**מיני פרוייקט**

**חישוב מונחה טבע**

**NBA GAME WINNING PREDICTOR**

**עמוד שער**

**מבוא**

הבעיה בה אנו מתעסקים היא לחזות איזו קבוצה תנצח בכל משחק בעונת כלשהי בליגת הכדורסל של ה-NBA שבארה"ב.

לפי מחקר שנעשה על ידי פורבס (1), הימורים במשחקי ספורט בארצות הברית בלבד הניבו רווח של מיליארד דולר בשנת 2020 (יש לציין שזו שנת קורונה, בה היו פחות משחקי ספורט). על פי התחזיות, המספר הזה צפוי לעלות בלפחות פי 6 עד שנת 2023. יתרה מכך, אם הימורים יהיו חוקיים בכל 50 המדינות בארה"ב הרווח אמור לגדול ב-19 מיליארד דולר בשנה.

אנחנו אוהבים כדורסל, ולכן רצינו לבנות מודל סטטיסטי מבוסס על למידת מכונה שלומד ממשחקים וממידע נוסף מעונה אחת וחוזה את תוצאות של העונה העוקבת. באופן יותר ספציפי, בהינתן מידע על משחק מסוים, נרצה לדעת איזו קבוצה תנצח באותו המשחק.   
לדוגמה, נצפה שהמודל יקח בחשבון פרמטרים מסוימים (שנקבעו מראש) בכל משחק בשביל החיזוי, לרבות רצפי ניצחונות, שחקני "אולסטאר", ביתיות וכו'.

בפרוייקט השתשמנו בשני סוגים של מודלים מבוססים למידת מכונה מסוג supervised:

1. עצי החלטה – בתהליך ההחלטה של המודל, בכל צומת פנימי בעץ נשאלת שאלה על פרמטר כלשהו בדאטה (שמויצג כווקטור) ובהתאם לתשובה ממשיכים לתת עץ אחר. מתחילים משורש העץ, ומקבלים את הפרדיקצייה בעלים.

Diagram

Description automatically generated

1. רשתות נוירונים – זהו מודל שמקבל דאטה שמויצג כוקטור של מספרים. הוקטור מועבר ברשת על ידי מספר של הכפלות של הוקטור במטריצות פנימיות, והפעלת פונקציית אקטביציה בכל שכבה של הרשת. את הפרדיקציה נמצא בשכבה האחרונה של הרשת, ובשביל לשפר את המודל – מופעלת פונקציית loss שבאופן כללי אומרת לרשת בכמה הפרדיקצייה הייתה רחוקה מהתוית האמיתית. בשביל למזער את ערך פונקציית ה-loss, מפעילים אלגוריתם שמשנה את משקלי הרשת בהתאם.

Diagram

Description automatically generated

**שיטה**

**ייצוג המידע**

באופן כללי, את המידע השגנו מהאתר Basketball reference (2) – באתר נמצא מידע על כל עונה בליגה, לרבות: תוצאות משחקים, שחקני "אולסטאר" וקבוצתם, טבלת מיקומים (standings) ועוד. לדוגמה:

* תוצאות משחקים:

Graphical user interface, application

Description automatically generated

**Figure 1**

* Standings:

Calendar

Description automatically generated

**Figure 2**

בפרוייקט, השתמשנו כ-training set בעונה 2017-2018 (בלי משחקי פלייאוף), וניסינו לחזות את עונת 2018-2019, כלומר עונת 2018-2019 הייתה ה-test set שלנו.  
באופן כללי, המידע ששימש אותנו היה המידע ב-**Figure 1** שעליו לעיתים הוספנו מידע על כל משחק כגון:

* רצפי ניצחונות של שתי הקבוצות (מספר טבעי)
* שחקני "אולסטאר" של שתי הקבוצות (מספר טבעי)
* איזו קבוצה מדורגת גבוה יותר (נקבע לפי **Figure 2** – או 0 או 1.
* האם קבוצת הבית ניצחה את קבוצת החוץ במשחק הקודם ביניהן – או 0 או 1.

את הקבוצות אנחנו מייצגים כ-one hot vector.

מבחינת כמויות המידע, ב-training set יש 1230 משחקים וב-test set יש 1213, כאשר כל משחק מיוצג כווקטור של מספרים (שאורכו משתנה לפי המידע שהוספנו).

**מודלים**

המודל הפשוט

ככלל, בכל משחק ספורט, ובפרט ב-NBA קיים יתרון לקבוצת הבית מכל מיני סיבות. מניסוי שערכנו בעונת 2018-2019 קבוצת הבית ניצחה 59% מהפעמים. כלומר, אם נבנה מודל שחוזה שהקבוצה שתנצח היא קבוצת הבית, מודל זה יזכה לדיוק של 59%. לכן, הגדרנו לעצמנו כמטרה לעבור את האחוז הנ"ל.

עצי החלטה

תחילה, ניסינו להשתמש בעצי החלטה כמודל שיחזה לנו את הקבוצה המנצחת בכל משחק. השתמשנו במימוש של ספריית scikit-learn ב-python. scikit-learn זו ספרייה קלה לשימוש שמכילה כלים ל-data science.

רשת נוירונים

השתמנו בשתי רשתות נוירונים שונות:

* רשת נוירונים שאנחנו כתבנו, **מבוססת על numpy בלבד**. רשת זו מכילה את כל הפונקציונליות העיקריות של רשתות נוירונים, כמו: forward pass, stochastic gradient descent, activation functions, backwards pass וכו'. ברשת שלנו כמות הנוירונים פר שכבה גדלה פי 2, עד שמגיעים לחצי מכמות השכבות, ואז זה קטן בחצי בכל שכבה.
* רשת הנוירונים הבסיסית של pytorch – linear fully connected (3) ברשתות שנשתמש בהן בשימוש הספרייה, נגדיר את כמות הנוירונים פר שכבה באופן דומה לרשת שלנו. החלטנו להשתמש במימוש של pytorch בשביל למדוד את איכות הרשת שאנחנו כתבנו.

כעת, נפרט על רשת הנוירונים שאנחנו בנינו. ניתן למצוא את הרשת בתיקיית NeuralNetwork ב-git של הפרוייקט.

הגדרנו מחלקה של רשת (network.py) שמורכבת בעיקרה מרשימה של שכבות לינאריות (מיוצגות על ידי מחלקה ב- layer.py), ומשכבת softmax כשכבה סופית שאחראית על הפרדיקציה.   
לכל שכבה יש פונקציית backward ו-forward ומחלקת הרשת היא שאחראית על העברת הדאטה בין השכבות ועל ביצוע ה-backpropagation.  
הרשת מבוססת numpy בלבד ולכן היינו צריכים לחשב נגזרות ולזכור את הגרדיינטים בעצמנו במהלך ה-forward pass. עבור כל רשת, שמרנו את הנגזרות פר שכבה (כשדה של המחלקה), ובמהלך ה- backward pass משתמשים בכל הנגזרות על מנת לחשב את הגרדיינט של כל הרשת. לשם ווידוא הגרדיינט, השתמנו במבחן גרדיינט המבוססים על קירוב טיילור.  
אתחלנו את המשקלים של כל שכבה באופן רנדומלי יוניפורמי בין 1 ל- 1-.

הערה: בשביל לעבור את המודל הפשוט, מדדנו את הדיוק של המודלים (עצי החלטה ורשתות נוירונים) על ידי פונקציית accuracy הבאה:

**ניסויים**

**עץ החלטה**

בשביל להגיע לעץ הכי טוב עבור הבעיה שלנו, ניסינו לשנות את הפרמטרים הבאים:

* Random state
* Criterion (Gini, Entropy)
* Node splitter (best, random)

תחילה ניסינו להריץ את את עץ ההחלטה עם הפרמטרים השונים על המידע הבסיסי (לפי**figure 1**) וקיבלנו תוצאות דומות מאוד עבור כל קומבינציה (התוצאות נעו סביב 0.549). בשביל לשפר את המודל, הוספנו את כל הפרמרטים הבאים לכל משחק:

* רצפי ניצחונות של שתי הקבוצות
* שחקני "אולסטאר" של שתי הקבוצות
* איזו קבוצה מדורגת גבוה יותר (נקבע לפי **Figure 2**).
* האם קבוצת הבית ניצחה את קבוצת החוץ במשחק הקודם ביניהן.

כאשר הוספנו את כל הפרמרטים, קיבלנו את התוצאות העיקריות הבאות:

|  |  |  |
| --- | --- | --- |
| Parameters | Best result | Random state |
| Random splitter  Gini | 0.561 | 2 |
| Best splitter  Entropy | 0.577 | 6 |
| Best splitter  Entropy | 0.567 | 10 |
| Best splitter  Entropy | 0.569 | 14 |
| Best splitter  Entropy | 0.578 | None |

לאחר מכן, ניסינו להריץ את עץ ההחלטה עם פחות פרמטרים, אך הדבר לא הניב תוצאות טובות יותר. כלומר הוספת הפרמטרים הנוספים היא הכרחית לשיפור המודל. נשים לב שהתוצאה הכי טובה של העץ קטנה יותר מאחוז הדיוק של המודל הפשוט שהוזכר קודם.

בשביל להשיג תוצאות טובות יותר, נעבור לרשתות הנוירונים.

**רשתות נוירונים**

תחילה, נתאר את הפרמטרים שאותם שינינו בשלב הניסוי:

* מספר השכבות (1,3,5,7)
* כמות ה-epochs (50,100,150)
* Batch size (1,10,15,20)
* Learning rate (0.01, 0.001, 0.0001)
* Activation functions (tanh, ReLU)

פונקציית הפרדיקציה שאנו נשתמש בה בשני סוגי הרשתות היא softmax ופונקציית ה-loss היא  
cross entropy.

רשת הנוירונים שלנו:

נחלק את התוצאות לפי שתי פונקציות האקטיבציה שאנו משתמשים בהם: tanh ו-ReLU. הטבלאות הבאות מציגות את התוצאות הטובות ביותר פר layer וזאת כאשר הוספנו את כל הפרמטרים האופציונליים לדאטה.

***Figure 3: Our NN with ReLU***

|  |  |
| --- | --- |
| Result | Parameters |
| 0.645 | Layers: 1  Epochs: 150  Batch size: 1  Learning rate: 0.01 |
| 0.631 | Layers: 3  Epochs: 100  Batch size: 1  Learning rate: 0. 01 |
| 0.625 | Layers: 5  Epochs: 150  Batch size: 1  Learning rate: 0.01 |
| 0.618 | Layers: 7  Epochs: 50  Batch size: 15  Learning rate: 0.01 |

***Figure 4: Our NN with tanh***

|  |  |
| --- | --- |
| Result | Parameters |
| 0.645 | Layers: 1  Epochs: 150  Batch size: 1  Learning rate: 0.01 |
| 0.647 | Layers: 3  Epochs: 100  Batch size: 1  Learning rate: 0. 01 |
| 0.621 | Layers: 5  Epochs: 150  Batch size: 1  Learning rate: 0.01 |
| 0.608 | Layers: 7  Epochs: 150  Batch size: 1  Learning rate: 0.01 |

רשת הנוירונים של pytorch:

נחלק גם פה את התוצאות לפי שתי פונקציות האקטיבציה שאנו משתמשים בהם: tanh ו-ReLU. הטבלאות הבאות מציגות את התוצאות הטובות ביותר פר layer בדומה לטבלאות הנ"ל.

***Figure 5: Pytorch NN with ReLU***

|  |  |
| --- | --- |
| Result | Parameters |
| 0.649 | Layers: 1  Epochs: 150  Batch size: 15  Learning rate: 0.01 |
| 0.645 | Layers: 3  Epochs: 100  Batch size: 1  Learning rate: 0. 001 |
| 0.639 | Layers: 5  Epochs: 100  Batch size: 15  Learning rate: 0. 01 |
| 0.648 | Layers: 7  Epochs: 150  Batch size: 10  Learning rate: 0.01 |

***Figure 6: Pytorch NN with tanh***

|  |  |
| --- | --- |
| Result | Parameters |
| 0.645 | Layers: 1  Epochs: 150  Batch size: 1  Learning rate: 0.01 |
| 0.631 | Layers: 3  Epochs: 100  Batch size: 1  Learning rate: 0. 01 |
| 0.625 | Layers: 5  Epochs: 150  Batch size: 1  Learning rate: 0.01 |
| 0.618 | Layers: 7  Epochs: 50  Batch size: 15  Learning rate: 0.01 |

**רפלקציה**

מסקנות

1. לפי דעתנו, הסיבה המרכזית שלא הגענו להצלחה יותר גבוהה בכל מודל שבדקנו היא שלא היה לנו הרבה מידע.  
   מחסור המידע נבע מכך שבמשחקי ה-NBA יש שוני גדול בין עונות משחק (למשל, הלייקרס יכולים להיות טובים בעונה אחת וגרועים בעונה אחרת). לכן, במידה וניקח עוד דאטה מעונות קודמות,נקבל יותר רעש והמידע יהפוך ללא רלוונטי. מסיבה זו, בחרנו להתמקד על דאטה מעונה אחת בלבד.
2. רשת הנוירונים שלנו ושל pytorch עקפה בדיוקה את המודל הפשוט ב-5% והייתה יותר טובה מעץ החלטה.
3. נשים לב שבשביל משימה זו, הרשת שלנו והרשת של pytorch בעלות תוצאות דומות מאוד. לכן ניתן להסיק שחוסר הדיוק הוא אינה בעיה טכנית של הרשת, כלומר כנראה מדובר בבעיה משותפת של דאטה שהוזכרה קודם.
4. מהתוצאות ניתן לראות שרשת עם שכבה אחת בלבד הייתה יותר טובה מרשתות גדולות יותר. כידוע, רשת עם שכבה אחת דומה באופיה למפריד לינארי. ייתכן שנשיג דיוק גדול יותר אם נשתמש במפריד לינארי סטנרדטי.

שיפורים עתידיים

1. המידע שהוספנו למידע הבסיסי לכל משחק נועד בשביל להקטין את הרעש בין עונות המשחק. עם זאת, צריך למצוא דרך יותר טובה להוסיף עונות ל-test set.
2. לנסות לייצג את הדאטה בצורה אחרת: פחות sparse, לנרמל או למרכז.
3. לנסות להשתמש במפריד לינארי כמודל machine learning.

**קישורים**

1. המאמר של פורבס

<https://www.forbes.com/sites/forbestechcouncil/2021/02/25/not-just-a-game-online-sports-betting-and-the-rise-of-corrosive-technology/?sh=7f24801570ec>

1. האתר ממנו נלקח הדאטה: [Basketball Statistics and History | Basketball-Reference.com](https://www.basketball-reference.com/)
2. Pytorch linear layer: [Linear — PyTorch 1.8.1 documentation](https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear)