The chapter focuses on vector space classification for text documents, specifically using methods like the **Rocchio classifier** and **k-nearest neighbors (kNN)** in a high-dimensional vector space. Here's the detailed theory and concepts covered in the first part, along with some numerical examples for clarity.

Part 1: Vector Space Model and Classification Hypothesis

1. Introduction to Vector Space Model

In traditional text classification, documents are represented as binary vectors or sequences of terms. However, in the **vector space model (VSM)**, each document is treated as a point in a high-dimensional space. Here:

- Each **dimension** corresponds to a unique term in the document.
- **Document vectors** are weighted based on term frequency-inverse document frequency (TF-IDF), enhancing term importance based on their frequency within a document and rarity across all documents.

This representation leads to a high-dimensional space $\mathbb{R}^{|V|}$, where $|V|$ is the vocabulary size.

2. Contiguity Hypothesis

The **contiguity hypothesis** underpins vector space classification:

- Documents belonging to the same class are grouped in contiguous regions of space, forming distinct clusters.
- Different classes form non-overlapping clusters, making boundaries that separate classes feasible.

Example: Suppose we have two document classes, "China" and "UK." Terms like "Beijing" and "Chinese" are frequent in the "China" documents, while "London" and "British" are common in "UK" documents. The VSM will place "China" documents closer in space than "UK" documents, allowing them to be classified based on these clustered regions.

Document Representation and Similarity Measures

1. Document Representation

For vector space classification, documents are converted into **normalized vectors**. This normalization, especially with **length-normalized TF-IDF weights**, mitigates the effect of document length, ensuring the distance between vectors is meaningful. The normalized vectors are often projected onto the surface of a hypersphere.

Example of TF-IDF Calculation:
For a document containing terms “Beijing” and “Shanghai” with term frequencies (TF) of 3 and 5, respectively, in a collection where these terms appear in 20% and 10% of documents, the TF-IDF scores for each term could be calculated as:

$$\text{TF-IDF(Beijing)} = 3 \times \log\left(\frac{1}{0.2}\right) = 3 \times 0.698 = 2.094$$

$$\text{TF-IDF(Shanghai)} = 5 \times \log\left(\frac{1}{0.1}\right) = 5 \times 1 = 5$$

Thus, the document vector would have these weighted values, which will be used in calculating distances and similarities with other documents.

2. Measuring Relatedness: Euclidean Distance and Cosine Similarity

Classification in VSM relies on **measuring the distance** or **similarity** between document vectors:

- **Cosine similarity** compares the angle between vectors, ideal for normalized vectors.
- **Euclidean distance** measures straight-line distance between points in space, useful for non-normalized data.

Example: Suppose we have two vectors representing documents: $\mathbf{d_1} = (2,3)$ and $\mathbf{d_2} = (4,1)$. The Euclidean distance between $\mathbf{d_1}$ and $\mathbf{d_2}$ would be:

$$d(\mathbf{d_1}, \mathbf{d_2}) = \sqrt{(4-2)^2 + (1-3)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$$

Alternatively, using cosine similarity:

$$\cos(\theta) = \frac{\mathbf{d_1} \cdot \mathbf{d_2}}{|\mathbf{d_1}||\mathbf{d_2}|} = \frac{(2 \times 4) + (3 \times 1)}{\sqrt{2^2+3^2} \times \sqrt{4^2+1^2}} = \frac{8+3}{\sqrt{13} \times \sqrt{17}} \approx 0.803$$

Summary for Part 1

The vector space model (VSM) for text classification leverages TF-IDF to represent documents in high-dimensional spaces, where classification is guided by hypotheses like contiguity, separating classes based on spatial clusters. Cosine similarity and Euclidean distance offer complementary ways to measure document proximity, forming the basis for classification algorithms like **Rocchio** and **kNN**, which will be explored in the following parts.

Let me know when you’d like to continue with Part 2 on **Rocchio classification**, where we delve deeper into using centroids to define class boundaries.

Part 2: Rocchio Classification

The **Rocchio classification method** is a centroid-based classification algorithm that uses vector space principles to determine class boundaries by averaging document vectors for each class. It operates on the concept of **centroids** or **prototypes**, representing the "center of mass" for documents in each class. Here's an in-depth look at how Rocchio classification works and an example calculation to clarify the concepts.

1. Centroid Calculation

To classify documents, Rocchio calculates a **centroid vector** for each class by taking the mean of all document vectors within that class. The centroid $\mu(c)$ for a class c is defined as:

$$\mu(c) = \frac{1}{|D_c|} \sum_{d \in D_c} v(d)$$

where:

- D_c is the set of all documents in class c,
- $v(d)$ is the vector representation of a document d in class c,
- $|D_c|$ is the number of documents in class c.

Each document vector in the class contributes to the centroid, making $\mu(c)$ an "average" representation of the class.

Rocchio Classification Example

To classify the test document "Chinese Chinese Chinese Tokyo Japan" using the **Rocchio classification** method, we need to calculate the **centroid** of each class ("China" and "Japan") based on the training dataset. Each class centroid is an average vector that represents the documents in that class.

Step 1: Construct the Vector Space Model

Let's first identify the vocabulary (all unique words) in the training dataset:

- Unique words: **Chinese, Beijing, Shanghai, Macao, Tokyo, Japan**

We'll assign each word an index in a vector space model. Each document can be represented as a vector where each position corresponds to the frequency of a particular word in that document.

Document ID	Words in Document	Class	Vector
1	Chinese Beijing Chinese	China	[2, 1, 0, 0, 0, 0]
2	Chinese Chinese Shanghai	China	[2, 0, 1, 0, 0, 0]
3	Chinese Macao	China	[1, 0, 0, 1, 0, 0]
4	Tokyo Japan Chinese	Japan	[1, 0, 0, 0, 1, 1]

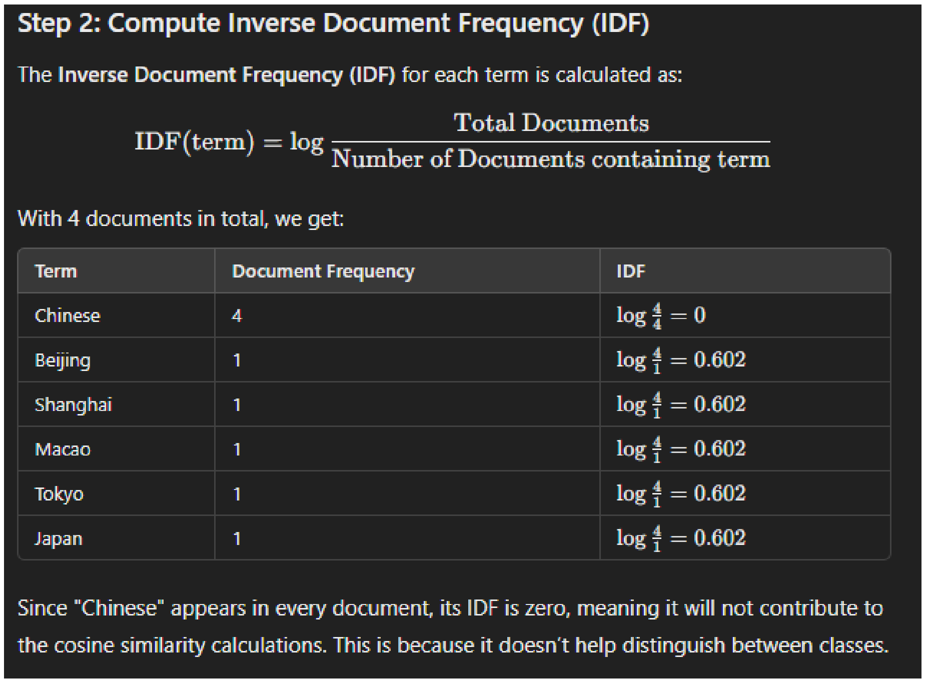
Table 1: Document Vectors

Step 2: Create Document Vectors

Using the identified words and their counts in each document, the vectors for each document become:

The word order in the vector is [**Chinese, Beijing, Shanghai, Macao, Tokyo, Japan**].

This is an additional step where we make use of idf-df weight table calculating the centroid for both the classes



Step 2: Compute Inverse Document Frequency (IDF)

The **Inverse Document Frequency (IDF)** for each term is calculated as:

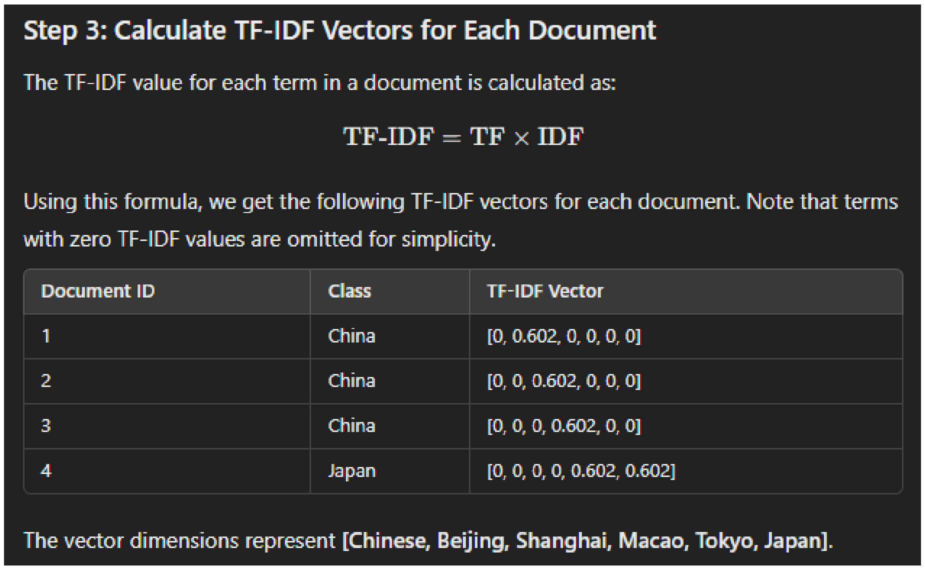
$$\text{IDF(term)} = \log \frac{\text{Total Documents}}{\text{Number of Documents containing term}}$$

With 4 documents in total, we get:

Term	Document Frequency	IDF
Chinese	4	$\log \frac{4}{4} = 0$
Beijing	1	$\log \frac{4}{1} = 0.602$
Shanghai	1	$\log \frac{4}{1} = 0.602$
Macao	1	$\log \frac{4}{1} = 0.602$
Tokyo	1	$\log \frac{4}{1} = 0.602$
Japan	1	$\log \frac{4}{1} = 0.602$

Since "Chinese" appears in every document, its IDF is zero, meaning it will not contribute to the cosine similarity calculations. This is because it doesn't help distinguish between classes.

Figure 1: This is an example image.



Step 3: Calculate TF-IDF Vectors for Each Document

The TF-IDF value for each term in a document is calculated as:

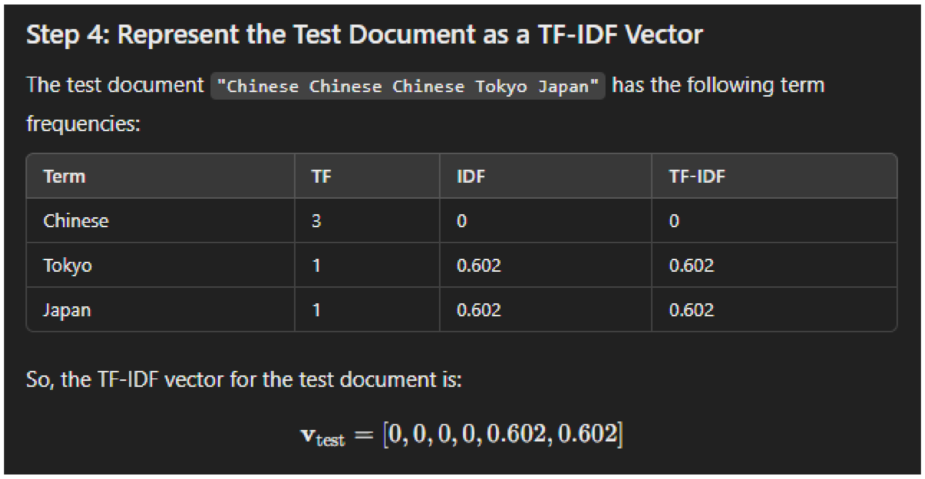
$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

Using this formula, we get the following TF-IDF vectors for each document. Note that terms with zero TF-IDF values are omitted for simplicity.

Document ID	Class	TF-IDF Vector
1	China	[0, 0.602, 0, 0, 0, 0]
2	China	[0, 0, 0.602, 0, 0, 0]
3	China	[0, 0, 0, 0.602, 0, 0]
4	Japan	[0, 0, 0, 0, 0.602, 0.602]

The vector dimensions represent **[Chinese, Beijing, Shanghai, Macao, Tokyo, Japan]**.

Figure 2: This is an example image.



Step 4: Represent the Test Document as a TF-IDF Vector

The test document `"Chinese Chinese Chinese Tokyo Japan"` has the following term frequencies:

Term	TF	IDF	TF-IDF
Chinese	3	0	0
Tokyo	1	0.602	0.602
Japan	1	0.602	0.602

So, the TF-IDF vector for the test document is:

$$\mathbf{v}_{\text{test}} = [0, 0, 0, 0, 0.602, 0.602]$$

Figure 3: This is an example image.

Step 3: Calculate the Centroid for Each Class

To find the centroid vector for each class, we calculate the average vector for the documents in each class.

1. **Centroid of Class "China"**:

$$\text{Sum of vectors for documents in Class "China" :}$$
$$[2,1,0,0,0,0] + [2,0,1,0,0,0] + [1,0,0,1,0,0] = [5,1,1,1,0,0]$$

- Number of documents in Class "China": 3 - Centroid vector for Class "China":

$$\mu(\text{China}) = \left[\frac{5}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0\right] \approx [1.67, 0.33, 0.33, 0.33, 0, 0]$$

2. **Centroid of Class "Japan"**:

$$\text{Sum of vectors for documents in Class "Japan" :}$$
$$[1,0,0,0,1,1]$$

- Number of documents in Class "Japan": 1 - Centroid vector for Class "Japan":

$$\mu(\text{Japan}) = [1,0,0,0,1,1]$$

Step 4: Represent the Test Document as a Vector

The test document "Chinese Chinese Chinese Tokyo Japan" can be represented as a vector based on the same word order **[Chinese, Beijing, Shanghai, Macao, Tokyo, Japan]**.

- Word counts in the test document: - Chinese: 3 - Beijing: 0 - Shanghai: 0 - Macao: 0 - Tokyo: 1 - Japan: 1

So, the vector representation of the test document is:

$$\mathbf{v}_{\text{test}} = [3,0,0,0,1,1]$$

Step 5: Calculate the Distance to Each Centroid

Using **Euclidean distance**, we calculate the distance between the test document vector and each class centroid.

1. **Distance to Centroid of Class "China"**:

$$d(\mathbf{v}_{\text{test}}, \mu(\text{China})) = \sqrt{(3-1.67)^2 + (0-0.33)^2 + (0-0.33)^2 + (0-0.33)^2 + (1-0)^2 + (1-0)^2}$$

$$= \sqrt{(1.33)^2 + (-0.33)^2 + (-0.33)^2 + (-0.33)^2 + 1^2 + 1^2}$$

$$= \sqrt{1.77 + 0.11 + 0.11 + 0.11 + 1 + 1} = \sqrt{4.1} \approx 2.02$$

2. **Distance to Centroid of Class "Japan"**:

$$d(\mathbf{v}_{\text{test}}, \mu(\text{Japan})) = \sqrt{(3-1)^2 + (0-0)^2 + (0-0)^2 + (0-0)^2 + (1-1)^2 + (1-1)^2}$$

$$= \sqrt{(2)^2 + 0 + 0 + 0 + 0 + 0} = \sqrt{4} = 2$$

Step 6: Classify the Test Document

Since the distance to **Class "Japan"** centroid (2.0) is slightly smaller than the distance to **Class "China"** centroid (2.02), the test document "Chinese Chinese Chinese Tokyo Japan" is classified as belonging to the **Japan** class.

Explanation

- The Rocchio classifier assigns the test document to the class whose centroid is closest to it. - Although the test document has more occurrences of "Chinese," the presence of "Tokyo" and "Japan" shifts the classification toward the "Japan" class centroid.

This example demonstrates how Rocchio classification can handle cases with overlapping terms by focusing on average positions of classes rather than individual words alone, making it useful for text classification problems where documents contain common terms across classes.

Example: Numerical Problem for Rocchio Classification

Let's walk through a numerical example using the **Rocchio classification** method to better understand how the **centroid vector** for each class is calculated.

Problem Statement

Suppose we have two classes of documents, **Class A** and **Class B**. Each document is represented as a vector in a 2-dimensional space. Here are the vector representations for each document in the classes:

- **Class A**:
  - Document 1: $\mathbf{v}(d_1) = [2, 3]$
  - Document 2: $\mathbf{v}(d_2) = [3, 5]$
  - Document 3: $\mathbf{v}(d_3) = [4, 4]$
- **Class B**:
  - Document 1: $\mathbf{v}(d_1) = [7, 8]$
  - Document 2: $\mathbf{v}(d_2) = [8, 7]$
  - Document 3: $\mathbf{v}(d_3) = [9, 9]$

Our goal is to calculate the **centroid vector** for each class. These centroids can then be used for classifying new document vectors.

Step 1: Calculate the Centroid for Class A

The centroid $\mu(A)$ for Class A is the mean of all document vectors in that class. According to the formula:

$$\mu(A) = \frac{1}{|D_A|} \sum_{d \in D_A} \mathbf{v}(d)$$

where:

- D_A is the set of all documents in Class A,
- $|D_A|$ is the number of documents in Class A (which is 3 in this case),
- $\mathbf{v}(d)$ is the vector representation of each document in Class A.

Calculation:
1. Sum the document vectors in Class A:

$$[2,3] + [3,5] + [4,4] = [2+3+4, 3+5+4] = [9,12]$$

2. Divide by the number of documents in Class A (3):

$$\mu(A) = \frac{1}{3}[9,12] = \left[\frac{9}{3}, \frac{12}{3}\right] = [3,4]$$

Thus, the centroid for **Class A** is $\mu(A) = [3,4]$.

Step 2: Calculate the Centroid for Class B

Similarly, the centroid $\mu(B)$ for Class B is the mean of all document vectors in that class:

$$\mu(B) = \frac{1}{|D_B|} \sum_{d \in D_B} \mathbf{v}(d)$$

where:

- D_B is the set of all documents in Class B,
- $|D_B|$ is the number of documents in Class B (which is also 3).

Calculation:
1. Sum the document vectors in Class B:

$$[7,8] + [8,7] + [9,9] = [7+8+9, 8+7+9] = [24,24]$$

2. Divide by the number of documents in Class B (3):

$$\mu(B) = \frac{1}{3}[24,24] = \left[\frac{24}{3}, \frac{24}{3}\right] = [8,8]$$

Thus, the centroid for **Class B** is $\mu(B) = [8,8]$.

Step 3: Classify a New Document

Suppose we have a new document with vector representation $\mathbf{v}_{\text{new}} = [5, 5]$, and we want to classify it as either belonging to Class A or Class B. We do this by calculating the **Euclidean distance** from $\mathbf{v}_{\text{new}}$ to each centroid.

1. **Distance to Centroid of Class A**:

$$d(\mathbf{v}_{\text{new}}, \mu(A)) = \sqrt{(5-3)^2 + (5-4)^2} = \sqrt{2^2 + 1^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$

2. **Distance to Centroid of Class B**:

$$d(\mathbf{v}_{\text{new}}, \mu(B)) = \sqrt{(5-8)^2 + (5-8)^2} = \sqrt{(-3)^2 + (-3)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$$

Since the distance to the centroid of **Class A** (2.24) is less than the distance to the centroid of **Class B** (4.24), we classify the new document as belonging to **Class A**.

Summary of Steps

1. **Calculate the centroid for each class** by averaging the document vectors. 2. **Find the Euclidean distance** between the new document vector and each class centroid. 3. **Classify the new document** into the class with the closest centroid.

This example illustrates the use of Rocchio classification for document categorization by computing centroids and using distance-based classification.

2. Classification Using Decision Boundaries

After centroids are calculated, the **decision boundaries** between classes are established by finding the points that are equidistant from the centroids of two classes. For a new, unclassified document, the Rocchio classifier:

- Assigns the document to the class with the closest centroid.
- Calculates **Euclidean distance** between the document vector and each class centroid.

Mathematical Representation of Decision Boundaries:
The boundary between two classes c_1 and c_2 is the **hyperplane** equidistant to their centroids:

$$\mathbf{w}^T \mathbf{x} = b$$

where:

- $\mathbf{w} = \mu(c_1) - \mu(c_2)$ is the normal vector to the hyperplane,
- $b = \frac{1}{2}(|\mu(c_1)|^2 - |\mu(c_2)|^2)$ is a constant representing the midpoint between the centroids.

Example of Centroid and Boundary Calculation

Consider a classification problem with two classes, A and B, and the following document vectors:

- Class A: $d_1 = (2, 3)$, $d_2 = (3, 4)$
- Class B: $d_3 = (6, 7)$, $d_4 = (7, 8)$

Step 1: Calculate Centroids

$$\mu(A) = \frac{1}{2}((2,3) + (3,4)) = \left(\frac{5}{2}, \frac{7}{2}\right) = (2.5, 3.5)$$

$$\mu(B) = \frac{1}{2}((6,7) + (7,8)) = \left(\frac{13}{2}, \frac{15}{2}\right) = (6.5, 7.5)$$

Step 2: Decision Boundary Equation
For this example, the boundary's normal vector $\mathbf{w}$ between A and B is:

$$\mathbf{w} = \mu(B) - \mu(A) = (6.5 - 2.5, 7.5 - 3.5) = (4, 4)$$

The constant b is:

$$b = \frac{1}{2}(|\mu(B)|^2 - |\mu(A)|^2)$$

Calculating the squared magnitudes:

$$|\mu(B)|^2 = 6.5^2 + 7.5^2 = 42.25 + 56.25 = 98.5$$

$$|\mu(A)|^2 = 2.5^2 + 3.5^2 = 6.25 + 12.25 = 18.5$$

Then:

$$b = \frac{1}{2}(98.5 - 18.5) = \frac{1}{2}(80) = 40$$

So, the equation of the boundary hyperplane is:

$$4x_1 + 4x_2 = 40 \quad \Rightarrow \quad x_1 + x_2 = 10$$

A document located at $(5, 5)$, for example, would fall on this boundary, whereas one located at $(4, 5)$ would be closer to class A.

Limitations of Rocchio Classification

Rocchio classification works best when:

1. **Classes are roughly spherical**: Rocchio assumes each class occupies a contiguous, compact region, close to spherical. Non-spherical distributions might lead to misclassification.
2. **Classes have similar spread**: If class distributions vary greatly in spread, Rocchio's centroid-based boundaries may be inaccurate.
3. **No multimodal distributions**: Classes that have multiple, disconnected regions (multimodal distributions) may not be well represented by a single centroid.

Summary for Part 2

Rocchio classification provides a **simple, computationally efficient** approach to classify documents by defining regions around centroids. By calculating the mean document vector for each class and assigning new documents based on proximity to these centroids, Rocchio effectively handles linearly separable data with roughly spherical class clusters.

Next up, we'll explore **k-Nearest Neighbors (kNN)**, a local, distance-based method that adapts to non-spherical and multimodal distributions. Let me know when you'd like to proceed.

Part 3: k-Nearest Neighbor (kNN) Classification

The **k-Nearest Neighbor (kNN)** classification method classifies a new document by examining its **k closest neighbors** in the vector space and assigning it the majority class among those neighbors. Unlike Rocchio, which uses a global centroid to represent each class, kNN is a **local, instance-based** approach, making it well-suited for complex, non-spherical, or multimodal class distributions.

1. Overview of kNN Classification

The **kNN algorithm** follows a straightforward approach:

1. **Distance Calculation**: Compute the distance between the test document and each document in the training set.
2. **Neighbor Selection**: Identify the k documents closest to the test document.
3. **Majority Voting**: Assign the test document to the class with the majority of representatives among the k nearest neighbors.

The choice of **k** (number of neighbors) significantly impacts the classifier:

- A small k (e.g., $k = 1$) makes the classifier sensitive to noise, as a single, possibly mislabeled neighbor could influence classification.
- A larger k smoothens decision boundaries, making the classifier more robust but potentially less sensitive to fine distinctions.

Example: For $k = 3$, if two of the three nearest neighbors belong to class "A" and one to class "B," the test document would be classified as "A."

2. Distance Measures in kNN

kNN typically uses **Euclidean distance** or **Cosine similarity** for calculating the distance or similarity between document vectors:

- **Euclidean Distance**: Measures the straight-line distance in the vector space.

$$d(\mathbf{d_1}, \mathbf{d_2}) = \sqrt{\sum_{i=1}^{n} (d_{1i} - d_{2i})^2}$$

- **Cosine Similarity**: Measures the angle between two vectors, useful for high-dimensional and sparse data.

$$\text{cosine}(\theta) = \frac{\mathbf{d_1} \cdot \mathbf{d_2}}{|\mathbf{d_1}||\mathbf{d_2}|} = \frac{\sum_{i=1}^{n} d_{1i} \cdot d_{2i}}{\sqrt{\sum_{i=1}^{n} d_{1i}^2} \cdot \sqrt{\sum_{i=1}^{n} d_{2i}^2}}$$

Example Calculation: Suppose we have a test document vector $(3, 4)$ and two training document vectors, $(5, 6)$ and $(2, 1)$.

- **Euclidean Distance** between test document $(3, 4)$ and $(5, 6)$:

$$d = \sqrt{(3-5)^2 + (4-6)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$$

- **Cosine Similarity** between test document $(3, 4)$ and $(5, 6)$:

$$\text{cosine}(\theta) = \frac{(3 \cdot 5) + (4 \cdot 6)}{\sqrt{3^2 + 4^2} \cdot \sqrt{5^2 + 6^2}} = \frac{15 + 24}{5 \cdot 7.81} \approx 0.99$$

Example: Solving the Same Problem Using k-Nearest Neighbors (kNN) Classifier

In this example, we'll use the **k-Nearest Neighbors (kNN)** classifier to classify the same new document as in the Rocchio classification example and compare the results. We will set $k = 3$, meaning that we'll consider the **3 nearest neighbors** of the new document to decide its class.

Problem Setup (Revisited)

The document vectors for each class are as follows:

- **Class A**:
  - Document 1: $\mathbf{v}(d_1) = [2, 3]$
  - Document 2: $\mathbf{v}(d_2) = [3, 5]$
  - Document 3: $\mathbf{v}(d_3) = [4, 4]$
- **Class B**:
  - Document 1: $\mathbf{v}(d_1) = [7, 8]$
  - Document 2: $\mathbf{v}(d_2) = [8, 7]$
  - Document 3: $\mathbf{v}(d_3) = [9, 9]$

We need to classify a **new document** with vector $\mathbf{v}_{\text{new}} = [5, 5]$ using kNN.
—

Step 1: Calculate the Distance to Each Document

We'll calculate the **Euclidean distance** from $\mathbf{v}_{\text{new}} = [5, 5]$ to each document in both classes.

1. Distance to Documents in Class A:

- Distance to $[2, 3]$:

$$\sqrt{(5-2)^2 + (5-3)^2} = \sqrt{3^2 + 2^2} = \sqrt{9+4} = \sqrt{13} \approx 3.61$$

- Distance to $[3, 5]$:

$$\sqrt{(5-3)^2 + (5-5)^2} = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

- Distance to $[4, 4]$:

$$\sqrt{(5-4)^2 + (5-4)^2} = \sqrt{1^2 + 1^2} = \sqrt{2} \approx 1.41$$

2. Distance to Documents in Class B:

- Distance to $[7, 8]$:

$$\sqrt{(5-7)^2 + (5-8)^2} = \sqrt{(-2)^2 + (-3)^2} = \sqrt{4+9} = \sqrt{13} \approx 3.61$$

- Distance to $[8, 7]$:

$$\sqrt{(5-8)^2 + (5-7)^2} = \sqrt{(-3)^2 + (-2)^2} = \sqrt{9+4} = \sqrt{13} \approx 3.61$$

- Distance to $[9, 9]$:

$$\sqrt{(5-9)^2 + (5-9)^2} = \sqrt{(-4)^2 + (-4)^2} = \sqrt{16+16} = \sqrt{32} \approx 5.66$$

—

Step 2: Identify the 3 Nearest Neighbors

The 3 closest documents to $\mathbf{v}_{\text{new}} = [5, 5]$ are:

1. Document at $[4, 4]$ from **Class A** with distance **1.41**
2. Document at $[3, 5]$ from **Class A** with distance **2.00**
3. Document at $[2, 3]$ from **Class A** with distance **3.61**

All three of the nearest neighbors belong to **Class A**.
—

Step 3: Classify the New Document

Since the majority of the 3 nearest neighbors (all 3 in this case) are from **Class A**, we classify the new document as belonging to **Class A**.

0.1 K-Nearest Neighbors (KNN) Classifier

To solve this question using the **K-Nearest Neighbors (KNN) classifier**, we need to find the **k nearest neighbors** of the test document among the training documents based on their distances. Here, let's assume $k = 3$ for the classification.

Given:

1. Training dataset:
   - Document 1: "Chinese Beijing Chinese" (Class: **China**)
   - Document 2: "Chinese Chinese Shanghai" (Class: **China**)
   - Document 3: "Chinese Macao" (Class: **China**)
   - Document 4: "Tokyo Japan Chinese" (Class: **Japan**)
2. Test document: "Chinese Chinese Chinese Tokyo Japan"

The vocabulary is the same as in the previous example: [**Chinese, Beijing, Shanghai, Macao, Tokyo, Japan**].

0.1.1 Step 1: Represent Documents as Vectors

We already have each document represented as a vector, and we'll use the same vectors from the previous example.

Document ID	Words in Document	Class	Vector
1	Chinese Beijing Chinese	China	[2, 1, 0, 0, 0, 0]
2	Chinese Chinese Shanghai	China	[2, 0, 1, 0, 0, 0]
3	Chinese Macao	China	[1, 0, 0, 1, 0, 0]
4	Tokyo Japan Chinese	Japan	[1, 0, 0, 0, 1, 1]

Table 2: Document vectors for KNN classification.

The test document vector is:

$$\mathbf{v}_{\text{test}} = [3, 0, 0, 0, 1, 1]$$

0.1.2 Step 2: Calculate the Distance Between Test Document and Each Training Document

Using **Euclidean distance**, we calculate the distance between the test document and each training document.

1. **Distance to Document 1** ("Chinese Beijing Chinese"):

$$d(\mathbf{v}_{\text{test}}, \mathbf{v}_1) = \sqrt{(3-2)^2 + (0-1)^2 + (0-0)^2 + (0-0)^2 + (1-0)^2 + (1-0)^2}$$
$$= \sqrt{(1)^2 + (-1)^2 + 0 + 0 + 1 + 1} = \sqrt{4} = 2$$

2. **Distance to Document 2** ("Chinese Chinese Shanghai"):

$$d(\mathbf{v}_{\text{test}}, \mathbf{v}_2) = \sqrt{(3-2)^2 + (0-0)^2 + (0-1)^2 + (0-0)^2 + (1-0)^2 + (1-0)^2}$$
$$= \sqrt{(1)^2 + 0 + (-1)^2 + 0 + 1 + 1} = \sqrt{4} = 2$$

3. **Distance to Document 3** ("Chinese Macao"):

$$d(\mathbf{v}_{\text{test}}, \mathbf{v}_3) = \sqrt{(3-1)^2 + (0-0)^2 + (0-0)^2 + (0-1)^2 + (1-0)^2 + (1-0)^2}$$
$$= \sqrt{(2)^2 + 0 + 0 + (-1)^2 + 1 + 1} = \sqrt{7} \approx 2.65$$

4. **Distance to Document 4** ("Tokyo Japan Chinese"):

$$d(\mathbf{v}_{\text{test}}, \mathbf{v}_4) = \sqrt{(3-1)^2 + (0-0)^2 + (0-0)^2 + (0-0)^2 + (1-1)^2 + (1-1)^2}$$
$$= \sqrt{(2)^2} = \sqrt{4} = 2$$

0.1.3 Step 3: Identify the k Nearest Neighbors

Since $k = 3$, we select the three documents with the smallest distances to the test document. Here are the distances in ascending order:

Document ID	Distance	Class
1	2	China
2	2	China
4	2	Japan

Table 3: Distances to the test document.

The three nearest neighbors are:

- Document 1 (Class: **China**)
- Document 2 (Class: **China**)
- Document 4 (Class: **Japan**)

0.1.4 Step 4: Determine the Class Based on Majority Voting

Among the three nearest neighbors, two are from the **China** class and one is from the **Japan** class. Based on majority voting, the test document is classified as belonging to the **China** class.

0.1.5 Summary and Difference from Rocchio Classifier

- **KNN Result**: The test document is classified as **China**.
- **Rocchio Result**: The test document was classified as **Japan** in the Rocchio classification.

Key Differences between Rocchio and KNN Classifiers:

1. **Rocchio Classifier**:
   - Uses centroids (mean vectors) to represent each class and classifies based on the distance from the test document to each centroid.
   - Better suited for cases where documents in a class are close together around a central point.
   - Sensitive to the average characteristics of each class.
2. **KNN Classifier**:
   - Considers the k nearest neighbors individually and uses majority voting for classification.
   - Works well for datasets with irregular or spread-out clusters, where centroids may not represent each class accurately.
   - More adaptable to varying local distributions within each class, as it looks at neighbors rather than a single centroid.

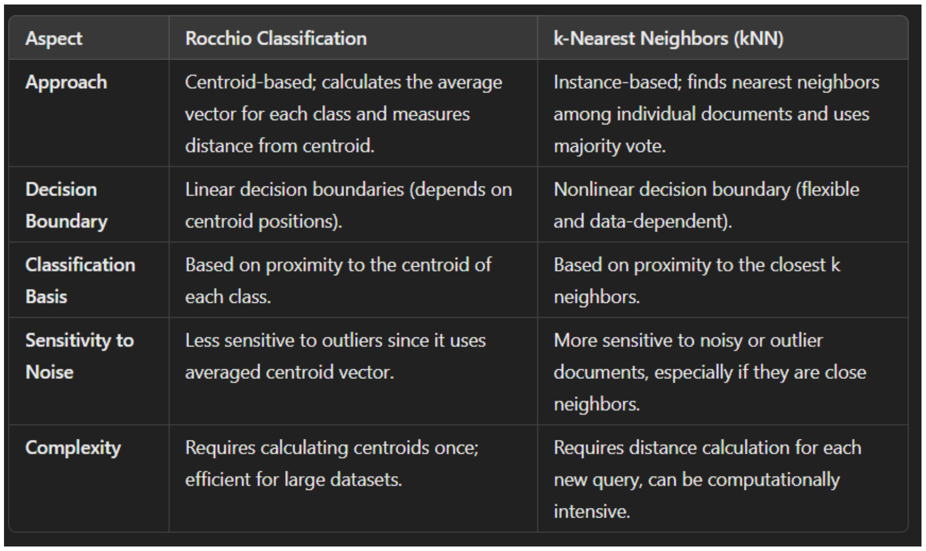
In this example, **Rocchio** considered the overall average for each class, while **KNN** focused on the specific instances nearest to the test document, leading to different classifications.

Comparison: Rocchio Classification vs. k-Nearest Neighbors (kNN)

In this example, both Rocchio and kNN classified the new document as belonging to **Class A**, but the methods differed in approach:

- **Rocchio Classification** relies on a **single representative point (centroid)** per class, making it efficient and less sensitive to individual document variations.
- **kNN Classification** considers individual document instances, which can lead to better results in cases with irregular class boundaries but may also increase sensitivity to outliers or noisy data points.

Both methods are useful in different contexts, with **Rocchio** being more efficient and **kNN** offering greater flexibility in complex data distributions.



Aspect	Rocchio Classification	k-Nearest Neighbors (kNN)
Approach	Centroid-based; calculates the average vector for each class and measures distance from centroid.	Instance-based; finds nearest neighbors among individual documents and uses majority vote.
Decision Boundary	Linear decision boundaries (depends on centroid positions).	Nonlinear decision boundary (flexible and data-dependent).
Classification Basis	Based on proximity to the centroid of each class.	Based on proximity to the closest k neighbors.
Sensitivity to Noise	Less sensitive to outliers since it uses averaged centroid vector.	More sensitive to noisy or outlier documents, especially if they are close neighbors.
Complexity	Requires calculating centroids once; efficient for large datasets.	Requires distance calculation for each new query, can be computationally intensive.

Figure 4: This is an example image.

3. Voronoi Tessellation and Decision Boundaries

The **Voronoi tessellation** concept is useful for visualizing kNN classification:

- The vector space is divided into **Voronoi cells**, where each cell contains all points (documents) closer to one specific training point than to any other.
- When $k = 1$, each document is classified by the class of its closest training document, resulting in **piecewise linear boundaries** between classes.

For larger k, these boundaries become less sharp but more stable, as they reflect the majority class of a group of nearby documents rather than a single nearest neighbor.

Example Visualization: For three classes represented as circles, triangles, and squares, each document is classified based on the majority of shapes in its k-nearest neighborhood.

4. Choosing and Optimizing k

Parameter Selection: The choice of k is usually determined by **cross-validation**:

- **Small k values**: Capture more localized details and adapt well to complex class shapes but can lead to overfitting.
- **Larger k values**: Provide smoother decision boundaries, reduce sensitivity to noise, and improve generalization but may overlook small class clusters.

Example of kNN Classification with Different k Values

Consider a dataset with two classes, A and B, and the following document vectors:

- **Class A**: $(2, 3), (3, 3), (3, 4)$
- **Class B**: $(6, 7), (7, 8), (8, 7)$
- **Test Document**: $(5, 6)$

- **For** $k = 1$: The closest document to $(5, 6)$ is $(6, 7)$ in Class B. The test document is assigned to Class B.
- **For** $k = 3$: The three closest documents are $(6, 7)$, $(7, 8)$, and $(3, 4)$. With two neighbors in Class B and one in Class A, the test document is assigned to Class B.
- **For** $k = 5$: If all five nearest neighbors are considered, with three from Class B and two from Class A, the test document remains classified as B.

5. Time Complexity of kNN

kNN is computationally intensive because:

- **Training time is minimal**, primarily involving storing the dataset.
- **Testing time** grows linearly with the size of the dataset, as it requires distance calculations between the test document and each training document.

In high-dimensional spaces, approximate methods like **locality-sensitive hashing (LSH)** or **inverted indexing** can improve efficiency, particularly when documents are sparse.

Summary for Part 3

The k-Nearest Neighbor (kNN) classifier is a powerful, flexible method suitable for capturing complex class distributions due to its instance-based, local approach. By adjusting k, it balances between sensitivity to noise and generalization, making it adaptable to various classification scenarios. kNN's reliance on distance calculations highlights the need for efficient indexing in large datasets, a critical factor in practical applications.

The final part will cover **Linear vs. Nonlinear Classifiers**, including the implications of the bias-variance tradeoff in selecting a suitable classifier. Let me know when you'd like to proceed.

Part 4: Linear vs. Nonlinear Classifiers and the Bias-Variance Tradeoff

In this final part, we explore the distinction between **linear** and **nonlinear classifiers** and the **bias-variance tradeoff**—a fundamental concept that guides the choice of classification methods by weighing model complexity against generalization ability.

1. Linear vs. Nonlinear Classifiers

Linear classifiers, as the name suggests, classify based on **linear decision boundaries**. A linear classifier assigns a document to a class based on whether it lies on one side or the other of a hyperplane in the vector space. Nonlinear classifiers, on the other hand, create more complex decision boundaries, capable of adapting to irregular or nested class distributions.

Linear Classifiers:

- Examples include **Naive Bayes**, **Rocchio**, and **Support Vector Machines (SVMs)** with linear kernels.
- Linear classifiers assume that the data can be separated by a straight line (or hyperplane in higher dimensions).
- The decision function for a linear classifier can be written as:

  $$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

  where $\mathbf{w}$ is the weight vector and b is the bias.

Nonlinear Classifiers:

- Examples include **kNN** (with arbitrary class boundaries) and **kernelized SVMs**.
- These classifiers use nonlinear decision boundaries and are more flexible in adapting to complex data distributions.
- Nonlinear classifiers are effective for **multimodal classes**, where class instances form multiple clusters or irregular patterns.

Example: Suppose we have two classes in a two-dimensional space, arranged in concentric circles. A linear classifier would struggle to separate them since it relies on straight lines. A nonlinear classifier, however, can capture the circular boundary and correctly classify instances.

2. The Bias-Variance Tradeoff

The **bias-variance tradeoff** is a central concept in model selection, balancing two sources of error:

- **Bias** is the error due to overly simplistic assumptions in the learning algorithm, leading to underfitting. High bias implies that the model consistently makes the same errors across different training sets.
- **Variance** is the error due to model sensitivity to small fluctuations in the training data, leading to overfitting. High variance means that the model's predictions vary greatly across different training sets.

The tradeoff is mathematically expressed as:

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

where the irreducible error represents the noise inherent in the data that no model can eliminate.

Linear Classifiers and Bias-Variance Tradeoff:

- **High Bias:** Linear classifiers are limited by their reliance on linear boundaries. If the true class boundaries are nonlinear, a linear model will systematically misclassify instances, resulting in high bias.
- **Low Variance:** Linear classifiers are typically stable across different training sets because they do not adjust to minor fluctuations in the data.

Nonlinear Classifiers and Bias-Variance Tradeoff:

- **Low Bias:** Nonlinear classifiers are more flexible and can better fit complex data structures, which reduces bias in cases where the class boundaries are not linear.
- **High Variance:** Because they can fit more complex boundaries, nonlinear classifiers are more likely to overfit the data, meaning their performance can vary significantly with changes in the training set.

---

3. Practical Implications of the Bias-Variance Tradeoff

Choosing Between Linear and Nonlinear Models:

- **When to prefer linear classifiers:** If the classes can be separated reasonably well by a straight line, a linear classifier is often preferred due to its simplicity and generalization ability. In high-dimensional text classification, linear classifiers like SVMs often perform well because the high dimensionality increases the likelihood that classes are linearly separable.
- **When to prefer nonlinear classifiers:** If the class distributions are complex, such as in multimodal or nested patterns, nonlinear classifiers like kNN or kernelized SVMs can better capture the underlying structure.

Example of Bias-Variance Tradeoff: Consider a classification problem with three models:

1. **A simple linear classifier** (e.g., using only a single feature): This model will likely underfit, with high bias but low variance.
2. **A more flexible linear model** (e.g., using a combination of features): This model may strike a good balance, achieving low bias without overly high variance.
3. **A highly flexible nonlinear model** (e.g., kNN with $k = 1$): This model will have low bias but high variance, potentially overfitting to noise.

---

4. Optimizing the Bias-Variance Tradeoff

Cross-Validation is typically used to find an optimal balance between bias and variance. By partitioning the data into training and validation sets, cross-validation helps assess the model's generalization ability on unseen data, highlighting issues with either underfitting or overfitting.

Regularization in Linear Models:

- Regularization techniques like **L2 regularization** (used in Ridge Regression or SVMs) add a penalty to the model complexity, which helps reduce variance.
- For instance, in text classification, adding regularization to logistic regression or SVMs prevents overfitting by discouraging overly complex decision boundaries.

Tuning Parameters in Nonlinear Models:

- For kNN, adjusting k helps control the model's flexibility. A small k reduces bias but increases variance, while a larger k smoothens the decision boundary, reducing variance but increasing bias.
- **Kernel methods** in SVMs allow flexible boundaries while maintaining some regularization, making them effective for nonlinear separability with controlled variance.

Summary for Part 4

The bias-variance tradeoff is essential in choosing between linear and nonlinear classifiers. While linear models offer simplicity and generalization, nonlinear models provide flexibility but risk overfitting. By balancing bias and variance—through cross-validation, parameter tuning, and regularization techniques—one can optimize the classifier's performance, selecting a model that best fits the data's underlying structure without overfitting to noise.

This concludes the in-depth exploration of the chapter's core concepts. Let me know if you'd like further examples, summaries, or elaborations on any part!