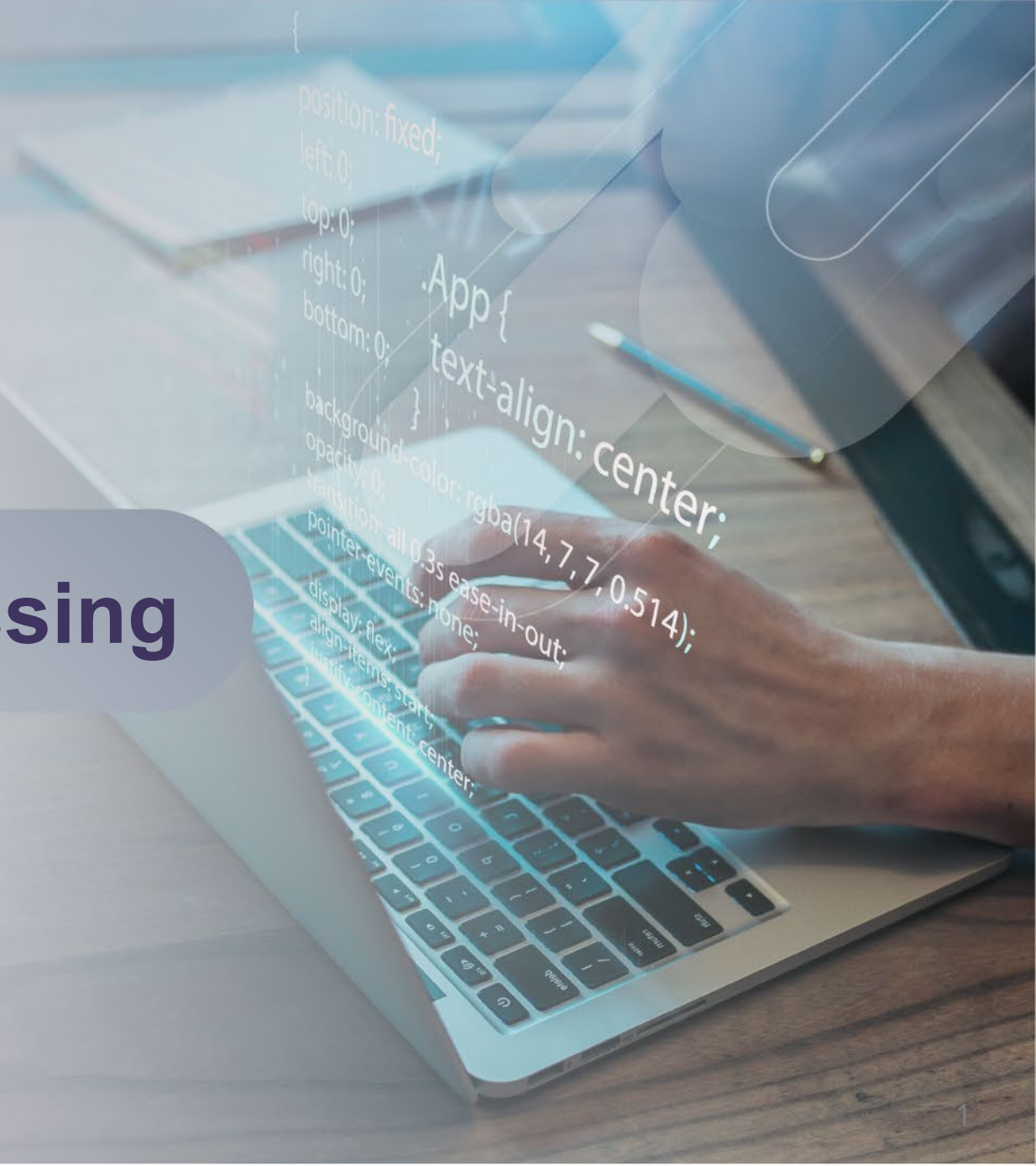
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

EE6405

Natural Language Processing



Dr. S. Supraja
NTU Electrical and Electronic Engineering

Traditional Machine Learning Methods and NLP Applications

Text Classification

Text classification tasks involve the labelling and classification of texts into various categories.

These categories can include:

- **Topics:**
  - Topic Modelling aims to classify text into its various topics based on the contents of the text.
- **Sentiment:**
  - Sentiment analysis analyses general sentiment towards a company, person or product.
  - Often applied on market research and reputation management tasks.
- **Languages**:
  - Language identification can be particularly useful in processing search engine queries.
- **Authors:**
  - Commonly used in forensics and cybersecurity.

Text Classification

- Traditional ML classifiers aim to implement a generic algorithm to generate a trained model using a labelled training set.
- The classifier learns the characteristics of each label from the training data to assign labels to new data.
- Text Classifiers generally feature:
  - Input:
    - A document, $\boldsymbol{d}$
    - A predefined set of classes $\{\boldsymbol{c_1}, \boldsymbol{c_2}, \boldsymbol{c_3} \ldots \boldsymbol{c_j}\}$
  - Output:
    - A predicted class $\boldsymbol{c} \in \boldsymbol{C}$

Text Classification

- Text classification can be achieved through:
  - Rule based classifiers
  - ML models:
    - Naïve-Bayes
    - Support Vector Machines (SVM)
    - Extreme Learning Machines (ELM)
    - Gaussian Processes
    - Linear Regression

Naïve Bayes

Naïve Bayes models for text classification take a BOW approach to text classification.

- Using the Bayes Theorem, for a document **d** and class **c**:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

- The most likely class c is given by:
  - $c = argmax_c(P(c|d))$ **<The most likely class is the one with the highest probability given document d**
  - $c = argmax_c\left(\frac{P(d|c)P(c)}{P(d)}\right)$ **<Using the Bayes Theorem**
  - $c = argmax_c(P(d|c)P(c))$ **<Dropping the denominator**
  - $c = argmax_c(P(x_1, x_2, x_3 \dots x_n|c)P(c))$ **< Where $x_1 \dots x_n$ are document features.**
  - $c = argmax_c\left(\prod_j P(x_j|c)\,P(c)\right)$ **< "Naïve" assumption that word probabilities are independent of each other.**

Naïve Bayes-Training

- Using maximum likelihood estimators from the training corpus:
  - Class Prior Probabilities:

$$P(c) = \frac{count(docs\ labelled\ c)}{count(total\ docs)}$$

  - Conditional Probabilities:

$$P(w_i|c) = \frac{count(docs\ with\ w_i\ labelled\ c)}{count(docs\ labelled\ c)}$$

Zero Probability Problem

Imagine the following scenario:

- If we were trying to classify reviews into classes ‘positive’ and ‘negative’ and run across the word “excellent” in ‘review1’, which does not appear in any other positive reviews.
  - $\boldsymbol{P(positive|review1) = 0}$

    because
  - $\boldsymbol{P("excellent"|positive) = 0}$
- These zero probabilities cannot be conditioned away no matter the evidence.

Naïve Bayes-Training

- To avoid this issue, we introduce Laplace (or Add-One smoothing)

$$P(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c) + 1)}$$

or

$$P(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}\big(count(w, c)\big) + |V|}$$

Where V is the size of the vocabulary

Naïve Bayes-Worked Example

$$P(c) = \frac{count(docs\ labelled\ c)}{count(total\ docs)}$$

$$P(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c)) + |V|}$$

Class Prior Prob:

$$P(ntu) = \frac{3}{4}$$

$$P(nus) = \frac{1}{4}$$

Class Prior Prob:

$$P(Hive|ntu) = \frac{5+1}{8+6} = \frac{6}{14} = \frac{3}{7}$$

$$P(Eusoff|ntu) = \frac{1}{14}$$

$$P(Temasek|ntu) = \frac{1}{14}$$

$$P(Hive|nus) = \frac{2}{9}$$

$$P(Eusoff|nus) = \frac{2}{9}$$

$$P(Temasek|nus) = \frac{2}{9}$$

Data	Doc	Words	Class
Training	1	Hive, Arc, Hive	NTU
	2	Hive, Hive, Spine	NTU
	3	Hive, Tamarind	NTU
	4	Eusoff, Temasek, Hive	NUS
Test	5	Hive, Hive, Hive, Eusoff, Temasek	?

Choosing a class:

$$P(ntu|d5) = \frac{3}{4} * \left(\frac{3}{7}\right)^3 * \frac{1}{14} * \frac{1}{14} \approx 0.0003$$

$$P(nus|d5) = \frac{1}{4} * \left(\frac{2}{9}\right)^3 * \frac{2}{9} * \frac{2}{9} \approx 0.0001$$

Data Preprocessing

- Use RegEx to remove punctuation and special characters, convert all characters to lowercase.
- Apply POS tagging and lemmatise input words
- Remove stop-words.
- Apply TF-IDF vectorisation.

```
Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.<br /><
br />This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.<br /><br />OK, first of all
when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing &
arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGEYMAN similar m
ovie, and instead i watched a drama with some meaningless thriller spots.<br /><br />3 out of 10 just for the well playing parents & desce
nt dialogs. As for the shots with Jake: just ignore them.
```

```
print(train_review_tfidf[3])
```

```
[[0. 0. 0. ... 0. 0. 0.]]
```

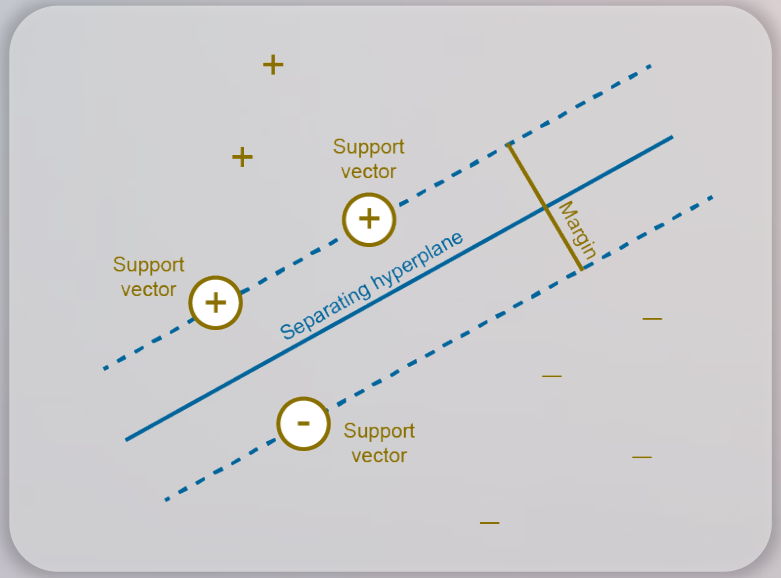
```
print(train_review_tfidf[3].shape)
```

```
(1, 42076)
```

Support Vector Machines

Support Vector Machines adopt a graphical approach to classifying data.

- SVMs aim to find hyperplanes that best separates data points of different classes in a high-dimensional space.
- SVMs select the hyperplane for which the closest points from the dataset are the farthest.



SVMs aim to maximise the margin between the hyperplanes and closest points from the dataset.

Support Vector Machines

To apply SVM in the context of NLP, the textual data need to be first represented as vectors.

- One of the most popular ways is to use TF-IDF vectorisation.
- The equation of a hyperplane can be given by $\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b} = \boldsymbol{0}$
- The distance between a point x to a hyperplane is $\frac{|w \cdot x + b|}{\|w\|^2}$
- The SVM algorithm aims to maximise this quantity by minimising the value of $\|\boldsymbol{w}\|^{\boldsymbol{2}}$
- This is known as the ‘primal’ problem of SVMs.

Support Vector Machines

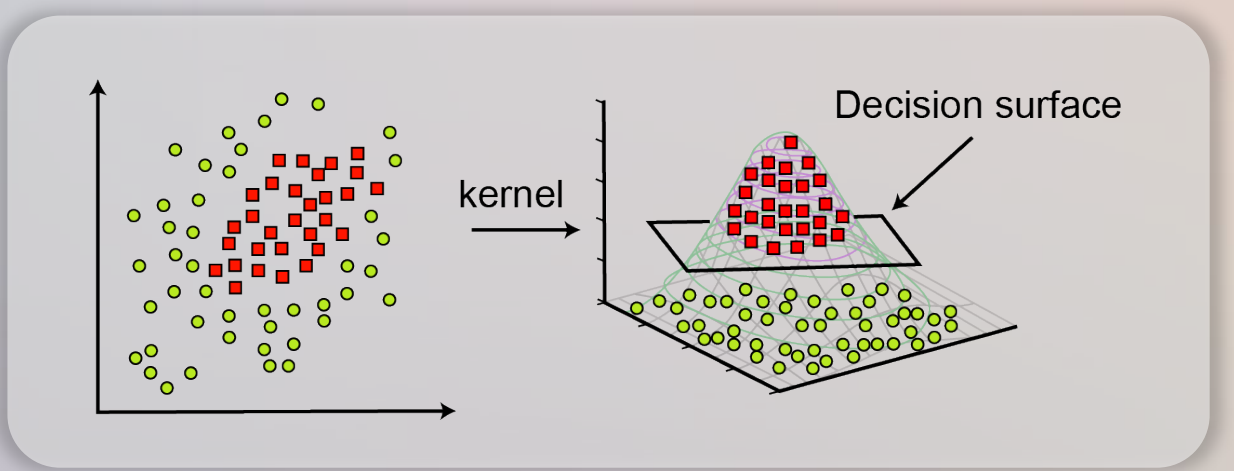
- To optimise the SVM model, we employ a hinge loss function:

$$c(x, y, f(x)) = \begin{cases} 0, if\ y * f(x) \geq 1 \\ 1 - y * f(x), \quad else \end{cases}$$

  - If the predicted value and the actual value are of the same sign, the cost is 0.
  - If not, we calculate the loss value.
  - A regularisation function, $C * ||w||^2$ is often added to discourage the model from fitting the training data too closely and overfitting.

Support Vector Machines

- In many real-life scenarios including text processing, the data in the input space of an SVM may not be linearly separable, posing a challenge for SVMs to find a hyperplane to cut the data.
- The kernel trick addresses this limitation by mapping the data into a higher-dimensional feature space where it becomes linearly separable.
- Instead of explicitly calculating the coordinates of data points in the higher-dimensional space, we compute the dot products between the data points in this space without ever computing the transformation explicitly.



Support Vector Machines

- The most commonly used kernels include:

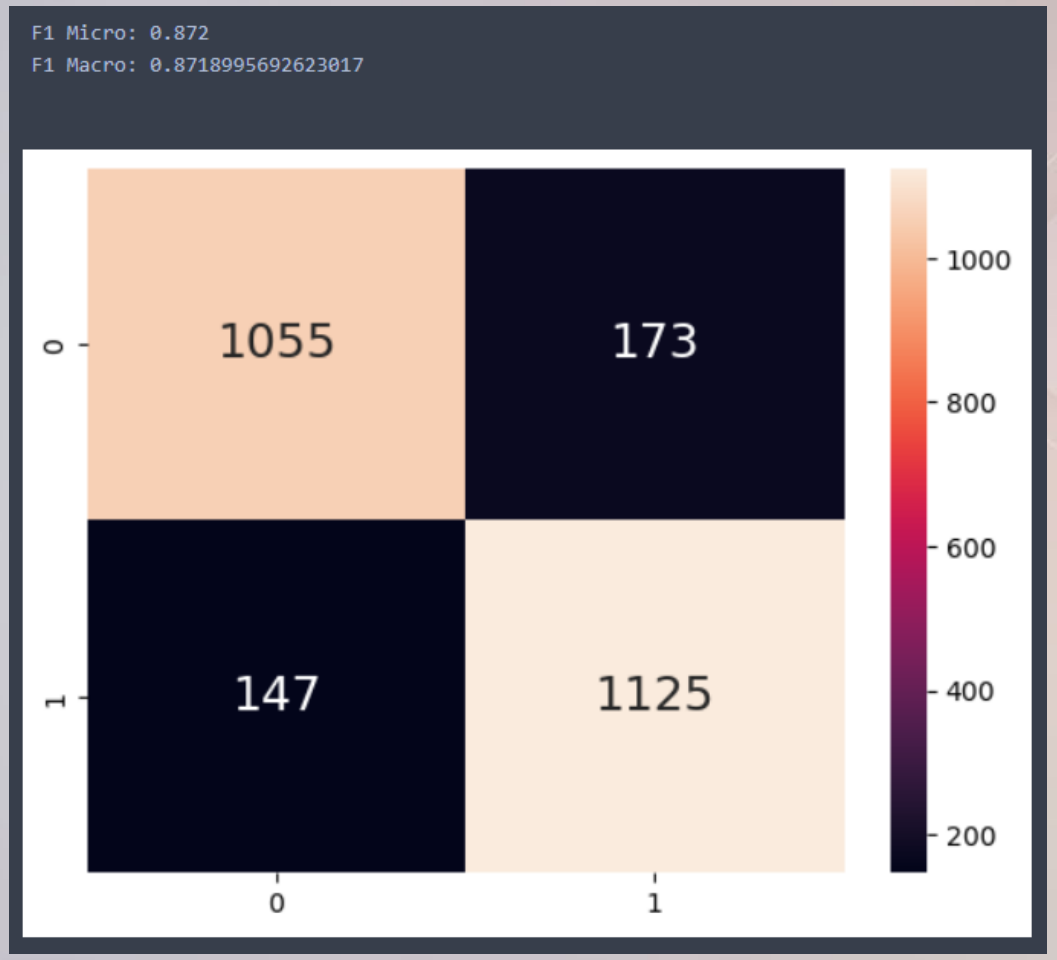
Type of SVM	Mercer Kernel	Description
Radial Basis Function	$K(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$	One class learning with σ as the width of the kernel
Linear	$K(x_1, x_2) = x_1^T x_2$	Two class learning
Polynomial	$K(x_1, x_2) = (x_1^T x_2 + 1)^\rho$	ρ is the order of the polynomial
Sigmoid	$K(x_1, x_2) = \tanh(\beta_0 x_1^T x_2 + \beta_1)$	A mercer kernel only for certain values of β_0 and β_1

Support Vector Machines

```
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='rbf') # RBF kernel

#Train the model using the training sets
clf.fit(train_review_tfidf, train_sent)

#Predict the response for test dataset
y_pred = clf.predict(test_review_tfidf)
```

Support Vector Machines

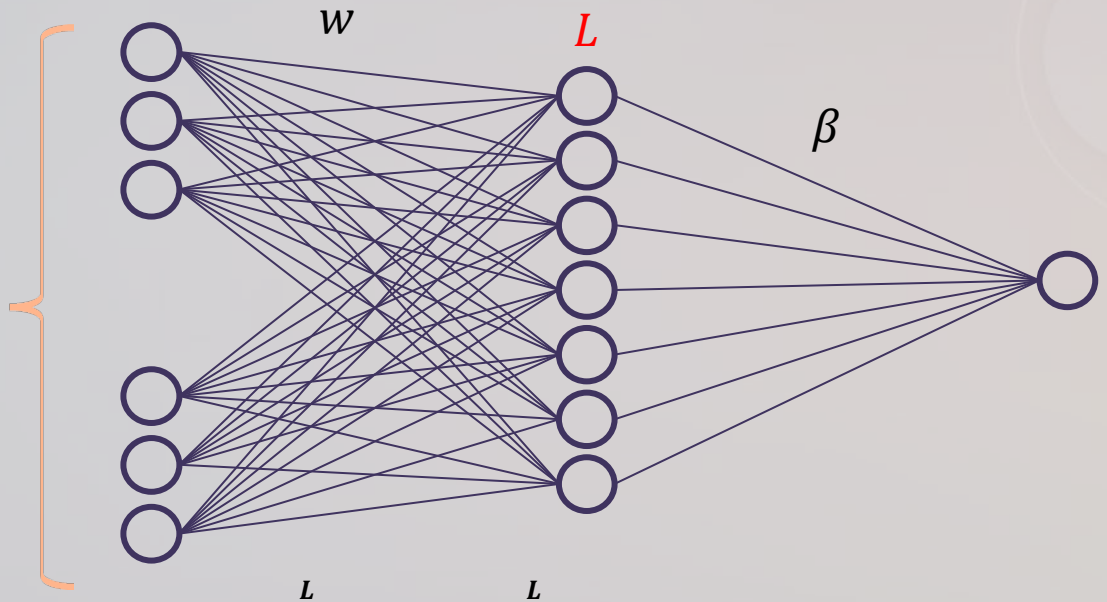
```
F1 Micro: 0.872
F1 Macro: 0.8718995692623017
```



Extreme Learning Machines (ELMs)

ELMs are shallow feedforward neural networks that feature a single hidden layer.

- Unlike traditional neural networks, ELMs do not feature a backpropagation step.



$$f_L(x) = \sum_{i=1}^{L} \beta_i g_i(x) = \sum_{i=1}^{L} \beta_i g(w_i * x_j + b_i), j = 1, \ldots, N$$

Extreme Learning Machines

- In ELMs, the weights between the input layer and the hidden layers are randomly initialized and not updated during training.
- The weights between the hidden layer and the output layer form a beta-matrix as shown below.

Where:

- *m* is the number of outputs
- *H* is the hidden layer output matrix
- *T* is the training data target matrix

$$H = \begin{bmatrix} g(w_1 * x_1 + b_1) & \dots & g(w_L * x_1 + b_L) \\ \vdots & & \\ g(w_1 * x_N + b_1) & \dots & g(w_L * x_N + b_L) \end{bmatrix}_{N \times L}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \qquad T = \begin{bmatrix} t_1^T \\ \vdots \\ t_L^T \end{bmatrix}_{N \times m}$$

Extreme Learning Machines

- An ELM aims to find this matrix β such that the error between the actual outputs (given by $H \cdot \beta$) and the target outputs (given by y) is minimised.
- This can be formulated as an SLE given by $H \cdot \beta = y$
- Since H is typically not square and may not have an exact inverse,
- We approximate β with $\beta = (H^{+}) \cdot y$
  - Where $H^{+}$is the Moore-Penrose Pseudoinverse of H
- In practice, H^{+}can be calculated using techniques like SVD.

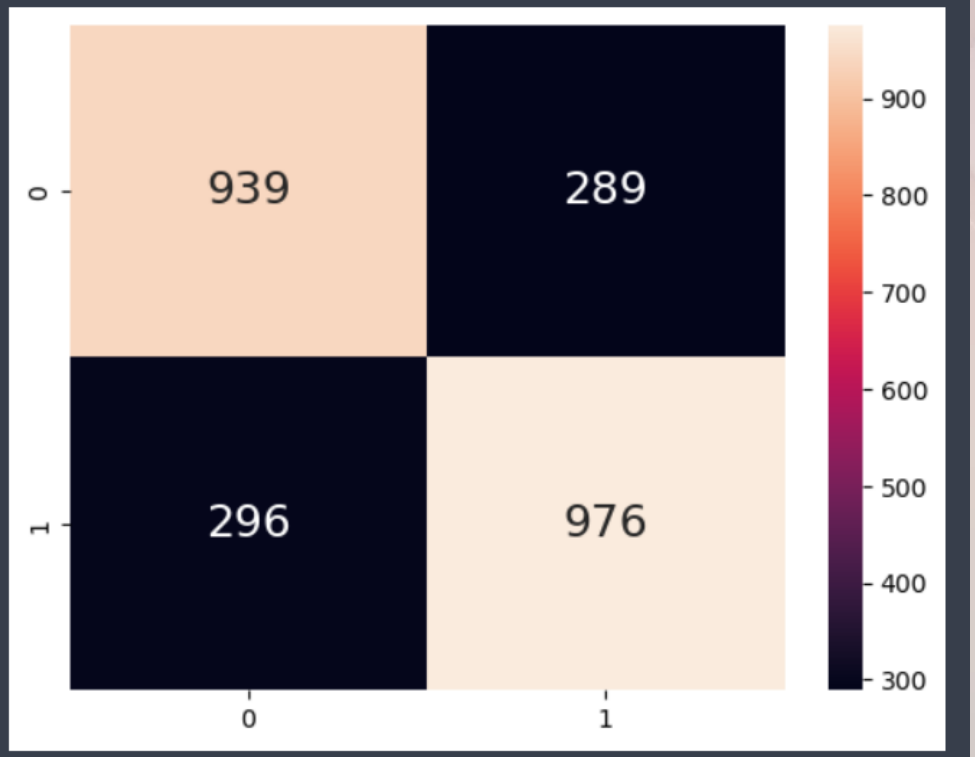
Extreme Learning Machines

```
from skelm import ELMClassifier
clf = ELMClassifier()
#Train the model using the training sets
clf.fit(train_review_tfidf, train_sent)

#Predict the response for test dataset
y_pred = clf.predict(test_review_tfidf)
```

Extreme Learning Machines

```
getConfMatrix(y_pred,test_sent)
```

```
F1 Micro: 0.766
F1 Macro: 0.7659487334105661
```



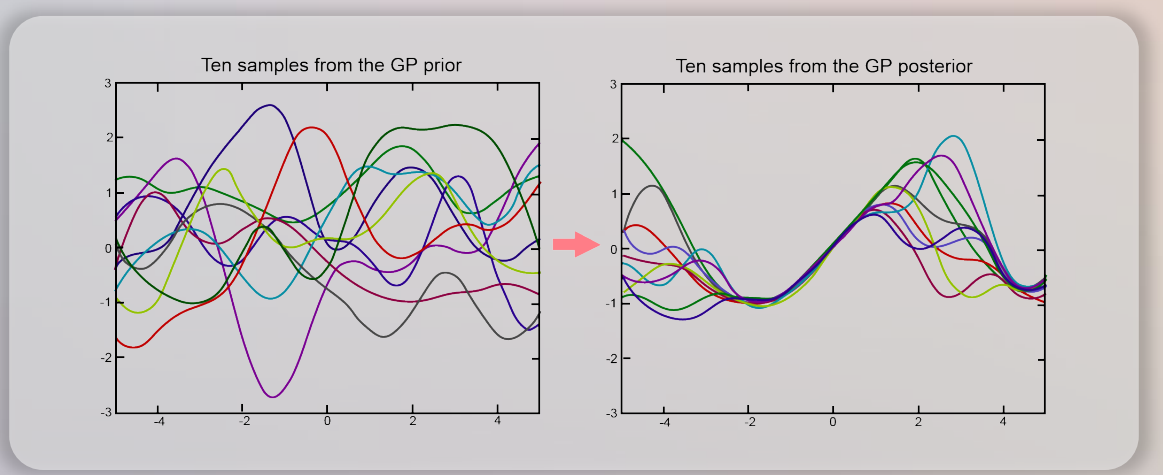
Gaussian Processes

A Gaussian Process is a probabilistic model that defines a distribution over functions.

- Instead of modeling data points as fixed parameters, GPs model entire functions as random variables.
- These functions are characterised by a mean function and a covariance function (or kernel function).
- To create a GP function, we need to specify:
  - A mean function, $\boldsymbol{E}[\boldsymbol{f}(\boldsymbol{x_i})] = \boldsymbol{\mu}(\boldsymbol{x_i})$
  - A covariance function aka a kernel function $\boldsymbol{Cov}\left(\boldsymbol{f}(\boldsymbol{x_i}), \boldsymbol{f}\left(\boldsymbol{x_j}\right)\right) = \boldsymbol{k}(\boldsymbol{x_i}, \boldsymbol{x_j})$

Gaussian Processes

- Let $\boldsymbol{K_x}$ be the kernel matrix for inputs *x*. The entries of this matrix are $\boldsymbol{k(x_i, x_j)}$. This matrix is also known as the gram matrix.
- $\boldsymbol{K_x}$ must be positive and semidefinite for any *X*.
- This forms a distribution over *function values* at an arbitrarily finite set of points.
- Using the Kolmogorov Extension Theorem, we can extend this to be a distribution over *functions*, which is called a Gaussian Process.



Gaussian Processes

- Similar to SVMs, we can choose from a range of kernel functions to map our data.
- A useful kernel is the RBF kernel:

The RBF kernel, $\boldsymbol{K_{RBF}(x_i, x_j) = \sigma^2 \exp\left(-\frac{\|x_i - x_j\|^2}{2l^2}\right)}$

  - It creates smooth, infinitely differentiable functions that are useful for modeling processes with smooth variations.
  - The hyperparameter $\boldsymbol{l^2}$controls the length scale or the width of the kernel. Smaller values result in more wiggly functions, while larger values create smoother functions.

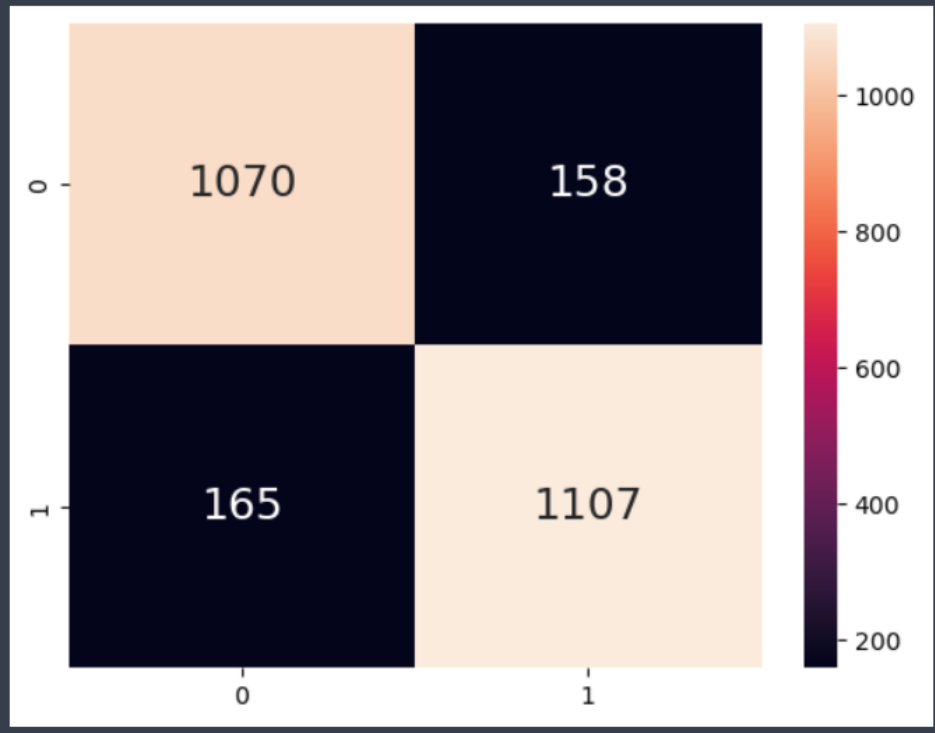
Gaussian Processes

```
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
kernel = 1.0 * RBF(1.0)
clf = GaussianProcessClassifier(kernel=kernel, random_state=0)
#Train the model using the training sets
clf.fit(train_review_tfidf, train_sent)

#Predict the response for test dataset
y_pred = clf.predict(test_review_tfidf)
```

Gaussian Processes

```
getConfMatrix(y_pred,test_sent)
```

```
F1 Micro: 0.8708
F1 Macro: 0.8707716938318169
```



Linear Regression

Linear Regression is a linear model that assumes the relationship between the input variables and the output is linear.

- Given the equation of a line:

$$y = mx + c$$

- The loss function to optimize the algorithm is defined as the squared error:

$$L(y, t) = \frac{1}{2}(y - t)^2$$

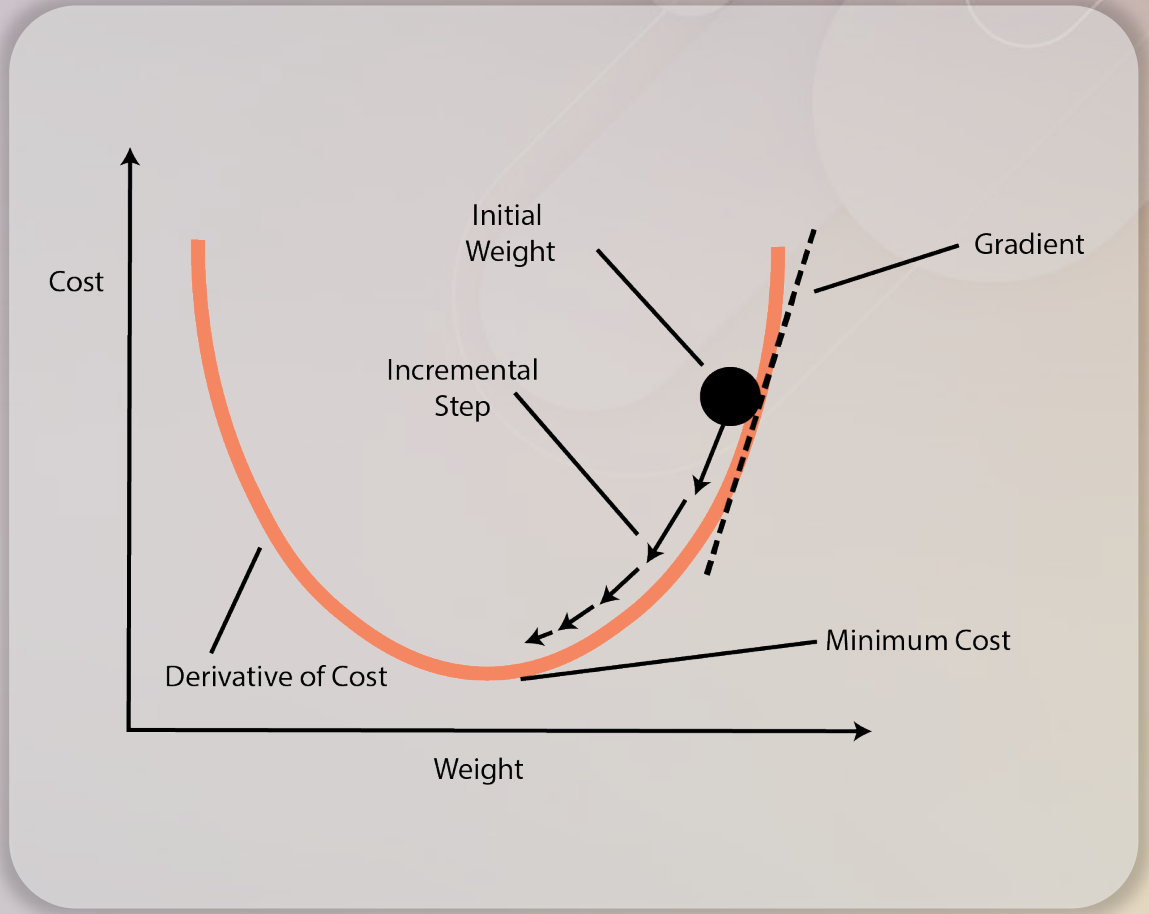
→ **To optimise this fit, we want to minimise y-t**

- The cost function is defined as the loss function averaged over all training data:

$$J(w, b) = \frac{1}{2N}\sum_{i=1}^{N}\left(y^i - t^i\right)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(wx^i + b - t^i\right)^2$$

Linear Regression

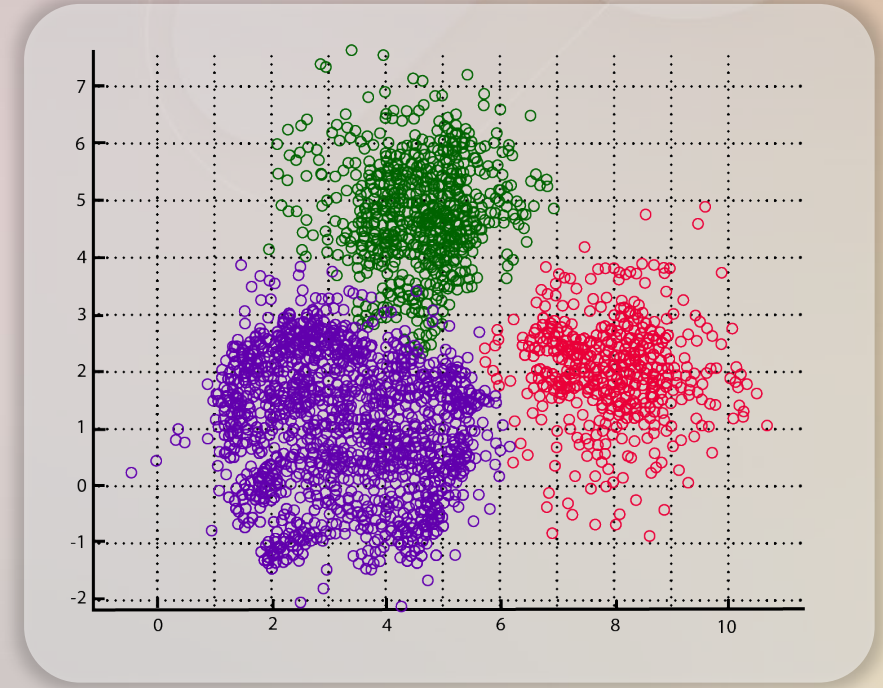
- To minimise this cost function, we apply the gradient descent algorithm, where we initialise the weights and iteratively adjust them in the direction of steepest descent.
- Consider:
  - **If $\frac{\partial J}{\partial w_j} > 0$, increasing w_j increases J**
  - **If $\frac{\partial J}{\partial w_j} < 0$, increasing w_j decreases J**
- To decrease the cost function, we apply $w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j}$
  - Where α is the learning rate of the algorithm.



Clustering

Clustering is an unsupervised machine learning technique that aims to group data into clusters within the input space.

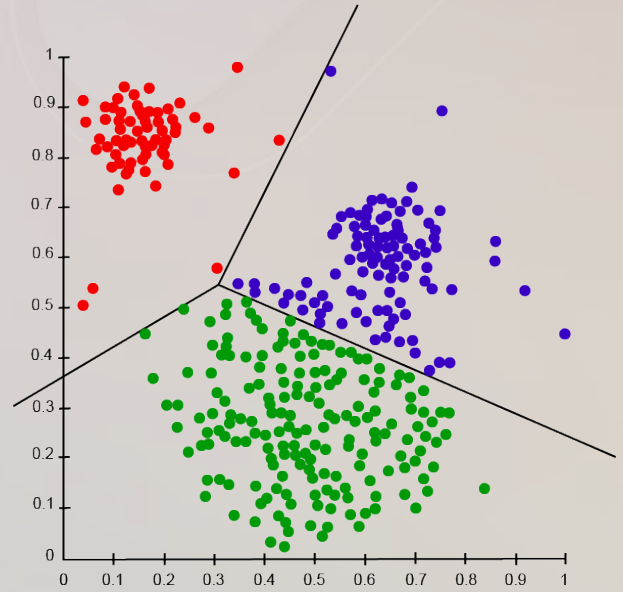
- The goal of clustering is to discover hidden structures in the data without any prior knowledge of the groupings.
- Clustering algorithms typically rely on a distance or similarity metric to measure how close or similar data points are in the feature space.



K-Means

K-Means assumes there are k clusters among N input samples, and each data point is close to its cluster center (the mean of points in the cluster).

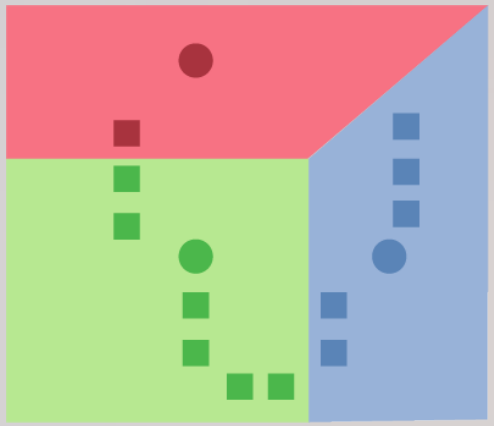
- To compute the cluster centers, the centers are randomly initialised, then iteratively moved towards their closest data points.



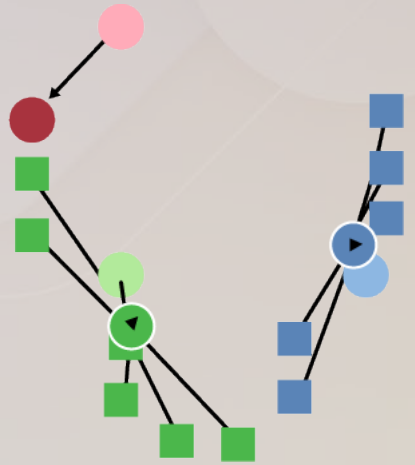
K-Means

The standard K-Means algorithm works as follows:

- **Initialisation**: The k centroids are randomly initialised within the Euclidean space.
- We then iteratively alternate between the following:
  - **Assignment**: Assign each data point to its closest cluster.
  - **Refitting**: Move the centroid to the center of the new cluster.



Assignment



Refitting

K-Means

```
from sklearn.cluster import KMeans
for seed in range(5):
    kmeans = KMeans(
        n_clusters=2,
        max_iter=100,
        n_init=1,
        random_state=seed,
    ).fit(train_review_tfidf)
    cluster_ids, cluster_sizes = np.unique(kmeans.labels_, return_counts=True)
    print(f"Number of elements assigned to each cluster: {cluster_sizes}")
print()
```

K-Means

```
Number of elements assigned to each cluster: [5542 1958]
Number of elements assigned to each cluster: [2079 5421]
Number of elements assigned to each cluster: [2081 5419]
Number of elements assigned to each cluster: [5433 2067]
Number of elements assigned to each cluster: [2069 5431]
```

K-Means

```
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer


lsa = make_pipeline(TruncatedSVD(n_components=100), Normalizer(copy=False))
train_review_lsa = lsa.fit_transform(train_review_tfidf)
explained_variance = lsa[0].explained_variance_ratio_.sum()
print(f"Explained variance of the SVD step: {explained_variance * 100:.1f}%")
```

```
Explained variance of the SVD step: 11.4%
```

K-Means

```
kmeans = KMeans(
    n_clusters=2,
    max_iter=100,
    n_init=5,
    random_state=seed,
).fit(train_review_lsa)
```

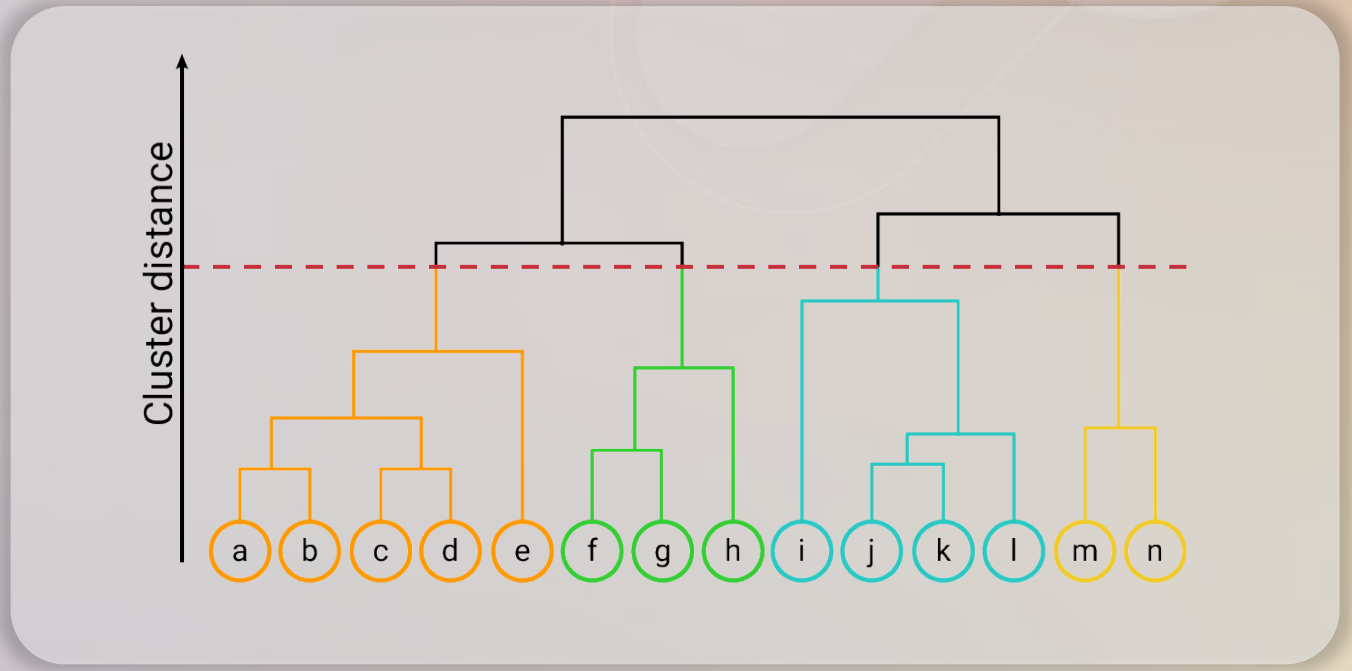
```
original_space_centroids = lsa[0].inverse_transform(kmeans.cluster_centers_)
order_centroids = original_space_centroids.argsort()[:, ::-1]
terms = tv.get_feature_names_out()

for i in range(2):
    print(f"Cluster {i}: ", end="")
    for ind in order_centroids[i, :10]:
        print(f"{terms[ind]} ", end="")
    print()
```

```
Cluster 0: movie bad good like watch make think time really film
Cluster 1: film make like good time story character movie great watch
```

Hierarchical Clustering

Clusters in **hierarchical clustering** are visually represented in a hierarchical tree called a dendrogram.

- There is no need to pre-specify the number of clusters. Instead, the dendrogram can be cut at the appropriate level to obtain the desired number of clusters.



Hierarchical Clustering

There are two general approaches to Hierarchical Clustering:

Agglomerative

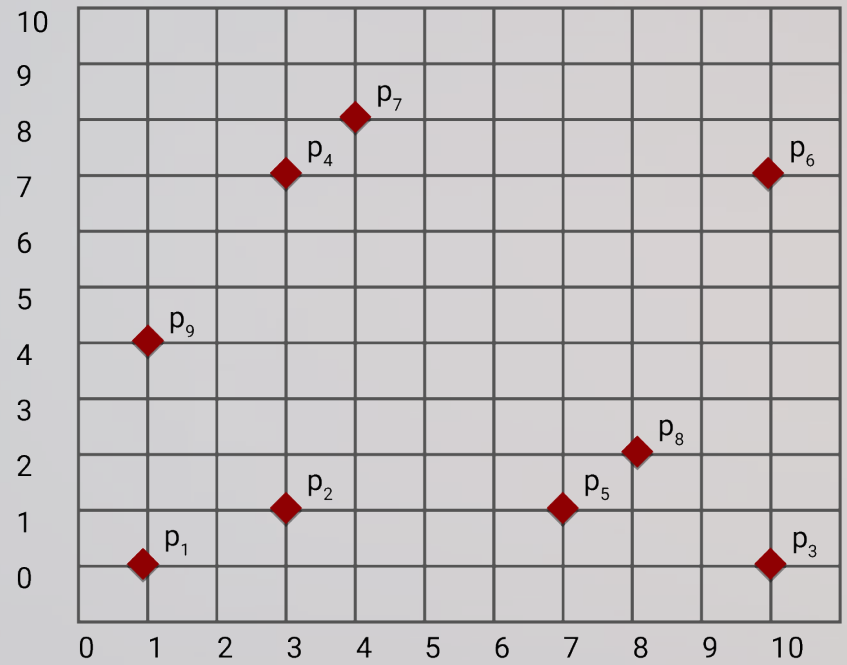
Each object is initially considered its own cluster. These clusters are merged with a distance metric until only one cluster remains. On a tree, this represents a bottom-up approach.

Divisive

Each object is initially considered as one cluster. These clusters are split with a distance metric until each object is its own cluster On a tree, this represents a top-down approach.

Hierarchical Clustering

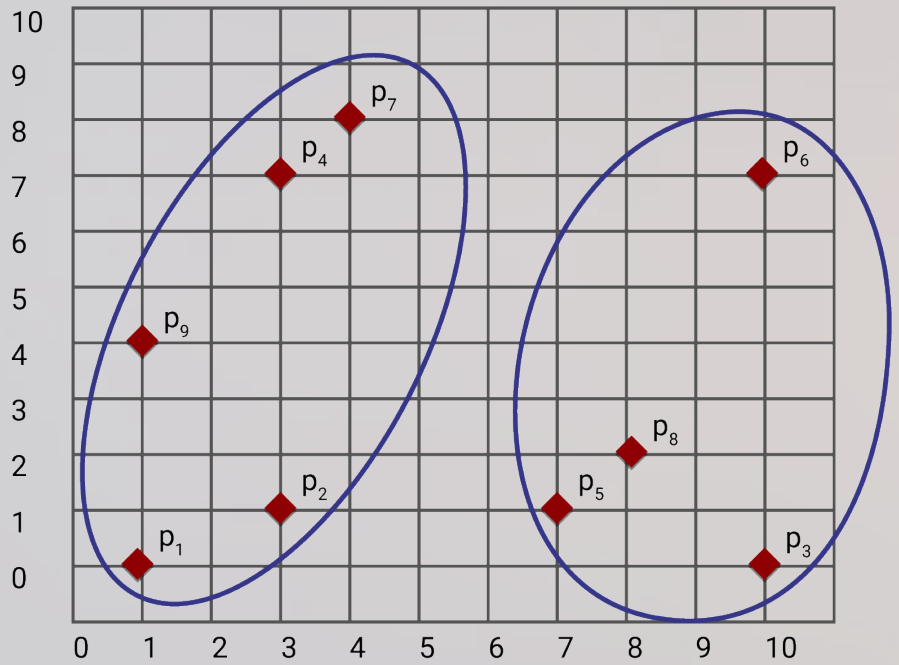
- Take the following two-dimensional dataset.



Point	x	y
p_1	1	1
p_2	3	2
p_3	9	1
p_4	3	7
p_5	7	2
p_6	9	7
p_7	4	8
p_8	8	3
p_9	1	4

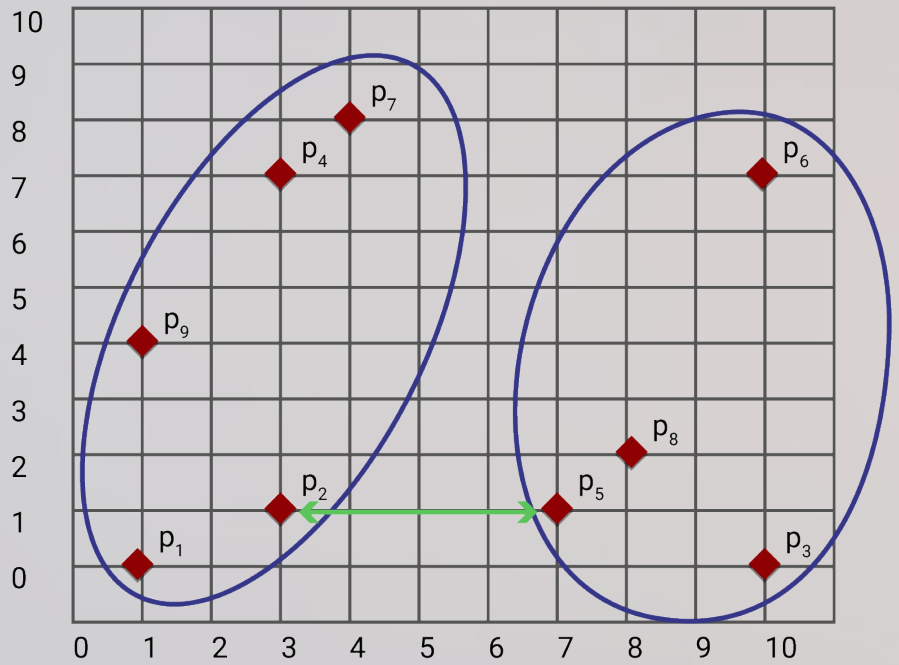
Hierarchical Clustering

- Split the data into two clusters.



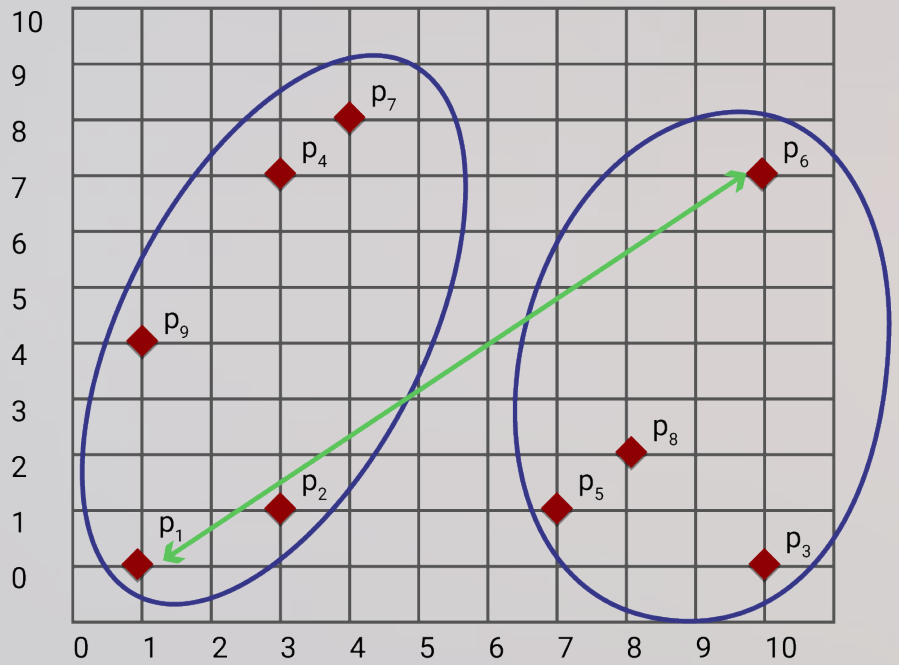
Hierarchical Clustering

- Min Linkage



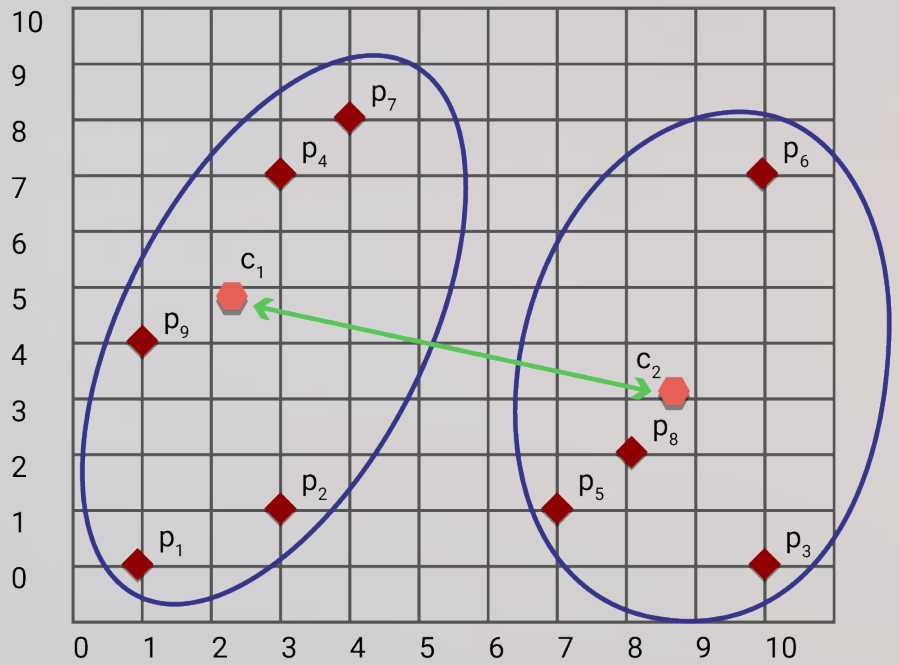
Hierarchical Clustering

- Max Linkage



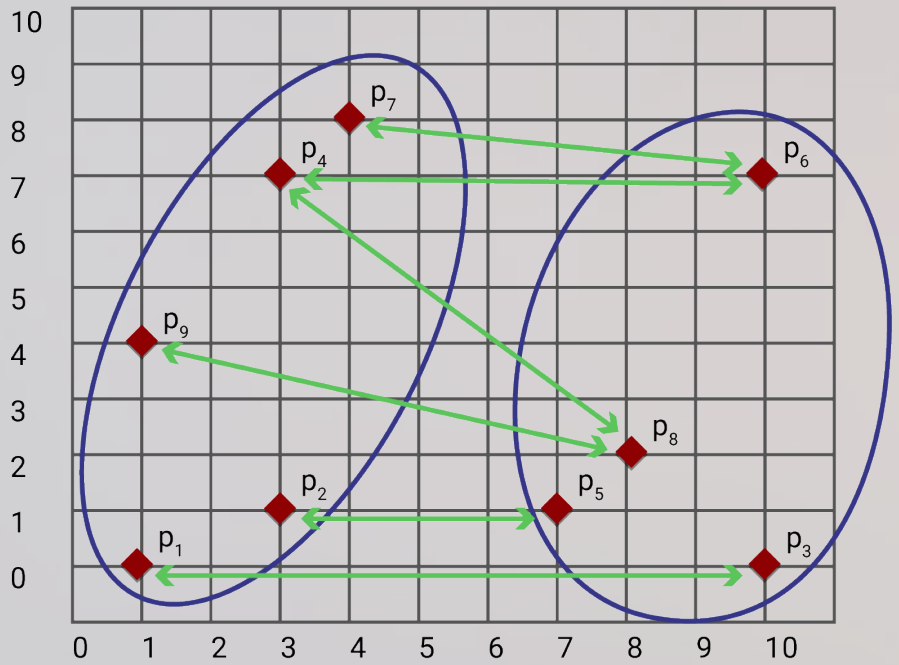
Hierarchical Clustering

- Centroid Linkage



Hierarchical Clustering

- Average Linkage



Hierarchical Clustering

- The ward linkage method computes the variance between the clusters rather than directly measuring the distance between classes.
- Compared to the distance-based measures described above, the Ward method is less susceptible to noise and outliers.

Fuzzy Clustering

Fuzzy Clustering is similar to the K-Means algorithm with 1 key difference:

- Data points within the fuzzy clusters do not belong to a singular cluster.
- Instead, each data point has a coefficient for each cluster, representing the likelihood of the datapoint being part of that cluster.
- The centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster.

NLP Applications

- Many traditional ML algorithms require structured numerical input to function effectively.
- For text classification, this means we have to find a numerical representation for textual data.
- This can come in the form of TF-IDF vectorisation or other vectorisation techniques (**covered next week**).
- In the context of NLP, classifiers can be used to perform sentiment analysis, intent classification and authorship attribution, just to name a few.
- Similarly, clustering techniques can help us perform document clustering and tasks like spam detection.