

הערכתה חלופית – תכונות מונחה עצמים

322450305 – ת.ז 2026

בפרויקט זה יצרנו אפליקציה לחקור אינטראקטיבי של מרחבי Embeddings, שהם ייצוגים וקטוריים של מילום במרחב לטני ריב-מדדי. המטרה המרכזית של המערכת היא לאפשר חקירה ויזואלית ומתמטית של קשרים סמנטיים בין מושגים, 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 בניתוח דו-ממדי והן בתצוגה תלת-ממדית, ומאפשרת ביצוע חישובים סמנטיים כגון מרחקים, חיפוש שכנים קרובים, הקרנות מותאמות אРИתמטיקה וקטורית. החישובים הסמנטיים (מרחוקים, שכנים, אРИתמטיקה וקטורית) מתבצעים תמיד במרחב הווקטורוני המלא (FULL), בעוד ש-PCA משמש לייצוג ויזואלי בלבד.

המשתמש טוען קבצים לייצג (אצלנו: `pca_vectors + pca_vectors + full`), בוחר יציג לתצוגה (לרוב PCA), בוחר צירים לתצוגה בהתאם למימד התצוגה, ועוד יכול לבחור מילה על המסך/ברשימה כדי לקבל שכנים קרובים, להריץ `Projection` בין שני עוגנים, לבצע אРИתמטיקה בין וקטורים ומילים ולהציג את מסלול החישוב. כך מתקבלת חקירה אינטראקטיבית של קשרים סמנטיים – גם ויזואלית וגם חישובית.

העיקנון המנחה בפרויקט הוא ייצור תשתיות מופשטת המסוגלת לייצג וקטוריים של כל ישות נדרשת, רק מילים, אלא גם תמונות, רצפי DNA או כל אובייקט שניתן להטמעה למרחב מסווני. המערכת אינה תליה בסוג הדטה אלא רק במבנה הווקטורוני שלו.

ניתן להציג מספר ייצוגים לאותו מרחב (למשל: וקטור מלא, PCA שהוא במימד מצומצם). כל יציג הוא אוסף וקטורים באותו מימד, ללא תלות בפרשנות הסמנטית שלהם. וכן יצרנו מחלקה שתרכז את כל הייצוגים הקיימים במערכת.

הארכיטקטורה מפרידה לחלעות בין שכבת החישוב לשכבות התצוגה. כל הלוגיקה המתמטית (מרחוקים, שכנים, הקרנות, אРИתמטיקה וקטורית) ממומשת בשכבת האפליקציה, בעוד ה-UI מתקשר עם שכבת האפליקציה דרך ממשק יחיד (`ExplorerUseCases`). כל תוכנות החישובים חזורות כ-`DTOs` (Data Transfer Object) כמו `NeighborView`, `ProjectionScoreView`, `TransferObject` ועוד. `LabResultView` שמותאים לתצוגה, כך שה-UI אינו תלוי במחלקות הפנימיות של המודול והאלגוריתמים.

חישובי המרחק ממושכים בתבנית Strategy (פולימורפיזם), כך שניתן לבחור בזמן ריצה בין Cosine Euclidean Distance ל-Distance. (Open/Closed Principle).

המעבר מציגת דו-ממדית לתלת-ממדית לא דרש שינוי במודול המתמטי, אלא רק הוספה View חדש. כלומר גם כאן שינוי תצוגה לא אמור להשפיע על לוגיקתภายในה.

בחורתה בארכיטקטורה שכבותית זו מ透ctor מטריה ליצור מערכת מודולרית, גמישה וניתנת להרחבה. רציתי להבטיח שהLAGIKA החישובית, ניהול הנתונים והציגות הגרפית יהיו מופרדים לחלוון, כך שניתן יהיה לשנות או להרחב כל רכיב באופן עצמאי. תכנון זה מאפשר הוספה מטירות חדשות, יציגים נוספים או סוגים תצוגה שונים – אוליפגעו בקוד הקיימ. בכך המערכת ממשת בפועל עקרונות מרכזיים בתוכנות מונחה עצמים כגון הפרדת אחראיות, פתיחות להרחבה וסגורות לשינוי, ושימוש בפולימורפיזם לצורך יצירה גמישות תכוננית.



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

המערכת כוללת בפטור ייעודי במשק ("Generate embeddings") המאפשר להריץ את סкриיפט הפיתון מ透ctor האפליקציה עצמה. בעת לחייב, האפליקציה מחלצת את קובץ `embedder.py` מ透ctor משאבי המערכת, מרים אותו בתהיליך חיצוני באמצעות `PythonScriptRunner`, ומיצרת מחדש את קובצי `hsos.json`, `full_vectors.json` ו-`hsos.j`. לאחר ייצורם, הקבצים נוענים באופן אוטומטי לדירקטוריון והמערכת מתחדשת מיידית.

נסקרו בקצרה את השבבות השונות שבנוינו בפרויקט, ואת המחלקות השונות בכל שכבה:

שכבה 0

אחריות על טעינת הנתונים החיצוניים. כמו למשל: קריאת קובצי JSON שנוצרו על ידי סкриיפט `hsos.py`. החל מהרצת סкриיפט חיצוני באמצעות `ProcessBuilder`, ולאחר מכן: בנית האובייקטים של המודול מתוך הנתונים הגלויים. השכבה מבודדת לחלוון מהLAGIKA המתמטית ומה-UI, כך שניתן להחליף מוקור נתונים בעתיד אולי לשנות את שאר המערכת.

Representation

מחלקה שמייצגת סוג ייצוג (Representation) כ-`Flyweight` (מופע ייחודי לכל ייצוג כך שכל שם קנווי קיים מופע ייחיד במערכת).

RepresentationSource

ממשק המגדיר מקור טעינה לייצוג מסוים של אובייקט גנרי כלשהו (T), האחראי לספק מיפוי בלתי-יניתן לשינוי בין מזהים לווקטורים. כל שימוש של הממשק יגדיר את מקור הנתונים ואופן שבו הוא יאחסן אותם בטיעונתם.

JsonSource

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

JsonFormat

אובייקט שיידיר 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 והמבנה של הפרויקט. היא אחראית על ייצוג האובייקטים כווקטורים למרחב לטני ועל ביצוע חישובים. שכבה זו פועלת באופן בלתי-תלוי במשחק המשמש באופן טיענת הנתונים, ומגדירה את חוקי העבודה של המרחב הווקטורית עצמו.

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

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

Vector

מחלקה בלתי-יניתנת לשינוי (immutable) המייצגת וקטור מספרי (מערך של double) ומכללה פעולות:

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

רכיב הווקטור הוא מרכזי בפרויקט שלנו.

EmbeddingSingle

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

EmbeddingGroup

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

מחלקות

EmbeddingItem

שימוש בלתי ניטרלי לשינוי של `EmbeddingSingle` שמחזק מפה מהווער-Representation מפה `Map` ל-`Vector` עבור ישות אחת.

EmbeddingStorage

שימוש בלתי ניטרלי לשינוי של `EmbeddingGroup` שמאחסן `EmbeddingSingle` בזיכרון ומודיע סט ייצוגים אחד לכל הפריטים. וכן יוצר מאגר מרכזי בזיכרון התוכנית שלם.

EmbeddingsAssembler

শর্মুক কোর্পস EmbeddingStorage শর্মুক কোর্পস factory
শর্মুক কোর্পস RepresentationSource), তোক এইচড লেকি মজাহিম ওকিফ শল স্ট মজাহিম (হাবীকতিম
শান মাইচগিম, মিলা ও ID অছৰ জহা বেল মকুরোত.

חישובי מרחק (Strategy Pattern)

המערכת תומכת במספר מטריות מרחק:

- Euclidean Distance
- Cosine Distance

המטריות 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.

הчисלוב מתבצע על המרחב הווקטורית המלא (לא על הממד המקורי ל特派חה. במודון שזה בבחירהו של מי שקורא למחלקה).

ניצור כאן אובייקט חדש שיעזר לנו: `Neighbor` שמייצג שכן קרוב בmazeה וערך מרחק מחושב.

חישובי שכנים מתרכזים באופן ישיר על כל הווקטורים בזיכרון (linear scan), מתוך הנחה שמדובר בגודל נתונים מחקרי סביר. הארכיטקטורה מאפשרת בעתיד החלפה למבנה נתונים יותר (בגון KD-Tree או ANN) מבלי לשנות את שבבת ה-IO.

Projection System

CustomProjectionService

שירות הבונה ציר סמנטי משתי ישויות ומקרין עליו את כל ה-embeddings בקבוצה.
(לדוגמה: "עבי" $\leftarrow \rightarrow$ "עшир").

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 טויה.

Vector Arithmetic Lab

מודול מתקדם המאפשר בניית ביטויים וקטוריים בוגן: $\text{king} - \text{man} + \text{woman}$ צפוי להניב תוצאה הרקובה ל-[queen](#).

הרענון הכללי

- מחשבים וקטור תוצאה מהביטוי.
- מעריכים את K השכנים הקרובים לוקטור התוצאה.
- מחזירים גם מסלול ביןיהם (במרחב התצוגה) כדי שה-IO יוכל להציג ייזואלייזציה של שלבי החישוב.

VectorArithmeticLab

פסאץ' שמחשב וקטור ומחזיר שכנים קרובים לתוצאה.

Term

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

VectorExpression

מייצג ביטוי שרשרת של איברים חתומים עם Builder נכון.

LabResult

תוצאת מעבדה הכוללת 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 עבודה על קבוצה של ישויות במקומות על ישות בודדת. המשמש יכול לבחור מספר מזהים (לדוגמה: מספר מילימטרים), והמערכת מחשבת את הווקטור המركבי שלהם (Centroid) במרחב הווקטורי המלא. גישה זו מאפשרת ניתוח סמנטי בرمת "תת-מרחב" (Subspace), כלומר חקר הקשרים המשותפים בין מספר מושגים בקבוצה ולא רק באופן פרטני. החישוב מתבצע כה:

- עבור 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 המركבי באמצעות המטריקה שנבחרה.

הчисוב מתבצע בשכבה השירותים (ExplorerApplicationService) על בסיס היצוג המלא ובהתאם למטריקת המרחק שנבחרה, והזע 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 במינד בלבד.
- `Point_2D`: מייצג נקודה דו-ממדית במרחב התצוגה.
- `Point_3D`: מייצג נקודה תלת-ממדית במרחב התצוגה.
- `LabeledPoint`: עוטף נקודת תצוגה ייחד עם מזהה ישות.
- `PointCloud`: אוסף נקודות מתיוגות המונן להצגה.
- `ViewMode`: מסתק 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 נקודות בהתאם ל-`embeddings`.

שכבת שירותים (Application Layer)

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

ExplorerUseCases

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

ExplorerApplicationService

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

AppliedViewConfig

DTO שמחזיר ל-UI את מצב התצוגה שאושר בפועל (2D/3D וצירים) אחרי ולידציה ותיקון.

MetricOption

DTO לייצוג אפשרויות metric לבחירה ב-UI (පו פנימי + תווית להציג).

NeighborView

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

ProjectionScoreView

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

LabResultView

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

שכבות UI באמצעות JavaFX

המשחק מאפשר בחירות צירים להציג על המסך, מעבר בין 2D ל-3D, חיפוש מזהים (מילימטרים במקורה שלנו), הצגת שכנים קרובים ע"י בחירת K קרובים, חישוב אריתמטיקה של וקטורים (מילימטרים) עם הדגשת המסלול.

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

FxApp

נקודת הפעלה. יוצר `ExplorerApplicationService`, בונה `MainView`, ומנסה לבצע נקודות הפעלה. שם הגדכנו את `loadDefaultIfPresent` – שם מתרחשת הטעינה במלואה בעת העלייה.

MainView

המסך הראשי של המערכת והמקום שבו מרכזת האינטראקציה עם המשתמש. הוא מרכיב את כל חלקי המשחק – תצוגת נקודות, בחירת צירים, חיפוש מילים, רשימת שכנים ולשונית המעבדה. תפקידו הוא להגיב לפעולות המשתמש, לשומר מצב בו המילה שנבחרה (selectedId), וקורא לעס Kas Use Cases בהתאם לפעולות משתמש – שם מתרחשת הלוגיקה.

Scatter2DView

אחריות על הצגת הנתונים במרחב דו-ממדי. היא מקבלת רשימה נקודות מוכנות לציר ומציגה אותן על גבי `Canvas`, כולל אפשרותズום, גירה, בחירה והדגשת שכנים (אינו מחשב מרחקים או שכנים – רק מבקש דרך דרך `ExplorerUseCases`).

Cloud3DView

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

- **מגבילות ידועות:** התצוגה התלת-ממדית קיימת באמצעות `Cloud3DView` אך דורשת אופטימיזציה ושיפור חווית שימוש (ביצועים/זום/שליטה במצולמה). הלוגיקה החישובית תקינה ומכוונה 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 ביותר. בנוסף, הוא אחראי לעדכון מסלול גרפי שמחיש את שלבי החישוב. הוא אינו מחשב בעצמו את הווקטורים אלא מפעיל את מנוע החישוב דרך `ExplorerUseCases`.

LabOverlayPathBuilder

היא מחלקת עזר שמתרגמת רצף של מזהים (מילימטרים) למסלול גרפי שנitinן לציר על גבי התצוגה. היא מקבלת ענן נקודות קיימות ומחזירה את נקודות התצוגה המתאימות לפי הסדר

הרצוי. כך ניתן להציג בצורה ויזואלית את שלבי החישוב של ביטוי וקטורי. מפעיל דרך `solveVectorArithmetic`, `ExplorerUseCases` ו-`overlay`.

NeighborDisplayFormatter

אחריות על עיצוב הטקסט של שכנים קרובים ברשימות הציגות. היא מקבלת נתונים שוכן ומחזירה מחרוזת מתאימה להציגה למשתמש.