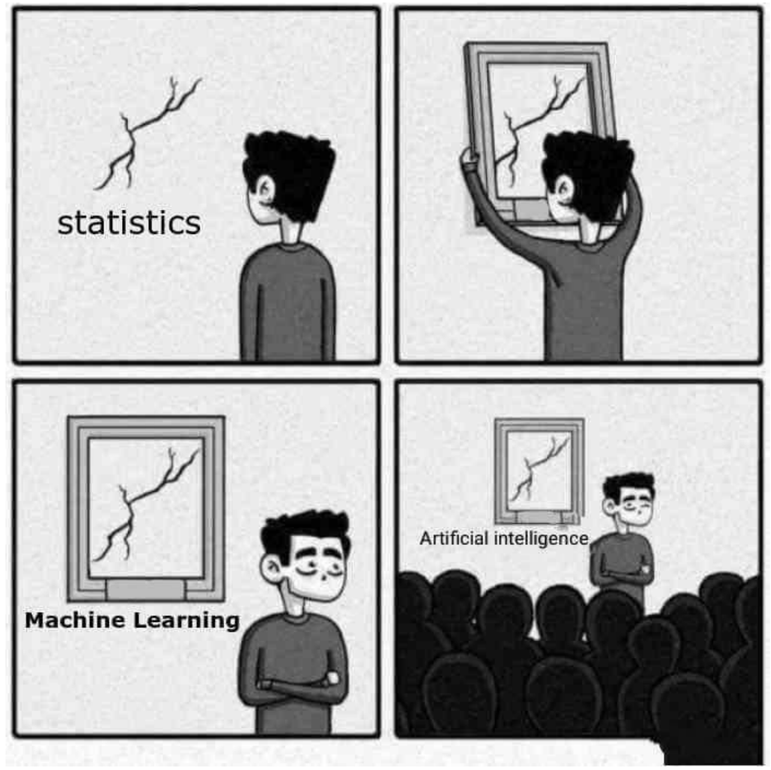
Week 2



Announcements

- HW1 correction and due next week on Blackboard

Agenda

- End to end machine learning
- Refresher on linear regression
- Gentle intro to gradient decent

End to end machine learning project

End to end ML project

Steps:

Understand the problem you want to solve

Get the data

Discover and visualize the data to gain insights

Prepare the data for machine learning algorithms

Select a model/s and train it

Fine-tune your model

Present your solution

Launch monitor and maintain your system

Understand and frame the problem

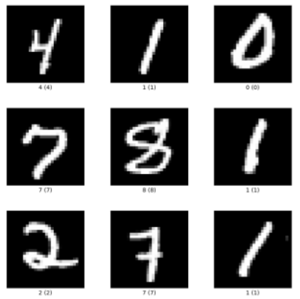
- Before you begin a machine learning project, you should understand the problem you’d like to solve.
- What is the objective?
- What do you hope to learn or achieve?
- How would a group of people, organization or society benefit from the project?
- Is there currently a solution or prediction method used on this problem? If so, how accurate is it? Use this as a basis of comparison for your new machine learning method.
- What type of problem is it? Supervised, unsupervised, classification, regression

Get the data

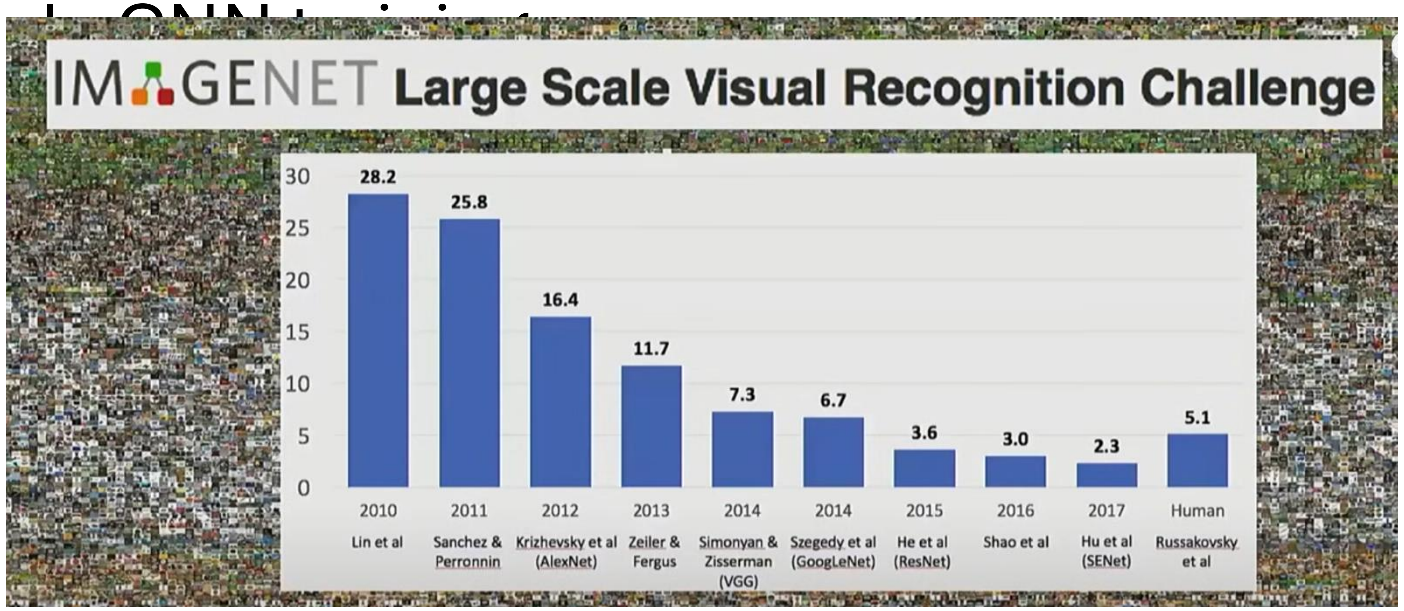
- For machine learning practice there are many sources.
    - UCI Machine Learning Repository
        - https://archive.ics.uci.edu/ml/index.php
    - Kaggle
        - https://www.kaggle.com/
    - Open Data on AWS
        - https://registry.opendata.aws/usage-examples/
    - Google dataset search
        - https://datasetsearch.research.google.com/
    - US Government Open Data (many other countries have something similar)
        - https://www.data.gov/
    - Github repository of “awesome” datasets
        - https://github.com/awesomedata/awesome-public-datasets

See more in textbook

Data is important in ML



- **1980 – Neocognitron (Fukushima)** introduces the concept of convolutional layers and hierarchical feature extraction.
- **1998 – LeNet-5**: a CNN architecture for digit recognition (MNIST), considered the first practical CNN success
- **2010 – ImageNet dataset** released (~1M images, 1K classes), setting the stage for large-sc



The problem – CA Housing prices

- Build a model to predict CA median housing prices by district
- Predict the price in any district given the following ***features/attributes/independent variables (X)***:
    - location, median age of homes, total rooms, total bedrooms, district population, number of households per district, median income, proximity to ocean.
    - we are also given the ***label/target/outcome/dependent variable (y)*** – median house value.
    - What type of machine learning algorithm will we use? Supervised or unsupervised?
    - What type of problem is this?

Pipeline

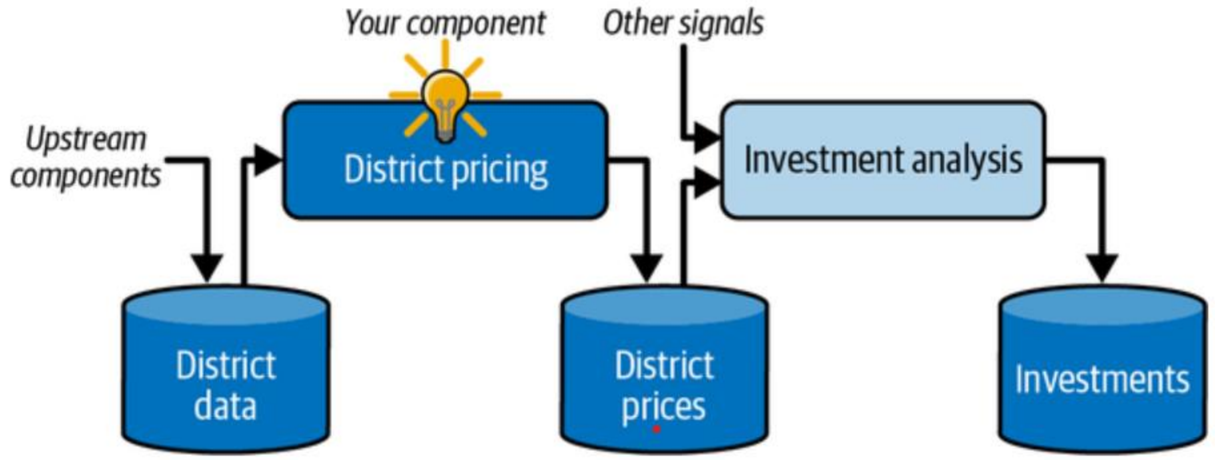


Figure 2-2. A machine learning pipeline for real estate investments

Your model will produce output that will be fed into another machine learning algorithm to determine if your company should invest in the area.
A system that reads in data and outputs data to be used in another model or system is called a ***pipeline***.

Select a performance measure

For our regression problem we will use root mean square error (RMSE) or the mean absolute error if there are outliers in the dataset.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} h\left(\mathbf{x}^{(i)}\right) - y^{(i)})^2}$$

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m} h\left|\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$$

RMSE and MAE measure the difference between the model's predictions and the actual labels.

Explore the data

- Learn about the data’s samples and attributes
- How many samples and how many attributes?
- Are they numeric or categorical or both?
- Are there missing values?

- How can we do this using a Pandas dataframe?

Create a training and test set

- Before you do any data clean up or scaling, create the training and test set.
- Use about 70% of the original dataset to train the machine learning algorithm. For example, use 70% of the data to find the regression line to make predictions.
- Set aside about 30% of the data for validation and testing
  - Make sure the same 30% is used for testing every time, otherwise you have a data or feature leakage problem
  - Confirm that the test set is representative of the training set and the real world
- Make predictions on the training, finetune on validation, evaluate test set.

Explore the training set data

- Create histograms of numeric attributes
- Check for outliers
- Look for normal distributions
- Check for the scale of the data
- Check if features are independent

- Determine if new attributes should be created
  - If distribution is skewed right, use the log or square root
  - If feature does not represent problem – e.g. total rooms in district should be converted to totals rooms per home
  - If a feature missing too many values – populate missing values or remove the feature

Visualize the training data

- If the training set is large, use a random sampling of the data to visualize it
- Create scatter plots and take advantage of sklearn’s c (color) and s (size) options to visualize features by outcome
- Look for correlations between each feature and the outcome using scatter plots or getting the correlation coefficient
- Look for features that are most correlated to the outcome
- Create scatter plots of the features against each other looking for correlations. You may be able to drop a correlated feature

Prepare the data: data cleansing

After exploring the data you may find some samples or attributes have missing values

Options:

1. Delete rows with missing values
2. Delete a feature with missing values
3. Use the median of the feature to fill in the missing value
4. Use some algorithm to generate data

Prepare the data: feature scaling

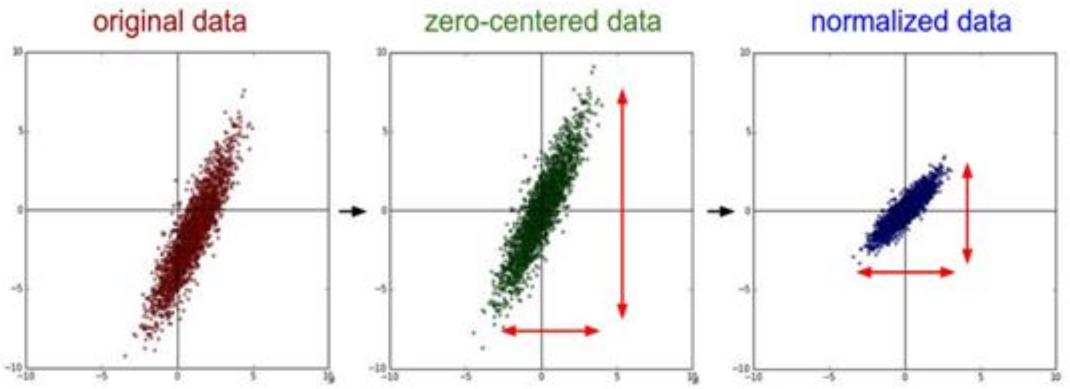
- Scale the numeric features of the training data if the range of values are large.

- Feature scaling will transform all of the feature values to within the same range.

- Some algorithms like regression perform much better with scaled data.
  - In the housing data example, the median income has a range of values from 0 -15 while the number of rooms range is 6 – 39,320. An algorithm like regression will end up ignoring the median income and focusing more on the number of rooms.

When it comes time to scale the test data, be sure to scale it using the same scale as the training data.

Feature scaling



- There are two types of feature scaling:
  - **min-max scaling (normalization) -** use when features have different scales. New values will be between 0 -1 (or -1 and 1).

    x_new = (x – x_min) / (x_max – x_min)

  - **standardization/z-score normalization –** use when features have normal distributions and have similar ranges. Creates a mean of 0 and standard deviation of 1. Does not scale to a given range.

    x_new = (x – mean) / standard deviation

If the feature distribution is not normal, convert the feature values first before scaling. For example, if the feature is right skewed, compute the log or square root of the feature before scaling.
If you scale the ***output/target/dependent variable,*** make sure to compute the inverse when presenting results.

Prepare the data: Handling text and categorical features

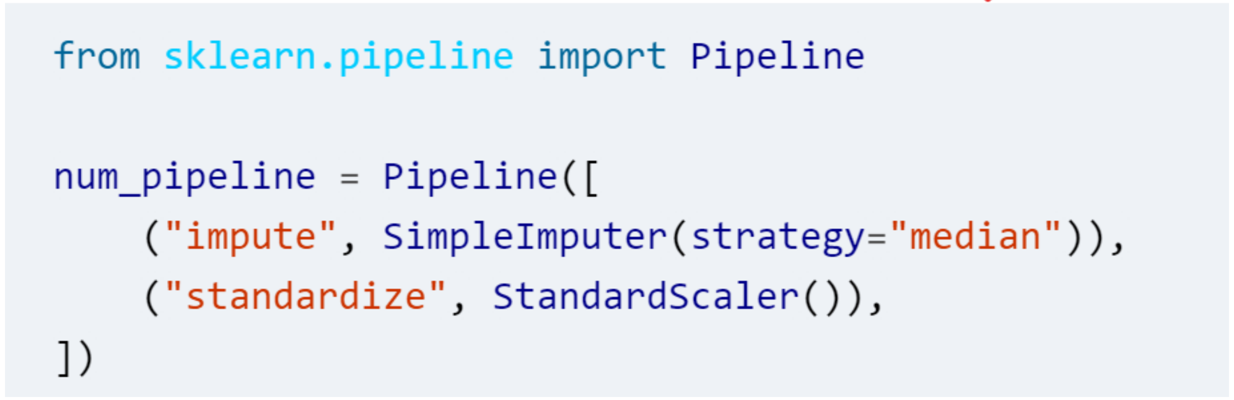
Features often come as text categories
Convert the text to numeric features

Integer encoding – convert each text attribute to a number.
Example poor=0, fair=1, good=2, very good=3, excellent=4
Ordinal encoding works well if the categories have a natural order like above.

One hot encoding – create a binary feature per category.
Example - create 3 new features – sunny, cloudy, hazy and set to either 0 or 1 based on the sample value.
sunny=0, cloudy = 1, hazy = 0 -> [0, 1, 0]
Each sample will have only one of the above features set to 1.
If the categories do not have a natural order like sunny, cloudy, hazy use one hot encoding.

Transformation pipelines

- SciKit-Learn provides a Pipeline class that facilitates a sequence of data transformations.



```python
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([
    ("impute", SimpleImputer(strategy="median")),
    ("standardize", StandardScaler()),
])
```

In the code example above, first any numeric missing values will be filled in with the median. Next the numeric features will be scaled.

Select and train models

Select a few models and run them with your prepared data.

Select one as a base line and compare the other models to it.

Select a performance measure and report accuracy results.

Look for ***underfitting*** and ***overfitting*** of the training data.

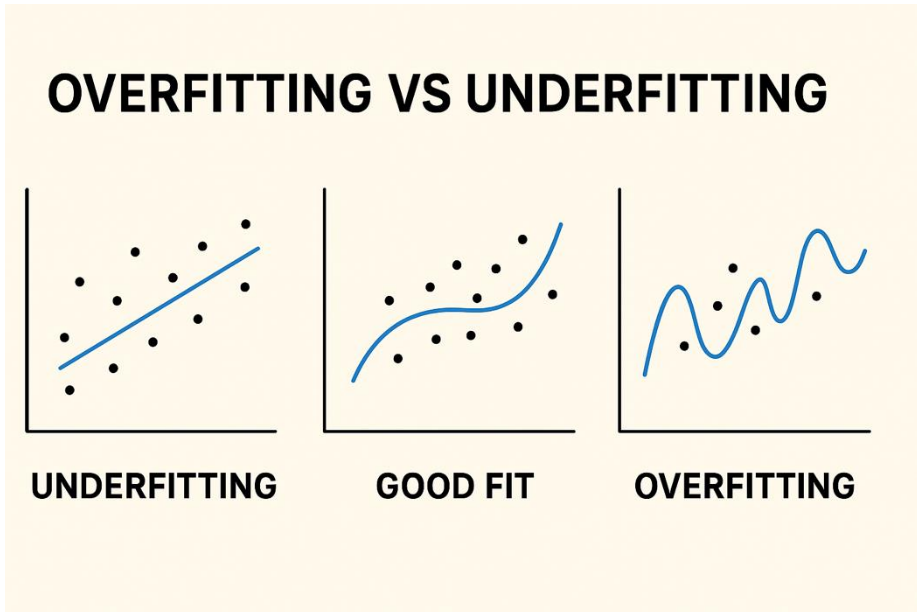
Model Performance

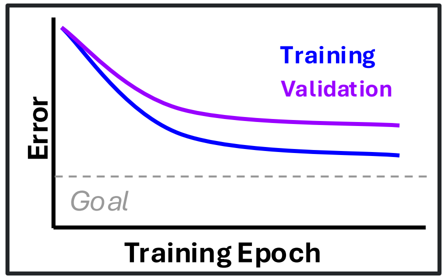
If the performance metric is too large (poor accuracy), the model is ***undefitting*** the training data. There is not enough information in the features.

- Ways to mitigate: Make model bigger, add features, tune hyper parameters, reduce regularization

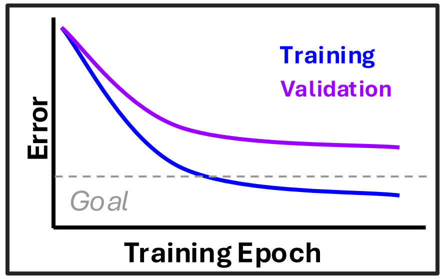
If the performance metric is too small (superb accuracy), the model is ***overfitting*** the training data and may not ***generalize*** well when it comes time to make predictions.

- Way to mitigate: Collect more data, normalization, reduce model size/complexity, add regularization

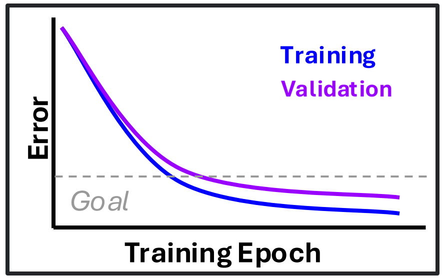




Underfitting



Overfitting



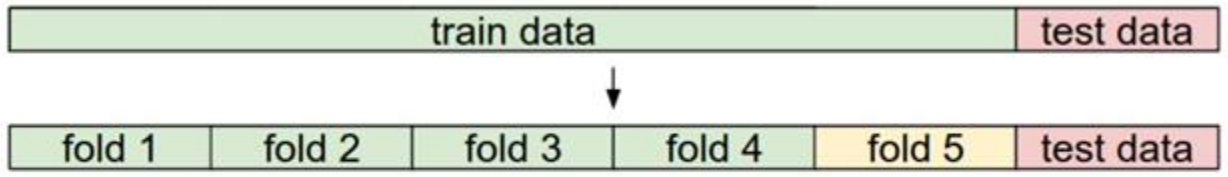
Good Fit

Cross validation

If your model is ***overfitting*** the training data, use cross validation to get a more accurate performance measure.

Cross validation splits the training set into **k-folds** of a data and a validation set and will run the model k times using different training samples each time.

For example, 80% of the original data set is set aside as training set, Cross validation will then take 90% of that to train a model and test it on 10% of the data. It will do that k times using different samples each time. This way the model is not trained on the entire training set, which can overfit the training data. Cross validation is a more accurate way to gauge the performance of a model.



Common data splits. A training and test set is given. The training set is split into folds (for example 5 folds here). The folds 1-4 become the training set. One fold (e.g. fold 5 here in yellow) is denoted as the Validation fold and is used to tune the hyperparameters. Cross-validation goes a step further and iterates over the choice of which fold is the validation fold, separately from 1-5. This would be referred to as 5-fold cross-validation. In the very end once the model is trained and all the best hyperparameters were determined, the model is evaluated a single time on the test data (red).

Evaluate on test set

All of the data preparation you did on the training set you must do on the testing set. i.e. take care of missing values, prepare categorical features with ordinal or one hot encoding, scale the data etc.

Run the test data through the algorithms and report results.

If the error is greater than the error in the training and cross validation sets, resist the temptation to tune the model to do well on the test set. It will like perform worse on new data.

Monitoring in Production

- **Data and Concept Drift**
  - Phenomenon where the statistical properties of the data change over time
  - Changes in the relationship between the input features and the target variable
- **Fairness & Trust**
  - Bias across subgroups
- **Performance & Reliability**
  - Latency, throughput, error rates
  - System health (CPU/GPU, memory, cost)

Monitoring in production: Epic example

An Algorithm That Predicts Deadly Infections Is Often Flawed

A study found that a system used to identify cases of sepsis missed most instances and frequently issued false alarms.

Widely used sepsis prediction tool is less effective than Michigan doctors thought

June 29, 2021

Sepsis is a severe inflammatory response to infection and affects 1.7 million U.S. adults each year. Common symptoms may include fever, a high heart rate, and a low white blood cell count, which can resemble other conditions. To help identify patients at risk of sepsis early, many doctors use prediction models embedded into a patient's electronic health record. If a patient shows signs of the inflammatory condition, their medical team receives an alert. However, a study in *JAMA Internal Medicine* found the Epic Sepsis Model (EMS), a common prediction tool, wasn't as effective at detecting sepsis as doctors at Michigan Medicine, part of the University of Michigan, originally thought.

To assess the effectiveness of EMS, researchers defined sepsis by using diagnostic criteria from the CDC and Medicare. Their definition varied from the developer's, which was based on sepsis-related treatment and care. After evaluating the medical records of 27,697 patients during a 10-month period in 2018-2019, which connected to 38,455 hospitalizations, the researchers found 2,552 hospitalizations, 7%, were due to sepsis. The EMS prediction missed 1,709 patients, 67%, who had sepsis even though the algorithm sent out alerts for 18% of hospitalized patients. Overall, the Michigan research team found the EMS was 63% effective at detecting sepsis early compared to the developer's predictions of 77-83%.

Generalization Failure: Looked good in development -> poor performance in real-world hospitals.

Data Leakage: Using proxy signals (like billing codes or physician actions) -> made the model look better than it really was.

Refresher on Linear Regression

- Goal: Predict an outcome **Y** from an input **X**
- Example: Predicting house prices from square footage
- Key idea: Fit a straight line that best represents the relationship

Regression Line

- Equation : $Y = \beta_0 + \beta_1 X$

- β$_0$: intercept → value of Y when X = 0
- β$_1$: slope → change in Y for each unit increase in X

- How to learn those parameters?

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

$\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means.

Measuring Accuracy

- Predictions are not perfect → there are errors (residuals)
- Accuracy is assessed by how small these residuals are
- Common metric: Root Mean Square Error (RMSE)= Residual Standard Error RSE
- Lower RMSE = better fit

$$\mathrm{RSE} = \sqrt{\frac{1}{n-2}\mathrm{RSS}} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

$$\mathrm{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

- Instead of solving equations directly, can we **search iteratively** for the best parameters?

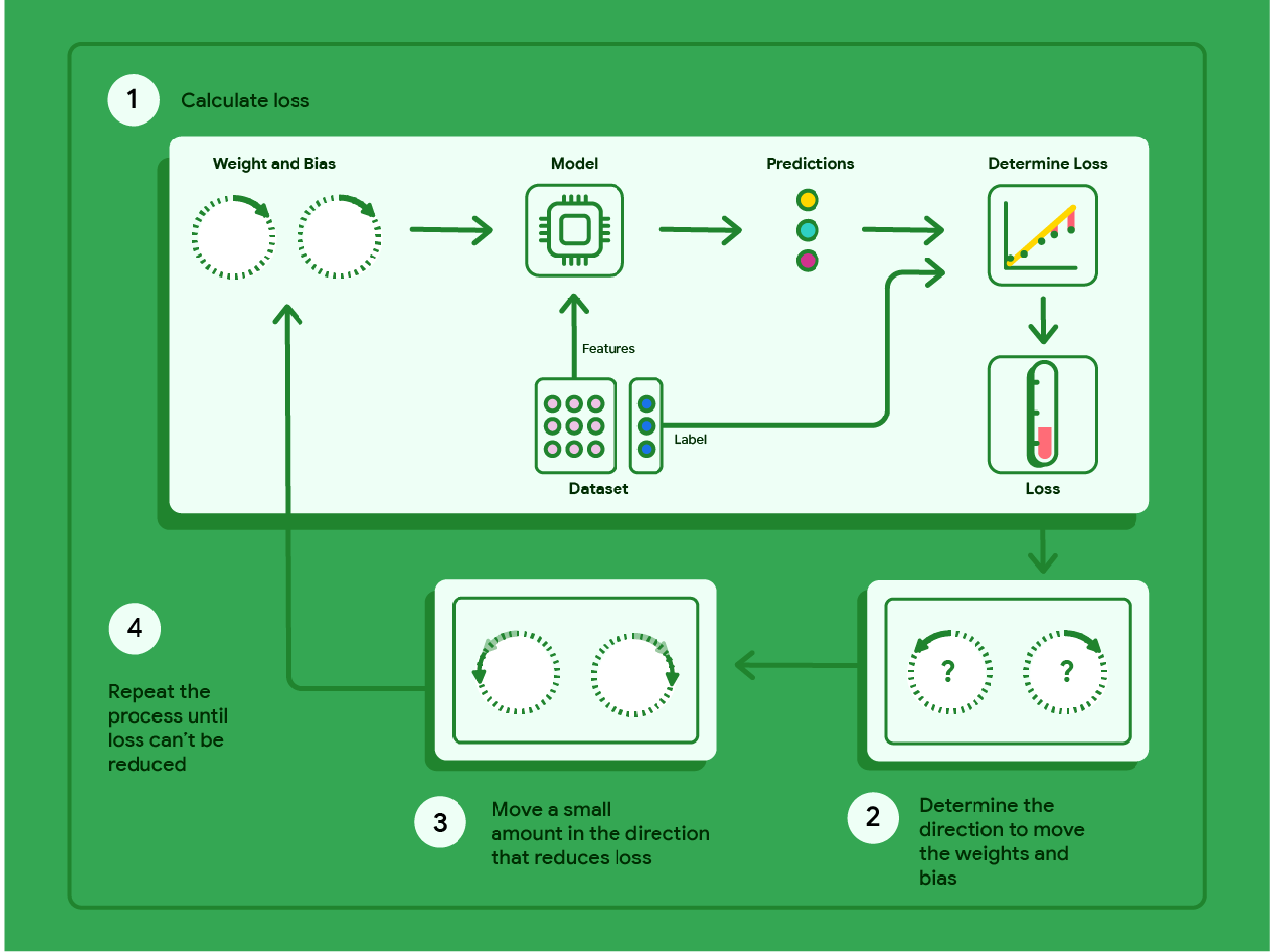
- This is where **gradient descent** comes in — a universal optimization method not just for linear regression, but also for logistic regression, neural networks, and many ML models.

Gradient Descent

- Finding the coefficients for the line of best fit using a closed form equation works well for single variable regression.

- When we have more than one feature, there is a closed form equation called the normal equation, but it is slow to compute the coefficients.

- Gradient descent is an algorithm used to find the coefficients for multiple linear regression and many other machine learning models like logistic regression and neural networks.


1
Calculate loss
Weight and Bias
Model
Predictions
Determine Loss
Features
Label
Dataset
Loss
4
Repeat the process until loss can't be reduced
?
?
3
Move a small amount in the direction that reduces loss
2
Determine the direction to move the weights and bias