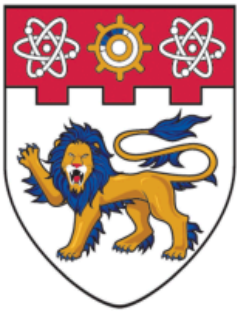


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

AY2025/2026 Semester 1
SC4020 DATA ANALYTICS & MINING
Group 32

Project 1

Student Name
Chow Jia Hui
Lim Pei Jun, Rena
Wonyeong Yoon
Shem Lau Wei Yi

Table Of Contents

1.0 Abstract

Similarity search is a fundamental technique in data mining, playing a crucial role in identifying patterns and relationships within datasets [1]. This technique is critical for applications ranging from recommendation systems to patient record retrieval. It is also important to ensure these searches are efficient and accurate especially for large-scale datasets. This project presents a comprehensive technical review and comparison of two primary approaches to similarity search: Brute Force K-Nearest Neighbours (K-NN) Retrieval with Cosine/Euclidean and Approximate Nearest Neighbour (ANN) using Annoy library. We evaluated these methods using two contrasting healthcare datasets: the Breast Cancer Wisconsin (Diagnostic) Dataset [2] and the MIMIC-III Clinical Dataset [3]. The performance is assessed using Precision@k and nDCG@k metrics, together with query time efficiency. Our experimental results demonstrated that while Brute Force K-NN ensures optimal accuracy, its computational cost becomes impractical for large-scale datasets. On the other hand, ANN provides substantial speed improvements without much loss in accuracy, making it suitable for large-scale, real-world applications. Furthermore, we compared the effectiveness of Cosine and Euclidean distance metrics, which revealed that Cosine distance typically delivers better retrieval performance for these datasets as it focuses on feature patterns rather than magnitudes.

2.0 Introduction

In today's data-driven world, searching through vast amounts of data to find similar items is a fundamental operation used in various applications from e-commerce recommendations to clinical decision support. This process, known as similarity search, moves beyond traditional database searches based on fixed numeric criteria, which instead aims to identify items that are alike based on certain criteria [4].

The importance of similarity search is evident in many fields. In e-commerce, it recommends products similar to what a user has viewed or purchased. In healthcare, our main focus for this study, it helps with medical imaging by comparing patient scans to identify similar cases and assist in diagnosis. However, there are challenges in these applications: achieving high accuracy in retrieval to ensure that the results are meaningfully similar, and maintaining high efficiency to make the search feasible on large-scale datasets [4].

In this project, we examine and assess the effectiveness of different similarity search methodologies by utilising the knowledge acquired in NTU's SC4020 Data Analytics and Mining Course. The Breast Cancer Wisconsin (Diagnostic) Dataset [2] and the MIMIC-III Clinical Dataset [3] will be used for this project. These datasets feature contrasting characteristics in terms of sample size and dimensionality, allowing for comprehensive evaluation of the search techniques.

Our team will use Brute Force K-Nearest Neighbours (K-NN) Retrieval with Cosine/Euclidean and Approximate Nearest Neighbour (ANN) using Annoy library to tackle the similarity search task. Brute Force K-NN ensures accuracy by comparing query points to every other point in the dataset, serving as an exact but computationally costly baseline. In contrast, ANN methods like Annoy use advanced indexing structures, such as random projection trees, to quickly retrieve approximate neighbours, trading a marginal degree of accuracy for significant increases in search speed and scalability.

3.0 Problem Statement

This project's main goal is to assess and contrast the performance of exact and approximate similarity search methods on real-world datasets. In applications such as clinical decision support or large-scale user recommendations, the ability to quickly find similar records is essential. However, a significant challenge exists in balancing the trade-off between search accuracy and computational efficiency, especially when dataset sizes get larger and larger to hundreds of thousands of samples.

In this project, we focus on two contrasting datasets:

1. **Breast Cancer Wisconsin (Diagnostic) Dataset [2]**
   This dataset contains computed features from digitised images of breast masses, describing characteristics like radius, texture, and perimeter. It is a clean, well-labelled, and higher-dimensional dataset, serving as an ideal controlled case for evaluating retrieval accuracy against a ground truth diagnosis.

2. **MIMIC-III Clinical Dataset [3]**
   This dataset provides a large collection of de-identified health data, including vital signs such as heart rate and blood pressure. It is a large-scale, lower-dimensional dataset lacking explicit labels, simulating a real-world scenario where similarity must be inferred directly from the data patterns.

The challenge lies in understanding how the different search algorithms **Brute Force K-NN** and **Annoy (ANN)** perform on these datasets. Each algorithm has distinct characteristics:

- **Brute Force K-NN** provides exact, optimal results for nay distance metric but has a linear time complexity that becomes prohibitively slow for large datasets.
- **Annoy (ANN)** achieves sub-linear query times and high scalability through index-based search, but its results are approximate and its accuracy depends on parameters like the number of trees.

Furthermore, we will evaluate these methods using both Euclidean and Cosine distance metrics to understand how the choice of distance measure impacts retrieval quality based on the underlying data patterns.

The goal of this project is to apply these algorithms and metrics to both datasets, evaluate their performance in terms of accuracy and efficiency, and determine the most suitable approach for each dataset type. Through this analysis, we aim to provide a comprehensive understanding of the practical trade-off in modern similarity search and offer insights into their effective application.

4.0 Methods and Experiments

4.1 Dataset 1: Breast Cancer Wisconsin (Diagnostic) Dataset

The Breast Cancer Wisconsin (Diagnostic) Dataset is a collection of features computed from digitalised images of fine-needle aspirates (FNA) of breast masses. It is used to analyse the characteristics of cell nuclei and facilitate the diagnosis of breast cancer.

4.1.1 Dataset Cleaning and Preprocessing

The dataset contains 569 samples and 30 numeric features used for prediction, with one class label for the diagnosis. Diagnosis ‘M’ represents ‘Malignant tumour’ and ‘B’ represents ‘Benign tumour’. The diagnosis is encoded with binary numbers, mapping ‘M’ to ‘1’ and ‘B’ to ‘0’.

Remove Irrelevant Columns

Columns ‘id’ and ‘Unnamed: 32’ are dropped as they do not carry meaningful information for our analysis.

Feature Scaling: Standardisation

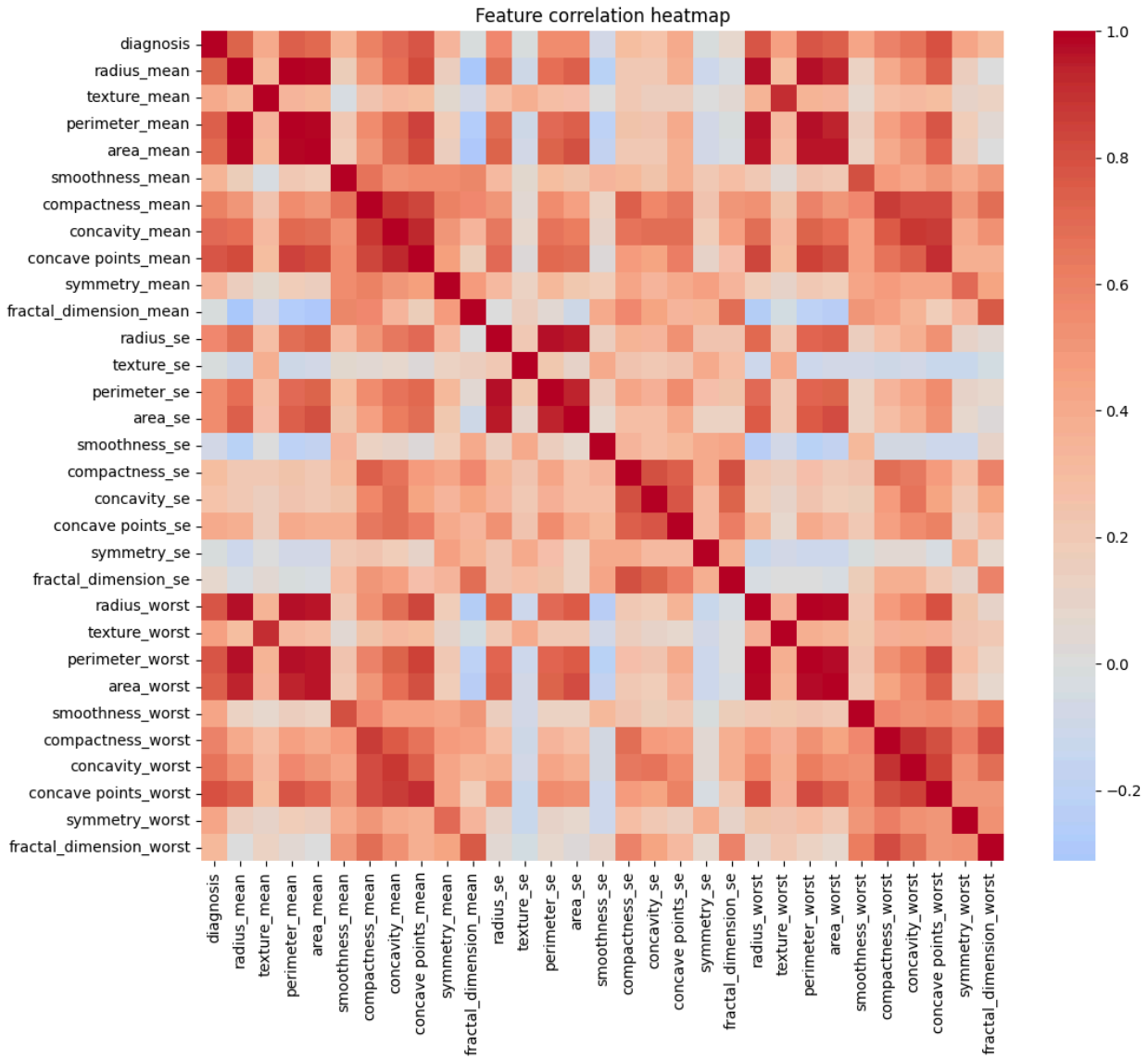
The data is standardized using Standardscaler() so features are scaled to follow standard normal distribution. This ensures that all features contribute equally to distance calculations.

Split the Data into Features and Target Variable

Independent variables (features) and the dependent variable (target) are prepared separately. ‘Diagnosis’ is the dependent variable.

Correlation Analysis

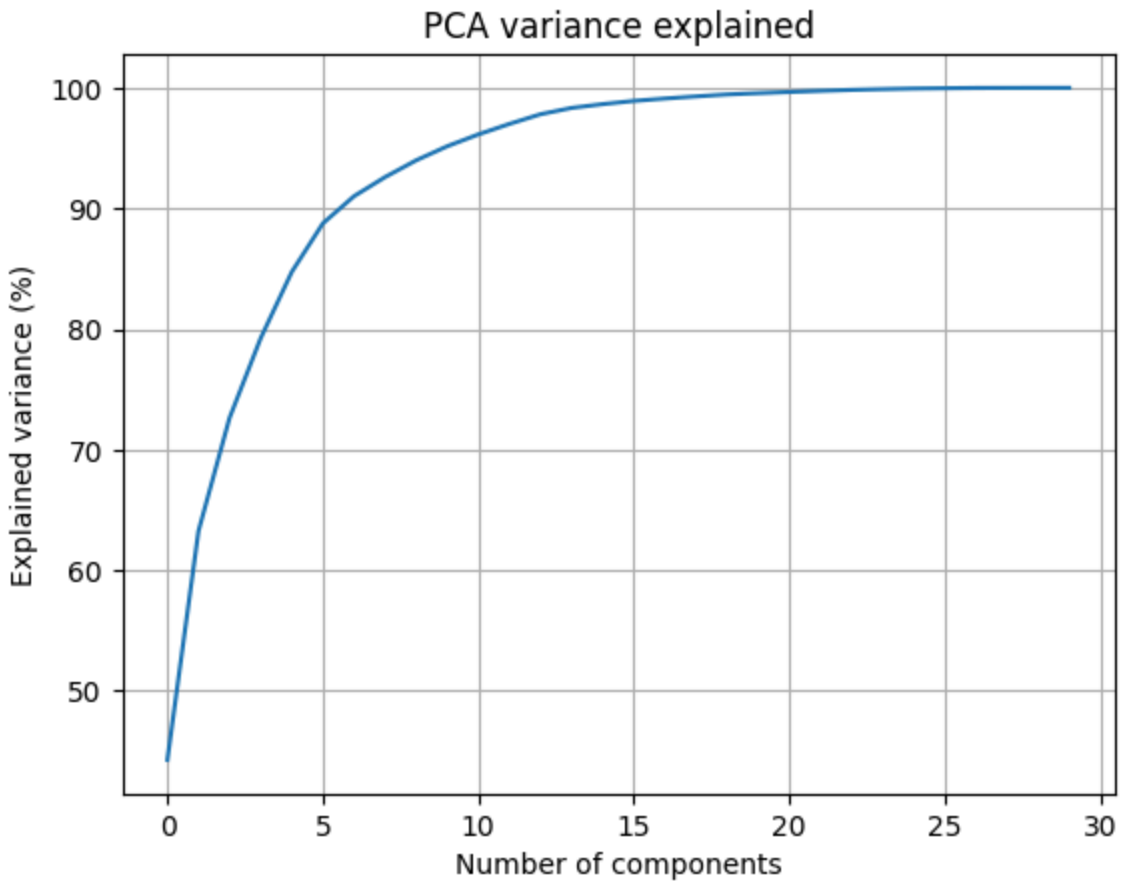
We can investigate the correlation between our dependent variable (diagnosis) and all predictors using a correlation plot. corr() helps to compute the Pearson correlation coefficients between all pairs of variables in our dataset.



From the correlation plot, radius, perimeter, area and concave points have correlation scores closest to 1, which means they exhibit the largest correlation with diagnosis.

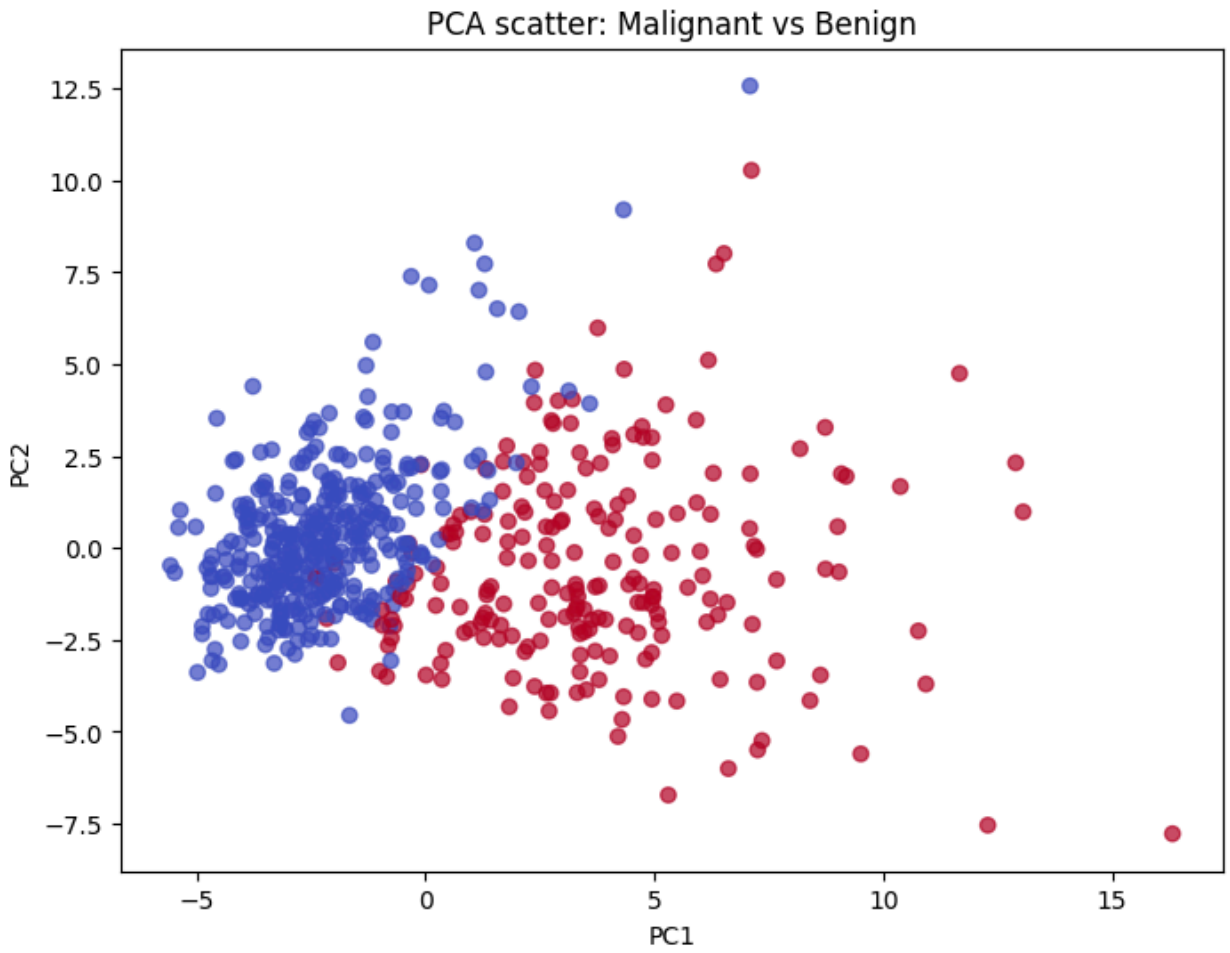
Principle Component Analysis (PCA)

PCA is a dimensionality reduction technique that allows us to transform the dataset into a set of principal components that retain most of the original information [5]. It reduces computational complexity due to a smaller feature space and better visualisation of the variations. The components are ordered in a descending order, by the amount of variance they can capture [6].



From the graph, we can see that the first 5 components can account for close to 90% of the variance. While there are 30 numerical features, only 5 to 10 components are necessary to capture the most important variance (~90% to 95%) in the original dataset. This allows us to reduce dimensionality without losing much information. It is useful for similarity searches as we are working with a less complex algorithm which minimises issues such as multicollinearity and overfitting.

Two main components are calculated in PCA, namely the first principal component (PC1) and second principal component (PC2). PC1 represents the direction in space along which the data points have the highest variance, while PC2 captures the second highest variance remaining for the data points. PC1 and PC2 have a correlation that is equal to zero. A scatter plot is used to show the relationship between PC1 and PC2.



Here, red represents malignant and blue represents benign. From the graph, we can see that the malignant and benign data points are separated into two distinct clusters based on PC1. Such differences in the clusters are likely due to the features that have heavy influence on PC1. On the other hand, PC2 captures little differences between the two clusters. Nonetheless, PC2 is still useful to show differences within each cluster. This means that PC2 helps to reveal variations in tumour characteristics among patients with the same diagnosis.

By using only PC1 and PC2, it allows us to reduce the dimensionality from 30 features to 2 features. While this allows us to capture the most variance, it does not necessarily capture the most class-relevant information. The variations may contain noises or unrelated features that are not useful for distinguishing classes.

Instead of using the two principal components that capture the largest variance, a better approach for the algorithm will be to use PCA with n_component = 0.95. This lets PCA select the minimum number of principal components that will explain 95% of the relevant variation. Full singular Value Decomposition (SVD) method is used in PCA to ensure maximum accuracy.

```
# Dimensionality reduction (keeping ~95% variance to speed up search)
pca = PCA(n_components=0.95, svd_solver='full')
X_vec = pca.fit_transform(X_scaled)
X_vec.shape
```

```
(569, 10)
```

From the output, PCA has reduced dimensionality to 10 while retaining 95% of the total relevant variation.

4.1.2 Method 1: Brute Force K-NN Retrieval With Cosine/Euclidean

Background Information

The k-Nearest Neighbours (KNN) algorithm is one of the most popular and conceptually simple methods for similarity search used in machine learning [7]. It works by identifying the k most similar neighbours to a given query based on distance metrics. These neighbours are analysed to understand similarity patterns or retrieve similar samples. In order to identify the nearest neighbours, we have to calculate the distance between the neighbours and the query point. Under brute force K-NN, all pairwise distances are calculated. We will be implementing two kinds of distance metrics, namely Euclidean Distance and Cosine Distance.

Euclidean Distance

Euclidean distance measures the shortest straight line distance between the query point and the neighbouring point. It is particularly useful when variations in the magnitude of the features carry important information. For example, differences in radius, smoothness and compactness can reflect physical distinctions between malignant and benign tumors. The formula for euclidean distance is as shown:

$$d_{euclidean}(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

where,

- i ranges from 1 to n
- $(x_1, x_2, x_3, ..., x_n)$ is the coordinate for the first data point
- $(y_1, y_2, y_3, ..., y_n)$ is the coordinate for the second data point

Cosine Distance

To talk about cosine distance we will first have to understand cosine similarity. Cosine similarity measures how similar two data points are based on the direction they point [8]. In an n-dimensional space, each data point is represented by a vector. Cosine similarity ranges from -1 to 1.

- cosine similarity = 1 indicates that the vectors are pointing in the same direction
- cosine similarity = 0 indicates that the vectors are orthogonal
- cosine similarity = -1 indicates that the vectors are pointing in opposite directions

The formula for cosine similarity is as shown:

$$similarity(x, y) = cos(\theta) = \frac{\bar{x} \cdot \bar{y}}{||\bar{x}||\,||\bar{y}||} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2}\sqrt{\sum_{i=1}^{n} (y_i)^2}}$$

where,

- θ is the angle between non-zero vectors x and y
- i ranges from 1 to n
- $(x_1, x_2, x_3, ..., x_n)$ is the vector coordinate for the first data point
- $(y_1, y_2, y_3, ..., y_n)$ is the vector coordinate for the second data point

While a higher value for cosine similarity indicates greater similarity, it is counter-intuitive when used in K-NN. K-NN algorithm is designed to work with distance where a lower value indicates greater similarity. Thus, we use cosine distance. The formula for cosine distance is as shown:

$$d_{cos}(x, y) = 1 - similarity(x, y)$$

Cosine distance ranges from 0 to 2:

- When cosine similarity = 1, **cosine distance** = 1-1 = **0** indicates that the vectors are pointing in the **same direction**
- When cosine similarity = 0, **cosine distance** = 1-0 = **1** indicates that the vectors are **orthogonal**
- When cosine similarity = -1, **cosine distance** = 1-(-1) = **2** indicates that the vectors are pointing in **opposite directions**

Cosine distance is useful when absolute magnitude does not matter as much as the relative relationship between the features. For example, two tumours may have different overall magnitudes but if the relative proportion of smoothness and concavity is similar, they can still belong to the same class. Moreover, when working with a high dimensional space, cosine distance is more suitable as compared to straight line distance calculations.

Evaluation of Algorithm Efficacy before Implementation

Before implementing K-NN retrieval, we use Retrieval Evaluation Metrics (Precision@k and NDCG@k) and Average Query Time to assess the efficiency of our K-NN algorithm.

Retrieval evaluation metrics are used to check if our algorithm can retrieve relevant information and return accurate results. Precision@k metric measures how many of the top-k retrieved neighbours share the same diagnosis as the query [9]. Normalised Discounted Cumulative Gain@k (NDCG@k) rewards correct order of retrieval, that is, to retrieve the most relevant neighbours first [9].

Average query time measures the time taken for the algorithm to retrieve neighbours for one query.

```
cosine    P@10=0.962  nDCG@10=0.988 avg query time=0.003
euclidean  P@10=0.942  nDCG@10=0.981 avg query time=0.001
```

From the above output, we can see that both distance metrics performed strongly in terms of retrieval evaluation metrics even though cosine distance performed slightly better than euclidean distance.

P@10=0.962 using cosine distance and P@10=0.942 using euclidean distance. On average, 96.2% of the top 10 neighbours found using cosine distance share the same diagnosis label as the query, compared to 94.2% of the top 10 neighbours found using euclidean distance. Cosine distance is slightly better when retrieving neighbours with the same diagnosis.

nDCG@10=0.988 using cosine distance and nDCG@10=0.981 using euclidean distance. Cosine distance retrieves relevant neighbours earlier in order than euclidean distance. This indicates that cosine distance provides an overall better neighbour ranking quality.

Since cosine distance performed better than euclidean distance, it indicates that relative similarity between tumour features is more important than absolute magnitude of tumour features. Looking at the relationship between features could provide a more effective way to distinguish between malignant and benign tumors.

Lastly, the average query time using both distance metrics are quite close, with euclidean distance being insignificantly faster. This is reasonable as both distance metrics involve calculations for all pairwise distances. It indicates that computational cost will be similar for both metrics.

Implement K-NN Brute Force Retrieval on Dataset

Step 1: Data Preparation - Standaridisation

For algorithms like K-NN that rely on distance metrics to evaluate similarity retrieval, it is crucial to standardise the dataset to ensure all features contribute equally to the computation. Standardisation removes all scale factors of the features and prevents bias in distance calculation.

Step 2: Execute K-NN Algorithm

```
# Exact (Brute-force) k-NN retrieval with Cosine/Euclidean

# Brute Force k-NN function
def brute_force_knn(X, q_index, k=10, metric="cosine"):
    q = X[q_index]
    d = pairwise_distances(q[None, :], X, metric=metric)[0]
    # exclude self
    order = np.argsort(d)
    order = order[order != q_index][:k]
    return order, d[order]
```

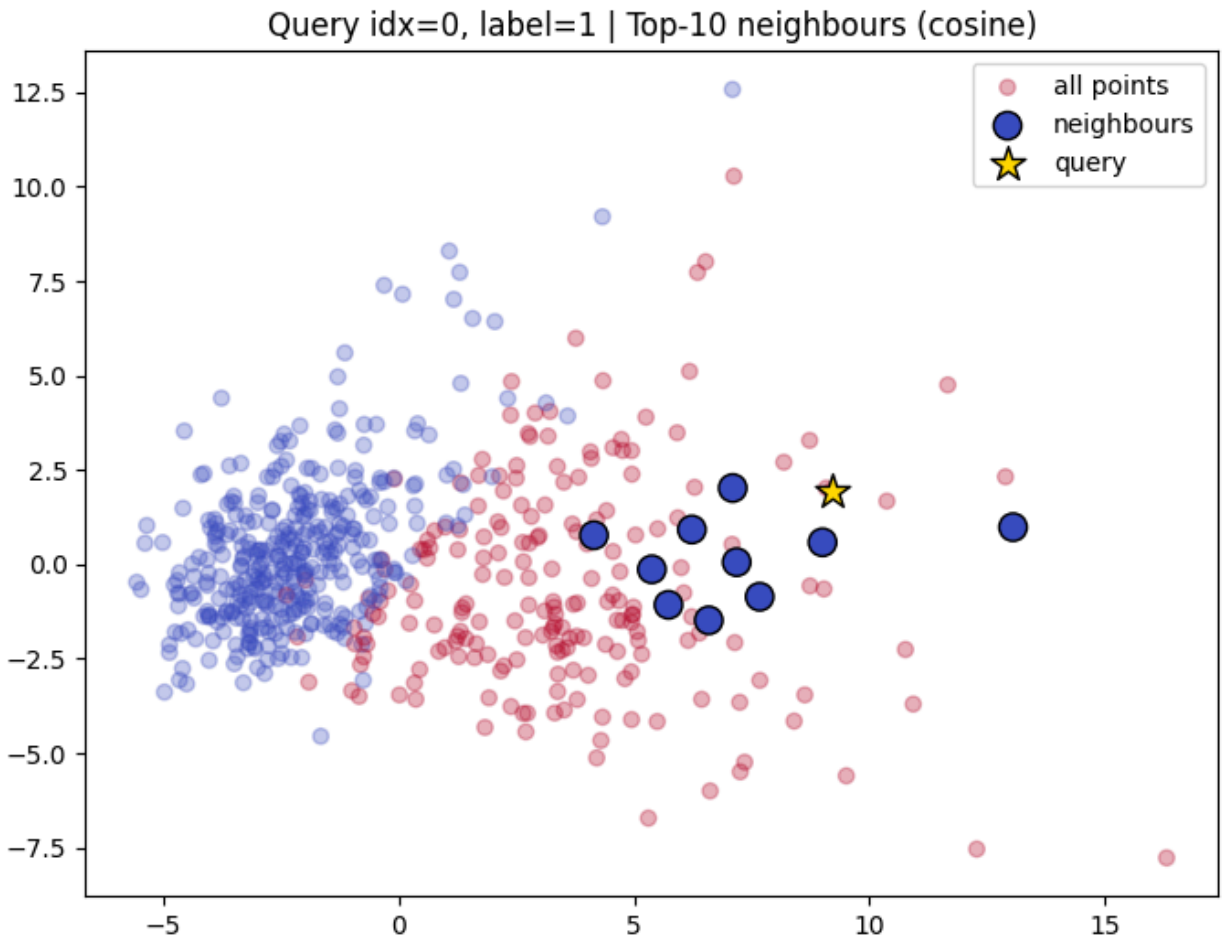
For a query sample at index, q_index, we will compute all pairwise distances between the query and all other data points in the dataset, X, using the chosen distance metric (Cosine/Euclidean). The distances will be sorted in ascending order and the indexes of the top k neighbours will be returned.

Step 3: Visualise Query Point and Neighbouring Point on Scatter Plots

For better visualisation of our results, we will use n_component=2 in PCA to reduce our high-dimensional data into a two dimensional space. n_component=0.95 is used when evaluating algorithm accuracy but not suitable for visualisation here.

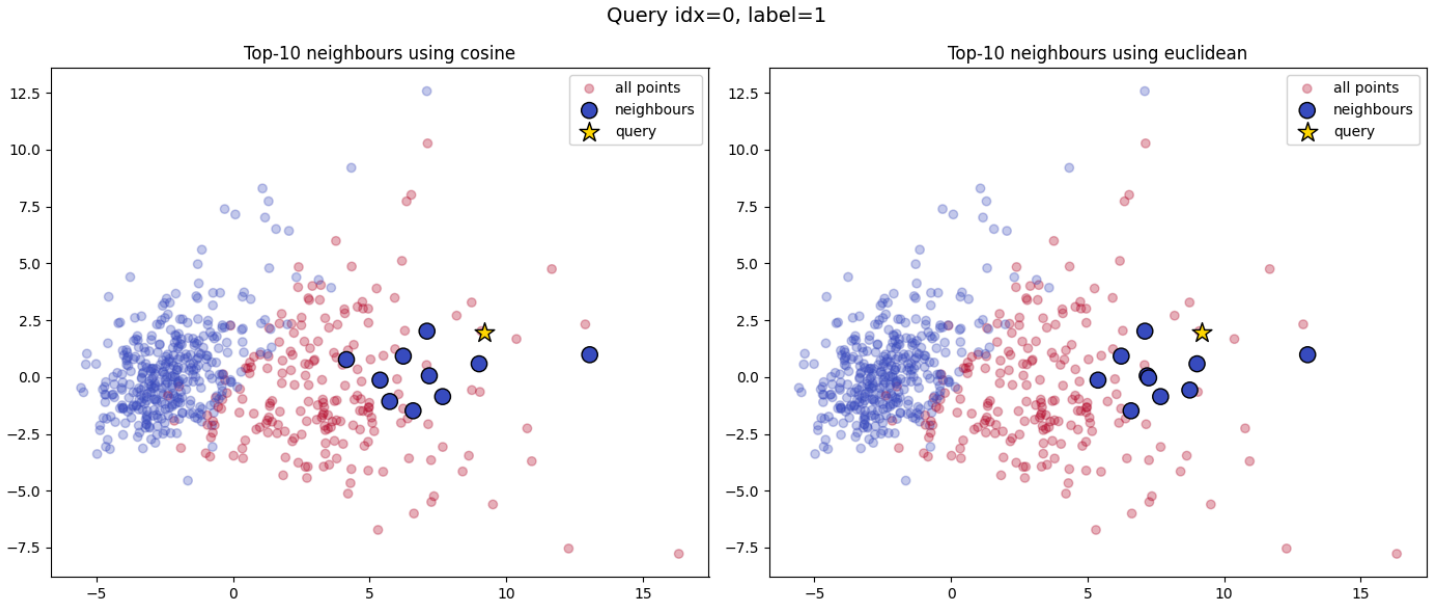
q_index=0 is set as the query point and will be marked with a “gold star”. The top k most similar points are marked with a “bigger blue dot”. All the other points are marked with their label colour (red or blue) using a “smaller dot”.

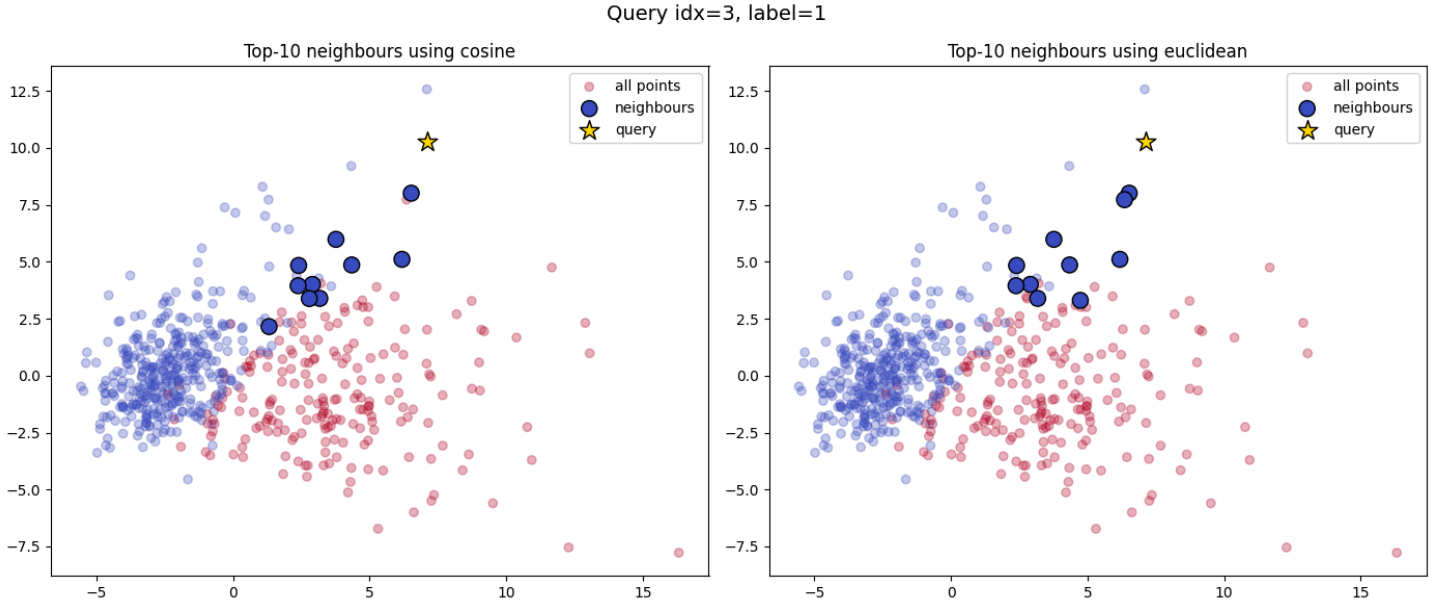
```python
# Reduce to 2D for plotting (even if you used 10D PCA for search)
pca2 = PCA(n_components=2)
X_2d = pca2.fit_transform(X_scaled)

def plot_query_and_neighbours(q_index, k=10, metric="cosine"):
    idxs, dists = brute_force_knn(X_vec, q_index, k=k, metric=metric)

    plt.figure(figsize=(8,6))

    # Plot all points faintly
    plt.scatter(X_2d[:,0], X_2d[:,1],
                c=y, cmap="coolwarm", alpha=0.3, label="all points")

    # Plot neighbours
    plt.scatter(X_2d[idxs,0], X_2d[idxs,1],
                edgecolor="black", c=y[idxs],
                cmap="coolwarm", s=120, marker="o", label="neighbours")

    # Plot query
    plt.scatter(X_2d[q_index,0], X_2d[q_index,1],
                c="gold", edgecolor="black",
                s=200, marker="*", label="query")

    plt.title(f"Query idx={q_index}, label={y[q_index]} | Top-{k} neighbours ({metric})")
    plt.legend()
    plt.show()

# Example: malignant case
plot_query_and_neighbours(q_index=0, k=10, metric="cosine")
```



Experimental Analysis

To better compare the two distance metrics, we have created 2 side-by-side plots to represent cosine distance on the left and euclidean distance on the right for the same query point. Two query points are selected for analysis: a normal query point (q_index=0), and an outlier query (q_index=3). This allows us to observe how each distance metric will perform under different query characteristics.





From the output, we observe that the neighbors chosen using euclidean distance and cosine distance are different for the same query point.

The two graphs in the first row illustrate the difference when a normal malignant query (q_index=0) is selected. The normal point selected lies close to the main cluster of malignant points. Both euclidean and cosine metric retrieve a similar set of neighbours, all of which

belongs to the malignant cluster. This shows that for data points located near the cluster centroid, the distance metric performed consistently well.

However, when an outlier malignant query (q_index=3) is selected, the difference in retrieval performance is clear between the two metrics. Even though both euclidean and cosine metric retrieve malignant and benign points, euclidean metric was able to retrieve more malignant points.

Experimental Result

Overall, this result indicates that while both metrics performed similarly for data points in dense and well clustered regions, euclidean distance provides a more accurate retrieval for points that are outliers and lying near the boundary. This highlights that absolute magnitude in features is more significant than relative relationship between features, for distinguishing characteristics in tumour.

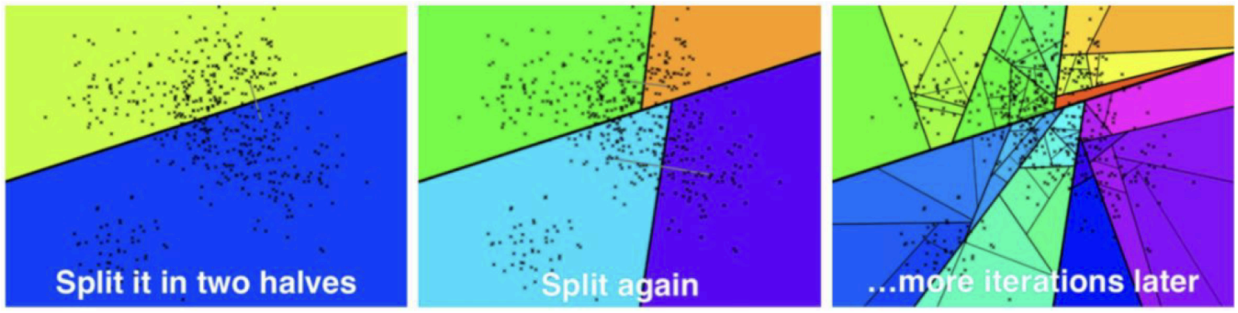
4.1.3 Method 2: Approximate Nearest Neighbours (ANN) Search

Background Information

ANN search is an algorithm used in searching for the nearest neighbours. It finds neighbours for the query point using approximation. Unlike traditional nearest neighbour algorithms which search exhaustively through all the data points, ANN settles for neighbours that are close enough though not necessarily the closest. This technique prioritizes efficiency and seeks to balance accuracy and computational feasibility, especially when dealing with a large dataset [10].

We will be using Approximate Nearest Neighbours Oh Yeah (ANNOY) to implement ANN search. ANNOY is an open source C++ library that uses its algorithm to perform similarity search. ANNOY builds multiple random projection trees which recursively partition the space using random hyperplanes. It starts with indexing.

Tree Construction (Indexing Phase)
We will be building the tree structure in a recursive process. In the first iteration, ANNOY picks two random vector data points, a and b, from the dataset. A hyperplane that lies equidistant from a and b is used to split the full hyperspace. The set of vector data points that lie in the ‘left half’ of the hyperspace is assigned to the left side of the tree while the data points in the ‘right half’ of the hyperspace is assigned to the right side of the tree. In the second iteration, each side of the tree repeats the process again. This iteration continues until a lead node has fewer than a pre-defined number of elements m. m is a user-defined parameter that defines the granularity of the partitions.



[10]

Mathematically, each split can be expressed with the formula shown:

$$sign(z) = \begin{cases} +1, if\ z > 0 \\ -1, if\ z < 0 \end{cases}$$

$$h(z) = sign(w \cdot x)$$

where,

- w is the random vector that defines the hyperplane
- x is the vector data point
- $w \cdot x$ is the dot product between w and x
- sign() determines which side of the hyperplane x belongs to
- If $w \cdot x > 0$, then h(x) = $sign(w \cdot x)$ = +1. The point goes to the right branch
- If $w \cdot x < 0$, then h(x) = $sign(w \cdot x)$ = -1. The point goes to the left branch

Query Phase

Once the index is fully built, we can move on to retrieve approximate nearest neighbours. Given a query vector, the search process traverses down from the root to the leaf of the trees:

1. At each node, ANNOY compares the distance from the query to the left and right reference vector (a and b). ANNOY assigns the query to the side of the hyperplane where it is closer to the reference point.
2. The algorithm follows the branch until it reaches a leaf node containing at most m vectors.
3. Within that leaf node, candidate vectors are ranked by their distance to the query and the top k nearest neighbours.

We will be using two kinds of distance metric in ANNOY, namely Angular Distance and Euclidean Distance. The angular distance in ANNOY is conceptually similar to cosine distance where both measure the angles between two vectors. However, angular computes the normalised angle between vectors while cosine directly uses 1- $cos(\theta)$. The formula for angular distance is as shown:

$$d_{angular} = \frac{arccos(\frac{\bar{x} \cdot \bar{y}}{||\bar{x}|| ||\bar{y}||})}{\pi}$$

where,

- $(x_1, x_2, x_3, ..., x_n)$ is the coordinate for the first data point
- $(y_1, y_2, y_3, ..., y_n)$ is the coordinate for the second data point

Evaluation of Algorithm Efficacy before Implementation

Similar to Brute Force K-NN, we will be assessing the efficiency of the algorithm using Retrieval Evaluation Metrics (Precision@k and NDCG@k) and build time.

```
Angular Annoy build_time=0.096s, P@10=0.964, nDCG@10=0.988
Euclidean Annoy build_time=0.055s, P@10=0.943, nDCG@10=0.981
```

From the output, we can see that both distance metrics achieve great results in terms of retrieval evaluation metrics with angular distance performing slightly better than euclidean distance.

P@10=0.964 using angular distance and P@10=0.943 using euclidean distance. On average, 96.4% of the top 10 neighbours found using angular distance share the same diagnosis label as the query, compared to 94.3% of the top 10 neighbours found using euclidean distance. Angular distance is slightly better when retrieving neighbours with the same diagnosis, suggesting that it can better preserve directional similarity of the feature vectors in a high-dimensional space.

nDCG@10=0.988 using angular distance and nDCG@10=0.981 using euclidean distance. Angular distance not only retrieves relevant neighbours more accurately but also rank them better than euclidean distance.

As for the build time, angular distance uses 0.096s while euclidean distance uses 0.055s. Angular distance takes slightly longer due to computational complexity.

Implement ANN search on Dataset

Step 1: Preprocessing to Build Annoy Index

To initialise the Annoy index, we use X_vec.shape[1] to specify the number of dimensions of each vector. Next, we will add all the items from the dataset into the index. To do so, each feature vector in X_vec is converted to a Python list and added to the index. The feature vectors are assigned an unique integer ID corresponding to its position in the dataset. Lastly, we build the index using ann.build(50). 50 trees are constructed internally. Increasing the number of trees increases the accuracy of the search by exploring more partitions of the hyperspace. However, this also increases the time it takes to build the index and more memory is required. Once the index is built, we can now efficiently query it for nearest neighbors of any vector without scanning the entire dataset.

```
for ax, metric in zip(axes, ["angular", "euclidean"]):
    # Build the appropriate Annoy index
    ann = AnnoyIndex(X_vec.shape[1], metric)
    for i, v in enumerate(X_vec):
        ann.add_item(i, v.tolist())
    ann.build(50)
```

Step 2: Query Nearest Neighbours

To find the nearest neighbours, we query the annoy index using the feature vector represented by X_vec[q_index]. We return k+1 neighbours instead of k neighbours because the query itself may appear as the closest neighbour, which we will be removing in the next step.

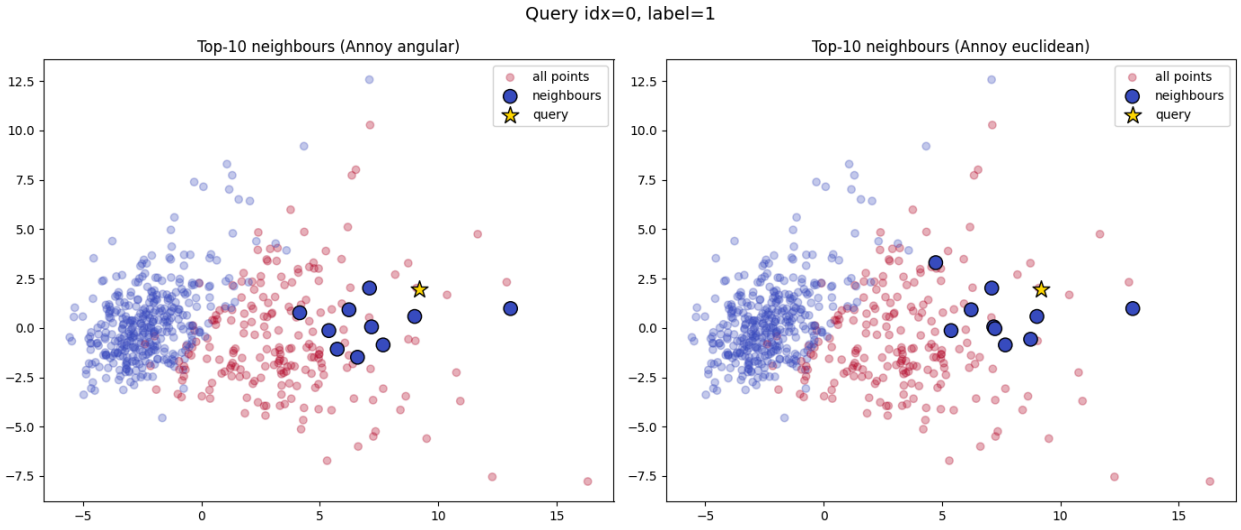
Step 3: Remove Query from Results

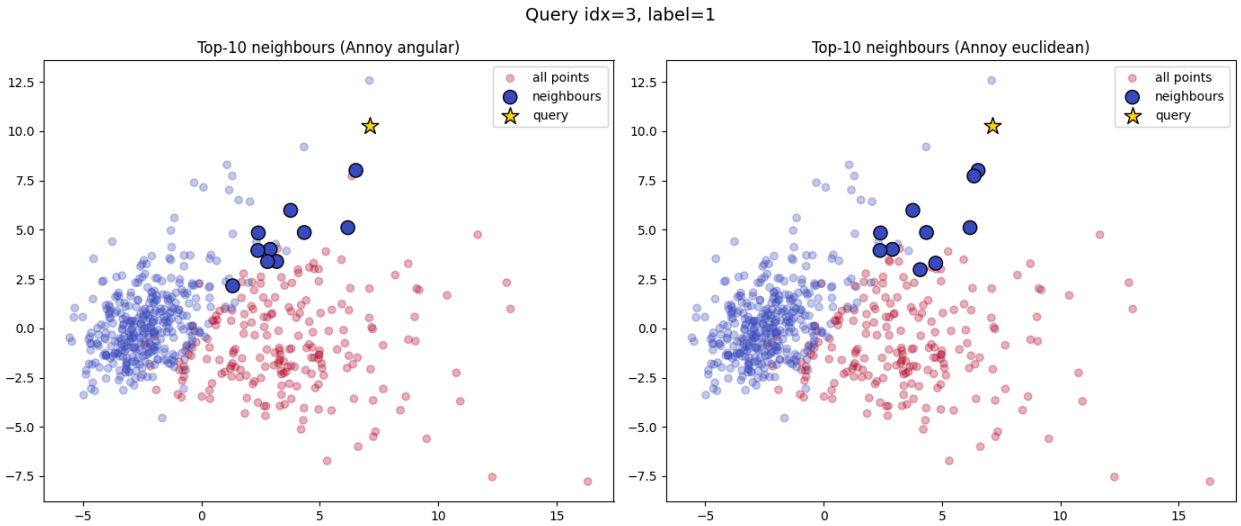
```
# Drop query itself if included
if idxs[0] == q_index:
    idxs = idxs[1:k+1]
else:
    idxs = idxs[:k]
```

The query is removed if itself is included.

Step 4: Visualise Query Point and Neighbouring Point on Scatter Plots

Similar to Brute Force K-NN Retrieval, we will be using a 2x2 plot to visualise the results.





From the output, we can observe that ANN search produces good results when a normal malignant query point is chosen (q_index=0). However, when an outlier malignant query point (q_index=3) is selected, both angular distance and euclidean distance retrieves both malignant and benign points. Euclidean still outperformed angular using ANN search.

Experimental Results

ANN search is a robust algorithm when querying a normal point, regardless of the distance metrics used. However, when querying an outlier or boundary point, angular distance will be a better choice for improving neighbours retrievals.

Comparison Between Brute Force K-NN and ANN Search

To determine if Brute Force K-NN or ANN Search is a better method for similarity search, we will compare the neighbours returned by both methods for the same query point. We will evaluate the neighbours across angular(cosine) distance and euclidean distance. To do so, we can evaluate how similar the neighbours are in terms of the neighbours retrieved and the ranking order of the neighbours. We will be employing Jacquard Index and Spearman's Rank Order Coefficient.

Jacquard Index

Jacquard similarity measures the similarity between two sets of vectors [11]. The index ranges from 0 to 1, with an index=1 indicating exact symmetry between the sets. The formula is as shown:

$$J(N_{cos}, N_{euclidean}) = \frac{|N_{cos} \cap N_{euclidean}|}{|N_{cos} \cup N_{euclidean}|}$$

where,

- N_{cos} is the set of neighbours retrieved with cosine distance

- $N_{euclidean}$ is the set of neighbours retrieved with euclidean distance
- $\left|N_{cos} \cap N_{euclidean}\right|$ is the number of neighbours common to both sets
- $\left|N_{cos} \cup N_{euclidean}\right|$ is the number of unique neighbours across both sets

Spearman's Rank Order Coefficient

Spearman's correlation coefficient (ρ) measures the strength and direction between two sets of ranked vectors [12]. For the neighbours in common, we will compare their ranks on each list and ρ will reveal if cosine and euclidean rank the common neighbours in similar order. The formula is as shown:

$$\rho = 1 - \frac{6\sum_{i=1}^{n}(d_i)^2}{n(n^2-1)}$$

where,

- n is the number of top k neighbours retrieved
- d_i is the difference in rank of the i-th item between the two sets of vectors

ρ ranges from -1 to 1.

- ρ=1 indicates a perfect positive correlation, meaning that the two set of vectors are in the same rank
- ρ=0 indicates no correlation, meaning there is no consistent relationship between the rankings of the two set of vectors
- ρ=-1 indicates a perfect negative correlation, meaning that the ranking between the two set of vectors are completely opposite

Implementation of Jacquard Index and Spearman's Rank Order Coefficient

```
# Compare brute-force vs Annoy using cosine/angular
summary, table = neighbour_overlap_report_ann(X_vec, y, q_index=0, k=10, metric="angular")
print(summary)
display(table.head(20))
```

```
# Compare brute-force vs Annoy using Euclidean
summary, table = neighbour_overlap_report_ann(X_vec, y, q_index=0, k=10, metric="euclidean")
print(summary)
display(table.head(20))
```

	idx	method	rank	distance	label	is_common
10	77	annoy	1	0.278616	1	True
11	108	annoy	2	0.397988	1	True
12	25	annoy	3	0.451296	1	True
13	45	annoy	4	0.504020	1	True
14	300	annoy	5	0.546312	1	True
15	393	annoy	6	0.556359	1	True
16	2	annoy	7	0.559175	1	True
17	392	annoy	8	0.567202	1	True
18	302	annoy	9	0.571369	1	True
19	181	annoy	10	0.576127	1	True
0	77	brute_force	1	0.038813	1	True
1	108	brute_force	2	0.079197	1	True
2	25	brute_force	3	0.101834	1	True
3	45	brute_force	4	0.127018	1	True
4	300	brute_force	5	0.149229	1	True
5	393	brute_force	6	0.154768	1	True
6	2	brute_force	7	0.156338	1	True
7	392	brute_force	8	0.160859	1	True
8	302	brute_force	9	0.163231	1	True
9	181	brute_force	10	0.165961	1	True

	idx	method	rank	distance	label	is_common
10	77	annoy	1	4.285353	1	True
11	25	annoy	2	4.721225	1	True
12	108	annoy	3	5.466882	1	True
13	393	annoy	4	5.710922	1	True
14	563	annoy	5	5.895807	1	True
15	181	annoy	6	5.916163	1	True
16	302	annoy	7	5.924642	1	True
17	45	annoy	8	6.057753	1	True
18	323	annoy	9	6.196607	1	True
0	77	brute_force	1	4.285352	1	True
1	25	brute_force	2	4.721225	1	True
2	108	brute_force	3	5.466882	1	True
3	393	brute_force	4	5.710923	1	True
5	563	brute_force	6	5.895807	1	True
6	181	brute_force	7	5.916164	1	True
7	302	brute_force	8	5.924642	1	True
8	45	brute_force	9	6.057753	1	True
9	323	brute_force	10	6.196607	1	True
19	22	annoy	10	6.255489	1	False
4	300	brute_force	5	5.876058	1	False

The two summary tables here represent the ranking of neighbours using angular distance and euclidean distance respectively. Each table contains neighbours from ANNOY and Brute Force K-NN.

Using angular distance, all top 10 neighbours are identical, yielding a jaccard similarity of 1.0 and a spearman correlation of 1.0. This indicates perfect symmetry in both the neighbour set and their ranking.

Using Euclidean distance, 9 out of 10 neighbours are identical, yielding a jaccard similarity of 0.82. For the 9 neighbours that overlap, the spearman correlation remains at 1.0. This indicates that while one neighbour differs, the ranking of the remaining neighbours is preserved.

Overall, ANNOY closely approximates Brute Force K-NN, with angular distance providing slightly more consistent neighbour retrieval.

4.2 Dataset 2: MIMIC-III (Clinical) Dataset

MIMIC-III Clinical Database is large, freely-available database comprising deindentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. It includes information such as demographics, vital sign measurements made at the bedside (~1 data point per hour), laboratory test results, procedures, medications, caregiver notes, imaging reports and mortality (including post-hospital discharge). For the purpose of this study, we will be using pivoted MIMIC-III for simplicity.

Pivoted MIMIC-III contains fewer features of HADM_ID, CHARTTIME, DBP, HR, MBP, O2 Saturation, RR and SBP.

4.2.1 Dataset Cleaning and Preprocessing

The dataset contains 287594 samples with 7 numerical features and 1 temporal feature. It does not include categorical diagnostic labels that can be used for supervised evaluation unlike the previous breast cancer dataset. Therefore, it is more suitable for unsupervised similarity search, where the goal is to retrieve patients with similar vitals or laboratory results. However, as it is unsupervised, there is no ground truth that we can rely on for calculation of accuracy scores.

Remove Irrelevant Columns

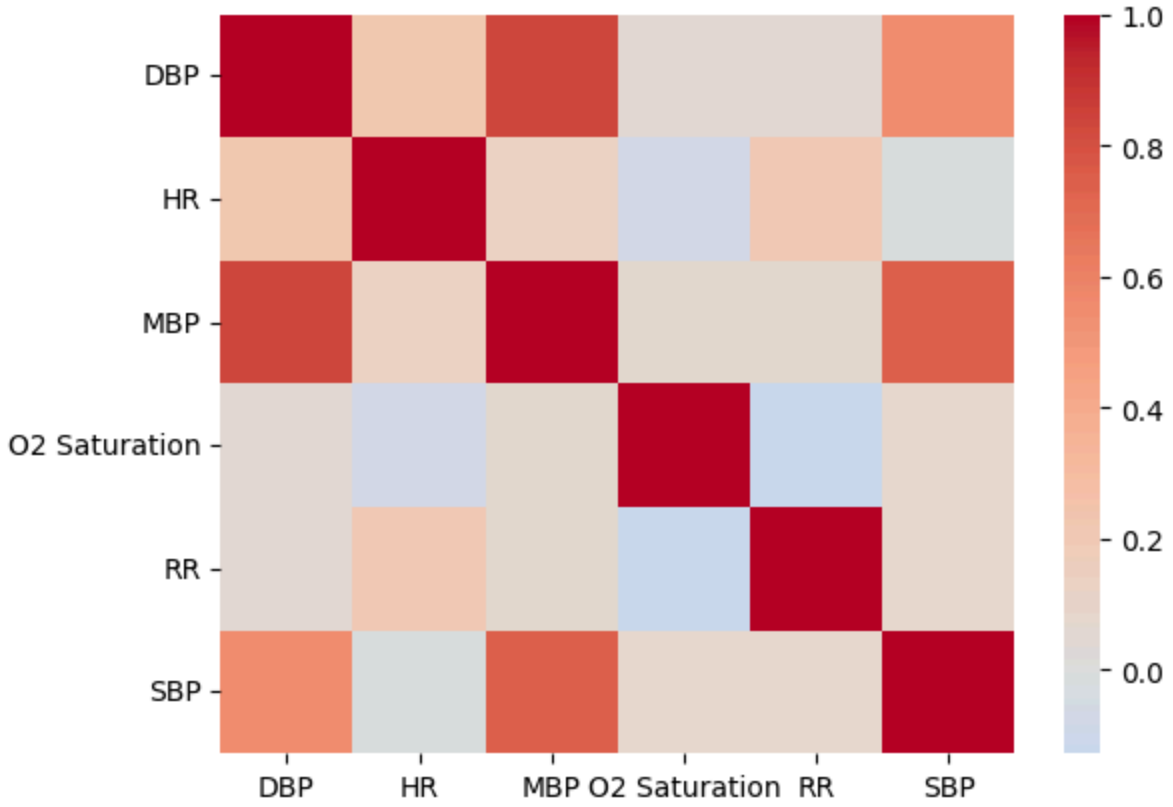
Columns ‘HADM_ID” is dropped as it does not carry meaningful information for our analysis.

Feature Scaling: Standardisation

The data is standardized using Standardscaler() so features are scaled to follow standard normal distribution. This ensures that all features contribute equally to distance calculations.

Correlation Analysis

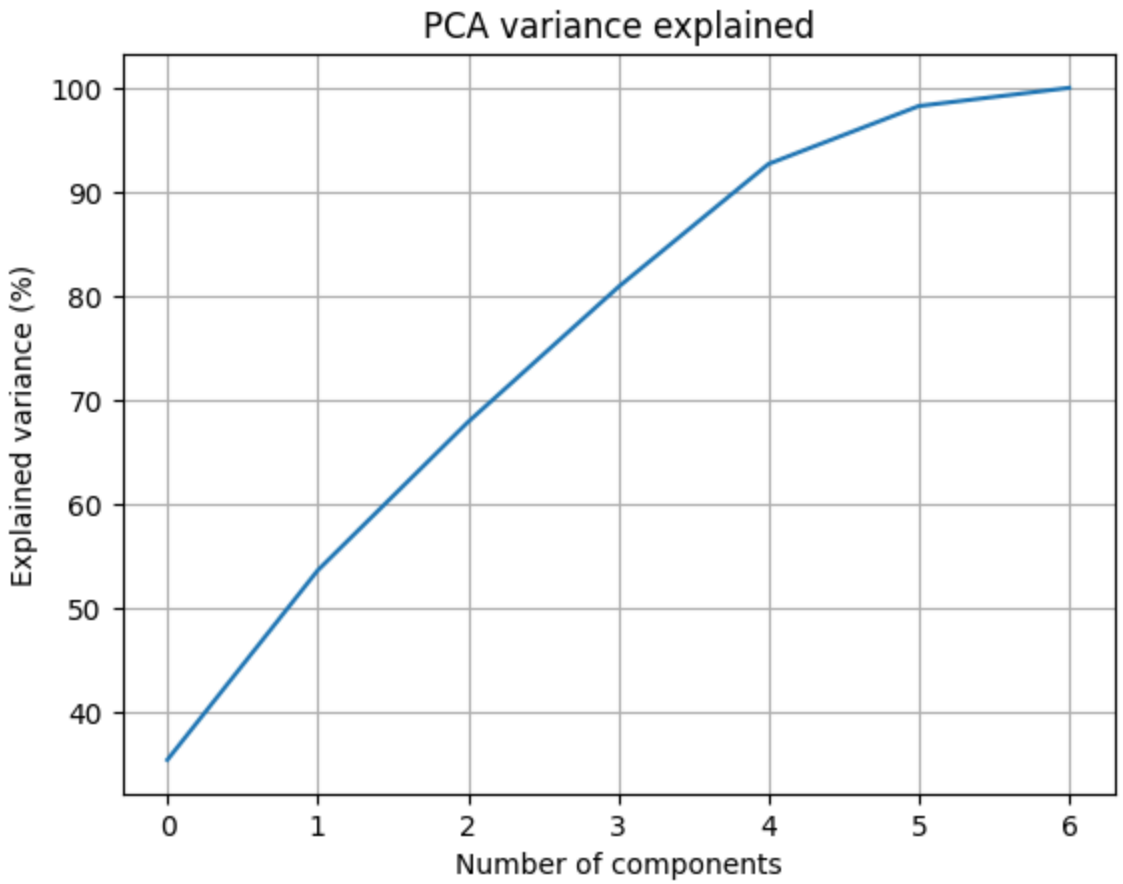
We can investigate the correlation between our dependent variable (diagnosis) and all predictors using a correlation plot. corr() helps to compute the Pearson correlation coefficients between all pairs of variables in our dataset.



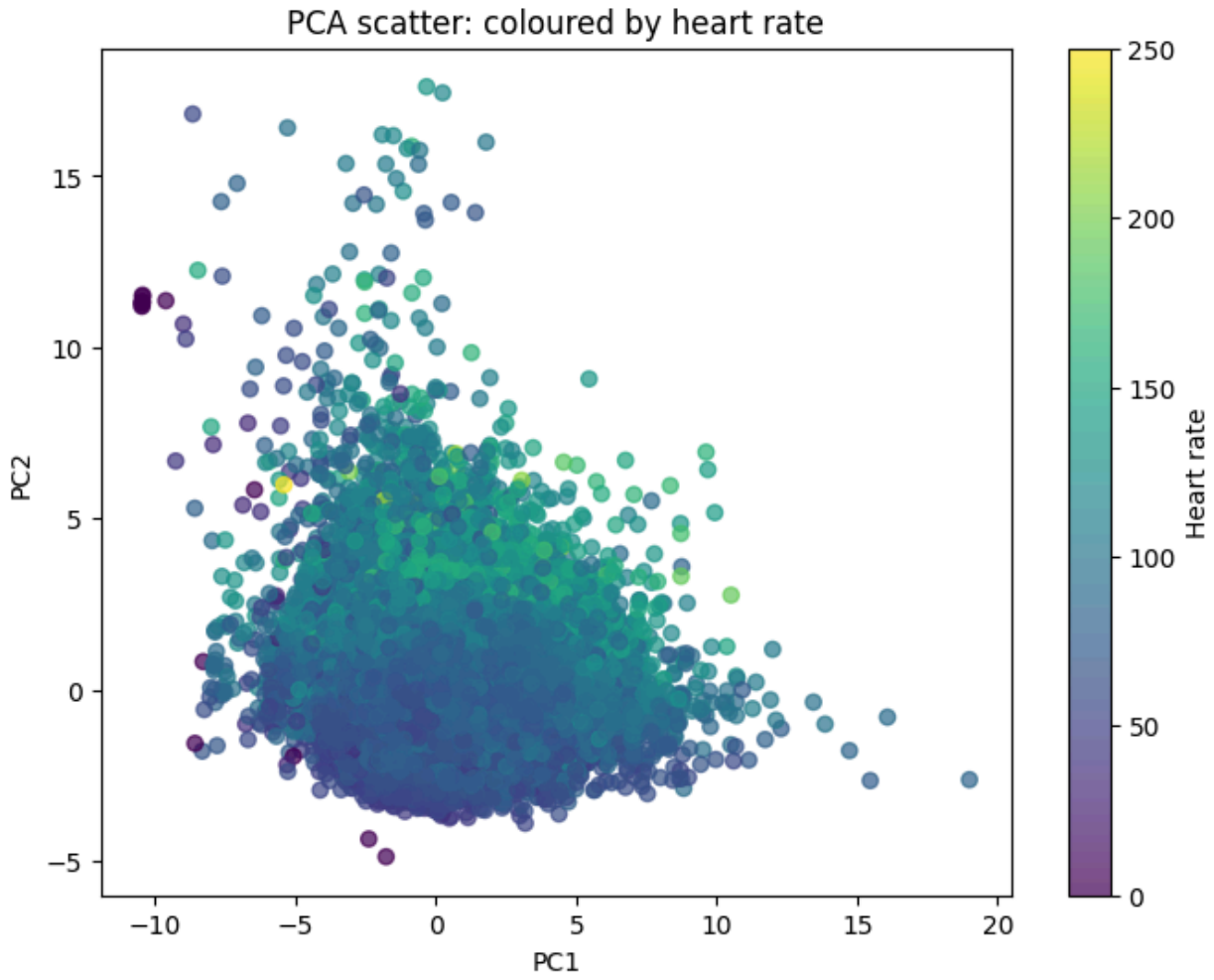
From the above figure, we can see that the strongest correlation is between MBP and DBP. While SBP shows some correlation with DBP, rest does not show strong correlations.

Principle Component Analysis (PCA)

PCA is a dimensionality reduction technique that allows us to transform the dataset into a set of principal components that retain most of the original information. It reduces computational complexity due to a smaller feature space and better visualisation of the variations. The components are ordered in a descending order, by the amount of variance they can capture.



PCA variance for MIMIC-III seems more linear in nature compared to PCA variance of Breast Cancer Dataset. Here, the variance explained increases to ~70% with 2 components, ~90% with 4, ~98% with 5. If ~95% were to be achieved, 4-5 components seem to be enough to explain most of the variances.



For 2D PCA visualisation, we are not able to colour the data points with distinct diagnosis as it is missing from MIMIC-III dataset. Hence, "HR" was chosen as the colour for visualisation. This is in contrast to Breast Cancer as there was ground truth of diagnosis to rely on for empirical truth.

```
# Dimensionality reduction (keeping ~95% variance to speed up search)
pca = PCA(n_components=0.95, svd_solver='full')
X_vec_mimic = pca.fit_transform(X_scaled)
X_vec_mimic.shape
```

```
(287594, 6)
```

In the case of the pivoted MIMIC-III dataset, we have to take all 6 features in order to get ~95% of the variance explained. However, as the number of features are small, it should still be quick in query retrieval.

4.2.2 Method 1: Brute Force K-NN Retrieval With Cosine/Euclidean

Background Information

(Refer to 4.1.2)

Evaluation of Algorithm Efficacy before Implementation

Append Cluster to each Point

For evaluation of K-NN Retrieval, due to the lack of target variable ('ground truths'), we have implemented k-means to attach pseudo clusters to each of the data points. For the purpose of this search, we used 5 clusters and it is adjustable according to the datasets.

```
n_clusters = 5  # adjust if needed
kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_scaled)
```

Implement Brute Force K-NN on Dataset

```
# Brute Force k-NN function
def brute_force_knn(X, q_index, k=10, metric="cosine"):
    q = X[q_index]
    d = pairwise_distances(q[None, :], X, metric=metric)[0]
    # exclude self
    order = np.argsort(d)
    order = order[order != q_index][:k]
    return order, d[order]
```

For a query sample at index, q_index, we will compute all pairwise distances between the query and all other data points in the dataset, X, using the chosen distance metric (Cosine/Euclidean). The distances will be sorted in ascending order and the indexes of the top k neighbours will be returned.

50 random patients were queried and used to calculate Precision@k and nDCG@k using the labels appended by k-means.

	Metric	Brute-Force Cosine	Brute-Force Euclidean
0	Average Precision@10	0.854	0.896
1	Average NDCG@10	0.944	0.959

From the above output, we can see that both distance metrics performed strongly in terms of retrieval evaluation metrics even though euclidean performed slightly better than cosine. This is in contrast to the previous dataset where cosine performed better than euclidean.

P@10=0.854 using cosine distance and P@10=0.896 using euclidean distance. On average, 85.4% of the top 10 neighbours found using cosine distance share the same diagnosis label as the query, compared to 89.6% of the top 10 neighbours found using euclidean distance. Cosine distance is slightly worse when retrieving neighbours with the same diagnosis.

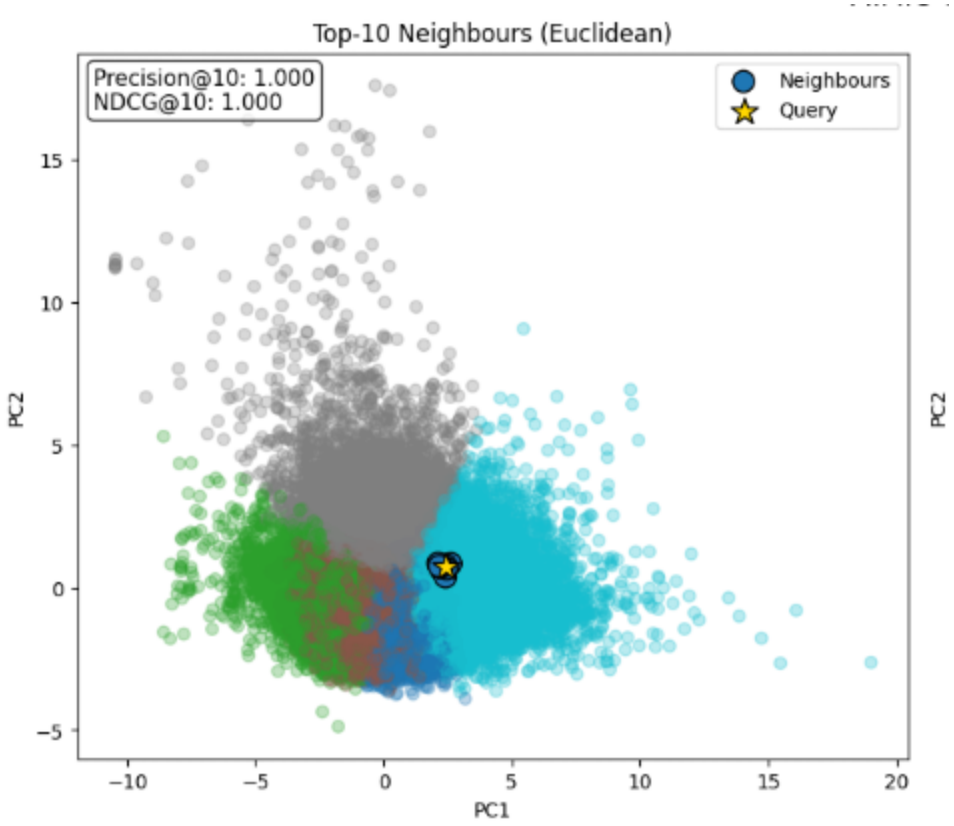
nDCG@10=0.944 using cosine distance and nDCG@10=0.959 using euclidean distance. Cosine distance retrieves relevant neighbours earlier in order than euclidean distance. This indicates that euclidean distance provides an overall better neighbour ranking quality.

Since euclidean distance performed better than cosine distance, it indicates that absolute magnitude between tumour features is more important than relative similarity of MIMIC-III features. However, these results have to be taken with a grain of salt as the results are calculated with arbitrary clusters assigned by k-means and are not the absolute truth.

Visualise Query Point and Neighbouring Point on Scatter Plots

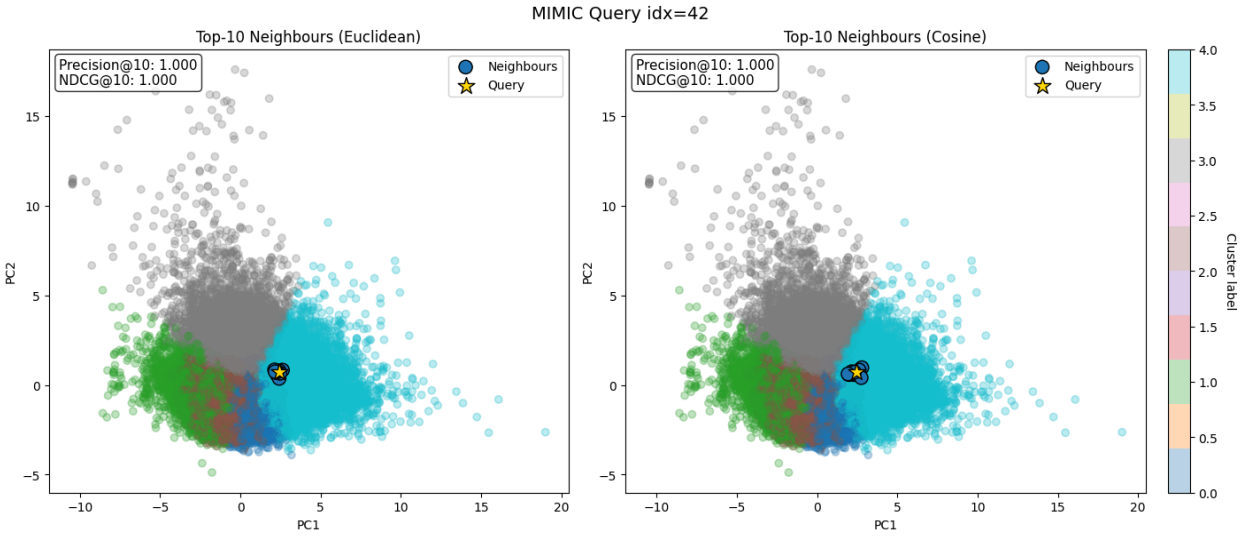
q_index=0 is set as the query point and will be marked with a “gold star”. The top k most similar points are marked with a “bigger blue dot”. All the other points are marked with their label colour taken from Matplotlib’s Tableau 10 colour palette using “smaller dots”. Different coloured dots represent the different clusters assigned by the k-means algorithm.

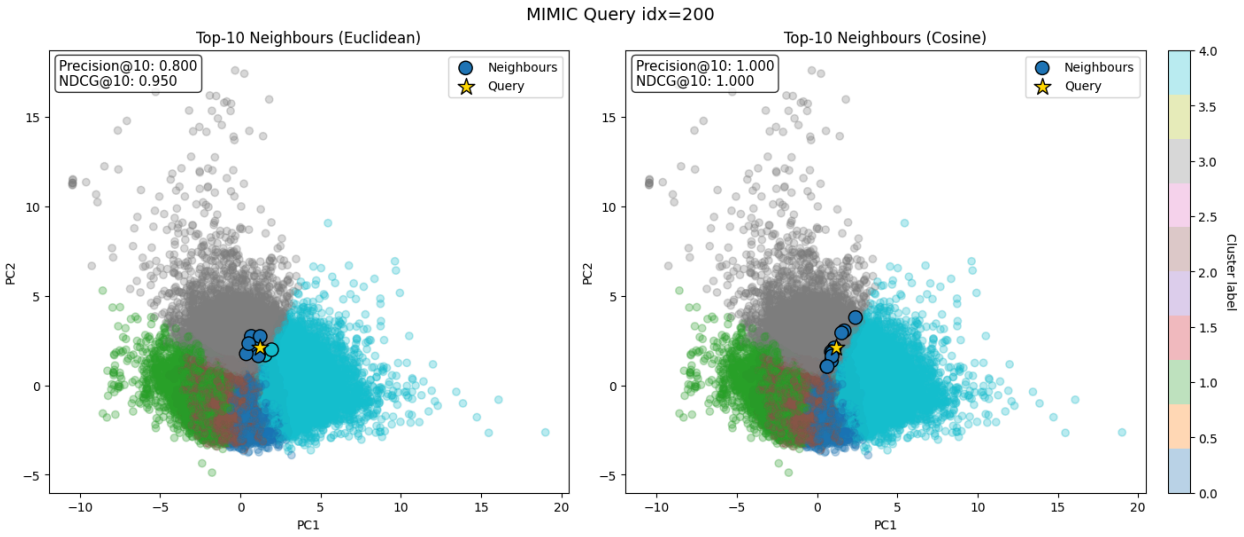
```
pca2 = PCA(n_components=2, random_state=42)
mimic_2d = pca2.fit_transform(X_scaled)

def plot_query_neighbors_side_by_side(q_index, k=10):
    metrics = ["euclidean", "cosine"]

    fig, axes = plt.subplots(1, 2, figsize=(14, 6), constrained_layout=True)

    for ax, metric in zip(axes, metrics):
        # Retrieve neighbours
        nbr_idx, nbr_dist = brute_force_knn(X_scaled, q_index, k=k, metric=metric)
        y_query = cluster_labels[q_index]
        y_neigh = cluster_labels[nbr_idx]
        relevance = (y_neigh == y_query).astype(int)

        # Compute Precision@k and NDCG@k
        prec = precision_at_k(y_query, y_neigh, k)
        ndcg = ndcg_at_k(relevance, k)

        # Plot all patients faintly by cluster
        scatter = ax.scatter(
            mimic_2d[:, 0], mimic_2d[:, 1],
            c=cluster_labels, cmap="tab10", alpha=0.3
        )

        # Plot neighbours
        ax.scatter(
            mimic_2d[nbr_idx, 0], mimic_2d[nbr_idx, 1],
            edgecolor="black", c=cluster_labels[nbr_idx],
            cmap="tab10", s=120, marker="o", label="Neighbours"
        )

       # Plot query point
       ax.scatter(
           mimic_2d[q_index, 0], mimic_2d[q_index, 1],
           c="gold", edgecolor="black",
           s=200, marker="*", label="Query"
       )

       # Titles, labels, and text annotations
       ax.set_title(f"Top-{k} Neighbours ({metric.capitalize()})", fontsize=12)
       ax.set_xlabel("PC1")
       ax.set_ylabel("PC2")
       ax.legend(loc="upper right")

       # Add quantitative metrics as text box
       textstr = f"Precision@{k}: {prec:.3f}\nNDCG@{k}: {ndcg:.3f}"
       ax.text(
           0.02, 0.98, textstr,
           transform=ax.transAxes, fontsize=11,
           verticalalignment="top", bbox=dict(boxstyle="round,pad=0.3",
                                              facecolor="white", alpha=0.8)
       )

   # Shared colorbar
   cbar = fig.colorbar(scatter, ax=axes, orientation="vertical", fraction=0.025, pad=0.02)
   cbar.set_label("Cluster label", rotation=270, labelpad=15)

   fig.suptitle(f"MIMIC Query idx={q_index}", fontsize=14)
   plt.show()
```



Experimental Analysis

To better compare the two distance metrics, we have created 2 side-by-side plots to represent euclidean distance on the left and cosine distance on the right for the same query point. Two query points were selected for analysis: a normal query point (q_index=42) and a point near edge of a cluster (q_index=200).





Here, even though the same query (data point) was selected, both of them produce different results based on the calculation method of their distances.

For the first figure, when a normal datapoint inside a cluster is chosen, both distance methods are able to retrieve data points within the same cluster. This shows that they are both able to perform well under normal circumstances.

However, when a point near the edge of a cluster is chosen (query_index=200), the two methods return different results. While cosine was able to deliver points from the same cluster consistently, euclidean retrieved 2 data points out of the cluster and from the other cluster. This will show up as error and be reflected in P@10 as lower score for cosine. On the other hand, when we take a look at the graph itself, we can see that the results returned by euclidean seem to more closely reflect results of similarity search compared to cosine.

4.2.3 Method 2: Approximate Nearest Neighbours (ANN) Search

Background Information

(Refer to 4.1.3)

Evaluation of Algorithm Efficacy before Implementation

Similar to Brute Force K-NN, we will be assessing the efficiency of the algorithm using Retrieval Evaluation Metrics (Precision@k and NDCG@k) and build time.

```
Angular (≈ Cosine) | Build time = 38.248s | P@10 = 0.875 | NDCG@10 = 0.956
Euclidean          | Build time = 30.050s | P@10 = 0.916 | NDCG@10 = 0.967
```

From the output, we can see that both distance metrics achieve great results in terms of retrieval evaluation metrics with euclidean distance performing slightly better than angular distance.

P@10=0.875 using angular distance and P@10=0.916 using euclidean distance. On average, 87.5% of the top 10 neighbours found using angular distance share the same diagnosis label as the query, compared to 91.6% of the top 10 neighbours found using euclidean distance. Euclidean distance is slightly better, suggesting that the absolute magnitude of the clinical features is more critical for determining patient similarity in this dataset than their relative pattern. This aligns with clinical intuition, where the precise values of vitals are often directly significant.

nDCG@10=0.956 using angular distance and nDCG@10=0.967 using euclidean distance. Euclidean distance not only retrieves relevant neighbours more accurately but also rank them better than Angular distance.

As for the build time, angular distance uses 38.248s while euclidean distance uses 30.050s. Angular distance takes slightly longer due to computational complexity.

Implement ANN search on Dataset

Step 1: Preprocessing to Build Annoy Index

To initialise the Annoy index, we use X_scaled.shape[1] to specify the number of dimensions of each vector. Next, we will add all the items from the dataset into the index. To do so, each feature vector in X_scaled is converted to a Python list and added to the index. The feature vectors are assigned an unique integer ID corresponding to its position in the dataset. Lastly, we build the index using ann.build(n_trees) where n_trees=50. 50 trees are constructed internally. Increasing the number of trees increases the accuracy of the search by exploring more partitions of the hyperspace. However, this also increases the time it takes to build the index and more memory is required. Once the index is built, we can now efficiently query it for nearest neighbors of any vector without scanning the entire dataset.

```
for ax, metric in zip(axes, ["angular", "euclidean"]):
    # --- Build Annoy index ---
    f = X_scaled.shape[1]
    ann = AnnoyIndex(f, metric)
    for i, v in enumerate(X_scaled):
        ann.add_item(i, v.tolist())
    ann.build(n_trees)
```

Step 2: Query Nearest Neighbours

To find the nearest neighbours, we query the annoy index using the feature vector represented by X_scaled[q_index]. We return k+1 neighbours instead of k neighbours because the query itself may appear as the closest neighbour, which we will be removing in the next step.

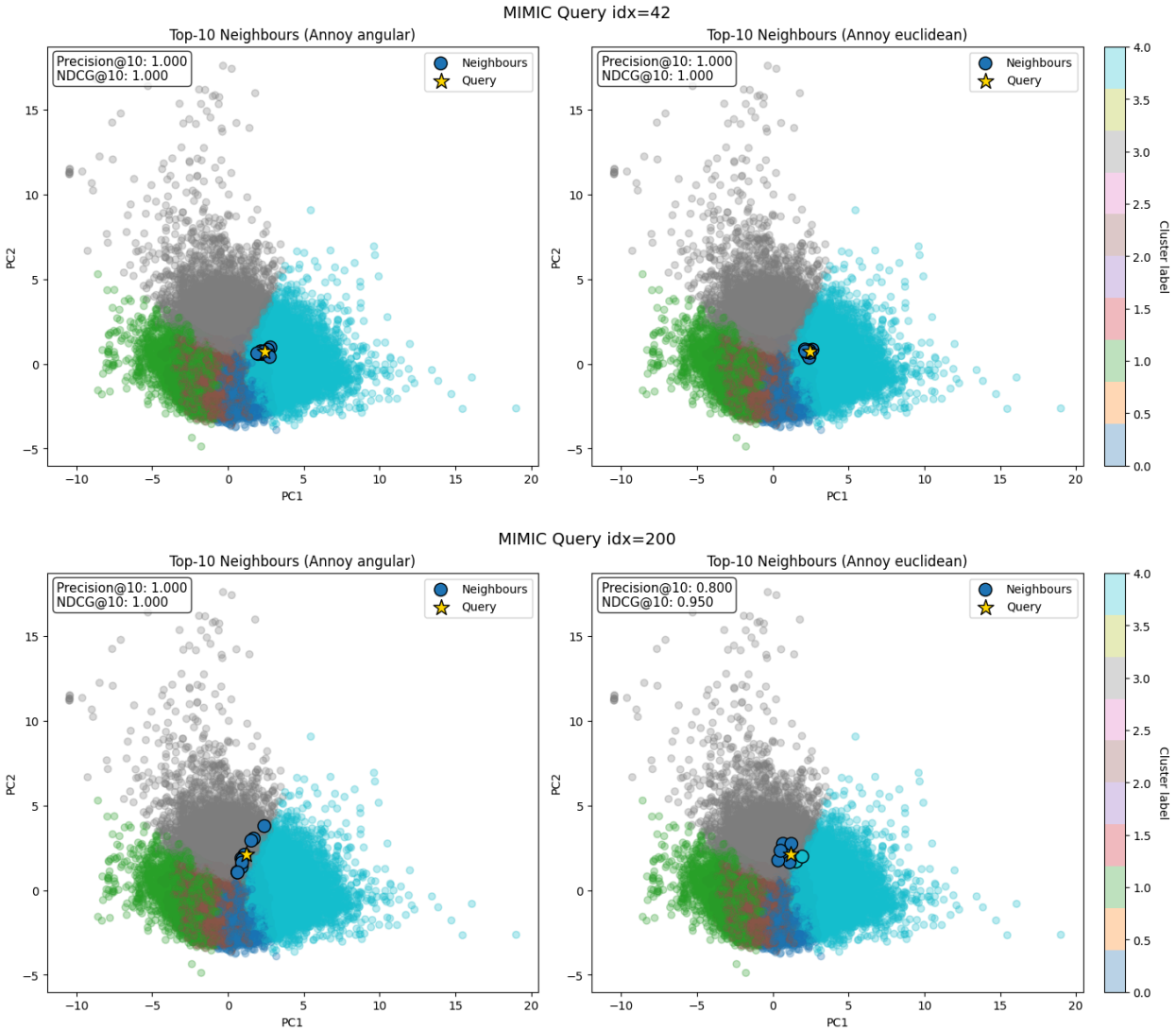
Step 3: Remove Query from Results

```
if idxs[0] == q_index:
    idxs = idxs[1:k+1]
else:
    idxs = idxs[:k]
```

The query is removed if itself is included.

Step 4: Visualise Query Point and Neighbouring Point on Scatter Plots

Similar to Brute Force K-NN Retrieval, we will be using a 2x2 plot to visualise the results.



From the output, we can observe that ANN search produces good results when a normal query point within a cluster is chosen (q_index=42). Both angular and euclidean distance achieved perfect retrieval scores, with all 10 neighbours belonging to the same K-Means cluster as the query. However, when a boundary query point (q_index=200) is selected, angular distance successfully retrieves all neighbours from the same cluster, while euclidean distance retrieves 8 correct neighbours and 2 neighbours from adjacent clusters. Angular distance outperformed euclidean using ANN search for this boundary case.

Experimental Results

This proves again that ANN search is a robust algorithm when querying a normal point, regardless of the distance metrics used. However, when querying an outlier or boundary point, angular distance will be a better choice for improving neighbours retrievals.

Comparison Between Brute Force K-NN and ANN Search

Again, we will evaluate the neighbours returned by Brute Force K-NN and ANN Search by using Jacquard Index and Spearman's Rank Order Coefficient.

Implementation of Jacquard Index and Spearman's Rank Order Coefficient

```
# Compare using Angular (≈ Cosine)
summary_ang, table_ang = neighbour_overlap_report_ann_mimic(
    X_scaled, cluster_labels, q_index=42, k=10, metric="angular", n_trees=50
)
print(summary_ang)
display(table_ang.head(20))

# Compare using Euclidean
summary_euc, table_euc = neighbour_overlap_report_ann_mimic(
    X_scaled, cluster_labels, q_index=42, k=10, metric="euclidean", n_trees=50
)
print(summary_euc)
display(table_euc.head(20))
```

	idx	method	rank	distance	cluster_label	is_common
10	129320	annoy	1	0.102982	4	True
11	22	annoy	2	0.114928	4	True
12	37984	annoy	3	0.117802	4	True
13	5707	annoy	4	0.118396	4	True
14	5686	annoy	5	0.128765	4	True
15	6176	annoy	6	0.129964	4	True
16	11840	annoy	7	0.133727	4	True
17	22991	annoy	8	0.135224	4	True
18	20346	annoy	9	0.135247	4	True
19	43254	annoy	10	0.136785	4	True
0	129320	brute_force	1	0.005303	4	True
1	22	brute_force	2	0.006604	4	True
2	37984	brute_force	3	0.006939	4	True
3	5707	brute_force	4	0.007009	4	True
4	5686	brute_force	5	0.008290	4	True
5	6176	brute_force	6	0.008445	4	True
6	11840	brute_force	7	0.008941	4	True
7	22991	brute_force	8	0.009143	4	True
8	20346	brute_force	9	0.009146	4	True
9	43254	brute_force	10	0.009355	4	True

	idx	method	rank	distance	cluster_label	is_common
10	37984	annoy	1	0.405233	4	True
11	11840	annoy	2	0.408150	4	True
12	14698	annoy	3	0.430689	4	True
13	5707	annoy	4	0.435364	4	True
14	41690	annoy	5	0.436036	4	True
15	20346	annoy	6	0.436829	4	True
16	22986	annoy	7	0.446919	4	True
17	5727	annoy	8	0.456323	4	True
18	23002	annoy	9	0.482610	4	True
19	6176	annoy	10	0.486610	4	True
0	37984	brute_force	1	0.405233	4	True
1	11840	brute_force	2	0.408150	4	True
2	14698	brute_force	3	0.430689	4	True
3	5707	brute_force	4	0.435364	4	True
4	41690	brute_force	5	0.436036	4	True
5	20346	brute_force	6	0.436829	4	True
6	22986	brute_force	7	0.446919	4	True
7	5727	brute_force	8	0.456323	4	True
8	23002	brute_force	9	0.482610	4	True
9	6176	brute_force	10	0.486610	4	True

The two summary tables here represent the ranking of neighbours using angular distance and euclidean distance respectively. Each table contains neighbours from ANNOY and Brute Force K-NN.

Using angular distance, all top 10 neighbours are identical, yielding a jaccard similarity of 1.0 and a spearman correlation of 1.0. This indicates perfect symmetry in both the neighbour set and their ranking, with all retrieved neighbours correctly belonging to cluster 4.

Similarly, using Euclidean distance, all 10 neighbours are identical, yielding a jaccard similarity of 1.0 and a spearman correlation of 1.0. This perfect alignment shows that both the neighbour composition and ranking order are preserved exactly.

Overall, ANNOY perfectly approximates Brute force K-NN for this clinical query point, with bioth distance metrics providing identical neighbour retrieval and ranking for patients within well-defined clusters in the MIMIC-III dataset.

5.0 Conclusion

Conclusion for Breast Cancer Dataset

	Method	Metric	Build Time	Avg Query Time	P@10	nDCG@10
0	Brute-force kNN	Cosine	-	3.190 ms	0.962	0.988
1	Brute-force kNN	Euclidean	-	1.864 ms	0.942	0.981
2	Annoy (50 trees)	Angular	0.142s	<1 ms	0.964	0.988
3	Annoy (50 trees)	Euclidean	0.054s	<1 ms	0.943	0.981

For the Breast Cancer Wisconsin (Diagnostic) Dataset, both Brute Force K-NN and Annoy (ANN) achieved very high retrieval quality, with Precision@10 and nDCG@10 scores consistently above 0.96. This indicates that the PCA-reduced feature space effectively captures the underlying patterns that distinguish malignant and benign tumours.

From the table above, we can see that Cosine/Angular distance metrics slightly outperformed Euclidean distance across both methods. This better performance can be attributed to the nature of the data; Cosine distance focuses on the directional pattern of tumour characteristics (e.g. the relationship between radius, texture, and concavity) rather than their absolute magnitudes. This makes it more robust for identifying tumours with similar malignant or benign profiles, regardless of their overall size.

In terms of efficiency, Brute Force K-NN search required approximately 2-3 ms per query. While this is acceptable for a dataset of this size (569 samples), its linear time complexity presents scalability limitations. Annoy addressed this by introducing a minimal upfront build time (0.05-0.14 seconds), which allowed for a near-instantaneous query time (<1 ms). This makes Annoy highly scalable even for smaller datasets.

The trade-off is clear: for a small, well-structured dataset for Breast Cancer Wisconsin, Brute Force K-NN is a simple and accurate solution. However, Annoy proves to be a better method even here, delivering equivalent accuracy with query speeds orders of magnitude faster, hence making it the more robust and scalable choice for production environments.

Conclusion for MIMIC-III Clinical Dataset

	Method	Metric	Build Time	Avg Query Time	P@10	nDCG@10
0	Brute-force kNN	Cosine	-	55.301 ms	0.926	0.971
1	Brute-force kNN	Euclidean	-	35.754 ms	0.932	0.966
2	Annoy (50 trees)	Angular (Cosine)	43.053s	0.554 ms	0.872	0.949
3	Annoy (50 trees)	Euclidean	25.002s	0.621 ms	0.912	0.972

For the MIMIC-III Clinical Dataset, we focused on retrieving patients with similar physiological states based on their vital signs and lab values. Both Brute Force K-NN and Annoy achieved high retrieval quality, with Precision@10 and nDCG@10 scores consistently above 0.87. This was evaluated using KMeans cluster pseudo-labels, which confirmed that the retrieved neighbors are physiologically similar.

In contrast to the Breast Cancer Dataset, Euclidean distance slightly outperformed Cosine/Angular distance. This key finding suggests that for clinical vitals, the absolute magnitude of values (e.g. a systolic BP of 180mmHg versus 120mmHg) carries critical, meaningful information for assessing patient similarity. A high magnitude in certain vitals can be a direct indicator of severity, which Euclidean distance is better equipped to capture. Despite being an approximate method, Annoy preserved most of the retrieval quality, with only a small percentage gap to the exact Brute Force K-NN method.

The computational trade-off was the most significant finding for this large-scale dataset. Brute Force K-NN retrieval required 35-55 ms per query, a cost that grows linearly and becomes unsustainable for real-time applications on large datasets. Annoy addressed this with a one-time build cost of 25-43 seconds, which enabled it to achieve sustained sub-millisecond query times thereafter. This makes tree-based approximate methods like Annoy not just beneficial but essential for large-scale clinical data.

The overall trade-off demonstrates that for large, high-dimensional datasets like MIMIC-III, Brute Force K-NN is computationally impractical even though it is accurate. Annoy offers an efficient solution: a one-time indexing investment enables persistent, ultra-fast query capabilities with a minimal and acceptable loss in accuracy, making it the definitive choice for scalable similarity search in healthcare.

Conclusion for Brute Force K-NN against the 2 datasets

Across both datasets, Brute Force K-NN served as the crucial baseline for accuracy, achieving the highest possible retrieval quality. Its performance is adequate for static, small- to

medium-sized datasets where computational cost is not a primary concern. Its main strength lies in its simplicity and guaranteed correct results, making it an invaluable tool for prototyping and validation. However, its linear time complexity (O(n)) is its fundamental weakness, rendering it unsuitable for large-scale, latency-sensitive applications.

Conclusion for ANN using Annoy library against the 2 datasets

We observed that Annoy serves as a highly scalable solution for similarity search. Its core strength is transforming an O(n) search problem into an O(log n) search using random projection trees. This enabled it to achieve query times of less than 1 millisecond on both datasets. For smaller Breast Cancer Dataset, it matched Brute Force K-NN accuracy, while for large MIMIC-III Dataset, it maintained high accuracy with a negligible performance gap. The requirement for a one-time index is a minor drawback that is heavily outweighed by the persistent query speed gains. Therefore, for any application requiring scalable and fast similarity search, ANN methods like Annoy are not just advantageous but necessary.

In summary, this project demonstrates the trade-off between exact and approximate similarity search depends directly on the dataset scale and latency requirements. Brute Force K-NN provides an accuracy baseline but scales poorly. On the other hand, Annoy offers a critical speed advantage with minimal accuracy loss, making it the better choice for practical, large-scale applications. Furthermore, the optimal distance metric depends on the context: Cosine distance is better for pattern-based similarity (e.g. tumour profiles), while Euclidean distance is more effective when absolute magnitudes are clinically significant (e.g. patient vitals).

References

1. *What is Similarity Search in Data Mining*. (2024, April 2). MYSCALE. https://www.myscale.com/blog/understanding-similarity-search-in-data-mining/

2. UCL Machine Learning & Ovsen. (2019). *Breast Cancer Wisconsin (Diagnostic) Data Set.* Kaggle. https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data/data

3. Johnson A., Pollard T. & Mark R. (2019, September 30). *MIMIC-III Clinical Database*. PhysioNet. https://physionet.org/content/mimiciii/1.4/

4. *What is Similarity Search & How Does it work?* (2024, May 2023). truefoundry. https://www.truefoundry.com/blog/similarity-search

5. *What is Principal Component Analysis (PCA)?* (n.d.). IBM. https://www.ibm.com/think/topics/principal-component-analysis

6. BioTuring Team. (2018, June 15). *Principal Component Analysis explained simply*. Medium. https://bioturing.medium.com/principal-component-analysis-explained-simply-894e8f6f4bfb

7. Kavlakoglu E. (n.d.). *What is the k-nearest neighbours (KNN) algorithm?* IBM. https://www.ibm.com/think/topics/knn

8. Shkhanukova M. (2023, Mar 4). *Cosine similarity and cosine distance.* Medium https://medium.com/@milana.shxanukova15/cosine-distance-and-cosine-similarity-a5da0e4d9ded

9. *Evaluation Metrics for Retrieval-Augmented Generation (RAG) Systems.* (2025, Oct 9). Geeksforgeeks. https://www.geeksforgeeks.org/nlp/evaluation-metrics-for-retrieval-augmented-generation-rag-systems/

10. Liu F. (2023, May 25). *Approximate Nearest Neighbors Oh Yeah (Annoy).* zilliz. https://zilliz.com/learn/approximate-nearest-neighbor-oh-yeah-ANNOY

11. Jadeja M. (2022, November 18). *Jaccard Similarity Made Simple: A Beginner's Guide to Data Comparison.* Medium. https://mayurdhvajsinhjadeja.medium.com/jaccard-similarity-34e2c15fb524

12. Numiqo. (2023, April 5). *Spearman Rank Correlation [Simply explained]* [Video]. YouTube. https://www.youtube.com/watch?v=XV_W1w4Nwoc