AI and Machine Learning

Zhiyun Lin

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Classification

- Classficication problem
- Perceptron model for classification
- Metrics to evaluate classifaction
- Logistic regression model for classfication
- Cross validation

Cat or dog?

- ☐ Machine learn to classify a cat or dog
- ✓ Offer a large number of pictures with cat or dog labels and train a model
- ✓ Use the trained model to predict where there is a cat or a dog

Is this a dog?

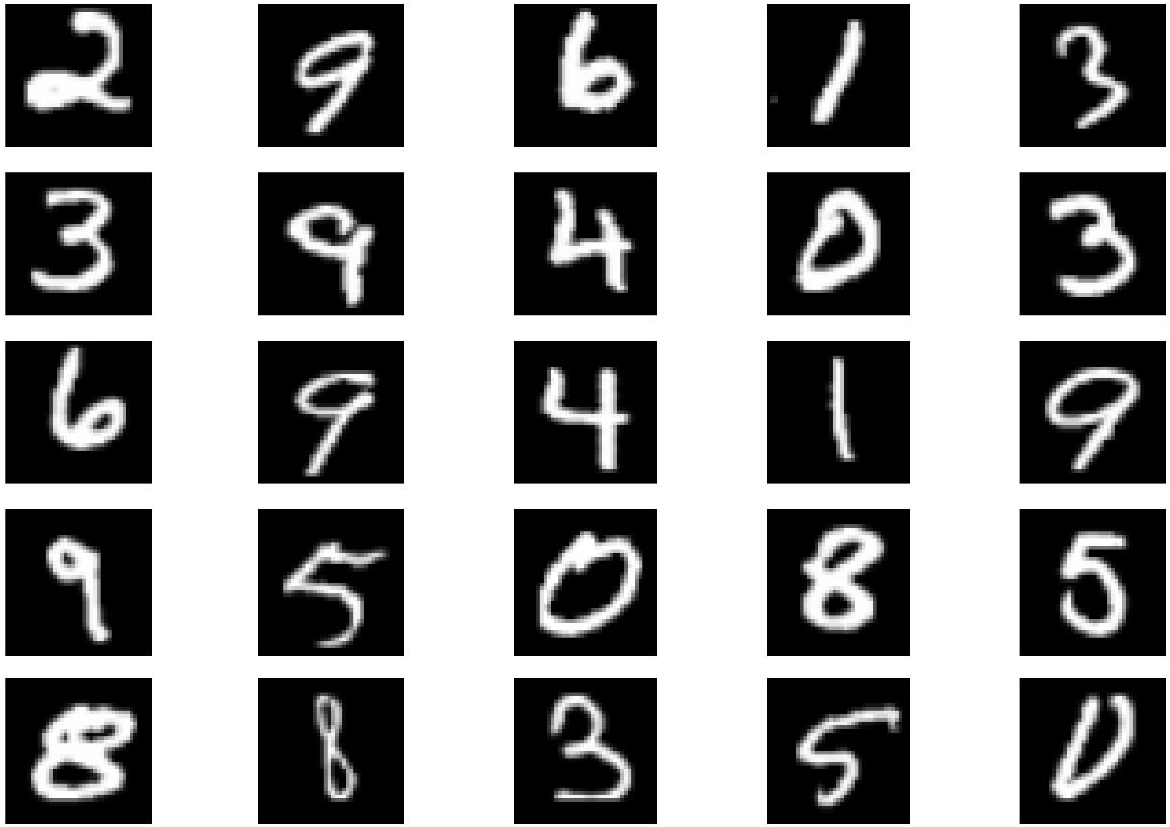





Am I going to pass exam?

Am I going to pass the exam?

What digit is this?



What digit is this?
How can I predict this? What are my input features?

Classification problem

- What do all these problems in common?
  - Categorical **outputs**, called **labels** (e.g., Yes/No, dog/cat/person/other)

- Assigning each input vector to one of a finite number of labels called **classification**.

  - **Binary classfication**: Two possible labels (e.g., Yes/No, 0/1, cat/dog)
  - **Multi-class classification**: Multiple possible labels

Classification vs Regression

- We are interested in mapping the input $x \in \mathcal{X}$ to an output $y \in \mathcal{Y}$

- For **regression problems**, typically $\mathcal{Y} = \mathbb{R}$
- For **classification problems**, $\mathcal{Y}$ is categorical

Treat classifcation as regression?

- Can we do this task using what we have learnt for regression problem?
- Simple hack: Ignore that the output is categorical!

- Suppose we have a binary problem: $t \in \{-1, 1\}$
- Assuming the standard model used for linear regression

$$y = z(x, w) = w^T \bar{x} \quad \text{where } x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \text{ and } \bar{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

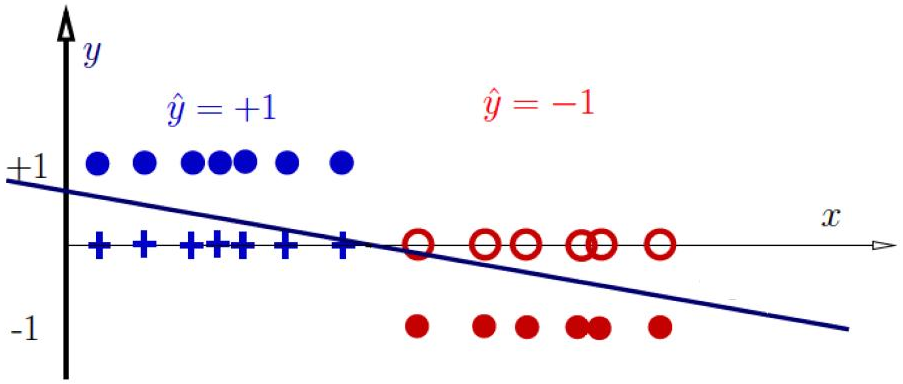
- **How can we obtain w?**

One dimensional example

- One dimensional example (input x is 1-dim)

- The colors indicate labels
  - A blue plus denotes that $t^{(i)}$ is from the first class (i.e., $t^{(i)} = 1$)
  - A red circle denotes that $t^{(i)}$ is from the second class (i.e., $t^{(i)} = -1$)

Decision rule



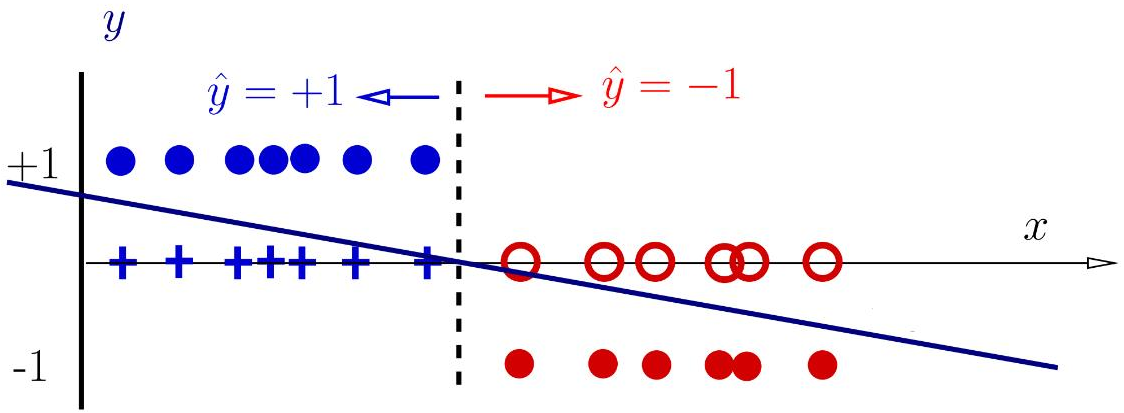
- Our linear regression model has the form

$$z(x, w) = w_0 + w_1 x = w^T \bar{x}, \quad \text{where } \bar{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

- A reasonable **decision rule** is

$$y = \begin{cases} 1 & \text{if } z(x, w) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron

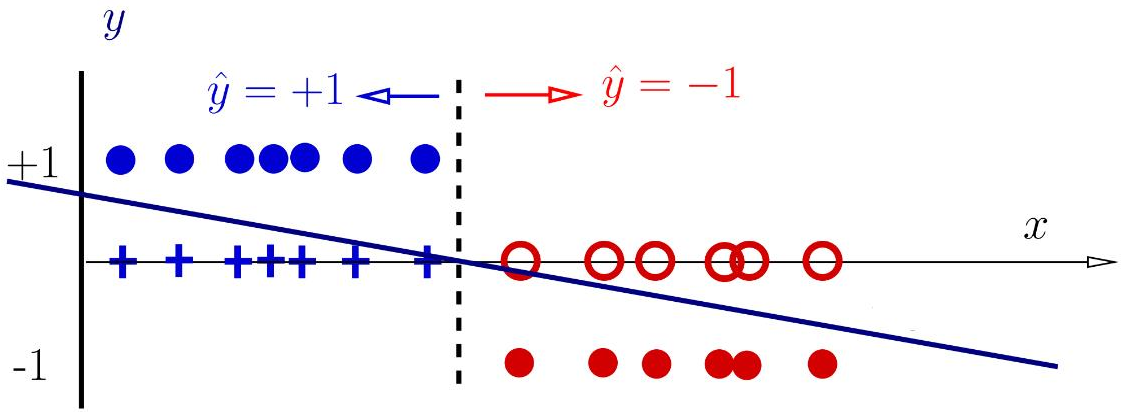


- How can we mathematically write this rule?

$$y = \text{sign}(z) = \text{sign}(w_0 + w_1 x)$$

- This specifies a **linear classifier**, called **perception**

Decision boundary

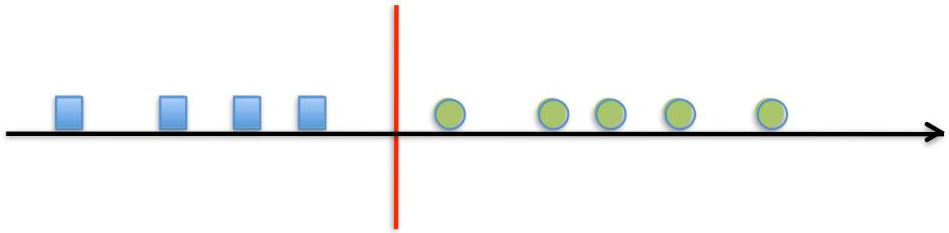


- **Perception** has a linear **decision boundary** (hyperplane)

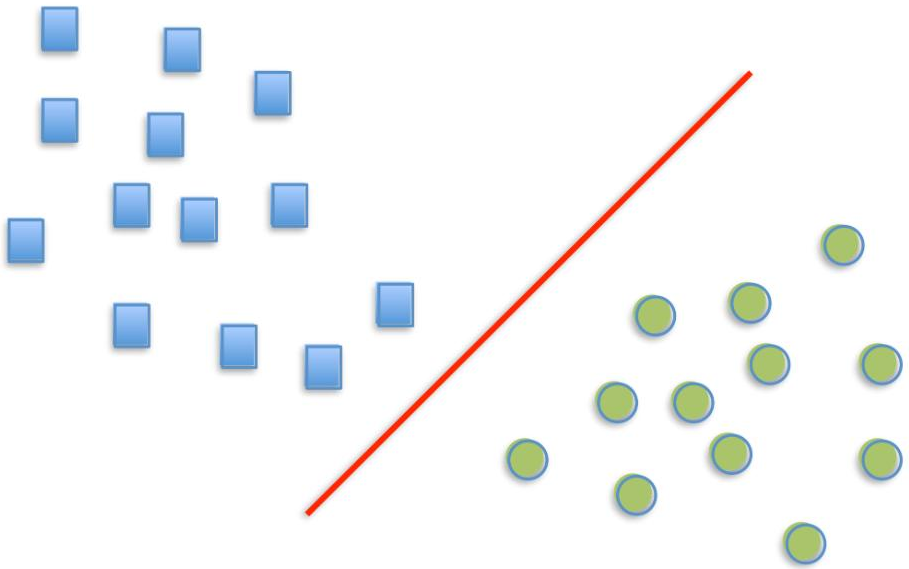
$$w_0 + w_1 x = 0$$

which seperates the space into two "half-spaces"

- In 1D, this is simply a **threshold**



Decision boundary in 2D



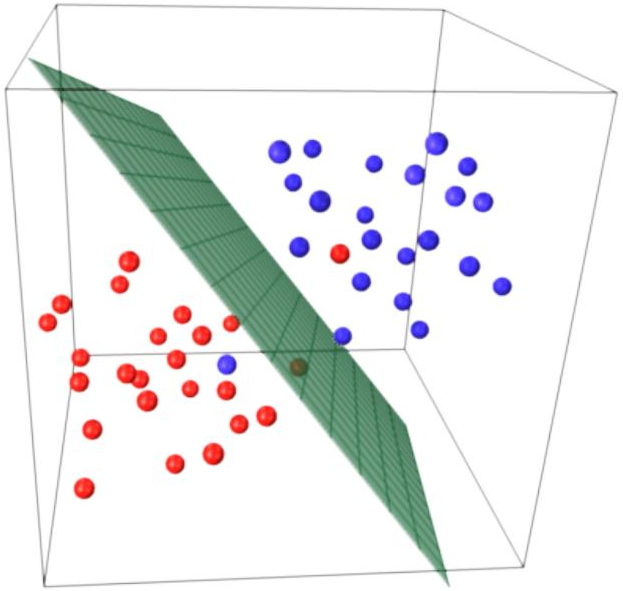
- **Perception** has a linear **decision boundary** (hyperplane)

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

which seperates the space into two "half-spaces"

- In 2D, this is simpnly a **line**

Decision boundary in 3D



- **Perception** has a linear **decision boundary** (hyperplane)

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$$

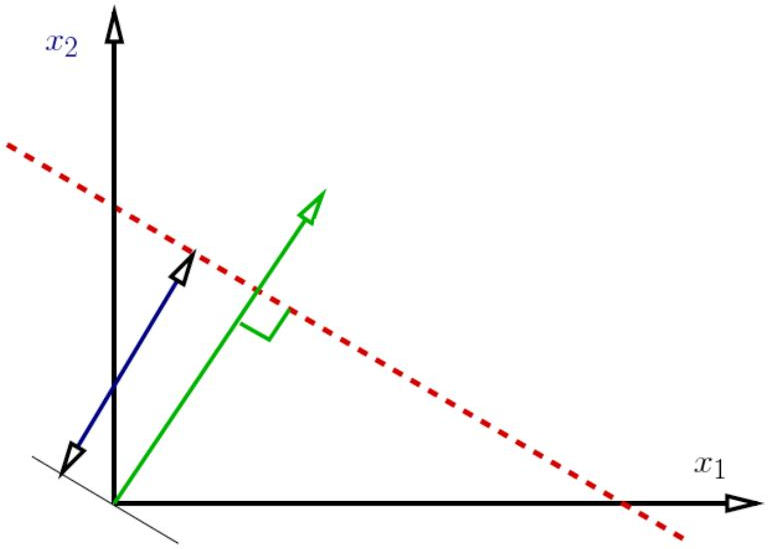
which seperates the space into two "half-spaces"

- In 3D, this is simpnly a **plane**

What about higher-dimensional spaces?

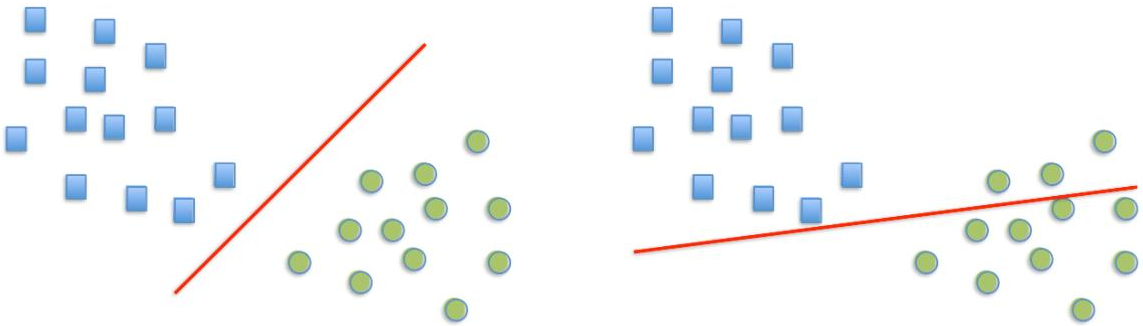
Geometry for decision boundary

- $w_0 + w_1 x_1 + w_2 x_2 = 0$ shifts $w_1 x_1 + w_2 x_2 = 0$ by w_0
- $w_1 x_1 + w_2 x_2 = 0$ passes through the origin and orthogonal to $u = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$



Learning perceptron

- Learning consists in estimating a "good" **decision boundary**
- We need to find $u = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ (direction) and w_0 (location) of the boundary



- Which boundary is good?
- We need a criteria that tell us how to select the parameters

Loss function for perceptron

- Linear model $z(x, w) = w^T \bar{x}$ with $\bar{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$
- **Perceptron**:

$$y(x, w) = \text{sign}(z(x, w)) = \text{sign}(w^T \bar{x})$$

- A possible loss to minimize is the **zero/one loss**

$$l(w) = \begin{cases} 0 & \text{if } y(x^{(n)}, w) = t^{(n)} \\ 1 & \text{if } y(x^{(n)}, w) \neq t^{(n)} \end{cases}$$

- Is this minimization easy to do? Why?

Make a new loss function

- Find one with non-vanishing gradient

$$l(w) = \left[-t^{(n)} z(x^{(n)}, w)\right]_+$$

- Then it can be known that

$$l(w) = \begin{cases} 0 & \text{if } y(x^{(n)}, w) = t^{(n)} \\ \text{positive} & \text{if } y(x^{(n)}, w) \neq t^{(n)} \end{cases}$$

- What about its gradient? Its **gradient**:

$$\nabla l(w) = \begin{cases} 0 & \text{if } y(x^{(n)}, w) = t^{(n)} \\ -t^{(n)} \bar{x}^{(n)} & \text{if } y(x^{(n)}, w) \neq t^{(n)} \end{cases}$$

SGD algorithm for perceptron

```
import numpy as np
import matplotlib.pyplot as plt

class Perceptron:

    def __init__(self, n_feature = 1, n_iter = 200, lr = 0.01, tol = None):
        self.n_iter = n_iter    # Maximum interation steps
        self.lr = lr            # Learning rate
        self.tol = tol          # Threshold for stopping iteration
        self.W = np.random.random(n_feature + 1) * 0.5          # Molel parameters
        self.loss = []          # The loss value
        self.best_loss = np.inf  # The best loss value
        self.patience = 10      # The patience for early stopping

    def _loss(self, y, y_pred):
        return - y_pred * y if y_pred * y < 0 else 0
```

SGD algorithm for perceptron

```
def _gradient(self, x_bar, y, y_pred):
    return -y * x_bar if y_pred * y <= 0 else 0

def _preprocess_data(self, X):
    m, n = X.shape
    X_ = np.empty([m, n+1])
    X_[:, 0] = 1
    X_[:,1:] = X
    return X_

def sgd_update(self, X, y):
    break_out = False
    epoch_no_improve = 0

    for iter in range(self.n_iter):
        for i, x in enumerate(X):
```

SGD algorithm for perceptron

```
y_pred = self._predict(x)
loss = self._loss(y[i], y_pred)
self.loss.append(loss)

if self.tol is not None:
    if loss < self.best_loss - self.tol:
        self.best_loss = loss
        epoch_no_improve = 0
    elif np.abs(loss - self.best_loss ) < self.tol:
        epoch_no_improve += 1
        if epoch_no_improve >= self.patience:
            print(f"Early stopping triggered due to the no improvement in loss.")
            break_out = True
            break
    else:
        epoch_no_improve = 0
```

SGD algorithm for perceptron

```
            grad = self._gradient(x, y[i], y_pred)
            self.W = self.W - self.lr * grad

        if break_out:
            break_out = False  # reset break status
            break  # Exit the outter loop

def _predict(self, X):
    return X @ self.W

def train(self, X_train, t_train):
    X_train_bar = self._preprocess_data(X_train)
    print(X_train_bar)
    self.sgd_update(X_train_bar, t_train)
    print(self.W)
```

SGD algorithm for perceptron

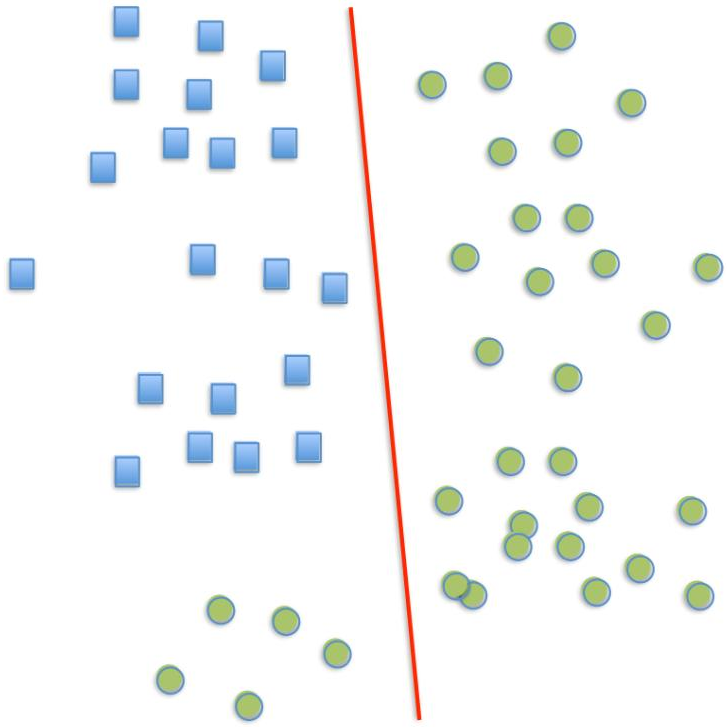
```
    def plot_loss(self):
        plt.plot(self.loss)
        plt.grid()
        plt.show()

if __name__ == '__main__':
    X_train = np.array([[-2, 4],[4, 1],[1, 6],[2, 4],[6, 2]])
    y_train = np.array([-1,-1,1,1,1])
    _, n_feature = X_train.shape
    model = Perceptron(n_feature=n_feature, n_iter=200, lr=1, tol=1.0e-3)
    model.train(X_train, y_train)
    print(f'Learned weights are {model.W}')
    y_pred =np.sign(model.predict(X_train))
    print(f'Predicted labels are {y_pred}')

    plt.figure()
    model.plot_loss()
```

Can we always separate the classes?

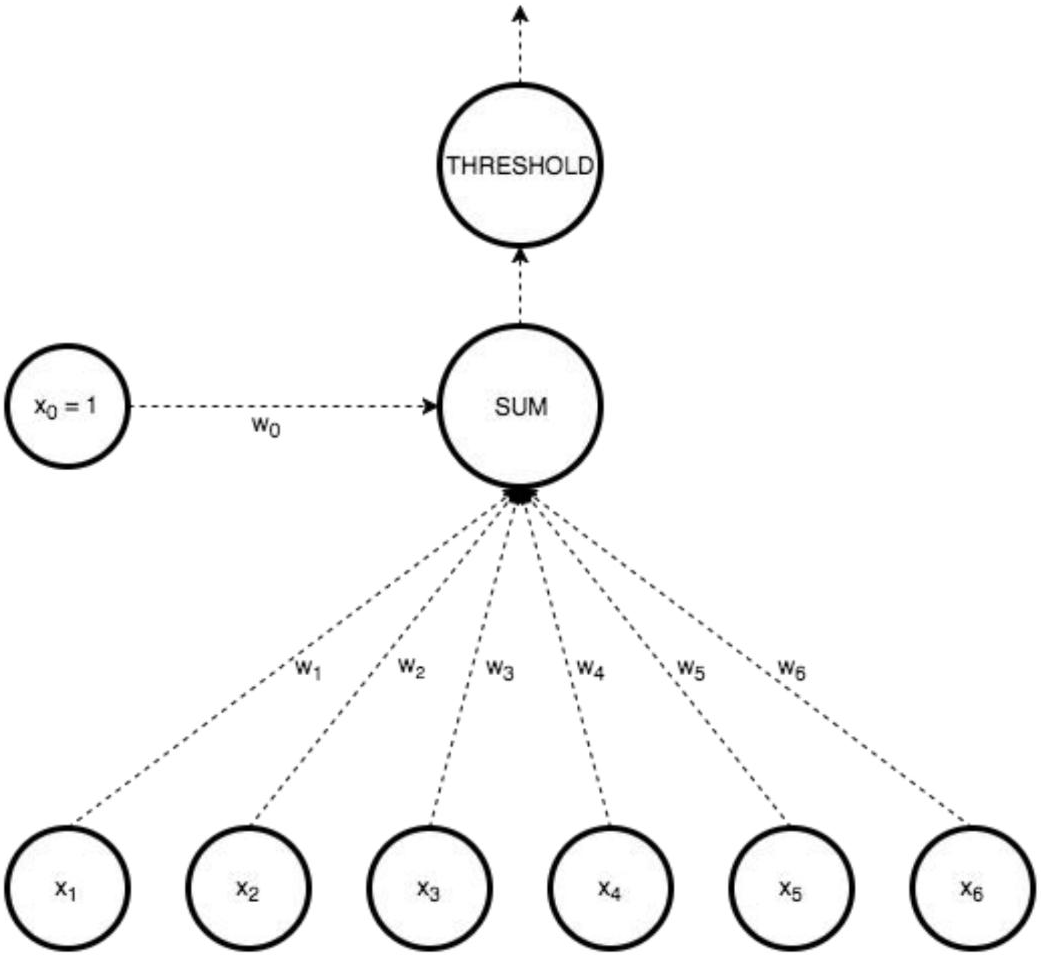
- If the problem is **linearly separable**, then perceptron can.



Can we always separate the classes?

- Causes of non perfect separation:
  - Model is too simple (perceptron)
  - Noises in the inputs (i.e., data attributes)
  - Simple features that do not account for all variations
  - Errors in data targets (mis-labellings)
- Should we make the model complex enough to have perfect sepration in the training data?

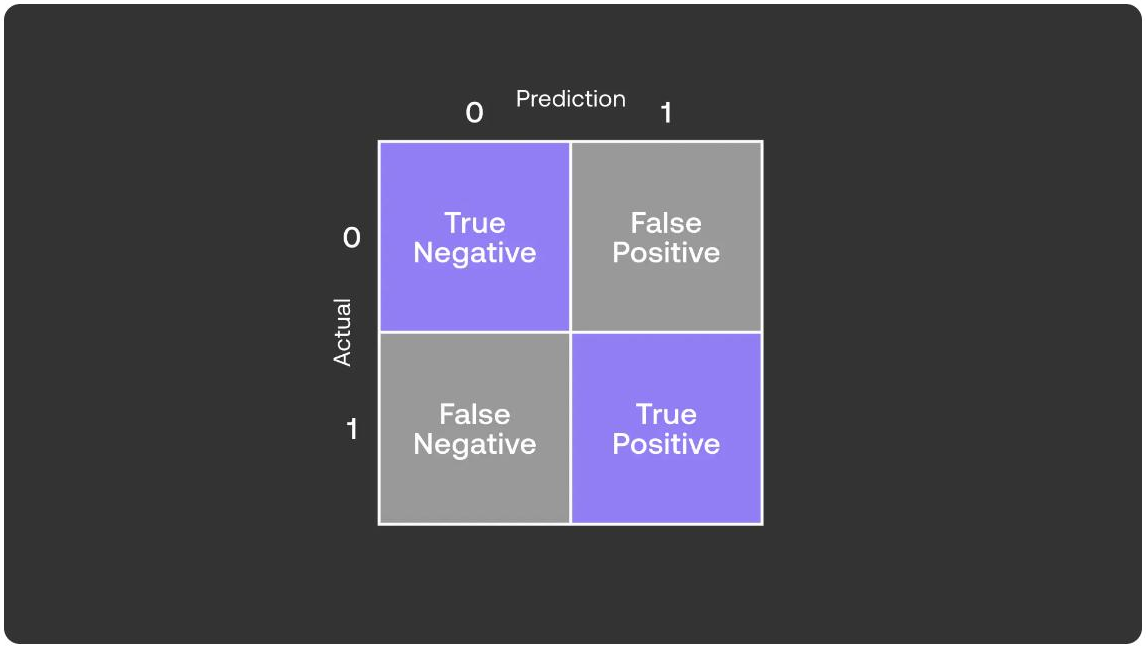
Summary of perceptron



Metrics

How to evaluate how good my classifier is? How is it doing on dog vs no-dog?

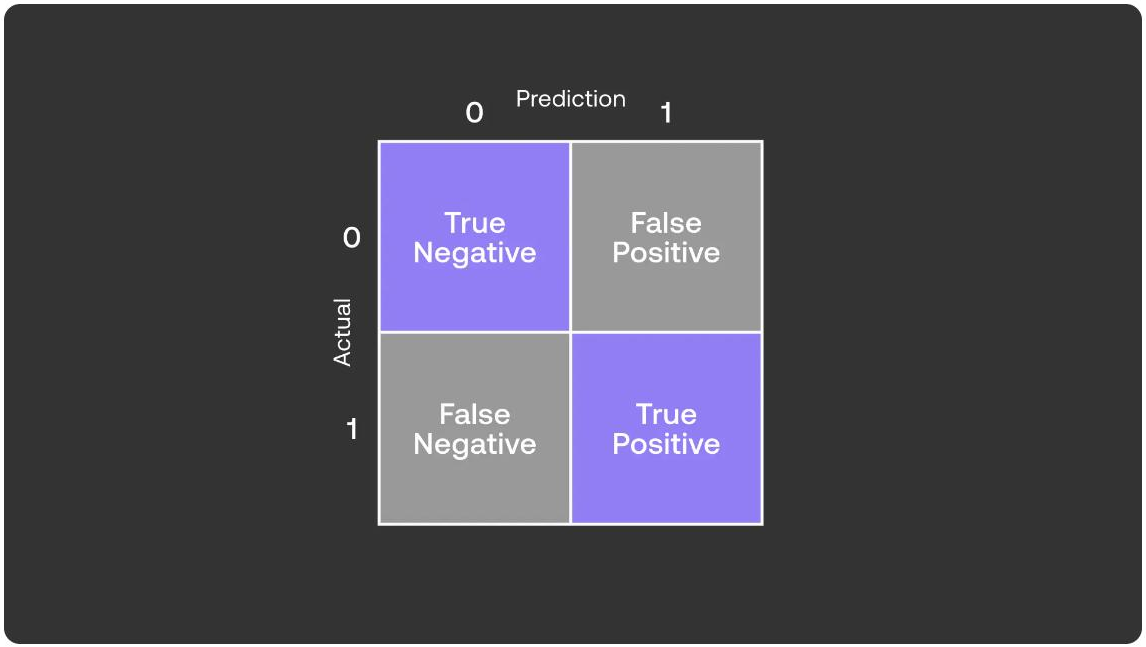
Metrics: Accuracy



- **Accuracy**: gives the percentage of correct classifications

$$A = \frac{TP + TN}{TP + FP + FN + TN}$$

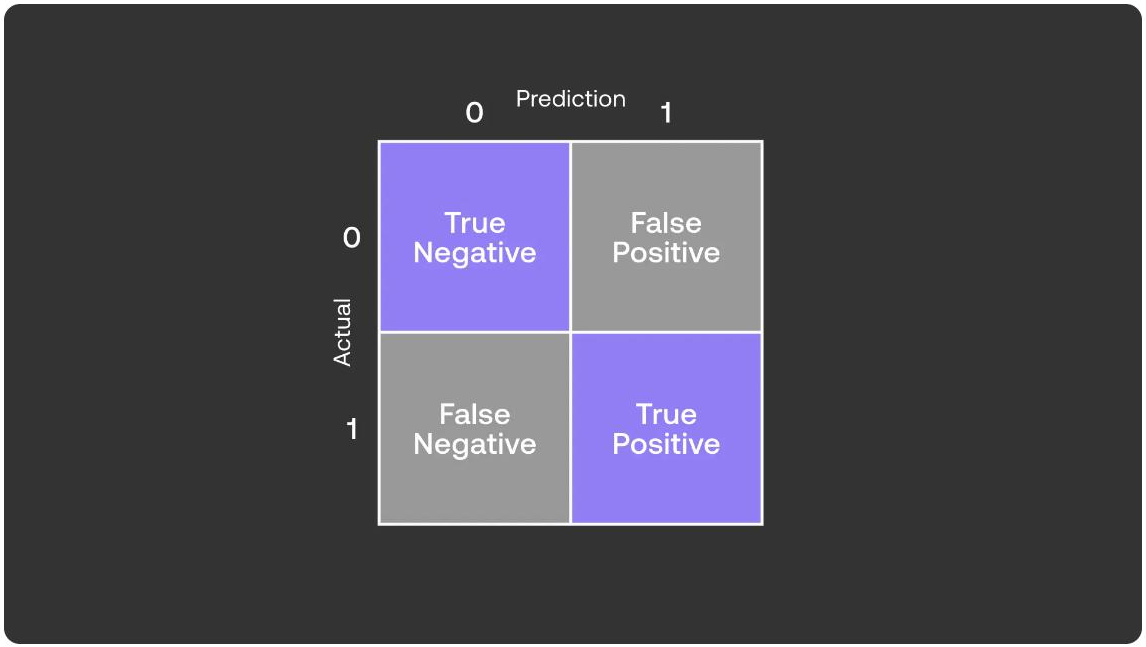
Metrics: Recall



- **Recall**: is the fraction of relevant instances that are retrieved

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all groundtruth instances}}$$

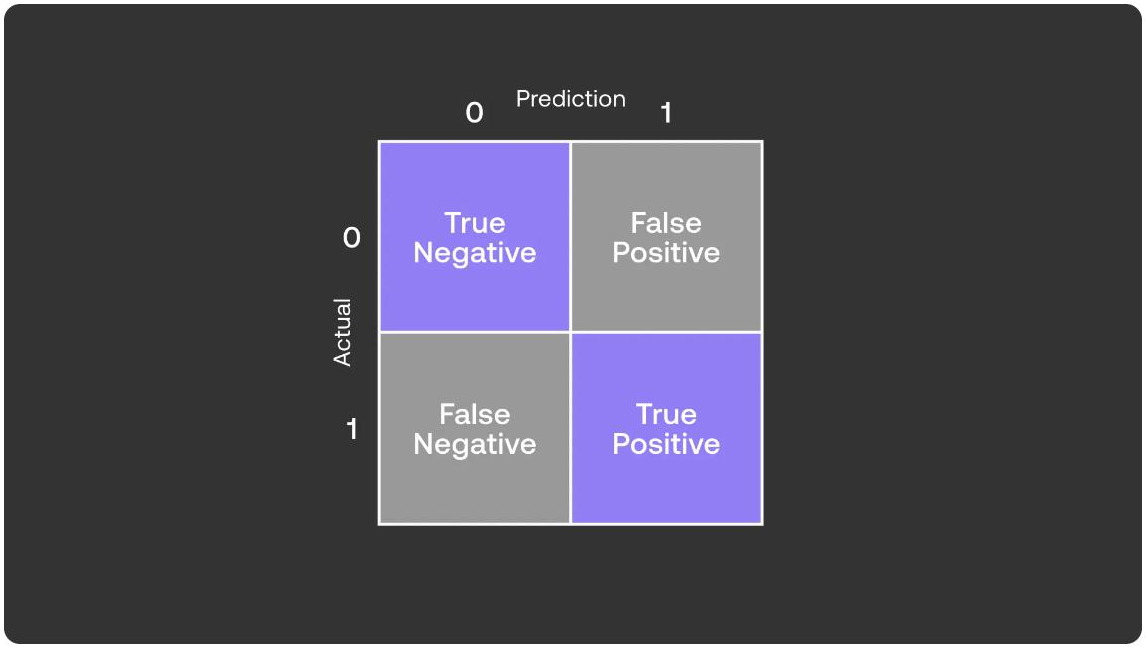
Metrics: Precision



- **Precision**: is the fraction of retrieved instances that are relevaent

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all predicted true}}$$

Metrics: F1 score



- **F1 score**: harmonic mean of precision and recall

$$F1 = 2\frac{P \cdot R}{P + R}$$

Metrics vs loss

- Metrics on a dataset is what we care about (performance)
- We typically **cannot** directly optimize for the metrics

- Loss function should reflect the problem we are solving. We then hope it will yield models that will do well on our dataset

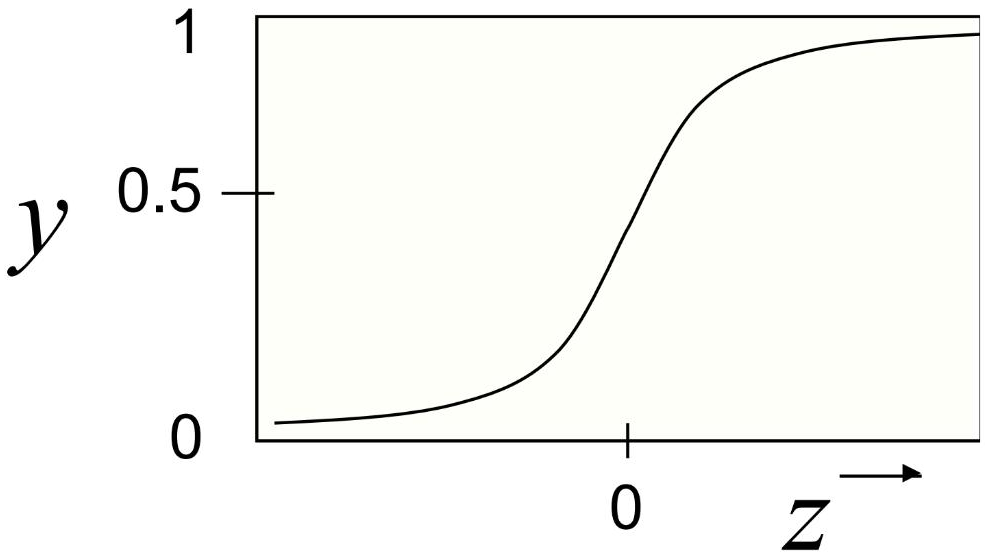
Logistic regression

- An alternative: replace $\text{sign}(\cdot)$ with the **sigmoid** (or **logistic**) function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y = \sigma(z) = \sigma\left(w^T \bar{x}\right)$$



Logistic regression

- We assumed a particular functional form: sigmoid applied to a linear function of the data

$$y = \sigma(z) = \sigma\left(w^T \bar{x}\right)$$

> The sigmoid is defined as
>
> $$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

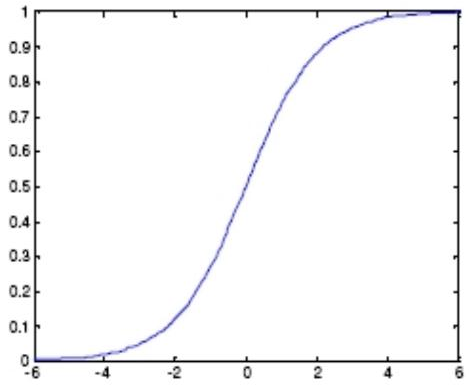
- One parameter per data dimension (feature) and the bias
- Features can be discrete or continuous
- Output of the model: value $y \in [0, 1]$
- Allows for gradient-based learning of the parameters

Shape of the logistic function

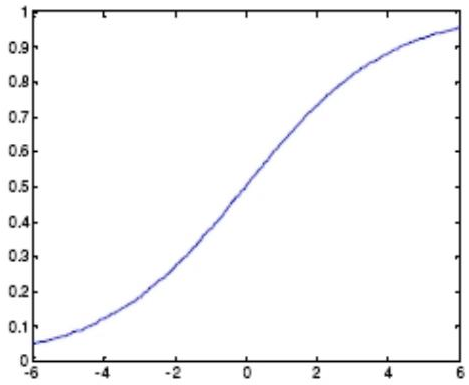
- Let's look at how modifying w changes the shape of the function
- 1D example:

$$y(x) = \sigma\left(w_0 + w_1 x\right)$$

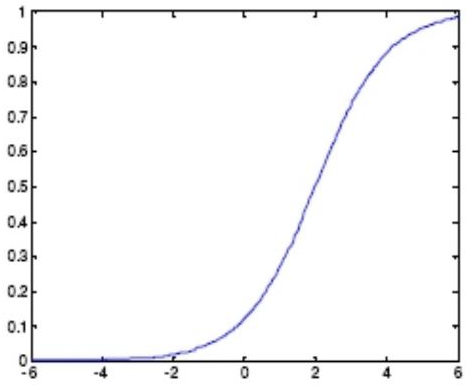
$w_0 = 0, w_1 = 1$



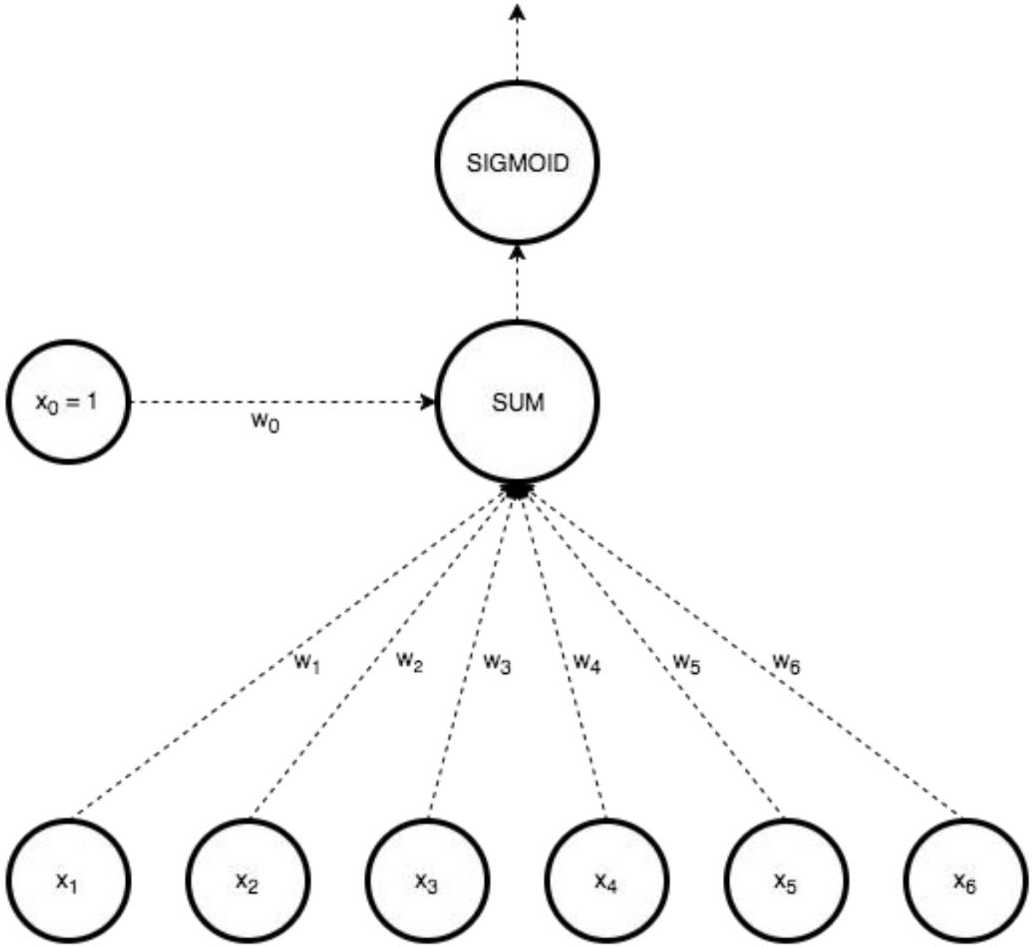
$w_0 = 0, w_1 = 0.5$



$w_0 = -2, w_1 = 1$



Logistic regression model



Detour: Probability

Probability

- **Probability** gives us a formal way of reasoning about our level of certainty.
- If we are **completely sure** that the image depicts a cat, we say that the probability that the corresponding label y is "cat", denoted $p(y = \text{cat}) = 1$.

Images of varying resolutions (10×10, 20×20, 40×40, 80×80, and 160×160 pixels).

Probability

- If we had no evidence to suggest that $y = \text{cat}$ or that $y = \text{dog}$, then we might say that the two possibilities were equally likely expressing this as $p(y = \text{cat}) = p(y = \text{dog}) = 0.5$.

Images of varying resolutions (10×10, 20×20, 40×40, 80×80, and 160×160 pixels).

Probability

- If we were reasonably confident, but not sure that the image depicted a cat, we might assign a probability $0.5 < p(y = \text{cat}) < 1$.

Images of varying resolutions (10×10, 20×20, 40×40, 80×80, and 160×160 pixels).

Basic propability theory

- We have some value of interest, but we are uncertain about the outcome.
- So probability is a **flexible language for reasoning** about our level of certainty, and it can be applied effectively in a broad set of contexts.

- Say that we cast a die and want to know what the chance is of seeing a 1 rather than another digit.
- If the die is fair, all the six outcomes $\{1, \ldots, 6\}$ are equally likely to occur, and thus we would see a 1 in one out of six cases. Formally we state that 1 occurs with probability 1/6.

Basic propability theory

- For a real die that we receive from a factory, we might not know those proportions and we would need to check whether it is tainted.
- One natural approach for each value is to take the individual count for that value and to divide it by the total number of tosses. This gives us an estimate of the probability of a given event.
- The law of large numbers tell us that as the number of tosses grows this estimate will draw closer and closer to the true underlying probability.

Axioms of probability theory

- When dealing with the rolls of a die, we call the set $S = 1, 2, 3, 4, 5, 6$ the **sample space** or **outcome space**, where each element is an **outcome**.
- An **event** is a set of outcomes from a given sample space.
- For instance, "seeing a 5" ($\{5\}$) and "seeing an odd number" ($\{1, 3, 5\}$) are both valid events of rolling a die.
- Note that if the outcome of a random experiment is in event A, then event A has occurred.
  - That is to say, if 3 dots faced up after rolling a die, since $3 \in \{1, 3, 5\}$, we can say that the event "seeing an odd number" has occurred.

Axioms of probability theory

- Formally, probability can be **thought of as a function** that maps a set to a real value.
- The probability of an event A in the given sample space S, denoted as $p(A)$, satisfies the following **properties**:
  - For any event A, its probability is never negative, i.e., $p(A) \geq 0$;
  - Probability of the entire sample space is 1, i.e., $p(S) = 1$;
  - For any countable sequence of events $A_1, A_2, \ldots$ that are mutually exclusive ($A_i \cap A_j = \emptyset$ for all $i \neq j$), the probability that any happens is equal to the sum of their individual probabilities, i.e.,

$$p\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} p(A_i).$$

Random variable

- In our random experiment of casting a die, we introduced the notion of a random variable.
  - A **random variable** can be pretty much any quantity and is not deterministic. It could take one value among a set of possibilities in a random experiment.
- Consider a random variable X whose value is in the sample space $S = \{1, 2, 3, 4, 5, 6\}$ of rolling a die.
  - We can denote the event "seeing a 5" as $\{X = 5\}$ or $X = 5$, and its probability as $p(\{X = 5\})$ or $p(X = 5)$.
  - By $p(X = a)$, we make a distinction between the random variable X and the values (e.g., a) that X can take.

Random variable

- For a compact notation, on one hand, we can just denote $p(X)$ as the distribution over the random variable X: the distribution tells us the probability that X takes any value.
  - On the other hand, we can simply write $p(a)$ to denote the probability that a random variable takes the value a.
- Since an event in probability theory is a set of outcomes from the sample space, we can specify a range of values for a random variable to take.
  - For example, $p(1 \leq X \leq 3)$ denotes the probability of the event $1 \leq X \leq 3$, which means $\{X = 1, 2, or, 3\}$.
  - Equivalently, $p(1 \leq X \leq 3)$ represents the probability that the random variable X can take a value from $\{1, 2, 3\}$

Dealing with multiple random variables

- Very often, we will want to consider **more than one random variable** at a time.
  - For instance, model the relationship between diseases and symptoms.
  - Given a disease and a symptom, say “flu” and “cough”, either may or may not occur in a patient with some probability.

- While we hope that the probability of both would be close to zero, we may want to estimate these probabilities and their relationships to each other so that we may apply our inferences to effect better medical care.

Joint probability

- The first is called the **joint probability** $p(A = a, B = b)$.
  - Given any values a and b, the joint probability lets us answer, what is the probability that $A = a$ and $B = b$ simultaneously?

- Note that for any values a and b, $p(A = a, B = b) \leq p(A = a)$.
  - This has to be the case, since for $A = a$ and $B = b$ to happen, $A = a$ has to happen and $B = b$ also has to happen (and vice versa).
  - Thus, $A = a$ and $B = b$ cannot be more likely than $A = a$ or $B = b$ individually.

Conditional probability

- This brings us to an interesting **ratio**:

$$0 \leq \frac{p(A = a, B = b)}{p(A = a)} \leq 1$$

- We call this ratio a **conditional probability** and denote it by

$$p(B = b | A = a)$$

> It is the probability of $B = b$, provided that $A = a$ has occurred.

Questions

1. Consider a sample space S and events A_1 and A_2 in S. Which statement below is true?
   (a) $p(A_1) \geq 0$.
   (b) $p(A_1)$ or $p(A_2)$ can be greater than 1.
   (c) If $A_1 = S$, then $p(A_1) = 1$.
   (d) If $A_1 \cup A_2 = S$, then $p(A_1 \cup A_2) = 1$.

2. For the example of casting a die, which of the following is correct?
   (a) $p(1 \leq X \leq 3) = p(X = 1, 2, \text{ or}, 3)$
   (b) $p(1 \leq X \leq 3)$ does not equal to $p(X = 1, 2, \text{ or}, 3)$
   (c) $p(1 \leq X \leq 3) = p(X = 1) + p(X = 2) + p(X = 3)$
   (d) $p(1 \leq X \leq 3)$ does not equal to $p(X = 1) + p(X = 2) + p(X = 3)$

Questions

3. Consider to cast two dies, represented by a random X_1 for the first die and X_2 for the second die. Which of the following is correct?
   (a) $p(X_1 = 3, X_2 = 3) = \frac{1}{6}$
   (b) $p(X_1 = 3, X_2 = 3) = \frac{1}{36}$
   (c) $p(X_1 = 3 | X_2 = 3) = \frac{1}{6}$
   (d) $p(X_1 = 3 | X_2 = 3) = \frac{1}{36}$

Back to logistic regression

Probabilistic interpretation

- Consider two classes: use $\{0, 1\}$ **(outcome space)** instead of $\{-1, 1\}$
- If we have a value between 0 and 1, let's use it to model class probability

$$p(C = 1|x) = \sigma(w^T \bar{x}) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Substituting we have

$$p(C = 1|x) = \frac{1}{1 + \exp(-w^T \bar{x})}$$

Probabilistic interpretation

- Suppose we have two classes, how can we compute $p(C = 0|x)$?
- Use the marginalization property of probability

$$p(C = 0|x) + p(C = 1|x) = 1$$

- Thus

$$p(C = 0|x) = 1 - \frac{1}{1 + \exp(-w^T \bar{x})} = \frac{\exp(-w^T \bar{x})}{1 + \exp(-w^T \bar{x})}$$

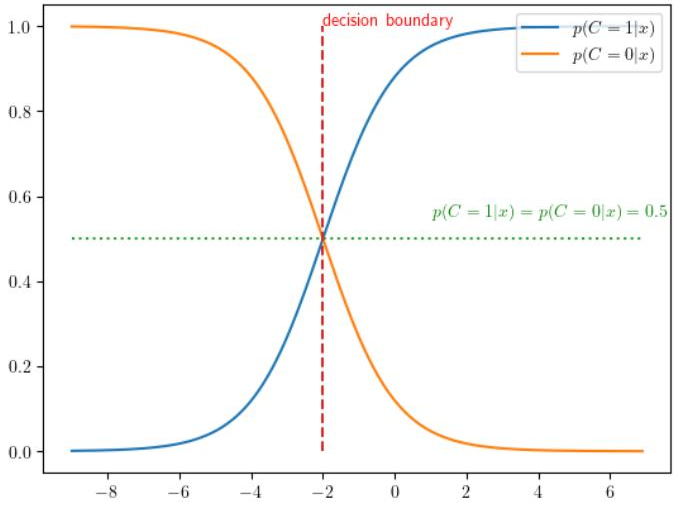
Decision boundary for logistic regression

- What is the decision boundary for logistic regression?

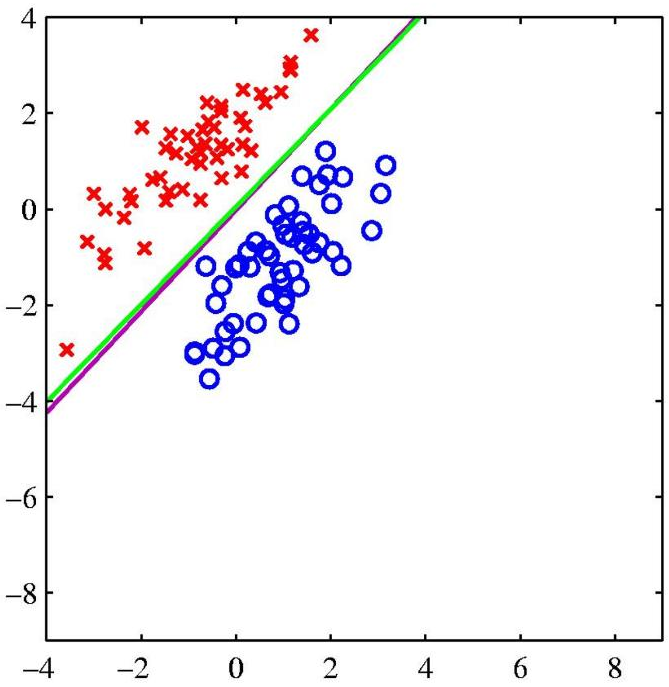
$$p(C = 1|x, w) = p(C = 0|x, w) = 0.5$$

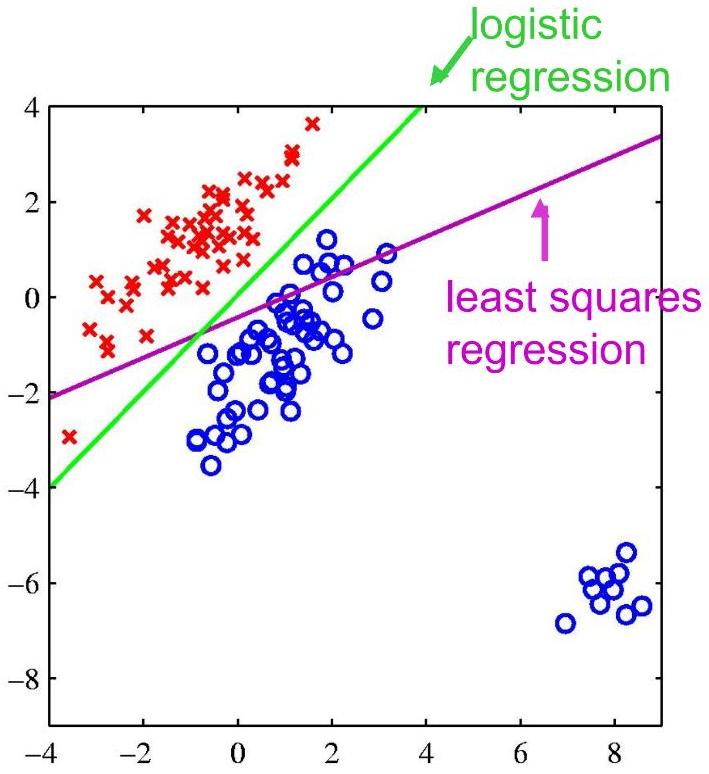
$$p(C = 1|x, w) = \sigma\left(w^T \bar{x}\right) = 0.5, \text{ where } \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Decision boundary: $w^T \bar{x} = w_0 + w_1 x_1 + \cdots + w_d x_d = 0$
- Logistic regression has a linear decision boundary



Logistic regression





- If the right answer is 1 and the model says 1.5, it loses, so it changes the boundary to avoid being "too correct" (tilts away from outliers)

Example

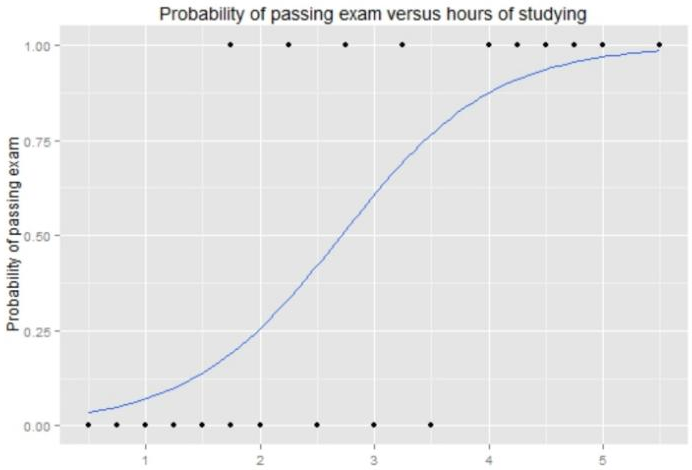
Problem: Given the number of hours a student spent learning, will (s)he pass the exam?

Training data (top row: $x^{(i)}$, bottom row: $t^{(i)}$)

Hours	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

Learn **w** for our model, i.e., logistic regression (coming up)

Make predictions:



Hours of study	Probability of passing exam
1	0.07
2	0.26
3	0.61
4	0.87
5	0.97

Learning?

- When we have a d-dim input $x \in \mathbb{R}^d$
- How should we learn the weights $w = (w_0, w_1, \cdots, w_d)$?
- We have a probabilistic model
- Let's use **maximum likelihood**

Detour: Likelyhood

Probability vs likelyhood

- You can estimate a probability of an event using the function that describes the **probability distribution and its parameters**.
  - For example, you can estimate the outcome of a fair coin flip by using the Bernoulli distribution and the probability of success $p(\text{success}) = 0.5$.
  - In this ideal case, you already know how the data is distributed.

> But the real world is messy. Often you don't know the exact parameter values, and you may not even know the probability distribution that describes your specific use case.
> Instead, you have to estimate the function and its parameters from the data.

- The **likelihood** describes the relative evidence that the data has a particular distribution and its associated parameters.

Likelyhood

- We can describe the **likelihood** as a function of an observed value of the data x, and the distributions' unknown parameter w.

$$L(x, w)$$

- In short, when estimating the probability, you go **from a distribution and its parameters to the event**.

$$\text{Probability:} \quad p(event|distribution)$$

- When estimating the likelihood, you go **from the data to the distribution and its parameters**.

$$\text{Likelyhood:} \quad L(distribution|data)$$

Likelyhood

- Recall that a coin flip is a Bernoulli trial, whose distribution can be described in the following function.

$$p(X = x) = p^x(1-p)^{1-x} \quad \text{where } x \in \{0, 1\}$$

> The probability p is a parameter of the function, representing the probability of being 1.

- To be consistent with the likelihood notation, we write down the formula for the likelihood function with w instead of p.

$$L(x, w) = w^x(1-w)^{1-x}$$

Likelyhood

- Let's say we throw the coin 3 times. It comes up heads the first 2 times. The last time it comes up tails. (Head: 1; Tail: 0).
- What is the likelihood that hypothesis A given the data?
- Now, we need a hypothesis about the parameter w.
- Assume that the coin is fair. The probability of obtaining heads is $w = 0.5$.

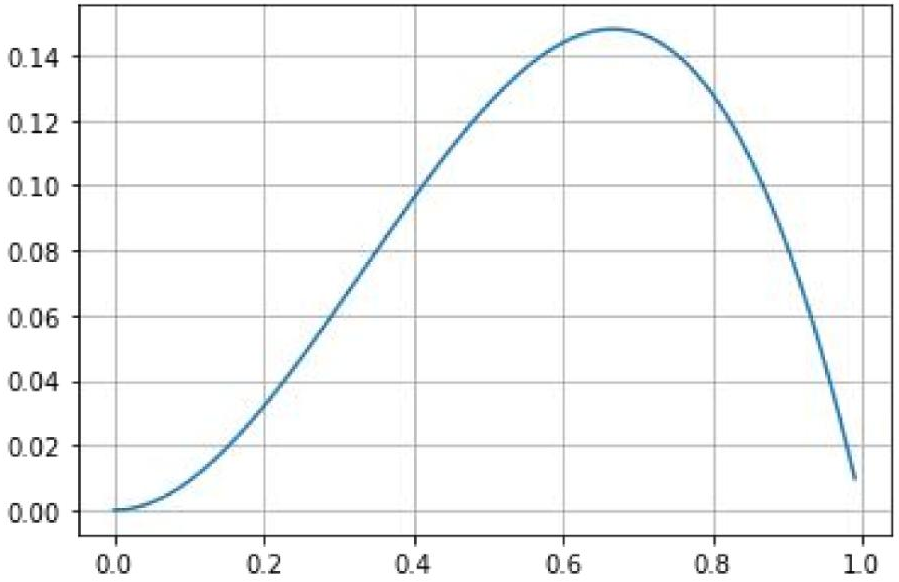
$$p(X = 1) = 0.5^1(1 - 0.5)^{1-1} = 0.5$$
$$p(X = 1) = 0.5^1(1 - 0.5)^{1-1} = 0.5$$
$$p(X = 0) = 0.5^0(1 - 0.5)^{1-0} = 0.5$$

- Multiplying all of these gives us the following value

$$\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$$

Likelyhood

- For three coin tosses with 2 heads, the plot would look like this by varying the parameter w from 0 to 1, for which the **likelihood is maximized** at 2/3.



- In other words: Given the fact that 2 of our 3 coin tosses landed up heads, it seems more likely that the true probability of getting heads is 2/3.

Maximum likelyhood estimation

- Mathematically, we can denote the **maximum likelihood estimation** as a function that results in optimal parameter w maximizing the likelihood.

$$w^* = \arg\max_w L(x, w) = \prod_{i=1}^{N} p(x^{(i)}, w)$$

- The variable x represents the range of examples drawn from the unknown data distribution, which we would like to approximate and N the number of examples.
- For most practical applications, **maximizing the log-likelihood** is often a better choice because the logarithm reduced operations by one level.
  - Multiplications become additions; powers become multiplications, etc.

$$w^* = \arg\max_w l(x, w) = \sum_{i=1}^{n} \log\Big(p(x^{(i)}, w)\Big)$$

Back to logistic regression

Conditional likelihood in logistic regression

- Assume $t \in \{0, 1\}$, we can write the probability distribution of each of our training points $p(t^{(1)}, \cdots, t^{(N)} | x^{(1)}, \cdots, x^{(N)}; w)$
- Assuming that the training examples are **sampled IID**: independent and identically distributed, we can write the **likelihood function**:

$$L(w) = p(t^{(1)}, \cdots, t^{(N)} | x^{(1)}, \cdots, x^{(N)}; w) = \prod_{i=1}^{N} p(t^{(i)} | x^{(i)}; w)$$

- We can write each probability as (will be useful later):

$$\begin{aligned} p(t^{(i)} | x^{(i)}; w) = & \quad p(C = 1 | x^{(i)}; w)^{t^{(i)}} p(C = 0 | x^{(i)}; w)^{1 - t^{(i)}} \\ = & \quad \Big(p(C = 1 | x^{(i)}; w) \Big)^{t^{(i)}} \Big(1 - p(C = 1 | x^{(i)}; w) \Big)^{1 - t^{(i)}} \end{aligned}$$

Maximizing the likelihood

- We can learn the model by maximizing the likelihood

$$\max_w L(w) = \max_w \prod_{i=1}^{N} p(t^{(i)}|x^{(i)}; w)$$

> $$L(w) = \prod_{i=1}^{N} p(t^{(i)}|x^{(i)}; w) \quad \text{(likelihood)}$$
> $$= \prod_{i=1}^{N} \left(p(C=1|x^{(i)})\right)^{t^{(i)}} \left(1 - p(C=1|x^{(i)})\right)^{1-t^{(i)}}$$

Loss function

- Easier to maximize the **log likelihood** $\log l(w)$
- We can convert the maximization problem into **minimization** so that we can write the **loss function**:

$$
\begin{aligned}
\ell_{\log}(w) &= -\log L(w) \\
&= -\sum_{i=1}^{N} \log p(t^{(i)}|x^{(i)}; w) \\
&= -\sum_{i=1}^{N} t^{(i)} \log p(C=1|x^{(i)}; w) - \sum_{i=1}^{N} (1-t^{(i)}) \left(1 - \log p(C=1|x^{(i)}; w)\right) \\
&= -\sum_{i=1}^{N} t^{(i)} \log y(x^{(i)}, w) - \sum_{i=1}^{N} (1-t^{(i)}) \left(1 - \log y(x^{(i)}, w)\right)
\end{aligned}
$$

- It's a convex function of w. Can we get the global optimum?

Gradient descent

$$\min_w l_{\log}(w) = \min_w \left\{ -\sum_{i=1}^{N} t^{(i)} \log p(C=1|x^{(i)};w) - \sum_{i=1}^{N} (1-t^{(i)}) \log \left(1 - p(C=1|x^{(i)};w)\right) \right\}$$

- **Gradient descent**: iterate and at each iteration compute steepest direction towards optimum, move in that direction, learning rate λ

$$w \leftarrow w - \lambda \nabla l(w)$$

> $$\nabla l(w) = \left[\frac{\partial \ell(w)}{\partial w_0}, \cdots, \frac{\partial \ell(w)}{\partial w_d} \right]^T$$

- But where is w?

$$p(C=1|x) = \frac{1}{1+\exp\left(-w^T \bar{x}\right)}$$

Let's compute the gradient

- Check the loss for each example

$$
\begin{aligned}
l_i(w) &= -t^{(i)} \log p(C=1|x^{(i)};w) - (1-t^{(i)}) \log \left(1 - p(C=1|x^{(i)};w)\right) \\
&= t^{(i)} \log \left(1+\exp(-w^T \bar{x}^{(i)})\right) + \left(1-t^{(i)}\right) \log \left(\frac{1+\exp(-w^T \bar{x}^{(i)})}{\exp(-w^T \bar{x}^{(i)})}\right) \\
&= t^{(i)} \log \left(1+\exp(-w^T \bar{x}^{(i)})\right) + \left(1-t^{(i)}\right) \log \left(1+\exp(-w^t \bar{x}^{(i)})\right) + \left(1-t^{(i)}\right) w^T \bar{x}^{(i)} \\
&= \log \left(1+\exp(-w^T \bar{x}^{(i)})\right) + \left(1-t^{(i)}\right) w^T \bar{x}^{(i)}
\end{aligned}
$$

- Now it's easy to take derivatives

$$
\begin{aligned}
\nabla l_i(w) &= \frac{-\bar{x}^{(i)} \exp(-w^T \bar{x}^{(i)})}{1+\exp(-w^T \bar{x}^{(i)})} + \left(1-t^{(i)}\right) \bar{x}^{(i)} \\
&= \bar{x}^{(i)} \left[\frac{-\exp(-w^T \bar{x}^{(i)}) + 1 + \exp(-w^T \bar{x}^{(i)})}{1+\exp(-w^T \bar{x}^{(i)})} - t^{(i)}\right] \\
&= \bar{x}^{(i)} \left[y(x^{(i)}, w) - t^{(i)}\right] = -\left[t^{(i)} - y(x^{(i)}, w)\right] \bar{x}^{(i)}
\end{aligned}
$$

The gradient for a batch

- The gradient for a batch

$$\nabla l(w) = -\sum_{i=1}^{N} \left[t^{(i)} - y(x^{(i)}, w) \right] \bar{x}^{(i)}$$

- In a matrix form, it becomes

$$\nabla l(w) = -\mathbf{X}^T (\mathbf{t} - \mathbf{y})$$

> $$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t^{(1)} \\ \vdots \\ t^{(N)} \end{bmatrix}, \text{ and } \mathbf{y} = \sigma(\mathbf{X}w)$$

- This is all there is to learning in logistic regression. Simple, huh?

Regularization

- We can define priors on parameters w
- This is a form of regularization
- Helps avoid large weights and **overfitting**

$$\min_w \quad \tilde{l}(w) = -\log\left(p(w)\prod_i p(t^{(i)}|x^{(i)};w)\right)$$

- What's $p(w)$?

Regularized logistic regression

- For example, define prior: normal distribution, zero mean and identity covariance

$$w \sim N(0, \alpha^{-1} I)$$

$$p(w) = \prod_{j=1}^{d+1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_j^2}{2\sigma^2}\right) \quad \text{where } \sigma^2 = \alpha^{-1}$$

- This prior pushes parameters towards zero. Plug above and obtain

$$\tilde{l}(w) = l(w) + \sum_{j=1}^{d+1} \left(\frac{1}{2}\log(2\pi\sigma^2) + \frac{w_j^2}{2\sigma^2}\right)$$

- Including this prior the new gradient is

$$\nabla\tilde{l}(w) = \nabla l(w) + \alpha w$$

> α is the importance of the regularization, and it's a hyper-parameter

Use of validation set

- Tuning hyper-parameters:
  - **Never Use Test Data For Tuning The Hyper-Parameters**
  - We can divide the set of training examples into two disjoint sets: training and validation
  - Use the first set (i.e., training) to estimate the weights w for different values of α
  - Use the second set (i.e., validation) to estimate the best α, by evaluating how well the classifier does on this second set
  - This tests how well it generalizes to unseen data

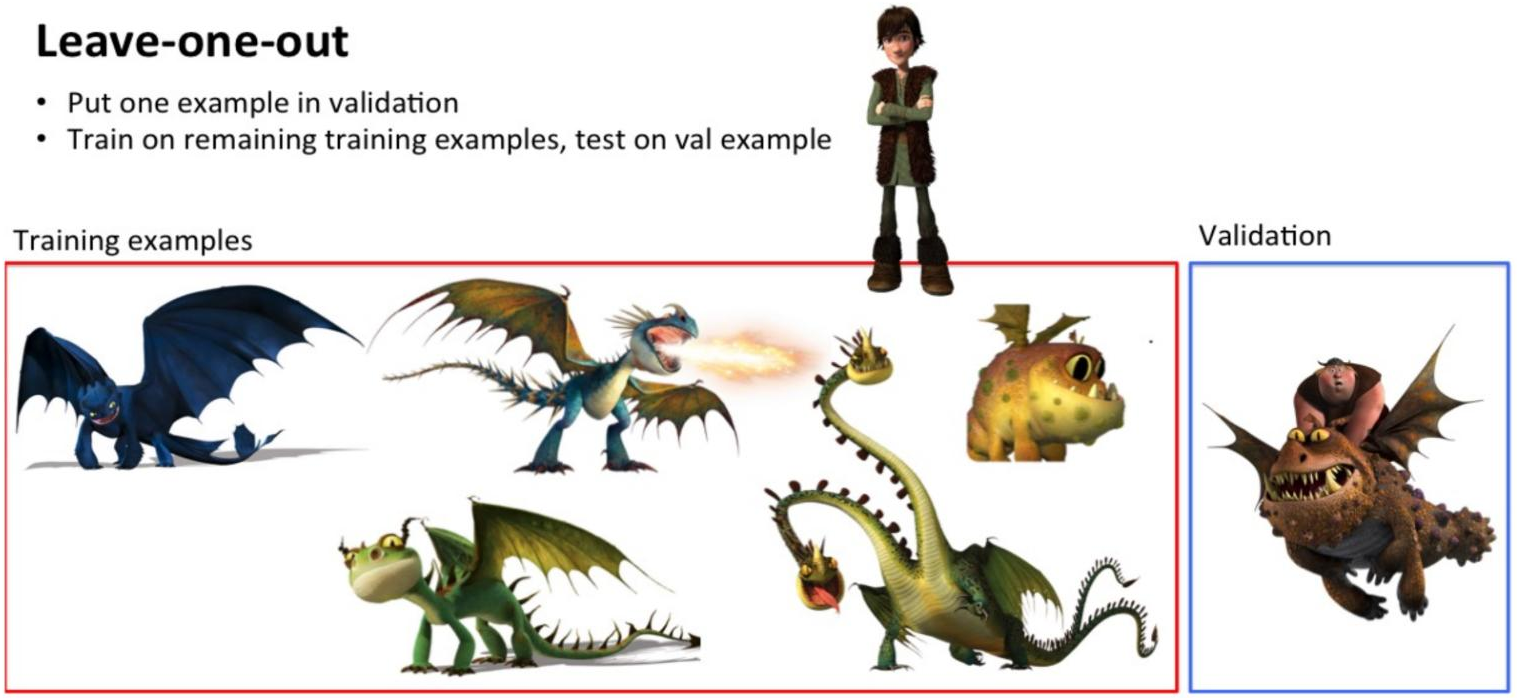
Cross-validation

- **Leave-p-out cross-validation**:
  - We use p observations as the validation set and the remaining observations as the training set.
  - This is repeated on all ways to cut the original training set.
  - It requires $\mathcal{C}_N^p$ for a set of N examples

- **Leave-1-out cross-validation**: When $p = 1$, does not have this problem

Cross-validation (with pictures)

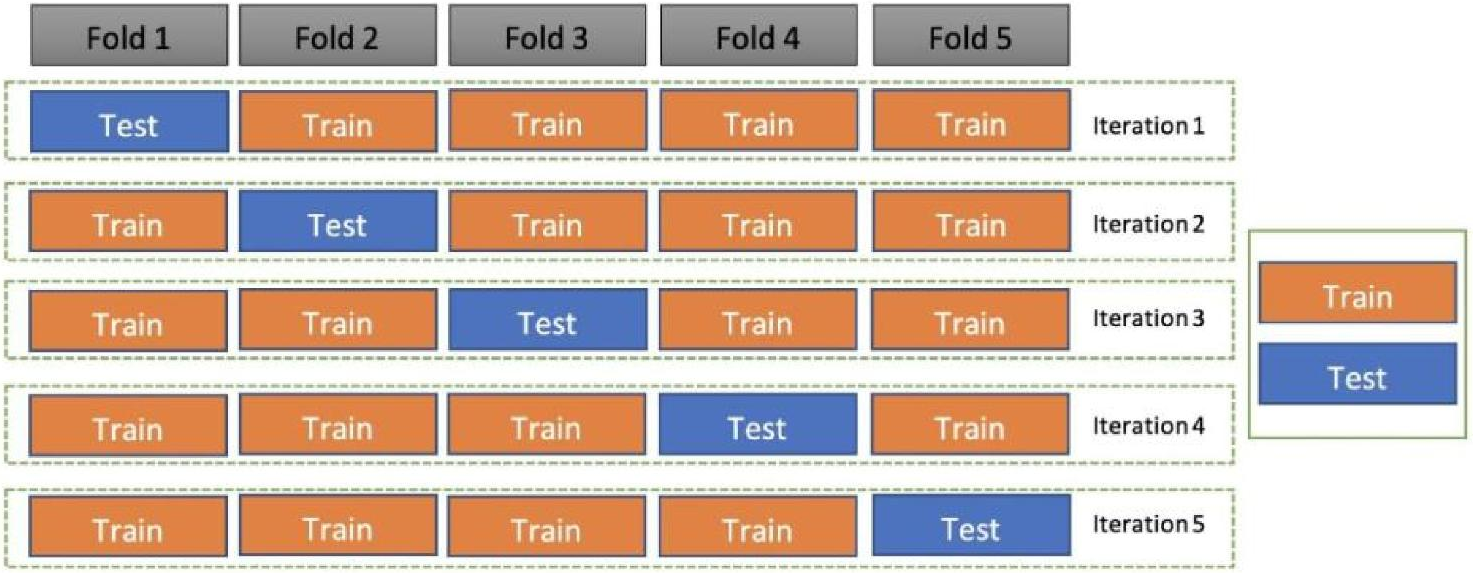
- Train Your Model:
  - Leave-one-out cross-validation:



Cross-validation

- **k-fold cross-validation**:
  - The training set is randomly partitioned into k equal size subsamples.
  - Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data.
  - The cross-validation process is then repeated k times (the folds).
  - The k results from the folds can then be averaged (or otherwise combined) to produce a single estimate

k-fold cross-validation



- The image above shows 5 folds and hence, 5 iterations.
- In each teration, one fold is the validation set and the other $k-1$ sets (4 sets) are the training set.
- To get the final accuracy, you need to take the accuracy of the k-models validation data.

BGD algorithm for logistic regression

```
import numpy as np
import matplotlib.pyplot as plt

class LogisticRegression:

    def __init__(self, n_feature = 1, n_iter = 200, lr = 1e-3, tol = None):
        self.n_iter = n_iter    # Maximum interation steps
        self.lr = lr            # Learning rate
        self.tol = tol          # Threshold for stopping iteration
        self.W = np.random.random(n_feature + 1) * 0.05          # Molel parameters
        self.loss = []          # The loss value

    def _linear_tf(self, X):
        return X @ self.W
```

BGD algorithm for logistic regression

```
def _sigmoid(self, z):
    out = 1./(1.+ np.exp(-z))
    return out

def _predict_probality(self, X):
    z = self._linear_tf(X)
    return self._sigmoid(z)

def _loss(self, y, y_pred):
    epsilon = 1e-5
    loss = - np.mean( y * np.log(y_pred + epsilon) +  (1-y) * np.log(1- y_pred + epsilon) )
    return loss


def _gradient(self, X, y, y_pred):
    return  -(y - y_pred) @ X / y.size
```

BGD algorithm for logistic regression

```
def _preprocess_data(self, X):
    m, n = X.shape
    X_ = np.empty([m, n+1])
    X_[:, 0] = 1
    X_[:,1:] = X

    return X_

def batch_update(self, X, y):

    if self.tol is not None:
        loss_old = np.inf

    for iter in range(self.n_iter):
        y_pred = self._predict_probality(X)
        loss = self._loss(y, y_pred)
        self.loss.append(loss)
```

BGD algorithm for logistic regression

```
        if self.tol is not None:
            if np.abs(loss_old - loss) < self.tol:
                break
            loss_old = loss

        grad = self._gradient(X, y, y_pred)
        self.W = self.W - self.lr * grad

    def train(self, X_train, y_train):
        X_train = self._preprocess_data(X_train)
        self.batch_update(X_train, y_train)

    def predict(self, X):
        X = self._preprocess_data(X)
        y_pred = self._predict_probality(X)
        return np.where(y_pred >=0.5, 1, 0)
```

Logistic regression wrap-Up

- Advantages:
  - Easily extended to multiple classes
  - Natural probabilistic view of class predictions
  - Quick to train
  - Fast at classification
  - Good accuracy for many simple data sets
  - Resistant to overfitting
  - Can interpret model coefficients as indicators of feature importance
- Less good:
  - Linear decision boundary (too simple for more complex problems)