SLAFES XXV
Barranquilla 2024

60 ICTP 1964–2024

Machine Learning

With scikit-learn

Luis Alejandro Torres
Administrador HPC – SC3UIS
Universidad Industrial de Santander

Javier Montoya
Profesor de Física
Universidad de Cartagena

SUPERVISED LEARNING

What is machine learning?

- Machine learning is the process whereby:
  - Computers are given the ability to learn to make decisions from data
  - without being explicitly programmed!

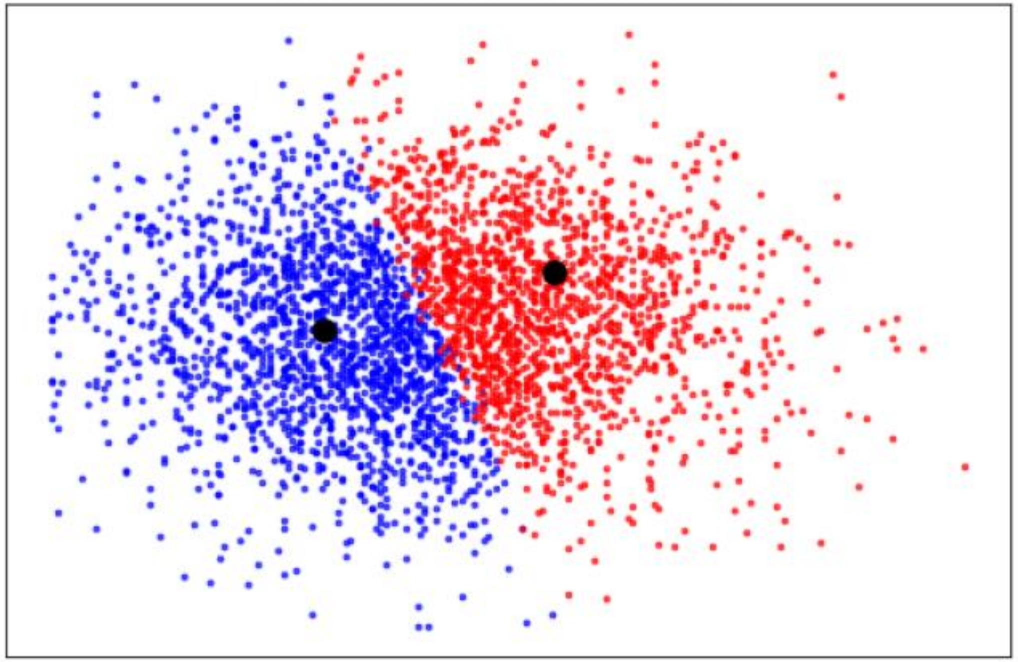
Examples

- Spam
- Books classificcation

Unsupervised learning

- Uncovering hidden patterns from unlabeled data
  - Example:
    - Grouping customers into distinct categories (Clustering)

Cluster Analysis for Customer Churn



A business may wish to group its customers into distinct categories based on their purchasing behavior without knowing in advance what these categories are.

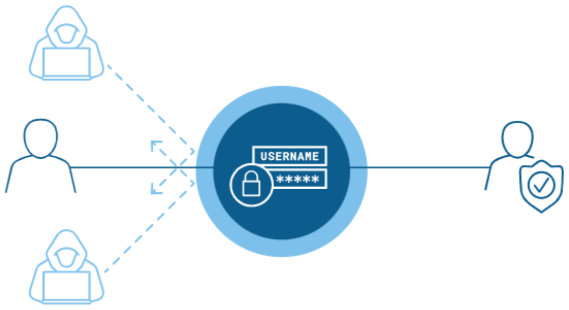
Supervised learning

- The predicted values are known
- **Aim**: Predict the target values of unseen data, given the features

	Features					Target variable
	points_per_game	assists_per_game	rebounds_per_game	steals_per_game	blocks_per_game	position
0	26.9	6.6	4.5	1.1	0.4	Point Guard
1	13	1.7	4	0.4	1.3	Center
2	17.6	2.3	7.9	1.00	0.8	Power Forward
3	22.6	4.5	4.4	1.2	0.4	Shooting Guard

Types of supervised learning

Classification: Target variable consists of categories



- We can predict whether a bank transaction is fraudulent or not. As there are two outcomes here - a fraudulent transaction, or non-fraudulent transaction, this is known as **binary classification**.

Regression: Target variable is continuous

- A model can use features such as number of bedrooms, and the size of a property, to predict the target variable, price of the property.

https://www.f5.com/solutions/stop-online-fraud
https://medium.datadriveninvestor.com/house-prices-linear-regression-eb57efeeb965

Naming conventions

- Feature = predictor variable = independent variable
- Target variable = dependent variable = response variable

	Features					Target variable
	points_per_game	assists_per_game	rebounds_per_game	steals_per_game	blocks_per_game	position
0	26.9	6.6	4.5	1.1	0.4	Point Guard
1	13	1.7	4	0.4	1.3	Center
2	17.6	2.3	7.9	1.00	0.8	Power Forward
3	22.6	4.5	4.4	1.2	0.4	Shooting Guard

Before you use supervised learning

- Requirements:
  - No missing values
  - Data in numeric format
  - Data stored in pandas DataFrame or NumPy array

- **Perform Exploratory Data Analysis (EDA) first**

scikit-learn syntax

```python
from sklearn.module import Model
model = Model()
model.fit(X, y)
predictions = model.predict(X_new)
print(predictions)
```

```
array([0, 0, 0, 0, 1, 0])
```

	Features					Target variable
	points_per_game	assists_per_game	rebounds_per_game	steals_per_game	blocks_per_game	position
0	26.9	6.6	4.5	1.1	0.4	Point Guard
1	13	1.7	4	0.4	1.3	Center
2	17.6	2.3	7.9	1.00	0.8	Power Forward
3	22.6	4.5	4.4	1.2	0.4	Shooting Guard

The classification challenge - Classifying labels of unseen data

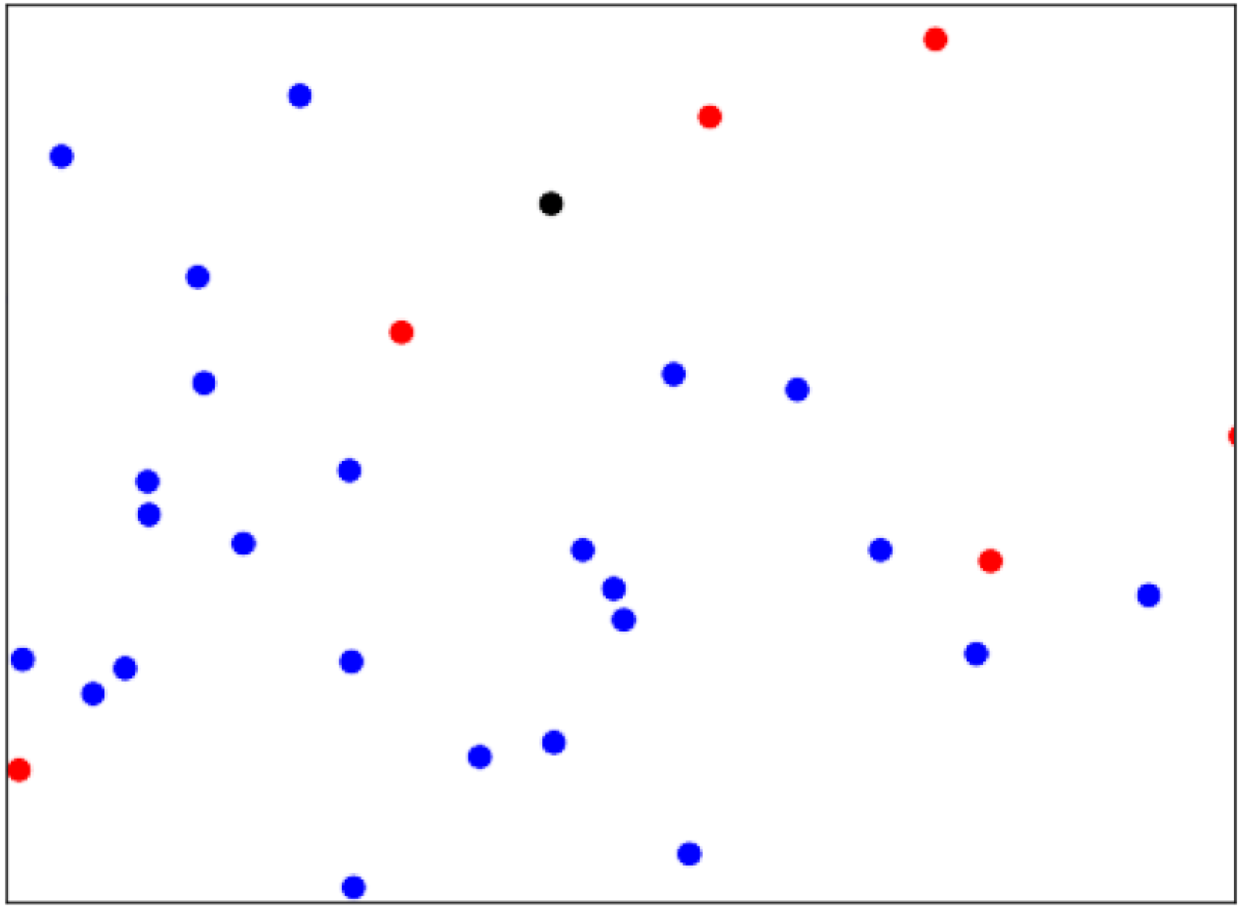
1. Build a model
2. Model learns from the labeled data we pass to it
3. Pass unlabeled data to the model as input
4. Model predicts the labels of the unseen data

Labeled data = training data

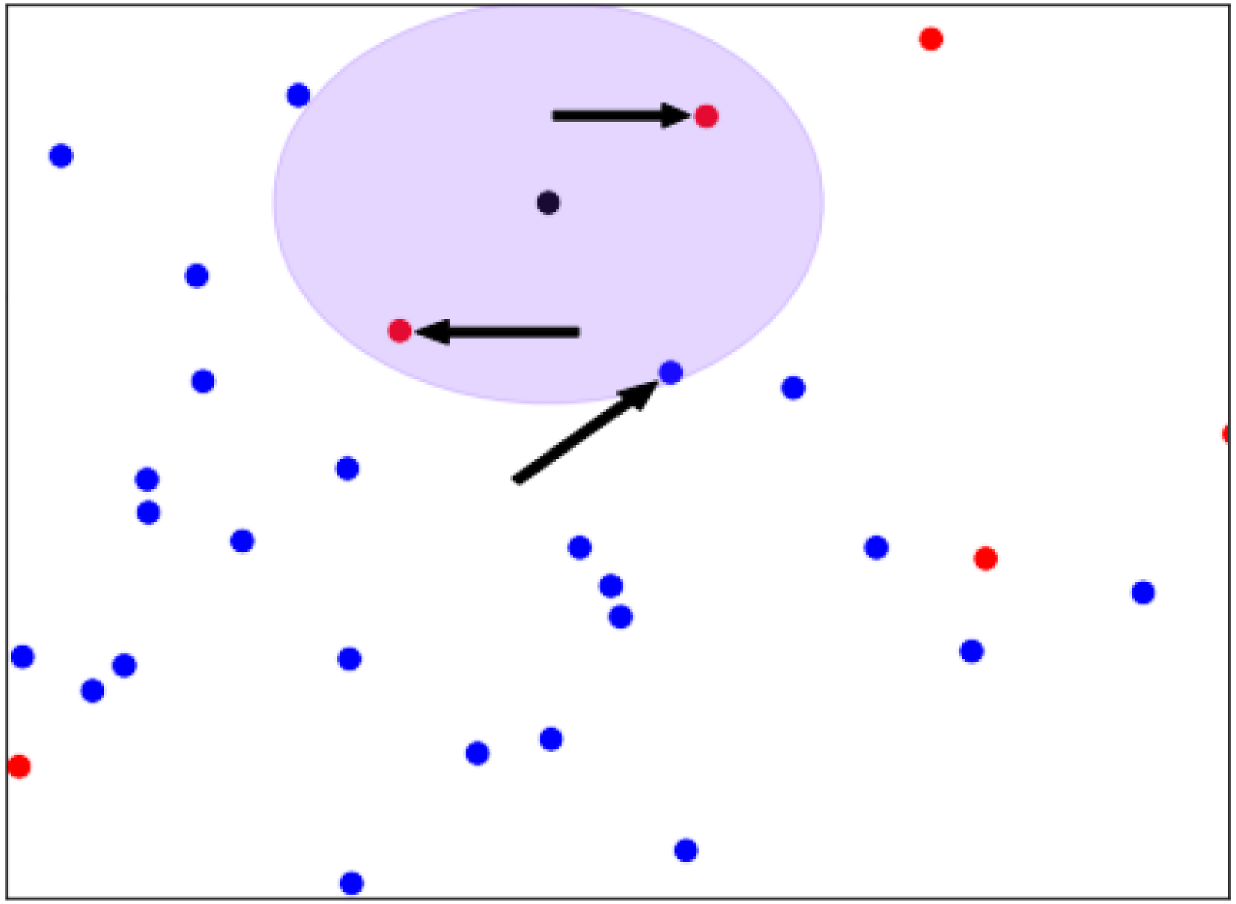
k-Nearest Neighbors - KNN

- Predict the label of a data point by
  - Looking at the k closest labeled data points
  - Taking a majority vote

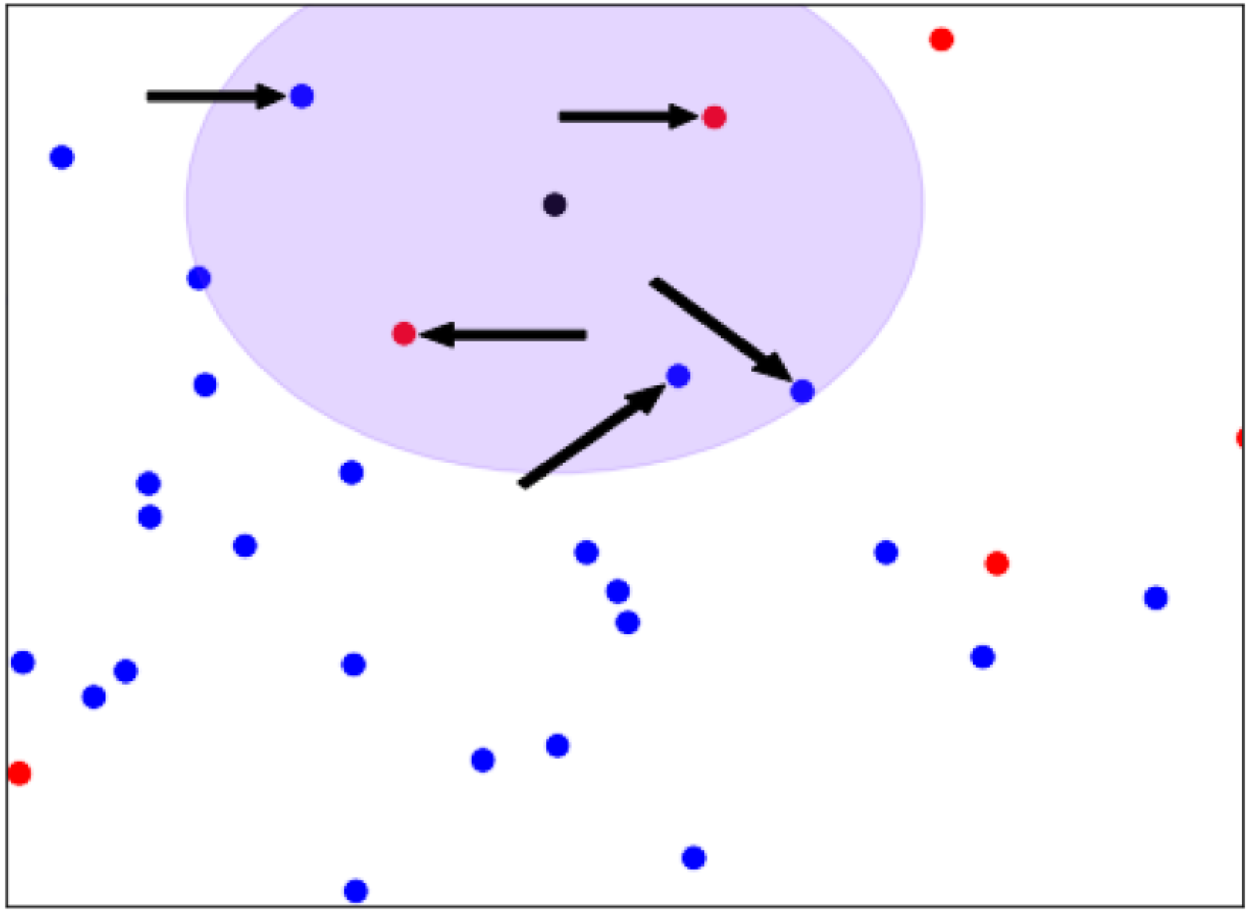
k-Nearest Neighbors - KNN



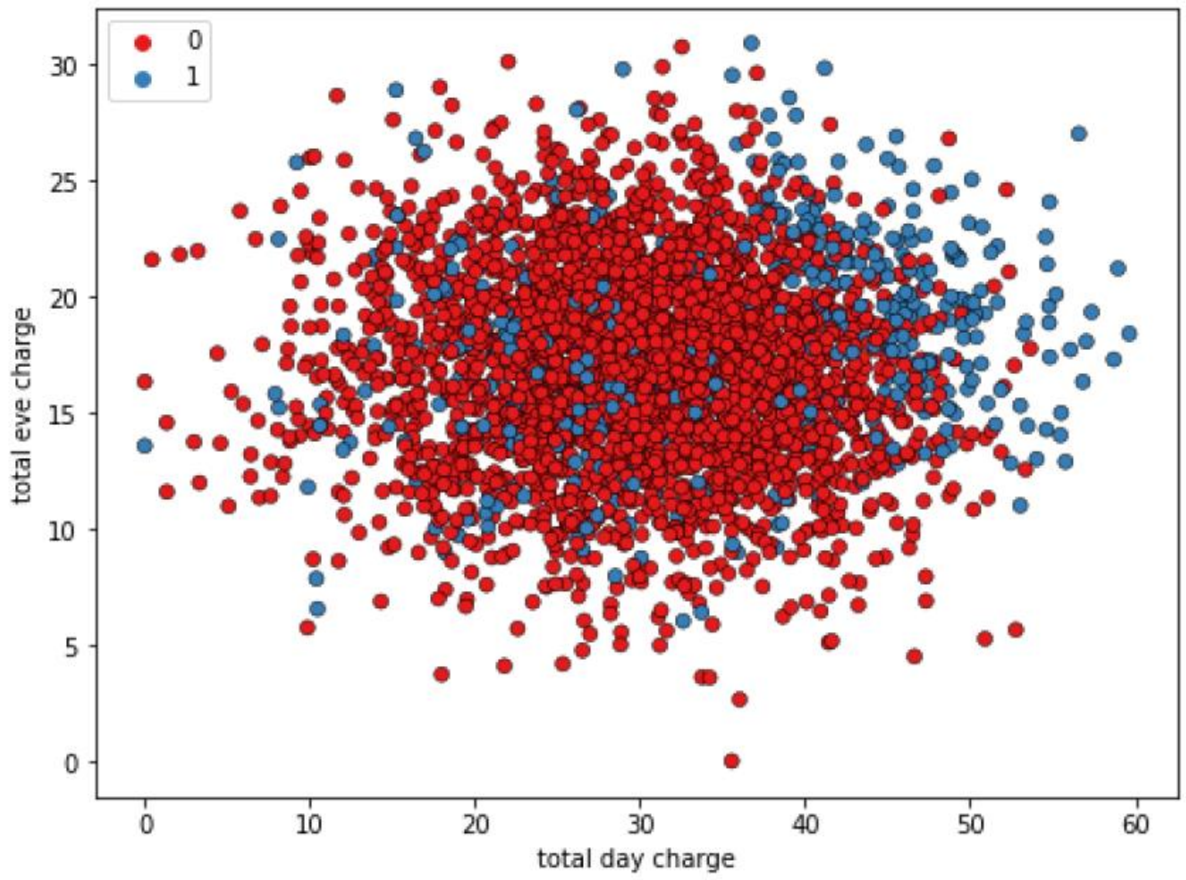
k-Nearest Neighbors - KNN



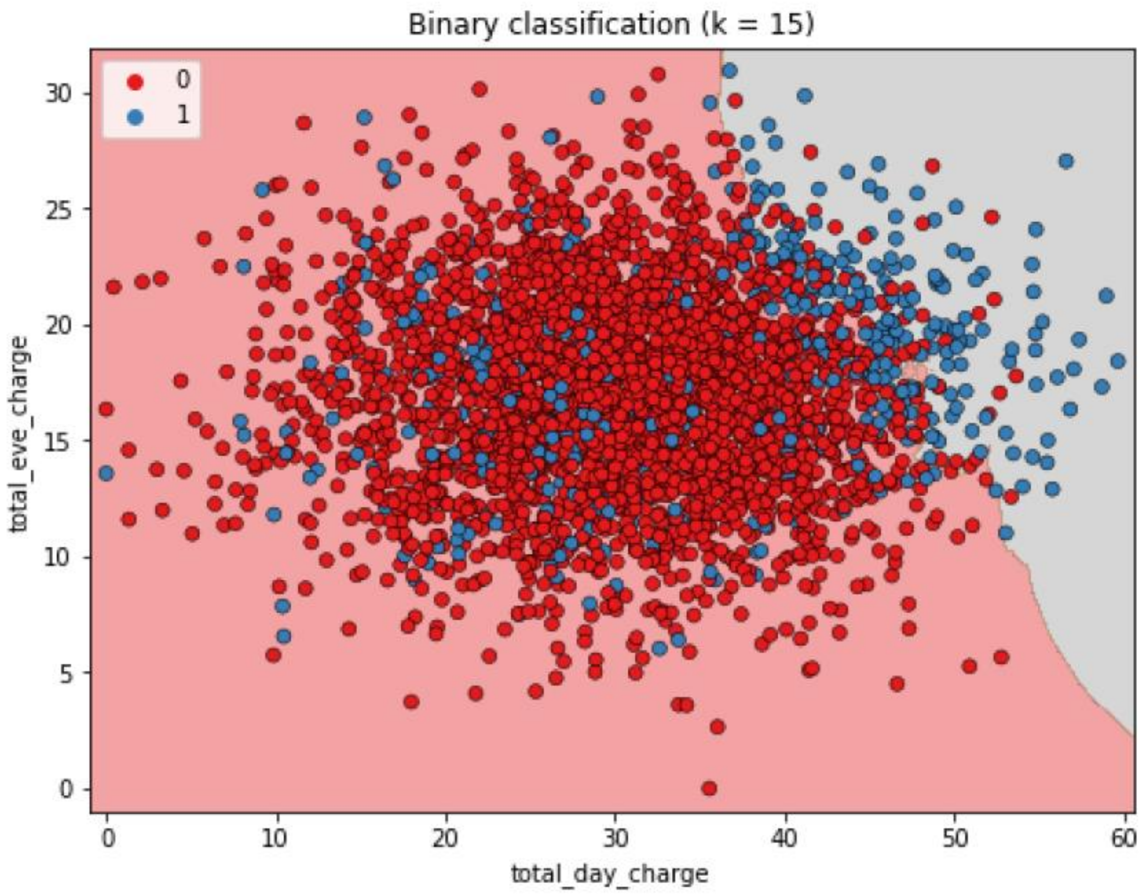
k-Nearest Neighbors - KNN



k-Nearest Neighbors - KNN Intuition



k-Nearest Neighbors - KNN Intuition



Using scikit-learn to fit a classifier

```python
from sklearn.neighbors import KNeighborsClassifier
X = churn_df[["total_day_charge", "total_eve_charge"]].values
y = churn_df["churn"].values
print(X.shape, y.shape)
```

```
(3333, 2), (3333,)
```

```python
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X, y)
```

Predicting on unlabeled data

```
X_new = np.array([[56.8, 17.5],
                  [24.4, 24.1],
                  [50.1, 10.9]])
print(X_new.shape)
```

```
(3, 2)
```

```
predictions = knn.predict(X_new)
print('Predictions: {}'.format(predictions))
```

```
Predictions: [1 0 0]
```

First practice!

Measuring model performance

Measuring model performance

- In classification, accuracy is a commonly used metric

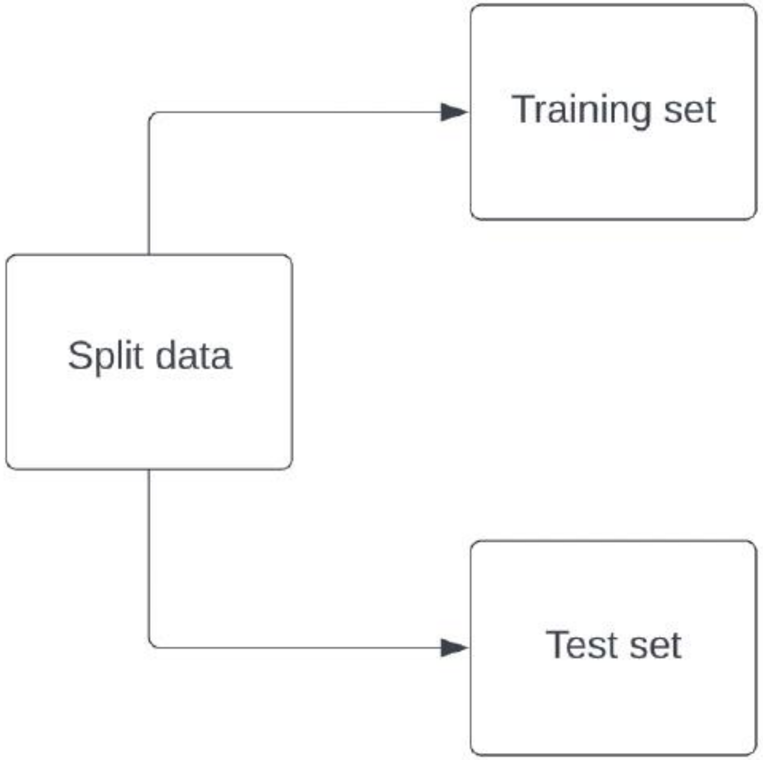
Accuracy:

$$\frac{correct\ predictions}{total\ observations}$$

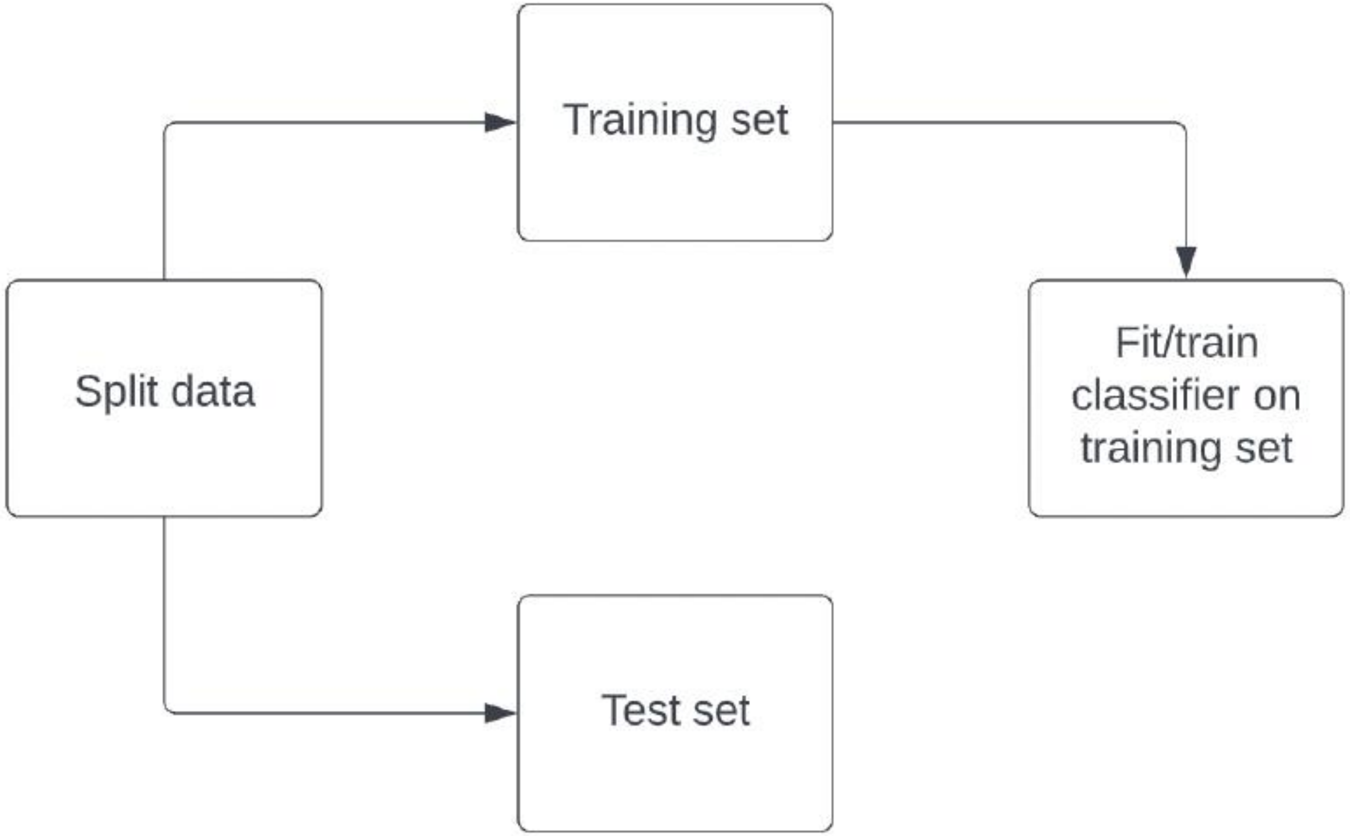
Measuring model performance

- How do we measure accuracy?
- Could compute accuracy on the data used to fit the classifier
- **NOT** indicative of ability to generalize

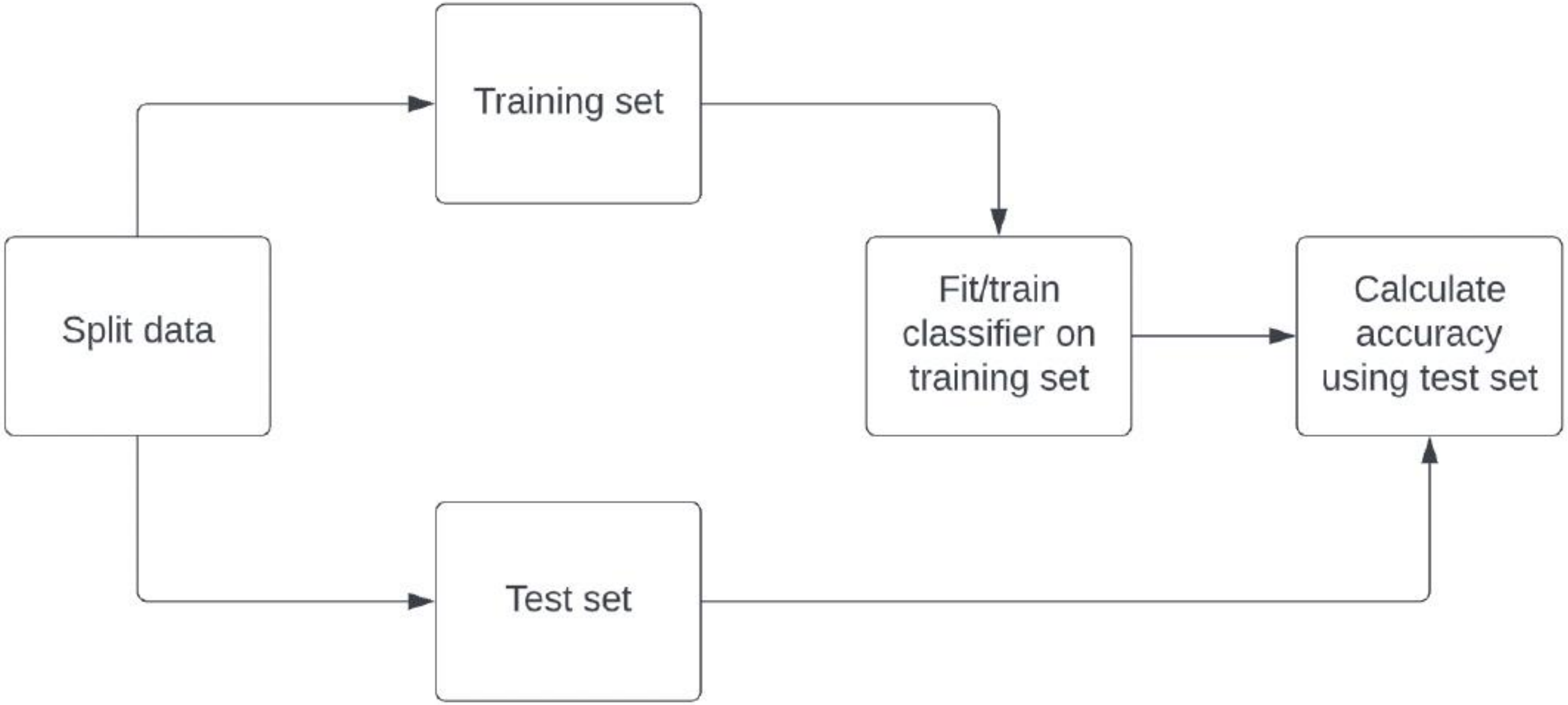
Computing accuracy



Computing accuracy



Computing accuracy



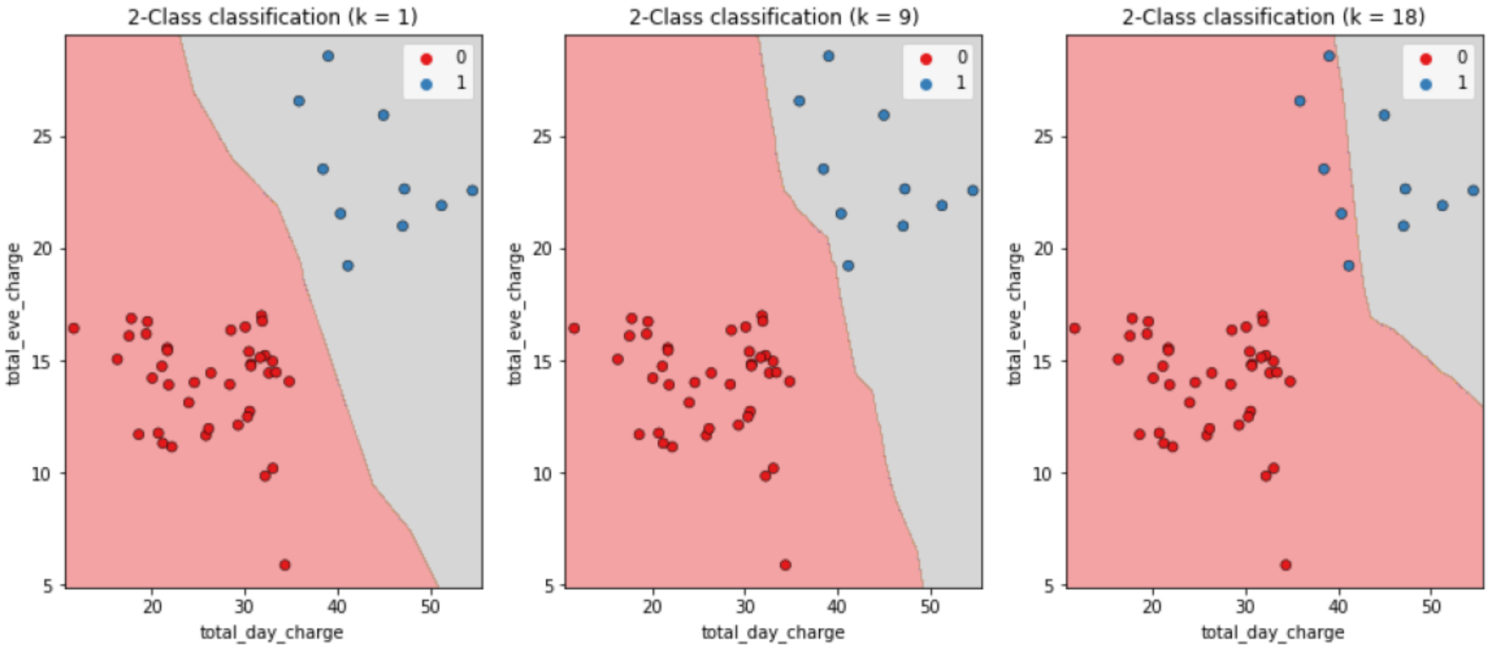
Train/test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=21, stratify=y)
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.8800599700149925
```

Model complexity

- **Larger k** = less complex model = can cause underfitting
- **Smaller k** = more complex model = can lead to overfitting



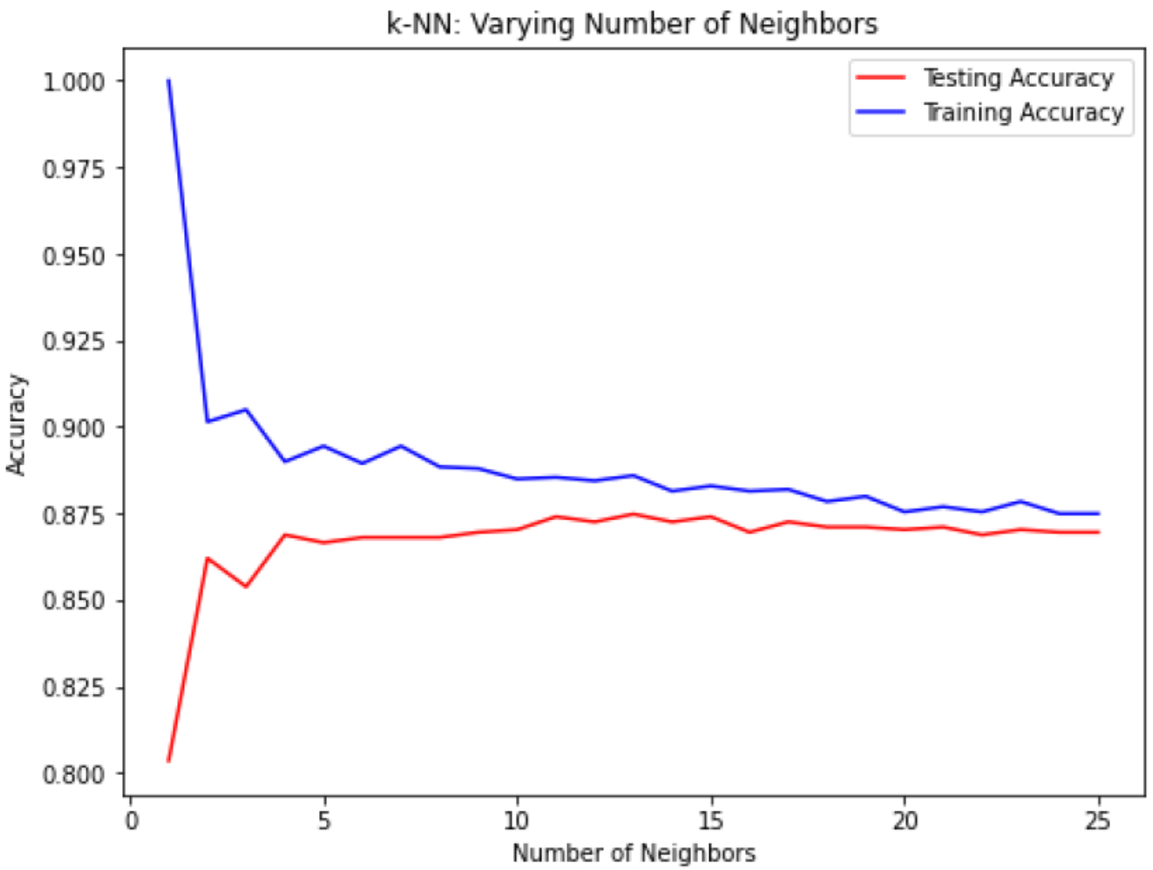
Model complexity and over/underfitting

```
train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1, 26)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)
```

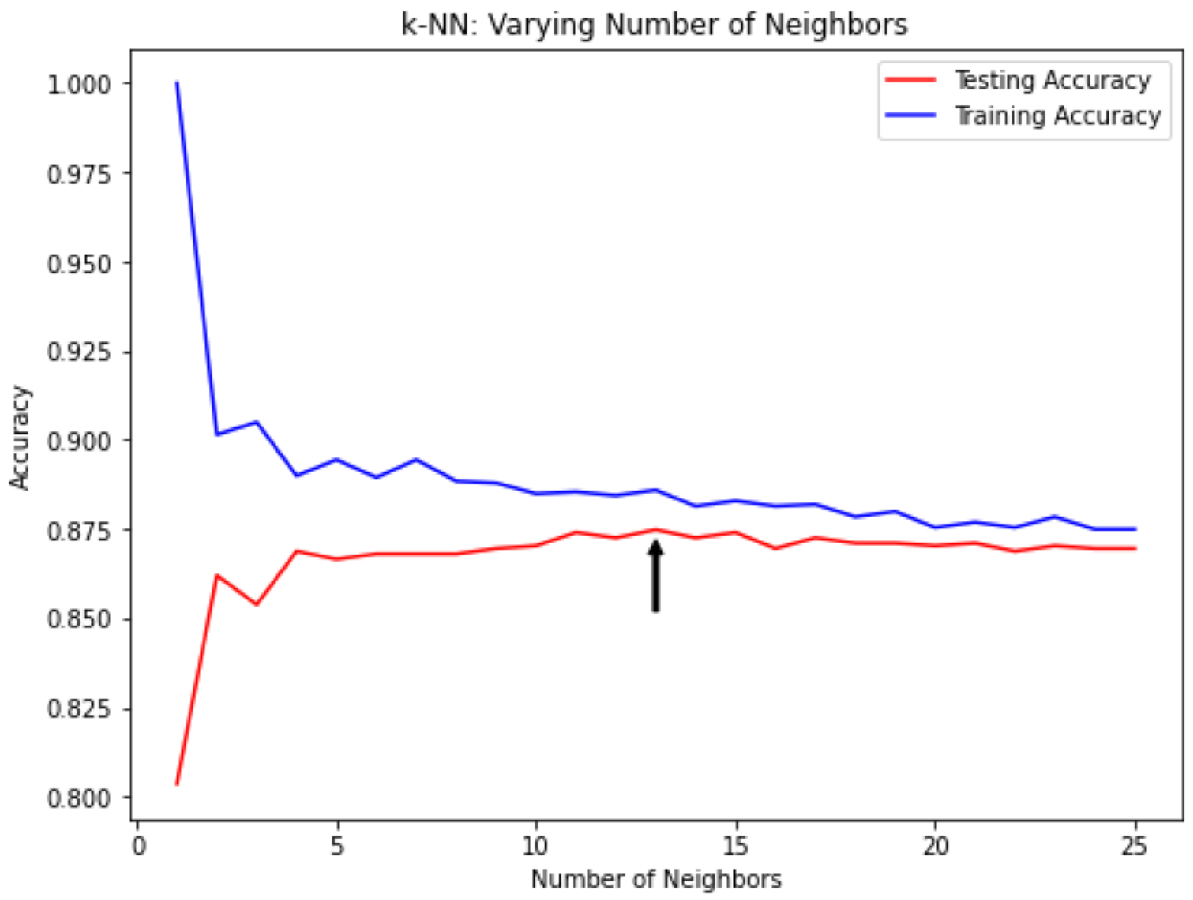
Plotting our results

```
plt.figure(figsize=(8, 6))
plt.title("KNN: Varying Number of Neighbors")
plt.plot(neighbors, train_accuracies.values(), label="Training Accuracy")
plt.plot(neighbors, test_accuracies.values(), label="Testing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

Model complexity curve



Model complexity curve



Introduction to regression

Introduction to regression

```python
import pandas as pd
diabetes_df = pd.read_csv("diabetes.csv")
print(diabetes_df.head())
```

	pregnancies	glucose	triceps	insulin	bmi	age	diabetes
0	6	148	35	0	33.6	50	1
1	1	85	29	0	26.6	31	0
2	8	183	0	0	23.3	32	1
3	1	89	23	94	28.1	21	0
4	0	137	35	168	43.1	33	1

Creating feature and target arrays

```
X = diabetes_df.drop("glucose", axis=1).values
y = diabetes_df["glucose"].values
print(type(X), type(y))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Making predictions from a single feature

```
X_bmi = X[:, 4]
print(y.shape, X_bmi.shape)
```

```
(768,) (768,)
```

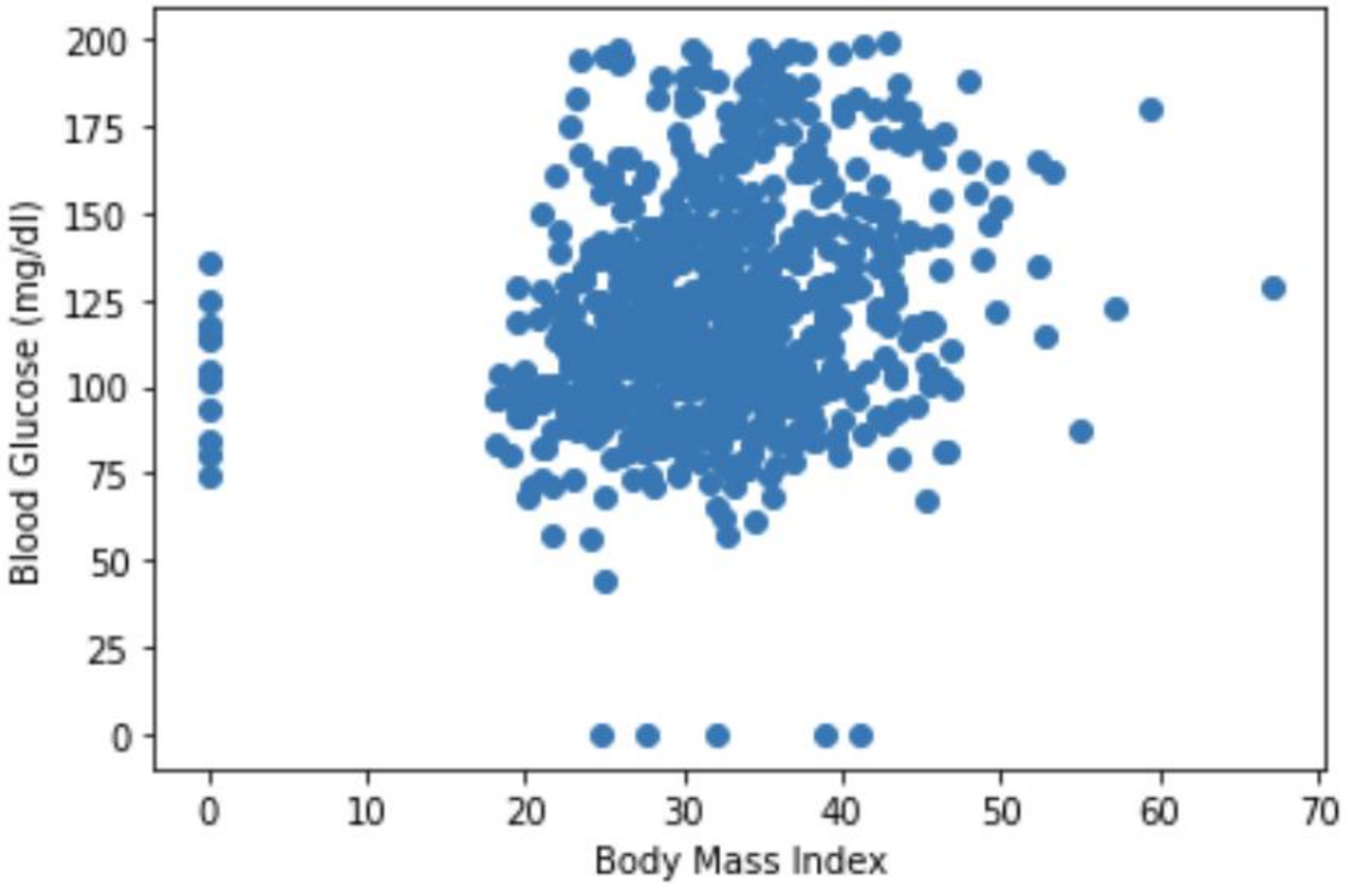
```
X_bmi = X_bmi.reshape(-1, 1)
print(X_bmi.shape)
```

```
(768, 1)
```

Plotting glucose vs. body mass index

```python
import matplotlib.pyplot as plt
plt.scatter(X_bmi, y)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```

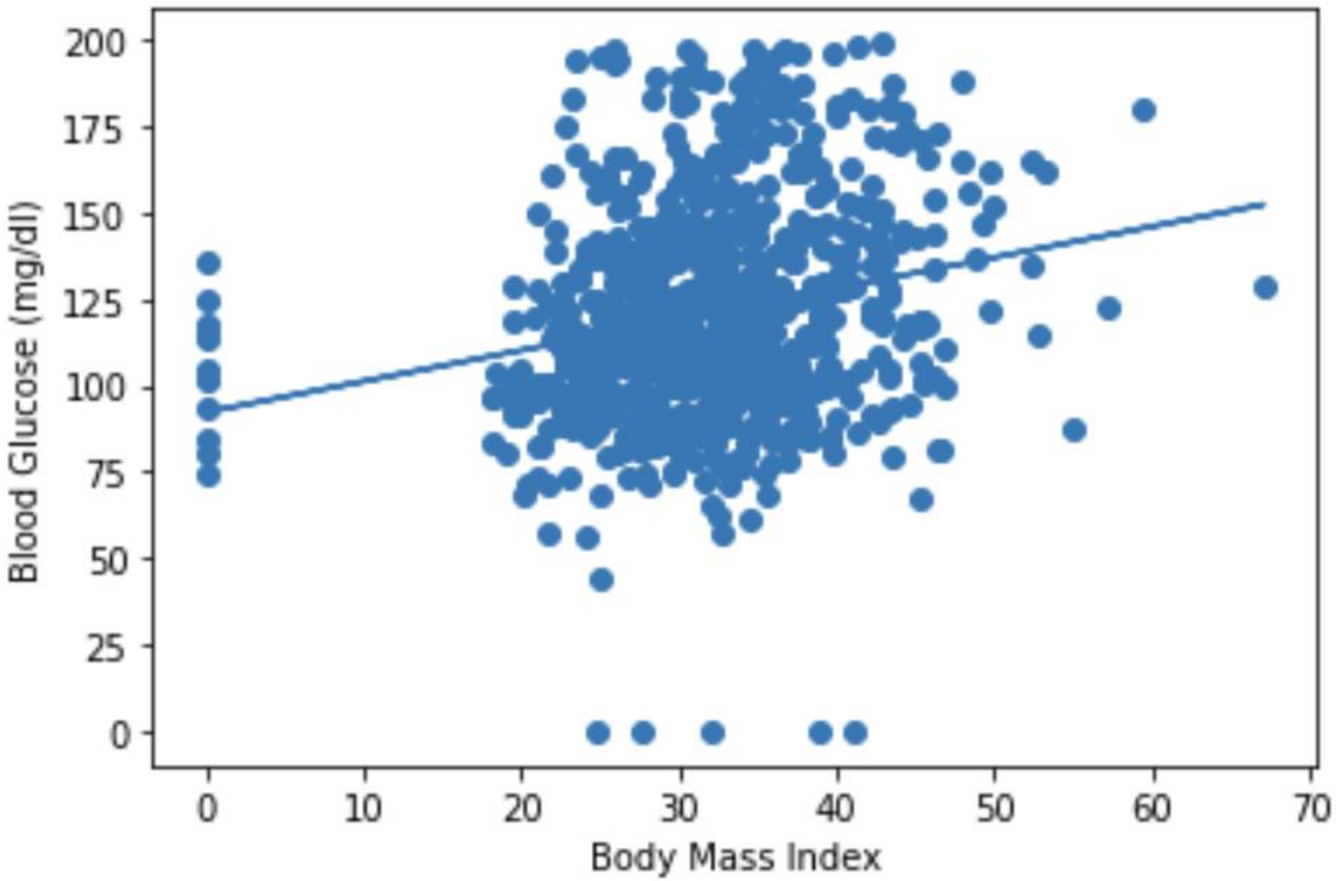
Plotting glucose vs. body mass index



Fitting a regression model

```python
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_bmi, y)
predictions = reg.predict(X_bmi)
plt.scatter(X_bmi, y)
plt.plot(X_bmi, predictions)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```

Fitting a regression model



SLAFES XXV Barranquilla 2024 ICTP 1964–2024

The basics of linear regression

Regression mechanics

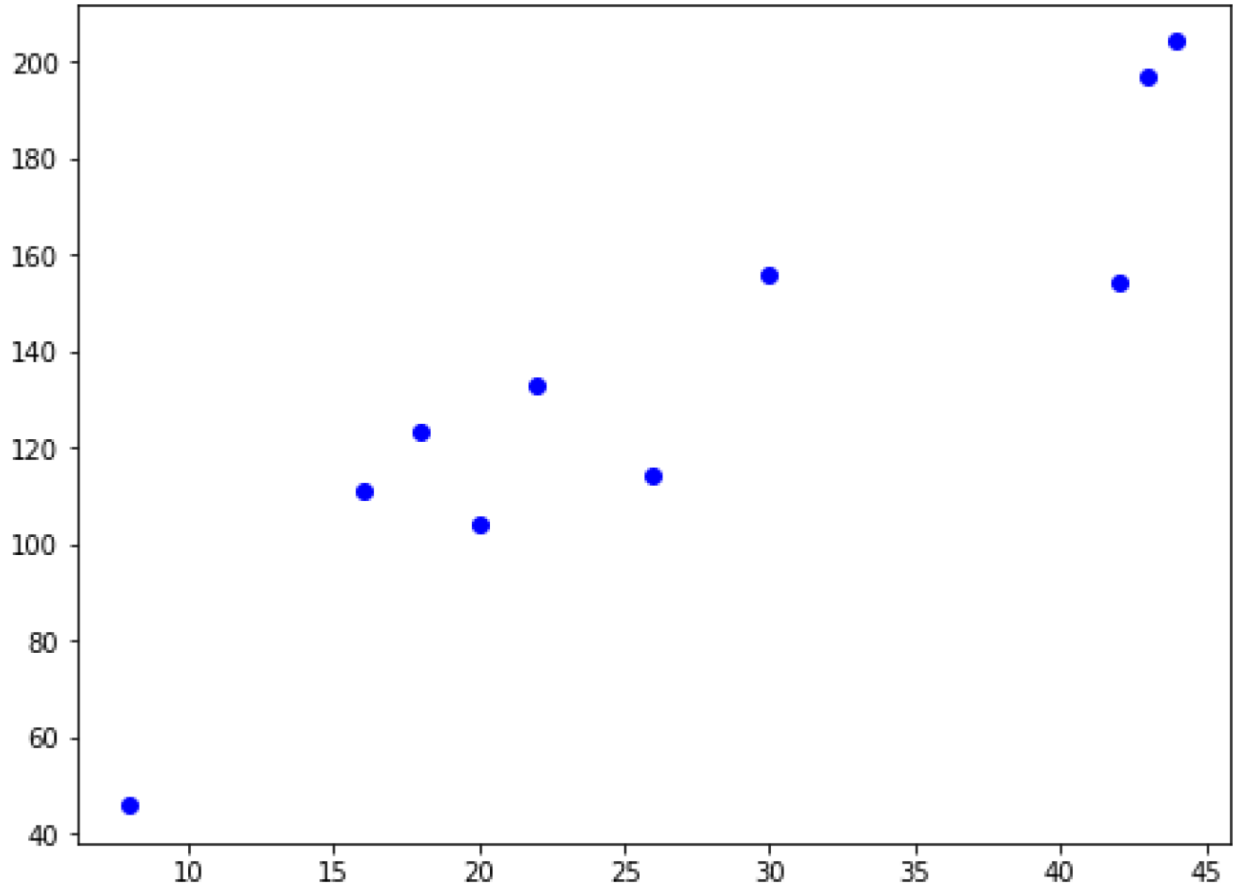
$$y = ax + b$$

- Simple linear regression uses one feature
  - y = target
  - x = single feature
  - a, b = parameters/coefficients of the model - slope, intercept

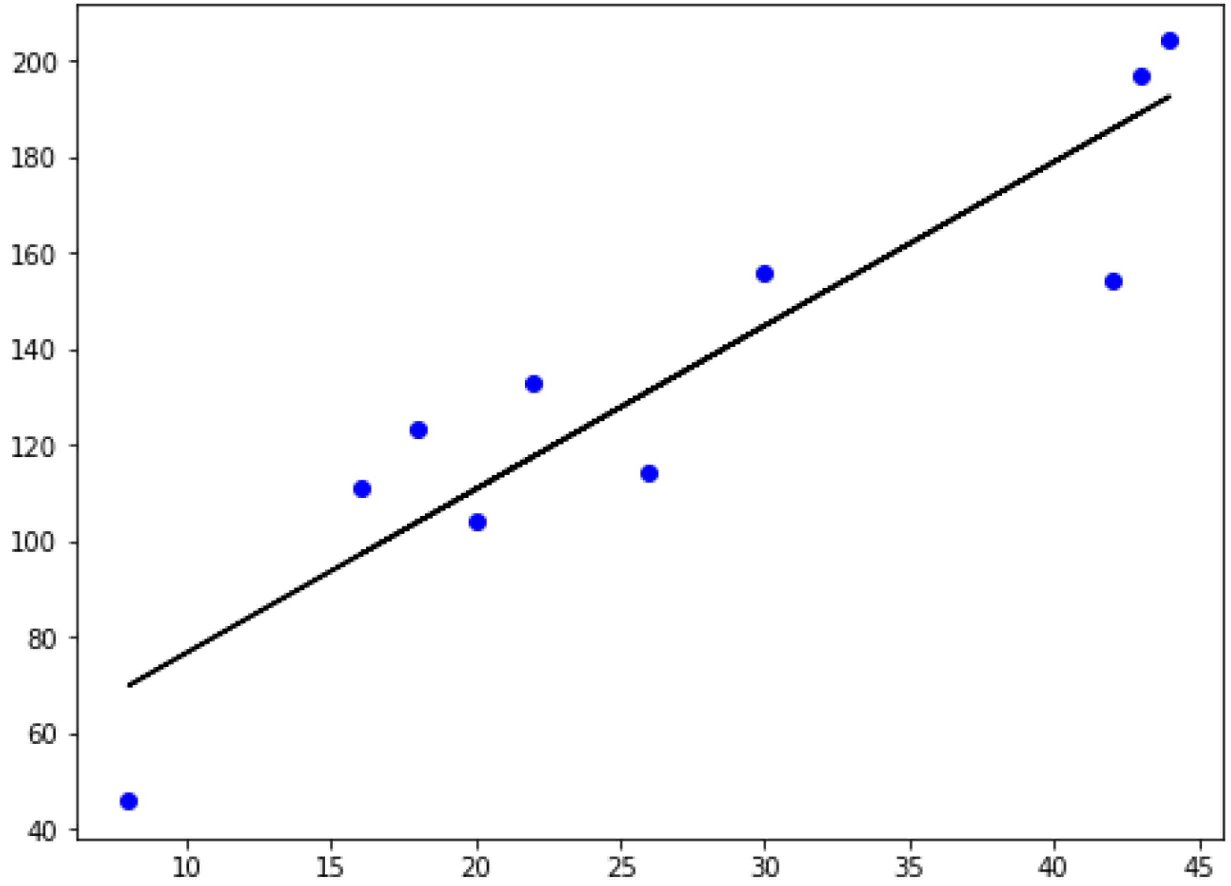
- How do we choose a and b?
  - Define an error function for any given line
  - Choose the line that minimizes the error function

- Error function = loss function = cost function

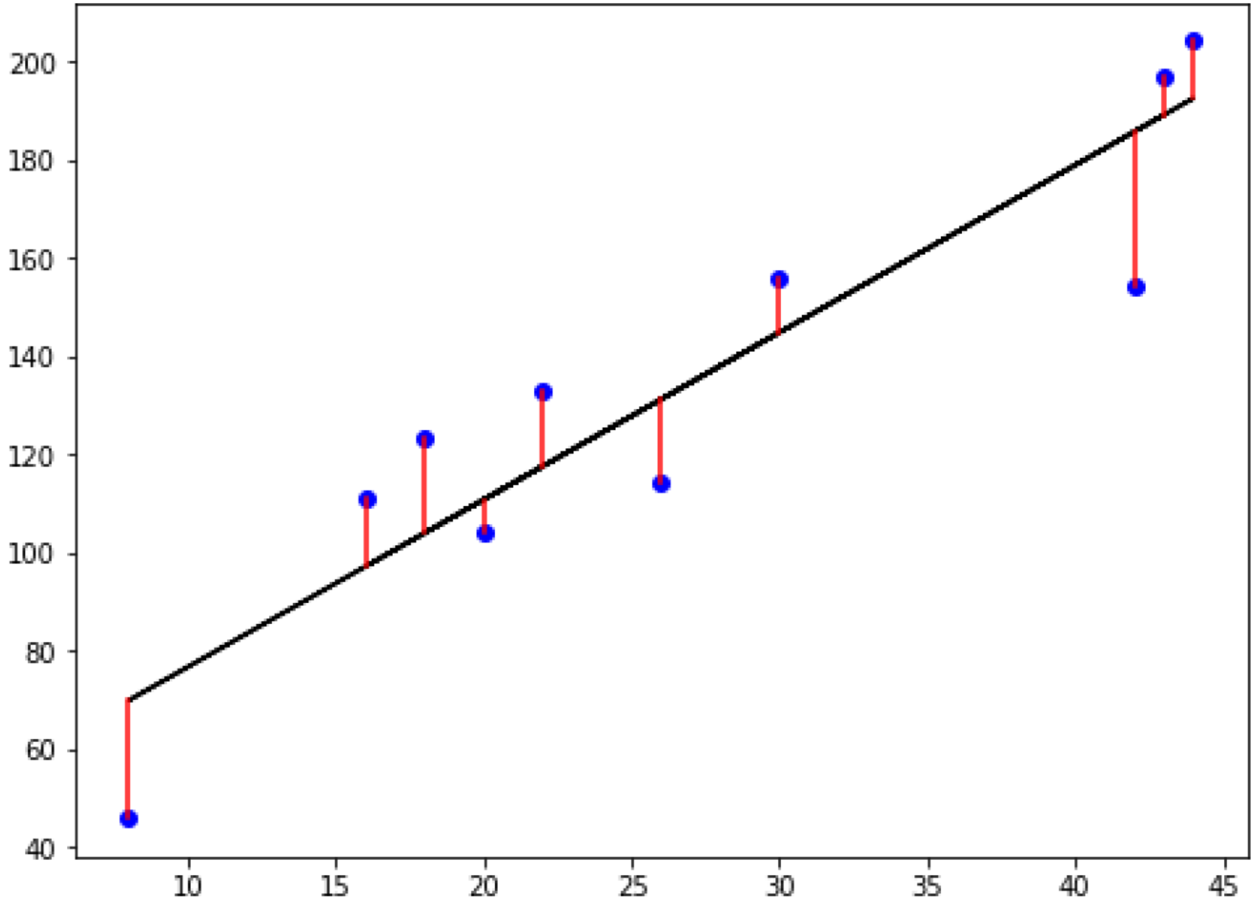
The loss function



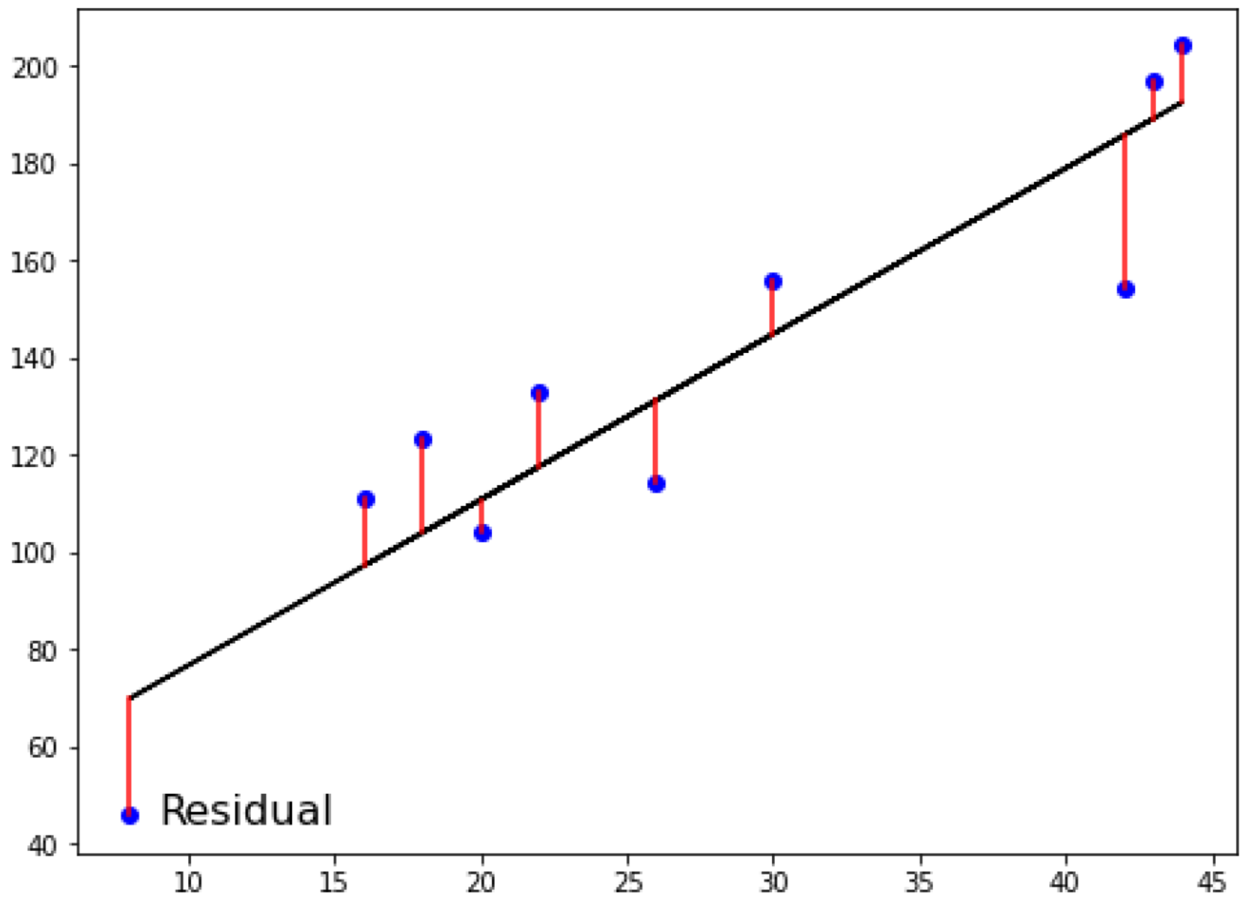
The loss function



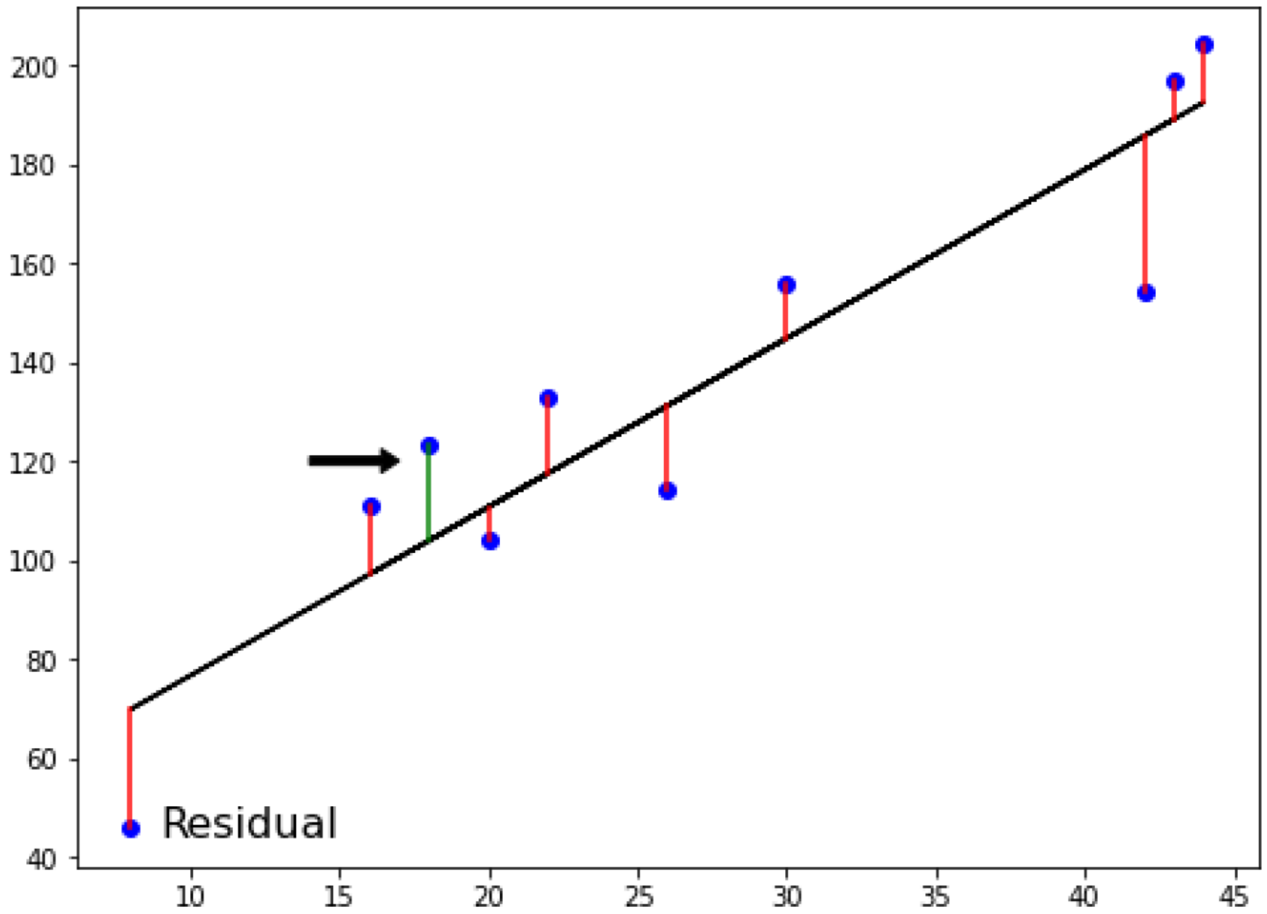
The loss function



The loss function



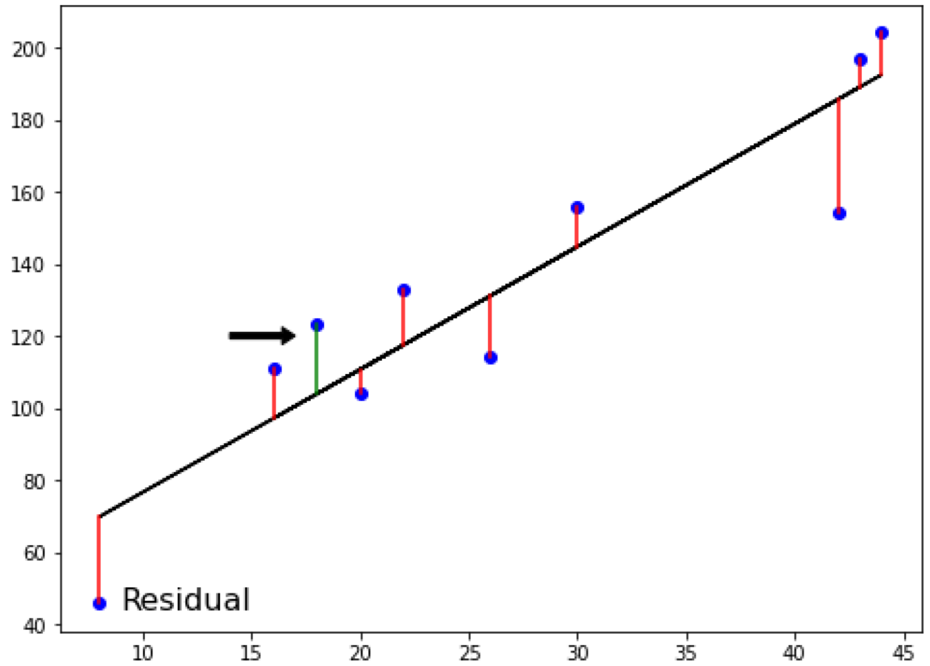
The loss function



The loss function
Ordinary Least Squares

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

Ordinary Least Squares (OLS): minimize RSS



Linear regression in higher dimensions

$$y = a_1x_1 + a_2x_2 + b$$

- To fit a linear regression model here:
  - Need to specify 3 variables: a_1, a_2,b
- In higher dimensions:
  - Known as multiple regression
  - Must specify coefficients for each feature and the variable b

$$y = a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n + b$$

- scikit-learn works exactly the same way:
  - Pass two arrays: features and target

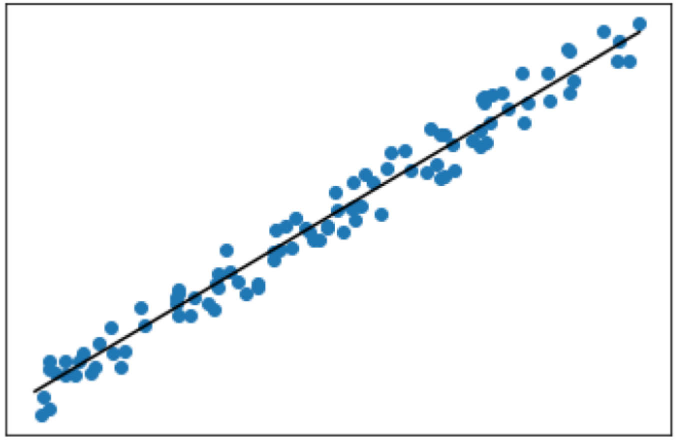
Linear regression using all features

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)
```

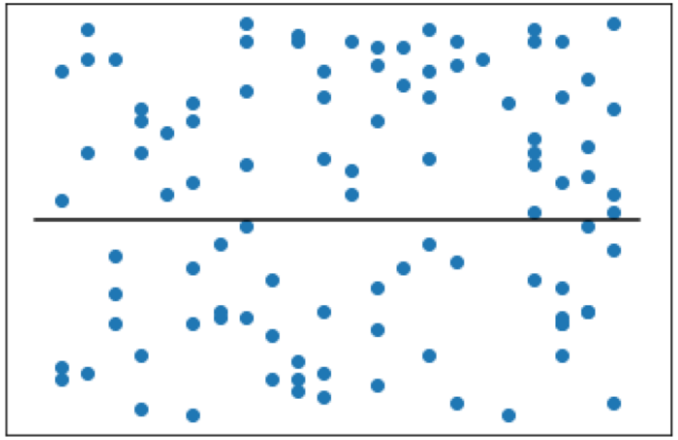
R-squared

- R^2: quantifies the variance in target values explained by the features
  - Values range from 0 to 1

High R^2:



Low R^2:



R-squared in scikit-learn

```
reg_all.score(X_test, y_test)
```

```
0.356302876407827
```

Mean squared error and root mean squared error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE is measured in target units, squared

$$RMSE = \sqrt{MSE}$$

Measure **RMSE** in the same units at the target variable

RMSE in scikit-learn

```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred, squared=False)
```

```
24.028109426907236
```

Cross-validation

Cross-validation motivation

- Model performance is dependent on the way we split up the data
- Not representative of the model's ability to generalize to unseen data
- **Solution**: Cross-validation!

Cross-validation basics

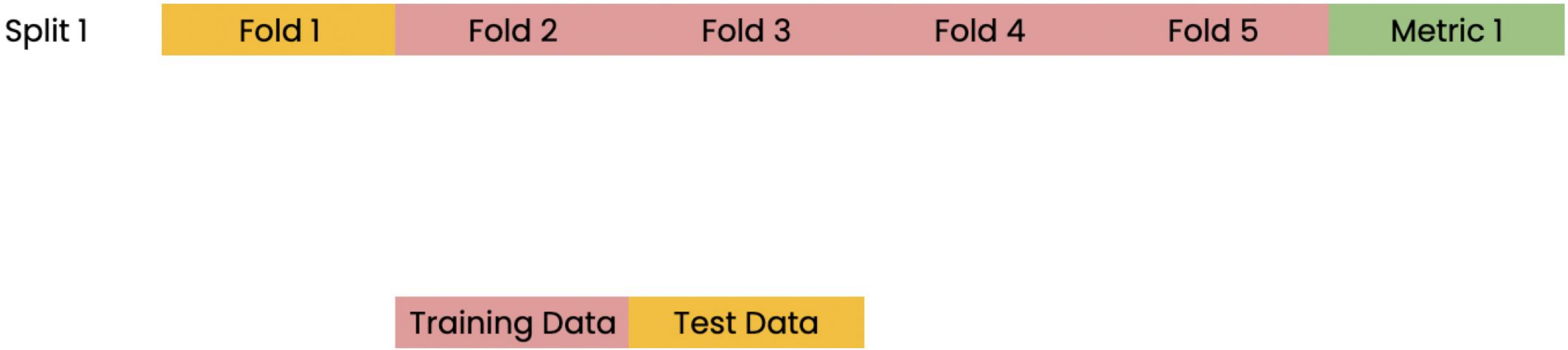
Split 1 Fold 1 Fold 2 Fold 3 Fold 4 Fold 5

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Test Data

Cross-validation basics



SLAFES XXV Barranquilla 2024
60 ICTP 1964–2024

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	

Training Data	Test Data

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	

Training Data	Test Data

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2

Training Data	Test Data

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3

Training Data	Test Data

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 4

Training Data	Test Data

Cross-validation basics

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 4
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 5

Training Data	Test Data

Cross-validation and model performance

- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k-fold CV
- More folds = More computationally expensive

Cross-validation in scikit-learn

```python
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(n_splits=6, shuffle=True, random_state=42)
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=kf)
```

Evaluating cross-validation peformance

```
print(cv_results)
```

```
[0.70262578, 0.7659624, 0.75188205, 0.76914482, 0.72551151, 0.73608277]
```

```
print(np.mean(cv_results), np.std(cv_results))
```

```
0.7418682216666667 0.023330243960652888
```

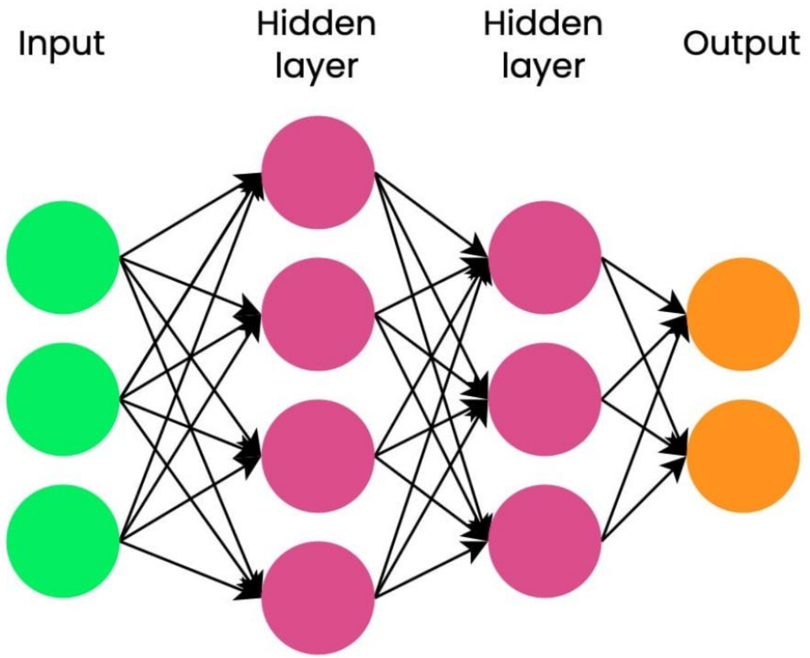
```
print(np.quantile(cv_results, [0.025, 0.975]))
```

```
array([0.7054865, 0.76874702])
```

Introduction to deep learning with PyTorch

What is deep learning?

- Deep learning is a subset of machine learning
- Inspired by connections in the human brain
- Models require large amount of data



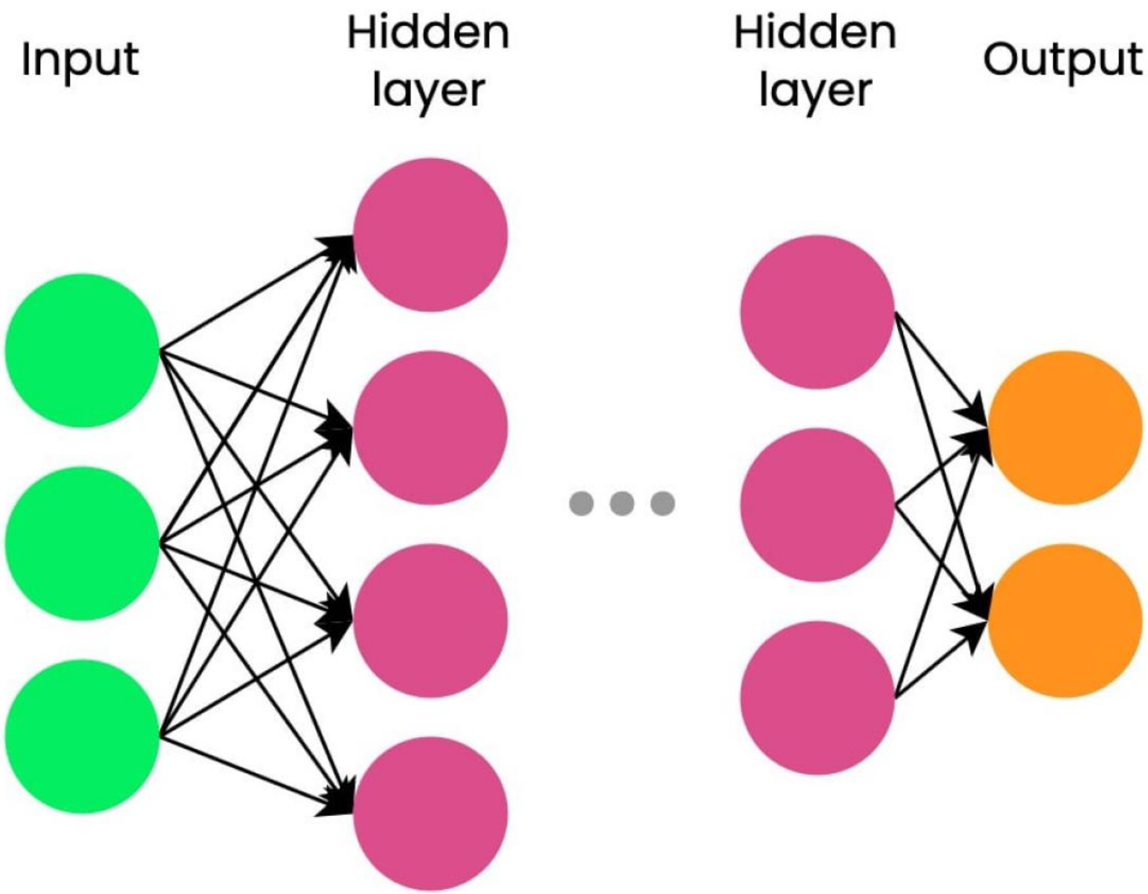
Importing PyTorch and related packages

- PyTorch import in Python

```python
import torch
```

- PyTorch supports
  - image data with torchvision
  - audio data with torchaudio
  - text data with torchtext

Creating our first neural network



Creating our first neural network

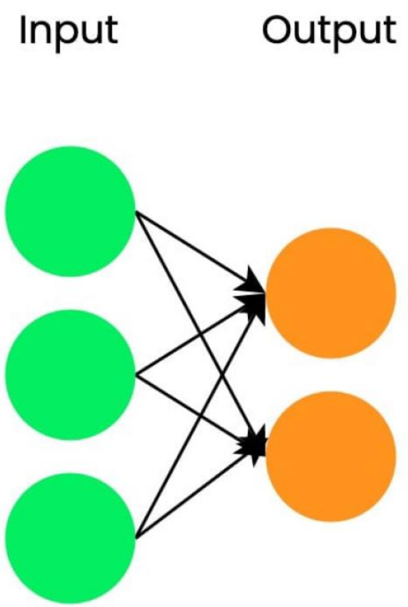
```python
import torch.nn as nn
```

```python
## Create input_tensor with three features
input_tensor = torch.tensor(
    [[0.3471, 0.4547, -0.2356]])
```

```python
# Define our first linear layer
linear_layer = nn.Linear(in_features=3, out_features=2)
```

```python
# Pass input through linear layer
output = linear_layer(input_tensor)
print(output)
```

```
tensor([[-0.2415, -0.1604]],
    grad_fn=<AddmmBackward0>)
```



Getting to know the linear layer operation

Each linear layer has a **.weight** and **.bias** property

```
linear_layer.weight
```

```
Parameter containing:
tensor([[-0.4799,  0.4996,  0.1123],
        [-0.0365, -0.1855,  0.0432]],
        requires_grad=True)
```

```
linear_layer.bias
```

```
Parameter containing:
tensor([0.0310, 0.1537], requires_grad=True)
```

Getting to know the linear layer operation

```
output = linear_layer(input_tensor)
```

For input X , weights X_0 and bias b_0, the linear layer performs

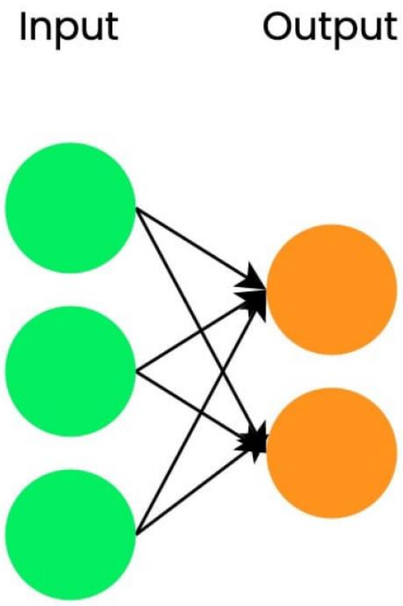
$$y_0 = W_0\, X + b_0$$

In PyTorch: output = W0 @ input + b0

- Weights and biases are initialized randomly
- They are not useful until they are tuned

Our two-layer network summary

- Input dimensions: 1×3
- Linear layer arguments:
  - in_features = 3
  - out_features = 2
- Output dimensions: 1×2
- Networks with only linear layers are called **fully connected**
- Each neuron in one layer is connected to each neuron in the next layer



Stacking layers with nn.Sequential()

```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(10, 18),
    nn.Linear(18, 20),
    nn.Linear(20, 5)
)
```

Stacking layers with nn.Sequential()

```
print(input_tensor)
```

```
tensor([[-0.0014,  0.4038,  1.0305,  0.7521, 0.7489, -0.3968,  0.0113, -1.3844, 0.8705, -0.9743]])
```

```
# Pass input_tensor to model to obtain output
output_tensor = model(input_tensor)
print(output_tensor)
```

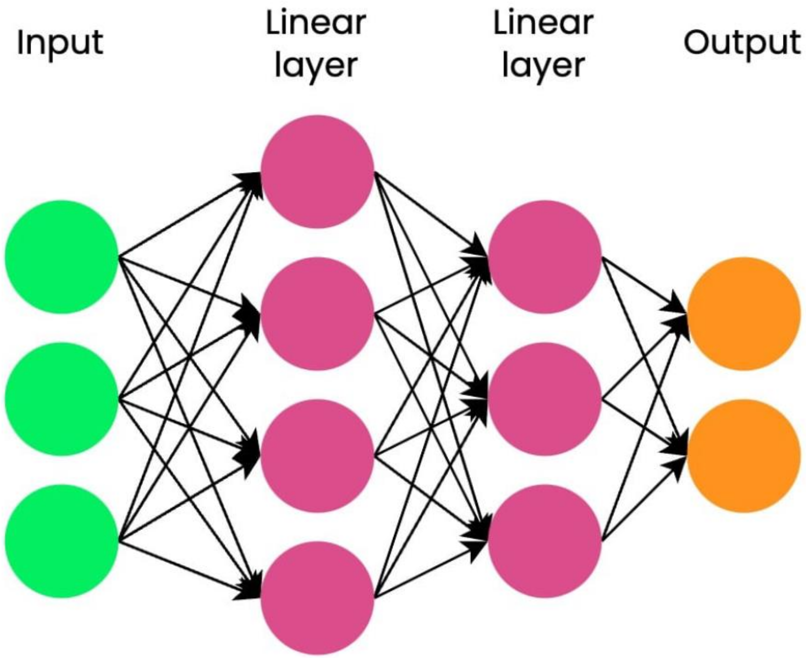
```
tensor([[-0.0254, -0.0673,  0.0763,
0.0008,  0.2561]], grad_fn=<AddmmBackward0>)
```

- We obtain output of 1×5 dimensions
- Output is still not yet meaningful

Discovering activation functions

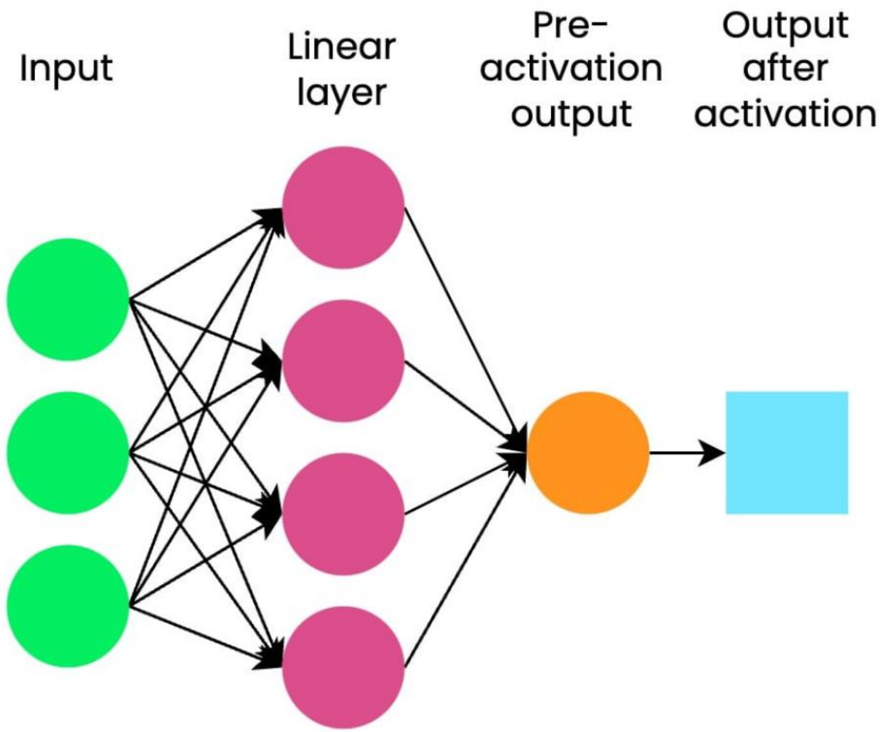
Stacked linear operations

- We have only seen linear layer networks
- Each linear layer multiplies its respective input with layer weights and adds biases
- Even with multiple stacked linear layers, output still has linear relationship with input



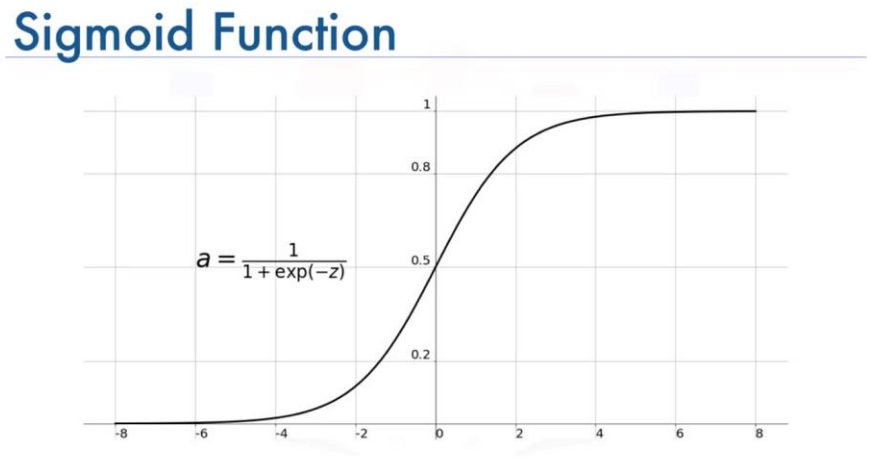
Why do we need activation functions?

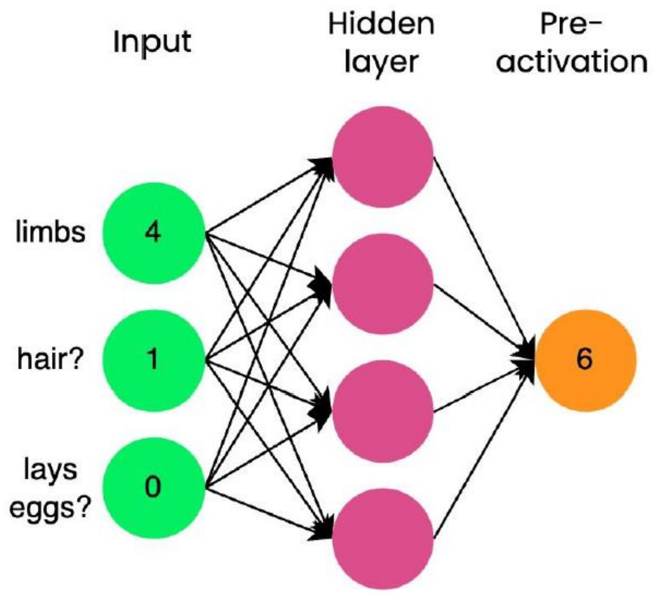
- Activation functions add **non-linearity** to the network
- A model can learn more **complex** relationships with non-linearity



Meet the sigmoid function

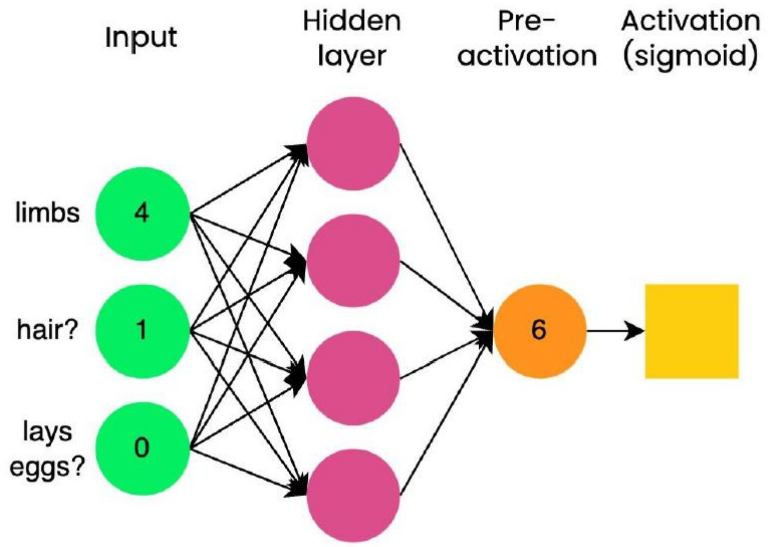
- Binary classification task:
  - To predict whether animal is 1 (**mammal**) or 0 (**not mammal**)





Meet the sigmoid function

- Binary classification task:
  - To predict whether animal is 1 (**mammal**) or 0 (**not mammal**)
  - we take the pre-activation (6), pass it to the sigmoid,



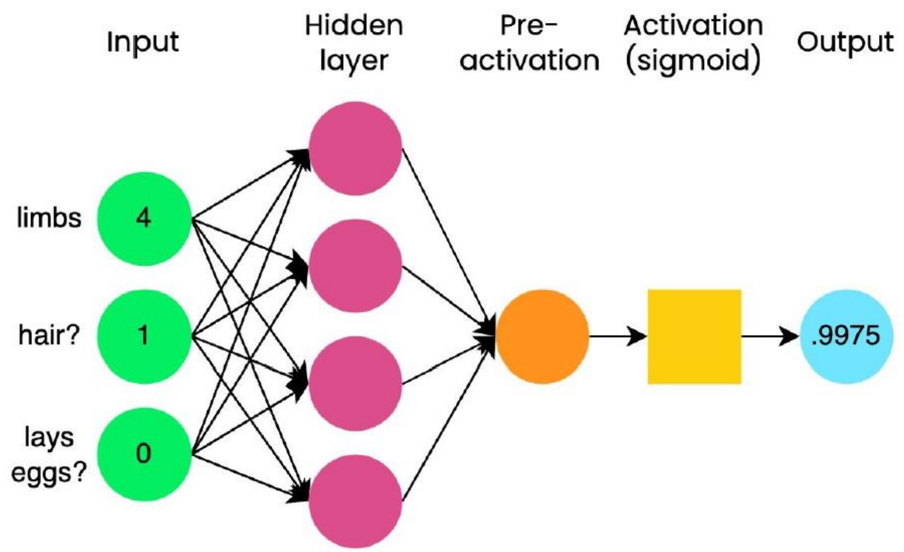
Meet the sigmoid function

Binary classification task:

- To predict whether animal is 1 (**mammal**) or 0 (**not mammal**)
- we take the pre-activation (6), pass it to the sigmoid
- and obtain a value between 0 and

Using the common threshold of 0.5:

- If output is > 0.5, class label = 1 (**mammal**)
- If output is <= 0.5, class label = 0 (**not mammal**)



Meet the sigmoid function

```python
import torch
import torch.nn as nn

input_tensor = torch.tensor([[6.0]])
sigmoid = nn.Sigmoid()
output = sigmoid(input_tensor)
```

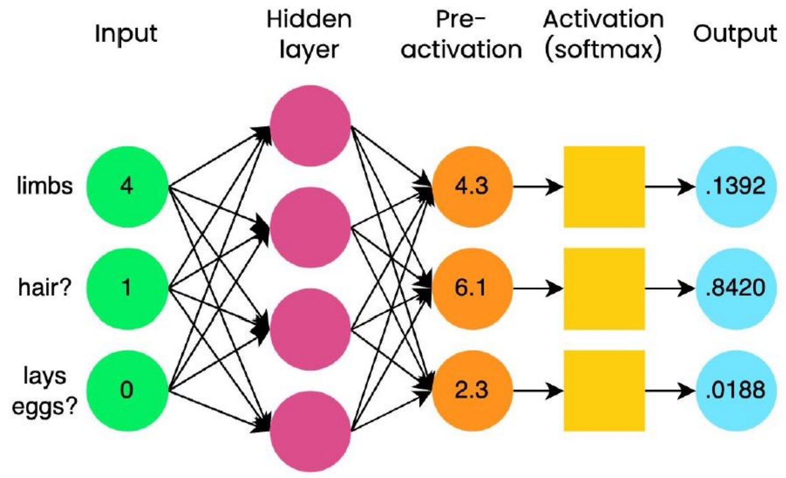
```
tensor([[0.9975]])
```

Activation function as the last layer

```
model = nn.Sequential(
  nn.Linear(6, 4), # First linear layer
  nn.Linear(4, 1), # Second linear layer
  nn.Sigmoid() # Sigmoid activation function
)
```

Getting acquainted with softmax

- Used for multi-class classification problems
- takes N-element vector as input and outputs vector of same size
- say N=3 classes:
  - bird (0), mammal (1), reptile (2)
  - output has three elements, so **softmax** has three elements
- outputs a probability distribution:
  - each element is a probability (it's bounded between 0 and 1)
  - the sum of the output vector is equal to 1



Getting acquainted with softmax

```python
import torch
import torch.nn as nn

# Create an input tensor
input_tensor = torch.tensor(
    [[4.3, 6.1, 2.3]])

# Apply softmax along the last dimension
probabilities = nn.Softmax(dim=-1)
output_tensor = probabilities(input_tensor)

print(output_tensor)
```

```
tensor([[0.1392, 0.8420, 0.0188]])
```

- dim = -1 indicates softmax is applied to the input tensor's last dimension
- **nn.Softmax()** can be used as last step in **nn.Sequential()**