COMP30027 MACHINE LEARNING TUTORIAL

Workshop - 7

Understanding Logistic Regression and Ensemble Methods

- Logistic regression
- Gradient ascent for logistic regression
- Ensemble methods (stacking, bagging, boosting)

Imagine we need a model to predict whether a tumour is cancerous based on its size, shape, and other characteristics.
The outcome is binary: **0 = not cancer**, **1 = cancer**

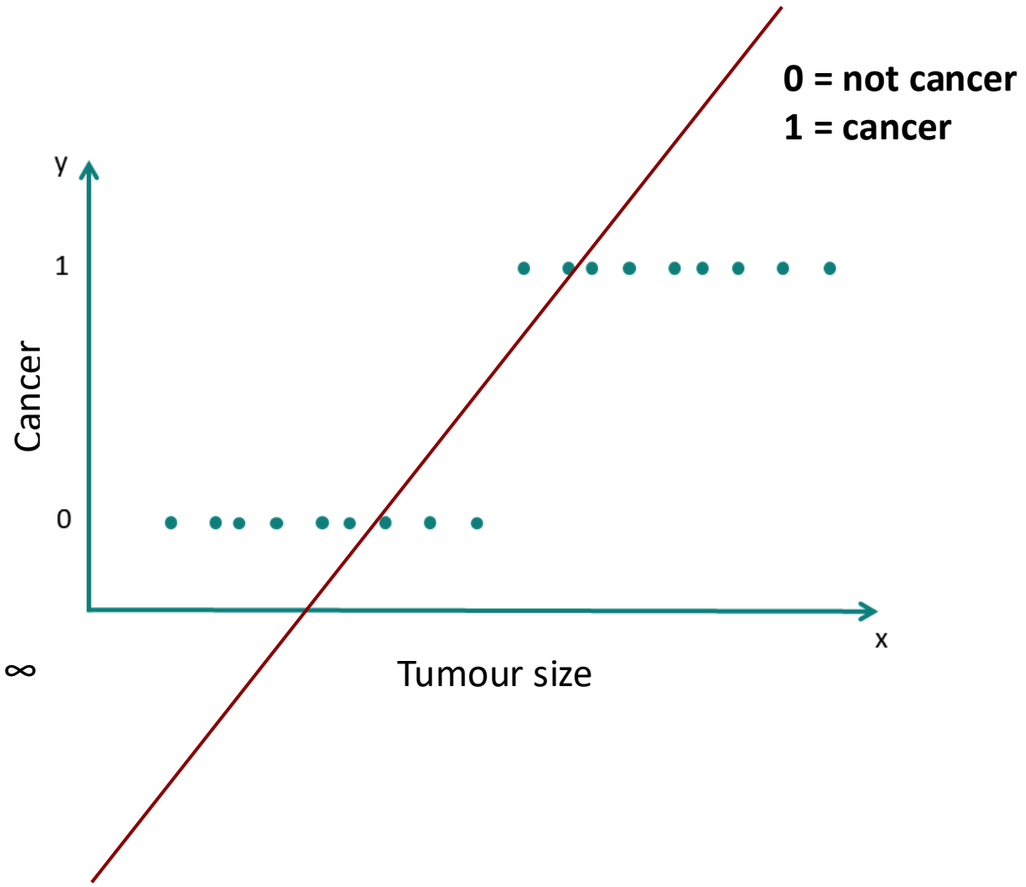
Let us represent the linear relationship between outcome (cancer/ not cancer) and features (size, shape, etc) as:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w^T x + b$$

Can we use linear regression to model this?

No, you might get y (outcome) values like **-5**, **5.3, or** any values between - ∞ to + ∞

So, we need a function that maps output to **between 0 and 1!**

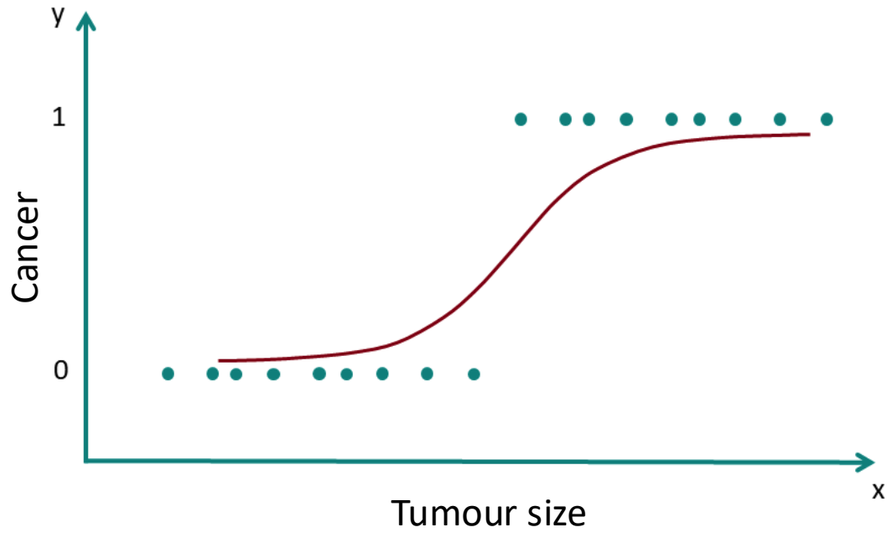


Sigmoid Function (S-curve)

0 = not cancer
1 = cancer

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

Where, $z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w^T x + b$



- The sigmoid function transforms linear predictions into probabilities. It creates a smooth S-curve.

- It takes any number from -∞ to +∞ and squashes it to a value between 0 and 1.

So, we model:

$$\hat{y} = \mathrm{P}(\mathrm{y} \mid \mathrm{x}) = \sigma(\mathrm{w}_0 + \mathrm{w}_1\mathrm{x}_1 + \mathrm{w}_2\mathrm{x}_2 + \cdots + \mathrm{w}_\mathrm{n}\mathrm{x}_\mathrm{n}) = \sigma(\mathrm{w}^T\mathrm{x})$$

We now have a model that gives us a **probability**!

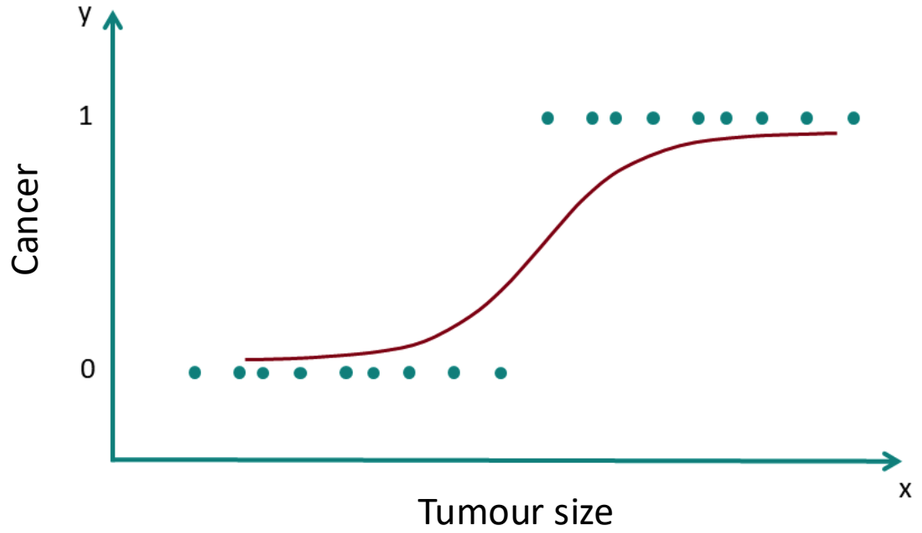
Logistic Regression Fundamentals

0 = not cancer
1 = cancer

Logistic regression helps us **predict the probability** of a **binary outcome** using input features. It models probability using the sigmoid function.

Unlike linear regression, outputs are constrained between 0 and 1.

This makes logistic regression perfect for classification tasks.



If $\hat{y} \geq 0.5$, we predict class **1**.
If $\hat{y} < 0.5$, we predict class **0**.

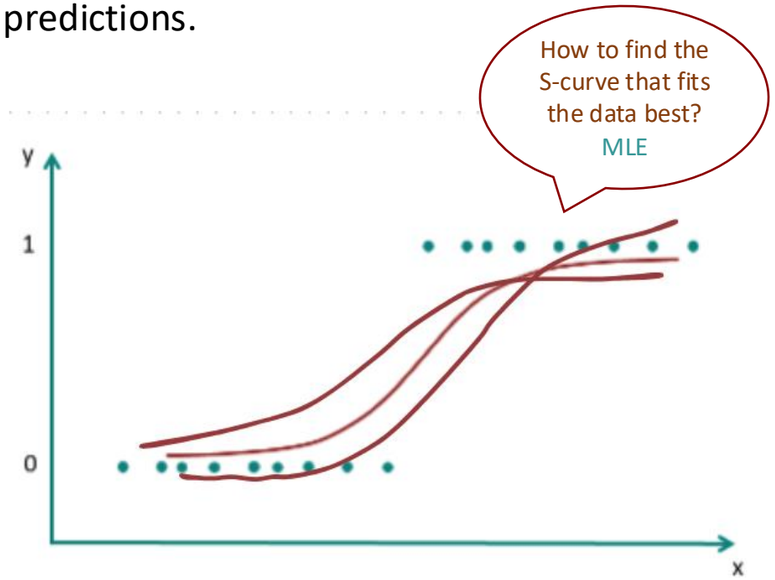
Where, $\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n) = \sigma(w^Tx)$ and 0.5 is the default threshold

So far, we know how to use logistic regression to predict. But how does it *learn* what weights to use for the model?

Maximum Likelihood Estimation (MLE)

The goal of training is to **find the best weights** w = [w_1,w_2,...,w_n] so that the predicted probabilities match the actual labels as closely as possible.

We use **likelihood** — which measures how probable the observed data is given the model's predictions.



For one test instance x:

If y=1, let the predicted probability be $\hat{y}$
If y=0, then the probability is 1− $\hat{y}$

So, we can write a unified formula:

$$P(y \mid x) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

Given a dataset of m training examples, the **likelihood** is the product of the probabilities for each example:

$$L(w) = \prod_{i=1}^{m} \hat{y}^{(i)y^{(i)}}(1-\hat{y}^{(i)})^{1-y^{(i)}}$$

We take the product because the probability of each training instance is independent of the others.

But products of small numbers (the probability values are between 0 and 1) can get numerically hard to compute, so we take the **log-likelihood** to ease mathematical computation:

$$\log L(w) = \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

$y^{(i)}$ is the ground truth of the i^{th} training instance

$\hat{y}^{(i)}$ is the predicted value for the i^{th} training instance

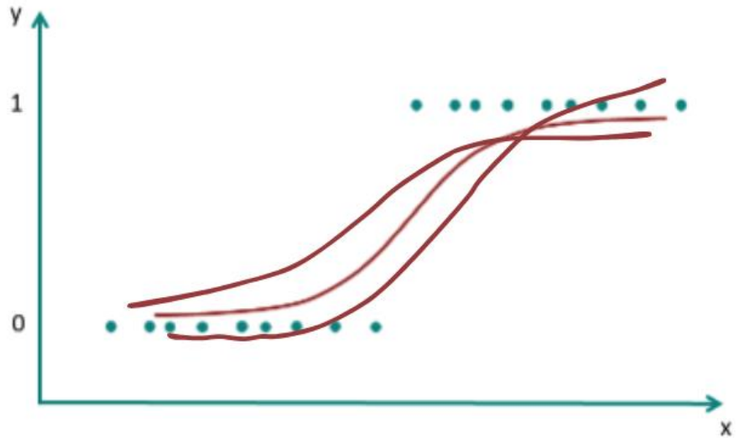
We know: $\hat{y} = \sigma(w^T x) = \sigma(w \cdot x)$

$$\log L(w) = \sum_{i=1}^{m} \left[y^{(i)} \log \sigma(w \cdot x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w \cdot x^{(i)})) \right]$$

This is the **log-likelihood function**, and we want to **maximize** it (or minimize the negative of it) to find the best weights.

There are two ways to do this:

- **Gradient Descent (to minimize negative log-likelihood)**
- **Gradient Ascent (to maximize likelihood)**



Gradient Descent for Logistic Regression

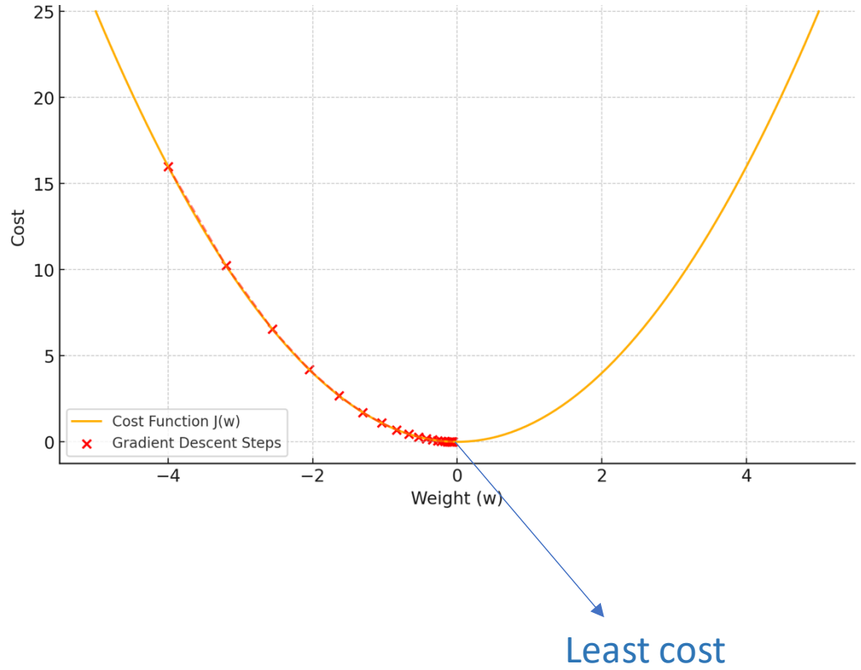
Goal: Update weights to minimize negative log-likelihood

Define **cost function** as the **negative log-likelihood**:

$$J(w) = -\sum_{i=1}^{m} \left[y^{(i)} \log \sigma(w \cdot x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w \cdot x^{(i)})) \right]$$

The gradient is:

$$\nabla_w J(w) = -\sum_{i=1}^{m} \left[y^{(i)} - \sigma(w \cdot x^{(i)}) \right] x^{(i)}$$



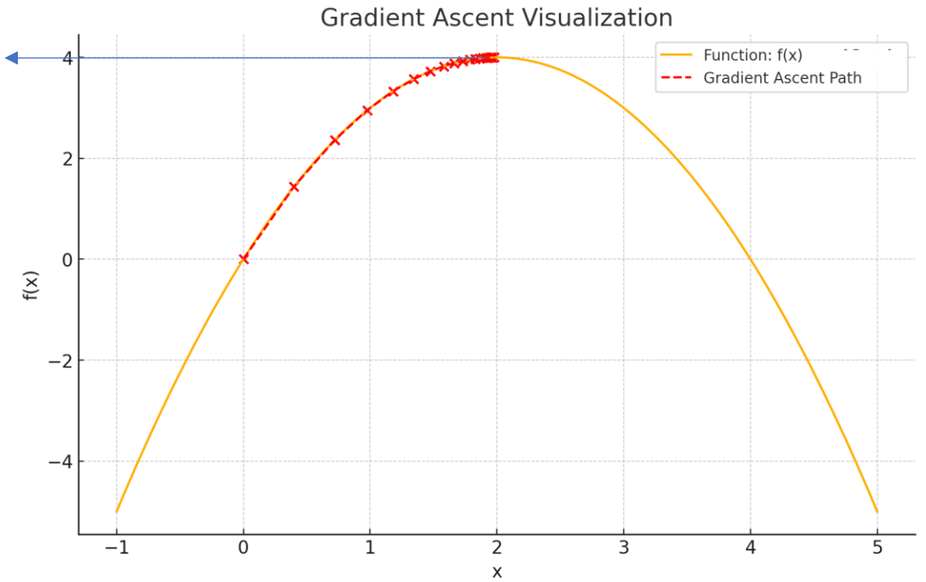
Gradient Descent Update Rule:

$$w := w - \alpha \cdot \sum_{i=1}^{m} \left[\hat{y}^{(i)} - y^{(i)} \right] x^{(i)}$$

- $\hat{y}^{(i)} = \sigma(w \cdot x^{(i)})$
- α is the learning rate

Gradient Ascent for Logistic Regression

Maximum log-likelihood



Goal: Updates weights to maximize likelihood

The gradient of the log-likelihood w.r.t. w:

$$\nabla_w \log L(w) = \sum_{i=1}^{m} \left[y^{(i)} - \sigma(w \cdot x^{(i)}) \right] x^{(i)}$$

Gradient Ascent Update Rule:

$$w := w + \alpha \cdot \sum_{i=1}^{m} \left[y^{(i)} - \hat{y}^{(i)} \right] x^{(i)}$$

- $\hat{y}^{(i)} = \sigma(w \cdot x^{(i)})$
- α is the learning rate

Q1

What is **logistic regression**? How is it "logistic" and what are we "regressing"?

*The term **logistic** refers to the logistic (sigmoid) function used in the model, which is defined as*

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

where z is a linear regression of the features X and their corresponding weights β

$$z = \vec{\beta} \cdot \vec{X} = \beta_0 + \beta_1 x_1 + \ldots + \beta_D x_D$$

Q2

In the dataset shown below we want identify if a piece of writing is about computer or fruit (e.g. 'new apple iPhone is very expensive' vs. 'an apple a day, keeps the doctor away'). To do so, we are using 4 terms (apple, ibm, lemon, sun) and the count of their occurrences in a piece of writing. So for example we know that document A includes 'apple' once and 'sun' five times.

Build a logistic regression classifier, which uses the counts of selected words in news articles to predict the class of the news article (fruit vs. computer). Use the weights $\hat{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4] = [0.2, 0.3, -2.2, 3.3, -0.2]$. Recall that β_0 is the bias.

Training data:

ID	apple	ibm	lemon	sun	Class
A	1	0	1	5	fruit (1)
B	1	0	1	2	fruit (1)
C	2	0	0	1	fruit (1)
D	2	2	0	0	computer (0)
E	1	2	1	7	computer (0)

Test instance:

ID	apple	ibm	lemon	sun	Class
T	1	2	1	5	computer (0)

1. Explain how these parameters $\hat{\beta}$ are used in a logistic regression model.
2. Predict the label for the test instance.
3. Recall the conditional likelihood objective $-log\mathcal{L}(\beta)$ defined below, which is used as the loss function for the model. Verify that this value is lower when the model predicts the correct label for instance T and higher when the model predicts the incorrect label.

$$-log\mathcal{L}(\beta) = -\sum_{i=1}^{n} y_i log(\sigma(x_i;\beta)) + (1-y_i) log(1-\sigma(x_i;\beta))$$

1. Explain how these parameters $\hat{\beta}$ are used in a logistic regression model.

Logistic regression uses a **linear combination of the input features** and applies the **sigmoid function** to produce a **probability**.

The prediction formula is:

$$P(y=1 \mid x_1, x_2, x_3, x_4) = \hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}, \quad \text{where } z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$$

Where:

- $x_1 = \text{count of "apple"}$
- $x_2 = \text{count of "ibm"}$
- $x_3 = \text{count of "lemon"}$
- $x_4 = \text{count of "sun"}$

The parameters (weights) are given:

$$\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4] = [0.2, 0.3, -2.2, 3.3, -0.2]$$

The weights β_1 to β_4 denote the respective importance of the features for predicting the positive class fruit. For example, β_2=−2.2 indicates how important feature 2 ('ibm') is for predicting the fruit class. Note that 'ibm' has a negative weight, so documents that contain 'ibm' are less likely to be fruit class and more likely to be computer class. β_0 is the bias term for the regression.

Predict the label for the test instance.

Input features for instance T:

$$x = [1, 2, 1, 5]$$

ID	apple	ibm	lemon	sun	Class
T	1	2	1	5	computer (0)

Compute:

$$z = 0.2 + (0.3 \cdot 1) + (-2.2 \cdot 2) + (3.3 \cdot 1) + (-0.2 \cdot 5)$$

$$z = 0.2 + 0.3 - 4.4 + 3.3 - 1.0 = -1.6$$

Now compute the sigmoid:

$$\hat{y} = \frac{1}{1 + e^{-(-1.6)}} =$$ 0.17, since 0.17 < 0.5, the predicted label is 0 (computer)

The ground truth label for this test instance is computer, so the logistic regression prediction is correct!

3. Recall the conditional likelihood objective $-log\mathcal{L}(\beta)$ defined below, which is used as the loss function for the model. Verify that this value is lower when the model predicts the correct label for instance T and higher when the model predicts the incorrect label.

$$-log\mathcal{L}(\beta) = -\sum_{i=1}^{n} y_i log(\sigma(x_i;\beta)) + (1-y_i)log(1-\sigma(x_i;\beta))$$

The general formula for the **loss of one instance** in logistic regression is:

$$\text{Loss} = -\left[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right]$$

Where:

- $\hat{y} = \sigma(z)$: predicted probability from the model
- y: true label (either 0 or 1)

Now, $\hat{y} = \sigma(-1.6) \approx 0.17$, as calculated earlier (model says **17% chance it's "fruit"**).

Part A: What if the true label is 1 ("fruit") and prediction is 0?

This means:

- The **actual label is 1**, but the **model thinks it's class 0 (computer) → Wrong prediction**.

So plug into the loss formula:

$$\text{Loss} = -[1 \cdot \log(0.17) + 0 \cdot \log(1-0.17)] = -\log(0.17) \approx 1.77$$

High loss → because the model confidently made the **wrong** prediction.

Part B: What if the true label is 0 ("computer") and prediction is 0?

This means:

- The actual label is 0, and the model predicted a low probability of it being "fruit" → **Correct prediction**.

$$\text{Loss} = -[0 \cdot \log(0.17) + 1 \cdot \log(1 - 0.17)] = -\log(0.83) \approx 0.19$$

Low loss → because the model confidently made the **right** prediction.

As expected, the negative log likelihood is lower when the predicted label is correct ($\hat{y}$ =0) than when the label is incorrect ($\hat{y}$ =1).

Q3

Compute a single gradient ascent update for the parameter β_1 given the model and training instances in question 2. Recall that for each feature j, we compute its weight update as

$$\beta_j \leftarrow \beta_j + \alpha \frac{\partial \mathcal{L}(\beta)}{\partial \beta_j}$$

with the conditional likelihood $\mathcal{L}(\beta)$ computed over all training instances i. Do the update assuming the current parameters $\hat{\beta}$ specified in question 2 and a learning rate $\alpha = 0.1$.

Gradient ascent is equivalent to gradient descent, but it aims to maximize a value, while gradient descent aims to minimize a value. In this case, we'll aim to maximize the log likelihood of the logistic regression model with parameters β.

For logistic regression, each weight update is:

$$\beta_j \leftarrow \beta_j + \alpha \cdot \frac{\partial \mathcal{L}(\beta)}{\partial \beta_j}$$

Where:

- α = learning rate = 0.1
- $\frac{\partial \mathcal{L}}{\partial \beta_j} = \sum_i (y_i - \hat{y}_i) x_{ij}$

So:

$$\beta_1 \leftarrow \beta_1 + 0.1 \cdot \sum_i (y_i - \hat{y}_i) x_{i1}$$

The Logistic Regression Prediction Formula

For each instance i, we compute:

$$z_i = \beta_0 + \beta_1 \cdot x_{i1} + \beta_2 \cdot x_{i2} + \beta_3 \cdot x_{i3} + \beta_4 \cdot x_{i4}$$

Then apply the **sigmoid function**:

$$\hat{y}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

ID	apple	ibm	lemon	sun	Class
A	1	0	1	5	fruit (1)
B	1	0	1	2	fruit (1)
C	2	0	0	1	fruit (1)
D	2	2	0	0	computer (0)
E	1	2	1	7	computer (0)

Given Parameters (from Q2)

$$\beta = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4] = [0.2, 0.3, -2.2, 3.3, -0.2]$$

(These are the weights for: intercept, apple, ibm, lemon, sun)

$$\beta_1 \leftarrow \beta_1 + 0.1 \cdot \sum_i (y_i - \hat{y}_i) x_{i1}$$

Document A

$$z_A = 0.2 + 0.3(1) + (-2.2)(0) + 3.3(1) + (-0.2)(5)$$

$$= 0.2 + 0.3 + 0 + 3.3 - 1.0 = 2.8$$

$$\hat{y}_A = \frac{1}{1 + e^{-2.8}} \approx \frac{1}{1 + 0.061} \approx 0.94$$

Document B

$$z_B = 0.2 + 0.3(1) + (-2.2)(0) + 3.3(1) + (-0.2)(2)$$

$$= 0.2 + 0.3 + 0 + 3.3 - 0.4 = 3.4$$

$$\hat{y}_B = \frac{1}{1 + e^{-3.4}} \approx \frac{1}{1 + 0.033} \approx 0.97$$

Document C

$$z_C = 0.2 + 0.3(2) + (-2.2)(0) + 3.3(0) + (-0.2)(1)$$

$$= 0.2 + 0.6 + 0 + 0 - 0.2 = 0.6$$

$$\hat{y}_C = \frac{1}{1 + e^{-0.6}} \approx \frac{1}{1 + 0.548} \approx 0.65$$

Document D

$$z_D = 0.2 + 0.3(2) + (-2.2)(2) + 3.3(0) + (-0.2)(0)$$

$$= 0.2 + 0.6 - 4.4 + 0 + 0 = -3.6$$

$$\hat{y}_D = \frac{1}{1 + e^{3.6}} \approx \frac{1}{1 + 36.6} \approx 0.027 \approx 0.03$$

Document E

$$z_E = 0.2 + 0.3(1) + (-2.2)(2) + 3.3(1) + (-0.2)(7)$$

$$= 0.2 + 0.3 - 4.4 + 3.3 - 1.4 = -2.0$$

$$\hat{y}_E = \frac{1}{1 + e^{2.0}} \approx \frac{1}{1 + 7.39} \approx 0.12$$

Doc	z_i	$\hat{y}_i = \sigma(z_i)$
A	2.8	0.94
B	3.4	0.97
C	0.6	0.65
D	-3.6	0.03
E	-2.0	0.12

Compute the Gradient for β_1

$$\beta_1 \leftarrow \beta_1 + 0.1 \cdot \sum_i (y_i - \hat{y}_i) x_{i1}$$

$$\sum_i (y_i - \hat{y}_i) x_{i1} = (1 - 0.94) \cdot 1 + (1 - 0.97) \cdot 1 + (1 - 0.65) \cdot 2 + (0 - 0.03) \cdot 2 + (0 - 0.12) \cdot 1$$

$$= (0.06 \cdot 1) + (0.03 \cdot 1) + (0.35 \cdot 2) + (-0.03 \cdot 2) + (-0.12 \cdot 1)$$

=0.06+0.03+0.70−0.06−0.12=0.61

Update β_1

Initial $\beta_1 = 0.3$

Learning rate $\alpha = 0.1$

$$\beta_1 = 0.3 + 0.1 \cdot 0.61 = 0.361$$

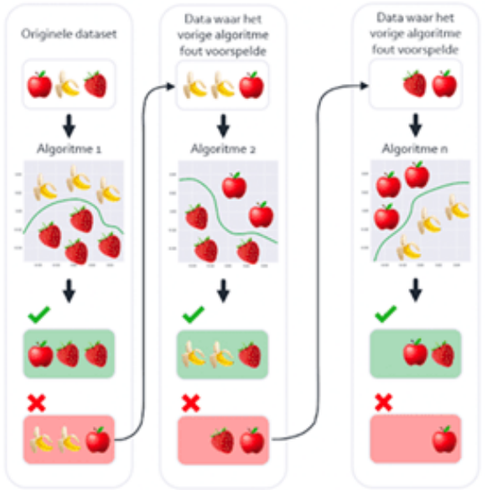
So the new value for $\beta 1$ is 0.361, slightly higher than the previous iteration's value (0.3). $\beta 1$ is the weight for the 'apple' feature, so this update means the model places slightly more weight on 'apple' when predicting the class fruit.

In the full gradient ascent algorithm, we would update all five parameters ($\beta 0$,$\beta 1$,...$\beta 4$), and continue this iterative optimisation process until the algorithm converges (the absolute change in log likelihood is below some threshold).

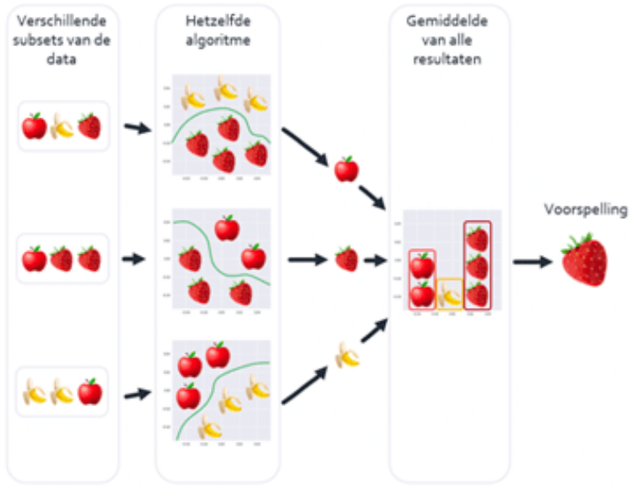
Ensemble Methods

Combine multiple models to make better predictions than any single model. It works by training multiple models on the same dataset and combining their output to produce a final prediction that is more accurate than the individual models.

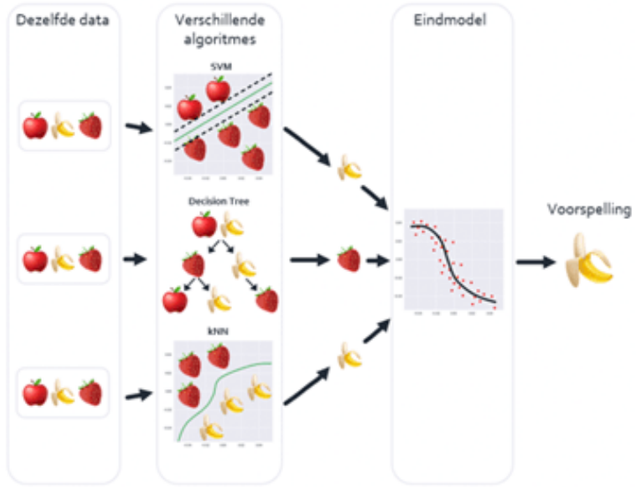
Types of Ensembles Learning Techniques:



Boosting

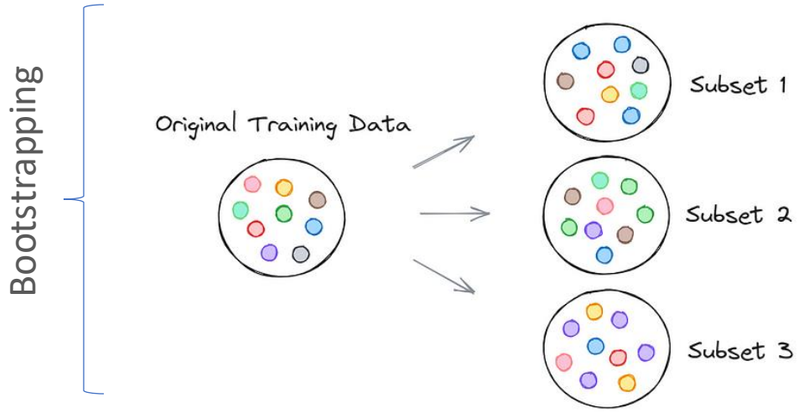


Bagging



Stacking

https://www.almabetter.com/bytes/articles/ensemble-learning



Bagging: Bootstrap Aggregating

Create Bootstrap Samples

Generate multiple datasets by sampling with replacement from training data.

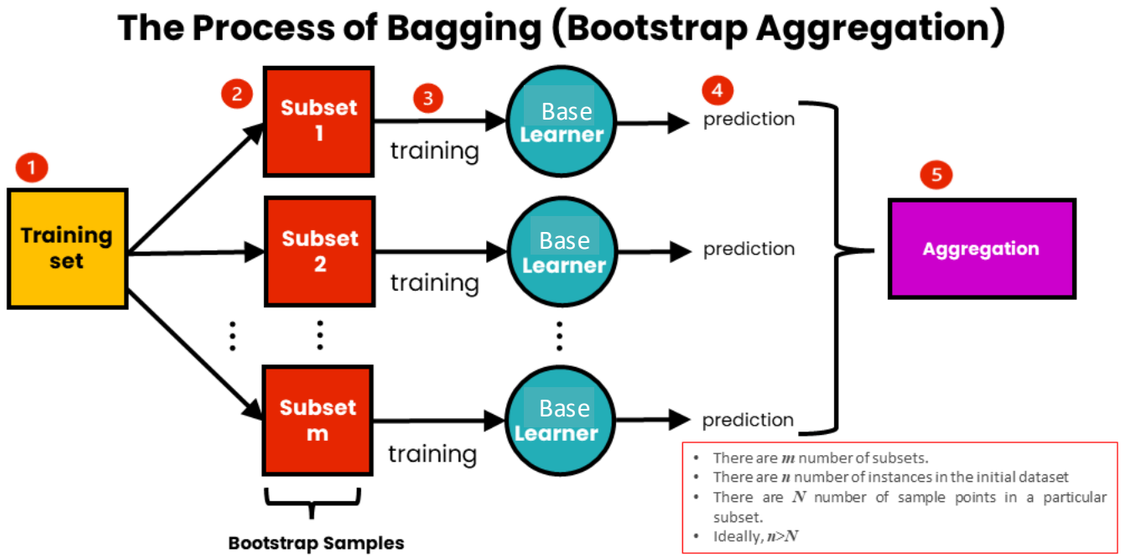
→ This is called **bootstrapping**.

Train Independent Models

Train a separate model on each of the subsets of the dataset generated from bootstrapping. Models are trained independently (in parallel).

Combine Predictions

Average predictions (regression) or take majority vote (classification).



All the base learners in bagging are of the **same type (homogeneous)** or the **same algorithm**.

Random Forest is an example of bagging

https://pub.towardsai.net/bagging-vs-boosting-the-power-of-ensemble-methods-in-machine-learning-6404e33524e6

https://www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/

Introduction to Random Forests

Random Forest is a powerful machine learning algorithm used for **classification** and **regression**.

It's based on **Random Trees, Bagging** (Bootstrap Aggregating), and introduces **extra randomness** for better generalisation.

Given: A dataset with **N instances** and **M features**, Random Forest works as follows:

Data Sampling (Bagging)

- For each tree, draw **N instances with replacement** from the original dataset.
- Some instances may appear multiple times; some not at all.

Feature Selection (Random Subspace Method)

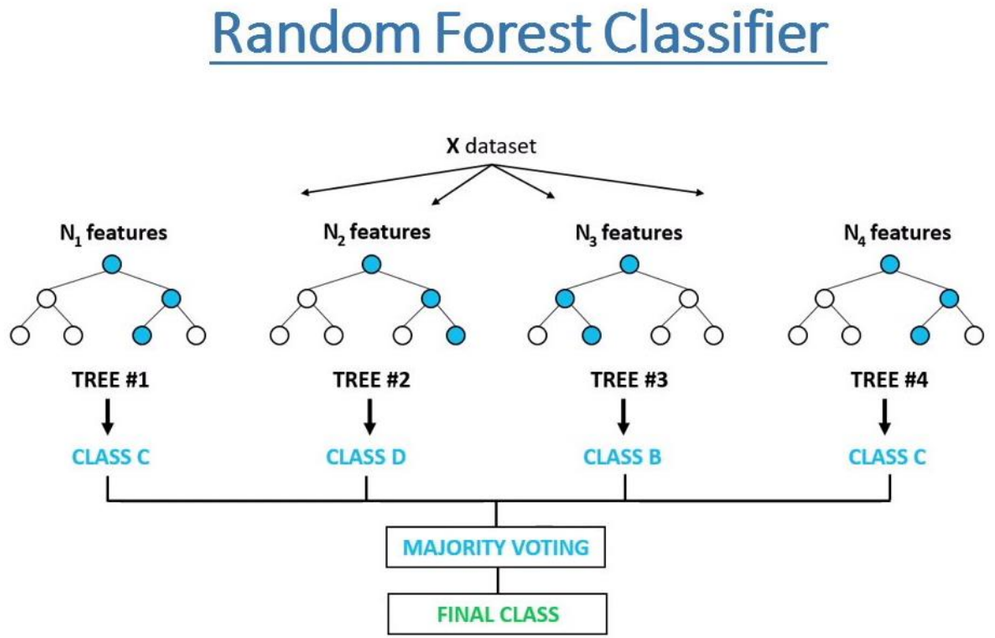
- For each tree, select a **random subset of features (k ≤ M)**.
- This is done **at each split**, not just once per tree.

Tree Construction

- Use the sampled data and random subset of features to grow a **random tree**.

Prediction

- For classification: final output = **majority vote** across all trees.
- For regression: final output = **average** of all tree outputs.



https://medium.com/@mrmaster907/introduction-random-forest-classification-by-example-6983d95c7b91

Q4

Describe how to build a **random forest** for a given dataset and answer the following questions:

1. What is the benefit of bagging?
2. What is the impact of the number of trees in a random forest?
3. What will happen if the random number of features chosen for splitting nodes in a random forest is very large?

Step-by-step process:

1.Start with your dataset (features X, labels y).

2.Decide how many trees you want in the forest (e.g., 100 trees).

3.For each tree in the forest:

1. **Draw a bootstrap sample** from the training data (random sample **with replacement**).
2. **Grow a decision tree** on that sample:
    1. At **each split**, choose a **random subset of features** (not all).
    2. Split on the feature that gives the best result (Gini, information gain, gain ratio, etc.).
    3. Repeat until a stopping condition is met (e.g., max depth, min samples).

4.Aggregate predictions:

1. For classification: **majority vote** across all trees.
2. For regression: **average** the predictions of all trees.

1. What is the benefit of bagging?

Bagging = Bootstrap Aggregating

- **Reduces variance**: By training multiple models on different data subsets, random fluctuations and overfitting are averaged out.
- **Increases stability**: A single decision tree may overfit, but averaging many of them (via bagging) creates a more robust model.
- **Improves generalization**: Better performance on unseen data.

---

2. What is the impact of the number of trees in a random forest?

# Trees	Impact
Too few	Model may be **unstable**, high variance
More trees	**Reduces variance**, increases stability
Too many	Very **small performance gains**, but **higher cost** (longer training/prediction time)

Performance improves quickly at first with more trees, but **plateaus** after a point.
100–500 trees is usually a good range.

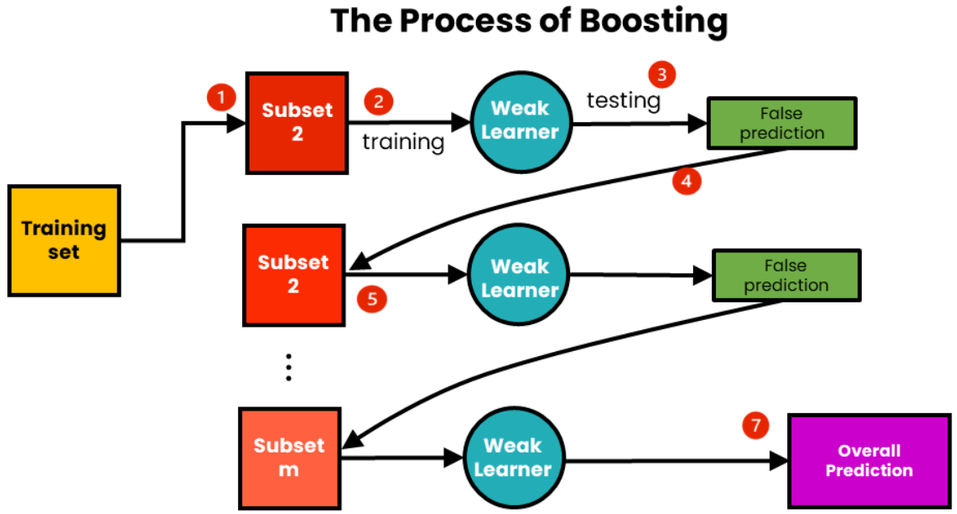
3. What happens if the number of random features chosen at each split is very large?

- You **lose the benefit of feature randomness**:
  - Trees become **more similar** (less diverse),
  - Leads to **higher correlation** among trees.
  - As a result, **variance reduction is less effective**.

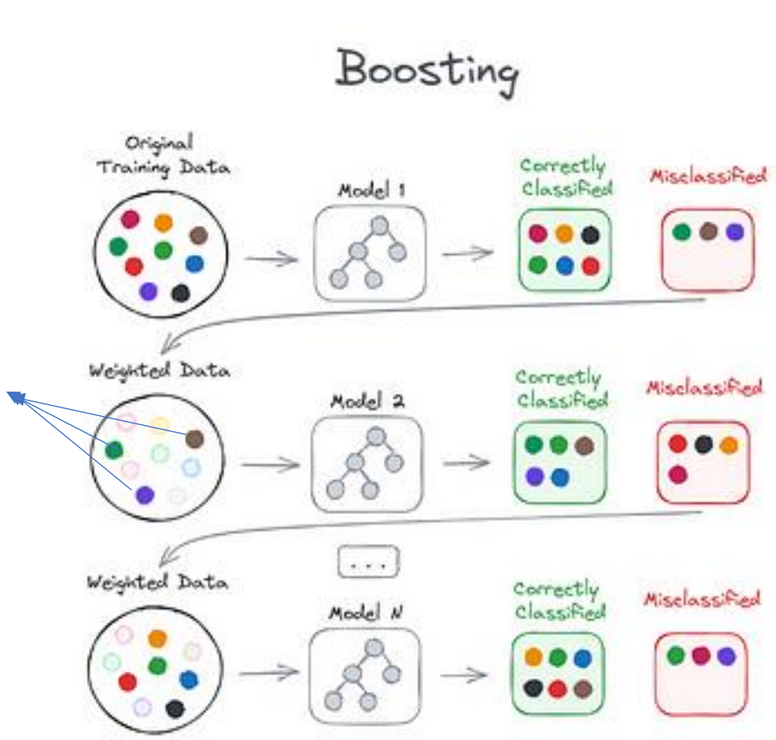
Consequence:

- Model may **overfit**, especially if strong predictors dominate all trees.
- Accuracy may **decrease** on unseen data.

Boosting: Sequential Ensemble Learning



All the base learners are of the **same type (homogeneous)**



Initial Weak Learner

Start with a simple model (stump or a shallow decision tree) that performs slightly better than random and train it on the whole dataset.

Focus on Errors

Increase weights for misclassified samples. Decrease the weights of the correctly classified ones. So, the next model focuses more on these misclassified cases.

Add New Model

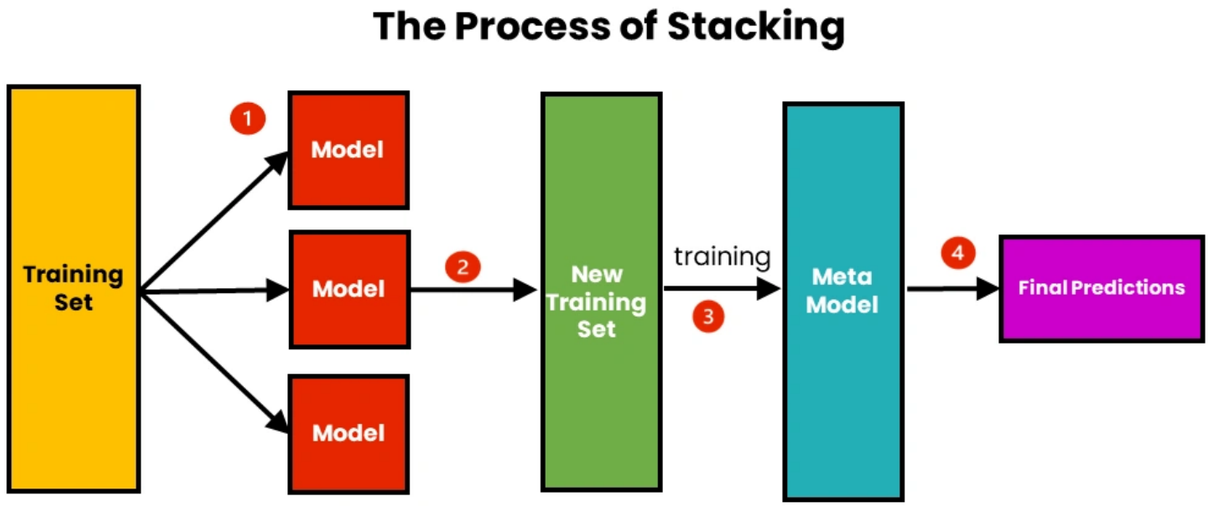
Train the next model on this weighted data. Each new model corrects errors made by the ensemble.

Weight and Combine

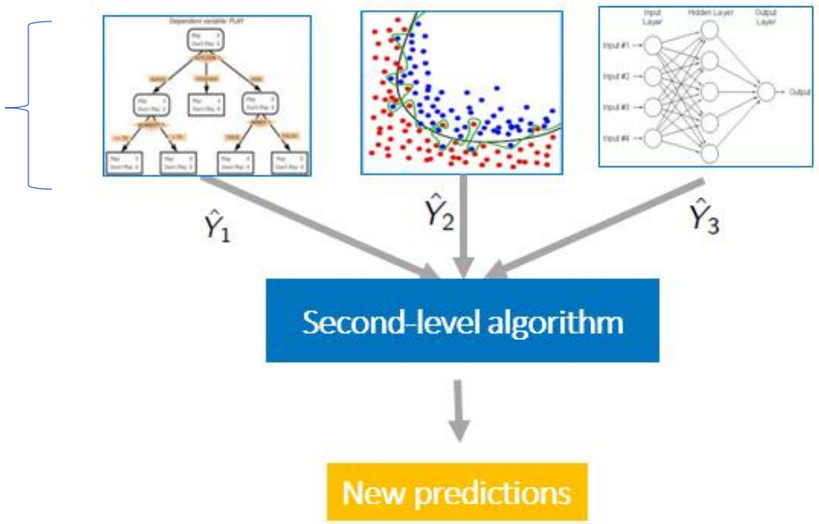
Combine the models using weighted votes (more accurate models get more say)

https://www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/
https://pub.towardsai.net/bagging-vs-boosting-the-power-of-ensemble-methods-in-machine-learning-6404e33524e6

Stacking: Meta-Ensemble Learning



The base models are different types of algorithms (**heterogeneous**)



Base Models

Train several different models (e.g., logistic regression, decision tree, SVM) on the same training data

Predictions Layer

Each model makes predictions on a validation set. These predictions become inputs (features) to the meta-learner

Meta-Learner

The meta-learner is trained to make the final prediction based on the base models' outputs

Stacking often outperforms individual models.

It combines their strengths while mitigating weaknesses.

https://www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/

https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

When to use Bagging vs Boosting vs Stacking?

	Bagging	Boosting	Stacking
Purpose	Reduce Variance	Reduce Bias	Improve Accuracy
Base Learner Types	Homogeneous	Homogeneous	Heterogeneous
Base Learner Training	Parallel	Sequential	Meta Model
Aggregation	Max Voting, Averaging	Weighted Averaging	Weighted Averaging

https://www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/