

הערכה חלופית – תכונות מונחה עצמים

2026 סמסטר א' – ת.ז 322450305

בפרויקט זה יצרנו אפליקציה לחקר אינטראקטיבי של ייצוגים לטנטיים, שהם וקטוריים המייצגים אובייקטים (כגון: מילים, תמונות, רצפי DNA) במרחב רב-ממדי. המטרה המרכזית של המערכת היא לאפשר חקירה ויזואלית ומתמטית של קשרים סמנטיים בין אובייקטים, תוך הפרדה ברורה בין:

- מנוע חישוב מתמטי.
- מודל נתונים.
- שכבת שירותים (Use Cases).
- שכבת תצוגה.
- שכבת UI גרפי – מומשה אצלנו ע"י JavaFX.

המערכת תומכת הן בתצוגה דו-ממדי והן בתצוגה תלת-ממדית, ומאפשרת ביצוע חישובים סמנטיים כגון מרחקים, חיפוש שכנים קרובים, הקרנות מותאמות ואריתמטיקה וקטורית.

החישובים הסמנטיים (מרחקים, שכנים, אריתמטיקה וקטורית) מתבצעים תמיד במרחב הווקטורי המלא (FULL), בעוד שהיצוג PCA (במימד קטן יותר) משמש לייצוג ויזואלי בלבד. לכן יצרנו מחלקה שתרכז את כל הייצוגים הקיימים במערכת.

העיקרון המנחה בפרויקט הוא יצירת תשתית מופשטת המסוגלת לייצג וקטורים של כל ישות נדרשת, לא רק מילים, אלא גם תמונות, רצפי DNA או כל אובייקט שניתן להטמעה למרחב מספרי.

במימוש שלנו, המערכת כוללת אפשרות להריץ את סקריפט הפיתוח שהנתון במטלה מתוך האפליקציה עצמה. בעת לחיצה על כפתור ייעודי ("Generate embeddings"), האפליקציה מחלצת את קובץ embedder.py מתוך משאבי המערכת, מריצה אותו כהליך חיצוני באמצעות PythonScriptRunner, ומייצרת מחדש את קובצי ה-pca_vectors.json ו-full_vectors.json. בתיקיית data. לאחר יצירתם, הקבצים נטענים באופן אוטומטי לזיכרון והמערכת מתעדכנת מיידית. כמו כן ניתן לבחור לטעון קובץ JSON ידנית.

לאחר טעינת הקבצים, המשתמש בוחר שיטת ייצוג עבור התצוגה, בוחר צירים לתצוגה בהתאם למימד התצוגה, ואז יכול לבחור מילה על המסך/ברשימה כדי לקבל שכנים קרובים, להריץ Projection בין שני עוגנים, לבצע אריתמטיקה בין וקטורים ומילים ולהציג את מסלול החישוב. כך מתקבלת חקירה אינטראקטיבית של קשרים סמנטיים – גם ויזואלית וגם חישובית.

הארכיטקטורה מפרידה לחלוטין בין שכבת החישוב לשכבת התצוגה. כל הלוגיקה המתמטית (מרחקים, שכנים, הקרנות, אריתמטיקה וקטורית) ממומשת בשכבת האפליקציה, בעוד ה-UI מתקשר עם שכבת האפליקציה דרך ממשק יחיד (ExplorerUseCases). כל תוצאות החישובים חוזרות כ-DTOs (Data Transfer Object) כמו NeighborView, ProjectionScoreView, LabResultView שמותאמים לתצוגה, כך שה-UI אינו תלוי באופן המימוש של מחלקות הפנימיות במודל.

חישובי המרחק ממומשים בתבנית Strategy (פולימורפיזם), כך שניתן לבחור בזמן ריצה בין Cosine Distance ל-Euclidean Distance ולהוסיף מטריקות חדשות בלי לשנות את מבלי לשנות את הקוד הקיים (Open/Closed Principle).

המעבר מתצוגת דו-ממדית לתלת-ממדית ללא דרש שינוי במודל המתמטי, אלא רק הוספת View חדש. כלומר גם כאן שינוי תצוגה לא אמור להשפיע על לוגיקת בליבה.

בחרתי בארכיטקטורה שכבתית זו מתוך מטרה לייצר מערכת מודולרית, גמישה וניתנת להרחבה. רציתי להבטיח שהלוגיקה החישובית, ניהול הנתונים והתצוגה הגרפית יהיו מופרדים לחלוטין, כך שניתן יהיה לשנות או להרחיב כל רכיב באופן עצמאי. תכנון זה מאפשר הוספת מטריקות חדשות, ייצוגים נוספים או סוגי תצוגה שונים – מבלי לפגוע בקוד הקיים. בכך המערכת מממשת בפועל עקרונות מרכזיים בתכנות מונחה עצמים כגון הפרדת אחריות, פתיחות להרחבה וסגירות לשינוי, ושימוש בפולימורפיזם לצורך יצירת גמישות תכנונית.



הפרויקט מגובה במערך בדיקות יחידה ובדיקות קבלה (JUnit), שמוודא נכונות מתמטית של הווקטורים, טעינת הנתונים, חישובי שכנים, Vector Arithmetic ו-Projection.

נסקור בקצרה את השכבות השונות שבנינו בפרויקט, ואת המחלקות השונות בכל שכבה:

שכבת IO

אחראית על טעינת הנתונים החיצוניים. החל מהרצת סקריפט Python שיוצר קבצי JSON, באמצעות ProcessBuilder, ולאחר מכאן קריאת הקבצים ובניית האובייקטים של המודל מתוך הנתונים הגולמיים. השכבה מבודדת לחלוטין מהלוגיקה המתמטית ומה-UI, כך שניתן להחליף מקור נתונים בעתיד מבלי לשנות את שאר המערכת.

Representation

מחלקה שמייצגת סוג ייצוג (Representation) כ-Flyweight (מופע ייחודי לכל ייצוג כך שכל שם קנוני קיים מופע יחיד במערכת).

RepresentationSource

ממשק המגדיר מקור טעינה לייצוג מסוים של אובייקט גנרי כלשהו (java generics), האחראי לספק מיפוי בלתי-ניתן לשינוי בין מזהים (של אובייקטים) לווקטורים. כל מימוש של הממשק יגדיר את מקור הנתונים ואופן שבו הוא יאחסן אותם בטעינתם.

JsonSource

מימוש של RepresentationSource שטוען embeddings מקובץ JSON באמצעות stream של JAVA, מאמת תקינות (מזהים/ממדים/כפילויות), ומחזיר Map בלתי ניתן לשינוי, עם cache אחרי טעינה ראשונה.

JsonFormat

אובייקט שיגדיר לנו את מבנה השדות בקובצי JSON שנטען (פעם יופיע לנו שדה ראשון של "WORD", אבל מחר נרצה "DNA").

PythonScriptRunner

מריצה סקריפט Python חיצוני באמצעות ProcessBuilder.

שכבת המודל

שכבת המודל מהווה את הליבה המתמטית והמבנית של הפרויקט. היא אחראית על ייצוג האובייקטים כווקטורים במרחב לטנטי ועל ביצוע חישובים. שכבה זו פועלת באופן בלתי תלוי בממשק המשתמש ובאופן טעינת הנתונים, ומגדירה את חוקי העבודה של המרחב הווקטורי עצמו.

במסגרת שכבה זו מוגדרים מבני הנתונים המרכזיים (כגון Vector ו-Embedding), וכן הפעולות המתמטיות בוקטור. כל לוגיקת החישוב מתבצעת כאן בלבד, בעוד ששכבות אחרות במערכת מסתמכות עליה דרך ממשקים ברורים ומופשטים.

תכנון זה מבטיח הפרדה בין אחריות חישובית לאחריות תצוגתית, תומך בגמישות ובהרחבה עתידית, ומאפשר בדיקות יחידה מנותקות מה-UI.

Vector

מחלקה בלתי-ניתנת לשינוי (immutable) המייצגת וקטור מספרי (מערך של double) ומכילה פעולות:

- חיבור
- חיסור
- נרמול
- חישובי dot product
- ממוצע (centroid)

רכיב הווקטור הוא מרכזי בפרויקט שלנו.

EmbeddingSingle

ממשק המייצג אובייקט סמנטי בודד (לדוגמה: מילה, במקרה דנן) בעלת מספר ייצוגים וקטוריים לפי סוג Representation.

EmbeddingGroup

ממשק המייצג אוסף ישויות EmbeddingSingle ומספק גישה מרוכזת לפי מזהה או ייצוג.

מחלקות

EmbeddingItem

מימוש בלתי ניתן לשינוי של EmbeddingSingle שמחזיק מפה מ-Representation ל-Vector עבור ישות אחת.

EmbeddingStorage

מימוש בלתי ניתן-לשינוי של EmbeddingGroup שמאחסן EmbeddingSingle בזיכרון ומוודא סט ייצוגים אחיד לכל הפריטים. וכך יוצר מאגר מרכזי בזיכרון התוכנית שלנו.

EmbeddingsAssembler

factory שמרכיב EmbeddingStorage ממספר מקורות ייצוגים (RepresentationSource), תוך איחוד לפי מזהים ואכיפת של סט מזהים (האובייקטים שאנו מייצגים, מילה או ID אחר) זהה בכל המקורות.

חישובי מרחק (Strategy Pattern)

המערכת תומכת במספר מטריקות מרחק:

- Euclidean Distance
- Cosine Distance

המטריקות ממומשות באמצעות ממשק משותף (DistanceMetric), מה שמאפשר הוספת מטריקה חדשה מבלי לשנות את שאר הקוד.

חישוב שכנים קרובים

המחלקה NearestNeighbors מחשבת את K השכנים הקרובים ביותר (תוך שימוש במטריקה שנבחרה) לוקטור יעד או אובייקט גנרי אותו אנו מייצגים.

החישוב מתבצע על המרחב הווקטורי המלא (לא על הממד המוקרן לתצוגה. כמובן שזה בבחירתו של מי שקורא למחלקה).

ניצור כאן אובייקט חדש שיעזור לנו: Neighbor שמייצג שכן קרוב כמזהה וערך מרחק מחושב.

חישובי שכנים מתבצעים באופן ישיר על כלל הווקטורים בזיכרון (linear scan), מתוך הנחה שמדובר בגודל נתונים מחקרי סביה. הארכיטקטורה מאפשרת בעתיד החלפה למבנה נתונים יעיל יותר (כגון KD-Tree או ANN Index) מבלי לשנות את שכבת ה-UI.

Projection System

CustomProjectionService

שירות הבונה ציר סמנטי משתי ישויות ומקרין עליו את כל ה-embeddings בקבוצה.
(לדוגמה: "עני" \leftarrow "עשיר").

ProjectionAxis

מייצג ציר חד-ממדי במרחב הווקטורי ומחשב מיקום וסטייה של וקטור ביחס אליו.

ProjectionScore

רשומת תוצאה המייצגת מזהה עם מיקומו על הציר, מרחקו האורתוגונלי וציון טוהר.

Vector Arithmetic Lab

מודול מתקדם המאפשר בניית ביטויים וקטוריים כגון: king - man + woman צפוי להניב תוצאה הקרובה ל-queen.

הרעיון הכללי

- מחשבים וקטור תוצאה מהביטוי.
- מאתרים את K השכנים הקרובים לוקטור התוצאה.
- מחזירים גם מסלול ביניים (במרחב התצוגה) כדי שה-UI יוכל להציג ויזואליזציה של שלבי החישוב.

VectorArithmeticLab

פסאז' שמחשב וקטור ומחזיר שכנים קרובים לתוצאה.

Term

מייצג איבר חתום בביטוי וקטורי (id+ או id-).

VectorExpression

מייצג ביטוי כשרשרת של איברים חתומים עם Builder נוח.

LabResult

תוצאת מעבדה הכוללת וקטור מחושב ורשימת שכנים קרובים.

VectorExpressionEvaluator

מחשב ביטוי וקטורי לוקטור תוצאה בייצוג נתון.

Subspace Grouping (Group Centroid Analysis)

מודול זה מרחיב את יכולות הניתוח בכך שהוא מאפשר עבודה על קבוצה של ישויות במקום על ישות בודדת. המשתמש יכול לבחור מספר מזהים (לדוגמה: מספר מילים), והמערכת מחשבת את הווקטור המרכזי שלהן (Centroid) במרחב הווקטורי המלא. גישה זו מאפשרת ניתוח סמנטי ברמת "תת-מרחב" (Subspace), כלומר חקר הקשרים המשותפים בין מספר מושגים בקבוצה ולא רק באופן פרטני. החישוב מתבצע כך:

- עבור כל מזהה נשלף הווקטור המלא (FULL representation).
- מבוצע חיבור של כלל הווקטורים.
- התוצאה מחולקת במספר האיברים לקבלת ממוצע וקטורי.
- לאחר מכן מתבצע חישוב K השכנים הקרובים ביותר לווקטור המרכזי באמצעות המטריקה שנבחרה.

החישוב מתבצע בשכבת השירותים (ExplorerApplicationService) על בסיס הייצוג המלא ובהתאם למטריקת המרחק שנבחרה, וה-UI מקבל את התוצאה כ- DTO ללא תלות בפרטי המימוש הפנימיים.

שכבת View

שכבה זו אחראית לייצוג נקודות במרחב – מבלי לבצע שום לוגיקה סמנטית, רק טרנספורמציה מתמטית לתצוגה:

- ViewPoint: ייצוג כללי של נקודה במרחב תצוגה במימד כלשהו.
 - Point_2D: מייצג נקודה דו-ממדית במרחב התצוגה.
 - Point_3D: מייצג נקודה תלת-ממדית במרחב התצוגה.
- LabeledPoint: עוטף נקודת תצוגה יחד עם מזהה ישות.
- PointCloud: אוסף נקודות מתויגות המוכן להצגה.
- ViewMode: ממשק המגדיר כיצד למפות וקטור למרחב תצוגה.
 - ViewMode2D: מיפוי דו-ממדי לפי שני רכיבי וקטור נבחרים.
 - ViewMode3D: מיפוי תלת-ממדי לפי שלושה רכיבי וקטור נבחרים.
- ViewSpace: מתרגם embeddings לענן נקודות בהתאם ל-ViewMode.

שכבת שירותים (Application Layer)

שכבה זו מתווכת בין שכבות המודל והפעולות לבין ה-UI (לא הצגה בלבד, אלא גם אינטראקציה של המשתמש):

ExplorerUseCases

ממשק שמגדיר את כל הפעולות שה-UI יכול לבצע: טעינה, תצוגה, שכנים, הקרנות וחישובי וקטורים. ה-UI עובד מולו בלבד, בלי להכיר מימושים פנימיים.

ExplorerApplicationService

המימוש בפועל של הפעולות בתוכנית: אחראי לטעינת הדאטה, בניית רכיבי המודל (KNN, Projection, Lab DTOS), ניהול מצב תצוגה ומטריקה, והחזרת תוצאות כ-DTOS מוכנים להצגה. הם "אובייקטים להעברה" (Data Transfer Objects), המטרה שלהם היא שה-UI יקבל מידע בצורה בטוחה ופשוטה, בלי להיות תלוי במחלקות פנימיות כמו Neighbor, ProjectionScore, וכו'.

AppliedViewConfig

DTO שמחזיר ל-UI את מצב התצוגה שאושר בפועל (D/3D2 וצירים) אחרי ולידציה ותיקון.

MetricOption

DTO לייצוג אפשרות metric לבחירה ב-UI (id פנימי + תווית להצגה).

NeighborView

DTO להצגת שכן קרוב: מזהה, מרחק וטקסט מוכן להצגה.

ProjectionScoreView

DTO להצגת תוצאת Projection: מיקום על הציר, סטייה ומדד איכות.

LabResultView

DTO שמחזיר תוצאת Vector Arithmetic כרשימת שכנים immutable להצגה.

שכבת UI באמצעות javafx

הממשק מאפשר בחירת צירים להצגה על המסך, מעבר בין D2 ל-D3, חיפוש מזהים (מילים במקרה שלנו), הצגת שכנים קרובים ע"י בחירת K קרובים, חישוב אריתמטיקה של וקטורים (מילים) עם הדגשת המסלול.

ה-UI אינו מבצע חישובים סמנטיים. במקום זאת הוא עובד מול ממשק ExplorerUseCases בלבד, ומקבל תוצאות כ-DTOS (כגון NeighborView, ProjectionScoreView, LabResultView).

FxApp

נקודת ההפעלה. יוצר ExplorerApplicationService, בונה MainView, ומנסה לבצע loadDefaultIfPresent() – שם הגדרנו את ה-DATA של הנתון בדוגמה במטלה בעת העלייה.

MainView

המסך הראשי של המערכת והמקום שבו מרוכזת האינטראקציה עם המשתמש. הוא מרכיב את כל חלקי הממשק – תצוגת נקודות, בחירת צירים, חיפוש מילים, רשימת שכנים ולשונית המעבדה. תפקידו הוא להגיב לפעולות המשתמש, לשמור מצב כמו המילה שנבחרה (selectedId), וקורא ל-Use Cases בהתאם לפעולות משתמש – שם מתרחשת הלוגיקה.

Scatter2DView

אחראית על הצגת הנתונים במרחב דו-ממדי. היא מקבלת רשימת נקודות מוכנות לציור ומציגה אותן על גבי Canvas, כולל אפשרות לזום, גרירה, בחירה והדגשת שכנים (אינו מחשב מרחקים או שכנים - רק מבקש אותם דרך ExplorerUseCases).

Cloud3DView

מציגה את הנתונים במרחב תלת-ממדי באמצעות JavaFX 3D. היא יוצרת כדורים עבור כל נקודה, מאפשרת סיבוב מצלמה וזום, ומדגישה בחירה ושכנים.

- **מגבלות ידועות:** התצוגה התלת-ממדית קיימת באמצעות Cloud3DView אך דורשת אופטימיזציה ושיפור חוויית שימוש (ביצועים/זום/שליטה במצלמה). הלוגיקה החישובית תקינה ומכוסה בבדיקות, אך רכיב ה-D3 נמצא בשלב שיפור.

LabTabController

מנהל את לשונית ה-Vector Arithmetic Lab. הוא מאפשר למשתמש לבנות ביטוי וקטורי, להפעיל חישוב, ולהציג את התוצאה והשכנים הקרובים ביותר. בנוסף, הוא אחראי לעדכון מסלול גרפי שממחיש את שלבי החישוב. הוא אינו מחשב בעצמו את הווקטורים אלא מפעיל את מנוע החישוב דרך ExplorerUseCases.

LabOverlayPathBuilder

היא מחלקת עזר שמתרגמת רצף של מזהים (מילים) למסלול גרפי שניתן לצייר על גבי התצוגה. היא מקבלת ענן נקודות קיים ומחזירה את נקודות התצוגה המתאימות לפי הסדר הרצוי. כך ניתן להציג בצורה ויזואלית את שלבי החישוב של ביטוי וקטורי. מפעיל solveVectorArithmetic דרך ExplorerUseCases, ומציג שכנים ותוואי overlay.

NeighborDisplayFormatter

אחראית על עיצוב הטקסט של שכנים קרובים ברשימות התצוגה. היא מקבלת נתוני שכן ומחזירה מחרוזת מתאימה להצגה למשתמש.