ML Model Training

6/15/2023

Some ML Vocabularies

- **Label:** is the thing we're predicting

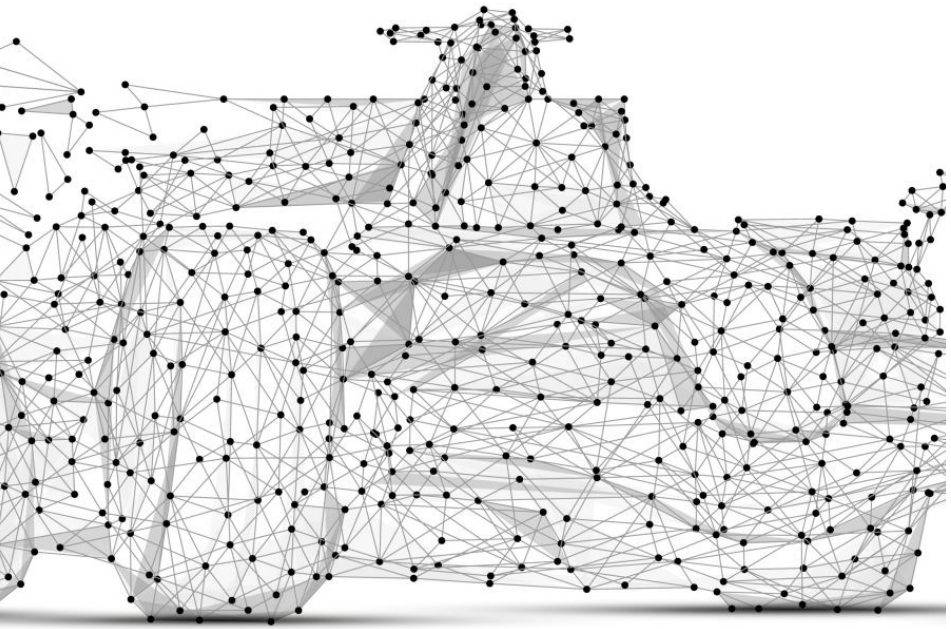
housingMedianAge (feature)	totalRooms (feature)	totalBedrooms (feature)	medianHouseValue (label)
15	5612	1283	66900
19	7650	1901	80100
17	720	174	85700
14	1501	337	73400
20	1454	326	65500

Labeled dataset

housingMedianAge (feature)	totalRooms (feature)	totalBedrooms (feature)
42	1686	361
34	1226	180
33	1077	271

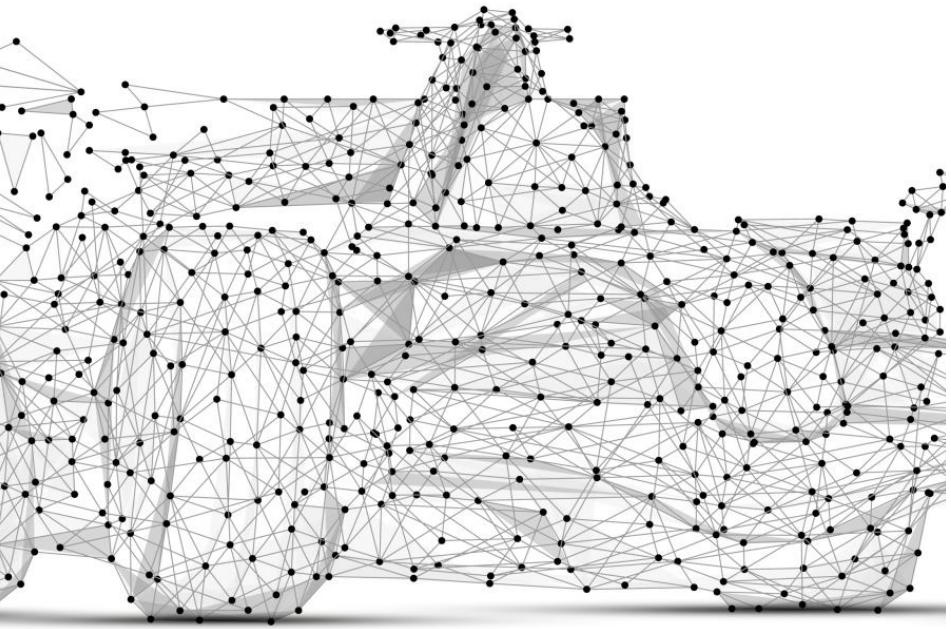
Unlabeled dataset

Some ML Vocabularies



- **Model** defines the relationship between features and label. For example, a spam detection model might associate certain features strongly with "spam".

- **Training** means creating or **learning** the model. That is, you show the model labeled examples and enable the model to gradually learn the relationships between features and label.



Some ML Vocabularies

- **Feature**: a feature is data that's used as the input for ML models to make predictions
- **Target:** a target is whatever the output of the input variables.

Machine Learning Libraries

- An ML library, is a collection of software tools and frameworks that provide developers with pre-built functions, algorithms, and utilities to facilitate the development and deployment of machine learning models
- ML libraries typically offer a range of functionalities, including
  - data preprocessing
  - feature extraction
  - model training
  - model evaluation
  - model deployment
- Reduce time for programming ☺

Machine Learning Libraries

- They provide implementations of various machine learning algorithms, such as
  - linear regression
  - decision trees
  - support vector machines
  - neural networks, and more

These libraries often include optimization algorithms, dimensionality reduction techniques, cross-validation methods, and other tools to aid in the machine learning workflow.

Some popular machine learning libraries

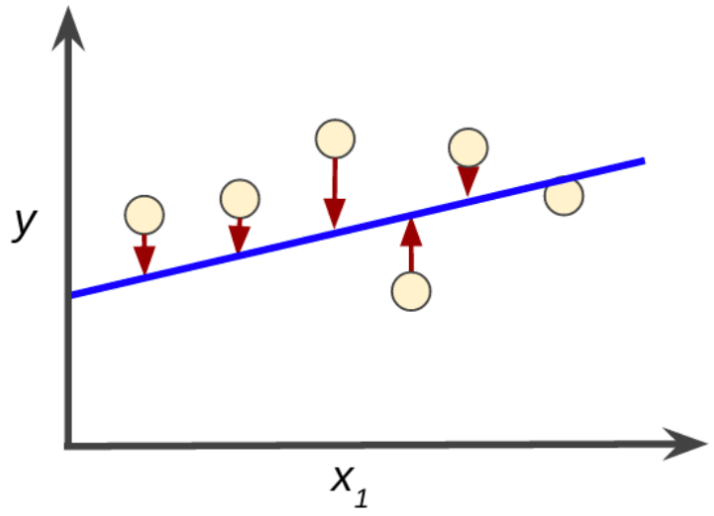
- **Scikit-learn**:
  - A widely used machine learning library in Python that provides a comprehensive set of tools for data preprocessing, feature selection, model training, and model evaluation. It supports various algorithms and has a user-friendly API.
- TensorFlow:
  - An open-source machine learning library developed by Google. TensorFlow offers a flexible framework for building and deploying machine learning models, particularly neural networks.

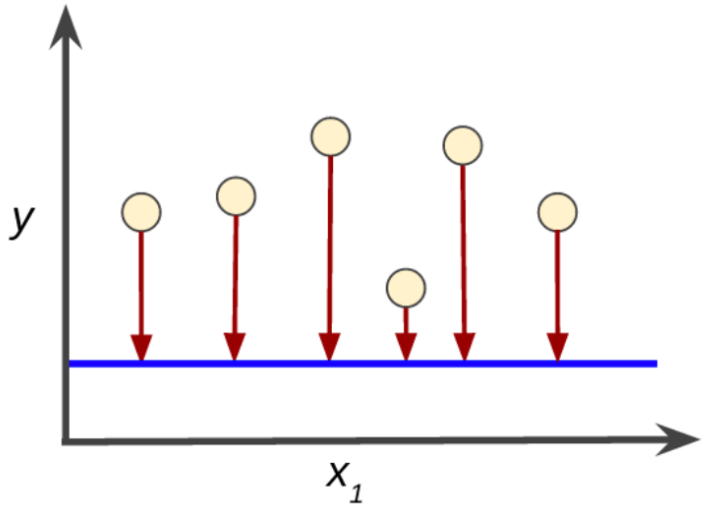
Some popular machine learning libraries

- PyTorch:
  - Another popular open-source machine learning library, primarily focused on deep learning. PyTorch provides a dynamic computational graph, making it easy to build and train neural networks. It is known for its simplicity and intuitive API.
- Keras:
- Initially built as a user-friendly interface for building neural networks, Keras is now integrated into TensorFlow and provides a high-level API. It simplifies the process of building and training deep learning models.

Training and Loss

- Loss is the penalty for a bad prediction
- If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater.





Squared loss: a popular loss function

- **squared loss** also known as **L_2 loss**

- **L_2 loss** = the square of the difference between the label and the prediction = (observation - prediction(**x**))^2 = (y - y')^2

Mean square error (MSE)

- **Mean square error** (**MSE**) is the average squared loss per example over the whole dataset.

where:

• (x,y) is an example in which
  - x is the set of features (for example, chirps/minute, age, gender) that the model uses to make predictions.
  - y is the example's label (for example, temperature).

• prediction(x) is a function of the weights and bias in combination with the set of features .

•D is a data set containing many labeled examples, which are pairs.

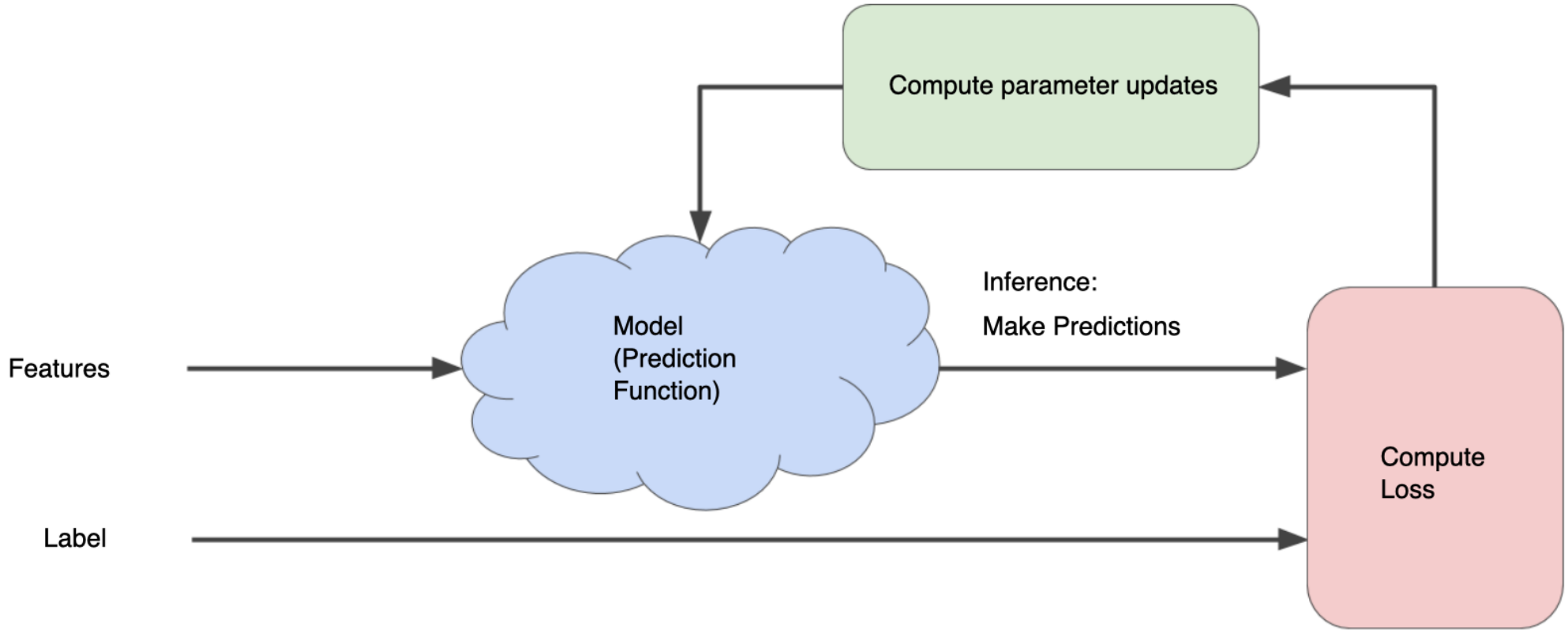
• N is the number of examples in .

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - prediction(x))^2$$

Reducing Loss

- Iterative learning
- "Hot and Cold" kid's game for finding a hidden object like a thimble.
- In this game, the "hidden object" is the best possible model. You'll start with a wild guess ("The value of is 0.") and wait for the system to tell you what the loss is. Then, you'll try another guess ("The value of is 0.5.") and see what the loss is. Aah, you're getting warmer. Actually, if you play this game right, you'll usually be getting warmer. The real trick to the game is trying to find the best possible model as efficiently as possible.

iterative trial-and-error process



Getting direction!

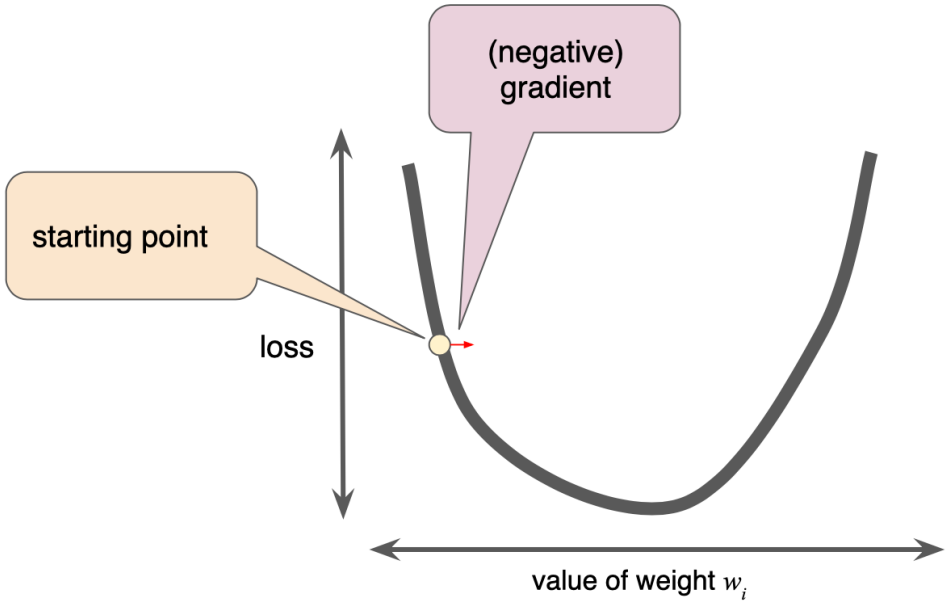
Credit for image

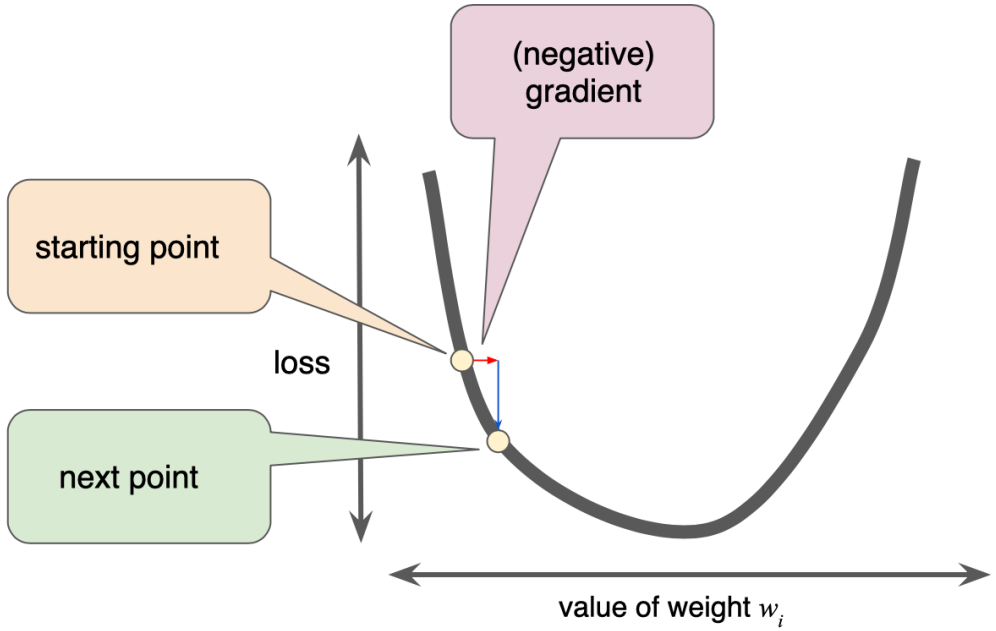
Getting direction- **Gradient Descent Algorithm**

- **How do we reduce loss? Gradient Descent Algorithm**
- One way to get a direction is to compute the gradient.

- Hyperparameters are the configuration settings used to tune how the model is trained.
- Derivative of $(y - y')^2$ with respect to the weights and biases tells us how loss changes for a given example
  - Simple to compute and convex
- So we repeatedly take small steps in the direction that minimizes loss
  - We call these **Gradient Steps** (But they're really negative Gradient Steps)
  - This strategy is called **Gradient Descent**

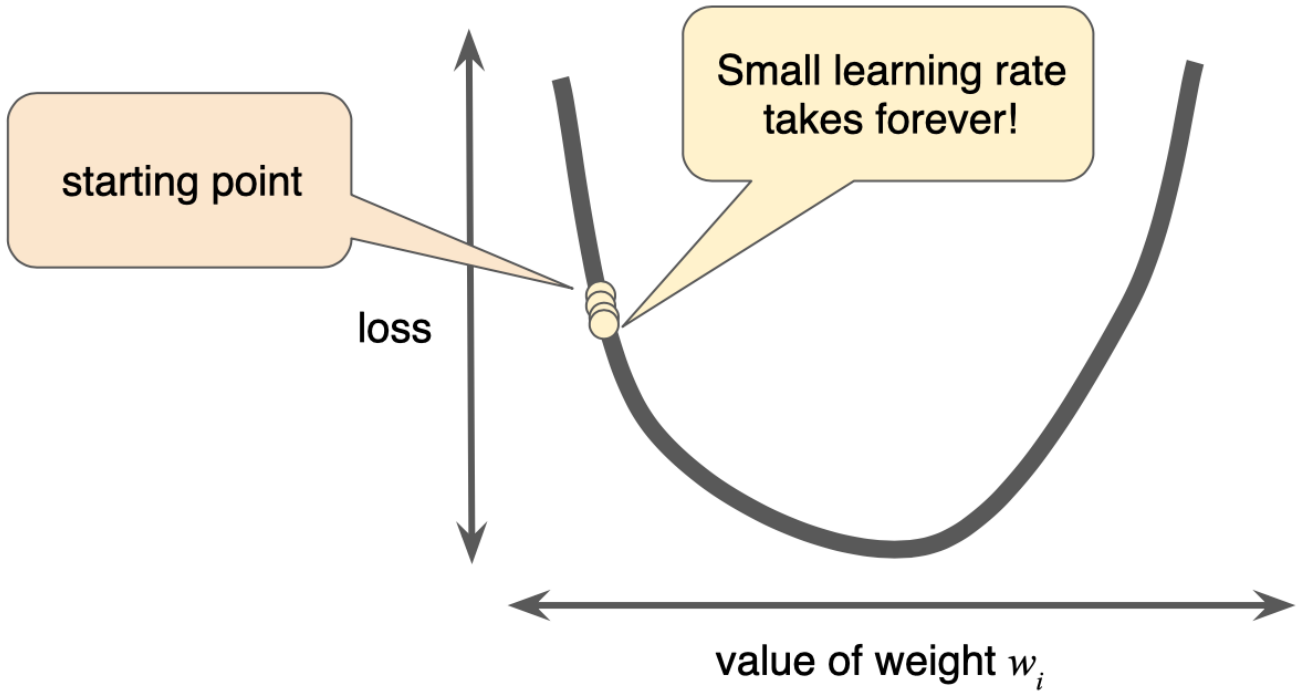
Getting direction- **Gradient Descent Algorithm**





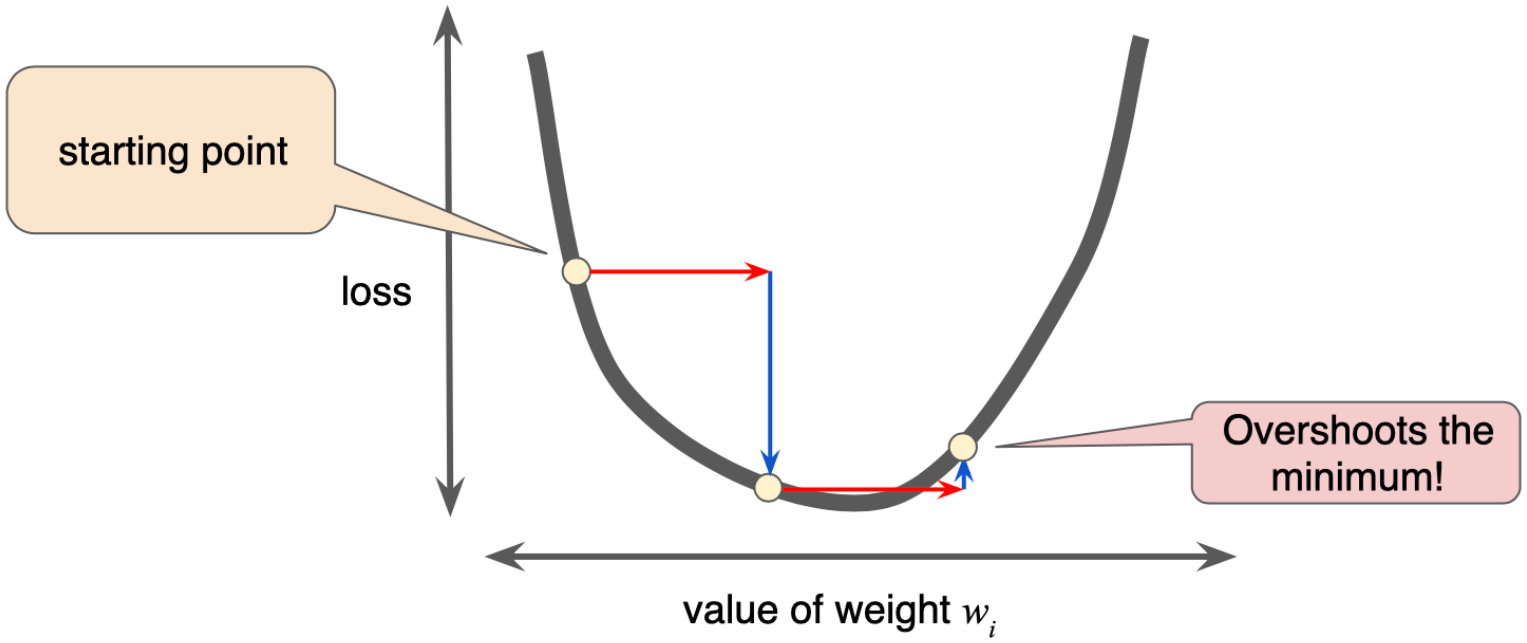
Credit for image

Reducing Loss: Learning Rate



Credit for image

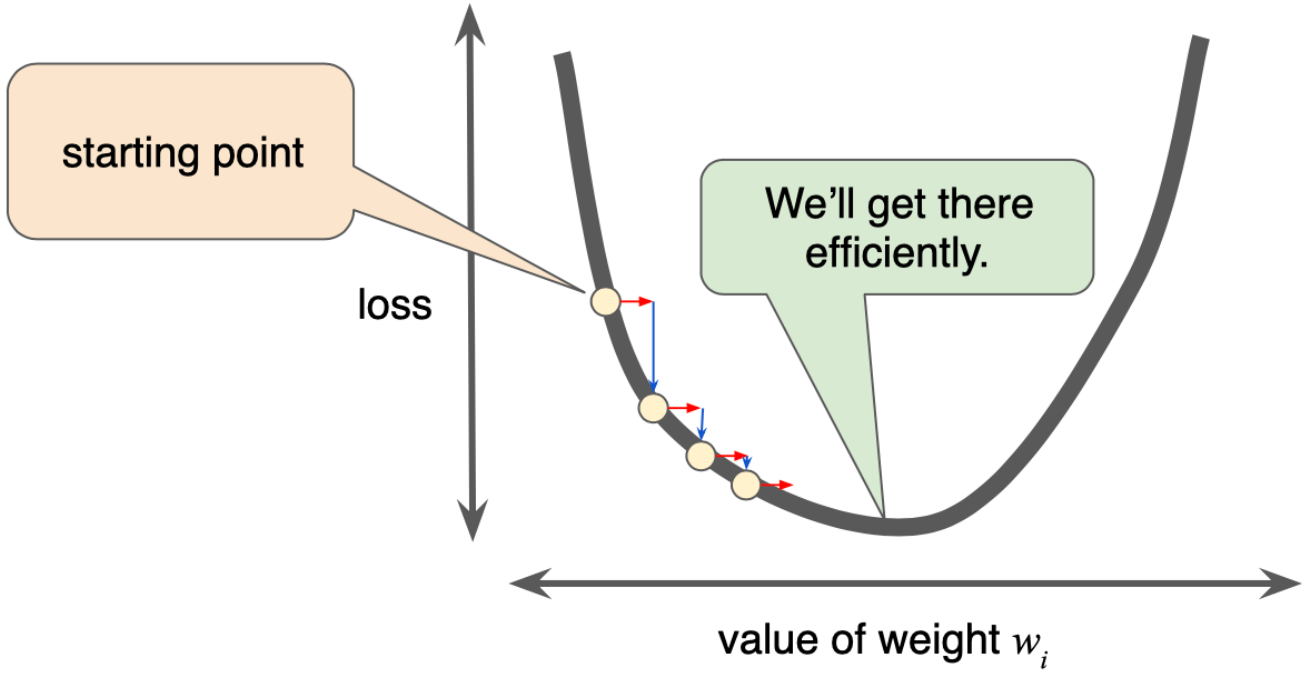
Reducing Loss: Learning Rate



Credit for image

Reducing Loss: Learning Rate

There's a Goldilocks learning rate for every regression problem. The Goldilocks value is related to how flat the loss function is. If you know the gradient of the loss function is small then you can safely try a larger learning rate, which compensates for the small gradient and results in a larger step size.



Credit for image

Learning Rate – where to start?

•some problems are convex, meaning that they're shaped like a giant bowl.
•if we start somewhere on the bowl and we take reasonable step sizes and follow the gradients, eventually we'll find our way to the bottom of the bowl.

•For convex problems, weights can start anywhere (say, all 0s)

- Convex: think of a bowl shape
- Just one minimum

Learning Rate – where to start?

•Many machine learning problems are not convex. Neural networks are notoriously not convex, meaning that rather than being shaped like a bowl, they are shaped more like an egg crate. Where there are many possible minimum values, some of which are better than others. So the initialization does matter.

Evaluating Classification Performance

- How do we evaluate classification models?
  - One possible measure: Accuracy
    - the fraction of predictions we got right
  - In many cases, accuracy is a poor or misleading metric
    - Most often when different kinds of mistakes have different costs
    - Typical case includes *class imbalance*, when positives or negatives are extremely rare

True Positives and False Positives

- For class-imbalanced problems, useful to separate out different kinds of errors

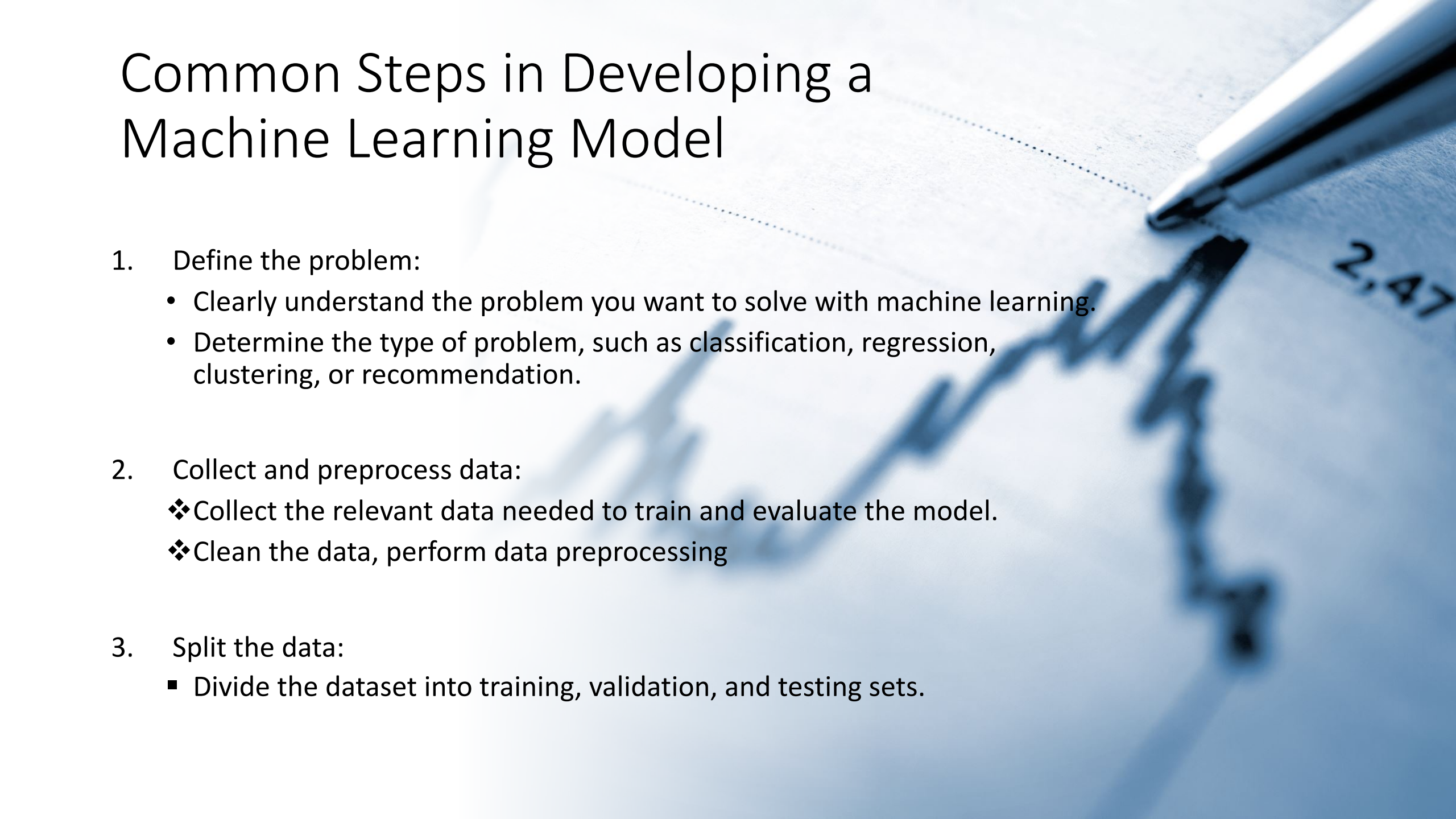
True Positives We correctly called wolf! We saved the town.	**False Positives** Error: we called wolf falsely. Everyone is mad at us.
False Negatives There was a wolf, but we didn't spot it. It ate all our chickens.	**True Negatives** No wolf, no alarm. Everyone is fine.

Evaluation Metrics: Precision and Recall

So we can combine these ideas into a couple of different metrics.

- **Precision:** (True Positives) / (All Positive Predictions)
  - When model said "positive" class, was it right?
  - Intuition: Did the model cry "wolf" too often?
- **Recall**: (True Positives) / (All Actual Positives)
  - Out of all the possible positives, how many did the model correctly identify?
  - Intuition: Did it miss any wolves?

Common Steps in Developing a Machine Learning Model



1. Define the problem:
   - Clearly understand the problem you want to solve with machine learning.
   - Determine the type of problem, such as classification, regression, clustering, or recommendation.

2. Collect and preprocess data:
   - ❖Collect the relevant data needed to train and evaluate the model.
   - ❖Clean the data, perform data preprocessing

3. Split the data:
   - Divide the dataset into training, validation, and testing sets.

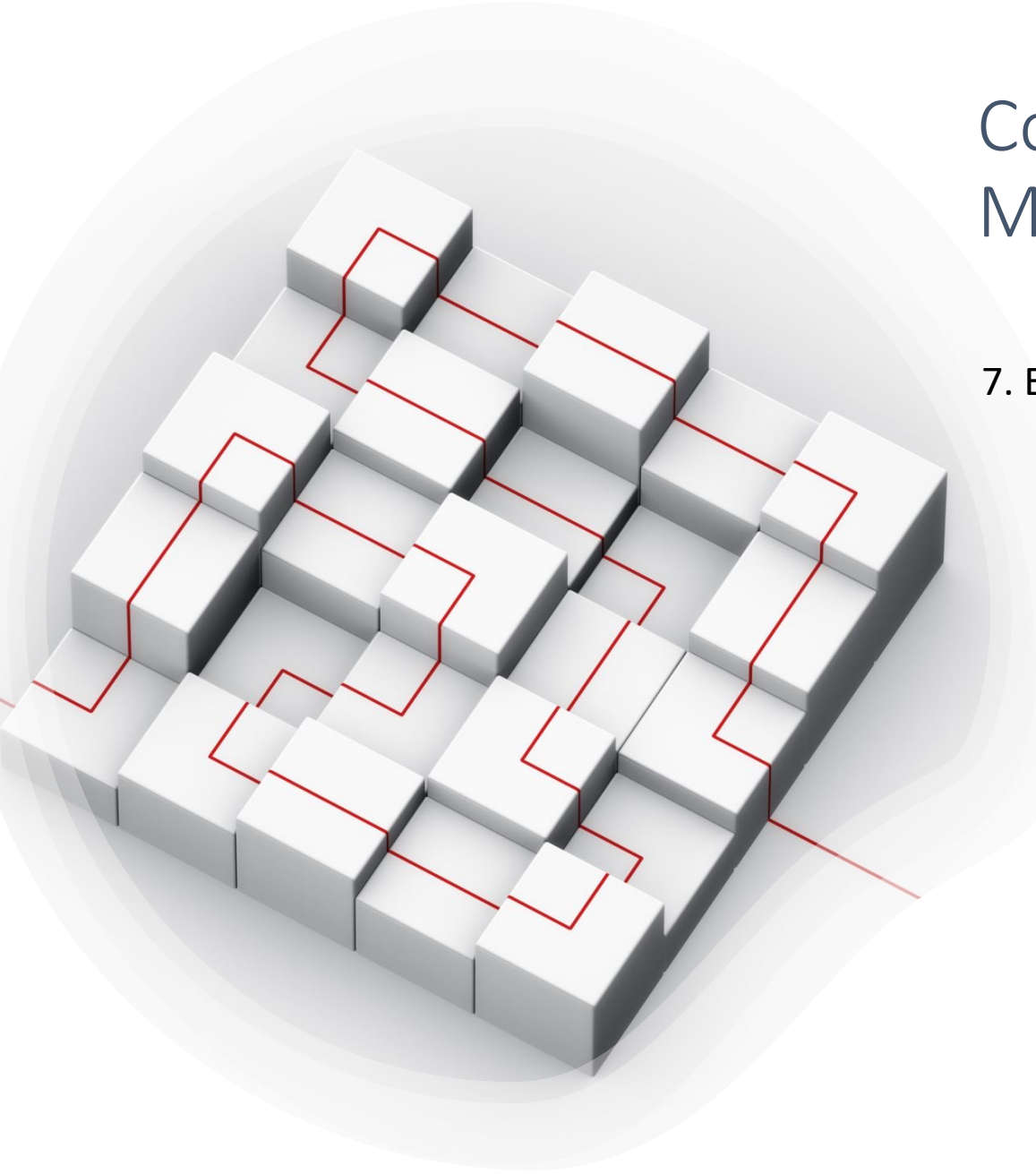
Common Steps in Developing a Machine Learning Model

4. Select a model:
   - Choose an appropriate machine learning algorithm based on the problem type, available data, and desired outcome.

5. Prepare the features:
   - ❖ Identify the relevant features (input variables) from the dataset.
   - ❖ Perform feature selection. This may involve transforming variables, creating new features, or selecting subsets of features.

Common Steps in Developing a Machine Learning Model

6. Train the model:
   - Feed the training data into the chosen machine learning algorithm.
   - The model learns the underlying patterns and relationships in the data and adjusts its internal parameters to minimize the prediction error.
   - The training process typically involves an optimization algorithm that iteratively updates the model's parameters based on the training data.



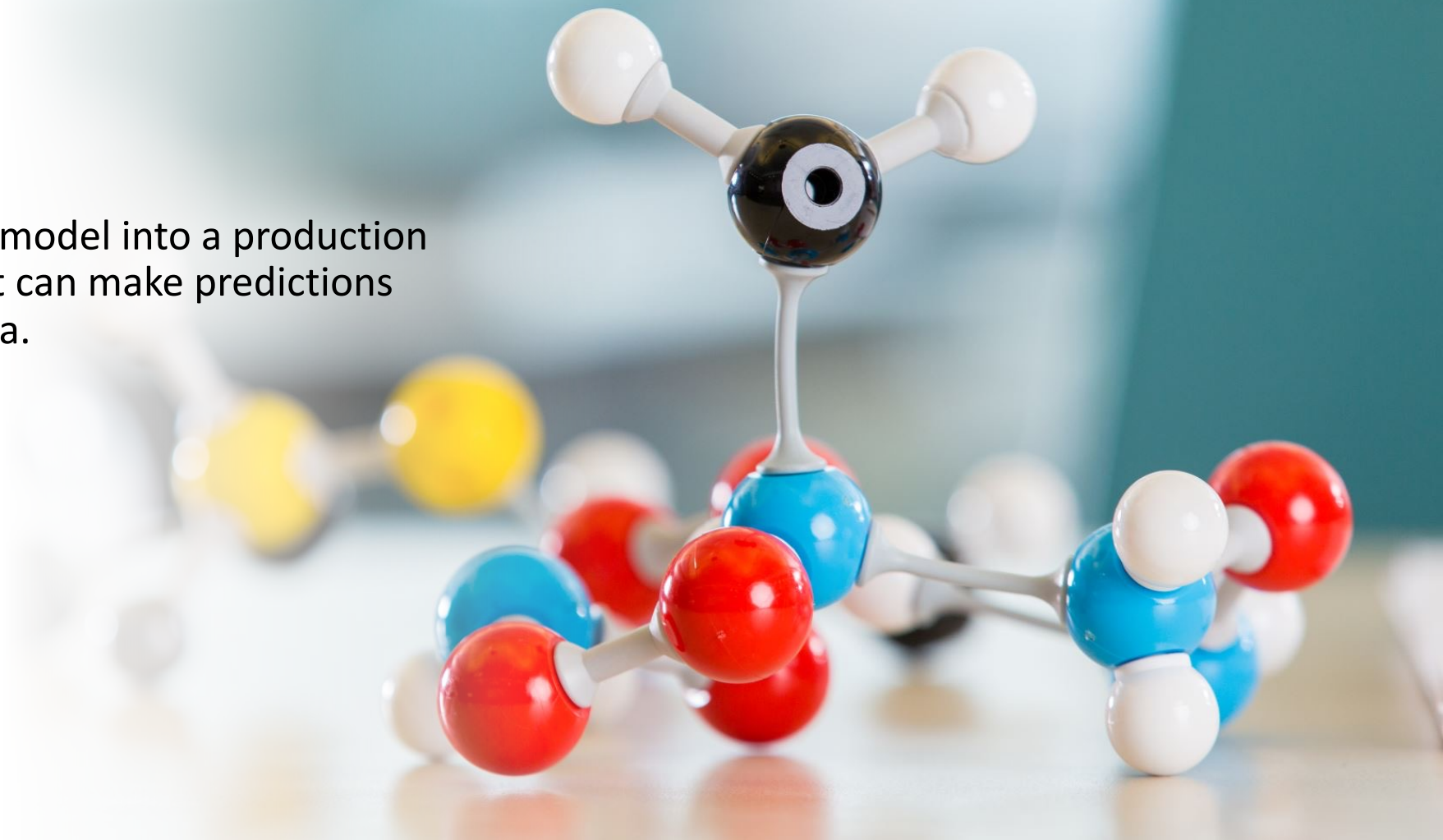
Common Steps in Developing a Machine Learning Model

7. Evaluate and validate the model:
    - Evaluation Metrics:
        - For classification, metrics like accuracy, precision, recall, and F1 score can be used.
        - For regression, metrics like mean squared error (MSE) or R-squared are commonly used.
    - Cross-Validation.
    - Confusion Matrix
    - ROC Curve and AUC-ROC
    - Train/Test Split:
        - Split the available data into a training set and a separate test set. Train the model on the training set and evaluate its performance on the test set. This gives an estimate of how well the model will perform on unseen data.

Common Steps in Developing a Machine Learning Model

10. Deploy the model:

- Integrate the trained model into a production environment where it can make predictions on new, real-time data.



Supervised Learning Models

- Regression
- Classification

Model training and evaluating

- Load Libraries and Data
- Create Feature Set
- Split the Data
- Initialize the Model
- Fit the Model
- Evaluate the Model

Linear Regression

```
# Load libraries
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston

# Load data with only two features
boston = load_boston()
features = boston.data[:,0:2]
target = boston.target

# Create linear regression
regression = LinearRegression()

# Fit the linear regression
model = regression.fit(features, target)

# Cross-validate the linear regression using R-squared
cross_val_score(regression, features, target, scoring='r2')
```

Classification

```
# Load libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn import datasets

# Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

# Create decision tree classifier object
decisiontree = DecisionTreeClassifier(random_state=0)

# Train model
model = decisiontree.fit(features, target)
# Cross-validate model using accuracy
cross_val_score(logit, X, y, scoring="accuracy")
```