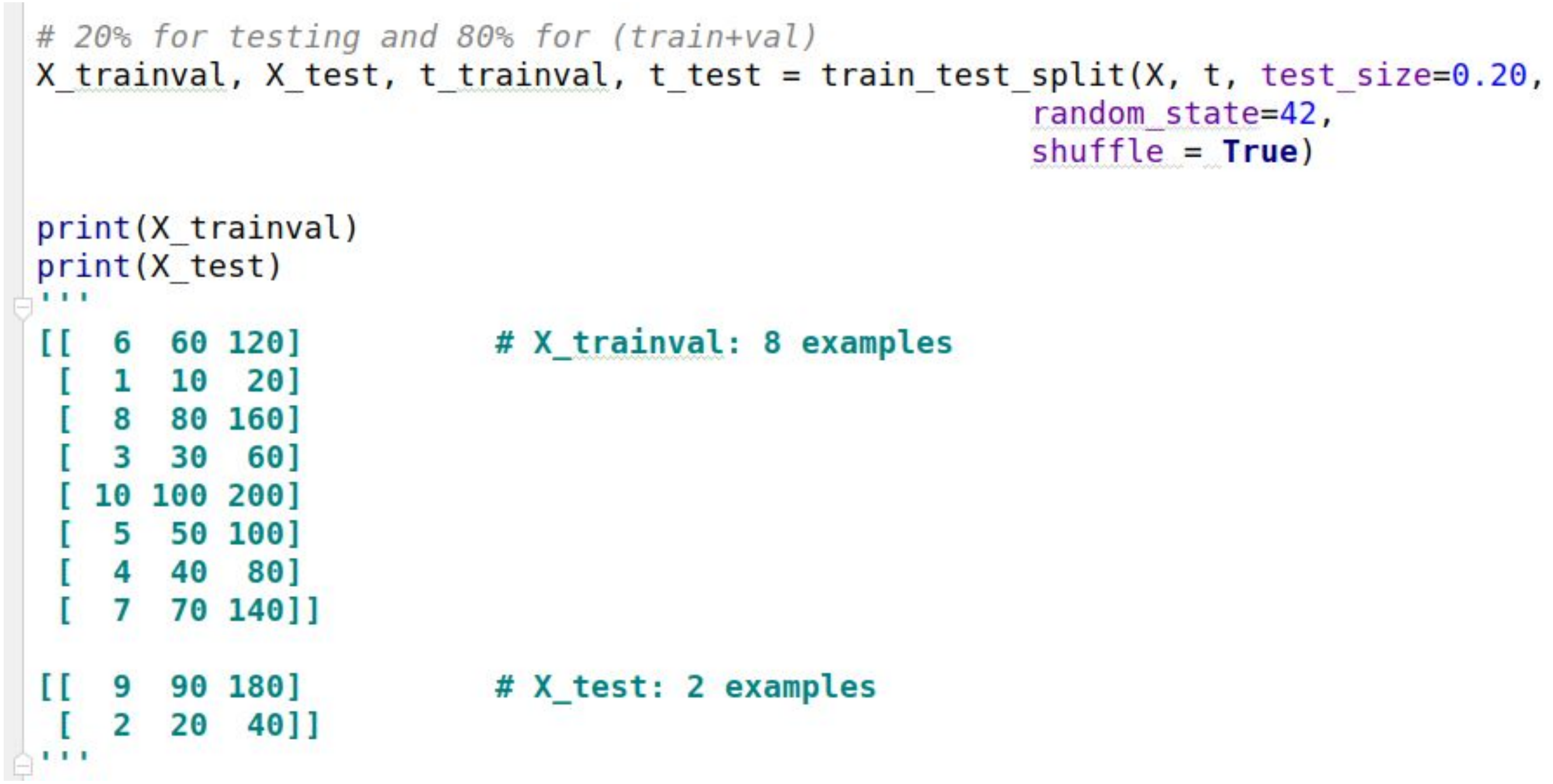
- Data leakage problem
  - Data leakage: information is leaking between training and inference resulting in a weak model
  - The pictures of the **SAME** animal will be in both train and test sets
    - Test then already knows well about such scenarios
  - To avoid this, we should first divide the 500 animals into train/validation/test sets
  - Then, the images of each animal are ONLY in one data part (e.g. train or val or test).
    - Entities (e.g. animals, patients, road segment) should per unique per data split

sklearn.model_selection.train_test_split

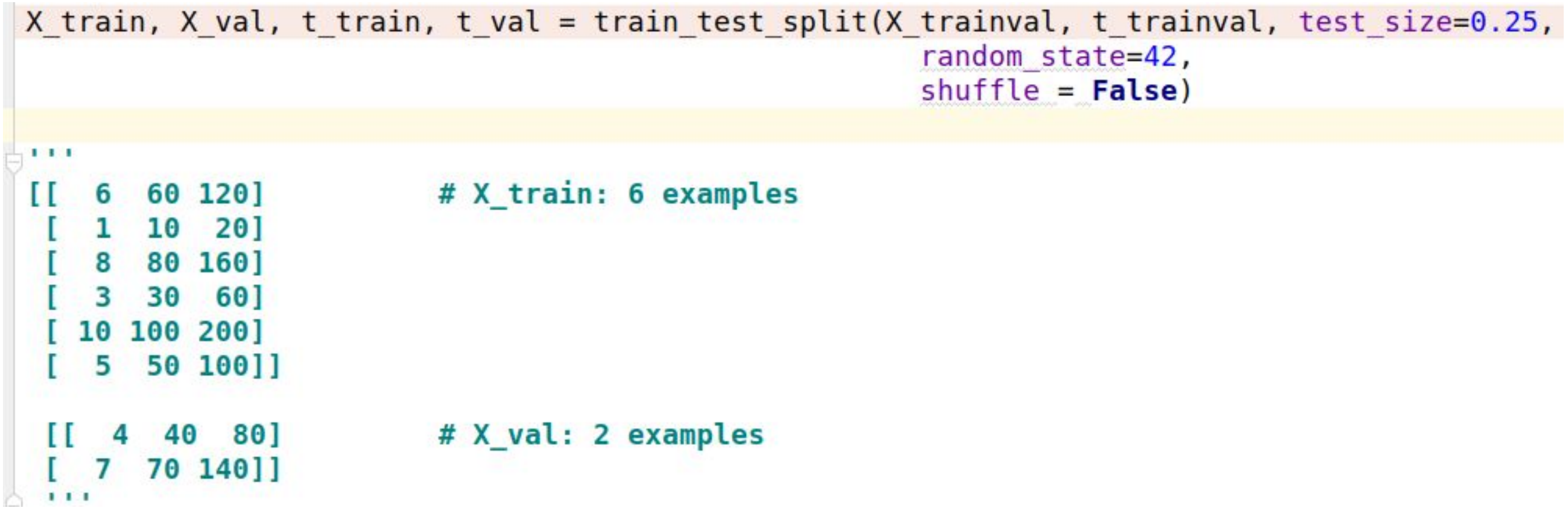
- Sklearn provides a function that splits data into two parts based on a specified percentage
- But we need 3 parts: train, val and test
- Assume we have 10 examples and we want to split into: 6 (train), 2 (validation), 2 (test)
- First, split it to 20% for testing and 80% for train+val
- The 80% of train+val are 8 examples. We want to split to 6 (train), 2 (val)
    - 2/8 = 0.25, So 25% of the trainval will be 2 examples
- I hope in the future they give better API



```
import numpy as np
from sklearn.model_selection import train_test_split

X = []
t = []
for i in range(1, 11, 1):   # 10 examples
    X.append([i, i * 10, i * 20])
    t.append(i * 100)
X = np.array(X)
t = np.array(t)
'''
[[  1  10  20]
 [  2  20  40]
 [  3  30  60]
 [  4  40  80]
 [  5  50 100]
 [  6  60 120]
 [  7  70 140]
 [  8  80 160]
 [  9  90 180]]
'''
```



```python
# 20% for testing and 80% for (train+val)
X_trainval, X_test, t_trainval, t_test = train_test_split(X, t, test_size=0.20,
                                                          random_state=42,
                                                          shuffle = True)


print(X_trainval)
print(X_test)
'''
[[  6  60 120]          # X_trainval: 8 examples
 [  1  10  20]
 [  8  80 160]
 [  3  30  60]
 [ 10 100 200]
 [  5  50 100]
 [  4  40  80]
 [  7  70 140]]

[[  9  90 180]          # X_test: 2 examples
 [  2  20  40]]
'''
```



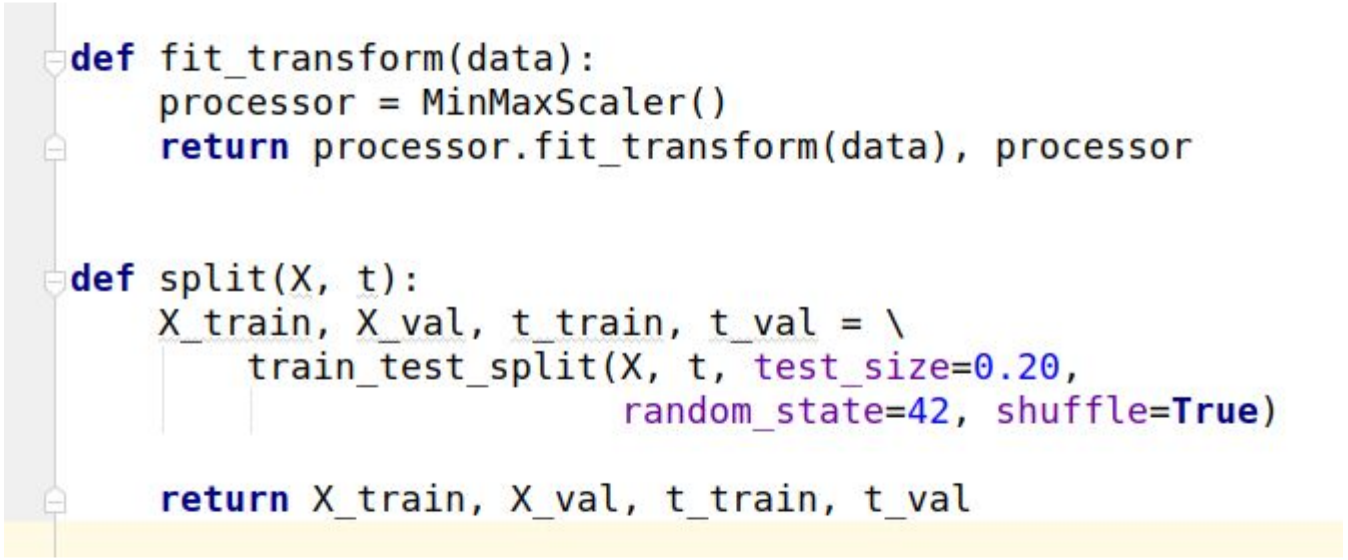
```
X_train, X_val, t_train, t_val = train_test_split(X_trainval, t_trainval, test_size=0.25,
                                                  random_state=42,
                                                  shuffle = False)

'''
[[  6  60 120]          # X_train: 6 examples
 [  1  10  20]
 [  8  80 160]
 [  3  30  60]
 [ 10 100 200]
 [  5  50 100]]

 [[  4  40  80]         # X_val: 2 examples
 [  7  70 140]]
 '''
```

Utilizing the ALL data for medium-sized datasets

- Sometimes our dataset has a few thouthands examples
  - Our deployed model should utilize all of our data as much as we can!
- Procedure
  - Fit a model using the **training** set
  - Fine-tune the parameters using the **validation** set
  - **Fix** your hyperparameters
  - Train a model using your **training set** and evaluate on your test set: Model 1
  - Train a model using your **training+validation** set and evaluate it on your test set: Model 2
    Performance shouldn't drop with more data, hopefully
  - Split your test set to: mini-test (maybe 3%) and most-test
  - Train a model using your **training+validation+most-test** data and evaluate on mini-test
    - *In large scale datasets, we can use only training model or trainval model*
  - Deploy this model final if everything looks good

Utilities

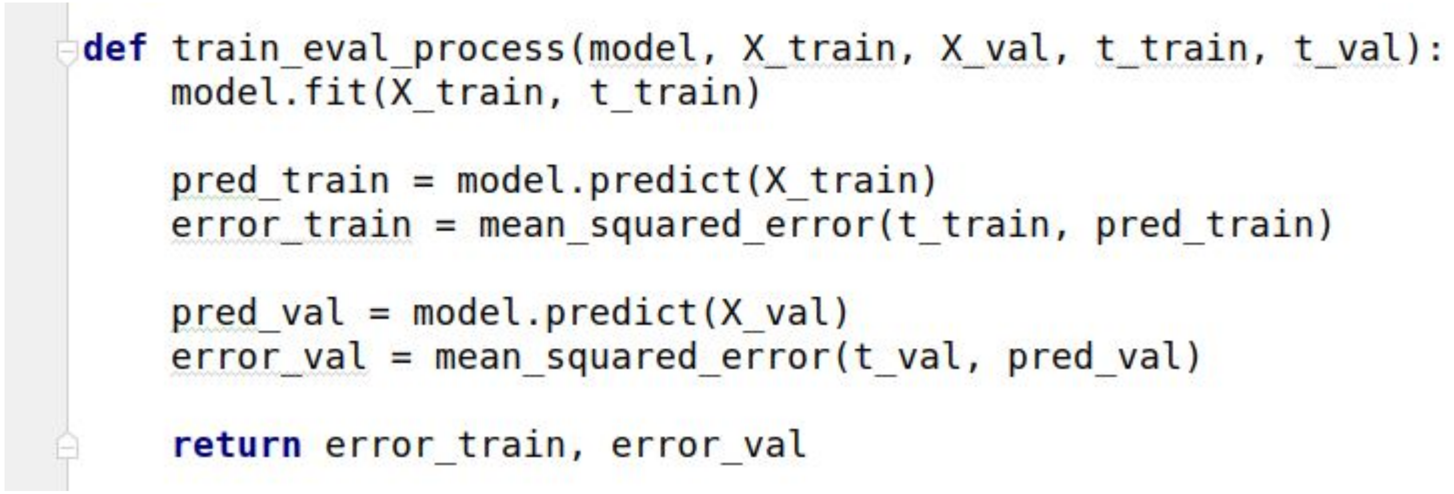
- Basic transformer
- Split the data into a training set and a validation set, where the validation set is 20% of the total data



```
def fit_transform(data):
    processor = MinMaxScaler()
    return processor.fit_transform(data), processor


def split(X, t):
    X_train, X_val, t_train, t_val = \
        train_test_split(X, t, test_size=0.20,
                                        random_state=42, shuffle=True)

    return X_train, X_val, t_train, t_val
```

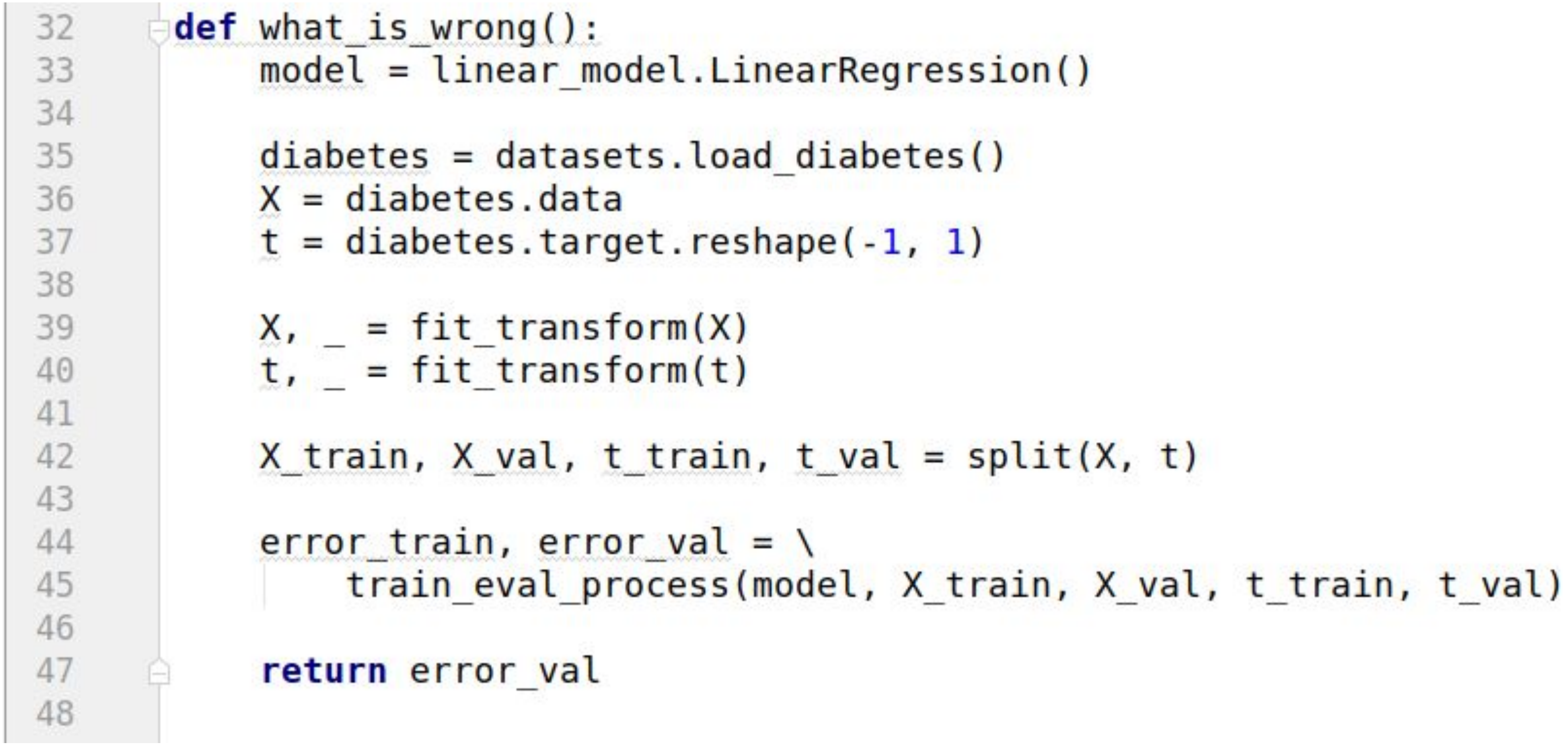
Utilities

- Fit a regression model
- Return the errors for both the training set and the validation set



```
def train_eval_process(model, X_train, X_val, t_train, t_val):
    model.fit(X_train, t_train)

    pred_train = model.predict(X_train)
    error_train = mean_squared_error(t_train, pred_train)

    pred_val = model.predict(X_val)
    error_val = mean_squared_error(t_val, pred_val)

    return error_train, error_val
```

Find a data leakage mistake!



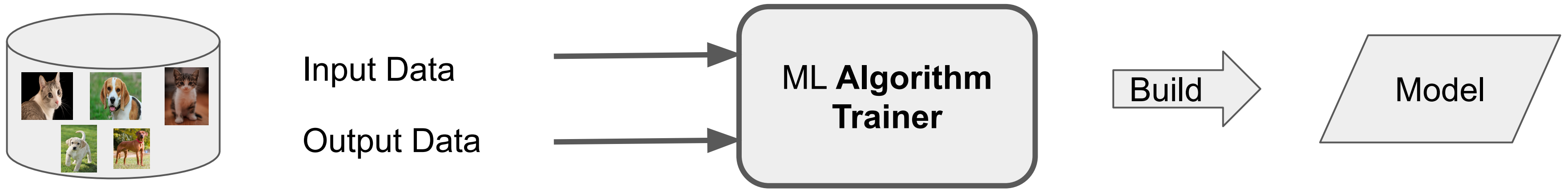
```python
def what_is_wrong():
    model = linear_model.LinearRegression()

    diabetes = datasets.load_diabetes()
    X = diabetes.data
    t = diabetes.target.reshape(-1, 1)

    X, _ = fit_transform(X)
    t, _ = fit_transform(t)

    X_train, X_val, t_train, t_val = split(X, t)

    error_train, error_val = \
        train_eval_process(model, X_train, X_val, t_train, t_val)

    return error_val
```

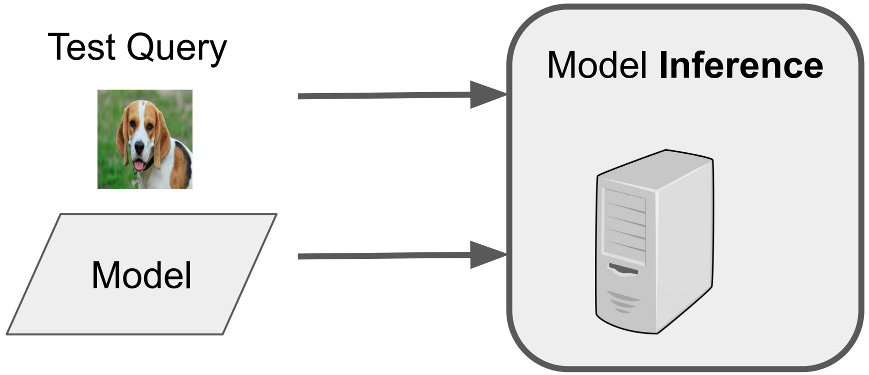
Data Leakage

- Let’s meet another common/hidden data leakage issue
- We know statistics/information about our collected data, but in practice we don’t have the test data. We need to simulate the performance evaluation as if we DON’T have statistics about the test-data (a single test point at a time)
- Assume a training feature has min-max range = [7, 2500] and the corresponding test/val feature has min-max range = [20, 2800]
    - We should use the range [7, 2500] to scale **EVERY DATA SPLIT**
    - If we used the combined data [7, 2800], we **leaked** an information from the test/val set to the training set!
- Correctness Rule: Preprocessing parameters used for the training data must be used to preprocess the WHOLE dataset
    - Note: recompute them for your trianval or trainvaltest final experiments

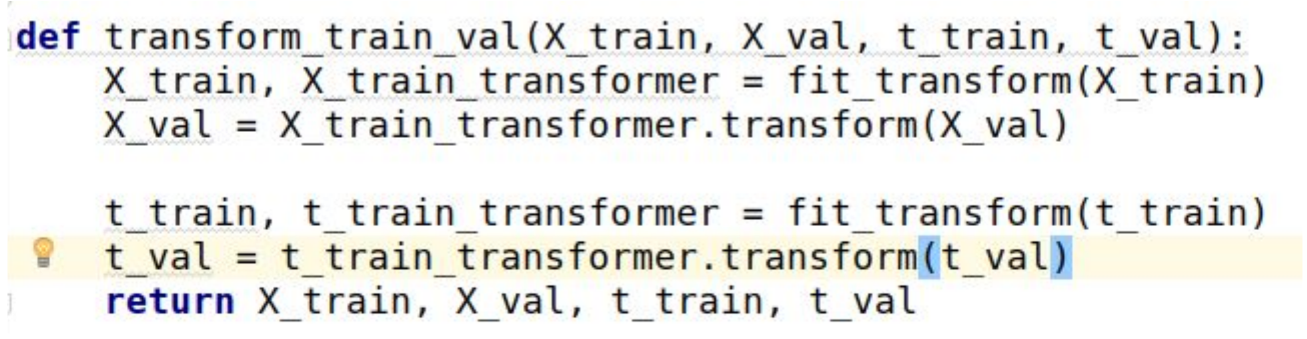
Inference on a single example!

- There is no such validation or testing set in real life deployment
  - We only get **separate** texting examples

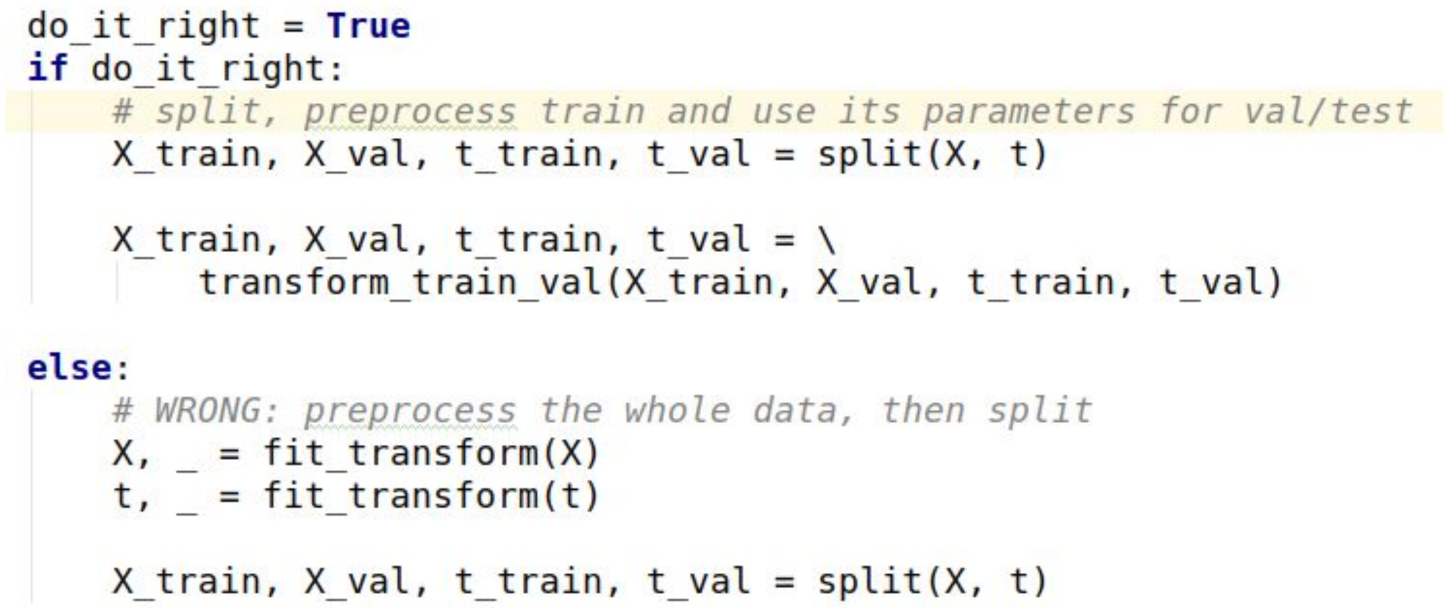




- We shouldn't build statistics from val/test sets
  - Don't exist in real life



```python
def transform_train_val(X_train, X_val, t_train, t_val):
    X_train, X_train_transformer = fit_transform(X_train)
    X_val = X_train_transformer.transform(X_val)

    t_train, t_train_transformer = fit_transform(t_train)
    t_val = t_train_transformer.transform(t_val)
    return X_train, X_val, t_train, t_val
```



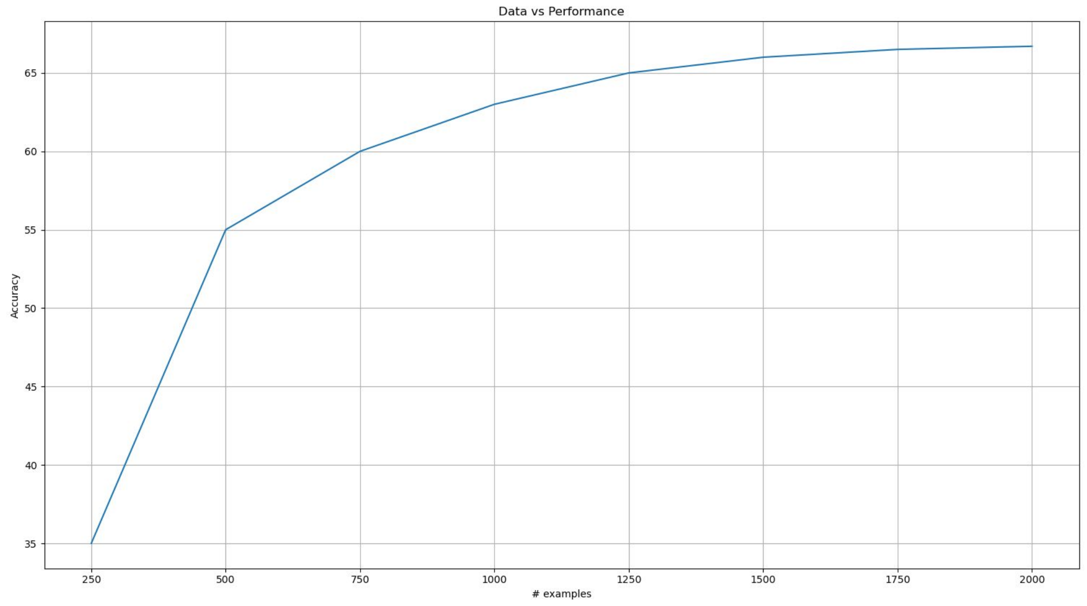
```python
do_it_right = True
if do_it_right:
    # split, preprocess train and use its parameters for val/test
    X_train, X_val, t_train, t_val = split(X, t)

    X_train, X_val, t_train, t_val = \
        transform_train_val(X_train, X_val, t_train, t_val)

else:
    # WRONG: preprocess the whole data, then split
    X, _ = fit_transform(X)
    t, _ = fit_transform(t)

    X_train, X_val, t_train, t_val = split(X, t)
```

Test Server

- We know that users can make mistakes during training
  - Sometimes, people intentionally abuse the data!
- Machine learning challenges often involve releasing a dataset for users to train on
- However, the final performance is evaluated on a separate server
  - The user is given test data (input only)
  - The user computes the answers and uploads them to the **server to judge it**
  - To prevent fine-tuning on the server, users are typically limited in the number of submissions they can make within a given time window
  - Meanwhile, still the user may fall in the previous mistake as the user have access to the test inputs

Question!

- Should we collect more data to improve our performance?
- Seems no much improvements with more data. Better improve the modeling



Relevant Materials

- What are i.i.d. random variables?
- Why Do We Need a Validation Set in Addition to Training and Test Sets?
- Side note: a lot of the philosophy of what we are doing in ML's *framework* has roots in **Scientific Research** (hypothesis, theory, experimentation, sample, bias, induction, generalization theory)

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”