

הערכתה חלופית – תכונות מונחה עצמים

322450305 – ת.ז 2026

בפרויקט זה יצרנו אפליקציה לחקור אינטראקטיבי של ייצוגים לטניטיים, שהם וקטוריים המיצגים אובייקטים (בגון: 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:

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

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

הчисובים הסמנטיים (מרחקים, שכנים, אריתמטיקה וקטורית) מתבצעים תמיד במרחב הווקטורוני המלא (FULL), בעוד שהציג PCA (במיוחד קטן יותר) משמש לייצוג ויזואלי בלבד. لكن יצרנו מחלוקת שתרכז את כל הייצוגים הקיימים במערכת.

העיקנון המנחה בפרויקט הוא ייצור תשתיית מופשטת המסוגלת לייצג וקטורים של כל ישות נדרשת, לא רק מילים, אלא גם תמונות, רצפי DNA או כל אובייקט שניין להטמעה למרחב מספרי.

בשימוש שלנו, המערכת כוללת אפשרות להריץ את סקריפט הפיתון שהנתון במללה מתוך האפליקציה עצמה. בעת לחיצה על כפתור "יעודי" ("Generate embeddings"), האפליקציה מחלצת את קובץ embedder.py מתיקו המקומי, מרים אותו בתהילך היינו באמצעות `pca_vectors.json` ו-`full_vectors.json`, ומיצרת מחדש את קובצי `hsoes` ו-`PythonScriptRunner` בתיקיית `data`. לאחר יצוריהם, הקבצים נטענים באופן אוטומטי לדיבון והמערכת מתחדשת מיידית. כמו כן ניתן לבחור לטעון קובץ JSON ידני.

לאחר טעינת הקבצים, המשמש בוחר שיטת ייצוג עבור התצוגה, בוחר צירוםلتצוגה בהתאם למינד התצוגה, ואז יכול לבחור מילה על המסר/ברשימה כדי לקבל שכנים קרובים, להריץ `Projection` בין שני עוגנים, לבצע אריתמטיקה בין וקטורים ומילים ולהציג את מסלול החישוב. כך מתאפשרת חקירה אינטראקטיבית של קשרים סמנטיים – גם ויזואלית וגם חישובית.

הארכיטקטורה מפרידה לחלוין 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 ייחיד (`ExplorerUseCases`). כל תוכאות החישובים חזורות כ-`DTOs` (`Data Transfer Object`) כמו `NeighborView`, `ProjectionScoreView`, `LabResultView` (שמותאים לתצוגה, אך SHA-UI אינם תלויים באופן המימוש של מחלקות הפנימיות במודול).

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

המעבר מתצוגת דו-ממדית לתלת-ממדית ללא צורך שינוי במודול המתמטי, אלא רק הוספת View חדש. כלומר גם כאן שינוי תצוגה לא אמור להשפיע על לוגיקתภายในה.

בחורתה בארכיטקטורה שכבותית זו מ透ctor מערכת ליצר מערכת מודולרית, גמישה וכייננת להרחבה. רציתי להבטיח שהLAGIKA החישובית, ניהול הנתונים והתצוגה הגרפית יהיו מופרדים לחלוועין, כך שניתן יהיה לשנות או להרחיב כל רכיב באופן עצמאי. תכנון זה מאפשר הוספה מטריקות חדשות, יציגים נוספים או סוגים תצוגה שונים – מבלי לפגוע בקוד המקורי. בכך המערכת ממשת בפועל עקרונות מרכזיים בתוכנות מונחה עצמים כגון הפרדת אחירות, פתיחות להרחבה וסגורות לשינוי, ושימוש בפולימורפיזם לצורך יצירה גמישות תכוננית.



הפרויקט מגובה במערך בדיקות יחידה ובדיקות קבלה (unit), שמודוא נוכנות מתמטית של הוקטורים, טעינת הנתונים, חישובי שכנים, Vector Arithmetic ו-Projection.

נסקרו בקצרה את השכבות השונות שבנוו בפרויקט, ואת המחלקות השונות בכל שכבה:

שכבה IO

אחריות על טעינת הנתונים החיצוניים. החל מהרצת סקריפט Python שיוצר קבצי JSON, באמצעות ProcessBuilder, ולאחר מכן קריית הקבצים ובנית האובייקטים של המודל מתוך הנתונים הגלומיים. השכבה מבודדת לחלוועין מהLAGIKA המתמטית ומה-UI, כך שניתן להחליף מקור נתונים בעתיד מבלי לשנות את שאר המערכת.

Representation

מחלקה שמייצגת סוג ייצוג (solution) Representation (מושע יחודי לכל ייצוג כך שככל שם קניי קיים מושע יחיד במערכת).

RepresentationSource

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

JsonSource

שימוש של RepresentationSource שטוען embeddings מקובץ JSON באמצעות stream של JAVA, אמת תקינות (מזהים/ממדים/כפליות), ומחריך Map בלתי ניתן לשינוי, עם אחריו טעינה ראשונה.

JsonFormat

אובייקט שיגיר לנו את מבנה השדות בקובץ JSON שנטען (פעם יופיע לנו שדה ראשון של "WORD", אבל אחר נרצה "DNA").

PythonScriptRunner

מריצה סקריפט Python חיצוני באמצעות .ProcessBuilder

שכבת המודל

שכבת המודל מהווה את הליבה המתמטית והמבנהית של הפרויקט. היא אחראית על ייצוג האובייקטים כוקטוריים למרחב לטני ועל ביצוע חישובים. שכבה זו פועלת באופן בלתי תלוי במשתמש ובאופן טעינת הנתונים, ומגדירה את חוקי העבודה של המרחב הווקטורי עצמו.

במסגרת שכבה זו מוגדרים מבני הנתונים המרכזיים (בגון Vector ו-Embedding), וכן הפעולות המתמטיות בווקטו. כל לוגיקת החישוב מתבצעת כאן בלבד, בעוד שכבות אחרות במערכת משתמשות עלייה דרך ממשקים ברורים ומופשיים.

תכונן זה מבטיח הפרדה בין אחראיות חישובית לאחראיות תצוגתית, תומך בגמיישות ובהרחבת עתידית, ומאפשר בדיקות ייחודית מנותקות מה-UI.

Vector

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

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

רכיב הווקטור הוא מרכזי בפרויקט שלנו.

EmbeddingSingle

משחק המציג אובייקט סמנטי בודד (לדוגמה: מילה, במקרה דן) בעלת מספר ייצוגים וקטוריים לפי סוג .Representation

EmbeddingGroup

ממשק המציג אוסף יישיות EmbeddingSingle ומספק גישה מרוכזת לפי מזהה או ייצוג.

מחלקות

EmbeddingItem

שימוש בלתי ניתן לשינוי של EmbeddingSingle שמחזיק מפה מהסוג - RepresentationVector עברו ישות אחת.

EmbeddingStorage

שימוש בלתי ניתן לשינוי של EmbeddingGroup שמאחסן EmbeddingSingle בזיכרון ומוגדר סט ייצוגים אחד לכל הפריטים. וכך יוצר מאגר מרכזי בזיכרון התוכנית שלו.

EmbeddingsAssembler

שימוש שמרכיב EmbeddingStorage מספר 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) לקטור יעד או אובייקט גנרי אותו אנו מייצגים.

הчисוב מתבצע על המרחב הווקטורי המלא (לא על הממד המקורי לתצוגה. כਮון זהה בבחירה של מי שקורא למחלקה).

ונוצר בכך אובייקט חדש שיעזר לנו: Neighbor שמייצג שכן קרוב במשהו וערך מרחק מחושב.

חישובי שכנים מתבצעים באופן ישיר על כל הווקטורים בזיכרון (linear scan), תוך הנחה שמדובר בגודל נתונים מחקרי סביר. הארכיטקטורה מאפשרת עתיד החלפה לבניה נוחנים יותר (כגון KD-Tree או ANN Index) מבלי לשנות את שכבת ה-UI.

Projection System

CustomProjectionService

שירות הבונה ציר סמנטי משתי ישויות ומפרקן עליו את כל ה-embeddings בקבוצה.
(לדוגמה: "עכבר" → "עריש").

ProjectionAxis

מייצג ציר חד-ממדי במרחב הווקטוריו ומחשב מיקום וטיטה של וקטור ביחס אליו.

ProjectionScore

רשומת תוצאה המייצגת מזאה עם מיקומו על הציר, מרחקו האורתוגונלי וציון טויה.

Vector Arithmetic Lab

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

הרענון הכללי

- מחשבים וקטור תוצאה מהביטוי.
- מארטרים את K השכנים הקרובים לוקטור התוצאה.
- מוחזרים גם מסלול ביןיהם (במרחב התצוגה) כדי שה-UI יוכל להציג ויזואלייזציה של שלבי החישוב.

VectorArithmeticLab

פasad' שמחשב וקטור ומחזר שכנים קרובים לתוצאה.

Term

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

VectorExpression

מייצג ביטוי שרשרת של איברים חתומים עם Builder נכון.

LabResult

תוצאה מעבדה הכללת וקטור מחושב ורשימת שכנים קרובים.

VectorExpressionEvaluator

מחשב ביטוי וקטורי לוקטור תוצאה ביצוג נתון.

Subspace Grouping (Group Centroid Analysis)

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

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

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

שכבה View

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

- **ViewPoint**: ייצוג כללי של נקודה במרחב תצוגה במימד בלבד.
- **Point_2D**: מייצג נקודה דו-ממדית במרחב התצוגה.
- **Point_3D**: מייצג נקודה תלת-ממדית במרחב התצוגה.
- **LabeledPoint**: עצף נקודת תצוגה ייחד עם מזהה ישות.
- **PointCloud**: אוסף נקודות מתיוגות המוכן להציגה.
- **ViewMode**: ממשק המגדיר כיצד למפות וקטור למרחב תצוגה.
- **ViewMode2D**: מיפוי דו-ממדי לפי שני רכיביו וקטור נבחרים.
- **ViewMode3D**: מיפוי תלת-ממדי לפי שלושה רכיביו וקטור נבחרים.
- **ViewSpace**: מתרגם embeddings לענן נקודות בהתאם ל-ViewMode.

שכבה שירותים (Application Layer)

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

ExplorerUseCases

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

ExplorerApplicationService

השימוש בפועל של הפעולות בתוכנית: אחראי לטעינת הדטה, בניית רכבי המודל (KNN, DTOS, Projection, Lab), ניהול מצב תצוגה ומטריקה, והחזרת תוצאות-*cso* מובנים להציג. הם "אובייקטים להעbara" (Data Transfer Objects), המטריה שלהם היא שה-*UI* קיבל מידע בצורה בטוחה ופושטת, בלי להיות תלוי במחלקות פנימיות כמו *ProjectionScore*, *Neighbor*, *ProjectionScoreView* וכו'.

AppliedViewConfig

DTO שמחזיר ל-*UI* את מצב התצוגה שאושר בפועל (2D/3D וצירים) אחורי ולידציה ותיקו.

MetricOption

DTO לייצוג אפשרות *metric* לבחירה ב-*UI* (eo פנימי + תווית להציג).

NeighborView

DTO להציג שכן קרוב: מזהה, מרחק וטקסט מוכן להציג.

ProjectionScoreView

DTO להציג תוצאות *Projection*: מיקום על הציר, סטייה ומדד איקות.

LabResultView

DTO שמחזיר תוצאה *Vector Arithmetic* ברשימה שכנים *immutable* להציג.

שכבות UI באמצעות JavaFX

המשك מאפשר בחירת צירים להציג על המסך, מעבר בין 2D ל-3D, חיפוש מזהים (מילים במקורה שלו), 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 משק *ExplorerUseCases* בלבד, ומתקבל תוצאות-*cso* (כגון *NeighborView*, *ProjectionScoreView*, *LabResultView*).

FxApp

נקודת ההפעלה. יוצר *ExplorerApplicationService*, בונה *MainView*, ומנסה לבצע *loadDefaultIfPresent* – שם הגדרנו את DATA של הנתון בדוגמה במטה בעת העליה.

MainView

המסך הראשי של המערכת והמקום שבו מרכזת האינטראקציה עם המשתמש. הוא מרכיב את כל חלקי המשק – תצוגת נקודות, בחירת צירים, חיפוש מילים, רשימה שכנים ולשונות המעבדה. תפקידו הוא להגיב לפעולות המשתמש, לשמר מצב כמו המילה שנבחרה (*selectedId*), וקורא ל-*Use Cases* בהתאם לפעולות משתמש – שם מתורחת הלוגיקה.

Scatter2DView

אחריות על הצגת הנתונים במרחב דו-ממדי. היא מקבלת רשימה נקודות מוכנות לציר ומציגה אותן על גבי Canvas, כולל אפשרות לזרום, גירה, בחירה והdagשת שבנים (אינו מחשב מרחקים או שבנים - רק מבקש דרך דרך ExplorerUseCases).

Cloud3DView

מציגה את הנתונים במרחב תלת-ממדי באמצעות JavaFX 3D. היא יוצרת כדרים עברם כל נקודה, מאפשרת סיבוב מצלמה וズום, ומדגישה בחירה ושבנים.

- מגבילות ידועות:** התצוגה התלת-ממדית קיימת באמצעות Cloud3DView אך דורשת אופטימיזציה ושיפור חווית שימוש (ביצועים/זום/שליטה במצלמה). הלוגיקה החישובית תקינה ומוכסה בבדיקות, אך רכיב ה-3D נמצא בשלב שיפור.

LabTabController

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

LabOverlayPathBuilder

היא מחלקת עוזר שמתווגמת וצף של מזהים (밀ימטר) למסלול גרפי שנייה לציר על גבי התצוגה. היא מקבלת ענן נקודות קיימים ומחזירה את נקודות התצוגה המתאימות לפי הסדר הרצוי. כך ניתן להציג בצורה ויזואלית את שלבי החישוב של ביטוי וקטורי. מפעיל OverlayPathBuilder דרך ExplorerUseCases solveVectorArithmetic.

NeighborDisplayFormatter

אחריות על עיצוב הטקסט של שבנים קרובים ברשימות התצוגה. היא מקבלת נתונים שכך ומחזירה מחרוזת מתאימה להציגו למשתמש.