חיזוי תוצאות משחקי כדורגל לשנת 2019/20 על סמך שלוש העונות האחרונות

פרויקט בבינה מלאכותית 236502

הפקולטה למדעי המחשב , טכניון

מגישים :

מוסא שאער

315651893

Mousa.shaer@campus.technion.ac.il

מוחמד איוב

318548666

Ayoub@campus.technion.ac.il

מבוא:

ספורט היה חלק מחיי האדם במשך אלפי שנים, והעניין להיות יותר מסתם קהל פשוט מחזיר אותנו לפני יותר מ -2000 שנה, אז היוונים הלכו לקולוסיאום כדי להמר על גלדיאטור שבחרו. מאז אותה תקופה גבר העניין לנסות להכיר את העתיד במשחקי הספורט.

לדוגמא, פרשנים בתכניות טלוויזיה מנחשים איזו קבוצה תזכה בסופר בול או מנסים להבין את התוצאה של משחק בליגת העל, אנשים בתקשורת דנים מי יזכה באליפות העולם בטניס. זה כבר חלק מחיינו.

בנוסף, ענף ההימורים הספורטיביים גדל עד כדי כך שבשנת 2019, סך ההכנסות הגולמיות של ענף ההימורים הסתכם ב -85 מיליארד דולר [1]. העניין לדעת את התוצאה של משחקי ספורט לפני סיומם ברור כעת וחיוני לתעשיית הימורי הספורט.

מכיוון שלכדור רגל יש מספר גדול מאוד של מאפיינים שקשורים ישירות לתוצאה, קשה לבני אדם להתחשב בכל התכונות ולחזות בדיוק רב את תוצאות המשחק. במצבים אלה, יש צורך בטכניקה בעלת ביצועים גבוהים בכדי להתמודד עם כל הנתונים, וכאן נכנסת הבינה המלאכותית. ההתקדמות העצומה בטכנולוגיה זו מאפשרת לעבד כמות מופרזת של נתונים כדי להסיק מסקנות שימושיות ביותר.

המטרה העיקרית של פרויקט זה היא לחקור טכניקות למידה שונות על מנת לחזות את התוצאה של משחקי כדורגל, באמצעות אירועי משחק ודירוגי השחקנים לפי אתר הFIFA .

הגדרת הבעיה:

בפרויקט הזה בחרנו להתמקד בחיזוי תוצאות המשחקים בליגה האנגלית לעונה 20\2019. כלומר בהינתן שתי קבוצות (קבוצת בית וקבוצת חוץ) המודל שלנו מנסה לחזות את התוצאה של המשחק ומחזיר אחת התוצאות הבאות:

H- הקבוצה הביתית ניצחה.

A- קבוצת החוץ ניצחה.

D- המשחק ניגמר בתיקו.

תיאור הפתרון המוצע לבעיה:

במסגרת פרויקט זה החלטנו לנסות להתמודד עם בעיה זו כבעיית למידה בה על סמך משחקים קודמים ונתונים קיימים ננסה לחזות את התכונות והסטטיסטיקה של המשחק ואז על סמך התכונות האלה ננסה לחזות את התוצאה של המשחק.

הרעיון שעמד מאחורי החלטה זו היא שרואים שבתוכניות טלוויזיה מנסים לחזות מה הולך לקראת במשחק על פי הנתונים שיש להם ומניסיון שלהם בניתוח משחקים קודמים ולכן יתכן שגם אלגוריתם למידה ידע לחזות איך ילך המשחק ואז יוכל לחזות את התוצאה.

הפתרון מחולק ל 3 שלבים עיקריים :

1) איסוף ויצירת תכונות שיעזרו לנו בשלבי הלמידה, שזה כולל עיבוד מקדים לנתונים וחישוב תכונות חדשות.
2) למידת תכונות שמקבלים תוך כדי המשחק עצמו (לדוגמא מספר קרנות...) ע"י רשת עצבים.
3) שימוש התכונות שחושבו ונוצרו בשלב 1 ביחד עם התכונות שחוזים בשלב 2 באלגוריתמי למידה כדי לקבל את התוצאה הסופית שאנו מנסים לקבל (ניצחון/תיקו/הפסד).

נפרט על זה בהמשך.

- **<u>איסוף מידע:</u>**

החלק המאתגר ביותר בפרויקט הוא איסוף המידע.
חיפשנו בכל מני אתרים על מידע שיהיה עדכוני ואמין על מנת לקבל את התוצאות הכי טובות.
בסיס הנתונים שמצאנו ראשון היה מכיל סטטיסטיקה על משחקים משנת 2016 עד 2020 המידע היה מכיל תאריך המשחק, קבוצת בית וקבוצת חוץ, בנוסף עבור כל קבוצה היה מכיל את מספר השערים, הקרנות, הבעיטות למסגרת ולחוץ המסגרת, כרטיסים צהובים ואדומים.
הנתונים האלו לא היו טובים מספיק בשבילנו כי הם מייצגים נתונים של משחק שכבר נערך ואנחנו היינו רוצים לחזות את התוצאות על סמך נתונים אחרים שקרו לפני המשחק, כמו למשל:

1) ההרכב של כל קבוצה, ידוע שכל קבוצה מכריזה על ההרכב שהולך לפתוח את המשחק שעה קודם מתחילת המשחק.
2) סטטיסטיקה על השחקנים של כל קבוצה, ופה בחרנו להסתמך על הנתונים של אתר ה-FIFA כי הם מאוד משקפים את היכולת של השחקנים.

ופה היינו צריכים להשתמש בשיטת web scrapping על מנת להוציא מידע מאתרי אינטרנט, כי לא מצאנו אתר שמכיל את הנתונים שאנחנו צריכים ולכן כתבנו קוד שקורא את הנתונים שאנחנו צריכים מאתרי אינטרנט מסוימים ואז מסדר לנו את המידע בקבצי אקסל שנוח לעבור איתם.

החלק הזה היה קצת מורכב כי זה פעם ראשונה שלנו שהיינו משתמשים בשיטת web scrapping והיינו צריכים ללמוד את זה בעצמנו, וכל זה היה אחרי שבזבזנו הרבה זמן לחפש אתרים שמכילים את המידע שמחפשים ולבדוק שאכן המידע נכון.

לכן הצלחנו בשיטה הזו למצוא עוד מידע עדכוני ואמין שמכיל:

ההרכב שכל קבוצה פתחה בו את המשחק, את השיטה שמשחקת בה כל קבוצה (כמה שחקנים בחלק הקדמי, באמצע המגרש ובהגנה) בנוסף הצלחנו למצוא את דירוגי ה FIFA של כל השחקנים כמו דירוג הכללי שלו, דירוג המהירות והיכולת הפיזית, דירוג של יכולת ההגנה וההתקפה ועוד.

אחרי שמצאנו את הנתונים האלו יצרנו תכונות חדשות שיעזרו לנו לחזות את הסטטיסטיקה בתוך המשחק עצמו. כמו למשל התכונות שמצאנו במאגר הנתונים הראשון (מספר בעיטת למסגרת, קרנות, כרטיסים אדומים וצהובים ועוד...), ויצרנו את התכונות הבאות:

1) רצינו קודם לקבל מידע על התקופה האחרונה של כל קבוצה ומה המצב שלה בליגה לכן ממאגר הנתונים הראשון שלנו עבור כל משחק בדקנו מה היה המצב של שתי הקבוצות בשלושת המשחקים האחרונים.
ולכן לקחנו ממוצעים של הסטטיסטיקות של כל קבוצה בשלושת המשחקים האחרונים כמו למשל ממוצע השערים שכבשה , ממוצע השערים שספגה, ממוצע הקרנות, ממוצע הכרטיסים הצהובים והאדומים ומספר הנקודות שכל קבוצה הצליחה להשיג.

2) בנוסף רצינו להוסיף תכונות על סמך העבר של המפגשים הישירים בין שתי הקבוצות ולכן יצרנו אותם תכונות אבל עכשיו עבור שתי המשחקים האחרונים הישירים בין שתי הקבוצות.

3) הוספנו עוד תכונות שתלויה בהרכב שכל קבוצה פתחה בו למשל:

- ממוצע המהירות של החלק הקדמי (שישקף את הכוח של החלק ההתקפי של הקבוצה).
- את הממוצע של הכוח הפיזי של הקשרים והמגנים שיכול לשקף את איכות ההגנה של הקבוצה.
- ההפרש בין ממוצע דירוג החלק ההתקפי לממוצע דירוג החלק האחורי.

בסיום השלב הזה קיבלנו את התכונות שיכולות לעזור לנו לחזות את הסטטיסטיקה של שתי הקבוצות בתוך המשחק, ולכן בנינו רשת נירונית שמקבלת ויקטור של התכונות שיצרנו ומחזירה ויקטור תכונות המייצג את החיזוי שמה הולך לקרות במשחק עצמו, נפרט על זה בהמשך.

לסיכום קיבלנו מידע אמין עבור שלושת העונות האחרונות כלומר עבור 1080 משחקים.

- **<u>עיבוד מקדים של הדוגמאות:</u>**

השלב הזה גם היה מורכב מכמה סיבות כאשר העיקריות בהן היו:

1. אספנו מידע ממקורות שונים לכן קיבלנו מידע שונה, למשל שמות הקבוצות והשחקנים הייתה שונה במסדי נתונים שונים.
את ההבדל בין שמות הקבוצות הצלחנו לסדר דרך פונקציה שלוקחת את שמות הקבוצות באחד הקבצים ועוברת על שאר הקבצים ומחליפה את השמות לפי מלון שנתנו לה, היה סך הכל 26 קבוצות שונות לכן היה ניתן למצוא mapping לשמות הקבוצות בקבצים השונים.

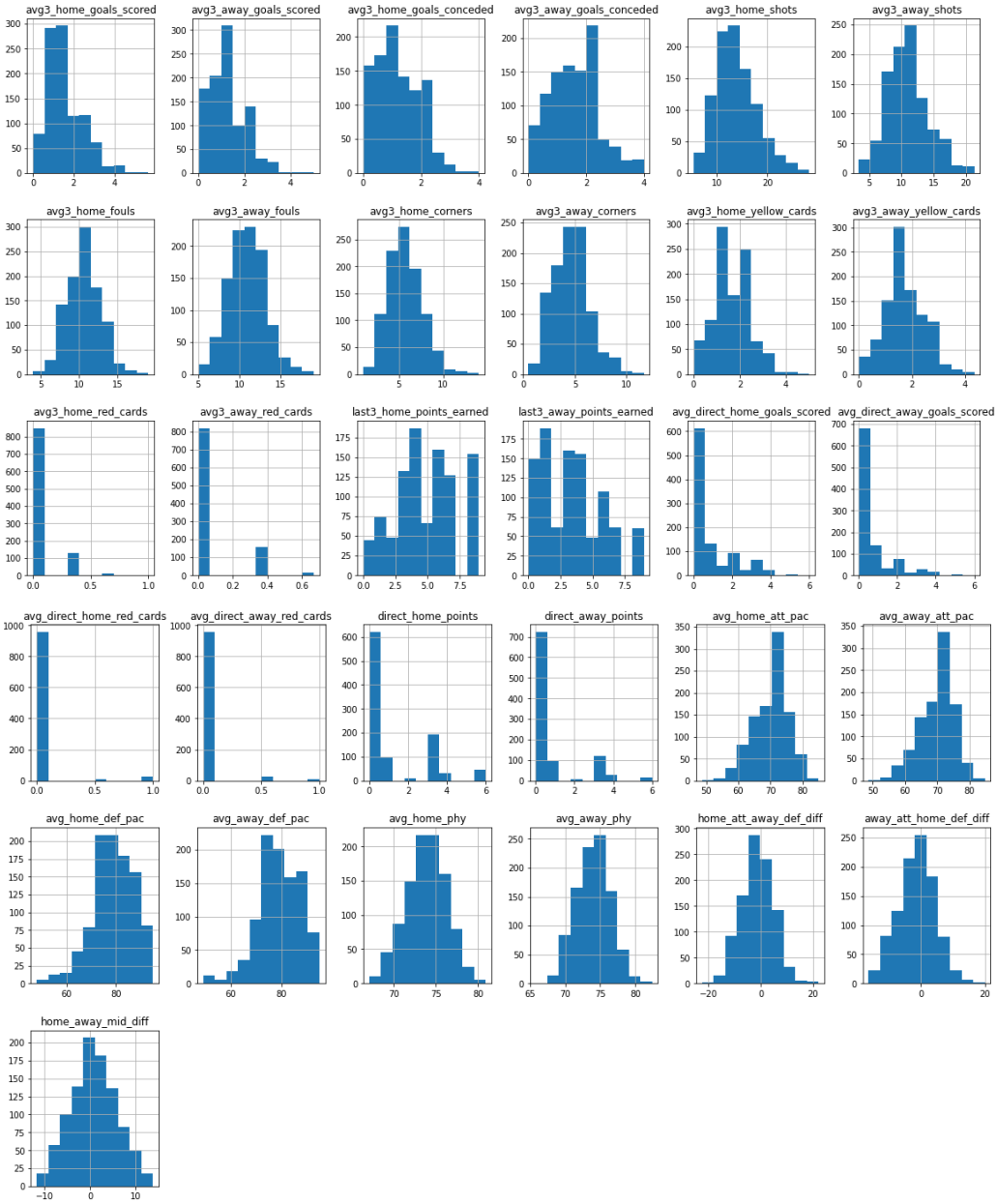
את ההבדל בין שמות השחקנים בקבצים השונים היה יותר מאתגר כי בכל קבוצה יש יותר מ 20 שחקנים ויש 20 קבוצות לכן היה מעל 400 שחקנים בכל עונה ולא היה אפשרי למצוא mapping לכן לא החלפנו את השמות של השחקנים בין הקבצים השונים אלא בנינו פונקציה שתעזור לנו במציאת השחקנים והיא עובדת באופן הבא:
קודם כל לכל שחקן במאגר המידע שלנו יש גם הקבוצה שהוא משחק בה ולכן יכלנו לצמצם את החיפוש מ מעל 400 שחקנים לתוך 30-20 שחקן ואז על יד פונקציית קירוב שמחזירה את ההתאמה הקרובה ביותר הצלחנו להחזיר את השחקן המבוקש, למשל עבור השחקן מסעוד אוזיל שמשחק בארסינל באחד הקבצים הוא כתוב בצורה הזו M. Özil ובקובץ אחר הוא כתוב כך Mesut Özil אז הפונקציה מחפשת בשחקני ארסינל את שם השחקן הכי קרוב על ידי השוואת מחרוזות ומחזירה אותו.
בנוסף עשינו ניסויים וודינו שהפונקציה עובדת כישר.

2. גם התאריכים היו שונים בקבצים שונים כלומר כל אחד היה בפורמט שונה לכן גם עשינו פונקציה שמאחדת את הפורמט בקבצים השונים.

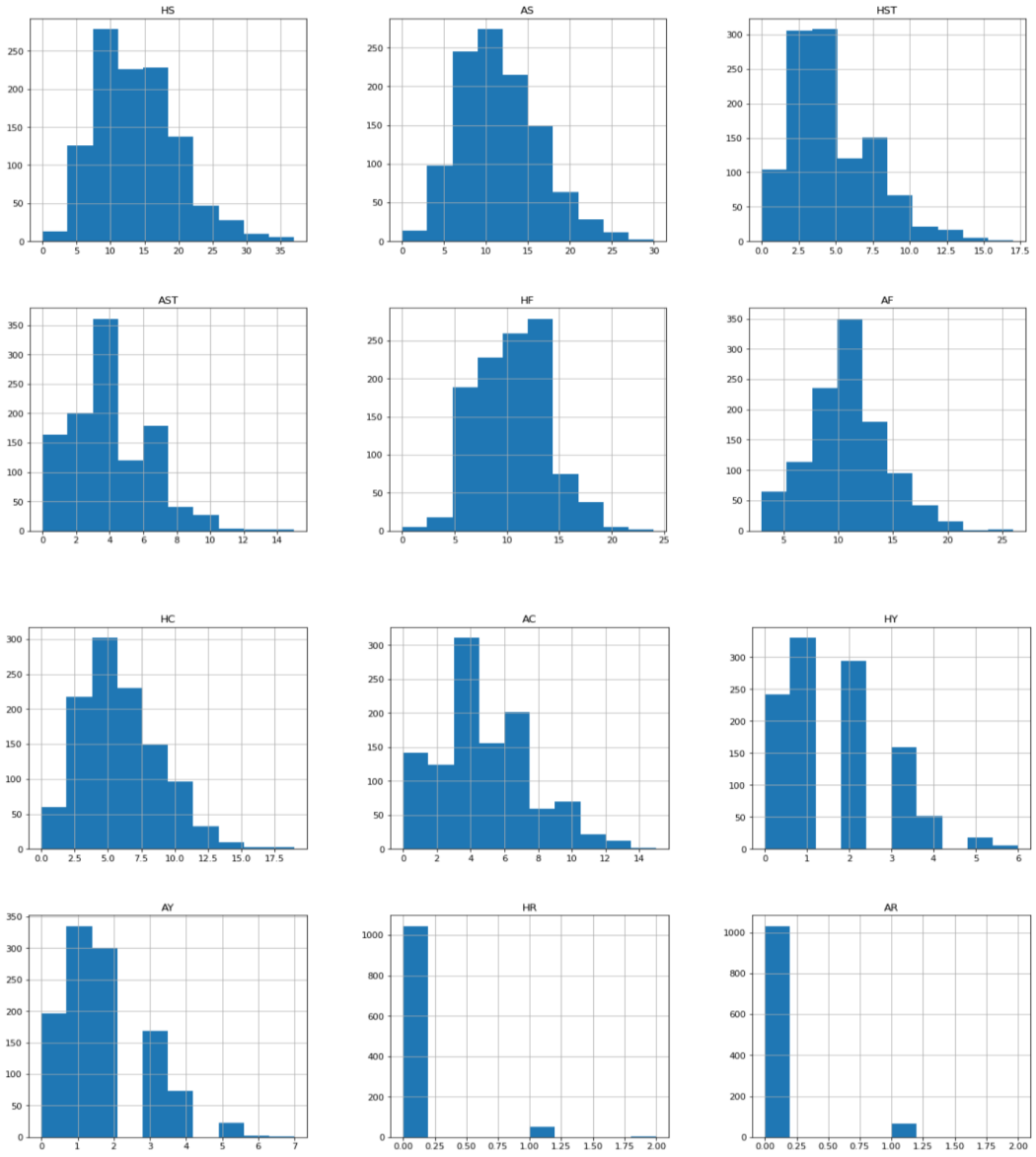
3. עוד בעיה שפגשנו הייתה חלון ההעברות, בתקופה מסוימת במהלך העונה יהיה אפשרות לשחקנם לעבור מקבוצה אחת לאחרת ואז יהיה לנו בעיה למצוא את השחקנים. ולכן החלטנו לאסוף את המידע עבור השחקנים לפני ואחרי חלון ההעברות ואז לחפש את השחקן

- **סטטיסטיקות:**

התפלגות התכונות :



רואים את התפלגות התכונות שמצאנו ויצרנו לשלב ראשון שמשמשים לחיזוי התכונות שמתרחשים במשחק עצמו, וגם לשלב שני בו חוזים את התוצאה הסופית איך שהגדרנו את הבעיה.
ניתן לראות כי רוב התכונות מתפלגים נורמלית והשאר מתרכזים בטווח מאוד מצומצם, כדי לעשות data normalization, האם להשתמש ב z-score or min-max (normal distribution), וזה חשוב לרשת נוירונית בגלל השגיאות הנומריות וגם לאלגוריתמי הלמידה.



וגם התפלגות התכונות שמתרחשים במשחק ומשמשים לשלב 2 של חיזוי התוצאה הסופית.
חשוב ל data normalization לסיבות הנ"ל.

- **<u>למידה:</u>**

שלב הלמידה השתמשנו באלגוריתמי למידה שונים על מנת לבנות מסווג שבהינתן משחק לא מסווג ידע לחזות את התוצאה של המשחק.
את האלגוריתמים האלה אימנו בשיטת cross validation כאשר חלקנו את מאגר הדוגמאות ל 5 קבוצות ובכל איטרציה האלגוריתם בוחר קבוצה מתוך חמש הקבוצות להיות קבוצת המבחן והשאר משמש לאימון, ואז מחזיר את ממוצע הדיוק עבור 5 האיטרציות.
חשוב לציין שבשלב הזה הרצנו את המסווגים עם פרמטרים שונים על מנת לקבל דיוק הטוב ביתר נפרט על זה בהמשך.

- **<u>דילול המאפיינים :</u>**

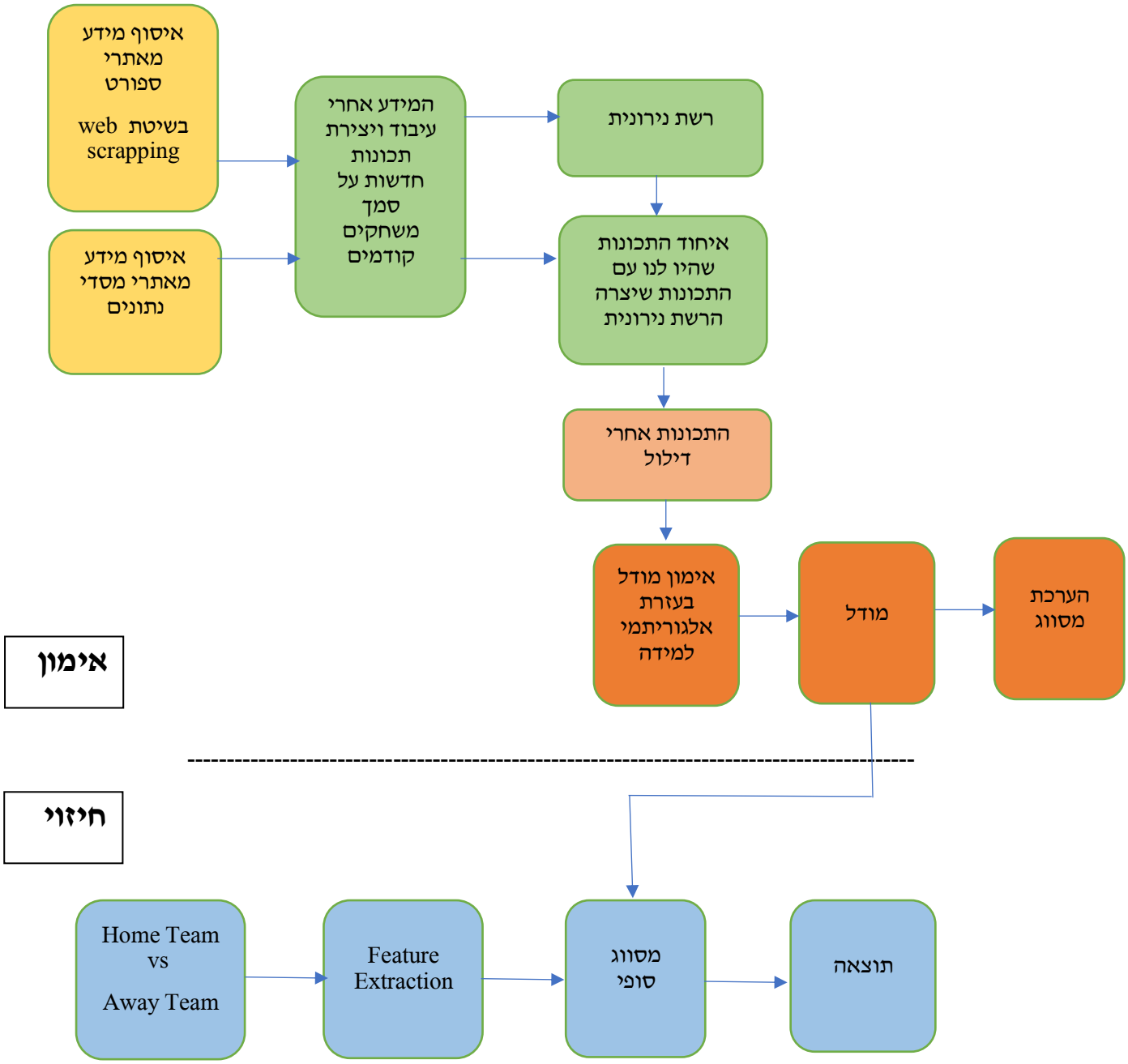
לאחר שלב עיבוד המידע הוספת וחיזוי תכונות ובחירת איך לאמן את המסווגים החלטנו ללמוד את התכונות הטובות ביותר כי חלק מהתכונות לא רלוונטיות ויכולות לגרום ל- overfitting ובנוסף תכונות רבות יגרמו להארכת זמן בניית המסווג.

על מנת לדלל את התכונות ביצענו features selection על ידי אלגוריתם select k best אשר בוחר את K המאפיינים המשמעותיים ביותר עבור הסיווג בהסתמך על score function כלשהי אותה נרצה לבחור כך שתיתן ביצועים טובים על הנתונים שלנו. (נפרט על זה בהמשך)

- **<u>יצירת מסווג סופי :</u>**

מבין כל אלגוריתמי הלמידה שהשתמשנו בהם נבחר את האלגוריתם שמחזיר את המסווג שנותן את הדיוק הטוב ביותר עם הפרמטרים שלו והמאפיינים הסופיים שבחרנו.
המסווג הסופי יהיה מוכן להפעלה על דוגמאות לא מסווגות, בהיתן שתי קבוצות המסווג מחזה את התוצאה של המשחק. (H, A, D)

- **תיאור כללי של דרך הפתרון :**



תיאור המערכת:

המערכת אותה בנינו לשם יישום דרך הפתרון מורכבת ממספר קבצי פיתון בנוסף לקבצי הנתונים, חלק אחד של קבצי מבצע עיבוד נתונים והחלק השני בוחן מסווגים שונים ובוחר מתוכן את המסווג האופטימלי שמשמש כפתרון לבעיה.

הקבצים כתובים ב 3 python ומשתמשים בהם בעיקר בספריית pandas לשם עיבוד הנתונים ו sklearn ללמידה.

חלקי המערכת :

- **שלב העיבוד המקדים של המידע :**

אחרי חילוץ הנתונים שצריכים מאתרי הספורט ויצירת מאגר הנתונים שעליו יתבססו האלגוריתמים השונים בבניית המסווגים, קיבלנו מספר קבצים שאחד מכיל את התוצאות של המשחקים וסטטיסטיקות של המשחק עצמו ומספר קבצים נקראים Lineups17-20.xlsx אשר מכילים בתוכם את ההרכב של שתי הקבוצות ועוד מספר קבצים נקראים Players17-20.xlsx שמכילים סטטיסטיקות על השחקנים, הקוד שמייצר את הקבצים נמצא בקובץ features_extraction.py.

כל הקבצים האלה עוברים עיבוד על מנת לקבל תאריך בפורמט אחיד בכל הקבצים ובנוסף על מנת לתאם את השמות של הקבוצות והשחקנים בקבצים השונים בשביל כך כתבנו קובץ פיתון הנקרא features_preprocessing.py אשר עובר על הקבצים האלה ומייצר קבצים חדשים באותו שם אבל בתיקייה שונה השומרים על מבנה אחיד וניתן לעבוד איתם.

בנוסף הקובץ הזה עובר על כל המשחקים מייצר את התכונות החדשות על ידי סריקת שלושת המשחקים הקודמים של שתי הקבוצות והמשחקים הישרים ביניהם, לאחר מכן מחלץ את ההרכב של כל קבוצה על ידי חיפוש המשחק בקובץ המתאים (באחד קבצי ה Lineups) ואז עבור כל שחקן מחלץ את הסטטיסטיקה שלו מהקובץ המתאים (אחד קבצי ה Players) ומייצר את התכונות החדשות, וכך נקבל עבור כל משחק ויקטור של תכונות המוכן לכנס לרשת ה-נירונית על מנת לחזות עוד תכונות.
בסיום ריצת הקובץ הזה נוצרו שני קבצים חדשים אחד בשם selected_features.xlsx והשני בשם selected_features_all_data.xlsx כאשר מכילים את התכונות החדשות שיצרנו על פי משחקים קודמים ב 3 העונות האחרונות ו- 4 העונות האחרונות בהתאמה, קבצים אלו שמרנו אותם בתיקייה שנקראת postprocessing_data.

- **שלב חיזוי התכונות:**

כתבנו קובץ נוסף בשם feature_predictions.py בשביל לחזות את הסטטיסטיקה בתוך המשחק, קובץ הזה משתמש בשני הקבצים שיצרנו בשלב הקודם, כאשר מבצע normalization ו- standardization למידע כמו שהסברנו מקודם ואז קיבלנו לכל משחק ויקטור תכונות שהולך לכנס לרשת הנירונית, בקובץ הזה השתמשנו בשלוש רשתות נירונית שונות (למלא) הקובץ הזה מריץ את שלושת הרשתות השונות (נרחיב על זה בשלב הניסויים) ומחזיר את הרשת הנירונית שקיבלנו עבורה את התוצאות הטובות ביותר.

- **שלב הלמידה והניסויים:**

השלב הזה מחולק לשני שלבים :
<u>חיזוי התכונות שמתרחשות במשחק עצמו :</u>
השתמשנו ב 3 רשתות שונות ובחרנו ברשת עם MSELoss מינמלי.

הרצנו את כל הרשתות ל 500 epochs עם, כך שבכל epoch אימנו את הרשתות על FootballDataSet שמכיל את המידע הרלוונטי שהכנו בעיבוד המקדים כך שהתכונות החדשות מנורמלות לפי ההתפלגות שלהן, בכל epoch אימנו את הרשתות עם batches בגודל 16 עם shuffle ע"י DataLoader.

תיאור הרשתות והפרמטרים שלהם נמצאים בתיאור הרשתות.

<u>חיזוי התוצאה הסופית :</u>
בשלב הזה השתמשנו במספר אלגוריתמי למידה :
KNN, Gradient Boosting, Decision Tree, Random Forest, SVM, Naïve Bayes
כולם על ידי חבילת sklearn.
כתבנו עוד קובץ בשם learning אשר מבצע את הדברים הבאים :

1. טוען את הקובץ selected_features.xlsx

2. מחלק את הקובץ לשתי קבוצות training and validation לפי אלגוריתם cross validation הנמצא בספריית sklearn עם פרמטר 5 כלומר מחלק את הקובץ ל 5 חלקים אשר 4 משמשים לtraining ואחת ל validation.

3. עבור כל אלגוריתם למידה מריץ את האלגוריתם עם פרמטרים שונים אשר הגדרנו כמלון ועבור כל אחד שומר את התוצאות בקובץ אקסל ממוינים בסדר יורד לפי הדיוק של האלגוריתם.

4. עבור כל אלגוריתם בוחרים את הפרמטרים שקיבלנו עבורם את התוצאות הטובות ביותר ומריצים עליהם select best K עבור ערכי K שונים הנעים מ 15-30 ובוחרים את האלגוריתם והערך K אשר החזירו את הדיוק הטוב ביותר.

- **המסווג הסופי:**

המודל הסופי נבחר לפי הניסויים שעשינו בשלב הלמידה, כאשר בוחרים את אלגוריתם הלמידה עם הפרמטרים וערך K אשר החזירו את התוצאה הטובות ביותר.

מתודולוגיה ניסויית:

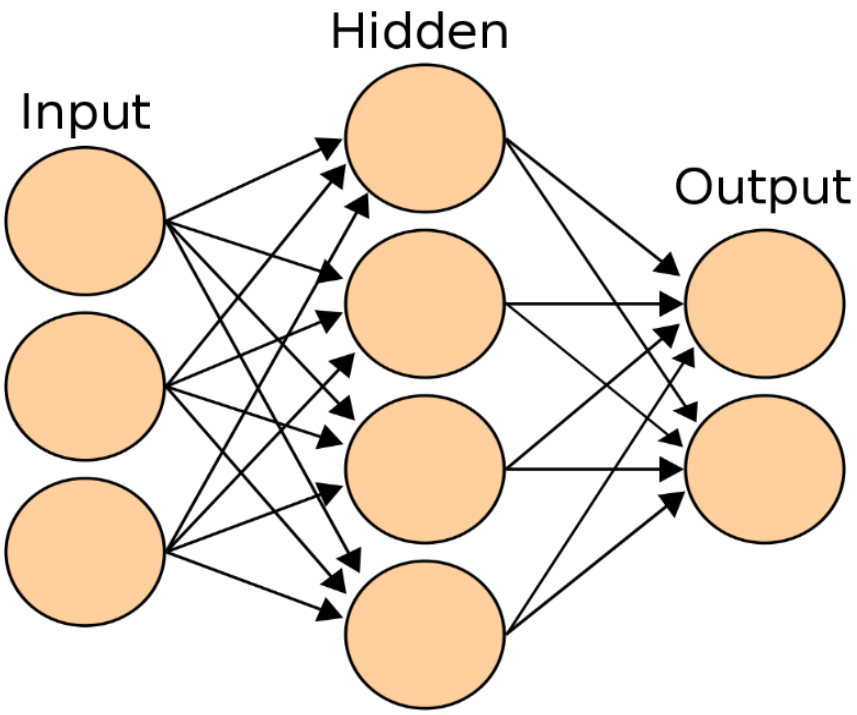
המטרה שלנו בפרויקט זה היא למצוא אלגוריתם למדיה בעל דיוק גבוה לבעיה שהצגנו.
חלק הניסויים התחלק ל 3 שלבים:
השלב הראשון היה ללמוד לחזות את התכונות עצמם על ידי רשת נירונית.
בשלב השני בצענו ניסויים שמטרתם למצוא את הפרמטרים המוצלחים עבור כל אלגוריתם למידה, ובחלק האחרון בצענו ניסויים שמטרתם מציאת המספר האידיאלי של מאפיינים בהם ישתמש האלגוריתם.
בחלק זה נציג את אלגוריתמי הלמידה שבחרנו, נסביר בקצרה את אופן הפעולה שלהם ונציין את הפרמטרים שבחנו. ובסוף נציג את תוצאות הניסויים ואת המסקנות מהם.

הצגת אלגוריתמי הלמידה:

- **אלגוריתמים חיזוי התכונות:**

Artificial Neural Networks:

רשת עצבית מלאכותית (ANN – Artificial Neural Network), רשת נוירונים או רשת קשרית הוא מודל מתמטי חישובי שפותח בהשראת תהליכים מוחיים או קוגניטיביים המתרחשים ברשת עצבית טבעית ומשמש במסגרת למידת מכונה .רשת מסוג זה מכילה בדרך כלל מספר רב של יחידות מידע) קלט ופלט (המקושרות זו לזו, קשרים שלעיתים קרובות עוברים דרך יחידות מידע "חבויות .(Hidden Layer) "צורת הקישור בין היחידות, המכילה מידע על חוזק הקשר, מדמה את אופן חיבור הנוירונים במוח. השימוש ברשתות עצביות מלאכותיות נפוץ בעיקר במדעים קוגניטיביים ,ובמערכות תוכנה שונות - בהן: מערכות רבות של אינטליגנציה מלאכותית המבצעות משימות מגוונות - זיהוי תווים ,זיהוי פנים ,זיהוי כתב יד ,חיזוי שוק ההון ,מערכת זיהוי דיבור , זיהוי תמונה, ניתוח טקסט ועוד.



<u>שיטות ומונחים שהשתמשנו בהם ברשתות:</u>

ראיתי שלא צריך להבהיר ולכן הוספתי לינקים למונחים עיקריים.

-) MSELoss : פונקציית המטרה שניסינו למזער.
-) learning rate.
-) momentum : momentum in neural networks is a variant of the stochastic gradient descent, it replaces the gradient with a momentum which is an aggregate of gradients . זה בקיצור
-) batch normalization.
-) Relu.
-) Softmax.
-) Sigmoid.

תיאור הרשתות :

1) רשת דינמית : הרשת הזו בנויה משכבת FC input layer ואז מספר רנדומלי בין 0 ל 3 שכבות פנימיות FC בגודל 128x128 (אותה שכבה משוכפלת) ובסוף FC output layer,כך שבין כל שתי שכבות כאלו יש שכבת Relu , וה-optimizer שהשתמשנו בו הוא SGD עם lr=0.01 and momentum=0.9.

2) הרשת השנייה בנויה משכבת FC input layer ואז שכבת FC בגודל 128x64 ובסוף FC output layer, כך שבין כל שתי שכבות כאלו יש שכבת Relu ובסוף יש שכבת softmax, וה-optimizer שהשתמשנו בו הוא SGD עם lr=0.05 and momentum=0.9.

3) הרשת השלישית בנויה משכבת FC input layer ואז שתי שכבות FC בגודל 32x64 ו- 64x32 ובסוף FC output layer, כך שבין כל שתי שכבות כאלו יש שכבת Relu ואז שכבת BatchNormalization, ובסוף יש שכבת sigmoid, , וה-optimizer שהשתמשנו בו הוא Adam עם lr=0.01 and.

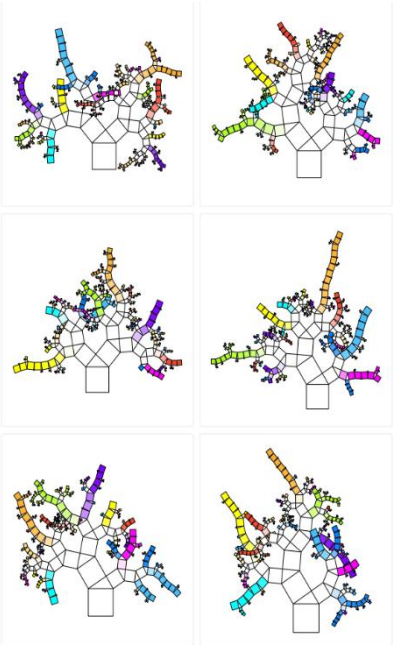
- הערה : הרשתות אומנו עם כמה optimizers ופרמטרים שונים אבל בחרנו את הפרמטרים שנתנו את התוצאות הכי טובות.

- ראיתי כי נכון להשתמש במומנטום כדי לא להיתקע בנקודות של מינימום מיקומי ולשפר את תהליך הלמידה.

- **אלגוריתמים שלב הלמידה:**

החלטנו להשתמש ב 6 אלגוריתמי למידה שונים:

Random Forest:



הוא אלגוריתם למידה המורכב מודעת עצי החלטה.

ועדה היא אוסף של עצים בה כל עץ מחזיר ערך משלו, הערך הסופי המוחזר הוא פונקציה המחושבת על כלל הערכים המוחזרים מעצי הועדה.

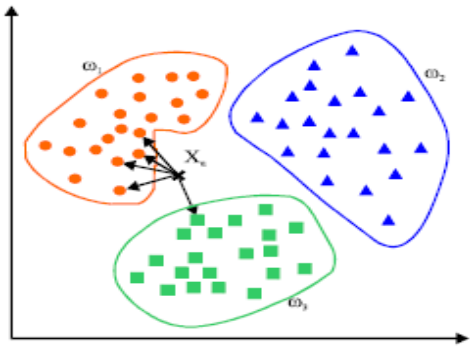
וועדת היער האקראי בנויה כך שכל עץ מאומן על תת קבוצה שונה של דוגמאות ובכל פיצול של צומת פנימי בעץ נבחרת תת קבוצה של תכונות באופן אקראי, הפונקציה המשמשת לחישוב הערך המוחזר היא ממוצע הערכים שהחזירו שאר העצים. יתרון מרכזי של גישה זו הוא הפחתת תופעת ה over-fitting.

היער האקראי בו נשתמש הוא היער האקראי הממומש ע"י ספריית scikit-learn.

הפרמטרים שבחרנו לכוונן עבור אלגוריתם זה הם:

- n_estimators: מספר העצים בוועדה.
- max_depth: עומק העץ המקסימלי המותר.
- min_samples_split: מספר הדוגמאות המינימלי הנדרש על מנת לפצל צומת פנימי בעץ

KNN – K nearest neighbors:



אלגוריתם זה מבוסס על ההנחה שלעצמים שקרובים זה לזה במרחב יהיה סיווג דומה. שלב הלמידה כולל שמירה של הדוגמאות המתויגות והסיווג שלהן, ובהינתן דוגמא חדשה האלגוריתם מחשב את המרחק בין הדוגמא החדשה לבין כל אחת מהדוגמאות הידועות לו, ונותן לדוגמא החדשה את הסיווג של רוב הדוגמאות מבין ה Kהקרובות אליו ביותר. במסווג זה בחנתי מספר פרמטרים:

- n_neighbors: מספר הדוגמאות שאיתן "יתייעץ" המסווג ועל ידי ההצבעה שלהן יינתן סיווג של דוגמא חדשה.

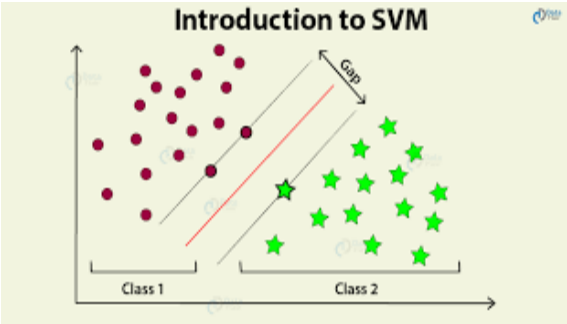
- Weight: פרמטר הקובע מה תהיה פונקציית המרחק בה ישתמש האלגוריתם על מנת לאמוד את המרחקים בין הדוגמאות שראה בשלב הלמידה

Decision Tree:

עץ החלטה הוא מודל חיזוי המייצר עץ שממפה תצפיות לערכים המתאימים עבורן, העץ בנוי מצמתי החלטה שבכל אחד מהם נבדק תנאי מסוים על מאפיין מסוים של התצפיות ועלים המכילים את הערך החזוי עבור התצפית המתאימה למסלול שמוביל אליהם בעץ.
בהינתן עץ החלטה, ההערכה ניתנת ע"י בדיקת התנאי בכל צומת אליו מגיעים ומעבר לצומת המייצג את התשובה החל מהשורש, הערך המוחזר הוא הערך השייך לעלה בסוף המסלול.
בפרויקט זה כל צומת פנימי מייצג מאפיין, התשובות האפשריות הן הערכים השונים עבור המאפיין, והעלים מייצגים סיווג טריניארי האם המשחק נגמר בניצחון ביתי, חוץ או תיקו.
עץ החלטה בו נשתמש הוא עץ ההחלטה הממומש ע"י ספריית scikit-learn.
הפרמטרים שבחרנו לכוונן הם:

- Max depth: עומק העץ המקסימלי המותר.
- Splitter: האסטרטגיה המשמשת לבחירת הפיצול בכל צומת. אסטרטגיות נתמכות הן "הטובות ביותר" לבחור את הפיצול הטוב ביותר ו"אקראי "לבחור את הפיצול האקראי הטוב ביותר.

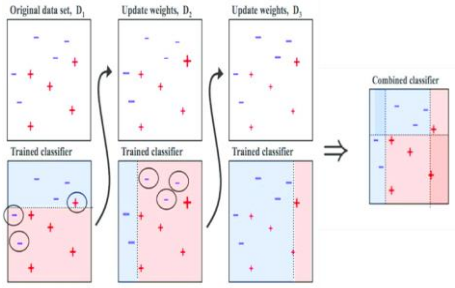
SVM (Support Vector Machine):



אלגוריתם זה מנסה לסווג את הדוגמאות על ידי מפריד ליניארי ,תוך ניסיון למקסם את ה"שוליים"- המרחק בין המפריד הלינארי לדוגמאות.
המפריד שיבחר בסוף פעולת האלגוריתם הוא זה שמפריד בין הדוגמאות בצורה ה"חזקה ביותר", כלומר זה שהשוליים בו הם הרחבים ביותר.
הפרמטרים שבחרנו לכוונן הם:

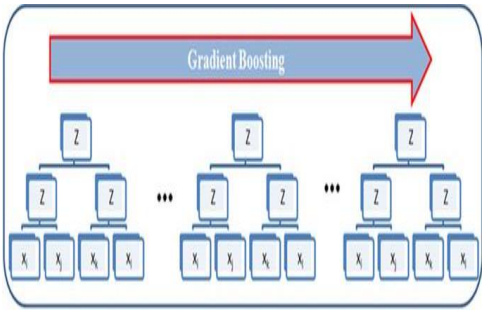
- C: לאחר ההפרדה יתכן מצב שבו ישנן דוגמאות אשר נמצאות באזור סיווג של דוגמאות המסווגות שונה מהן. פרמטר זה מגדיר מה יהיה ה"קנס "עבור דוגמאות אלו, והוא בעצם מבטא את ההעדפה שלנו באיזון בין שגיאות אלו לבין פונקציה פשוטה.
- Kernel: פרמטר זה מגדיר את סוג ההפרדה בו האלגוריתם ישתמש.

AdaBoost:



הוא אלגוריתם למידה המייצר מסווג לינארי כקומבינציה של מספר מסווגים אשר נבנים בצורה סדרתית.
בכל איטירציה ממשקלים מחדש את קבוצת הדוגמאות, אחרי המשקול החדש מופעל אלגוריתם הלמידה כדי לבנות את המסווג , משקול הדוגמאות לשלב הנוכחי מתבצע לפי ביצועי המסווג בשלב שלפני, דוגמאות שסווגו לא נכון עולות במשקל כך שיצטרך המסווג החדש להתמודד איתן.
n_estimators : פרמטר הקובע את מספר עצי ההחלטה בוועדה

Gradient Boosting:



GB מייצר מודל חיזוי בצורה של אנסמבל של דגמי חיזוי חלשים, בדרך כלל עצי החלטה.
הוא בונה את המודל בצורה שלבית כמו שיטות boosting אחרות, והוא מכליל אותם בכך שהוא מאפשר לבצע אופטימיזציה של פונקציית אובדן מובחנת שרירותית.
הפרמטרים שנבחן הם :

- min_samples_split: המספר המינימלי של דוגמאות הנדרש לפיצול צומת פנימי.
- min_samples_leaf : המספר המינימלי של דגימות הנדרש להיות בצומת עלים
- max_depth : עומק מקסימאלי של אומדן רגרסיה בודד.

- **דילול תכונות:**

השתמשנו באלגוריתם SelectKBest ב sklearn.
ובדקנו את ביצועי המסווגים עבור הערך הדיפולטיבי של func_Score שהוא classif_f (בגלל שזו בעיית קלסיפיקציה זו הפונקציה המתאימה).
והמשתנה K : שהוא מספר המאפיינים.
האלגוריתם מחזיר את K המאפיינים שקיבלו את ה"ציון" הגבוה ביותר.

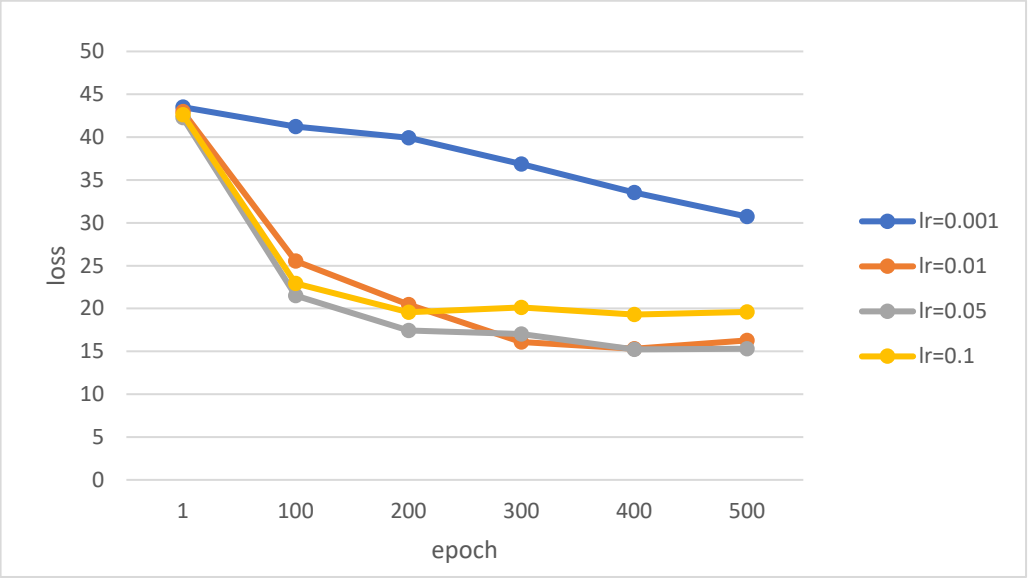
תוצאות הניסויים:

תוצאת שלב חיזוי התכונות בלמידה + כוונון פרמטרים :

בגרפים הבאים נראה התפתחות כל רשת כל 100 epochs עם מומנטום קבוע 0.9 לשתי הרשתות הראשונות, כך שכל קו מייצג learning rate שונה.

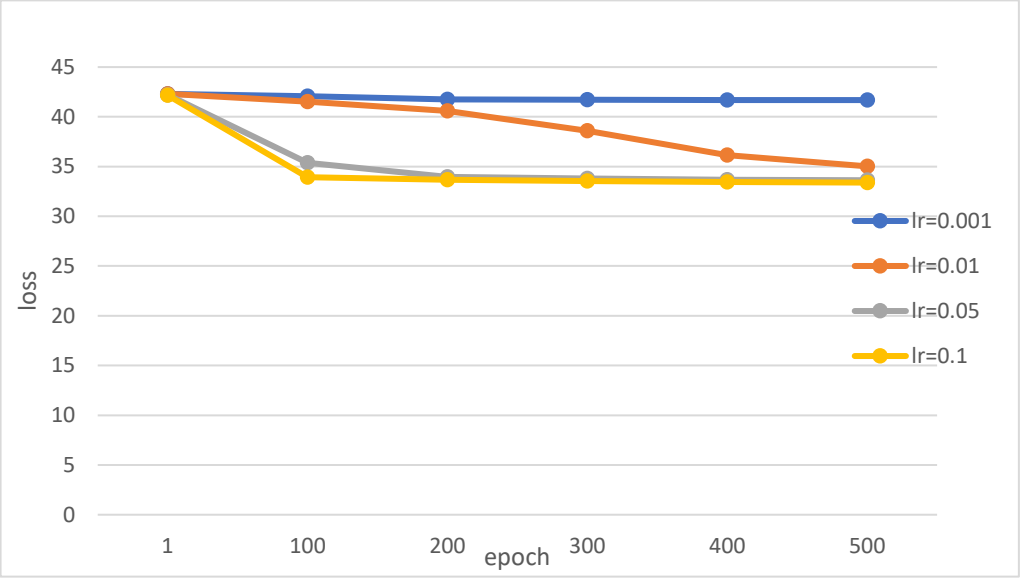
- הבהרה : הניסוי היה הרבה מאוד רחב ובדקנו מומנטים שונים עבור כל learning rate אבל לא רצינו לצרף הכל כדי לשמור על הסדר בדוח.

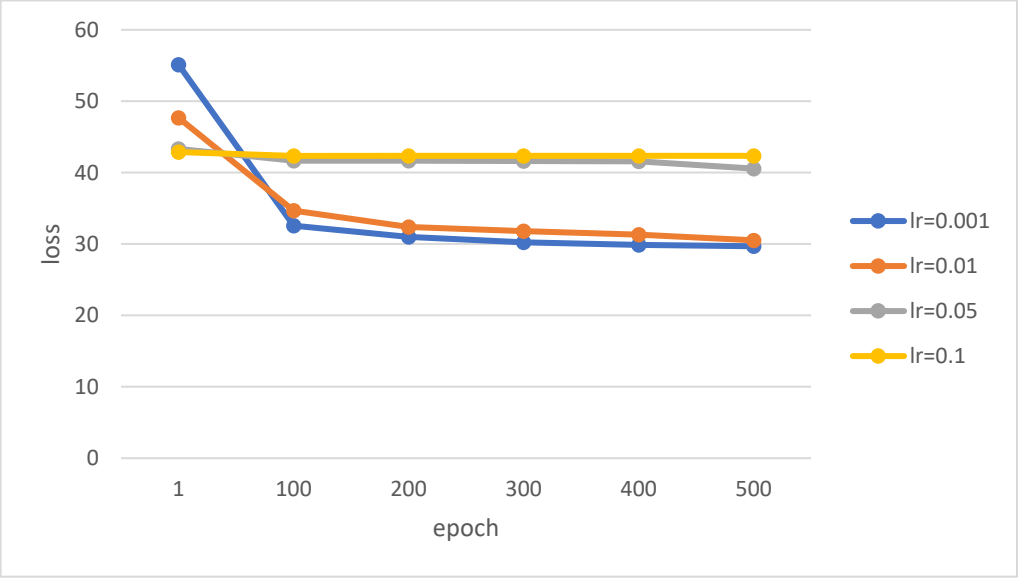
1) רשת 1 :



הרשת הזו נתנה את התוצאות הכי טובות מבין כל הרשתות, יכולים לראות כי התוצאות מתחילות להתכנס אחרי 500 ריצות, וקיבלנו ההפסד (loss) הכי נמוך ב lr=0.05 שהינו 15.308 שמאוד קרוב לתוצאה של lr=0.01 עם הפסד של 16.283, יכולים להסיק שזה הטווח של פרמטר הלמידה הכי טוב.
לגבי פרמטר למידה 0.001 זה מאוד קטן מה שלא נותן לרשת שלנו להתקדם בכיוון הגראדיינט בקצב אופטימלי כי אין לנו מספיק נתונים.
פרמטר למידה גדול יותר הוא אכן לומד בקצב יותר מהיר אבל הוא מפספס את נקודת המינימום איך שרואים בגלל הקפיצות הגדולות יחסית.

2) רשת 2+3 :



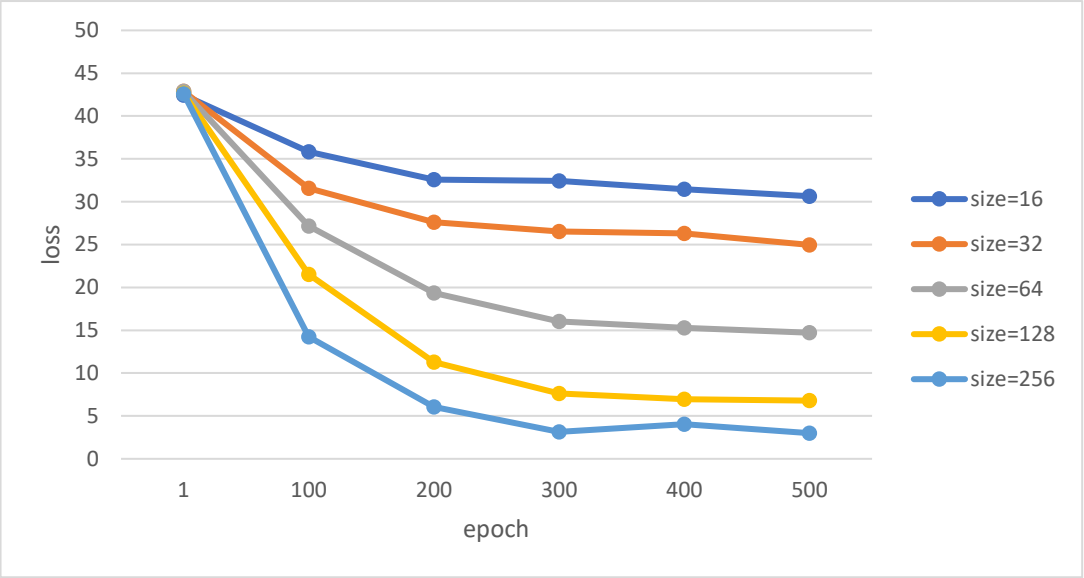


יכולים לראות שתוצאות הרשתות האלו פחות טובות, ברשת הראשונה קצב ההתקדמות אפסי עבור פרמטר למידה נמוך יחסית ומתכנסות להפסד גדול, לגבי הרשת השנייה פרמטר למידה גבוה מכניס את הרשת במהירות למינימום מקומי (זו הרשת שלא השתמשנו בה במומנטום).

ניתן ללמוד מזה שגודל הרשת ומספר שכבות גדול אינו תמיד הפתרון לכל בעיה, כי יש בעיות יותר מסובכות וכן מצריכות הרבה שכבות אבל אצלנו זה לא היה המקרה, אלא כוונון הפרמטרים, מציאת מספר שכבות מתאים ומציאת הערכים האידאלים.

השפעת גודל שכבת hidden fully connected ברשת הדינמית :

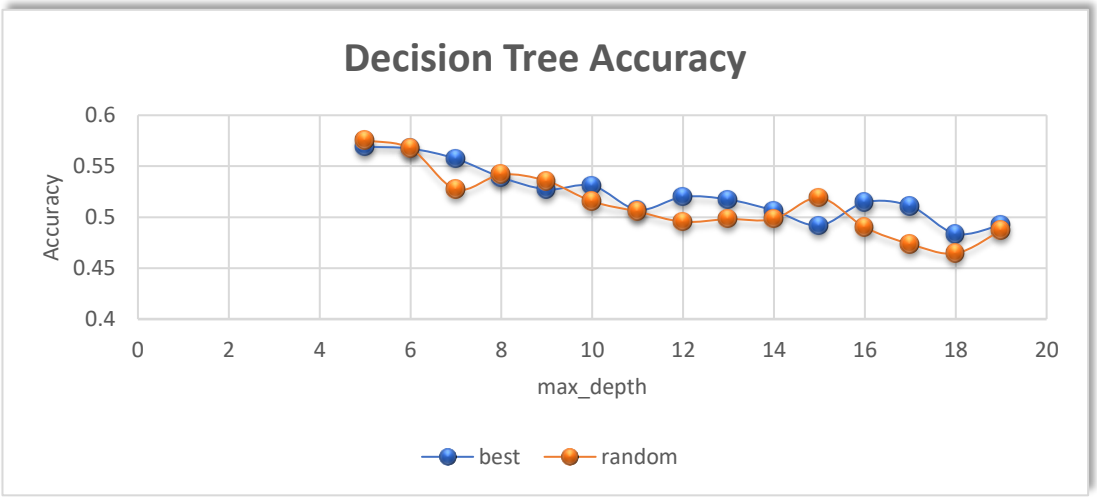
עשינו עוד ניסוי כדי לדעת את מידת השפעת גודל השכבה הפנימית על תוצאות הניסוי ברשת הדינמית הראשונה אחרי שנתנה לנו את התוצאות הכי טובות, והנה התוצאות :



קיבלנו שכל שהשכבה הפנימית תגדל בממדים שלה היא נותנת תוצאות יותר טובות, אפשר לייחס את זה לכך ששכבה יותר גדולה מכילה יותר מידע ולומדת דברים יותר מסובכים, אבל צריך להיזהר כי אולי המידע הזה הינו יכול להוות overfitting, הרשת תלמד את סט האימון. ולכן בסוף בחרתי בגודל 128 כי הינו קרוב בהפסד ל 256 ופחות קרוב ל overfitting.

- היינו יכולים לנתח עוד ולכתוב את כל הניסויים שעשינו אבל זה מאריך את הדוח ונכנס להרבה פרטים טכניים ברשתות עצבים.

Decision Tree:



Max Depth	Split	Accuracy
5	random	**0.575**
5	best	**0.569**
6	random	**0.567**
6	best	**0.567**
7	best	**0.557**

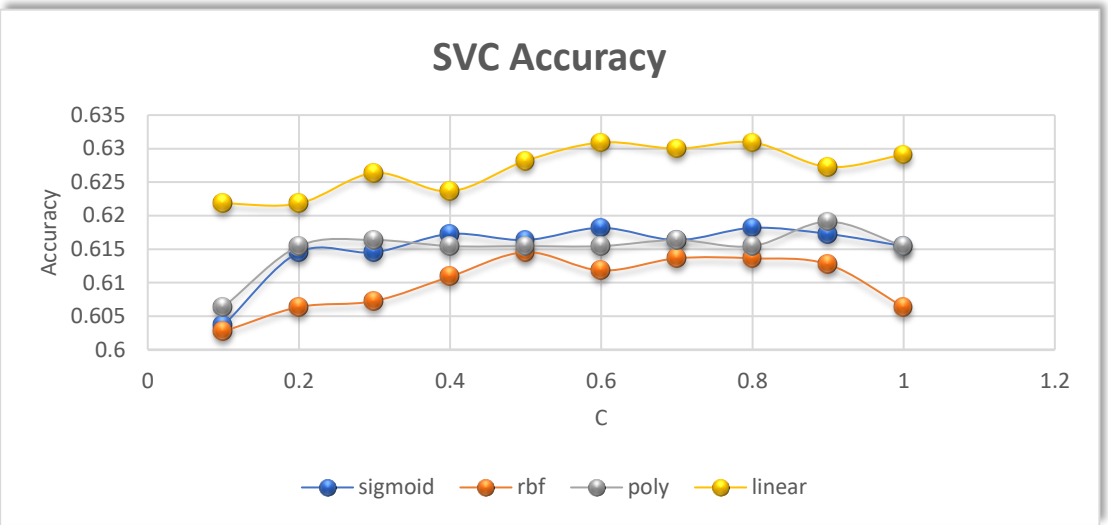
החלטנו למדוד את הביצועים של אלגוריתם Decesion Tree על פי שני פרמטרים, העומק המקסימאלי וטיב הפיצול, ניתן לראות את התוצאות בגרף מלמעלה ובנוסף הטבלה מכילה את 5 התוצאות הטובות ביותר שקיבלנו.

ניתן לראות שבשני שיטות הפיצול האלגוריתם מחזיר את התוצאות הטובות ביותר עבור עומק העץ המינימאלי שנתנו לו, ואם מגדילים את העומק המקסימאלי מקבלים תוצאות פחות טובות וזה נובע מתופעת ה overfitiing, בנוסף העדפנו לא להקטין את העומק המקסימאלי לפחות מ 5 כי אז המודל שלנו לא יהיה גמיש במיוחד.

בנוסף רואים שפונקציית הפיצול לא ממש משפיעה על הדיוק בחיזוי, יש מקרים שבה לפי פונקציה best מקבלים תוצאות טובות יותר ובמקרים אחרים להפך.

לסיכום את התוצאה הטובה ביותר קיבלנו עבור פיצול רנדומאלי ו- עומק מקסימאלי 5 כאשר קיבלנו דיוק של 0.575

SVC:



kernel / C	sigmod	rbf	poly	linear
0.2	**0.614**	**0.606**	**0.615**	**0.621**
0.3	**0.614**	**0.607**	**0.616**	**0.626**
0.4	**0.617**	**0.610**	**0.615**	**0.623**
0.5	**0.616**	**0.614**	**0.615**	**0.628**
0.6	**0.618**	**0.611**	**0.615**	**0.630**

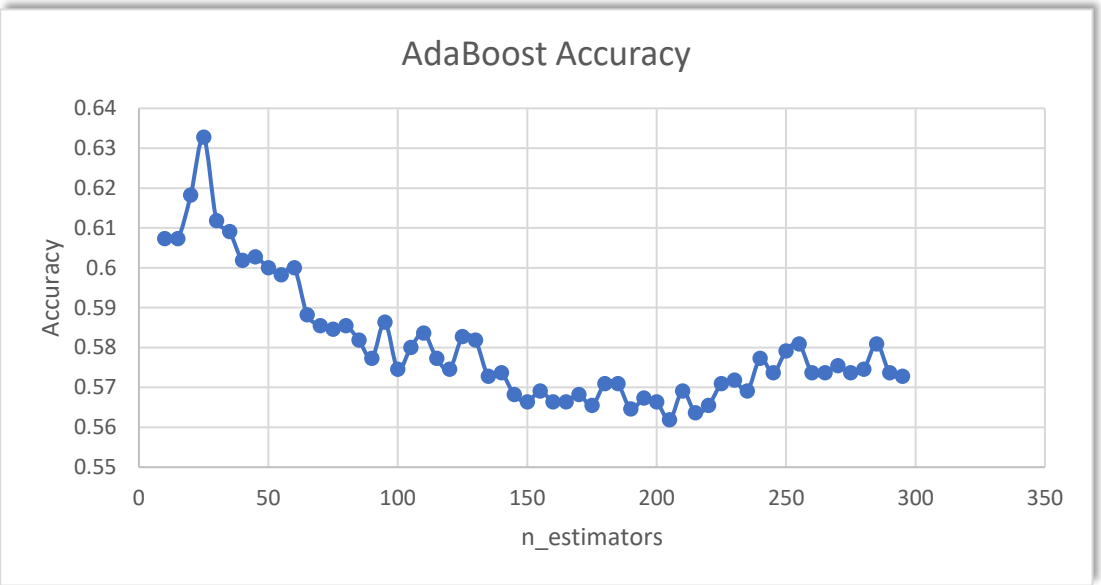
בניסוי הזה בחרנו למדוד שני פרמטרים שונים C ו- kernel שהסברתי עליהם בשלב המתדולוגיה הניסויית.

ניתן לראות בבירור שהדיוק הטוב ביותר קיבלנו עבור kernel = linear הסיבה לכך יכול להיות בגלל שיש לנו מספר קטן יחסית של דוגמאות (1080) ו-מספר רב של תכונות (יותר מ 40 תכונה) זה מקשה על מיפוי דוגמאות למימד גבוה יותר, ולכן האלגוריתם קיבל דיוק יותר טוב עבור הפרדה לינארית.

בנוסף ניתן לראות כשיש לנו ערכים גדולים יותר עבור פרמט c נקבל תוצאות טובות יותר, כמו שהסברנו קודם פרמטר זה קובע את הקנס עבור סיווג שגוי מה שדורש לדיוק גדול יותר של פונקציית ההפרדה.

לסיכום את התוצאה הטובה ביותר קיבלנו עבור מפריד לינארי וערך c=0.6 כאשר קיבלנו אחוז דיוק של 0.63.

AdaBoost:

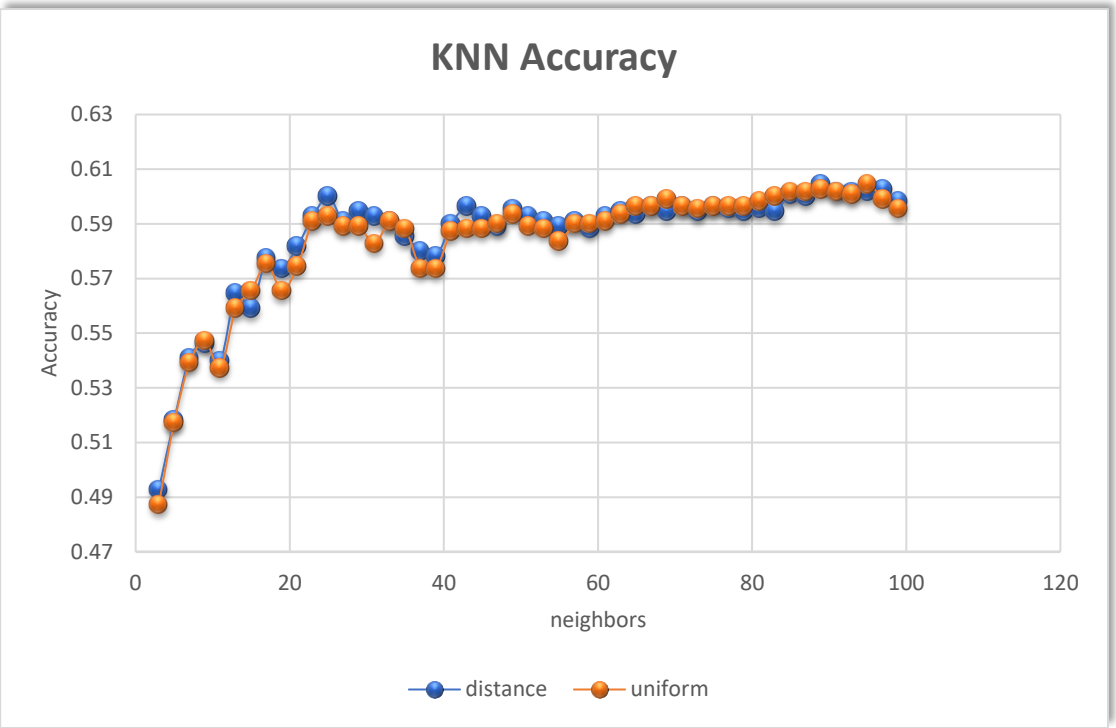


n_estimators	Accuracy
25	**0.632**
20	**0.618**
30	**0.611**
35	**0.609**
10	**0.607**

בניסוי הזה בחרנו לבדוק את ההשפעה של פרמטר יחיד שהוא ה n_estimators אפשר לראות שהגדלת הפרמטר מובילה לתוצאות פחות טובות, כמו שהסברנו מקודם ש adaboost ממשקל את הדוגמאות לפי תוצאת האיטרציה הקודמת ומריץ את האלגוריתם מחדש ולכן ניתן לראות שקיבלנו ביצועים יותר טובים מ עץ החלטה אבל בכל זאת כשמספר ה estimators עלה הביצועים התחילו לרדת.

לסיכום התוצאה הטובה ביותר קיבלנו עבור n_estimators=25 כאשר קיבלנו דיוק של 0.632.

KNN:



weights / n_neighbors	distance	uniform
89	0.604	0.602
95	0.601	0.604
97	0.602	0.599
89	0.604	0.602
29	0.594	0.589

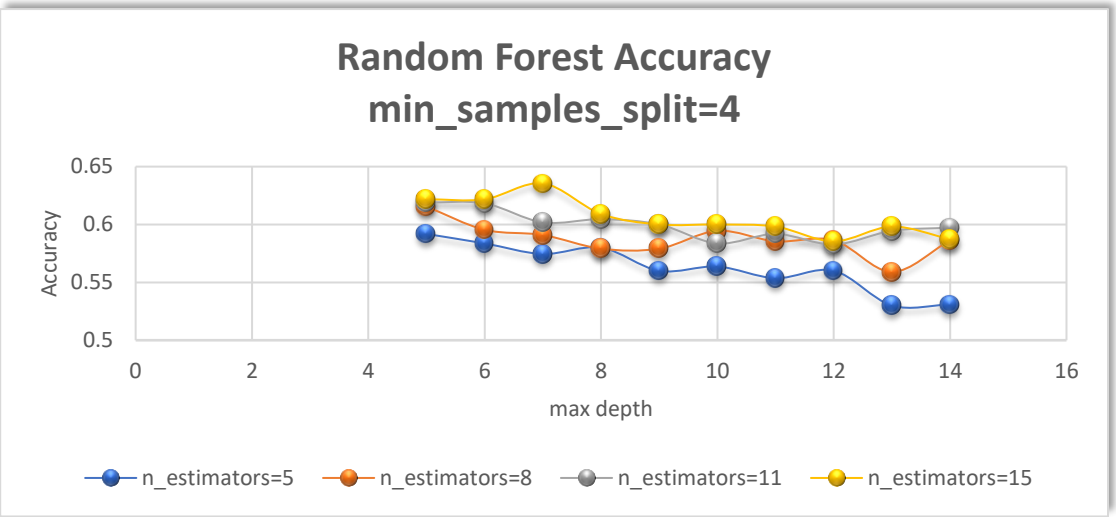
בניסוי הזה בחרנו לבדוק השפעת שני פרמטרים שונים, מספר השכנים ופונקציית המרחק.

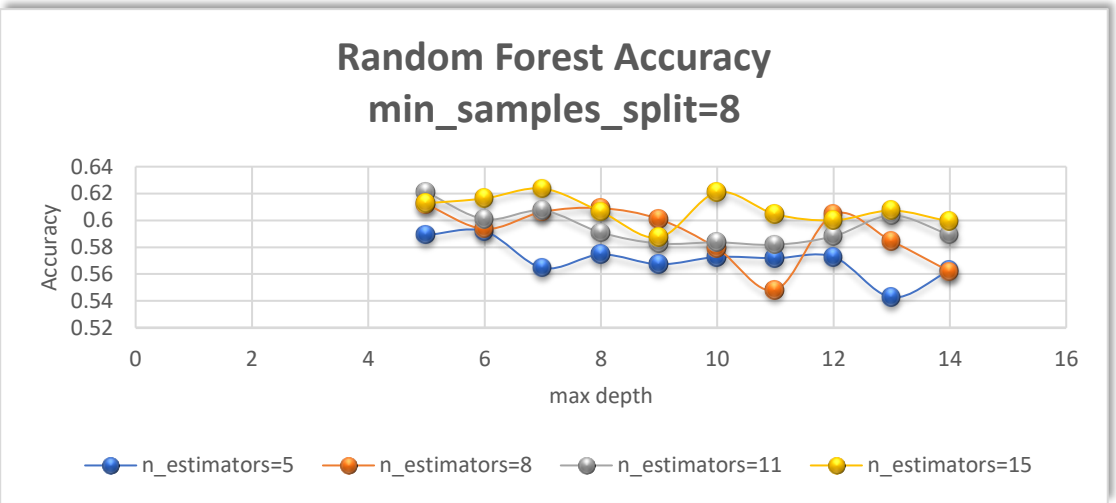
ניתן לראות שפונקציית המרחק לא הייתה לה השפעה משמעותית כאשר קיבלנו עבור שתי הפונקציות שבחנו תוצאות קרובות.

אבל ניתן לראות שעבור מספר השכנים קיבלנו תוצאות יותר טובות ככל שמספר השכנים גדל, וזה יכול לנבוע מכך שיש רעש בדוגמאות ולכן נצטרך יותר שכנים על מנת להתגבר על הרעש הזה, בנוסף ניתן לראות שאם מעלים את מספר השכנים יותר הביצועים יתחילו להתייצב למשל ניתן לראות שהפרש הדיוק בין 29 שכנים לבין 89 (שזה הדיוק הטוב ביותר) הוא בסך הכל 1 אחוז.

לסיכום אם את הדיוק הטוב ביותר מקבלים עבור מספר שכנים שווה ל 89 והפונקציה distance והדיוק שווה ל 0.604.

Random Forest:



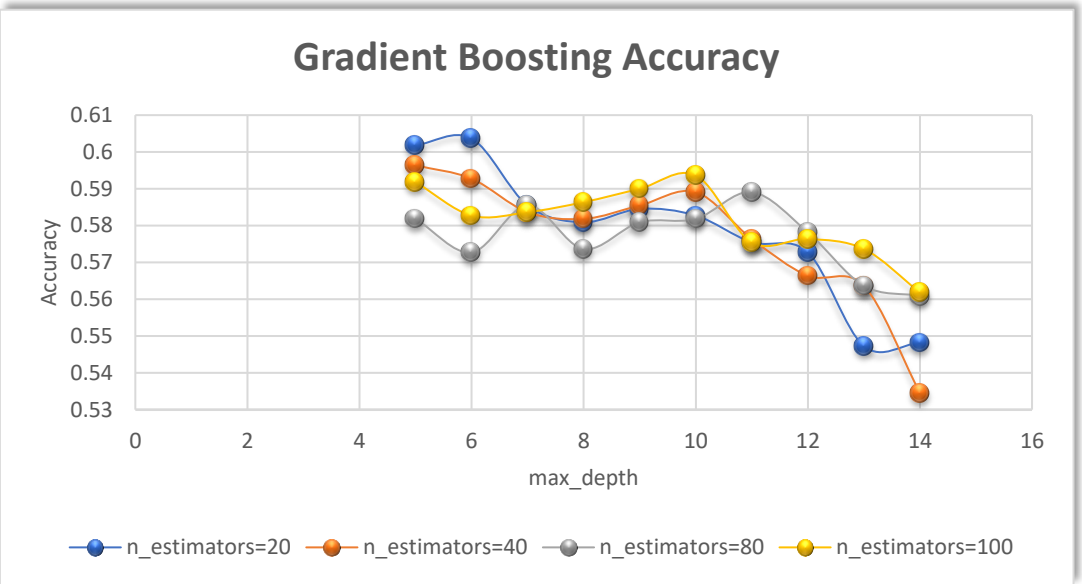


Max depth	min samples split	n_estimators	Accuracy
7	**4**	**15**	**0.635**
6	**4**	**13**	**0.625**
9	**4**	**14**	**0.624**
7	**8**	**15**	**0.623**
7	**8**	**13**	**0.623**

בניסוי הזה בחרנו לבחון שלושה פרמטרים n_estimators, min samples split ו- העומק המקסימאלי ניתן לראות שקיבלנו תוצאות יותר טובות ביחס לעץ החלטה בודד כאשר קיבלנו שיפור של 6 אחוז, וזה כנראה בגלל שמגוון העצים הצליח להתגבר על הרעש. בנוסף את עומק העץ המקסימאלי קבענו בין 6 ל 14 בגלל התוצאות שקיבלנו עבור עץ החלטה בודד כאשר ראינו שם שמגדילים את העומק של העץ נקבל תוצאות פחות טובות ואפשר לראות את זה גם פה, והפרמטר אחרון בדקנו רק עבור מספר ערכים שונים [4, 6, 8, 10, 12] והצגנו כן את שתי

התוצאות הטובות מבינן, קיבלנו את התוצאה הטובה ביותר עבור ערך 4 זה יכול לנבוע מכך שיש לנו הרבה תכונות (מעל 40) ומספר קטן של דוגמאות יחסית (1080) ולכן קיבלנו שמספר 4 הצליח להשיג את התוצאה הטובה כי מצד אחד מפחית את תופעת ה overfitting ומצד שני מתאים לעץ שיחסית קטן.

Gradient Boosting:



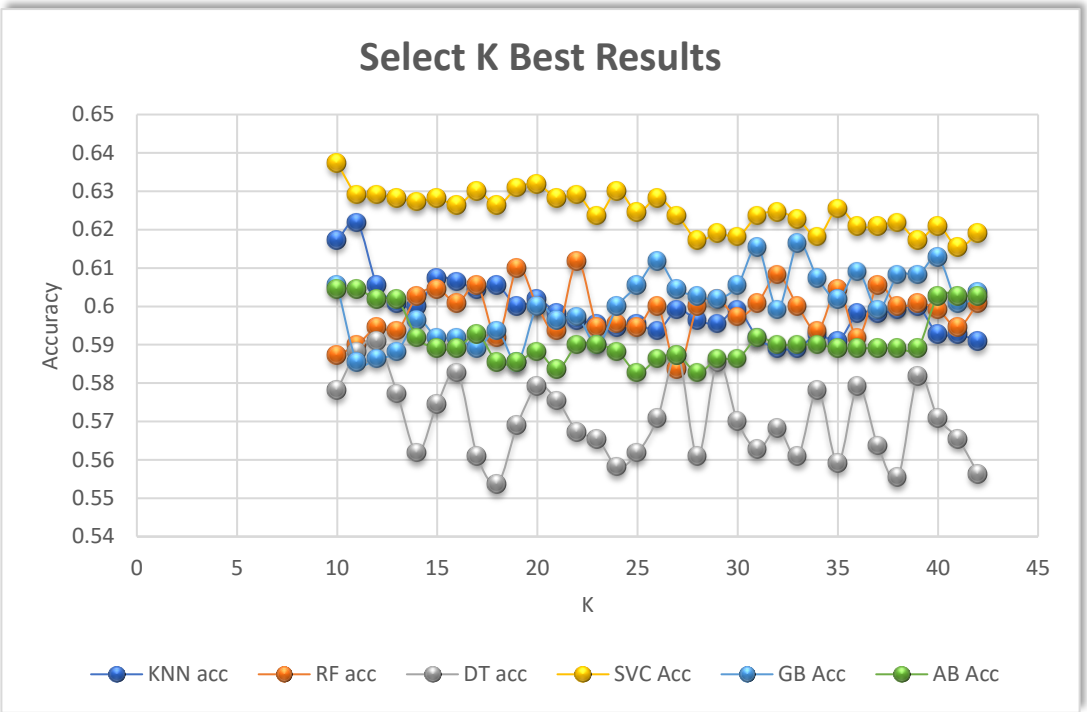
בניסוי הזה בחרנו לבדוק השפעת שני פרמטרים שונים, העומק המקסימאלי ו-מספר המסווגים, ניתן לראות שכאשר העומק המקסימלי יגדל הדיוק ירד לא משנה מה מספר המסווגים, וזה מחזק את התוצאות הקודמות כאשר ראינו שהעומק האידיאלי הוא 6-7.

Max Depth	n_estimators	Accuracy
6	**20**	**0.603**
5	**20**	**0.601**
5	**40**	**0.596**
10	**100**	**0.593**
6	**40**	**0.592**

בנוסף ניתן לראות שגם שמעלים את מספר המסווגים הביצועים ירדו שזה דומה גם לתוצאות שקיבלנו באלגוריתמים קודמים שגם שם קיבלנו את הביצועים הטובים ביותר עבור מספר מסווגים 20-30.

לסיכום קיבלנו שהדיוק הטוב שווה ל 0.603 והוא מתקבל עבור עומק מקסימאלי השווה ל 6 ומספר מסווגים ששווה ל 20.

Select K Best :

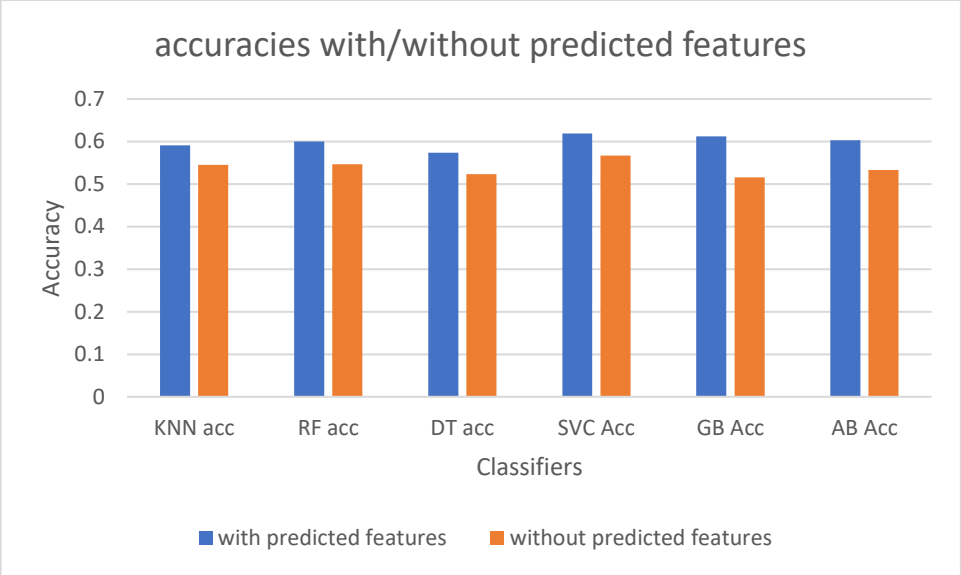


K	KNN	RF	DT	SVC	GB	AB	Best
10	**0.617**	**0.587**	**0.578**	**0.637**	**0.605**	**0.604**	**0.637**
20	**0.601**	**0.6**	**0.579**	**0.631**	**0.6**	**0.588**	**0.631**
19	**0.6**	**0.61**	**0.569**	**0.63**	**0.585**	**0.585**	**0.63**
24	**0.594**	**0.595**	**0.558**	**0.63**	**0.6**	**0.588**	**0.63**
17	**0.604**	**0.605**	**0.560**	**0.63**	**0.589**	**0.592**	**0.63**

קיבלנו כי השפעת הפרמטר K קטנה על כל אלגוריתם למידה וכי האלגוריתמים שמרו על הדיוק שלהם ועל יחס הדיוק שבניהם. מכך ששינוי מספר התכונות לא השפיע משמעותית על התוצאות אנו מקבלים שהבחירה של K=10 הייתה סבירה ואף מוצלחת.

השפעת התכונות שחזינו על התוצאות הסופיות של המסווגים :

רצינו לדעת איך משפיעות התכונות שחזינו על התוצאה הסופית של המסווגים עם הפרמטרים עם החיזוי הכי טוב, בניסוי חזינו את התוצאה הסופית עם/בלי התכונות הנוספות (שחזינו ע״י הרשת) שמקבלים במשחק עצמו.



אנו יכולים להסיק שהתכונות הנוספות אכן משפיעות בצורה חיובית על החיזוי, רואים שעם התכונות הנוספות מקבלים חיזויים יותר מדוייקים.

סיכום:

מטרתנו בפרויקט זה הייתה לבחון אלגוריתמי למידה שונים וכיצד הם מתמודדים עם בעיית חיזוי תוצאות כדורגל בליגה האנגלית על סמך משחקים קודמים וסטטיסטיקה של השחקנים והקבוצות.

הצלחנו לבנות מודל שמחזה תכונות של משחק על ידי רשת נירונית וסטטיסטיקה על משחקים קודמים ואז מצליח לחזות יותר מחצי המשחקים באופן נכון כאשר הגענו לאחוז דיוק של 0.539 במקרה הטוב ו-0.51 בממוצע.

בפרויקט הזה למדנו שחיזוי תוצאות כדורגל אינה משימה קלה, יש הרבה גורמים שיכולים להשפיע על תוצאות משחק כמו למשל:

- איסוף נתונים עדכניים: אספנו הרבה נתונים ממקורות שונים במהלך הפרויקט אבל הנתונים לא היו משקפים במאה אחוז את המצב של השחקנים לפני המשחק, אספנו נתונים מאתר הפיפא שלא מתייחס לכמה דברים שיכולים להיות חשובים, למשל את המצב של השחקן מחוץ לדשה, החיים האישיים יכולה להשפיע מאוד על שחקן וראינו הרבה שחקנים שאיבדו את הקריירה שלהם בגלל בעיות מחוץ למגרש.
  בנוסף את מספר המשחקים ששיחקה הקבוצה במהלך השבוע, יש קבוצות שמשחקות כמעט 3 פעמים בשבוע כי יש להם עוד התחייבות וזה גם גורם מרכזי שלא יכלנו להתייחס אליו, ויש עוד כמה גורמים שהיה קשה לנו לקחת בחשבון.

- עוד גורם מרכזי הוא המזל! המזל הוא אחד הגורמים החשובים במשחקי כדורגל במיוחד, יש הרבה "הפתעות" שרואים בכל שבוע למשל קבוצה המועמדת לרדת ליגה מנצחת את הקבוצה המועמדת להיות אלופה!.

- עוד גורם שהוסיף רמה של קושי זה שבחרנו לחזות את המשחקים בליגה האנגלית ידוע שהיא הליגה הכי תחרותית ומשוגעת בעולם. למשל שנת 2016 הייתה עדה על נס כאשר ליסטר סיטי הצליחה להרים את גביע הליגה והתגברה על סיכויים של 1 ל 5000! .

בכל זאת הצלחנו להשיג תוצאות טובות כאשר בדקנו שבממוצע אתרי ההימורים מצליחים לחזות משחקים בדיוק של 60% ואנחנו לא ממש רחוקים מזה.

בנוסף פרויקט זה נתן לנו ההזדמנות לעבוד עם דברים בפעם הראשונה כמו למשל להשתמש ב web scrapping לעבוד עם רשתות נירוניות שונות להתנסות עם אלגוריתמי למידה שונים , להבין לעומק אופן פעולת כל מסווג.

כיוונים למחקר עתידי:

כמו שהזכרנו קודם יש הרבה גורמים שלא יכלנו להתייחס אליהם במסגרת פרויקט זה ולכן אפשר בעתיד ל:

- שינוי שיטת חילוץ הנתונים: בפרויקט הזה אספנו את כל הנתונים לפי עונה כי אם היינו רוצים לקחת נתונים עבור כל משחק לפי תאריך שבו המשחק נערך היה לוקח לנו ימים על מנת לחלץ את הנתונים בשיטת ה web scrapping כי זה מבקש מאתנו לדפדף בין דפי אינטרנט הרבה פעמים (לפי כל משחק ולא לפי עונה) וזו הפעולה היקרה ותלויה במהירות האינטרנט, וגם בבחירה שלנו לחלץ נתונים לפי עונה זה היה לוקח זמן ובמיוחד כשיש ניתוקים באינטרנט אז היינו מאבדים את המידע ומתחילים מחדש.
אבל בכל זאת אפשר בעתיד לנסות ולחלץ נתונים לפי התאריך שבו המשחק נערך ולחלץ נתונים על כל שחקן בנפרד ואז לבנות פונקציית דירוג במקום להסתמך על אתר הפיפא.

- להתייחס לבעיה כבעיית רגרסיה: בפרויקט הזה החלטנו להתייחס לבעיה כבעיית קלסיפיקציה כמו שהוסבר בהגדרת הבעיה.
לכן אפשר להתייחס לבעיה כבעיית רגרסיה כאשר מנסים לחזות למשל כמה שערים תבקיע כל אחת משתי הקבוצות ואז על סמך התוצאה לקבוע מי ניצח את המשחק.

- להרחיב את הפרויקט לחזות תוצאת כדורגל באירופה למשל ולא רק בליגה האנגלית ואז נקבל בסיס נתונים גדול יותר שיכול לשפר את הלמידה ולקבל ביצועים טובים יותר.

- **הקובץ אשר יריץ המשתמש:**

בקובץ הזה שבשם predictor.py שמרנו את המסווג הסופי שבחרנו בשלב קודם.
הקובץ הזה טוען את הקובץ selected_features_all_data.xlsx ומאמן את המודל שלנו עליו.
ואז המשתמש יכול להכניס כקלט את המשחק שירצה לחזות, בשביל זה הוספנו command line arguments באופן הבא:

הוספנו את הדגלים הבאים:
h- : יציג הודעת עזרה.
T- : יציג את כל הקבוצות בפרימייר ליג.
H- : הקבוצה הביתית.
A- : קבוצת החוץ.

הדגל h- ידפיס את ההודעה הבאה:

```
(pythonProject) Rihams-MacBook-Pro:AI-Project rihamshaer$ python predictor.py -h
usage: predictor.py [options]

Premier League Match Predictor:

optional arguments:
  -h, --help          show this help message and exit
  -H , --home_team    Home Team
  -A , --away_team    Away Team
  -T                  Displays available teams
(pythonProject) Rihams-MacBook-Pro:AI-Project rihamshaer$
```

Structure Favorites Git Run TODO Problems Terminal Python Console
PEP 8: E501 line too long (127 > 120 characters)

הדגל T- ידפיס את ההודעה הבאה:

```
(pythonProject) Rihams-MacBook-Pro:AI-Project rihamshaer$ python predictor.py -T
Premier League Teams List For Season 2019/20:
Arsenal
Everton
Chelsea
Liverpool
Tottenham Hotspur
Leicester City
Brighton and Hove Albion
Manchester City
Wolverhampton Wanderers
Manchester United
Newcastle United
Norwich City
West Ham United
Aston Villa
AFC Bournemouth
Sheffield United
Crystal Palace
Burnley
Watford
Southampton
(pythonProject) Rihams-MacBook-Pro:AI-Project rihamshaer$
```

Structure Favorites Git Run TODO Problems Terminal Python Console
PEP 8: E501 line too long (127 > 120 characters)

ודוגמה להרצה:

```
(pythonProject) Rihams-MacBook-Pro:AI-Project rihamshaer$ python predictor.py -H "Watford" -A "Manchester City"
Manchester City
(pythonProject) Rihams-MacBook-Pro:AI-Project rihamshaer$
```

Favorites | Git | Run | TODO | Problems | Terminal | Python Console

PEP 8: E501 line too long (127 > 120 characters)

ביבליוגרפיה:

sports gambling industry worth .

http://football-data.co.uk/englandm.php?fbclid=IwAR0ofoEEdeiM4V-1nbExhQ6A8wOV0V9wLUqXLmyC3vsZ07nMGPxp_6g4OjY

https://sofifa.com/?fbclid=IwAR3i_JXN57MSQWi5vROfme28gsaovD0sDwwaDNGyfdJqF5ysVeETRDeW5Us

https://www.premierleague.com/matchweek/5687/blog

also we used different sites for technical needs.