

הערכת חלופית – תכונות מונחה עצמים

322450305 – ת.ז 2026

בפרויקט זה יצרנו אפליקציה לחקור אינטראקטיבי של מרחבי Embeddings, שהם ייצוגים וקטוריים של מילים במרחב לטני רב-ממדי. 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 ואריתמטיקה וקטורית.

הчисובים הסמנטיים (מרחקים, שכנים, אריתמטיקה וקטורית) מtabצעים תמיד במרחב הווקטורוני המלא (FULL), בעוד PCA- מושך לייצוג ויזואלי בלבד.

המערכת כוללת בפתרון "יעודי" במשחק ("Generate embeddings") המאפשר להריץ את סקורייפט הפיתון מתוך האפליקציה עצמה. בעת לחריצה, האפליקציה מחלצת את קובץ `embedder.py` מתוך משאבי המערכת, מרים אותו כתהיליך חיוני באמצעות `PythonScriptRunner`, ומיצרת מחדש את קובצי `hisos.json` ו-`full_vectors.data`. לאחר יצירתם, הקבצים נטענים באופן אוטומטי לדיברין והמערכת מתחדשת מיידית.

לאחר טעינת הקבצים, המשמש בוחר שיטת ייצוג עבור התצוגה (באמור במקרה שלנו: PCA), בוחר צירים לתצוגה בהתאם למינד 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 מקבלת חקירה אינטראקטיבית של קשרים סמנטיים – גם ויזואלית וגם חישובית.

העיקנון המנחה בפרויקט הוא ייצור תשתיות מופשטת המסוגלת לייצג וקטורים של בל ישות נדרשת, רק מילים, אלא גם תמונות, רצפי DNA או כל אובייקט שניתן להטעה למרחב מסווני. המערכת אינה תליה בסוג הדעתה אלא רק במבנה הווקטוריו שלו.

ניתן להציג מספר ייצוגים לאוטו מרחב (למשל: וקטור מלא, PCA שהוא במינד מצומצם). כל ייצוג הוא אוסף וקטורים באותו מינד, ללא תלות בפרשנות הסמנטית שלהם. וכן יצרנו מחלוקת שתרכז את כל הייצוגים הקיימים במערכת.

הארQUITktורה מפרידה לחלוטין בין שכבת החישוב לשכבות התצוגה. כל הלוגיקה המתמטית (מרחקים, שכנים, הקרנות, אריתמטיקה וקטורית) ממומשת בשכבה האפליקציה, בעוד ה-UI מתקשר עם שכבת האפליקציה דרך ממשק ייחיד (`ExplorerUseCases`). כל תוצאות החישובים

חוירות כ-`DTOs` (Data Transfer Object) כמו `NeighborView`, `ProjectionScoreView`, `LabResultView`) שמותאים לתצוגה, אך שה-`UI` אינו תלוי בחלוקת הפנימית של המודול והאלגוריתמים.

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 צורך שינוי במודול המתמטי, אלא רק הוספה `View` חדשה. כלומר גם בכך שינוי תצוגה לא אמרו להשפיע על לוגיקתภายในה.

בחرتבי ארכיטקטורה שכבותית זו מ透ctor מערכת למודולרית, גמישה וניתנת להרחבה. רציתי להבטיח שהלוגיקה החישובית, ניהול הנתונים והתצוגה הגרפית יהיו מופרדים לחלווטין, וכך ניתן יהיה לשנות או להרחב כל רכיב באופן עצמאי. תכונן זה מאפשר הוספה מטריקות חדשות, יוצרים נוספים או סוגים נוספים – מבלי לפגוע בקוד המקורי. בכך המערכת ממשתת בפועל עקרונות מרכזיים בתוכנות מונחה עצמים בגין הפרדת אחריות, פתיחות להרחבה וסירות לשינוי, ושימוש בפולימורפיזם לצורך יצירה גמישהות תכונונית.



הפרויקט מגובה במערך בדיקות יחידה ובדיקות קבלה (Unit Test), שמודא נכונות מתמטית של

הווקטורים, 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. כמו למשל: קריאתקובצי `JSON` שנוצרו על ידי `Python`. החל מהרצת `ProcessBuilder` חיצוני באמצעות `ProcessBuilder`, ולאחר מכן: בנית האובייקטים של המודול מתוך הנתונים הגלומיים. השכבה מבוצעת לחלווטין מהלוגיקה המתמטית ומה-`UI`, אך שינוי להחליף מקור נתונים בעתיד מבלי לשנות את שאר המערכת.

Representation

מחלקה שמייצגת סוג יציג (Flyweight Representation) כ-`Flyweight` (מופע יחודי לכל יציג אך שבל שם קניוני קיים מופע יחיד במערכת).

RepresentationSource

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

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 מנוטקות 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 Representation - ל- Vector עברו ישות אחת.

EmbeddingStorage

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

EmbeddingsAssembler

הארכיב EmbeddingStorage שמרכזו 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 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.

הчисוב מתבצע על המרחב הוקטורית המלא (לא על הממד המקורי ל特派יה. כਮון זהה בבחירה של מי שקורא למחלקה).

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

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

Projection System

CustomProjectionService

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

ProjectionAxis

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

ProjectionScore

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

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.
- מאיירים את K השכנים הקרובים לוקטור התוצאה.
- מחדירים גם מסלול ביןיהם (במרחב התצוגה) כדי שה-*itself* יוכל להציג ויזואлизציה של שלבי החישוב.

VectorArithmeticLab

פסאוד' שמחساب וקטור ומחדר שכנים קרובים לתוצאה.

Term

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

VectorExpression

מייצג ביטוי כשרשרת של איברים חתומים עם Builder נכון.

LabResult

תוצאת מעבדה הכוללת וקטור מחושב ורשימת שכנים קרובים.

VectorExpressionEvaluator

מחساب 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, והזע מקבל את התוצאה כ-`DTO` ללא תלות בפרטי השימוש 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`: עוטף נקודת תצוגה יחיד עם מזהה ייחודי.
- `PointCloud`: אוסף נקודות מתיוגות המוכן להציג.
- `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-

שכבה שירותים (Application Layer)

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

ExplorerUseCases

ממשק שמאגד את כל הפעולות שה-`I`-`U` יכול לבצע: טעינה, תצוגה, שכבים, הקרנות וחישובי וקטוריים. ה-`I`-`U` 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 התצוגה ומטריקה, והחזרת תוצאות ב-*DTOs* 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 (2/3D וצירים) אחרי ולידציה ותיקון.

MetricOption

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

NeighborView

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

ProjectionScoreView

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

LabResultView

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

שכבות UI באמצעות JavaFX

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

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

FxApp

נקודת ההפעלה. יוצר *ExplorerApplicationService*, בונה *MainView*, ומנסה לבצע נקודת הפעלה. שם הגדכנו את *loadDefaultIfPresent*() – שם מתרחשת בדוגמה במטה בעת העליה.

MainView

המסך הראשי של המערכת והמקום שבו מרכזת האינטראקציה עם המשתמש. הוא מרכיב את כל חלקי המשחק – תצוגת נקודות, בחירת צירים, חיפוש מיילים, רשימה שכנים ולשונית המעבדה. תפקידו הוא להגיב לפעולות המשתמש, לשמר מצב כמו המילה שנבחרה (selectedId), וקורא לא-*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 והdagשת שבנים (אינו מחשב מרחוקים או שבנים - רק מבקש אותם דרך `ExplorerUseCases`).

Cloud3DView

מציגה את הנתונים במרחב תלת-ממדי באמצעות `JavaFX 3D`. היא יוצרת כדרים עברם כל נקודה, מאפשרת סיבוב מצלמה וזום, 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 שמתורגמת וצף של מזהים (밀ימטר) למסלול גרפי שנייה לציר 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`.

NeighborDisplayFormatter

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