Lecture 7

K-Means
Linear Discriminant Analysis (LDA)
Nearest Centroid Classifier

The Nearest Centroid Classifier (NCC) also known as the **Nearest Mean Classifier** or **Centroid-Based Classifier**, is a simple and interpretable classification algorithm used to assign data points to the class whose centroid (mean) is nearest to the data point in feature space. It is particularly suitable for multi-class classification problems and can work well when class distributions are approximately Gaussian or have similar shapes.

Training Phase: For each class in the training dataset, calculate the mean (centroid) of the feature vectors of the data points belonging to that class. These centroids represent the "average" data point for each class.

Classification Phase: Given a new data point (the one you want to classify), calculate its distance (e.g., Euclidean distance) to each class centroid. Assign the data point to the class whose centroid is closest, i.e., the class with the smallest distance.

Mathematically, during the classification phase, the algorithm calculates the distance between a data point x and each class centroid c_i and assigns the data point to the class with the minimum distance:

$$Class(x) = \underset{i}{\operatorname{argmin}} \|x - c_i\|$$

Where:
x is the data point you want to classify.
$c_i$is the centroid of class i.
$\|x - c_i\|$ represents the distance between x and c_i. This distance can be calculated using various distance metrics, such as the Euclidean distance.

Advantages of the Nearest Centroid Classifier:

- *Simplicity*: NCC is straightforward and easy to understand, making it a good choice for quick prototyping.
- *Interpretable*: The classifier's decision is based on the proximity to class centroids, providing transparency and interpretability.

Limitations of the Nearest Centroid Classifier:

- *Sensitivity to Outliers*: NCC can be sensitive to outliers because it uses the mean (centroid) of each class, and a single outlier can significantly affect the centroid.
- *Assumes Gaussian Distribution*: NCC works best when class distributions are approximately Gaussian or have similar shapes. It may not perform well with highly skewed data.
- *May not capture complex decision boundaries*: NCC assumes that class centroids are representative of the entire class, which may not be the case in situations where classes are complex or overlap.

The Nearest Centroid Classifier is often used in applications where interpretability and simplicity are essential, such as image compression, text classification, and feature selection. However, for more

complex classification tasks with non-linear decision boundaries, other algorithms like support vector machines (SVMs), decision trees, or deep learning models may be more suitable.

Linear Discriminant Analysis (LDA) is a statistical technique used for dimensionality reduction and classification. It's a supervised learning algorithm that aims to find the linear combinations of features that best separate two or more classes in a dataset. LDA is commonly used in the context of pattern recognition, machine learning, and statistical analysis.

Key Concepts of Linear Discriminant Analysis:
Objective: The primary objective of LDA is to maximize the separation between multiple classes while minimizing the variation within each class.

Assumption: LDA assumes that the features are normally distributed and that the classes have the same covariance matrix.

Linear Combinations: LDA finds linear combinations of the original features that create new axes, known as discriminant functions. These discriminant functions are chosen to maximize the distance between the means of different classes while minimizing the spread (variance) within each class.

Decision Rule: The decision rule in LDA involves comparing the posterior probabilities of a data point belonging to different classes. The class with the highest posterior probability is assigned to the data point.

Covariance Matrix: LDA calculates a pooled covariance matrix that represents the sum of the covariance matrices of individual classes, weighted by their sample sizes.

Steps in Linear Discriminant Analysis:
Compute the Within-Class Scatter Matrix (S_W):
Calculate the scatter matrix for each class and sum them to get the within-class scatter matrix.

$$S_W = \sum_{i=1}^{c} (X_i - M_i)(X_i - M_i)^T$$

where c is the number of classes, X_i is the matrix of data points for class i, and M_i is the mean vector for class i.

Compute the Between-Class Scatter Matrix (S_B):
Calculate the between-class scatter matrix.

$$S_B = \sum_{i=1}^{c} N_i (M_i - M)(M_i - M)^T$$

where N_i is the number of data points in class i, M_i is the mean vector for class i, M is the overall mean vector.

Compute the Eigenvalues and Eigenvectors:
Solve the generalized eigenvalue problem $S_W^{-1} S_B$ to obtain the eigenvalues and eigenvectors.

Select Discriminant Functions: The discriminant functions are chosen based on the eigenvalues and eigenvectors. The number of discriminant functions is at most c^{-1}, where c is the number of classes.

Project Data onto Discriminant Functions: Project the original data onto the selected discriminant functions to obtain a lower-dimensional representation.

Classification: Use the projected data for classification, often by applying a threshold or decision rule based on class centroids.

Pros and Cons of Linear Discriminant Analysis:

Pros:

- *Dimensionality Reduction*: LDA provides a way to reduce the dimensionality of the dataset while preserving class discriminatory information.
- *Feature Extraction*: It helps identify the most informative features for classification.
- *Regularization*: LDA can be regularized to handle cases where the covariance matrices are singular or poorly conditioned.

Cons:

- *Assumption of Normality*: LDA assumes that the features are normally distributed within each class.
- *Sensitive to Outliers*: LDA can be sensitive to outliers.
- *Binary Classification*: LDA is inherently designed for binary classification, but extensions exist for multiple classes.
- *Assumption of Equal Covariances*: LDA assumes that the classes have the same covariance matrix, which may not hold in all situations.

In summary, Linear Discriminant Analysis is a powerful technique for dimensionality reduction and classification, particularly when the goal is to separate multiple classes in a dataset. It is widely used in various fields, including pattern recognition and machine learning.

Data clustering techniques are essential in data mining for discovering meaningful patterns and structures in datasets. Clustering aims to group similar data points together while separating dissimilar ones. Here are some common data clustering techniques used in data mining:

K-Means Clustering:
Method: K-Means partitions data into k clusters based on distance from cluster centroids.
Advantages: It is computationally efficient and works well with large datasets.
Considerations: The number of clusters (k) must be specified in advance, and it is sensitive to initial centroid selection.

Hierarchical Clustering:
Method: Hierarchical clustering creates a tree-like structure of clusters, with data points or groups merging step by step.
Advantages: It provides a hierarchy of clusters, allowing different granularity levels for analysis.
Considerations: Hierarchical clustering can be computationally intensive, and the choice of linkage method (single, complete, average, etc.) impacts results.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
Method: DBSCAN groups data points into clusters based on their density and connectivity.
Advantages: It can discover clusters of arbitrary shapes and is robust to noise.

Considerations: DBSCAN requires setting parameters like the minimum number of points in a neighborhood.

Agglomerative Clustering:
Method: Agglomerative clustering starts with each data point as a single cluster and recursively merges the closest clusters.
Advantages: It is straightforward to implement and allows for a flexible number of clusters.
Considerations: Agglomerative clustering can be computationally demanding for large datasets.

Spectral Clustering:
Method: Spectral clustering transforms data into a lower-dimensional space and performs clustering in that space.
Advantages: It is effective for data with complex structures and works well when clusters have non-convex shapes.
Considerations: Spectral clustering involves eigenvalue decomposition and can be computationally expensive.

Fuzzy C-Means Clustering:
Method: Fuzzy C-Means assigns data points to clusters with membership degrees, allowing points to belong to multiple clusters to varying degrees.
Advantages: It accommodates data points that have ambiguous cluster assignments.
Considerations: It requires the tuning of a fuzziness parameter.

Mean Shift Clustering:
Method: Mean Shift identifies clusters by seeking modes in the density of data points.
Advantages: It is robust to initializations and can discover clusters of varying shapes and sizes.
Considerations: Parameter selection, especially bandwidth, can impact results.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies):
Method: BIRCH is a hierarchical clustering method optimized for large datasets and online clustering.
Advantages: It is memory-efficient and can handle streaming data.
Considerations: BIRCH is sensitive to the choice of parameters.

Self-Organizing Maps (SOM):
Method: SOM is a type of artificial neural network that maps high-dimensional data to a lower-dimensional grid while preserving topological relationships.
Advantages: It can visualize complex data structures and clusters.
Considerations: SOM requires the tuning of grid size and learning rate.

OPTICS (Ordering Points To Identify the Clustering Structure):
Method: OPTICS is an extension of DBSCAN that produces a reachability plot to reveal the cluster structure.
Advantages: It provides a more detailed view of clusters and density.
Considerations: OPTICS can be computationally intensive, and parameter tuning is required.

The choice of clustering technique depends on the characteristics of the data, the desired granularity of clusters, and the goals of the data mining task. It often involves experimenting with different methods and evaluating the quality of clustering results based on domain-specific criteria.

K-Means clustering is one of the most widely used and straightforward clustering techniques in data mining and machine learning. It's used to partition a dataset into groups or clusters based on the similarity of data points.

Basic Procedure:
Initialization: Choose the number of clusters (k) you want to create and initialize k cluster centroids randomly or using some predefined method.

Assignment Step: Assign each data point to the nearest cluster centroid based on a distance metric, typically Euclidean distance. Each data point belongs to the cluster with the closest centroid.

Update Step: Recalculate the cluster centroids as the mean (average) of all data points assigned to that cluster.

Iteration: Repeat the assignment and update steps until the clusters stabilize or until a convergence criterion is met. Common convergence criteria include a maximum number of iterations, minimal centroid movement, or a predefined error threshold.

Key Points:
Choosing the Number of Clusters (k): Determining the appropriate number of clusters is often a critical decision. Common methods for selecting k include the Elbow Method and the Silhouette Score. It's often necessary to experiment with different values of k to find the most suitable one for your data.

Initialization: The choice of initial cluster centroids can impact the final clustering results. Random initialization can lead to different results in each run. The K-Means++ initialization method is often preferred as it distributes the initial centroids more effectively.

Distance Metric: The choice of distance metric (usually Euclidean distance) can be modified to suit the data and problem, such as using Manhattan distance or other custom distance functions.

Sensitivity to Initializations: K-Means can converge to local optima, which means the results can vary with different initializations. Running the algorithm multiple times with different initializations and selecting the best result is a common practice.

Scalability: K-Means can be computationally expensive, especially for large datasets, as it requires calculating distances between data points and centroids. Techniques like Mini-Batch K-Means can be used to make it more scalable.

Cluster Shape: K-Means assumes that clusters are spherical, equally sized, and isotropic, which means it may not perform well when dealing with non-spherical or unevenly sized clusters.

Outliers: K-Means can be sensitive to outliers, as they can significantly affect cluster centroids.

Applications: K-Means clustering is used in various domains and applications, including:

- Customer segmentation in marketing.
- Image compression and color quantization.
- Document classification.
- Anomaly detection.

- Genomic data analysis.
- Natural language processing.
- Recommendation systems.
- Identifying species in biology.
- Image and video analysis.

K-Means is a fundamental and widely used clustering technique, but it may not be suitable for all types of data and clustering tasks, especially when dealing with complex cluster shapes or varying cluster sizes. In such cases, more advanced clustering methods like hierarchical clustering or density-based clustering (e.g., DBSCAN) may be more appropriate.

In this example, we'll use the built-in *iris* dataset to perform K-Means clustering. This dataset contains measurements of sepal length, sepal width, petal length, and petal width for 150 iris flowers, with three species: setosa, versicolor, and virginica. We'll aim to cluster the flowers into three groups based on these measurements.

Here's how to perform K-Means clustering in R:

```
# Load the iris dataset
data(iris)
# Select the relevant features (sepal length and sepal width)
iris_features <- iris[, c("Sepal.Length", "Sepal.Width")]
# Set the number of clusters (k)
k <- 3
# Perform K-Means clustering
kmeans_result <- kmeans(iris_features, centers = k)
# Display the cluster assignments
cluster_assignments <- kmeans_result$cluster
print(cluster_assignments)
# Display the cluster centers
cluster_centers <- kmeans_result$centers
print(cluster_centers)
# Visualize the data and cluster centers
library(ggplot2)
# Plot the data points
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color =
factor(cluster_assignments))) + geom_point() + geom_point(data =
as.data.frame(cluster_centers), aes(x = Sepal.Length, y =
Sepal.Width), color = "black", size = 4, shape = 3) + labs(title = "K-
Means Clustering of Iris Data", color = "Cluster") + theme_minimal()
```

This code does the following:

- Loads the iris dataset and selects the relevant features (sepal length and sepal width).
- Sets the number of clusters (k) to 3.
- Performs K-Means clustering using the kmeans function.
- Displays the cluster assignments, which indicate which data point belongs to which cluster.
- Displays the cluster centers, which represent the mean values of each cluster's data points.
- Visualizes the data points with different colors representing the assigned clusters and marks the cluster centers on the plot.

When you run this code, you'll see a plot with data points colored by cluster, and the cluster centers marked as black triangles. K-Means clustering has grouped the data points into three clusters based on their sepal length and sepal width measurements.

You can adjust the value of k to experiment with different numbers of clusters and observe how it affects the clustering results. K-Means is sometimes used as a semi-supervised classification model, with k chosen according to the number of classes that are needed. Identification of which clusters coincide with which of the initial classes will have to be determined manually.

Resources:

1. https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/
2. https://uc-r.github.io/kmeans_clustering
3. https://www.datacamp.com/tutorial/k-means-clustering-r
4. https://www.geeksforgeeks.org/k-means-clustering-in-r-programming/
5. https://www.guru99.com/r-k-means-clustering.html
6. https://www.statology.org/k-means-clustering-in-r/
7. https://towardsdatascience.com/k-means-clustering-in-r-feb4a4740aa
8. https://www.geeksforgeeks.org/linear-discriminant-analysis-in-r-programming/
9. https://www.r-bloggers.com/2021/05/linear-discriminant-analysis-in-r/
10. https://towardsdatascience.com/linear-discriminant-analysis-lda-101-using-r-6a97217a55a6
11. https://www.statology.org/linear-discriminant-analysis-in-r/
12. http://www.sthda.com/english/articles/36-classification-methods-essentials/146-discriminant-analysis-essentials-in-r/
13. https://uw.pressbooks.pub/appliedmultivariatestatistics/chapter/discriminant-analysis/
14. https://medium.com/edureka/linear-discriminant-analysis-88fa8ad59d0f
15. https://www.geeksforgeeks.org/ml-nearest-centroid-classifier/
16. https://cran.r-hub.io/web/packages/lolR/vignettes/nearestCentroid.html
17. https://idc9.github.io/stor390/notes/classification/classification.html

FYI, there is another LDA that comes up in data science contexts. We won't be learning this particular algorithm, but I've included some optional information about it here to help you distinguish the two.

LDA, which stands for *Latent Dirichlet Allocation*, is a popular probabilistic model used for topic modeling and document classification in natural language processing and text mining. LDA is a generative statistical model that helps discover underlying topics in a collection of documents and assigns topics to individual documents. Here's how LDA works:

Assumptions: LDA assumes that documents are mixtures of topics, and topics are mixtures of words. In other words, it assumes that there is a hidden (latent) structure of topics that generates the observed text data.

Initialization: LDA begins with a set of documents and a predefined number of topics (a hyperparameter). Each document is represented as a bag of words, where the order of words does not matter.

Random Assignment: Initially, LDA randomly assigns words in each document to one of the topics. This is called the "initialization phase."

Iterations: LDA iterates through the following steps until it converges to a stable state:

1. For Each Word: LDA goes through each word in each document and reassigns it to a topic based on a probability distribution.
2. For Each Topic: LDA updates the topic distribution for each document based on the words currently assigned to the topic.

Mathematics: LDA uses Bayesian statistics to compute the probability of a word belonging to a topic and the probability of a document containing a mixture of topics. It uses the Dirichlet distribution to model these probabilities.

Convergence: The iterations continue until the model converges or reaches a predetermined number of iterations.

Output: Once the model has converged, LDA provides two key outputs:

1. Topic-Word Distribution: This shows the probability of each word in the vocabulary belonging to each topic.
2. Document-Topic Distribution: This shows the probability of each document containing a mixture of topics.

Interpretation: After running LDA, you can interpret the results by examining the most probable words for each topic. This allows you to assign human-readable labels to the discovered topics.

Key Points:

- LDA is a generative model that uncovers the underlying structure of topics in a collection of documents.
- It is based on the idea that each document is a mixture of topics, and each topic is a mixture of words.
- LDA uses probability distributions to iteratively update the assignment of words to topics and the distribution of topics in documents.
- LDA is commonly used for document clustering, topic modeling, and content recommendation.
- LDA is a valuable tool for understanding and organizing large text datasets. It has applications in information retrieval, recommendation systems, and content analysis, among others.