Advanced text analysis

Jisu Kim, PhD

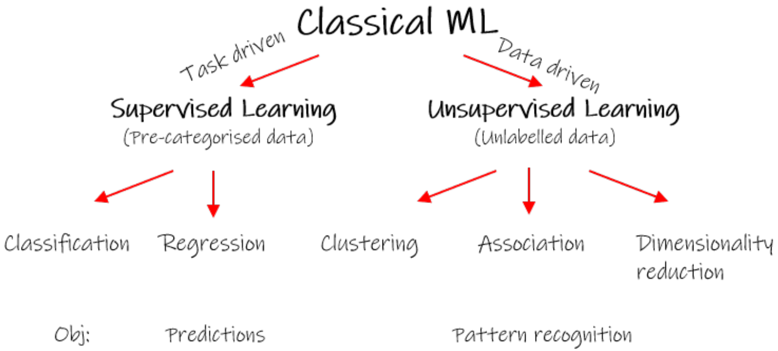
June 2024

Materials

Slides + Codes are available here:

`https://github.com/jisukimmmm/NCCR_MWQTA_2024`

2 types of approaches



Popular libraries and frameworks for building text classifiers:

- Scikit-learn: `https://scikit-learn.org/stable/`
- NLTK: `https://www.nltk.org/`
- spaCy: `https://spacy.io/`
- gensim: `https://pypi.org/project/gensim/`

Unsupervised Learning

- Lack of Labelled Data/response variables: In unsupervised learning, the algorithm only receives input data without corresponding labels or categories.
- **Objective**: It aims to find inherent structures or patterns in the data without being explicitly told what to look for.
- **Output**: the model produces output based solely on the input data, often in the form of patterns, clusters, semantic structures, or probabilistic distributions.

Application of Unsupervised ML

- Text clustering
- Topic modelling
- Anomaly detection

Examples of Unsupervised ML

- Principle Component Analysis (PCA)
- Clustering
  - K-means clustering
  - Model-based clustering: M-clust
  - Hierarchical clustering
- Co-occurrence matrix
- Word embeddings (Word2Vec, GloVe, FastText)
- Similarity
- Pointwise Mutual Information
- Topic Modelling
  - Latent Semantic Analysis
  - Latent Dirichlet Allocation

Principle Component Analysis (PCA)

PCA is a dimensionality reduction technique that transforms data from a high-dimensional space to a lower-dimensional space by identifying the directions (principal components) along which the variance of the data is maximised.

It can be a useful tool to provide a summary of text or as an input to further analysis.

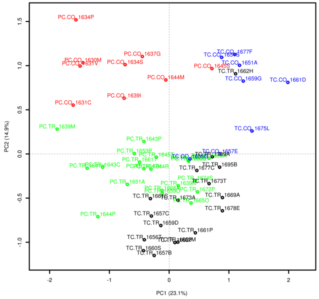


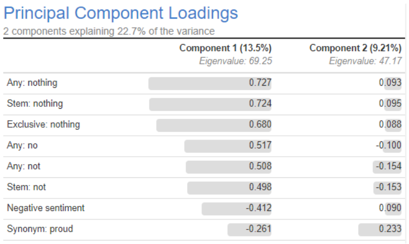
Figure: by Christof Schöch

Steps

1. **Standardise the data**: PCA assumes that the features have been standardized (i.e., centered around zero with a standard deviation of 1).
2. **Compute the Covariance Matrix**: to understand how the variables in the dataset are varying from the mean with respect to each other.
3. **Compute Eigenvectors and Eigenvalues**: The eigenvectors represent the directions (or axes) of maximum variance, while the eigenvalues represent the magnitude (amount of variance) in these directions. In short, eigenvectors show us where the main patterns are, and eigenvalues tell us how much those patterns matter.

Steps (Cont'd)

4. **Select Principal Components**: Sort the eigenvectors by their corresponding eigenvalues in descending order. The eigenvector with the highest eigenvalue is the first principal component, the one with the second-highest eigenvalue is the second principal component, and so on.
5. **Project the Data onto the Principal Components**: This transformation yields the principal component scores, which are the coordinates of the data points in the reduced-dimensional space.



Principal Component Loadings

2 components explaining 22.7% of the variance

	Component 1 (13.5%) Eigenvalue: 69.25	Component 2 (9.21%) Eigenvalue: 47.17
Any: nothing	0.727	0.093
Stem: nothing	0.724	0.095
Exclusive: nothing	0.680	0.088
Any: no	0.517	-0.100
Any: not	0.508	-0.154
Stem: not	0.498	-0.153
Negative sentiment	-0.412	0.090
Synonym: proud	-0.261	0.233

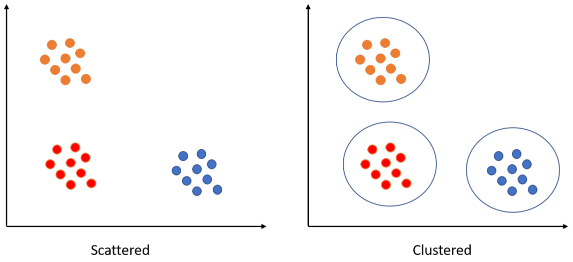
Figure: by Tim Bock

Advantages and Challenges

- Advantages
  - Dimensionality Reduction: PCA reduces the number of features in the dataset while retaining most of the variability, making it easier to visualize and analyze.
  - Noise Reduction: By keeping only the most significant components, PCA can help reduce noise in the data.
- Challenges
  - Interpretability: The new principal components are linear combinations of the original features, which can be difficult to interpret in a meaningful way.
  - Linearity Assumption: PCA assumes linear relationships among features, which may not capture more complex structures in the data.
  - Loss of Information: By reducing dimensions, some information from the original data is inevitably lost, which might be critical for certain analyses.

K-means clustering

K-means clustering is an algorithm used for document clustering. It groups similar documents together based on their feature vectors, allowing for the discovery of natural clusters within the text corpus. K-means clustering can help organise large collections of documents into coherent groups without requiring labelled data.



Assumptions

- **Clusters are Spherical**: It assumes that the clusters are roughly spherical in shape. This means that the distance from the cluster centre to any point in the cluster is relatively uniform in all directions.
- **Equal Cluster Sizes**: K-means performs best when the clusters have similar sizes or variances. If the clusters are significantly different in size, the algorithm might incorrectly merge smaller clusters into larger ones or split larger clusters into smaller ones.
- **Equal Cluster Density**: The algorithm assumes that the density of data points is relatively uniform across clusters. If some clusters are much denser than others, K-means might not correctly identify the clusters.
- **No Overlapping Clusters**: K-means assumes that clusters do not overlap. If clusters are overlapping, points near the boundary between clusters might be incorrectly assigned.

Advantages and Challenges

- Advantages
  - Scalability: K-means is computationally efficient and can handle large datasets.
  - Simplicity: Easy to implement and understand.
- Challenges
  - Choosing k: Determining the optimal number of clusters k can be challenging.
  - Sensitivity to Initial Centroids: The algorithm can converge to different results based on the initial placement of centroids.
  - Interpretability: Clusters might not always be easily interpretable, especially with high-dimensional text data.

Choosing k

Choosing the optimal number of clusters k in K-means clustering is a crucial step that can significantly affect the results of your analysis. Several methods can help determine the most appropriate value for k:

1. Elbow Method

The Elbow Method involves plotting the Within-Cluster Sum of Squares (WCSS) against the number of clusters k. The WCSS measures the total variance within each cluster. The idea is to choose k at the “elbow” point where the rate of decrease sharply slows down.

Within-Cluster Sum of Squares (WCSS)

The Within-Cluster Sum of Squares (WCSS), also known as the Sum of Squared Errors (SSE), is a measure used to evaluate the performance of clustering algorithms like K-means. It quantifies the total variance within each cluster. The goal of clustering is to minimize this value, as a lower WCSS indicates more compact and well-defined clusters.

Within-Cluster Sum of Squares (WCSS)

WCSS Formula For a given set of clusters, the WCSS is calculated as the sum of squared distances between each point and the centroid of the cluster it belongs to. Mathematically, it can be expressed as:

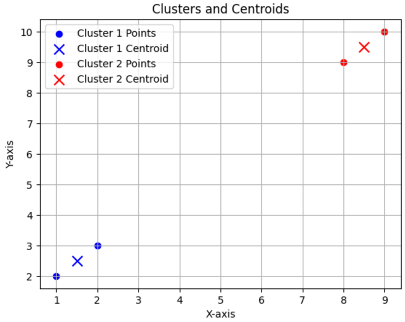
$$\text{WCSS} = \sum_{k=1}^{K} \sum_{i \in C_k} \|\mathbf{x}_i - \mu_k\|^2$$

Where:

- K is the number of clusters.
- C_k is the set of points in cluster k.
- $\mathbf{x}_i$ is a data point in cluster k.
- μ_k is the centroid of cluster k.
- $\|\mathbf{x}_i - \mu_k\|^2$ is the squared Euclidean distance between data point $\mathbf{x}_i$ and the centroid μ_k.

Explanation

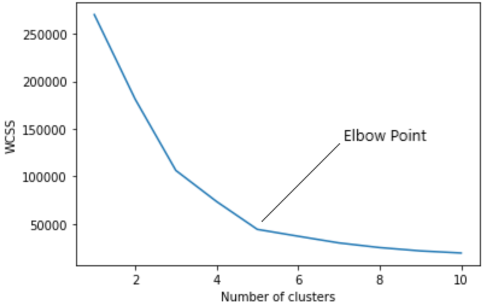
1. **Sum Over Clusters**: The outer sum $\left(\sum_{k=1}^{K}\right)$ iterates over all clusters.
2. **Sum Over Points in Each Cluster**: The inner sum $\left(\sum_{i \in C_k}\right)$ iterates over all points within a specific cluster k.
3. **Squared Distance**: $\|\mathbf{x}_i - \mu_k\|^2$ calculates the squared Euclidean distance between each data point $\mathbf{x}_i$ and the cluster centroid μ_k.



Steps

Steps:

1. Run K-means clustering for a range of k values (e.g., 1 to 10).
2. Calculate the WCSS for each k.
3. Plot the WCSS against k.
4. Look for the "elbow" point where the curve starts to flatten.

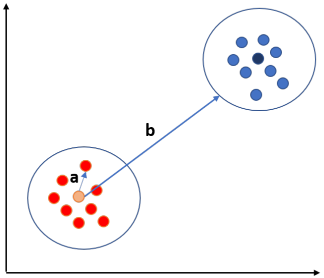


[1]

[1] source. `https://www.analyticsvidhya.com/`

2. Silhouette Score

It measures how similar each point is to its own cluster compared to other clusters. It ranges from -1 to 1, with higher values indicating better-defined clusters.



$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \qquad (1)$$

where:

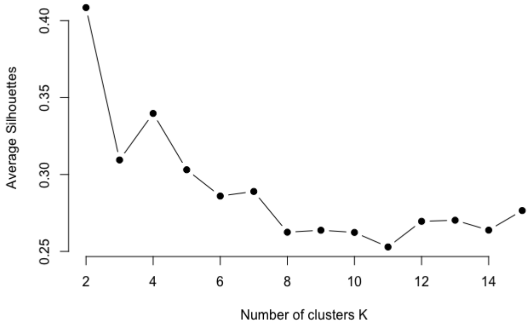
$a(i)$: The mean distance between a sample and all other points in the same cluster. It represents cohesion.

$b(i)$: The mean distance between a sample and all points in the nearest cluster that the sample is not a part of. It represents separation.

Steps

Steps:

1. Run K-means clustering for a range of k values.
2. Calculate the silhouette score for each k.
3. Choose the k with the highest silhouette score.



[2]

[2] source. `https://afit-r.github.io/`

Generalisation K-means: Model-based clustering

A statistical approach to clustering that assumes data are generated by a mixture of underlying probability distributions, each representing a different cluster.

Number of Clusters:The number of clusters k is typically determined using model selection criteria such as the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC).

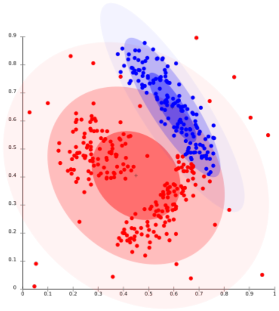


Figure: source. www.geeksforgeeks.org

Assumptions

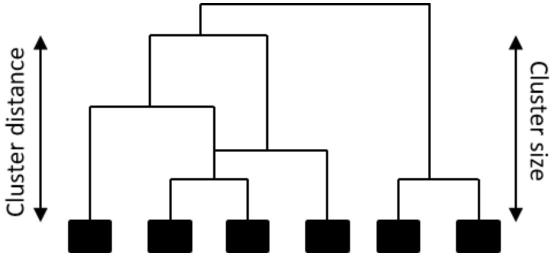
- Data Generation: The data is assumed to be generated from a mixture of underlying probability distributions.
- Cluster Shape: The shape of each cluster is determined by the form of the chosen distribution (e.g., Gaussian). For GMM, clusters are ellipsoidal with a specific mean and covariance.
- Independence: Given the cluster assignment, data points are assumed to be independent of each other.

Advantages and Limitations

- Advantages:
  - Flexibility: Can model clusters of different shapes, sizes, and orientations.
  - Probabilistic: Provides a probabilistic assignment of data points to clusters, which can be useful for understanding the uncertainty in the clustering.
- Limitations:
  - Complexity: More computationally intensive than simpler methods like K-means.
  - Initialisation Sensitivity: The EM algorithm can converge to local optima, so good initialisation is important.

Hierarchical clustering

Hierarchical clustering organises data points into a hierarchy of clusters. In text analysis, it can be used to build a dendrogram of documents based on their similarity or dissimilarity measures, allowing for a hierarchical organisation of the document collection. It is particularly useful when the number of clusters is not known a priori or when the data has a nested structure.



Hierarchical clustering (Cont'd)

It builds a hierarchy of clusters by recursively merging or splitting clusters until a termination criterion is met. There are two main approaches to hierarchical clustering: agglomerative (bottom-up) and divisive (top-down).

The divisive method can be less commonly used compared to the agglomerative approach because it requires more computational resources and may not always produce intuitive clustering solutions.

Steps

Here's how hierarchical clustering typically works, focusing on the agglomerative approach:

1. **Initialization**: Start with each data point as a single cluster.
2. **Calculate distances**: Compute the distance between each pair of clusters.
3. **Merge closest clusters**: Find the closest pair of clusters based on the chosen distance metric and merge them into a single cluster. The distance between clusters is usually determined by a linkage criterion, such as single linkage, complete linkage, or average linkage.
4. **Update distance matrix**: Recalculate the distances between the newly formed cluster and all other clusters. This step is necessary to update the distance matrix for the next iteration.

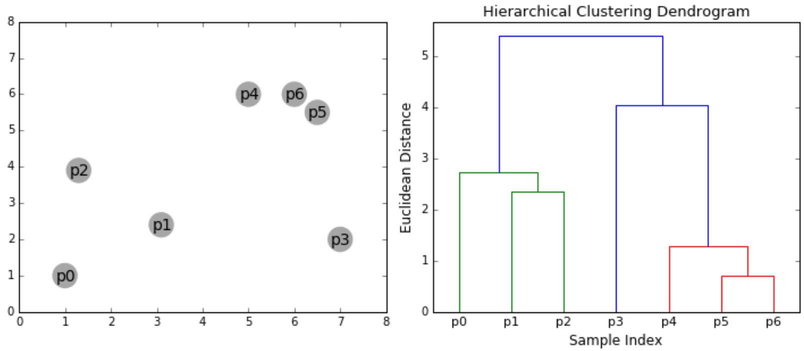


Figure: Example by David Sheehan

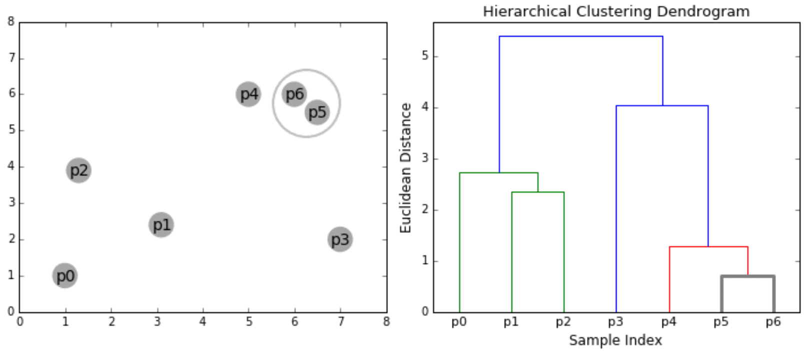


Figure: Example by David Sheehan

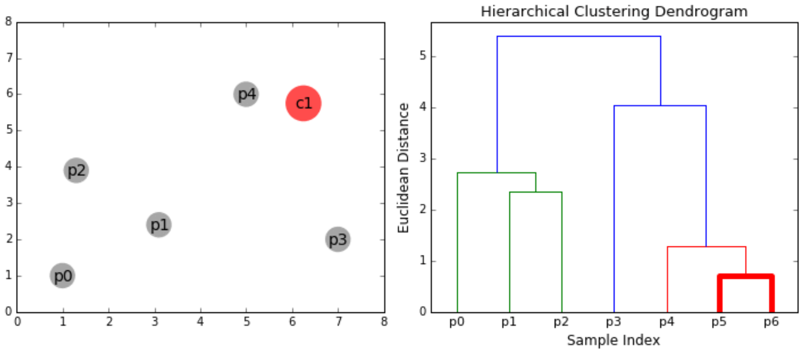


Figure: Example by David Sheehan

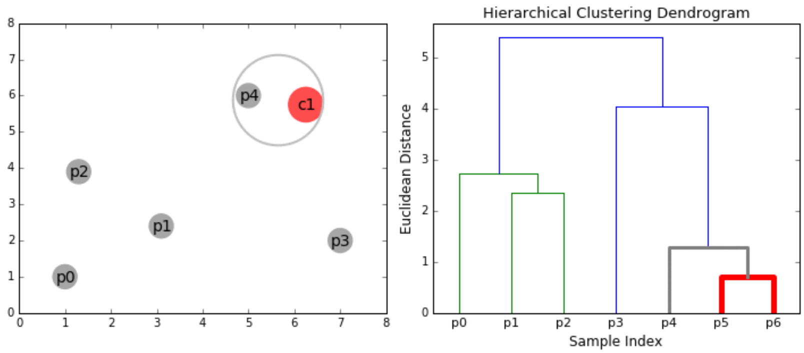


Figure: Example by David Sheehan

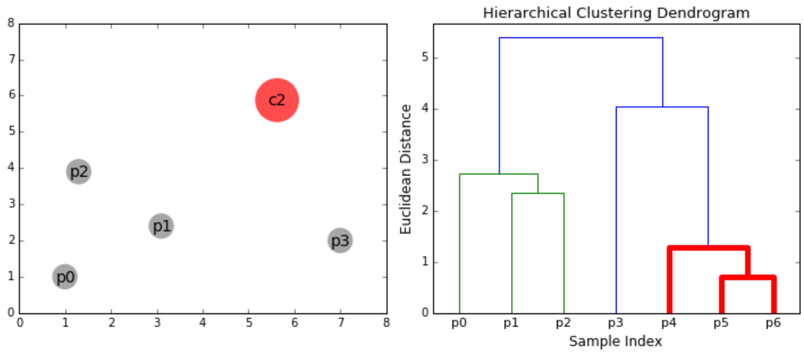


Figure: Example by David Sheehan

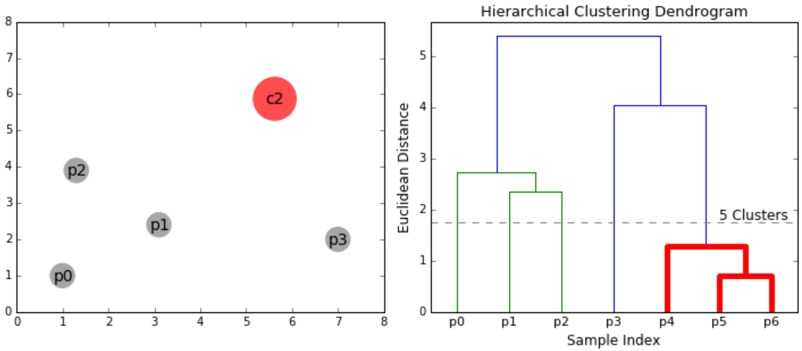


Figure: Example by David Sheehan

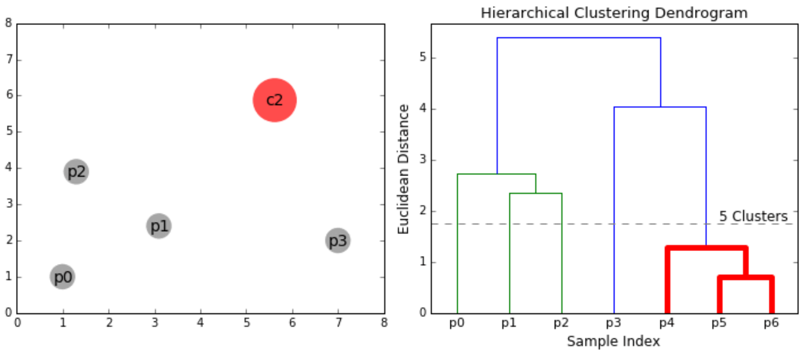


Figure: Example by David Sheehan

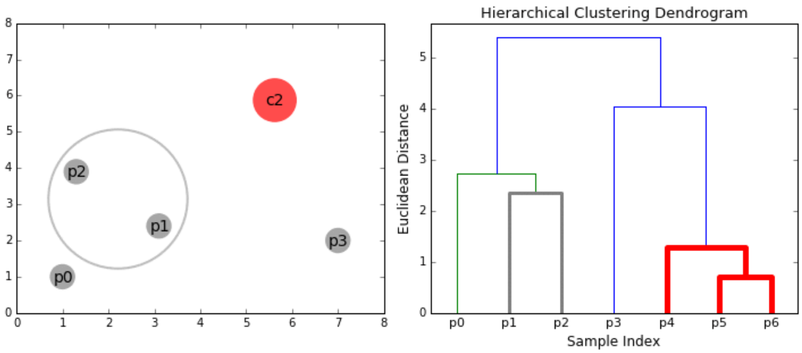


Figure: Example by David Sheehan

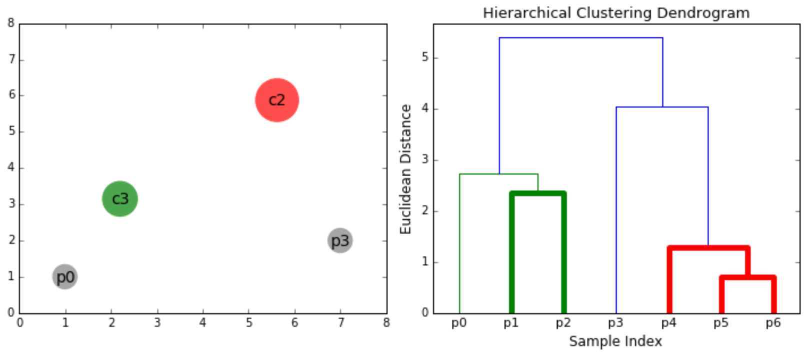


Figure: Example by David Sheehan

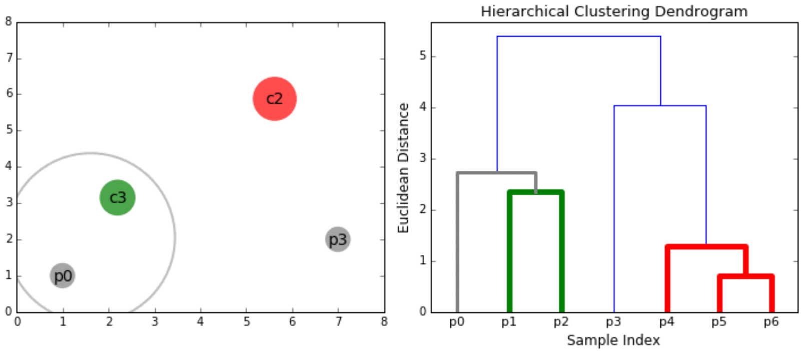


Figure: Example by David Sheehan

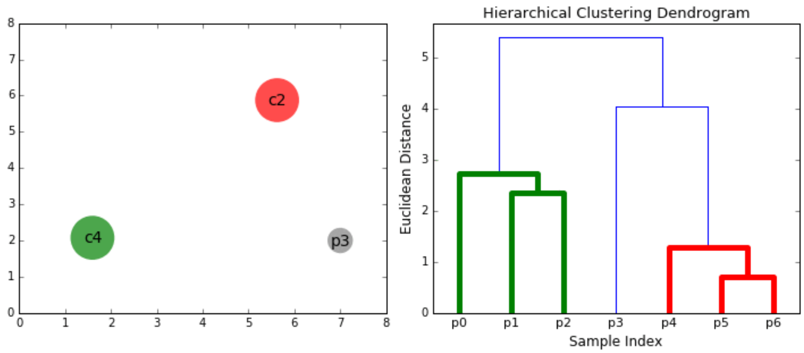


Figure: Example by David Sheehan

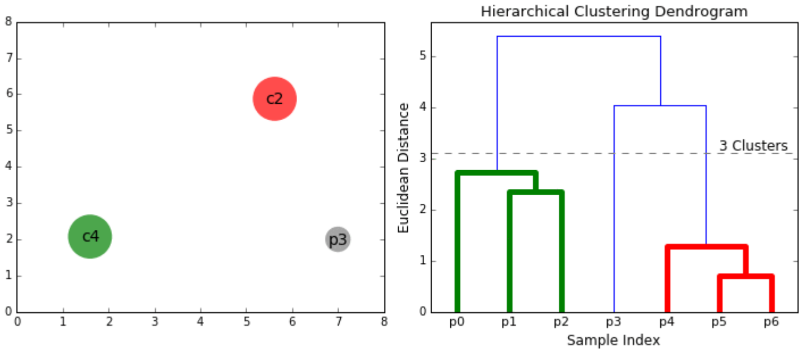


Figure: Example by David Sheehan

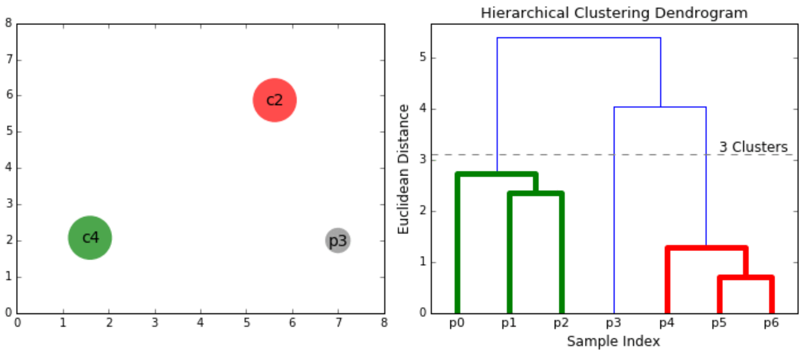


Figure: Example by David Sheehan

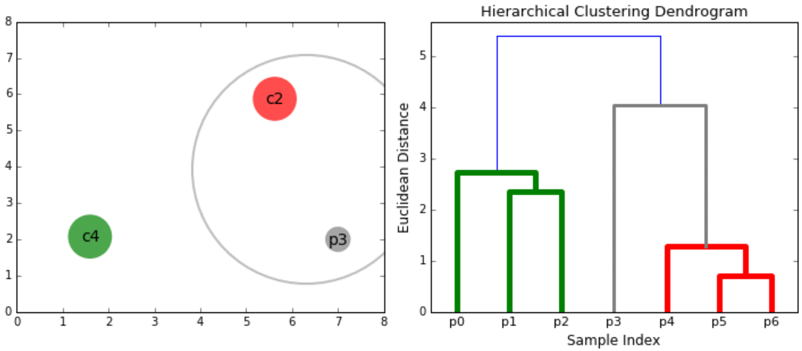


Figure: Example by David Sheehan

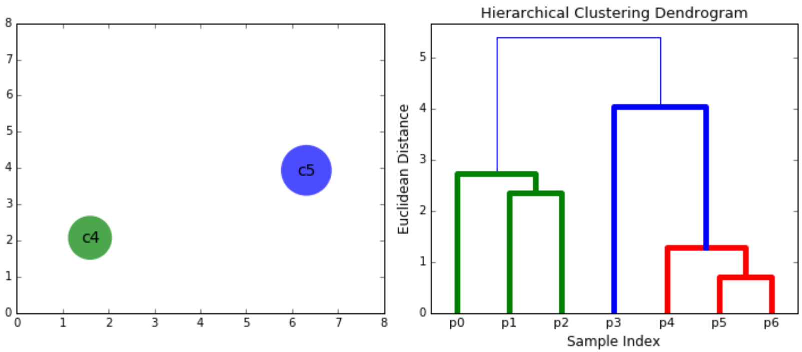


Figure: Example by David Sheehan

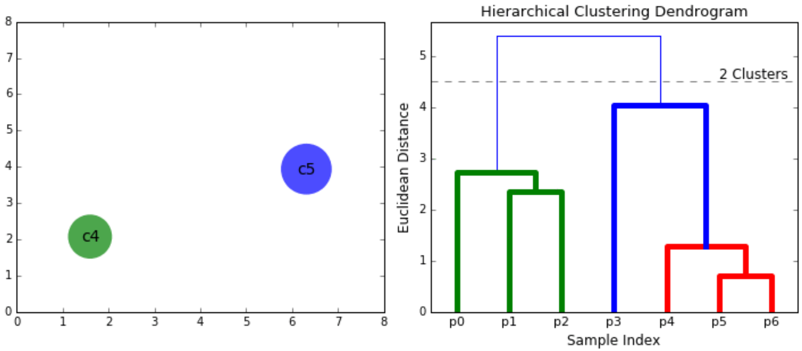


Figure: Example by David Sheehan

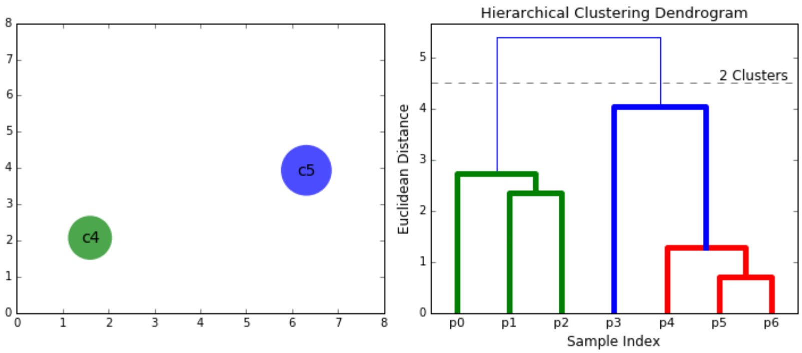


Figure: Example by David Sheehan

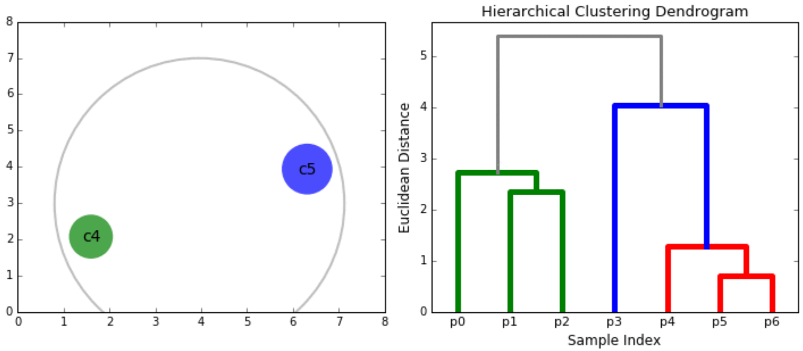


Figure: Example by David Sheehan

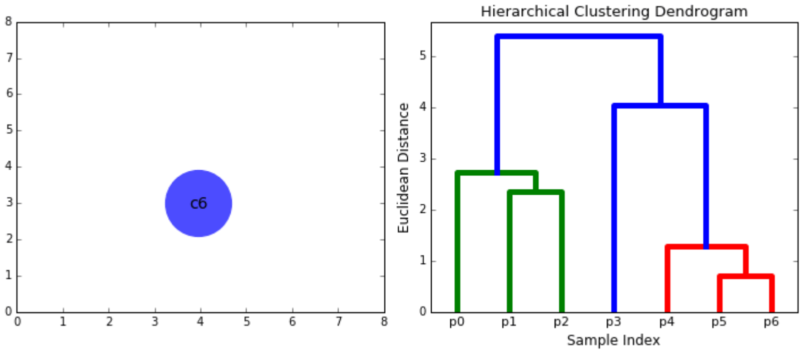


Figure: Example by David Sheehan

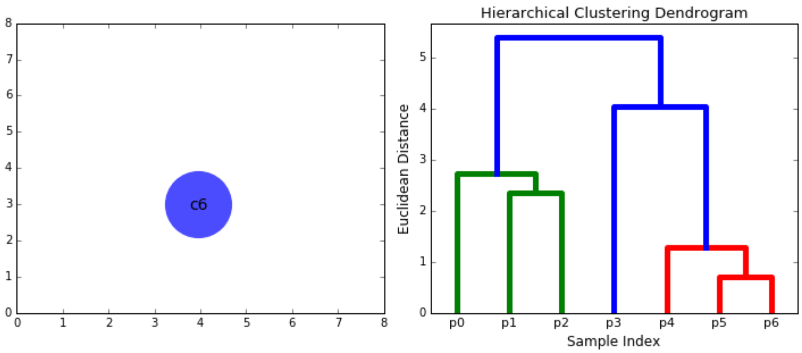
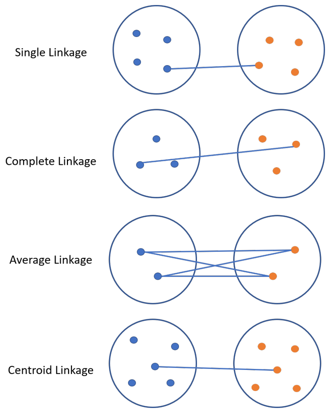


Figure: Example by David Sheehan

Linkage Criteria

- **Single Linkage (Minimum Linkage)**: Defines the distance between two clusters as the shortest distance between any two points in the clusters.
- **Complete Linkage (Maximum Linkage)**: Defines the distance between two clusters as the maximum distance between any two points in the clusters.
- **Average Linkage**: Defines the distance between two clusters as the average distance between all pairs of points in the clusters.
- **Centroid Linkage (UPGMA)**: Defines the distance between two clusters as the distance between their centroids (average points).



Steps (Cont'd)

5. **Repeat steps 3 and 4**: Continue merging clusters iteratively until only one cluster remains.
6. **Dendrogram**: A visual representation of the hierarchical clustering process. It shows the order in which clusters are merged and the distances between them. Each node in the dendrogram represents a cluster, and the height of the branches represents the distance between clusters.
7. **Selecting the number of clusters**: To determine the optimal number of clusters, one can use the dendrogram to identify the level at which merging clusters results in significant increases in distance.

Advantages

- **No need to specify the number of clusters**: Unlike k-means clustering, it does not require specifying the number of clusters beforehand. The dendrogram provides a visual representation, allowing users to choose the number of clusters based on their interpretation.
- **Hierarchical structure**: it produces a tree-like structure (dendrogram) that shows the relationships between clusters at different levels of granularity. This can provide insights into the natural grouping of data.
- **Interpretability**: The dendrogram allows for easy interpretation of the clustering results.
- **Robust to noise**: Hierarchical clustering can be more robust to noise and outliers compared to partitioning methods like k-means, especially when using linkage criteria that are less sensitive to outliers.

Challenges

- **Computationally intensive**: It can be computationally intensive, especially when dealing with large datasets.
- **Sensitive to noise**: Hierarchical clustering can be sensitive to noise and outliers, especially when using linkage criteria that are sensitive to small variations in distance.
- **Lack of flexibility in merging**: Once clusters are merged in hierarchical clustering, they cannot be split again in subsequent steps, leading to a lack of flexibility in the clustering process. This can result in sub-optimal clustering solutions in some cases.

Rand Index[3]

Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the (adjusted or unadjusted) Rand index is a function that measures the similarity of the two assignments.

Examples.
rand_score([0, 0, 1, 1], [1, 1, 0, 0])
Outcome: 1.0
rand_score([0, 0, 0, 1, 1, 1], [0, 0, 1, 1, 2, 2])
Outcome: 0.66

[3]Source. `scikit-learn.org/`

Co-occurrence matrix

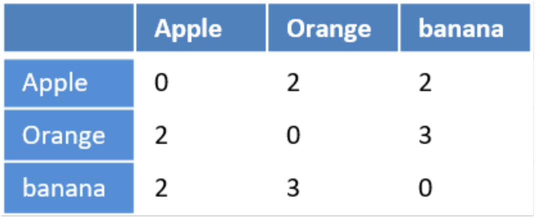
A co-occurrence matrix is a square matrix that represents the frequency of co-occurrence of terms in a given text corpus. Each row and column of the matrix corresponds to a unique term in the vocabulary, and the value at the intersection of row *i* and column *j* indicates how often term *i* and term *j* appear together within a specified context window or document.

Example:

Text 1: "apple orange apple banana"

Text 2: "orange banana banana banana"

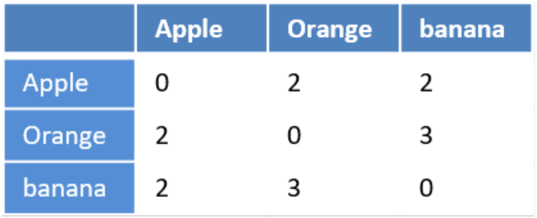
Text 3: "apple banana banana apple"



	Apple	Orange	banana
Apple	0	2	2
Orange	2	0	3
banana	2	3	0

Co-occurrence matrix (Cont’d)

Example:
Text 1: “apple orange apple banana”
Text 2: “orange banana banana banana”
Text 3: “apple banana banana apple“



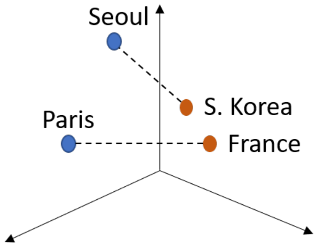
	Apple	Orange	banana
Apple	0	2	2
Orange	2	0	3
banana	2	3	0

For example, the cell at row ”apple” and column ”orange” has a value of 2, indicating that ”apple” and ”orange” co-occur twice within the corpus. The matrix is symmetric because co-occurrence is being measured bidirectionally.

Word Embeddings

Word embeddings techniques such as Word2Vec, GloVe, and FastText are used to represent words as dense, low-dimensional vectors in a continuous vector space.

These embeddings capture semantic relationships between words based on their co-occurrence patterns in large text corpora.



Word2Vec

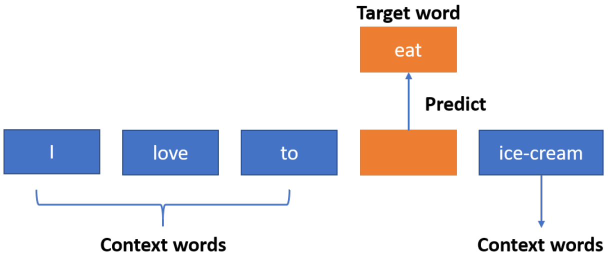
It's for learning vector representations of words from large text corpora. Developed by a team of researchers at Google, Word2Vec models represent words as dense, low-dimensional vectors in a continuous vector space, where semantically similar words are located close to each other.

There are two main architectures for training Word2Vec models: Continuous Bag of Words (CBOW) and Skip-gram.

Both architectures aim to learn distributed representations of words by predicting context words given a target word (CBOW) or predicting a target word given context words (Skip-gram).

CBOW

1. **Context Matters:** Imagine predicting a missing word in a sentence by looking at the words around it. For example, in "I love to ___", you might guess "eat" because it fits the context.
2. **Learning from Context:** CBOW does something similar, learning from words around a target word to predict it. It learns from lots of examples in a text corpus.
3. **Predicting the Target Word:** CBOW predicts the target word based on its context. For example, given "I", "love", and "to" as input, it might predict "eat" as the missing word.



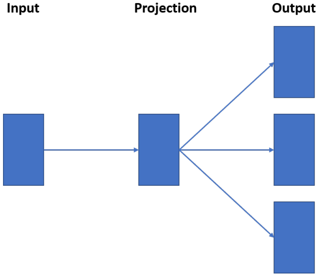
CBOW (Cont'd)

4. **Training the Model:** CBOW uses a neural network to learn from examples and adjust its parameters (weights) to make better predictions over time.
5. **Creating Word Embeddings:** CBOW learns to represent words as fixed-size vectors called word embeddings, capturing their meanings based on context.
6. **Using Word Embeddings:** Once trained, CBOW provides word embeddings for any word. These embeddings help understand word relationships and solve various language tasks.

In essence, CBOW is a way for a computer to learn from context to understand word usage and represent that understanding as word embeddings.

Skip-gram

1. **Word Relationships:** Imagine understanding words by looking at how they relate to other words around them. For instance, understanding "king" by noting its relationship with "queen" and "royal".
2. **Learning Context:** Skip-gram does just that. It learns to predict the context words around a given target word.



3. **Predicting Context:** Given a target word like "king", Skip-gram tries to predict the surrounding context words like "queen", "royal", or "monarch".

Here, the target word is used to predict the context.

Skip-gram (Cont'd)

1. **Training the Model:** Skip-gram uses a neural network to learn from many examples and adjust its parameters (weights) to make better predictions.
2. **Creating Word Embeddings:** As it learns, Skip-gram creates word embeddings, which are compact representations of words based on their context.
3. **Using Word Embeddings:** Once trained, Skip-gram provides word embeddings that capture relationships between words, helping in various language tasks like understanding word similarities or analogies.

In essence, Skip-gram learns from how words are used in context to create word embeddings that capture their meanings and relationships.

Global Vectors for Word Representation (GloVe)

- GloVe is used for obtaining vector representations for words developed by researchers at Stanford University.
- It works by aggregating global word co-occurrence statistics from a corpus and using them to learn word vectors.
- The key idea behind GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.
- By incorporating both global and local information about word co-occurrences, GloVe produces embeddings that capture both semantic and syntactic relationships between words.

FastText

- FastText is an extension of word2vec, a technique developed by Facebook AI Research.
- Unlike word2vec, which represents words as vectors of fixed dimensions, FastText represents each word as a bag of character n-grams.
- It learns vector representations for words by considering subword information, which helps in capturing morphological and semantic similarities even for rare words or misspellings.
- It is particularly useful for handling out-of-vocabulary words and dealing with languages with rich morphology[4].

[4] the study of the internal structure of words: e.g., unreachable −> un, reach, able

What to choose?

- **Choose Word2Vec** if you need a fast and efficient model that produces high-quality embeddings and your dataset is sufficiently large.
- **Choose GloVe** if you want to capture global co-occurrence statistics and have the computational resources to handle large-scale training.
- **Choose FastText** if you need to handle out-of-vocabulary words, work with morphologically rich languages, or require efficient training and inference on large datasets.

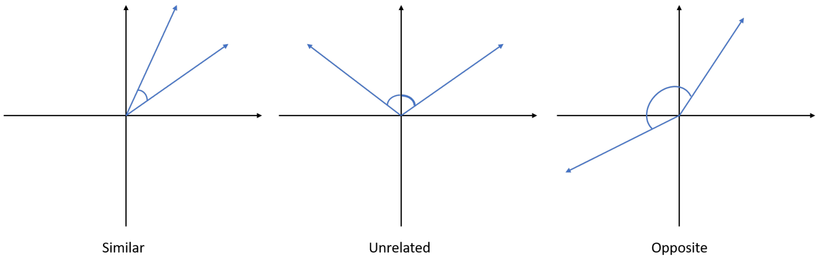
Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. In the context of text analysis, cosine similarity is often used to determine how similar two documents are based on their content.

The vectors typically represent the term frequencies (TF), term frequency-inverse document frequency (TF-IDF).

Other types of similarity measures can also be used. E.g., Jaccard similarity, Euclidean Distance...

Cosine similarity



$$\text{Cosine Similarity} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|} \tag{2}$$

Word embeddings vs. Similarity

Table: Comparison of Word Embeddings and Similarity

Word Embeddings	**Similarity**
Dense vector representations of words in a continuous space, capturing semantic relationships.	Measures how similar two word embeddings are, reflecting their semantic similarity.
Learned from text corpora using techniques like Word2Vec, GloVe, or FastText.	Often quantified using cosine similarity, where higher values indicate closer similarity.
Useful for tasks like NLP, sentiment analysis, and document classification.	Helps understand word relationships, find similar words, or solve analogies.

Examples

Word embedding:
In a word embedding space, the vectors for “king” and “queen” might be close together because they often appear in similar contexts and have similar semantic meanings.

Similarity:
If the cosine similarity between the embeddings of “king” and “queen” is high, it indicates that these words are closely related or similar in meaning in the embedding space.

Pointwise Mutual Information

Pointwise Mutual Information (PMI) is a measure used to determine the association strength between two terms in a corpus. It compares the probability of the co-occurrence of two terms with their individual probabilities of occurrence. PMI is commonly used to identify meaningful associations between words or phrases.

$$\text{PMI}(w_1, w_2) = \log_2 \left(\frac{P(w_1, w_2)}{P(w_1) \cdot P(w_2)} \right) \tag{3}$$

Where:
w_1 and w_2 are the two terms being compared.
PMI(w_1,w_2)= is the probability of the co-occurrence of w_1 and w_2 in the corpus.
P(w_1) and P(w_2)are the individual probabilities of occurrence of w_1 and w_2 in the corpus.

Pointwise Mutual Information

PMI produces a numerical value representing the strength of association between two terms:

- Positive values indicate that the terms co-occur more frequently together than expected by chance, suggesting a strong association.
- Negative values indicate that the terms co-occur less frequently together than expected by chance, suggesting a weak or inverse association.
- PMI values closer to zero indicate that the terms occur together at a frequency similar to what would be expected by chance.

PMI vs. Similarity

Table: Comparison of PMI and Cosine Similarity

PMI	**Cosine Similarity**
Measures the association between two words based on co-occurrence probabilities.	Measures the similarity between two vectors based on the cosine of the angle between them.
Indicates how much more likely two words co-occur than by chance.	Indicates how similar two word vectors are in terms of their direction.
Values can be positive or negative. High values indicate strong association.	Values range from -1 to 1. High values indicate high similarity.
Useful for identifying word associations and collocations.	Useful for comparing word or document embeddings, and for tasks like information retrieval and clustering.

Examples

Word embedding:
In a word embedding space, the vectors for “king” and “queen” might be close together because they often appear in similar contexts and have similar semantic meanings.

Similarity:
If the cosine similarity between the embeddings of “king” and “queen” is high, it indicates that these words are closely related or similar in meaning in the embedding space.

PMI:
If the PMI between “king” and ”queen” is high, it indicates that these words frequently co-occur in the corpus more often than would be expected by chance.

Topic modelling

Topic modelling is used to identify hidden themes or topics within a collection of documents. It helps in discovering the abstract topics that occur in a collection of text.

Applications:

- Exploring large text corpora to identify main themes.
- Summarising and organising documents.
- Improving information retrieval systems by categorising documents.

Example: Analysing a collection of news articles to discover topics like “politics,” “economics,” and “technology” without prior knowledge of these categories.

Latent Dirichlet Allocation (LDA)

LDA is a generative probabilistic model used for topic modeling in text analysis. It assumes that each document is a mixture of topics, and each topic is a probability distribution over words. LDA automatically discovers latent topics present in a collection of documents and assigns each document a distribution over these topics, enabling researchers to explore the underlying themes within the text corpus.

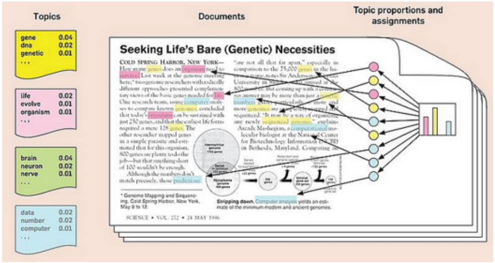


Figure: Source. Kim et al., 2019

LDA-Steps

1. **Initialisation**: Choose the number of topics you want the model to identify.
2. **Training**: LDA takes a collection of documents as input and iteratively updates its estimates of two probability distributions:
   - The distribution of words for each topic.

$$p(\text{word } w | \text{ topic } t) \tag{4}$$

   - The distribution of topics for each document.

$$p(\text{topic } t | \text{document } d) \tag{5}$$

3. **Inference**: Once trained, LDA can infer the topic mixture for new documents. It does this by analysing the distribution of words in the new document and comparing it to the learned distributions of topics.

Choosing number of topics

- Elbow method
- Perplexity score: Measures how well the model predicts the test data. Lower perplexity indicates a better model fit. It is not directly interpretable like accuracy; it's more useful for comparing different models or configurations on the same dataset.
- Coherence: Measures the semantic similarity between high scoring words in a topic. Higher coherence scores generally indicate more interpretable topics. (Score range 0-1)

Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) is a technique used to analyze the relationships between a set of documents and the terms they contain. LSA assumes that words that appear in similar contexts tend to have similar meanings.

It is based on the principle of reducing the dimensionality of the term-document matrix through singular value decomposition (SVD), a linear algebra technique.

LSA 2

Term-Document Matrix: LSA starts by constructing a term-document matrix, where rows represent terms and columns represent documents. The cells of this matrix contain some measure of the importance of each term in each document, often using term frequency-inverse document frequency (TF-IDF) weighting.

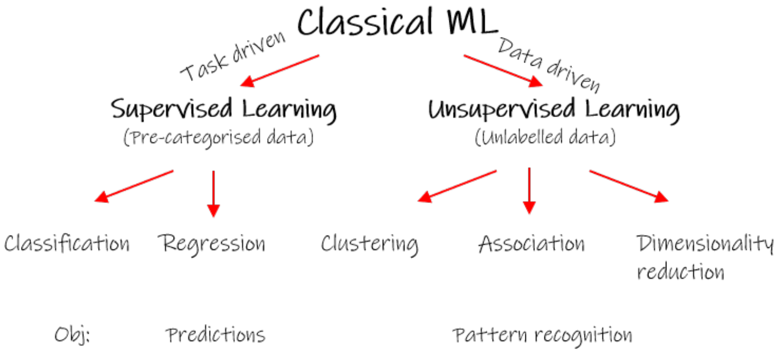
Dimensionality Reduction: LSA then applies a dimensionality reduction technique, typically singular value decomposition (SVD), to this matrix. SVD breaks down the matrix into three matrices: U, Σ, and V^T. The Σ matrix contains singular values, which represent the importance of each dimension.

LSA 3

Conceptualization: By reducing the dimensionality of the term-document matrix, LSA identifies underlying "concepts" or "latent semantics" in the text. These concepts represent relationships between terms and documents that are not immediately obvious from the raw data.

Document and Term Similarity: LSA can calculate the similarity between documents or terms based on their vector representations in the reduced-dimensional space. This similarity is often measured using techniques like cosine similarity.

Supervised Learning in Text analysis



Overview of Supervised Learning

Supervised learning is a machine learning paradigm where the algorithm learns from labeled training data, consisting of input-output pairs.

The goal is to learn a mapping from input variables to output variables, making predictions on unseen data based on the learned patterns from the training data.

Principles of Supervised Machine Learning

Labeled Data: Supervised learning requires a labeled dataset where each example is associated with a known output or target label. The algorithm learns patterns and relationships from the input-output pairs provided in the training data.

Classification and Regression: Text analysis tasks often involve classification (assigning documents to predefined categories or labels) or regression (predicting continuous values based on textual features).

For example, document categorisation, sentiment analysis, and named entity recognition are classification tasks, while predicting the popularity of a news article based on its content length and keywords is a regression task.

Application of Supervised Learning

- Spam detection
- Movie ratings
- Sentiment analysis
- Topic classification
- Document categorisation

Examples of Supervised Learning algorithms

- Naive Bayes,
- Support Vector Machines,
- K-Nearest Neighbours (KNN)
- Decision Trees and Random Forests
- Topic classification

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem with the "naive" assumption of independence among features. It's commonly used for text classification tasks like spam detection and sentiment analysis.

"naive" because it assumes that the presence of one feature in a class is independent of the presence of any other feature, which is often not entirely true in real-world data.

Assumptions of Naive Bayes

- Independence of Features: The features are considered conditionally independent of each other, given the class label.
- Normal Distribution of Continuous Features: In the case of continuous features, it is assumed that they follow a normal distribution within each class.
- Multinomial Distribution of Discrete Features: If a feature is discrete, it is assumed to follow a multinomial distribution within each class.
- Equal Importance of Features: All features are assumed to have an equal contribution to the prediction of the class label.
- No missing data

Bayes theorem

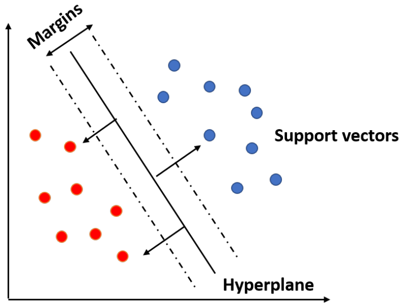
Bayes' theorem calculates the probability of a hypothesis (H) given some observed evidence (E). In the context of text analysis, the hypothesis is the class label (e.g., positive sentiment, spam), and the evidence is the presence of certain words or features in the text.

$$P(H|E) = (P(E|H) * P(H))/P(E) \tag{6}$$

$P(H|E)$ is the probability of the hypothesis given the evidence (posterior).
$P(E|H)$ is the probability of the evidence given the hypothesis (likelihood).
P(H) is the prior probability of the hypothesis.
P(E) is the prior probability of the evidence.

Support Vector Machines (SVM)

SVM is used for classification and regression tasks. In text analysis, SVMs are often employed for tasks like document categorization and text classification, where they find an optimal hyperplane to separate different classes.

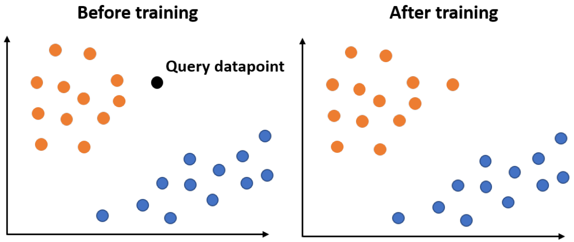


K-Nearest Neighbour (KNN)

KNN is a simple algorithm that classifies data points based on the majority class of their nearest neighbours.

It's a lazy learning algorithm, meaning it doesn't explicitly learn a model during training. Instead, it memorises the training instances and uses them to make predictions at runtime.

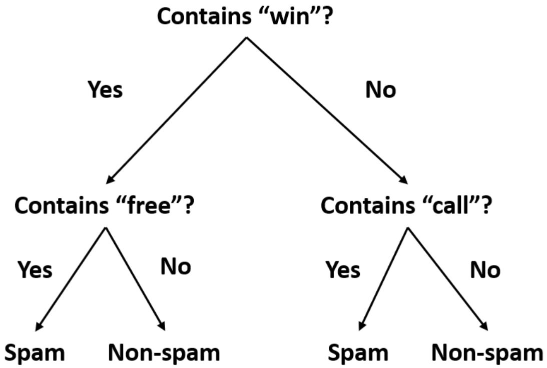
Non-Parametric: KNN doesn't assume anything about the underlying data distribution.



The performance depends on factors like the choice of distance metric, the value of k, and the distribution of the data.

Decision Trees and Random Forests

Decision trees and Random Forests are used for both classification and regression tasks. They work by splitting the data into subsets based on the value of input features, creating a tree-like model of decisions.



Decision trees and random forests

- a rule-based supervised learning
- Both for classification and regression tasks
- Classification Tree: provides a discrete output where the output node (leaf) typically set to the most common value.
- Regression Tree: provides a continuous output where the output node (leaf) value typically set to the mean value in data.
- Need some way to prune the tree!

Random Forest

- One of the most popular variants of decision trees.
- Addresses overfitting by training multiple trees on subsampling of features among other things.
- Each tree is trained on a random sample of the data (with replacement).
- **Feature Randomness**: At each split, a random subset of features is considered to create more diverse trees.

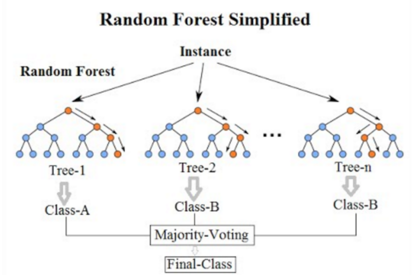


Figure: by Will Koehrsen

Gini Impurity

Gini impurity is a metric used to evaluate the quality of a split in decision trees. It measures how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset.

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{n} p_i^2 \tag{7}$$

Where p_i is the probability of a randomly selected element being classified to a particular class i, and n is the total number of classes.

Gini impurity (Cont'd)

- Perfectly Pure Node: If all elements are of the same class (pure node), the Gini impurity is 0. This means there's no impurity or uncertainty in the node.
- Maximum Impurity: If the classes are perfectly mixed (e.g., 50% one class and 50% the other in binary classification), the Gini impurity reaches its maximum value. For two classes, this maximum is 0.5.

Topic classification

Supervised topic classification is a task commonly addressed in text transformer models.

We will look at it more closely tomorrow!

Handling Imbalanced Data:

Strategies for dealing with imbalanced datasets:

- Resampling techniques (e.g., oversampling[5], under-sampling[6])
- Synthetic data generation: involves creating artificial data points for the minority class in order to balance the class distribution.
- Cost-sensitive learning approaches: involves modifying the learning algorithm to account for the imbalance in class distribution by assigning different misclassification costs to different classes. Instead of treating all misclassifications equally, cost-sensitive learning penalises misclassifications of the minority class more heavily.

[5] Increase the number of instances in the minority class by duplicating or creating synthetic samples.

[6] Reduce the number of instances in the majority class by randomly removing samples.

Performance Evaluation

- For classification tasks, metrics like accuracy, precision, recall, F1-score, and confusion matrix are commonly used.
- Regression tasks may use metrics like mean squared error (MSE) or mean absolute error (MAE).

		Predicted	
		Positive	**Negative**
Actual	**Positive**	TP	FN
	Negative	FP	TN

Table: Confusion Matrix: provides a detailed breakdown of the model's predictions compared to the actual labels. The matrix helps identify not just the overall accuracy of the model, but also where it is making errors, providing insights into the types of mistakes and their frequencies.

F1 score

F1-score measures the accuracy;
$F1score = 2 * \frac{precision * recall}{precision + recall}$

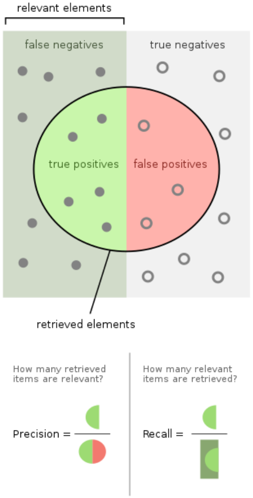
- Precision: How many selected items are relevant?

$$\frac{Truepositive}{Truepositive + Falsepositive}$$

- Recall: How many relevant items are selected?

$$\frac{Truepositive}{Truepositive + Falsenegative}$$

The score ranges from its worst score of 0 to its best score 1.



Source. Wiki

MSE and MAE

evaluate the performance of regression models.

- MSE: metric to measure the average squared difference between the actual and predicted values.

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

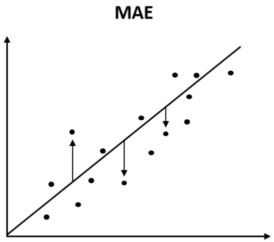
- MAE: measures the average absolute difference between the actual and predicted values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Where:
n: The number of data points.
y_i: The actual value of the i-th data point.
$\hat{y}_i$: The predicted value of the i-th data point.

MSE vs. MAE

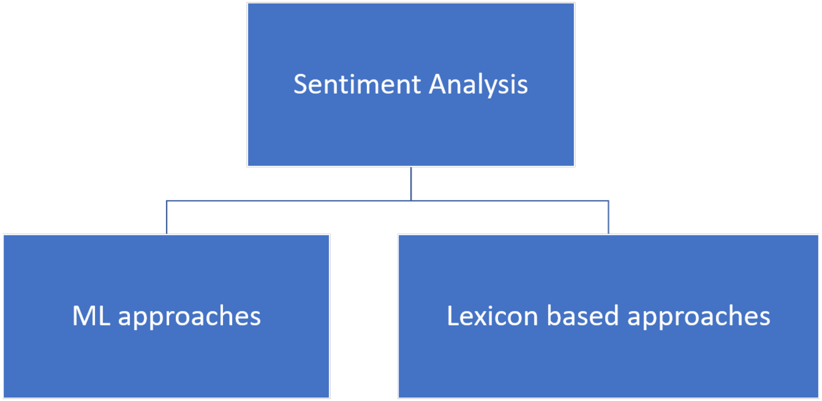
- Robustness to Outliers: MAE treats all errors equally, making it less sensitive to outliers compared to MSE.
- Interpretability: MAE is more interpretable because it represents the average error in the same units as the data.



Sentiment analysis

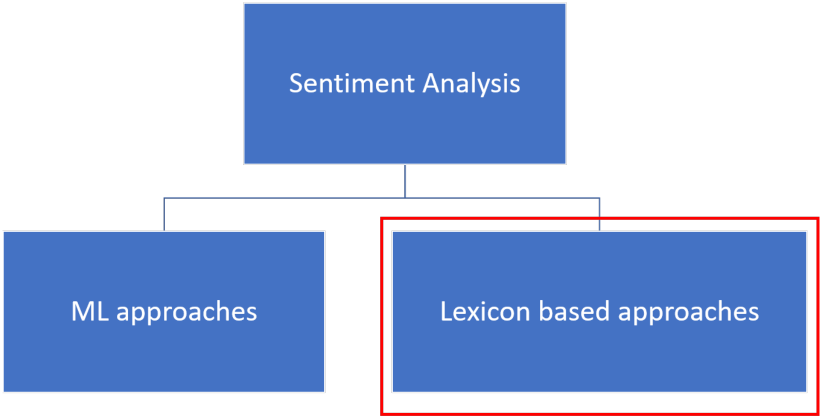
Sentiment analysis, also known as opinion mining, is a process of computationally identifying and categorising opinions expressed in a piece of text to determine whether the sentiment expressed is positive, negative, or neutral. The main goal of sentiment analysis is to understand the overall attitude or emotional tone conveyed within the text.

Sentiment analysis



Source. adapted from Medhat et al., 2014

Sentiment analysis



Source. adapted from Medhat et al., 2014

Lexicon-based

It identifies, classifies, and scores specific keywords based on predetermined lexicons. Lexicons are compilations of words representing the writer's intent, emotion, and mood. It assigns sentiment scores to positive and negative lexicons to reflect the emotional weight of different expressions. To determine if a sentence is positive, negative, or neutral, the software scans for words listed in the lexicon and sums up the sentiment score. The final score is compared against the sentiment boundaries to determine the overall emotional bearing.

$$\text{Sentiment score} = \frac{\#\text{ of positive words} - \#\text{ of negative words}}{\text{Total number of words}} \tag{8}$$

Simple example

Consider the sentence: "I love this product, it's amazing!"

1. Lexicon: Assume a simple lexicon:
   "love" $= +2$
   "amazing" $= +3$
2. Preprocessing: Tokenize and lowercase the sentence:
   Tokens: ["i", "love", "this", "product", "it's", "amazing"]
3. Sentiment Calculation:
   "love" matches with $+2$
   "amazing" matches with $+3$
   Sum $= 2 + 3 = +5$
4. Result Interpretation:
   The overall sentiment score is $+5$, indicating a positive sentiment.

Examples of Lexicon-based models

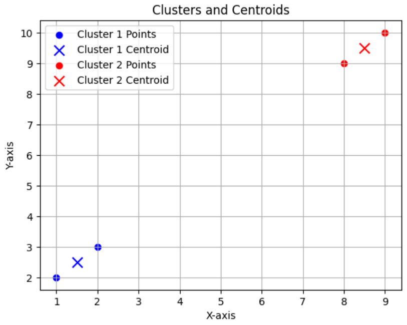
- VADER (Valence Aware Dictionary and sEntiment Reasoner)
- TextBlob

Supplementary materials

Example Calculation

Here's a small example to illustrate the calculation:
Suppose we have two clusters with the following points and centroids:

- **Cluster 1**:
  - Points: $[1, 2]$, $[2, 3]$
  - Centroid: $[1.5, 2.5]$
- **Cluster 2**:
  - Points: $[8, 9]$, $[9, 10]$
  - Centroid: $[8.5, 9.5]$



Example Calculation (Cont'd)

Calculation for Cluster 1

- Point $[1, 2]$:

$$\begin{aligned} \|[1,2] - [1.5, 2.5]\|^2 &= (1 - 1.5)^2 + (2 - 2.5)^2 \\ &= (-0.5)^2 + (-0.5)^2 \\ &= 0.25 + 0.25 \\ &= 0.5 \end{aligned}$$

- Point $[2, 3]$:

$$\begin{aligned} \|[2,3] - [1.5, 2.5]\|^2 &= (2 - 1.5)^2 + (3 - 2.5)^2 \\ &= (0.5)^2 + (0.5)^2 \\ &= 0.25 + 0.25 \\ &= 0.5 \end{aligned}$$

- **WCSS for Cluster 1**:

$$0.5 + 0.5 = 1.0$$

Example Calculation (Cont'd)

Calculation for Cluster 2

- Point $[8, 9]$:

$$\begin{aligned}\|[8,9]-[8.5,9.5]\|^2 &= (8-8.5)^2+(9-9.5)^2\\ &= (-0.5)^2+(-0.5)^2\\ &= 0.25+0.25\\ &= 0.5\end{aligned}$$

- Point $[9, 10]$:

$$\begin{aligned}\|[9,10]-[8.5,9.5]\|^2 &= (9-8.5)^2+(10-9.5)^2\\ &= (0.5)^2+(0.5)^2\\ &= 0.25+0.25\\ &= 0.5\end{aligned}$$

- **WCSS for Cluster 2**:

$$0.5+0.5=1.0$$

Cont'd

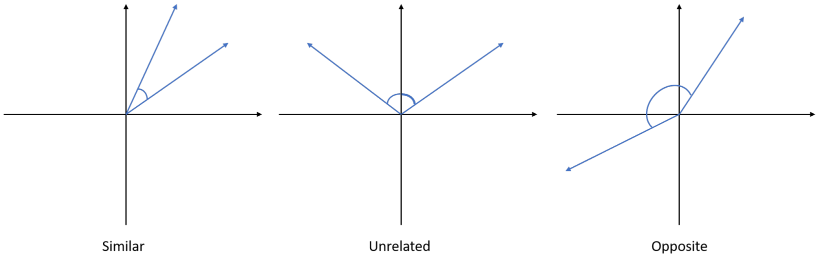
Total WCSS

$$\text{WCSS} = \text{WCSS for Cluster 1} + \text{WCSS for Cluster 2} = 1.0 + 1.0 = 2.0$$

By summing up the squared distances within each cluster, we get the total WCSS. This process helps in determining how well the clustering has grouped the data points.

Cosine Similarity

Cosine similarity measures the similarity between two vectors of an inner product space in general. It measures the cosine of the angle between two embeddings and determines whether they are pointing in roughly the same direction or not.



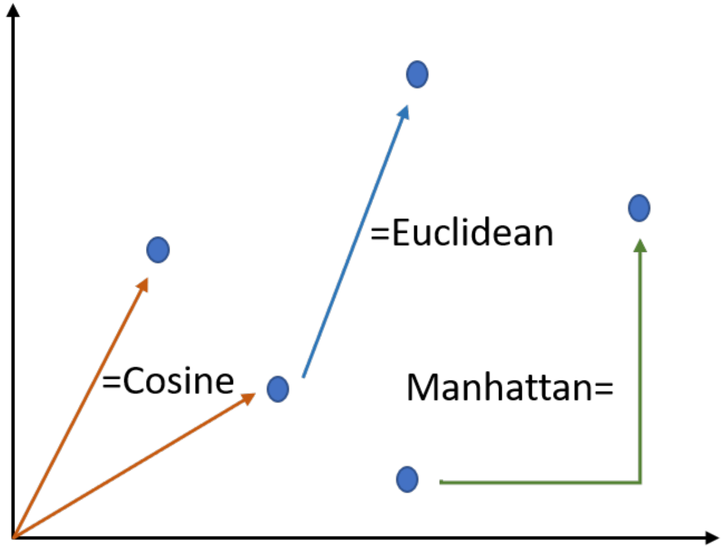
$$\text{Cosine Similarity} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|} \tag{9}$$

Jaccard Similarity

Jaccard Similarity is also known as the Jaccard index and Intersection over Union. It is the intersection of two sentences/texts between which the similarity is being calculated divided by the union of those two which refers to the number of common words over a total number of words. The Jaccard Similarity score ranges from 0 to 1, where 1 represents most and 0 represents least similar.

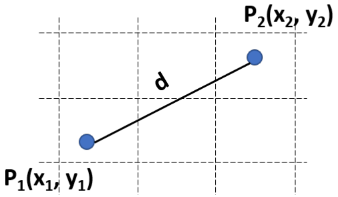
$$\text{Jaccard Similarity} = \frac{|A \cap B|}{|A \cup B|} \tag{10}$$

Different distance measures



Euclidean Distance

The Euclidean Distance formula is the most common formula to calculate distance between two points/coordinates. To get it we just need to subtract the points from the vectors, raise them to squares, add them up and take the square root of them. Similarly, it's used to calculate the distance between the embedding vectors of the words, supposing each vector represents a set of coordinates.



$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} \tag{11}$$

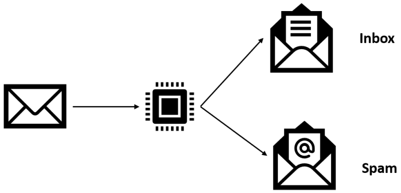
SVD

- Left Singular Vectors (U): In LSA, the left singular vectors (U matrix) capture the "concepts" or "latent topics" in the data. Each column of the U matrix represents a concept, and each row represents a term. By selecting the top k columns (where k is the desired number of dimensions or concepts), we can reduce the dimensionality of the data while preserving the most significant information.
- Singular Values (Σ): The singular values represent the importance or strength of each concept. They are typically sorted in decreasing order, so the first few singular values capture the most significant patterns in the data.
- Right Singular Vectors (V^T): In LSA, the right singular vectors (V^T matrix) represent the relationship between documents and concepts. Each row of V^T corresponds to a document, and each column corresponds to a concept. By selecting the top k rows (again, where k is the desired number of dimensions), we can obtain a low-dimensional representation of the documents.

Example

Email Spam Detection
You're tasked with building an email spam detection system to filter out unwanted spam emails from users' inboxes. You have a dataset of emails, each labeled as either spam or non-spam.

Your goal is to develop a model that can accurately classify incoming emails as spam or non-spam based on their content.



Example (Cont'd)

Table: Example Dataset

Email Content	**Class**
"buy cheap viagra"	Spam
"cheap viagra online"	Spam
"hi mom how are you"	Non-Spam
"let's have lunch"	Non-Spam
"cheap lunch deal"	Spam

Example (Cont'd)

1. Training: You train a Naive Bayes classifier using the labelled emails. For each class (spam and non-spam), you calculate the probabilities of each word occurring given that class (P(word|class)). This involves counting the occurrences of each word in emails of each class and normalizing them to obtain probabilities.

We calculate the prior probabilities of each class (spam and non-spam).
Total number of emails: 5
Number of spam emails: 3
Number of non-spam emails: 2

$$P(\text{Spam}) = \frac{\text{Number of spam emails}}{\text{Total number of emails}} = \frac{3}{5} = 0.6$$

$$P(\text{Non-Spam}) = \frac{\text{Number of non-spam emails}}{\text{Total number of emails}} = \frac{2}{5} = 0.4$$

Example (Cont'd)

Calculate Likelihood Probabilities (P(word|class)): We calculate the probabilities of each word given the class.

Let's first list all unique words in the dataset:
Vocabulary: [“buy”, “cheap”, “viagra”, “online”, “hi”, “mom”, “how”, “are”, “you”, “let's”, “have”, “lunch”, “deal”]

Word Counts by Class: Count the occurrences of each word in spam and non-spam emails.
Spam emails: [“buy”: 1, “cheap”: 3, “viagra”: 2, ‘online”: 1, “lunch”: 1, “deal”: 1]
Non-spam emails: [“hi”: 1, “mom”: 1, “how”: 1, “are”: 1, “you”: 1, “let's”: 1, “have”: 1, “lunch”: 1]

Example (Cont'd)

Calculating Probabilities:

Total words in spam:3+3+3 = 9 (including duplicates)
Total words in non-spam: 5+4 = 9 (including duplicates)
Size of vocabulary (V): 13

$$P(\text{word}|\text{Spam}) = \frac{\text{Count of word in Spam}+1}{\text{Total words in Spam}+\text{V}}$$

$$P(\text{word}|\text{Non-Spam}) = \frac{\text{Count of word in Non-Spam}+1}{\text{Total words in Non-Spam}+\text{V}}$$

Example (Cont'd)

Let's calculate some probabilities:
For "cheap" given Spam:

$$P(\text{cheap}|\text{Spam}) = \frac{3+1}{9+13} = \frac{4}{22} = 0.182$$

For "cheap" given Non-Spam:

$$P(\text{cheap}|\text{Non-Spam}) = \frac{0+1}{9+13} = \frac{1}{22} = 0.0455$$

(Continue this for all words in the vocabulary)

Example (Cont'd)

2. Classification: To classify a new, unseen email, you apply the Naive Bayes algorithm. You calculate the probability of the email belonging to each class (spam or non-spam) using Bayes' theorem. By multiplying the probabilities of each word in the email given each class and the prior probabilities of each class, you determine whether the email is spam or non-spam.

Let's classify a new email: "cheap lunch"
Calculate likelihoods for Spam:

$$P(\text{cheap}|\text{Spam}) \times P(\text{lunch}|\text{Spam})$$

Assuming the calculated probabilities:

$$P(\text{cheap}|\text{Spam}) = 0.182$$

$$P(\text{lunch}|\text{Spam}) = \frac{1+1}{9+13} = \frac{2}{22} = 0.091$$

$$P(\text{Email}|\text{Spam}) = 0.182 \times 0.091 = 0.0165$$

Kim, Junhong et al. (2019). “Insider threat detection based on user behavior modeling and anomaly detection algorithms”. In: *Applied Sciences* 9.19, p. 4018.

Medhat, Walaa et al. (2014). “Sentiment analysis algorithms and applications: A survey”. In: *Ain Shams engineering journal* 5.4, pp. 1093–1113.