2022

Disease-Prediction

ספר פרויקט

קרויזר בתיה

מכללה: סמינר בנות אלישבע.

שם סטודנט: בתיה קרויזר.

ת"ז סטודנט: 212887467.

שם המנחה: גב' שולמית ברלין.

תאריך הגשה:

תוכן

הצעת פרויקט

סמל מוסד: 189084

שם מכללה: בנות אלישבע

שם הסטודנט: בתיה קרויזר

ת"ז הסטודנט: 324817162

שם הפרויקט: אבחון מחלות

תיאור הפרויקט:

תוכנה המקבלת מהמשתמש תסמינים ומאבחנת את המחלה.

הגדרת הבעיה האלגוריתמית:

האלגוריתם צריך לעבד את התסמינים ולזהות על ידם את המחלה.

רקע תיאורטי בתחום הפרויקט:

לפני שאדם פונה לרופא הוא יעדיף ללמוד על מה התסמינים מראים ולפנות לרופא ספציפי שמומחה במחלה שהתסמינים מראים עליה.

ולאחר זיהוי המחלה הוא ישמח לקבל עליה פרטים בצורה מסודרת.

תהליכים עיקריים בפרויקט:

- קבלת תסמינים מהמשתמש
- עיבוד התסמינים למובנים לאלגוריתם.
- חיזוי המחלה על יד האלגוריתם,
- מציאת חומר על המחלה.
- תגובה למשתמש-המחלה כולל פרטים.

תיאור הטכנולוגיה:

צד שרת: python

צד לקוח: react

מסד נתונים: מחלות ותסמינים הלקוח מ-kaggle

פרוטוקולי תקשורת:

לוחות זמנים:

- אוקטובר - תכנון הפרויקט.
- נובמבר - לימוד האלגוריתם.
- דצמבר - כתיבת האלגוריתם.
- ינואר - ניסוי ובדיקות.
- פברואר-מרץ - בנית ממשק ופונקציות נוספות.
- אפריל - ספר פרויקט.
- מאי- הגשת הפרויקט.

חתימת הסטודנט: בתי קרויזר

חתימת רכז המגמה:

אישור משרד החינוך:

מבוא

הרקע לפרויקט:

בעולם המודרני אנו מביאים לאנשים כל שירות שאפשר למסך הפרטי שלהם,

בכמה שפחות מאמץ וכמה שפחות עלות.

בן אדם לעיתים חש סימפטומים שונים שגורמים לו לחשוש ממחלות,

כאשר הוא ירצה לקבוע תור לבדיקה אצל רופא הוא יתקשה להתמקד באיזה רופא לבחור על ידי הסימפטומים לבדם, ויקבע אצל רופא כללי עפ"י רוב, והרופא הזה יכוון אותו לרופא המסוים שמטפל במחלות המאפיינות את הסימפטומים שלו.

אך לפני שאדם פונה לרופא הוא יעדיף ללמוד על מה התסמינים מראים ולפנות לרופא ספציפי שמומחה במחלה שהתסמינים מראים עליה.

הפרויקט שלי פותר את הבעיה בכך שהוא מעניק ממשק נוח למשתמש שבו ניתן להכניס רשימת תסמינים ולקבל אבחון של המחלה והסברים מפורטים עליה.

תהליך המחקר:

א. הבעיה שבה הפרויקט עוסק היא איך לאבחן את המחלה ע"פ הסימפטומים. החלטתי להתבסס על אלגוריתם שעובד על בינה מלאכותית כדי לפתור את הבעיה בצורה החכמה והדינאמית יותר.

בינה מלאכותית עובדת כך שהאלגוריתם מקבל מסד נתונים שמכיל תסמינים ואת המחלות המאובחנות וישתמש במודל סיווג שילמד את הנתונים בכך שירוץ עליהם ויבנה מודל שיאבחן את המחלה בצורה הטובה ביותר.

כך שבהתבסוסת על מסד נתונים שמכיל מספיק נתונים נוכל לפתור את הבעיה בצורה המהירה והפשוטה ביותר.

כך שבאלגוריתם שלי החלק החשוב ביותר היה חיפוש מסד נתונים:

ישנם מסדי נתונים רבים שמכילים מידע על מחלות מסוימות וספציפיות כגון מחלות לב, ריאה וכו' אך אפשרות זאת לא הייתה יעילה לאלגוריתם הספציפי כי הרעיון שלו הוא משהו שייתן פתרון כללי לכל מחלה ולא רק למחלות ספציפיות, כמו כן רציתי פתרון שלא ידרוש

מהמשתמש שום דבר מלבד התיאור של הסימפטומים הספציפיים שלו, כך שמסדי נתונים המשתמשים בצילומי סיטי או בדיקות דם לא יכלו לעזור לי.

כך שמסד הנתונים שחיפשתי היה צריך להכיל תסמינים ומחלות וכמו כן להיות גדול מספיק כדי ליצור מודל שמתבסס על הכללה טובה מספיק.

ב. כדי לאבחן בצורה הטובה ביותר למדתי על פתרון בעיות סיווג דרך בינה מלאכותית, חיפשתי אלגוריתמי סיווג, ובחרתי את היעילים מבניהם.

ג. כדי להציג מידע למשתמש על המחלה החלטתי להתבסס על מאגר המידע הגדול ביותר שאני מכירה- גוגל, כך שהאלגוריתם שלי יחפש מידע על המחלה, וכאן חיפשתי ספריה שיודעת לגשת דרך פייתון לחיפוש גוגל ואלגוריתם שידע לשלוף את הנתונים מדף html של גוגל

סקירת ספרות:

stackOverflow,רשת-טק,GitHub,analyticsvidhya, kaggle .

מטרות ויעדים בפרויקט

מטרות:

- למידת התחום של סיווג באמצעות בינה מלאכותית.
- חיזוי מחלה על ידי קבלת תסמינים.
- נתינת מידע ללקוח על המחלה בצורה מפורטת ומסודרת.

יעדים:

- מציאת מסד נתונים מדויק המכיל תסמינים ומחלות.
- מציאת המודלים הטובים ביותר לחיזוי המחלה.
- יצירת אלגוריתם למציאת מידע על מושג, באמצעות google search.
- יצירת ממשק נוח למשתמש.
- שימוש בטכנולוגיות חדשניות ובשפות הנחשבות ביותר בעולם הפיתוח, תוך לימוד תחומים חדשים והתנסות בטכניקות המתקדמות ביותר.
- עבודה מסודרת, תוך שימת דגש על תיעוד מלא וברור.
- מציאת דרך יעילה להצגת מידע למשתמש.

אתגרים בפרויקט

- להגיע לחיזוי ברמה גבוהה ביותר: על ידי מסד נתונים טוב ומודלים מדויקים.
- ליצור למשתמש ממשק קל לשימוש:

מסד הנתונים עליו התבססתי מכיל 132 פרמטרים-תסמינים וכדי שהמשתמש לא יצטרך לעבוד ולבחור מכל 132 התסמינים מה הוא מרגיש ומה לא, רציתי ליצור אלגוריתם שיקבל רשימה של תסמינים שנכתבו בשפה חופשית על ידי המשתמש וימיר אותם למשהו שהמודל יוכל לקבל.

- לכתוב אלגוריתם לחיפוש מידע על מחלה שימצא כמה שיותר מידע בזמן המינימלי האפשרי- בשביל לקצר את זמן החיפוש האלגוריתם יעבוד כך שאם האלגוריתם חיפש בעבר מידע הוא ישמור אותו לפעם הבאה.

מדדי הצלחה למערכת

- חיזוי המחלה ברמה של לפחות 95% הצלחה.
- הפיכת התסמינים שהמשתמש הכניס לרשימה קריאה למודל בצורה יעילה ומדויקת כמה שיותר.
- חיפוש מידע בצורה מקיפה והצגה למשתמש בצורה מובנת וברורה.

תיאור מצב קיים

המצב הקיים בתחום הוא שישנם אפליקציות החוזות מחלות אך המשתמש צריך לבחור תסמינים ממאגר קיים דבר שגורם לחוסר בחוויית המשתמש וכמו כן הוא אינו מקבל מידע על המחלה שחזו לו ונשאר עם השם שלה ללא ידע נוסף עליה.

כמו כן הדברים הקיימים בתחום, חוזים בהצלחה נמוכה ביותר ובדרך כלל בהתבססות על נתונים כמו בדיקות דם וצילומים, מה שלא עוזר לאדם שאין לו את הנתונים האלה.

כמו כן לא ראיתי שקימות אפליקציות המחפשות מידע בהתבססות על גוגל ומציגות מידע בצורה ברורה למשתמש.

רקע תיאורטי, ניתוח חלופות

מטרת הפרויקט יכלה להיות מושגת אם כל אדם שלא מרגיש טוב ילך לרופא ואחרי שהרופא יתשאל אותו ויבדוק את תסמיניו הוא יאבחן את המחלה וישלח אותו לרופא המומחה למחלה הזאת או יציע תרופות המתאימות לבעיה זאת והאדם יכול לבחור אם להמשיך ולשאול את הרופא פרטים או לברר בעצמו.

לפתרון זה חסרונות רבים

1 זמן: הליכה לרופא קביעת תור זמן המתנה וכל אלו לוקחים זמן יקר מהאנשים.

2 כסף: בדיקת הרופא עולה סכום ואם אין ביטוח הסכום אף יקר ביותר.

3 חוסר ידע על המחלה: במידה ואדם מחליט לברר בעצמו על המחלה הוא לא מצליח להקיף את כל הנושאים משלל החומר הנמצא על מחלה זו באינטרנט ולפעמים מאבד ידיים ורגליים ויוצא מבולבל.

בבואי לפתור את הבעיה האלגוריתמית עמדה בפני עוד דרך:

יכולתי לפתור את האלגוריתם בצורה מסורתית על ידי בנית מסד נתונים המכיל את התסמינים הידועים על כל מחלה וליצור אלגוריתם המשתמש במשתני תנאי וכך מאבחן את המחלה.

גם לבעיה זו חסרונות רבים:

ראשית העבודה לא מדויקת אם נתייחס לחולשותיו הטבעיים של כל בן אנוש.

היא אינה דינאמית כיוון שכל תנאי תלוי בשני והוספה של מחלה /סימפטום חדש מסובכת ומצריכה כתיבת קוד חדש.

וכמובן לא מכלילה מכיוון שהיא מתאימה רק לתסמינים שעליהם בניתי את הקוד לכל מחלה.

תיאור החלופה הנבחרת

מה שבחרתי לעשות זה לאמן מודלי סיווג על מסד נתונים המכיל סימפטומים ואבחון מחלות וכך האלגוריתם יסווג בהינתן רשימת סימפטומים את המחלה בצורה המדויקת והפשוטה ביותר.

הפתרון הוא דינאמי-ניתן לשנות את מסד הנתונים ולהוסיף סימפטומים ומחלות בקלות ומכליל- מודל שמאומן היטב הוא מודל שיכליל לכל סוגי הנתונים והסימפטומים.

והוא ייתן אפשרות למשתמש לקבל את המידע על המחלה בצורה מפורטת וברורה.

לסיכום: החלופה שלי כוללת אתר שנותן למשתמש להכניס את הסימפטומים שהוא חש בשפה חופשית, מעבדת את התסמינים לשפה מובנת למודל, שולחת אותם אל המודל ומאבחנת את המחלה, ואת שם המחלה מחזירה למשתמש, לאחר מכן היא נותנת אופציה לקבל פרטים על המחלה, כך שהיא מחפשת את הפרטים באמצעות ספרייה של פייתון בחיפוש גוגל, מחזירה אותם לאתר והוא מציג אותם בצורה ברורה למשתמש.

אפיון המערכת שהוגדרה

ניתוח דרישות המערכת:

- מעל 95% הצלחה בזיהוי נכון של המחלה.

- איסוף מידע אמין אודות המחלה מחולק לפי נושאים
- סיבוכיות מינימלית ככל הניתן.
- כתיבה בסטנדרטיים מקצועיים, סדר ותיעוד.
- שימוש זמין ונוח.

מודל המערכת:

- קליטת רשימת תסמינים של המשתמש.
- שליחת התסמינים לשרת.
- מציאת מבין התסמינים שקיימים במסד נתונים שעליו אומן המודל את התסמינים הדומים ביותר לרשימת התסמינים שהגיעה מהמשתמש על ידי אלגוריתם לדמיון משפטים.
- שליחת התסמינים למודל ומציאת המחלה.
- שליחת התשובה ללקוח.
- נתינת אופציה ללקוח לקבל פרטים על המחלה המנובאת או מחלה אחרת.
- קבלת שם מחלה בשרת.
- חיפוש ב google search שאלות על מחלות ששמורות במסד נתונים שהכנתי והחזרה ללקוח את התוצאות שהאלגוריתם מגרד מהעמוד של google search .
- שליחת התוצאות ללקוח והצגה בצורה ברורה ומפורטת על המסך.

אפיון פונקציונלי:

- predict_disease()
- disease_search()
- search_text()
- search_image()
- search_defination()

ביצועים עיקריים:

- predict_disease() - קבלת התסמינים מהלקוח ובנית רשימה חד משמעית המובנת למודל, שליחה למודל שינבא את המחלה.
- disease_search() - קבלת שם מחלה מהלקוח ושליחה לחיפוש המידע על המחלה.
- search_text()- חיפוש טקסט בgoogle search על ידי שאילתה לחיפוש.
- search_image() - חיפוש תמונות ב google photo על יד מילה לחיפוש.
- search_defination()- חיפוש הגדרה של מילה בויקיפדיה.

אילוצים:

- כתיבת קוד בפייתון.
- מודל הסיווג צריך לתת תוצאה מדויקת מספיק.
- הפרטים על המחלה אמורים להישלח לצד לקוח בצורה טובה.
- לסיים עד חודש יוני.

תיאור הארכיטקטורה

הארכיטקטורה של הפתרון המוצע בפורמט של Design level Down-Top**:**

הארכיטקטורה בפרויקט הינה בפורמט design-level-down-top, תכנון הפרויקט נעשה מן הכלל אל הפרט. בתחילה הפרויקט תוכנן באופן כללי ,ובכל שלב עברתי לתכנן בצורה יותר פרטנית עד לתוצאה הסופית הנמצאת בידינו.

תיאור הרכיבים בפתרון:

תיאור פרוטוקולי התקשורת:

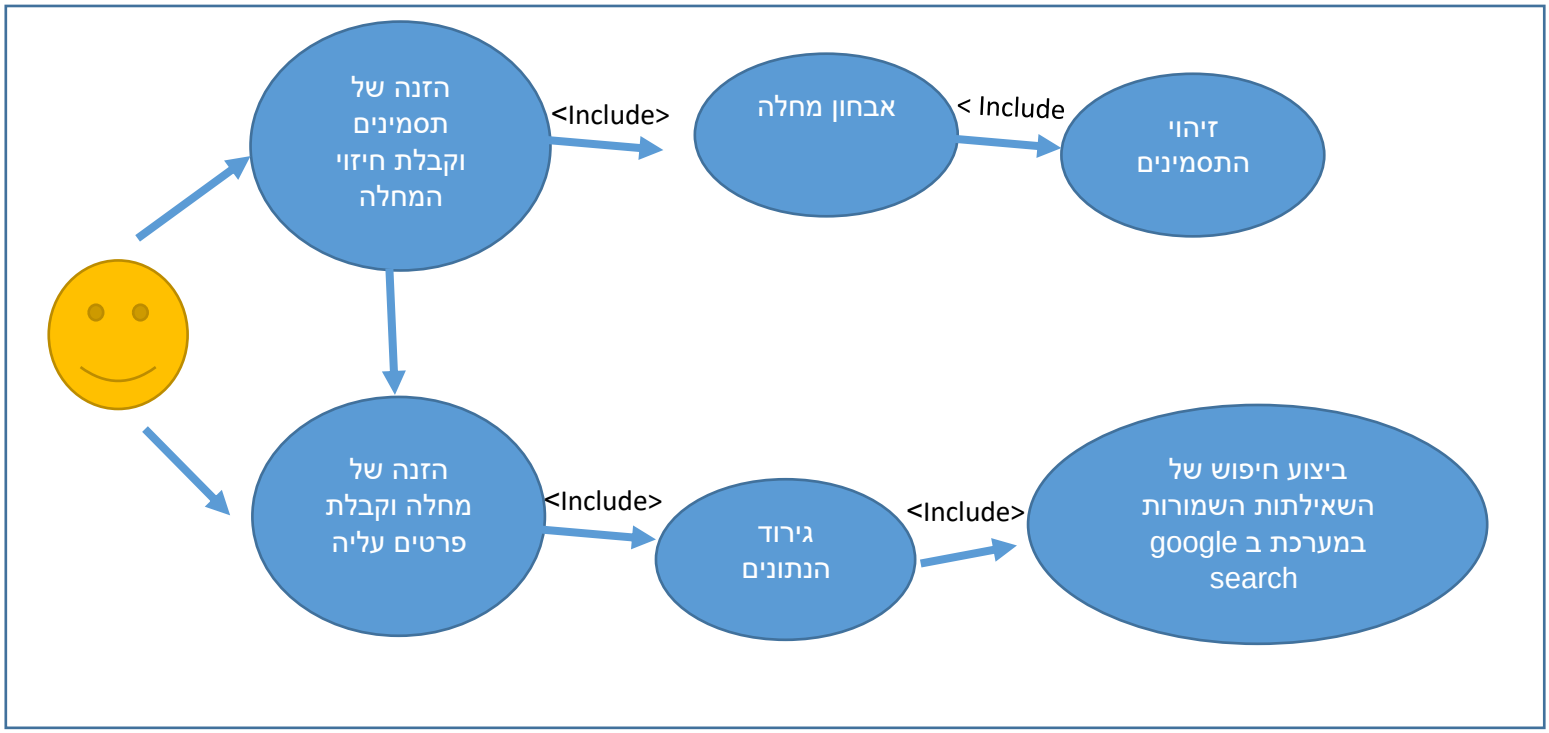
http.

שרת-לקוח:

- טכנולוגיית צד שרת – python.
- טכנולוגיית צד לקוח – react.

ניתוח ותרשים UML / Use cases של המערכת המוצעת

רשימת Use cases **:**



תיאור ה-UC **העיקריים של המערכת:**

-Use Case Documentation

Name: הזנה של תסמינים וקבלת חיזוי המחלה

Identifier: UC1

Description: המשתמש ה -admin מעלה רשימה שמכילה תסמינים בשפה חופשית שהוא מרגיש.

Actors: ה-admin - המפעיל והאחראי על העלאת הרשימה.

Status: בתהליכי עבודה בתיאום הדרישות.

Frequency: כל פעם שהלקוח מעלה.

Conditions-Pre: רשימה תקינה של תסמינים.

Conditions-Post: מחלה מאובחנת.

Extended Use Case: .ל"אמ.

Case User Included:

- זיהוי תסמינים – השרת יקבל את הנתונים ו–באמצעות אלגוריתם לדמיון משפטים הוא בוחר מרשימת 132 הסימפטומים הקיימים במערכת את הסימפטום הדומה ביותר לזה שהמשתמש הכניס.

Case User Included:

אבחון המחלה-השרת מאבחן את המחלה באמצעות מודלי סיווג שאומנו על מסד נתונים (המכיל רשימה של 132 סימפטומים והמחלה), את המחלה, ושולח ללקוח את התוצאה.

Assumptions: הקובץ שהועלה הינו רשימת תסמינים.

Basic Course of Actions:

1. המשתמש נדרש להעלות רשימת תסמינים.
2. לאחר העלאה המשתמש לוחץ על כפתור "predict".
3. המחלה המאובחנת חוזרת למשתמש.

Name: הזנה של מחלה וקבלת פרטים עליה

Identifier: UC2

Description: המשתמש ה -admin מעלה שם של מחלה לקבלת פרטים עליה.

Actors: ה-admin - המפעיל והאחראי על העלאת שם המחלה.

Status: בתהליכי עבודה בתיאום הדרישות.

Frequency: כל פעם שהלקוח שולח.

Conditions-Pre: שם מחלה תקין.

Conditions-Post: רשימת פרטים על המחלה.

Extended Use Case: .ל"אמ.

Case User Included:

- גירוד הנתונים – השרת יקבל את הנתונים ו–באמצעות אלגוריתם לגירוד תוצאות החיפוש בגוגל הוא ייקח את הנתונים ויחזיר למשתמש.

Case User Included:

ביצוע חיפוש של השאילתות השמורות במערכת בgoogle search - השרת יקבל את שם המחלה ויבצע חיפוש בגוגל של שאילתות על המחלה.

Assumptions: המילה שהועלתה היא שם של מחלה.

Basic Course of Actions:

1 .המשתמש נדרש להעלות שם מחלה.

2 .לאחר העלאה המשתמש לוחץ על כפתור "details".

3 הפרטים על המחלה חוזרים למשתמש.

מבני נתונים הבאים לידי שימוש בפרויקט:

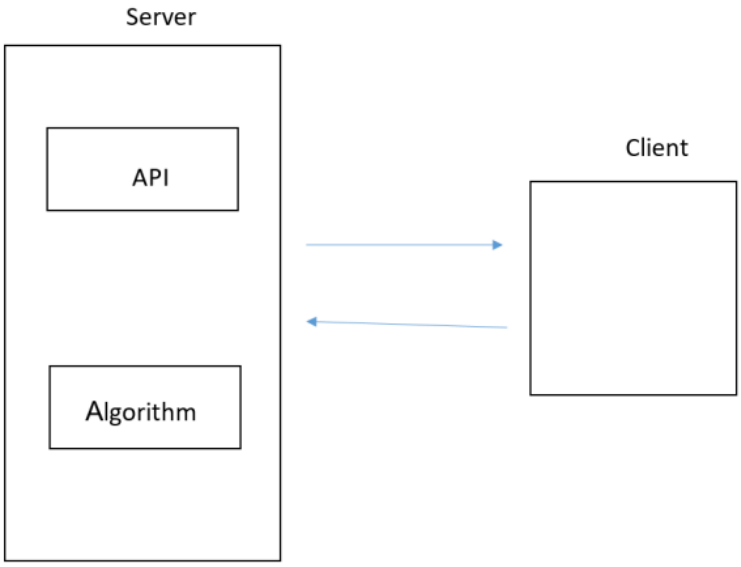
מילון-Dictionary:

להכנסת נתונים למסדי הנתונים ולשמירת הנתונים כ json כדי לשלוח אותם ללקוח.

Pandas:

כדי לקרא ולכתוב למסדי נתונים נשמור אותם במבנה נתונים מסוג pandas .

תרשים מודלים:



תרשים מחלקות:

search_object	DiseasePrediction	function	list_object
search•	_load_train_dataset•	disease_search•	toJSON•
search_text•	_load_test_dataset•	predict_disease•	
search_image•	_feature_correlation•		
search_links•	_train_val_split•		
search_defination•	train_model•		
toJSON•	make_prediction•		

תיאור המחלקות המוצעות:

היחידות הכלולות בפרויקט הם: api, server וclient.

פירוט:

Server:

בתוך יחידה זו כלול כל התהליך הראשי של האלגוריתם.

להלן פירוט על יחידות נבחרות בה:

1. model:

תפקיד: בניית מודלים שידעו לסווג ולזהות על פי תסמינים מחלה.

קלט: מסד נתונים המכיל לכל קלט רשימה של 132 תסמינים כאשר כל אחד מהם מסומן על ידי 0 או 1 ואת המחלה.

פלט: מודלים אותם יצרה המחלקה

2. Function:

תפקיד: יחידה ראשית המפעילה את כל הפונקציונליות של המערכת.

מכילה 2 פונקציות ראשיות שהם:

- disease_search:

מקבלת שם של מחלה ועוברת על מסד הנתונים של השאלות כך שלכל שאלה היא יוצרת אובייקט מסוג search_object ומחפשת על ידי הפונקציות של המחלקה של אובייקט טקסט, תמונה, הגדרה או לינקים ומחזירה רשימה של כל האובייקטים.

קלט: שם מחלה שהזין המשתמש.

פלט: רשימת פרטים על המחלה.

- predict_disease:

מקבל רשימה של תסמינים שהזין המשתמש ועוברת בעבור כל תסמין על כל רשימת 132 התסמינים שקימת במסד הנתונים עליו יתאמנו המודלים, ובודקת מה המרחק בין המשפטים ובוחרת את התסמין עם הקרבה המקסימלית לתסמין שכתב המשתמש כתסמין הנבחר, לאחר מכן המערכת שיצרה רשימה של תסמינים נבחרים מציבה אותם כנבחרים במילון שאותו היא שולחת למודלים שמנבאים את המחלה ומתוך התוצאות היא בוחרת תמחלה שהכי הרבה מודלים בחרו בה.

קלט: רשימת תסמינים שהזין המשתמש.

פלט: מחלה שאותה ניבאו המודלים על פי התסמינים.

3. search_object:

תפקיד: יחידה שמכילה את כל פונקציות החיפוש של המערכת.

קלט: שאילתה לחיפוש וסוג החיפוש הרצוי (לדוג': לינקים, טקסט או הגדרה).

פלט: תוצאות החיפוש וסוג התוצאות (לדוג' מילון, לינקים, או טקסט).

יש עוד פונקציות שמפורטות בחלק 14

חלק שני- API:

תפקיד יחידה זו הוא לקשר בין ה- client ל- server

באמצעות ספריית ה- Flask-Python .

מכיל שתי פונקציות:

- search:

קלט: קבלת שם מחלה שהועלתה על ידי המשתמש.

פלט: החזרה לצד הלקוח את הפירוט על המחלה.

- Predict:

קלט: קבלת רשימת תסמינים שנכתבה על ידי המשתמש.

פלט: החזרה לצד הלקוח את המחלה שנובאה.

חלק שלישי-client:

תפקיד יחידה זאת היא להכין את הממשק למשתמש בצורה נוחה לשימוש.

פירוט על קומפוננטות נבחרות:

- InputSymps:
  קומפוננטה שמציגה שדות קלט לרשימת התסמינים של המשתמש.
- InputSubject:
  קומפוננטה שמציגה שדה קלט להכנסת שם מחלה וכפתור לשליחת שם המחלה לשרת כדי לקבל פרטים עליה.
- AllWindow:
  קומפוננטה שמציגה את התוצאות של חיפוש הפרטים על המחלה למשתמש בצורה ברורה.

תיאור התוכנה

סביבת עבודה:

Pycharm

Visual studio code

שפות תכנות:

פייתון

Jsx/js

אלגוריתמים מרכזיים- תיאור האלגוריתמים במילים ובקוד

- בנית מודל הסיווג:

בתכנות המסורתי באלגוריתם מסווג היינו צריכים לעבוד באמצעות משתני תנאי כך שכאשר יש הרבה משתנים האלגוריתם יהיה ארוך, מסובך ולא מכליל כך שנצטרך לכתוב תנאים רבים בעבור כל מחלה שמבוססים על כל התסמינים האפשריים.

מודלים המבוססים על בינה מלאכותית פועלים כך שהמתכנת מכניס נתונים רבים ככל הניתן והאלגוריתם עובר על כל הנתונים ומנסה להגיע לסוג של הכללה תכנותית שמוציאה את רב הפלטים בצורה נכונה, לדוג' האלגוריתם רואה שכשבשורת הקלט לאדם יש רעד על פי רב המחלה שלו תהיה אלרגיה וכך האלגוריתם מוציא מודל שאליו המשתמש יכניס קלטים והמודל יוציא את המחלה המסווגת.

הקוד:

יצרתי קלאס לחיזוי מחלות:

```
class DiseasePrediction:
    # Initialize
    def __init__(self, model_name=None):
```

טענתי את מסד הנתונים ל-train מסד הנתונים עליו האלגוריתם בודק ובוחר את דרך החישוב שמגיעה מהתסמינים למחלה, ואת מסד הנתונים ל-test שעליו האלגוריתם בודק ובוחן את המודל שלו ואת מידת הצלחתו:

```
# Load Training Data
self.train_features, self.train_labels, self.train_df =
self._load_train_dataset()
# Load Test Data
self.test_features, self.test_labels, self.test_df =
self._load_test_dataset()
# Feature Correlation in Training Data
self._feature_correlation(data_frame=self.train_df, show_fig=False)
```

כמו כן הגדרתי את המודל ואת נתיב השמירה שלו:

```
# Model Definition
self.model_name = model_name
# Model Save Path
self.model_save_path = './saved_model/'
```

כעת הכנתי תפונקציה שבה מתבצעת הטעינה עצמה של מסד הנתונים ל-train

train_features– מכיל את עמודות התסמינים.

train_labels-מכיל את עמודת המחלה בהתאמה לתסמינים.

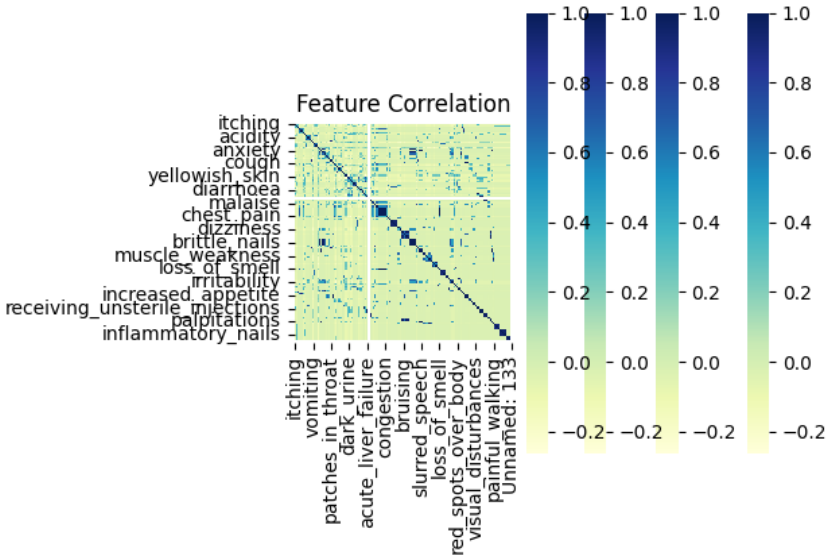
```
# Function to Load Train Dataset
def _load_train_dataset(self):
    df_train = pd.read_csv('./dataset/training_data.csv')
    cols = df_train.columns
    cols = cols[:-2]
    train_features = df_train[cols]
    train_labels = df_train['prognosis']

    # Check for data sanity
    assert (len(train_features.iloc[0]) == 132)
    assert (len(train_labels) == train_features.shape[0])
    print("Length of Training Data: ", df_train.shape)
    print("Training Features: ", train_features.shape)
    print("Training Labels: ", train_labels.shape)
    return train_features, train_labels, df_train
```

וכך גם יצרתי פונקציה לטעינת מסד הנתונים לtest.

וכדי לראות בעין את הנתונים ואת ההתפלגות שלהם, דבר שיעזור לי בניתוח ההגיוני שלהם, ודבר שיעזור לי לראות האם קיימת התפלגות שתאפשר סיווג הגיוני, יצרתי פונקציה שיוצרת גרף המכיל את התפלגות הנתונים:

```
# Features Correlation
def _feature_correlation(self, data_frame=None, show_fig=False):
    # Get Feature Correlation
    corr = data_frame.corr()
    sn.heatmap(corr, square=True, annot=False, cmap="YlGnBu")
    plt.title("Feature Correlation")
    plt.tight_layout()
    if show_fig:
        plt.show()
    plt.savefig('feature_correlation.png')
```

וזהו הגרף הנוצר:



וכעת אפשר לראות שנוצר גרף משורטט ומדויק יחסית דבר שמעיד על נתונים שמתפלגים בצורה הגיונית וכוללנית.

כעת לאחר שייבאנו את הנתונים ובחנו אותם ניתן לחלק אותם בשביל המודל:

```
# Dataset Train Validation Split
def _train_val_split(self):
    X_train, X_val, y_train, y_val = train_test_split(self.train_features,
self.train_labels,
                                                      test_size=0.33,
                                                      random_state=101)
    print("Number of Training Features: {0}\tNumber of Training Labels:
```

```
{1}".format(len(X_train), len(y_train)))
    print("Number of Validation Features: {0}\tNumber of Validation Labels:
{1}".format(len(X_val), len(y_val)))
    return X_train, y_train, X_val, y_val
```

רציתי לאמן כמה מודלים ולא להסתפק באחד כדי ליצור הכללה כמה שיותר מדויקת לכל אחד מהנתונים ובחרתי בשביל הסיווג את המודלים:

- Naive Bayes:

סיווג בייסיאני הוא מודל המסווג אובייקט בעל מאפיינים לאחת מ K-קטגוריות אפשריות.

סיווג בייסיאני עובד על פי חוק בייס:

$$p(y_k|x) = \frac{p(y_k, x)}{p(x)}$$

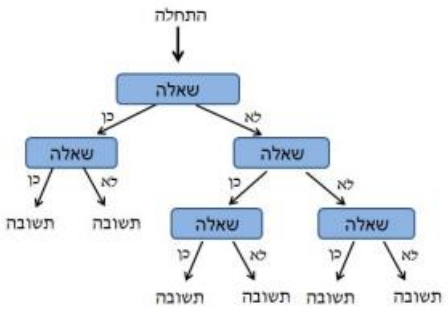
על פי החוק הזה הוא בודק מה ההסתברות בהינתן מאפיינים מסוימים לכל אחת מהקטגוריות ובוחר את הקטגוריה בעלת ההסתברות המקסימלית,

לדוג': בהינתן שורה המכילה תסמינים הוא בודק מה הייתה כל אחת מהמחלות בהינתן התסמינים המסוימים הללו בשאר שורות מסד הנתונים, ובוחר את המחלה שהופיעה הכי הרבה פעמים וההסתברות שתופיעה גם פה היא הגבוהה ביותר.

- decision_tree:

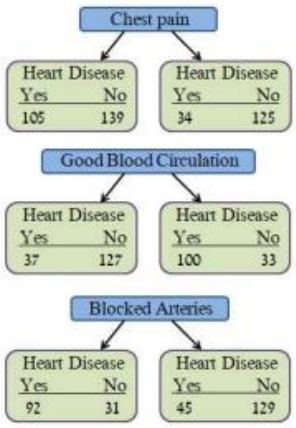
עץ החלטה הינו אלגוריתם לומד היכול לשמש הן לבעיות סיווג והן לבעיות רגרסיה. באופן כללי,

נראה איור של עץ החלטה:



בעץ סיווג כל תצפית תשויך לקבוצה בעל תווית משותפת. למשל, נניח ומעוניינים לסווג מטופל מסוים האם יש לו מחלת לב או לא. אנו יכולים לבנות עץ החלטה על בסיס מאפיינים של חולים שאובחנו בעבר ואנו יודעים להגיד מי מהם באמת חולה לב ומי לא, ועל בסיס העץ

הזה להחליט עבור כל מטופל חדש האם הוא דומה במאפיינים שלו למטופלים שאובחנו בעבר כחולי לב או לא. כך שהתשובה שהעץ נותן היא החלטה - "כן חולה לב" או "לא חולה לב". מלבד החיזוי של התווית, עץ סיווג מספק גם יחסים בין הקבוצות השונות בקרב תצפיות האימון שנופלים באותו אזור. נתבונן בדוגמא שתמחיש את העניין:



באיור לעיל ניתן לראות שלושה פרמטרים (שבמקרה הזה הם סימפטומים של מטופל) בעזרתם מנסים לסווג האם למטופל יש מחלת לב או לא. כפי שניתן לראות אף אחד מהמשתנים אינו יכול לענות על שאלה זו בפני עצמו, כיוון שבאף אחד מהעלים אין אחידות בתצפיות. משתנים כאלה, אשר אינם יכולים בפני עצמם לספק סיווג מושלם, נקראים משתנים לא הומוגניים - לא טהורים כיוון שברוב המקרים כל המשתנים אינם הומוגניים, יש למצוא דרך כיצד לבחור באחד מהמשתנים להיות המשתנה שבראש העץ

המדד שבו השתמשתי באלגוריתם הוא מתבסס על האנטרופיה של העלים:

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i$$

אנטרופיה באה למדוד את השגיאה של התפלגות המשתנה הנבחן מול משתנה המטרה ,הפיצול האופטימלי נבחר על ידי המשתנה בעל מדד האנטרופיה הנמוך ביותר. אם כל התצפיות בעלה מסוים משויכות לאותה קבוצה אזי מדד האנטרופיה יהיה 0 .מאידך, כאשר בעלה מסוים יש התפלגות שווה בין 2 קבוצות של המשתנה המוסבר, מדד האנטרופיה יהיה 1 שזה הערך המקסימלי שמדד אנטרופיה יכול לקבל.

נסביר עם דוגמא שתואמת למסד הנתונים שבו אנחנו משתמשים:

נניח שיש במסד הנתונים 4 שורות של נתונים 2 מתוכם מכילות את התסמין רעד ומסומנות כאלרגיה ו2 את התסמין פצעים ומסומנות אחת כדלקת פרקים ואחת כשפעת וארבעתן

מכילות את התסמין חום, אנחנו נרצה לחלק את הנתונים בצורה הכי מועילה וכעת ננסה לפצל על פי כל אחד מהתסמינים:

כאשר פיצלתי לפי התסמין פצעים קיבלתי אנטרופיה גבוהה כי עדיין לא ניתן לסווג את המחלות עם התסמין פצעים כי החלוקה היא חצי מהנתונים שיש להם את התסמין פצעים הם מחלה אחת וחצי מהם הם מחלה אחרת.

ואם ננסה לפי התסמין חום אני יקבל אנטרופיה זהה למה שהיה לפני החלוקה כי כל ארבעת הנתונים מכילים את התסמין חום.

אך אם נחלק לפי התסמין רעד נקבל את האנטרופיה הכי נמוכה ונוכל לסווג כך שלמי שיש את התסמין רעד הוא חולה באלרגיה וכך נמשיך בחלוקה.

- random_forest:

יער הוא שילוב של כמה עצי החלטה על מנת לקבל תוצאה טובה יותר, במקום לאמן עץ אחד נצטרך לאמן כמה. כמובן שאם נאמן כמה עצים עם אותו מידע נקבל עצים זהים, מכאן נובע שעלינו לפצל את המידע בצורה כלשהי, אנחנו נבחר את המידע באופן רנדומלי. המודל מקבל כפרמטר כמה עצים הוא רוצה-N, לפי כך הוא בוחר רנדומלית כל פעם $\frac{1}{N}$ מכלל המידע ומאמן את העץ על המידע הזה. לאחר שכל העצים מוכנים, נחזה בעזרת כל אחד מהם מה הוא הפתרון.

- gradient_boost:

המודל משפר עצי החלטה על פי פונקציית הפסד.

```
# Model Selection
def select_model(self):
    if self.model_name == 'mnb':
        self.clf = MultinomialNB()
    elif self.model_name == 'decision_tree':
        self.clf = DecisionTreeClassifier(criterion='entropy')
    elif self.model_name == 'random_forest':
        self.clf = RandomForestClassifier(n_estimators=10)
    elif self.model_name == 'gradient_boost':
        self.clf = GradientBoostingClassifier(n_estimators=150,
                                              criterion='friedman_mse')
    print('select model is ',self.model_name)
    return self.clf
```

לאחר מכן הכנתי פונקציית אימון המודלים ושמירה שלהם:

```python
# ML Model
def train_model(self):
    # Get the Data
    X_train, y_train, X_val, y_val = self._train_val_split()
    classifier = self.select_model()
    # Training the Model
    classifier = classifier.fit(X_train, y_train)
    # Trained Model Evaluation on Validation Dataset
    confidence = classifier.score(X_val, y_val)
    # Validation Data Prediction
    y_pred = classifier.predict(X_val)
    # Model Validation Accuracy
    accuracy = accuracy_score(y_val, y_pred)
    # Model Confusion Matrix
    conf_mat = confusion_matrix(y_val, y_pred)
    # Model Classification Report
    clf_report = sklearn.metrics.classification_report(y_val, y_pred)
    # Model Cross Validation Score
    score = cross_val_score(classifier, X_val, y_val, cv=3)
    print('\nTraining Accuracy: ', confidence)
    print('\nValidation Prediction: ', y_pred)
    print('\nValidation Accuracy: ', accuracy)
    print('\nValidation Confusion Matrix: \n', conf_mat)
    print('\nCross Validation Score: \n', score)
    print('\nClassification Report: \n', clf_report)

    # Save Trained Model
    dump(classifier, str(self.model_save_path + self.model_name +
".joblib"))
```

וכעת הפעלתי את הקוד, והנה תוצאות האימון:

```
select model is  mnb
Model Test Accuracy:  1.0
select model is  decision_tree
Model Test Accuracy:  1.0
select model is  random_forest
Model Test Accuracy:  0.9761904761904762
select model is  gradient_boost
Model Test Accuracy:  0.9761904761904762
```

מכיוון שהמודלים יצאו כולם עם accuracy כמעט זהה נשתמש בכולם כאשר לכל אבחון נבחר את התוצאה שנבחרה בהכי הרבה מודלים.

- את הסיווג עצמו כתבתי בפונקציה predict_disease:

בפונקציה אני מקבלת input שהוא רשימה של סימפטומים שמכניס המשתמש,

לאחר מכן האלגוריתם יוצר אובייקט שיכול להישלח למודל:

הוא לוקח את רשימת התסמינים שמוגדרת במסד הנתונים ויוצר ממנה מילון ומשבץ בכולו אפסים, לאחר מכן הוא ממיר את שמות התסמינים לצורה קריאה ומנסה למצוא את התסמינים שהכניס המשתמש ברשימה של התסמינים שטען ממסד הנתונים.

האלגוריתם עובר בעבור כל תסמין שהכניס המשתמש-X על כל רשימת התסמינים שטען ממסד הנתונים-Y ומוצא את התסמין מרשימת ה-Y הכי קרוב ל-x

נמדוד את הדמיון בין כל x לבין כל y על ידי אלגוריתם cosine similarity:

האלגוריתם דבר ראשון יהפוך את המשפטים לווקטורים, יוריד מהם מילים חסרות משמעות במשפט כמו:'the' ו-'is' ולאחר מכן נגדיר את הדמיון באמצעות הפונקציה הבאה:

$$v \cdot w = v^T w = \sum_{i=1}^{N} v_i w_i$$

ונבחר את כל הy שנתנו את תוצאות הפונקציה הגבוהות ביותר לכל x בהתאמה:

לאחר מכן נציב במילון שיצרנו בתחילת הפונקציה בכל אחד מהy שהגדרנו כתוצאות הגבוהות ביותר-1 ונשלח את המילון לפונקציה predict בכל אחד מהמודלים שאימנו, לאחר מכן נמיין את התוצאות והתוצאה שיצאה מספר מקסימלי של פעמים תיבחר ותשלח למשתמש.

הקוד:

```
def predict_disease(input):
    # make list from the str that the client send
    input = input.split(',')
    # make dict from all the 132 symptoms
    symtoms_list = pd.read_csv("dataset/training_data.csv").columns[:-2]
    symptoms = dict(zip(symtoms_list, np.zeros(132)))
    # make the symptoms names as a normal sentences
    symtoms_list = [i.replace('_', ' ') for i in symtoms_list]
```

```
    # Program to measure the similarity between
    # two sentences using cosine similarity.

    X = input
    # create a list of the symptoms that most close to the symptoms that
the client enter
    max_similarity = []
    for x in X:
        if (x != ''):
            x = x.lower()
            max_similarity_acc = 0
            max_similarity_obj = ''
            for y in symtoms_list:
                # tokenization
                X_list = word_tokenize(x)
                Y_list = word_tokenize(y)

                # sw contains the list of stopwords
                sw = stopwords.words('english')
                l1 = [];
                l2 = []

                # remove stop words from the string
                X_set = {w for w in X_list if not w in sw}
                Y_set = {w for w in Y_list if not w in sw}

                # form a set containing keywords of both strings
                rvector = X_set.union(Y_set)
                for w in rvector:
                    if w in X_set:
                        l1.append(1)  # create a vector
                    else:
                        l1.append(0)
                    if w in Y_set:
                        l2.append(1)
                    else:
                        l2.append(0)
                c = 0
```

```
                # cosine formula
                for i in range(len(rvector)):
                    c += l1[i] * l2[i]
                cosine = c / float((sum(l1) * sum(l2)) ** 0.5)
                if (cosine > max_similarity_acc):
                    max_similarity_obj = y
                    max_similarity_acc = cosine
            if (max_similarity_acc > 0):
                max_similarity.append(max_similarity_obj)

    # Set value of 1 corresponding to the symptoms
    for x in max_similarity:
        symptoms[x.replace(' ', '_')] = 1
    # Prepare Test Data
    df_test = pd.DataFrame(columns=list(symptoms.keys()))
    df_test.loc[0] = np.array(list(symptoms.values()))
    clf = ['', '', '', '']
    # Load all pre-trained models
    clf[0] = load(str("./saved_model/random_forest.joblib"))
    clf[1] = load(str("./saved_model/decision_tree.joblib"))
    clf[2] = load(str("./saved_model/gradient_boost.joblib"))
    clf[3] = load(str("./saved_model/mnb.joblib"))
    result = []
    for i in clf:
        result.append(i.predict(df_test))
    # check which disease chose by max models
    chose_disease =
pd.Series(result).value_counts().index.tolist()[np.argmax(list(pd.Series(re
sult).value_counts()))]
    return chose_disease
```

- את אלגוריתם החיפוש יצרתי כך שהשתמשתי בdriver של chrome כדי לחפש בגוגל דרך ספריית selenium, Selenium, מגיעה על מנת לתת פתרון פשוט, נוח, וקל לתפעול לשליטה על דפדפנים (סביבת העבודה של עולם הWeb).

באלגוריתם שלי יצרתי אובייקט לכל שאילתה מסוג search_object, האובייקט מכיל פונקציות לחיפוש שאותם הקוד מפעיל לפי העניין.

הקוד עובד כך שניתן יהיה לקבל פירוט נוסף על דברים,
כך שנניח יש שאילתה המחפשת בתי חולים והיא מוצאת רשימה של בתי חולים, היא תפעיל חיפושים נוספים לכל אחד מהאובייקטים ברשימה, כדי שהמשתמש יוכל לקבל גם כן פרטים על כל אחד מבתי החולים ברשימה, האלגוריתם עושה את זה על ידי יצירת אובייקט מסוג list_object לכל אחד מהאובייקטים ברשימה.

הקוד המפורט יופיע בפרק הבא

קוד התוכנית – על פי סטנדרטים בליווי תיעוד

פונקציות קריטיות /חשובות /עיקריות:

- disease_search:

קלט: הפונקציה מקבלת שם מחלה מהמשתמש.

פלט: הפונקציה מחזירה פרטים שמצאה על המחלה

קוד והסבר:

```
def disease_search(word):
```

קריאת מסד הנתונים של השאילתות שאותם אני רוצה לחפש וקריאת מסד הנתונים שמכיל שמות מחלות כדי לבדוק האם מה שהמשתמש שלח הוא אכן מחלה.

```
    # read the csv with disease word and the csv with the queries
    df = pd.read_csv("word_classification.csv")
    df_calsses = pd.read_csv("calasses.csv")
    word = word.lower()
    result = []
    # check if the word is disease
    for index, i in enumerate(df_calsses['class']):
        for h in df[i].str.find(" " + word + " "):
            if (h != -1 and not math.isnan(h)):
```

כעת האלגוריתם בודק האם חיפשתי בעבר את המחלה הספציפית הזאת. (כדי לחסוך בחיפושים ובזמן האלגוריתם שומר חיפושים קודמים בתוך מסד הנתונים.)

```
# try to find the disease in the csv of the saved old search
                class_csv = pd.read_csv('disease.csv')
                for indexes, z in
enumerate(class_csv['disease'].str.find(word)):
```

```
                    if (not math.isnan(z) and z != -1):
                        class_csv = class_csv.loc[indexes,
:].values.tolist()
                        class_csv.pop(0)
                        for i in range(len(class_csv)):
                            # converting string into dictionary
                            class_csv[i] = ast.literal_eval(class_csv[i])
```

מכיוון שאני לא שומרת בחיפושים הקודמים את ההגדרה מפאת חוסר מקום בקובץ csv אני אחפש רק את ההגדרה מחדש ואחזיר את התוצאה

```
                            if (class_csv[i]['kind'] == 'defination'):
                                class_csv[i] = search_object('defination',
'text').search_defination(word).toJSON()

                        return class_csv
```

אם לא מצאנו את המחלה בחיפושים הקודמים ניצור חיפוש חדש שיכיל הגדרה תמונות ושאילתות.

```
                # search definition,images and all the queries from the
dataset
                result.append(search_object('defination',
'text').search_defination(word).toJSON())
                result.append(search_object('image',
'image').search_image(word).toJSON())
                if isinstance(df_calsses.iloc[index]["search_text"], str):
                    for j in
df_calsses.iloc[index]["search_text"].split(','):
                        result.append(search_object(j,
'text').search_text(j + word).toJSON())
                if isinstance(df_calsses.iloc[index]["search_links"], str):
                    for j in
df_calsses.iloc[index]["search_links"].split(','):
                        result.append(search_object(j,
'link').search_links(j + word).toJSON())
                temp = result
                # add the disease name to the result
                temp.insert(0, word)
                test_keys = 'disease', 'defination', 'images', 'places in
the United States that treat', 'chances of recovery', 'medicines',
```

```
'symptoms', 'risk groups'
```

כעת נכתוב את מה שמצאנו למסד הנתונים המכיל חיפושים קודמים וחזיר את התוצאה למשתמש:

```
                # open the file in the write mode to save this search to the future
                res = dict(zip(test_keys, temp))

                with open('disease.csv', 'a', encoding='UTF8') as f:
                    # create the csv writer
                    writer = csv.DictWriter(f, fieldnames=test_keys)
                    # because the definition detail include large text that can't insert to one cell on csv file we will'not save it
                    temp = res['defination']['result'][0]['details']
                    res['defination']['result'][0]['details'] = ""
                    # write a row to the csv file

                    writer.writerow(res)
                # delete the disease name from the result list
                result.pop(0)
                # put the definition detail back on the result
                result[0]['result'][0]['details'] = temp
                return result
    # if not found this disease
    return False
```

- class search_object:

קלט:

הקלט שנכנס ביצירת אובייקט חיפוש הוא: kind- השאילתה וtype-סוג השאילתה.

פלט:

הפלט הוא אובייקט מסוג חיפוש שמכיל את תוצאות החיפוש שבוצע עליו.

קוד והסבר:

אתחול האובייקט: יצירת webdriver בשביל יצירת חיפוש גוגל והצבת הנתונים מהלקוח.

```
# init object
def __init__(self, kind, type):
```

```
    # open browser
    option = webdriver.ChromeOptions()
    option.add_argument('headless')
    s = Service('chromedriver.exe')
    self.browser = webdriver.Chrome(service=s, options=option)
    self.kind = kind
    self.type = type
```

פונקציה שמקבלת שאילתה לחיפוש בגוגל ומחזירה דף תוצאות מסוג bs4.

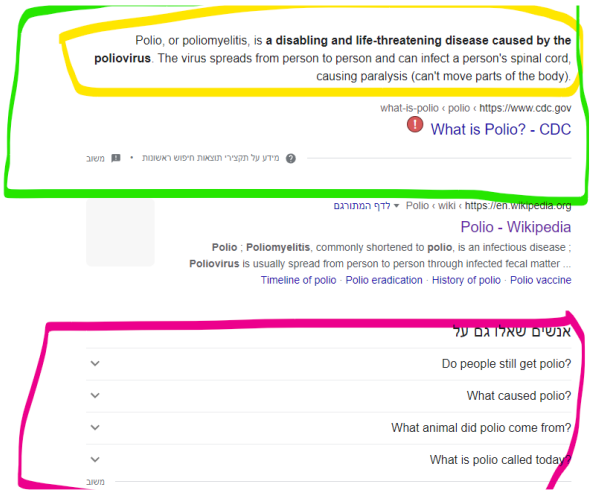
```
# serch in google search
def search(self, search_str):
    search_string = search_str.replace(' ', '+')
    self.browser.get("https://www.google.com/search?q=" + search_string)
    html = self.browser.page_source
    # convert to BeautifulSoup type
    return BeautifulSoup(html)
```

הפונקציה הבאה היא פונקציה מסובכת מעט יותר והיא מיועדת לחיפוש טקסט ב-google search הפונקציה תשתמש באובייקט מסוג list_object שהוא אובייקט המכיל אובייקט, פרטים עליו וסוג הפרטים.

```
# class to object with details
class list_object:
    def __init__(self,browser, item):
        self.item = item
        search = search_object(browser, 'detail', 'text').search_text(item)
        self.type, self.details = search.type, search.result

    # convert to json type
    def toJSON(self):
        dict = {"item": self.item,
                "details": self.details,
                "type": self.type}
        return dict
```

וכעת נפנה לפונקציה עצמה, הפונקציה מקבלת שאילתה לחיפוש ומחזירה אובייקט עם תוצאה, הפונקציה מחפשת בגוגל ובודקת האם קיים בראש העמוד טקסט שמופיע כאשר החיפוש הוא מדויק כאשר הטקסט הזה הוא התקציר של תוצאת החיפוש הראשונה הטקסט מופיע תחת class='yp1CPe' בתוך class='wDYxhc'

ואם לא נמצאה תוצאה מדויקת הוא יבדוק האם קימות שאלות דומות שאולי מדויקות יותר מהשאלה ששלח המשתמש ויכוונו אותו לעמוד עם תוצאות של טקסט ואם קיימות הוא מחפש שוב כך שלושה פעמים, ניתן איור להדגמה:



לפי הציור נראה שהירוק הוא החלק עם המוקף בירוק הוא class='yp1CPe' והמוקף בצהוב הוא הטקסט שאני מחפשת class='wDYxhc' והטקסט המוקף בוורוד זה המקום אליו נגיע אם אין את הטקסט הצהוב.

כמו כן אם נצא טקסט שמופיע בצורת מילון או רשימה לדוגמא רשימת בתי חולים מטפלים אנחנו נרצה עליהם פרטים נוספים ואותם נקבל ע"י יצרת אובייקט שתיארנו קודם שהוא-list_object:

```
# search text by take the text in the header of google search page that
apear if the query is accurate
def search_text(self, search_str):
    query = search_str
    soup = self.search(query)
    # I try to find results 3 times
    for i in range(3):
        # yp1CPe is class name of Summary The first search result from the
Internet
        all_yp1CPe = soup.find_all('div', {'class': lambda x: x and
'yp1CPe' in x.split()})
        if (len(all_yp1CPe) > 0):
            # wDYxhc is the text that include the Summary of The first
search result from the Internet
            all_wDYxhc = all_yp1CPe[0].select('.wDYxhc')
            if (len(all_wDYxhc) > 0):
                # check if the text is list
```

```
                all_li = all_wDYxhc[0].find_all('li')
                if (len(all_li) > 0):
                    list_li = []
                    for i in all_li:
                        # add details to all the object in the list
                        list_li.append(list_object(i.text,
self.browser).toJSON())
                    self.type, self.result = 'list', list_li
                    return self
                # check if the text is a table
                all_td = all_wDYxhc[0].find_all('td')
                if (len(all_td) > 0):
                    dict_td = dict()
                    i = 0
                    while i < len(all_td):
                        i = i + 1
                        if (all_td[i + 1].text in dict_td):
                            j = 1
                            while all_td[i + 1].text + ' ' + str(j) in
dict_td:
                                j += 1
                            # add details to all the object in the table
                            dict_td[all_td[i + 1].text + ' ' + str(j)] =
list_object(all_td[i].text).toJSON()
                        else:
                            # add details to all the object in the table
                            dict_td[all_td[i + 1].text] =
list_object(all_td[i].text).toJSON()
                        i = i + 2
                    self.type, self.result = 'dict', dict_td
                    return self
                # check if the text is a text that represented by span
                all_span = all_wDYxhc[0].find_all('span')
                if (len(all_span) > 0):
                    all_text = ''
                    text = all_span
                    for info in text:
                        all_text += " " + info.text
                    self.type, self.result = 'text', all_text
```

```
                    return self

                # check if the text is a text that represented by div
                all_div = all_wDYxhc[0].find_all('div')
                if (len(all_div) > 0):
                    all_text = ''
                    text = all_div
                    for info in text:
                        all_text += " " + info.text
                    self.type, self.result = 'text', all_text
                    return self

        # Wt5Tfe is query that close to the meaning of the orginal query
        all_Wt5Tfe = soup.find_all('div', {'class': lambda x: x and
'Wt5Tfe' in x.split()})
        if (len(all_Wt5Tfe) > 0 and len(all_Wt5Tfe[0].select('.wQiwMc')) >
0):
            # query is the new search
            query =
all_Wt5Tfe[0].select('.wQiwMc')[0].select('span')[0].text
            # search the new query
            soup = self.search(query)
        else:
            break
    # if the program did not found results
    self.type, self.result = False, False
    return self
```

הפונקציה הבאה היא פונקציה לחיפוש תמונות על הנושא בgoogle-photo

בפונקציה נחפש את שם המחלה ונחזיר 5 תמונות שמצאנו עליה:

```
# search images by google image search
def search_image(self, search_query):
    search_url =
f"https://www.google.com/search?site=&tbm=isch&source=hp&biw=1873&bih=990&q
={search_query}"
    images_url = []
    # open browser and begin search
    self.browser.get(search_url)

    html = self.browser.page_source
```

```
    soup = BeautifulSoup(html)

    elements = soup.find_all('img', {'class': lambda x: x and 'rg_i' in
x.split()})
    count = 0
    # find maximoom 5 image and return
    for e in elements:
        images_url.append(e['src'])
        count += 1
        if count == 5:
            break
    self.type, self.result = 'image', images_url
    return self
```

הפונקציה הבאה מחפשת הגדרה על המחלה, החיפוש מכיל הגדרה מקוצרת ופרטים מפורטים, כאשר ההגדרה המקוצרת מתקבלת עם חיפוש טקסט פשוט והפרטים הארוכים על ידי חיפוש בויקיפדיה והפיכת עמוד הויקיפדיה לקריא והחזרת אובייקט מסוג מילון:

```
# search defination by google search and wikipedia
def search_defination(self, query):
    # search wikipedia for the any term
    wikipedia_api_link =
"https://en.wikipedia.org/w/api.php?format=json&action=query&list=search&sr
search="
    search_term = query
    url = wikipedia_api_link + search_term
    # get the url of the first article that the search provides
    r = ''
    while r == '':
        try:
            r = requests.get(url)
            break
        except:
            print("Connection refused by the server..")
            print("Let me sleep for 5 seconds")
            print("ZZzzzz...")
            time.sleep(5)
            print("Was a nice sleep, now let me continue...")
            continue
```

```
    json_output = r.json()
    # if block by netfree
    if ("blockByNetFree" in json_output):
        self.type, self.result = False, False
        return self
    if not json_output['query']['search']:
        self.type, self.result = False, False
        return self
    article_title = json_output['query']['search'][0]['title']
    article_title = article_title.replace(' ', '_')
    wikipedia_link_article = "https://en.wikipedia.org/wiki/" +
article_title

    # scrape the HTML content from the page
    def request_webpage(url):
        res = ''
        while res == '':
            try:
                res = requests.get(url)
                break
            except:
                print("Connection refused by the server..")
                print("Let me sleep for 5 seconds")
                print("ZZzzzz...")
                time.sleep(5)
                print("Was a nice sleep, now let me continue...")
                continue

        try:
            res.raise_for_status()
        except:
            print('There is a problem with the request')
        return res

    page = request_webpage(wikipedia_link_article)
    bs_page = BeautifulSoup(page.text, 'html.parser')
    # get only specific HTML tags
    tags = bs_page.find_all(["h1", "h2", "p"])
    # extract the text from these tags
```

```
    text = ''
    for tag in tags:
        text += tag.getText()
    # sanitation
    import re
    # replace new lines
    text = text.replace('\n', ' ').replace('\r', ' ')
    # remove non-ascii characters
    import string
    printable = set(string.printable)
    ''.join(filter(lambda x: x in printable, text))
    # use regex to remove '[x]' -
    # reference links in the wikipedia text
    text = re.sub(r'\[\d+?\]', '', text)
    # separate into list of sentences
    text = re.sub(r'[.\?!]', "#eos#", text)
    sentences = text.split('#eos#')
    sentences = [item.strip() + '.' for item in sentences]

    # limit the len to 400 words
    MAX_LEN = 400

    paragraphs = ['']
    x = 0
    for sentence in sentences:
        sentence_len = len(sentence.split())
        paragraph_len = len(paragraphs[x].split())
        if (paragraph_len + sentence_len) <= MAX_LEN:
            paragraphs[x] += ' ' + sentence
        else:
            paragraphs.append(sentence)
            x += 1
    paragraphs = ' '.join(paragraphs)
    # convert to json type
    dict = {"item": self.search_text(query).result,
            "details": paragraphs,
            'type': 'text'}
    self.type, self.result = 'list', [dict]
    return self
```

הפונקציה הבא מחפשת לינקים על ידי לקיחת 10 התוצאות הראשונות בחיפוש גוגל:

```
# search link by googleSearch tool
def search_links(self, search_str):
    from googlesearch import search
    query = search_str
    my_results_list = []
    # find 10 first links
    for i in search(query,  # The query you want to run
                    tld='com',  # The top level domain
                    lang='en',  # The language
                    num=10,  # Number of results per page
                    start=0,  # First result to retrieve
                    stop=10,  # Last result to retrieve
                    pause=2.0,  # Lapse between HTTP requests
                    ):
        my_results_list.append(i)
    if (len(my_results_list) > 0):
        self.type, self.result = 'link', my_results_list
        return self
    self.type, self.result = False, False
    return self
```

המרת האובייקט לאובייקט מסוג json בשביל שליחה לצד לקוח בצורה קריאה:

```
# convert to json type
def toJSON(self):
    dict = {"kind": self.kind,
            "type": self.type,
            "result": self.result}
    return dict
```

תיאור מסד הנתונים

- מסד הנתונים בשביל בנית המודל:

מסד הנתונים מכיל 133 עמודות מתוכן 132 עמודות של תסמינים כאשר ערך כל עמודה הוא בוליאני 0/1 -התסמין מופיע/לא מופיע והעמודה האחרונה מכילה את האבחנה-המחלה

המאובחנת. מסד הנתונים מחולק ל2 קבצים train ו-test כשביל תהליך האימון של המודל והניסוי שלו.

- מסד נתונים המכיל את רשימת השאילתות:

כדי ליצור אפשרות שינוי פשוטה של שאילתות יצרתי מסד נתונים המכיל שאילתות המחולקות לעמודות הבאות:

Search_link-שאילתות לחיפוש לינקים.

Search_text-שאילתות לחיפוש טקסט.

- מסד נתונים המכיל רשימת שמות מחלות:

שאבתי את השמות מאתר שמכיל שמות מחלות Z-A ומתוך מסד הנתונים שהשתמשתי בבניית המודל שלי.

מסד הנתונים נועד לבדיקת תקינות של שמות המחלה שנשלחות מצד הלקוח.

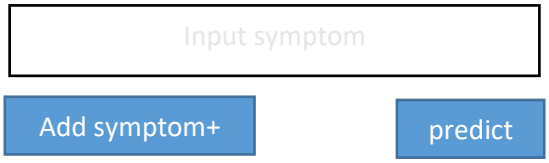
תיאור מסכים ומדריך למשתמש

המסך שנפתח הוא מסך המכיל שני אזורים:

האחד מכיל שדה להכנסת תסמינים ושני כפתורים: אחד להוספת תסמין ואחד לאבחון המחלה.

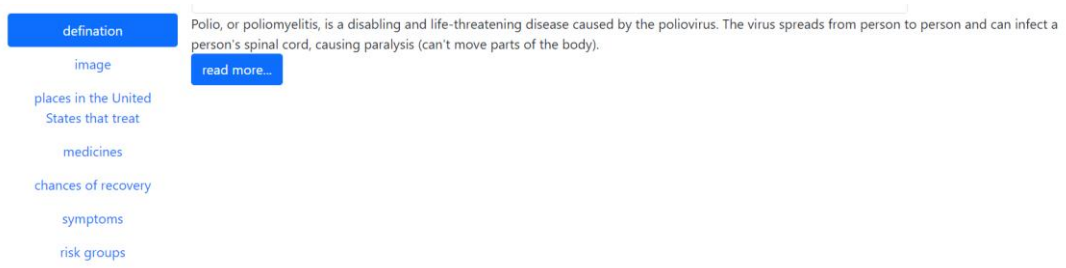
והשני מכיל שדה להכנסת מחלה וכפתור לקבלת פרטים על המחלה.

האזור הראשון נראה כך:



והאזור השני נראה כך:

לאחר שמכניסים את שם המחלה ולוחצים על details התשובה מגיעה מהשרת והמסך של הפרטים שמוצגים ללקוח נראה כך:



כאשר המסך הזה בעצם מכיל את כל המידע בחלוקה לנושאים, כאשר כל לחיצה על הכפתורים בצד יציג למשתמש נושא אחר

וכמו כן למשתמש יש אופציה נוספת כאשר מופיע לו הכפתור read more ללחוץ עליו ולקבל פרטים נוספים.

בדיקות והערכה

- האלגוריתם המרכזי שדרש בדיקות נכונות הוא כמובן אלגוריתם הסיווג, את בדיקת ההערכה עליו עשיתי באמצעות מסד הנתונים test שמכיל ערכים ותוצאת וכך בדקתי שהערכה של המודל אכן טובה מספיק.

כפי שניתן לראות בתוצאות הaccuracy של ההרצה שלו על הtest data התוצאה ממש גבוהה ונעה בין 0.9-1

```
select model is  mnb
Model Test Accuracy:  1.0
select model is  decision_tree
Model Test Accuracy:  1.0
select model is  random_forest
Model Test Accuracy:  0.9761904761904762
select model is  gradient_boost
Model Test Accuracy:  0.9761904761904762
```

ניתוח יעילות

סיבוכיות השימוש במודל:

לאחר שהמודל בנוי- סיבוכיות השימוש בו היא קבועה. זו בדיקה פשוטה בלבד.

סיבוכיות החיפוש בגוגל:

הסיבוכיות עצמה גבוהה, כי החיפושים עורכים זמן רב, אבל כאשר הוספתי את מסד הנתונים ששומר חיפושים קודמים סיבוכיות המקרה הממוצע קטנה בהרבה והתוצאות מגיעות תוך זמן קצר.

מסקנות

במהלך חיפוש רעיון לפרויקט עברתי דרך רעיונות רבים וניסיונות של אלגוריתמים שונים, ובעקבות תהליך החיפוש הזה גם אם לא השתמשתי בסופו של דבר באלגוריתמים אותם למדתי והתנסיתי זכיתי בלימוד של חומר שלא הייתי נתקלת בו אם לא הפרויקט, והמסקנה הראשונה שלי היא שהדרך חשובה לא פחות מהתוצאה, ולמרות שהדרך שלי הייתה ארוכה אף יותר מהפרויקט עצמו היא הייתה בעלת משמעות לא פחותה מהפרויקט.

מסקנה נוספת היא שלמרות שמושך להתחיל לעבוד מהקל לכבד, מהפונקציות הפשוטות ורק אחר כך לעבוד על האלגוריתמים המסובכים, מכיוון שרק לאחר כתיבת האלגוריתם הראשי אפשר לדייק את הפונקציות, כאשר עובדים בדרך הפוכה חלק גדול מהפונקציות דורש שכתוב מחדש ולעיתים אף מחיקה, דבר שיוצר עבודה כפולה.

מסקנה שלישית היא משהו שעזר לי מאוד בתהליך בנית הפרויקט, עבודה רציפה ללא הפסקות. כאשר עובדים באופן לא רציף עם הפסקות זה אולי נשמע קל יותר ויעיל, אך עבודה אמיתית וחיסכון בזמן ובכח נוצרים כאשר עובדים כמה שיותר רציף ובכמה שפחות הפסקות, ראיתי עבודה שעם הפסקות יכלה לקחת שבוע וכאשר היא נעשתה בלי הפסקות היא לקחה 4 שעות לכל היותר.

כמו כן העשרתי את הידע שלי וקיבלתי הרבה דברים בתהליך הפרויקט.

פיתוחים עתידיים

בעתיד ניתן לפתח את הפרויקט על ידי הוספת יעוץ באמצעות צ'אט בוט שמבוסס על בינה מלאכותית, דבר שייתן חוויה טובה יותר למשתמש, כמו כן ניתן להגדיל את מסד הנתונים על ידי משוב מהלקוחות דבר שישפר את האבחון.

בבליוגרפיה

מסדי נתונים:

Kaggle

מודלי סיווג:

רשתטק

https://github.com/AvrahamRaviv/Deep-Learning-in-Hebrew

:Python

stackOverflow

W3 Schools

/https://www.geeksforgeeks.org

:React

W3 Schools