בס"ד

למידת מכונה - פרויקט

תיאור המאגר:

המאגר שלנו מכיל נתונים על הליגה האנגלית הבכירה (English Primer League) מעונת 2000-2001 עד עונת 2017-2018.

המאגר מכיל המון נתונים, והנתונים שאנו הוצאנו ממנו מתוארים להלן:

Date = Match Date (dd/mm/yy)
HomeTeam = Home Team
AwayTeam = Away Team
FTHG = Full Time Home Team Goals
FTAG = Full Time Away Team Goals
FTR = Full Time Result (H=Home Win, D=Draw, A=Away Win)
HTHG = Half Time Home Team Goals
HTAG = Half Time Away Team Goals
HTR = Half Time Result (H=Home Win, D=Draw, A=Away Win)
Referee = Match Referee
HS = Home Team Shots
AS = Away Team Shots
HST = Home Team Shots on Target
AST = Away Team Shots on Target
HC = Home Team Corners
AC = Away Team Corners
HF = Home Team Fouls Committed
AF = Away Team Fouls Committed
HY = Home Team Yellow Cards
AY = Away Team Yellow Cards
HR = Home Team Red Cards
AR = Away Team Red Cards

	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	...	HST	AST	HC	AC	HF	AF	HY	AY	HR	AR
0	11/08/17	Arsenal	Leicester	4	3	H	2	2	D	M Dean	...	10	3	9	4	9	12	0	1	0	0
1	12/08/17	Brighton	Man City	0	2	A	0	0	D	M Oliver	...	2	4	3	10	6	9	0	2	0	0
2	12/08/17	Chelsea	Burnley	2	3	A	0	3	A	C Pawson	...	6	5	8	5	16	11	3	3	2	0
3	12/08/17	Crystal Palace	Huddersfield	0	3	A	0	2	A	J Moss	...	4	6	12	9	7	19	1	3	0	0
4	12/08/17	Everton	Stoke	1	0	H	1	0	H	N Swarbrick	...	4	1	6	7	13	10	1	1	0	0
5	12/08/17	Southampton	Swansea	0	0	D	0	0	D	M Jones	...	2	0	13	0	10	13	2	1	0	0
6	12/08/17	Watford	Liverpool	3	3	D	2	1	H	A Taylor	...	4	5	3	3	14	8	0	3	0	0
7	12/08/17	West Brom	Bournemouth	1	0	H	1	0	H	R Madley	...	6	2	8	2	15	3	3	1	0	0
8	13/08/17	Man United	West Ham	4	0	H	1	0	H	M Atkinson	...	6	1	11	1	19	7	2	2	0	0
9	13/08/17	Newcastle	Tottenham	0	2	A	0	0	D	A Marriner	...	3	6	5	7	6	10	1	2	1	0

<u>שתי השאלות הראשונות שנרצה לענות עליהן:</u>

1. חיזוי ניצחון/תיקו/הפסד על קבוצת הבית במשחק נתון.
2. האם אפשר ללמוד מתוצאת המחצית משהו לגבי תוצאת הסיום.

מכיוון שבעמודות 'FTR', 'HTR' הערכים בצורת 'H','D','A' נמיר אותם לערך מספרי: 2 לניצחון, 1 לתיקו ו-0 להפסד (ההתייחסות היא על קבוצת הבית).

כעת נוסיף עמודה חדשה שמכילה את מספר המחזור בליגה (יש 38 מחזורים), כדי שבהמשך נוכל לנרמל את הדאטה לפי מחזור. למשל, אם אחרי 3 מחזורים לקבוצה מסויימת יש 7 נקודות, אז נרצה לקבל את זה כ-2.33 נקודות למחזור.

עכשיו נחשב את הפרש השערים של קבוצת הבית וקבוצת החוץ מנורמל לפי מספר המחזור. נעשה זאת גם לכמות השערים של משחק שלם וגם לכמות השערים עד המחצית.

```python
# gets the goals diff agg arranged by teams and matchweek

def get_goals_diff(playing_stat):
    # create a dictionary with team names as keys
    teams = {}
    teams_half = {}
    for i in playing_stat.groupby('HomeTeam').mean().T.columns:
        teams[i] = []
        teams_half[i] = []

    # the value corresponding to keys is a list containing the match location.
    for i in range(len(playing_stat)):

        # home team goal diff = goals scored - goals conceded
        HTGD = playing_stat.iloc[i]['FTHG'] - playing_stat.iloc[i]['FTAG']

        # away team goal diff = goals scored - goals conceded
        ATGD = playing_stat.iloc[i]['FTAG'] - playing_stat.iloc[i]['FTHG']

        teams[playing_stat.iloc[i].HomeTeam].append(HTGD)
        teams[playing_stat.iloc[i].AwayTeam].append(ATGD)

        # home team half time goal diff = goals scored - goals conceded
        HTHGD = playing_stat.iloc[i]['HTHG'] - playing_stat.iloc[i]['HTAG']

        # away team half time goal diff = goals scored - goals conceded
        ATHGD = playing_stat.iloc[i]['HTAG'] - playing_stat.iloc[i]['HTHG']

        teams_half[playing_stat.iloc[i].HomeTeam].append(HTHGD)
        teams_half[playing_stat.iloc[i].AwayTeam].append(ATHGD)

    # create a dataframe for goals diff where rows are teams and cols are matchweek.
    GoalsDiff = pd.DataFrame(data=teams, index = [i for i in range(1,39)]).T
    GoalsDiff[0] = 0

    # create a dataframe for time goal goals diff where rows are teams and cols are matchweek.
    GoalsHDiff = pd.DataFrame(data=teams_half, index = [i for i in range(1,39)]).T
    GoalsHDiff[0] = 0

    # aggregate to get uptil that point
    for i in range(2,39):
        GoalsDiff[i] = GoalsDiff[i] + GoalsDiff[i-1]
        GoalsHDiff[i] = GoalsHDiff[i] + GoalsHDiff[i-1]
```

```
    j = 0
    HTGD = []
    ATGD = []
    HTHGD = []
    ATHGD = []

    for i in range(380):
        ht = playing_stat.iloc[i].HomeTeam
        at = playing_stat.iloc[i].AwayTeam
        HTGD.append(GoalsDiff.loc[ht][j])
        ATGD.append(GoalsDiff.loc[at][j])
        HTHGD.append(GoalsHDiff.loc[ht][j])
        ATHGD.append(GoalsHDiff.loc[at][j])

        if ((i + 1)% 10) == 0:
            j = j + 1

    playing_stat['HTGD'] = HTGD
    playing_stat['HTGD'] = playing_stat['HTGD'] / playing_stat.MW

    playing_stat['ATGD'] = ATGD
    playing_stat['ATGD'] = playing_stat['ATGD'] / playing_stat.MW

    playing_stat['HTHGD'] = HTHGD
    playing_stat['HTHGD'] = playing_stat['HTHGD'] / playing_stat.MW

    playing_stat['ATHGD'] = ATHGD
    playing_stat['ATHGD'] = playing_stat['ATHGD'] / playing_stat.MW

    return playing_stat
```

עכשיו נעשה את אותו נרמול לפי מחזור הפרש הנקודות בין הקבוצות שמשחקות אחת נגד השניה.
הקוד זהה לכן לא נכניס אותו כאן.

נשים לב כי אנחנו לא רוצים לחזות תוצאה של משחק נתון לפי כל היסטורית המשחקים של הקבוצה, מכיוון שיש לנו 18 עונות בהן שחקנים ומאמנים מתחלפים, קבוצות נקנות על ידי בעלים עשירים ולכן הקבוצה משנה את פניה, וכן הלאה. אנחנו נסתכל חמישה משחקים אחורה כל פעם כדי לחזות מה יקרה, כנהוג בהרבה אפליקציות ואתרי ספורט והימורים.
במהלך העבודה על הפרויקט היה לנו בעיה שבחלק מהמודלים קיבלנו תוצאות ממש גבוהות (גם קבענו פגישה בזום לגבי זה), ועד כמה שהצלחנו להבין זה קרה מכיוון שממש עשינו groupby לפי כל ההיסטוריה של הקבוצות ולכן היה לנו זליגת נתונים. פתרנו זאת על ידי כך שאנחנו נסתכל תמיד על ההיסטוריה הקרובה של הקבוצות, מה שיותר יכול ללמד אותנו על הפורמה בה הקבוצה נמצאת.

```python
def add_form_result(playing_stat, num):
    form_result = get_match_result(playing_stat)
    temp_result = form_result.copy()

    form_half = get_half_result(playing_stat)
    temp_half = form_half.copy()

    for i in range(num, 39):
        temp_result[i] = ''
        temp_half[i] = ''
        j = 0
        while j < num:
            temp_result[i] += form_result[i-j]
            temp_half[i] += form_half[i-j]
            j += 1

    final_form_result = temp_result
    h_result = ['M' for i in range(num * 10)]  # since form is not available for n MW (n*10)
    a_result = ['M' for i in range(num * 10)]

    final_form_half = temp_half
    h_half = ['M' for i in range(num * 10)]  # since form is not available for n MW (n*10)
    a_half = ['M' for i in range(num * 10)]

    j = num
    for i in range((num*10),380):
        ht = playing_stat.iloc[i].HomeTeam
        at = playing_stat.iloc[i].AwayTeam

        past_result = final_form_result.loc[ht][j]          # get past n results
        h_result.append(past_result[num-1])                 # 0 index is most recent

        past_result = final_form_result.loc[at][j]          # get past n results
        a_result.append(past_result[num-1])                 # 0 index is most recent

        past_half = final_form_half.loc[at][j]              # get past n results
        a_half.append(past_half[num-1])                     # 0 index is most recent

        if ((i + 1)% 10) == 0:
            j = j + 1

    playing_stat['HM' + str(num)] = h_result
    playing_stat['AM' + str(num)] = a_result

    playing_stat['HMHALF' + str(num)] = h_half
    playing_stat['AMHALF' + str(num)] = a_half

    return playing_stat
```

נעביר לפונקציה את המספרים 1,2,3,4,5 כפרמטר num, כדי לקבל את חמשת המחזורים האחרונים.

בס"ד

עכשיו כשיש לנו ביחד את חמשת המחזורים האחרונים, נבצע שוב חישוב של הפרש הנקודות בין הקבוצות בחמשת המחזורים האלו, לפי תוצאה סופית ולפי תוצאת המחצית מנורמלים לפי מספר המחזור.

```
playing_stat['HTFormTemp'] = playing_stat['HM1'] + playing_stat['HM2'] + playing_stat['HM3'] + playing_stat['HM4'] + playing_stat['HM5']
playing_stat['ATFormTemp'] = playing_stat['AM1'] + playing_stat['AM2'] + playing_stat['AM3'] + playing_stat['AM4'] + playing_stat['AM5']

playing_stat['HTFormPts'] = playing_stat['HTFormTemp'].apply(get_form_points)
playing_stat['ATFormPts'] = playing_stat['ATFormTemp'].apply(get_form_points)

playing_stat['DiffFormPts'] = (playing_stat['HTFormPts'] - playing_stat['ATFormPts']) / playing_stat.MW

playing_stat['HTHalfFormTemp'] = playing_stat['HMHALF1'] + playing_stat['HMHALF2'] + playing_stat['HMHALF3'] + playing_stat['HMHALF4'] + playing_stat['HMHALF5']
playing_stat['ATHalfFormTemp'] = playing_stat['AMHALF1'] + playing_stat['AMHALF2'] + playing_stat['AMHALF3'] + playing_stat['AMHALF4'] + playing_stat['AMHALF5']

playing_stat['HTHalfFormPts'] = playing_stat['HTHalfFormTemp'].apply(get_form_points)
playing_stat['ATHalfFormPts'] = playing_stat['ATHalfFormTemp'].apply(get_form_points)

playing_stat['DiffHalfFormPts'] = (playing_stat['HTHalfFormPts'] - playing_stat['ATHalfFormPts']) / playing_stat.MW
```

כעת נייצר data set חדש ללא העמודות הלא הכרחיות:

```
# remove unnecessary columns for dataset2 - match result
dataset2 = dataset.copy().drop(columns =['Date','HomeTeam','AwayTeam','FTHG','FTAG','HTHG','HTAG','HTR','Referee','HS','AS',
                                         'HST','AST','HC','AC','HF','AF','HY','AY','HR','AR','MW','HTFormTemp','ATFormTemp','HTFormPts','ATFormPts',
                                         'HTHalfFormPts','ATHalfFormPts'])

dataset2.keys()
✓ 0.7s
Index(['Unnamed: 0', 'FTR', 'HTGD', 'ATGD', 'HTHGD', 'ATHGD', 'HTPD', 'ATPD',
       'HM1', 'AM1', 'HMHALF1', 'AMHALF1', 'HM2', 'AM2', 'HMHALF2', 'AMHALF2',
       'HM3', 'AM3', 'HMHALF3', 'AMHALF3', 'HM4', 'AM4', 'HMHALF4', 'AMHALF4',
       'HM5', 'AM5', 'HMHALF5', 'AMHALF5', 'DiffFormPts', 'HTHalfFormTemp',
       'ATHalfFormTemp', 'DiffHalfFormPts'],
      dtype='object')
```

נמיר את תוכן העמודות של חמשת המחזורים האחרונים ממחרוזת לערך מספרי, כדי שנוכל לעבוד עם זה.

נפריד את ה-data set לפיצ'רים ומטרה, ונחלק לקבוצות אימון ומבחן:

```
# match result
# separate into feature set and target variable (FTR)
x_all_result = dataset2.drop(['FTR'],1)
y_all_result = dataset2['FTR']
✓ 0.1s

from sklearn.model_selection import train_test_split

# shuffle and split the dataset into training and testing set
x_train_result, x_test_result, y_train_result, y_test_result = train_test_split(x_all_result, y_all_result, test_size = 0.35,
                                                                                 random_state = 42, stratify = y_all_result)
```

"ALL MODELS ARE WRONG
But Some Are Useful"
George E. P. Box

כעת נריץ שישה מודלים ונראה מה יהיו התוצאות:

```
RandomForestClassifier()
match result:
              precision    recall  f1-score   support

           0       0.49      0.41      0.45       669
           1       0.31      0.07      0.12       613
           2       0.54      0.82      0.65      1112

    accuracy                           0.51      2394
   macro avg       0.45      0.43      0.41      2394
weighted avg       0.47      0.51      0.46      2394
```

```
KNeighborsClassifier()
match result:
              precision    recall  f1-score   support

           0       0.30      0.35      0.32       669
           1       0.26      0.24      0.25       613
           2       0.47      0.44      0.46      1112

    accuracy                           0.36      2394
   macro avg       0.34      0.34      0.34      2394
weighted avg       0.37      0.36      0.37      2394
```

```
SVC()
match result:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       669
           1       0.00      0.00      0.00       613
           2       0.46      1.00      0.63      1112

    accuracy                           0.46      2394
   macro avg       0.15      0.33      0.21      2394
weighted avg       0.22      0.46      0.29      2394
```

```
LogisticRegression()
match result:
              precision    recall  f1-score   support

           0       0.48      0.48      0.48       669
           1       0.57      0.01      0.03       613
           2       0.55      0.84      0.66      1112

    accuracy                           0.53      2394
   macro avg       0.53      0.44      0.39      2394
weighted avg       0.53      0.53      0.45      2394
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
match result:
              precision    recall  f1-score   support

           0       0.45      0.40      0.43       669
           1       0.33      0.17      0.22       613
           2       0.56      0.76      0.65      1112

    accuracy                           0.51      2394
   macro avg       0.45      0.44      0.43      2394
weighted avg       0.47      0.51      0.48      2394
```

```
GaussianNB()
match result:
              precision    recall  f1-score   support

           0       0.34      0.46      0.39       669
           1       0.27      0.32      0.29       613
           2       0.54      0.38      0.44      1112

    accuracy                           0.38      2394
   macro avg       0.38      0.38      0.38      2394
weighted avg       0.42      0.38      0.39      2394
```

אפשר לראות שהתוצאות לא מספיק טובות, ונרצה לשפר את החיזוי.
כעת בניסיון לענות על שאלה 2: "האם אפשר ללמוד מתוצאת המחצית משהו לגבי תוצאת הסיום", נבצע שוב בדיוק את אותם השלבים ונריץ את אותם הפונקציות, אך את שורות הקוד של מה שקשור למחצית נשים בהערה. נרצה לראות אם תוצאת החיזוי משתפרת:

```
RandomForestClassifier()
match result:
              precision    recall  f1-score   support

           0       0.63      0.65      0.64       669
           1       0.42      0.26      0.32       613
           2       0.69      0.81      0.74      1112

    accuracy                           0.63      2394
   macro avg       0.58      0.58      0.57      2394
weighted avg       0.60      0.63      0.61      2394
```

```
KNeighborsClassifier()
match result:
              precision    recall  f1-score   support

           0       0.30      0.35      0.33       669
           1       0.25      0.24      0.25       613
           2       0.49      0.46      0.48      1112

    accuracy                           0.37      2394
   macro avg       0.35      0.35      0.35      2394
weighted avg       0.38      0.37      0.38      2394
```

```
SVC()
match result:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       669
           1       0.00      0.00      0.00       613
           2       0.46      1.00      0.63      1112

    accuracy                           0.46      2394
   macro avg       0.15      0.33      0.21      2394
weighted avg       0.22      0.46      0.29      2394
```

```
LogisticRegression()
match result:
              precision    recall  f1-score   support

           0       0.64      0.67      0.66       669
           1       0.41      0.23      0.29       613
           2       0.68      0.83      0.75      1112

    accuracy                           0.63      2394
   macro avg       0.58      0.58      0.57      2394
weighted avg       0.60      0.63      0.61      2394
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
match result:
              precision    recall  f1-score   support

           0       0.63      0.61      0.62       669
           1       0.38      0.28      0.32       613
           2       0.67      0.78      0.72      1112

    accuracy                           0.60      2394
   macro avg       0.56      0.56      0.55      2394
weighted avg       0.59      0.60      0.59      2394
```

```
GaussianNB()
match result:
              precision    recall  f1-score   support

           0       0.63      0.64      0.63       669
           1       0.40      0.44      0.42       613
           2       0.75      0.71      0.73      1112

    accuracy                           0.62      2394
   macro avg       0.59      0.60      0.59      2394
weighted avg       0.63      0.62      0.62      2394
```

אפשר לראות שהתוצאות השתפרו למעט במודלים SVM, KNN.
ננסה לנתח למה זה קרה:
נבדוק ב-data set הכללי שלנו לפני מחיקת העמודות הלא רלוונטיות, את העמודות 'FTR', 'HTR' שכזכור 2 מסמן ניצחון לקבוצת הבית, 1 מסמן תיקו, 0 מסמן הפסד לקבוצת הבית.

```
print(dataset['FTR'].value_counts())
print(dataset['HTR'].value_counts())
```

0.3s

```
2    3176
0    1913
1    1751
Name: FTR, dtype: int64
1    2853
2    2415
0    1572
Name: HTR, dtype: int64
```

ניתן לראות שבעוד שבמשחק מלא קבוצת הבית מנצחת כמעט בחצי מהמשחקים (יש כ-6800 משחקים במאגר), ההתפלגות על התוצאה במחצית המשחק הרבה יותר מאוזנת, ולכן הוספת הפיצ'ר של מחצית המשחק מוסיפה רעש ומקטינה את אחוזי הדיוק.

אלגוריתם SVM- מתפקד פחות טוב כאן מכיוון שכאשר ישנם רעשי רקע כלומר הסיווגים חופפים הרבה אחד על השני, אזי זה גורם לאלגוריתם הזה להחזיר תשובות הרבה פחות טובות.
אלגוריתם KNN - לא מתפקד טוב כאשר ישנם הרבה מימדים, ולכן כאשר אנחנו מוסיפים מימד אנחנו מורידים את איכות התוצאות.

לעומת זאת, שאר האלגוריתמים Random Forest, Naive Base, Logistic Regression, XGBoost הם אלגוריתמים שטובים מאוד כאשר ישנם הרבה מימדים, ולכן אנחנו מקבלים מהם תוצאות טובות מאוד.

כידוע, בתחנות ההימורים כשבאים להמר על מספר של כרטיסים וקרנות, כמעט תמיד ההימור הוא על טווח, מכיוון שקשה מאוד לפגוע במספר מדויק של כרטיסים וקרנות. השאלות הבאות שלנו יעסקו בחיזוי מספר הכרטיסים והקרנות במשחק נתון. מכיוון שכל פעם טווח ההימור משתנה בהתאם לקבוצות, למשל במשחק של קבוצה שמכסחת המספר עליו מהמרים מעל או מתחת יהיה גבוה יותר מאשר משחק בין קבוצות שהסטטיסטיקה שלהן לגבי כרטיסים הוא נמוך. בעקבות כך, החלטנו לא לנסות לנסח לכל משחק מחדש מה יהיו הטווח כרטיסים וקרנות שלו (חברות ההימורים מעסיקות אנליסטים מומחים כדי לקבוע דברים כגון אלה), אלא מכיוון שיש לנו את הנתונים על כמות הכרטיסים והקרנות במשחק, ניקח את המספר המנורמל של כרטיסים וקרנות למשחק ואותו אנו ננסה לחזות.

<u>השאלות הבאות עליהן נרצה לענות:</u>

3. חיזוי מספר הכרטיסים הכולל (צהובים ואדומים) במשחק נתון.
4. האם בהינתן שם השופט של המשחק אפשר ללמוד ולדייק את חיזוי מספר הוצאת הכרטיסים.

לצורך שאלות אלו נשתמש בנתונים הבאים: Referee, HF, AF, HY, AY, HR, AR

גם כאן נשתמש באותם הטכניקות והפונקציות כמו שעשינו לעיל. נאסוף את כמות הכרטיסים לקבוצת הבית ולקבוצת החוץ, וננרמל לפי מספר המחזור. נעשה זאת גם על כמות העבירות (פאולים) לקבוצת הבית ולקבוצת החוץ מנורמל לפי מחזור. בתור התחלה נרצה לקחת את הפיצ'ר של כמות העבירות כי יש יסוד להניח שאולי מספר עבירות גורר בערך מספר כרטיסים. בנוסף נוסיף עמודה של כמות העבירות של שתי הקבוצות מנורמל לפי מספר מחזור. עמודת המטרה תהיה עמודה נוספת של כמות הכרטיסים של שתי הקבוצות מנורמל לפי מספר מחזור, כלומר מספר הכרטיסים המנורמל המשותף שלהם למשחק.

כעת נייצר data set חדש ללא העמודות הלא הכרחיות:

```
# remove unnecessary columns for dataset3 - cards
dataset3 = dataset.copy().drop(columns =['Date','HomeTeam','AwayTeam','FTHG','FTAG','FTR','HTHG','HTAG','HTR','Referee','HS','AS',
                                         'HST','AST','HC','AC','HF','AF','HY','AY','HR','AR','MW',
                                         'HTOTCRD','ATOTCRD','HTOTFOUL','ATOTFOUL'])

dataset3.keys()
```

✓ 0.6s

```
Index(['Unnamed: 0', 'HTCAD', 'ATCAD', 'HTFD', 'ATFD', 'HCARD1', 'ACARD1',
       'HCARD2', 'ACARD2', 'HCARD3', 'ACARD3', 'HCARD4', 'ACARD4', 'HCARD5',
       'ACARD5', 'HF1', 'AF1', 'HF2', 'AF2', 'HF3', 'AF3', 'HF4', 'AF4', 'HF5',
       'AF5', 'TOTALCARDS', 'TOTALFOULS'],
      dtype='object')
```

נפריד את ה-data set לפיצ'רים ומטרה, ונחלק לקבוצות אימון ומבחן:

```
# cards
# separate into feature set and target variable (TOTALCARDS)
x_all_cards = dataset3.drop(['TOTALCARDS'],1)
y_all_cards = dataset3['TOTALCARDS']
```

```
from sklearn.model_selection import train_test_split

# shuffle and split the dataset into training and testing set
x_train_cards, x_test_cards, y_train_cards, y_test_cards = train_test_split(x_all_cards, y_all_cards, test_size = 0.35, random_state = 42)
```

כעת נריץ שישה מודלים ונראה מה יהיו התוצאות:

```
RandomForestClassifier()
match cards:
              precision    recall  f1-score   support

         0.0       0.94      0.96      0.95      2001
         1.0       0.56      0.51      0.53       267
         2.0       0.45      0.42      0.43        69
         3.0       0.30      0.13      0.18        23
         4.0       0.14      0.17      0.15        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
         9.0       0.00      0.00      0.00         0
        10.0       0.00      0.00      0.00         1

    accuracy                           0.87      2394
   macro avg       0.22      0.20      0.20      2394
weighted avg       0.86      0.87      0.87      2394
```

```
KNeighborsClassifier()
match cards:
              precision    recall  f1-score   support

         0.0       0.92      0.95      0.93      2001
         1.0       0.48      0.42      0.44       267
         2.0       0.44      0.33      0.38        69
         3.0       0.35      0.30      0.33        23
         4.0       0.40      0.33      0.36        18
         5.0       0.20      0.17      0.18         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.86      2394
   macro avg       0.28      0.25      0.26      2394
weighted avg       0.84      0.86      0.85      2394
```

```
SVC()
match cards:
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91      2001
         1.0       0.00      0.00      0.00       267
         2.0       0.00      0.00      0.00        69
         3.0       0.00      0.00      0.00        23
         4.0       0.00      0.00      0.00        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.84      2394
   macro avg       0.08      0.10      0.09      2394
weighted avg       0.70      0.84      0.76      2394
```

```
LogisticRegression()
match cards:
              precision    recall  f1-score   support

         0.0       0.87      0.99      0.93      2001
         1.0       0.10      0.04      0.05       267
         2.0       0.00      0.00      0.00        69
         3.0       0.00      0.00      0.00        23
         4.0       0.00      0.00      0.00        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.83      2394
   macro avg       0.10      0.10      0.10      2394
weighted avg       0.74      0.83      0.78      2394
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
match cards:
              precision    recall  f1-score   support

         0.0       0.94      0.96      0.95      2001
         1.0       0.59      0.55      0.57       267
         2.0       0.37      0.30      0.33        69
         3.0       0.27      0.30      0.29        23
         4.0       0.26      0.33      0.29        18
         5.0       0.00      0.00      0.00         6
         6.0       0.33      0.20      0.25         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.88      2394
   macro avg       0.28      0.27      0.27      2394
weighted avg       0.87      0.88      0.87      2394
```

בס"ד

```
GaussianNB()
match cards:
              precision    recall  f1-score   support

         0.0       0.93      0.95      0.94      2001
         1.0       0.56      0.23      0.33       267
         2.0       0.14      0.14      0.14        69
         3.0       0.02      0.04      0.03        23
         4.0       0.05      0.17      0.08        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.03      0.33      0.05         3
         8.0       0.04      1.00      0.07         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.83      2394
   macro avg       0.18      0.29      0.16      2394
weighted avg       0.85      0.83      0.83      2394
```

נבדוק עכשיו ללא התחשבות בעבירות ונראה מה נקבל:

```
RandomForestClassifier()
match cards:
              precision    recall  f1-score   support

         0.0       0.90      0.97      0.93      2001
         1.0       0.45      0.19      0.26       267
         2.0       0.37      0.29      0.33        69
         3.0       0.22      0.26      0.24        23
         4.0       0.27      0.22      0.24        18
         5.0       0.00      0.00      0.00         6
         6.0       0.17      0.20      0.18         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.85      2394
   macro avg       0.24      0.21      0.22      2394
weighted avg       0.81      0.85      0.82      2394
```

```
KNeighborsClassifier()
match cards:
              precision    recall  f1-score   support

         0.0       0.93      0.95      0.94      2001
         1.0       0.49      0.47      0.48       267
         2.0       0.45      0.35      0.39        69
         3.0       0.40      0.26      0.32        23
         4.0       0.33      0.22      0.27        18
         5.0       0.33      0.17      0.22         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.86      2394
   macro avg       0.29      0.24      0.26      2394
weighted avg       0.85      0.86      0.85      2394
```

```
SVC()
match cards:
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91      2001
         1.0       0.00      0.00      0.00       267
         2.0       0.00      0.00      0.00        69
         3.0       0.00      0.00      0.00        23
         4.0       0.00      0.00      0.00        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.84      2394
   macro avg       0.08      0.10      0.09      2394
weighted avg       0.70      0.84      0.76      2394
```

```
LogisticRegression()
match cards:
              precision    recall  f1-score   support

         0.0       0.86      0.99      0.92      2001
         1.0       0.10      0.03      0.05       267
         2.0       0.00      0.00      0.00        69
         3.0       0.00      0.00      0.00        23
         4.0       0.00      0.00      0.00        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.83      2394
   macro avg       0.10      0.10      0.10      2394
weighted avg       0.73      0.83      0.78      2394
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
match cards:
              precision    recall  f1-score   support

         0.0       0.92      0.96      0.94      2001
         1.0       0.48      0.38      0.42       267
         2.0       0.41      0.32      0.36        69
         3.0       0.22      0.22      0.22        23
         4.0       0.38      0.28      0.32        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.00      0.00      0.00         3
         8.0       0.00      0.00      0.00         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.85      2394
   macro avg       0.24      0.21      0.23      2394
weighted avg       0.84      0.85      0.84      2394
```

```
GaussianNB()
match cards:
              precision    recall  f1-score   support

         0.0       0.91      0.92      0.92      2001
         1.0       0.36      0.07      0.12       267
         2.0       0.08      0.14      0.11        69
         3.0       0.02      0.04      0.03        23
         4.0       0.06      0.22      0.09        18
         5.0       0.00      0.00      0.00         6
         6.0       0.00      0.00      0.00         5
         7.0       0.03      0.33      0.05         3
         8.0       0.03      1.00      0.06         1
        10.0       0.00      0.00      0.00         1

    accuracy                           0.79      2394
   macro avg       0.15      0.27      0.14      2394
weighted avg       0.81      0.79      0.79      2394
```

התוצאות ממש זהות (עד כדי אפסילון) כאשר מריצים את המודלים עם עבירות ובלי עבירות, ולכן נוכל להסיק שכמות העבירות **לא משפיעה** על מספר הכרטיסים.
אנחנו יכולים לראות שהאלגוריתמים SVM ו- Logistic Regression מתפקדים כמו עיוורים ומנסים לנחש שהכל זה 0 כרטיסים לפי רוב הכרטיסים שיש לנו ב-data set.
ארבעת האלגוריתמים האחרים לעומת זאת מצליחים יותר טוב לנחש גם במספר כרטיסים אחר (שיש לנו פחות מהם ב-data set).
כנראה שזה נובע מכיוון שכמו שאמרנו לעיל SVM לא מסתדר עם data set כמו שלנו שיש הרבה רעשי רקע.
וכעט ה- Logistic Regression גם הוא נכשל מכיוון שכנראה אין קו לינארי אחד שיכול לפצל את הdata כמו שצריך.
לעומת זאת שאר האלגוריתמים שלנו מצליחים לנחש ברמה טובה כאשר Random forest שוב מוביל.

<u>כעת נתייחס לשאלה מספר 4 בקשר לשופטים:</u>
לצערנו לא נוכל לענות על שאלה זו.
מספר השופטים קטן ממספר הקבוצות. בינינו את הדאטה סט שלנו בצורה שמתאימה לנתונים על משחקים כדי שנוכל לקבל תוצאות טובות יותר בשאר השאלות. כדי שנוכל להצליח לעבוד עם השופטים, אנחנו צריכים להפוך את כל הדאטה סט להיות מכוון שופטים ולא מכוון משחקים.

<u>נענה על השאלה האחרונה:</u>

5. חיזוי מספר הקרנות במשחק נתון.

לצורך שאלות אלו נשתמש בנתונים הבאים: HS, AS, HST, AST, HC, AC

נעבוד באותה השיטה. נאסוף את כמות הקרנות של קבוצת הבית ושל קבוצת החוץ, וננרמל לפי מספר המחזור. נעשה זאת גם על כמות הבעיטות לשער והבעיטות למסגרת לקבוצת הבית ולקבוצת החוץ מנורמל לפי מחזור. בתור התחלה נרצה לקחת את הפיצ'ר של כמות הבעיטות לשער והבעיטות למסגרת כי יש יסוד להניח שאולי זה גורר את מספר הקרנות. בנוסף נוסיף עמודה של כמות הקרנות של שתי הקבוצות מנורמל לפי מספר מחזור. עמודת המטרה תהיה עמודה נוספת של כמות הקרנות של שתי הקבוצות מנורמל לפי מספר מחזור, כלומר מספר הקרנות המנורמל המשותף שלהם למשחק.

כעת נייצר data set חדש ללא העמודות הלא הכרחיות:

```
# remove unnecessary columns for dataset4 - corners
dataset4 = dataset.copy().drop(columns =['Date','HomeTeam','AwayTeam','FTHG','FTAG','FTR','HTHG','HTAG','HTR','Referee','HS','AS',
                                         'HST','AST','HC','AC','HF','AF','HY','AY','HR','AR','MW',
                                         'HTOTCORNER','ATOTCORNER','HTOTSHOT','ATOTSHOT'])

dataset4.keys()
✓ 0.7s
Index(['Unnamed: 0', 'HTCOD', 'ATCOD', 'HTSHOTSD', 'ATSHOTSD', 'HCORNER1',
       'ACORNER1', 'HCORNER2', 'ACORNER2', 'HCORNER3', 'ACORNER3', 'HCORNER4',
       'ACORNER4', 'HCORNER5', 'ACORNER5', 'HSHOT1', 'ASHOT1', 'HSHOT2',
       'ASHOT2', 'HSHOT3', 'ASHOT3', 'HSHOT4', 'ASHOT4', 'HSHOT5', 'ASHOT5',
       'TOTALCORNER', 'TOTALSHOT'],
      dtype='object')
```

נפריד את ה-data set לפיצ'רים ומטרה, ונחלק לקבוצות אימון ומבחן:

```
# corners
# separate into feature set and target variable (TOTALCORNER)
x_all_corners = dataset4.drop(['TOTALCORNER'],1)
y_all_corners = dataset4['TOTALCORNER']

✓ 0.4s

from sklearn.model_selection import train_test_split

# shuffle and split the dataset into training and testing set
x_train_corners, x_test_corners, y_train_corners, y_test_corners = train_test_split(x_all_corners, y_all_corners, test_size = 0.35, random_state = 42)
```

כעת נריץ שישה מודלים ונראה מה יהיו התוצאות:

```
RandomForestClassifier()
match corners:
              precision    recall  f1-score   support

         0.0       0.84      0.79      0.81      1105
         1.0       0.67      0.77      0.72       848
         2.0       0.57      0.47      0.52       200
         3.0       0.43      0.33      0.38        72
         4.0       0.33      0.31      0.32        51
         5.0       0.31      0.20      0.24        20
         6.0       0.36      0.45      0.40        20
         7.0       0.33      0.09      0.14        11
         8.0       0.22      0.11      0.14        19
         9.0       0.11      0.09      0.10        11
        10.0       0.08      0.20      0.11         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        19.0       0.00      0.00      0.00         0
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.70      2394
   macro avg       0.20      0.18      0.18      2394
weighted avg       0.70      0.70      0.70      2394
```

```
KNeighborsClassifier()
match corners:
              precision    recall  f1-score   support

         0.0       0.82      0.84      0.83      1105
         1.0       0.70      0.71      0.70       848
         2.0       0.47      0.48      0.48       200
         3.0       0.26      0.25      0.25        72
         4.0       0.17      0.16      0.16        51
         5.0       0.20      0.10      0.13        20
         6.0       0.18      0.30      0.22        20
         7.0       0.10      0.09      0.10        11
         8.0       0.08      0.05      0.06        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.69      2394
   macro avg       0.15      0.15      0.15      2394
weighted avg       0.68      0.69      0.69      2394
```

```
SVC()
match corners:
              precision    recall  f1-score   support

         0.0       0.46      1.00      0.63      1105
         1.0       0.00      0.00      0.00       848
         2.0       0.00      0.00      0.00       200
         3.0       0.00      0.00      0.00        72
         4.0       0.00      0.00      0.00        51
         5.0       0.00      0.00      0.00        20
         6.0       0.00      0.00      0.00        20
         7.0       0.00      0.00      0.00        11
         8.0       0.00      0.00      0.00        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.46      2394
   macro avg       0.02      0.05      0.03      2394
weighted avg       0.21      0.46      0.29      2394
```

```
LogisticRegression()
match corners:
              precision    recall  f1-score   support

         0.0       0.49      1.00      0.66      1105
         1.0       0.00      0.00      0.00       848
         2.0       0.00      0.00      0.00       200
         3.0       0.00      0.00      0.00        72
         4.0       0.00      0.00      0.00        51
         5.0       0.00      0.00      0.00        20
         6.0       0.00      0.00      0.00        20
         7.0       0.00      0.00      0.00        11
         8.0       0.00      0.00      0.00        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.46      2394
   macro avg       0.02      0.05      0.03      2394
weighted avg       0.23      0.46      0.30      2394
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
match corners:
              precision    recall  f1-score   support

         0.0       0.82      0.80      0.81      1105
         1.0       0.67      0.71      0.69       848
         2.0       0.50      0.52      0.51       200
         3.0       0.35      0.25      0.29        72
         4.0       0.33      0.35      0.34        51
         5.0       0.17      0.20      0.18        20
         6.0       0.29      0.30      0.29        20
         7.0       0.00      0.00      0.00        11
         8.0       0.11      0.05      0.07        19
         9.0       0.22      0.18      0.20        11
        10.0       0.08      0.20      0.12         5
        11.0       0.20      0.14      0.17         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        17.0       0.00      0.00      0.00         0
        18.0       0.00      0.00      0.00         2
        19.0       0.00      0.00      0.00         0
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.68      2394
   macro avg       0.17      0.17      0.17      2394
weighted avg       0.68      0.68      0.68      2394
```

```
GaussianNB()
match corners:
              precision    recall  f1-score   support

         0.0       0.73      0.94      0.82      1105
         1.0       0.74      0.48      0.59       848
         2.0       0.47      0.27      0.34       200
         3.0       0.05      0.03      0.04        72
         4.0       0.18      0.27      0.22        51
         5.0       0.21      0.35      0.26        20
         6.0       0.04      0.05      0.05        20
         7.0       0.00      0.00      0.00        11
         8.0       0.09      0.26      0.13        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.10      0.14      0.12         7
        12.0       0.05      0.40      0.08         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        19.0       0.00      0.00      0.00         0
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.64      2394
   macro avg       0.13      0.15      0.13      2394
weighted avg       0.64      0.64      0.62      2394
```

נבדוק עכשיו ללא התחשבות בבעיטות ונראה מה נקבל:

```
RandomForestClassifier()
match corners:
              precision    recall  f1-score   support

         0.0       0.63      0.75      0.68      1105
         1.0       0.52      0.46      0.49       848
         2.0       0.49      0.29      0.36       200
         3.0       0.30      0.25      0.27        72
         4.0       0.32      0.20      0.24        51
         5.0       0.00      0.00      0.00        20
         6.0       0.24      0.35      0.29        20
         7.0       0.17      0.18      0.17        11
         8.0       0.13      0.11      0.12        19
         9.0       0.00      0.00      0.00        11
        10.0       0.17      0.20      0.18         5
        11.0       0.00      0.00      0.00         7
        12.0       0.11      0.20      0.14         5
        13.0       0.33      0.29      0.31         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        17.0       0.00      0.00      0.00         0
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.55      2394
   macro avg       0.16      0.16      0.16      2394
weighted avg       0.54      0.55      0.54      2394
```

```
KNeighborsClassifier()
match corners:
              precision    recall  f1-score   support

         0.0       0.82      0.83      0.83      1105
         1.0       0.71      0.71      0.71       848
         2.0       0.45      0.48      0.47       200
         3.0       0.23      0.21      0.22        72
         4.0       0.17      0.14      0.15        51
         5.0       0.07      0.05      0.06        20
         6.0       0.17      0.30      0.21        20
         7.0       0.08      0.09      0.09        11
         8.0       0.09      0.05      0.07        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.69      2394
   macro avg       0.14      0.14      0.14      2394
weighted avg       0.68      0.69      0.68      2394
```

```
SVC()
match corners:
              precision    recall  f1-score   support

         0.0       0.46      1.00      0.63      1105
         1.0       0.00      0.00      0.00       848
         2.0       0.00      0.00      0.00       200
         3.0       0.00      0.00      0.00        72
         4.0       0.00      0.00      0.00        51
         5.0       0.00      0.00      0.00        20
         6.0       0.00      0.00      0.00        20
         7.0       0.00      0.00      0.00        11
         8.0       0.00      0.00      0.00        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.46      2394
   macro avg       0.02      0.05      0.03      2394
weighted avg       0.21      0.46      0.29      2394
```

```
LogisticRegression()
match corners:
              precision    recall  f1-score   support

         0.0       0.46      1.00      0.63      1105
         1.0       0.00      0.00      0.00       848
         2.0       0.00      0.00      0.00       200
         3.0       0.00      0.00      0.00        72
         4.0       0.00      0.00      0.00        51
         5.0       0.00      0.00      0.00        20
         6.0       0.00      0.00      0.00        20
         7.0       0.00      0.00      0.00        11
         8.0       0.00      0.00      0.00        19
         9.0       0.00      0.00      0.00        11
        10.0       0.00      0.00      0.00         5
        11.0       0.00      0.00      0.00         7
        12.0       0.00      0.00      0.00         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.46      2394
   macro avg       0.02      0.05      0.03      2394
weighted avg       0.21      0.46      0.29      2394
```

בס"ד

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
match corners:
              precision    recall  f1-score   support

         0.0       0.82      0.82      0.82      1105
         1.0       0.68      0.71      0.70       848
         2.0       0.53      0.48      0.50       200
         3.0       0.38      0.26      0.31        72
         4.0       0.19      0.22      0.20        51
         5.0       0.27      0.20      0.23        20
         6.0       0.10      0.15      0.12        20
         7.0       0.29      0.18      0.22        11
         8.0       0.00      0.00      0.00        19
         9.0       0.00      0.00      0.00        11
        10.0       0.14      0.20      0.17         5
        11.0       0.00      0.00      0.00         7
        12.0       0.11      0.20      0.14         5
        13.0       0.40      0.29      0.33         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        17.0       0.00      0.00      0.00         0
        18.0       0.00      0.00      0.00         2
        19.0       0.00      0.00      0.00         0
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.69      2394
   macro avg       0.18      0.17      0.17      2394
weighted avg       0.68      0.69      0.69      2394
```

```
GaussianNB()
match corners:
              precision    recall  f1-score   support

         0.0       0.59      0.77      0.67      1105
         1.0       0.46      0.27      0.34       848
         2.0       0.41      0.07      0.12       200
         3.0       0.03      0.03      0.03        72
         4.0       0.08      0.27      0.12        51
         5.0       0.18      0.25      0.21        20
         6.0       0.05      0.10      0.07        20
         7.0       0.00      0.00      0.00        11
         8.0       0.12      0.37      0.19        19
         9.0       0.00      0.00      0.00        11
        10.0       0.14      0.20      0.17         5
        11.0       0.00      0.00      0.00         7
        12.0       0.06      0.60      0.10         5
        13.0       0.00      0.00      0.00         7
        14.0       0.00      0.00      0.00         3
        15.0       0.00      0.00      0.00         4
        16.0       0.00      0.00      0.00         2
        18.0       0.00      0.00      0.00         2
        21.0       0.00      0.00      0.00         1
        23.0       0.00      0.00      0.00         1

    accuracy                           0.47      2394
   macro avg       0.11      0.15      0.10      2394
weighted avg       0.48      0.47      0.45      2394
```

כמו שראינו כאשר ניתחנו את הכרטיסים, גם בעניין הקרנות האלגוריתמים הבעייתים הם SVM ו Logistic Regression, כנראה מאותם סיבות כמו שראינו לעיל.

KNN ו-XGBoost נותנים את אותם התוצאות עם וללא בעיטות.

Random Forest ו-Naive Base נותנים תוצאות פחות טובות בלי בעיטות. אפשר לטעון שיש קשר הדוק בין בעיטות לקרנות, ולכן הוספת הפיצ'ר לא מוסיפה לאלגוריתמים האלה רעש כי הם יודעים להתמודד עם זה.