מבוא למערכות לומדות - 236756 - תרגיל בית מס' 3

ת"ז: 300816634, 300551140

Mandatory Assignment:

:*Data Preparation*

הרצנו את מניפולציות ה־$Data\ Preparation$ מגליון שעבר:

- $Outliers$ - ע"י $Z-Score$ ומי שחורג יותר מדי, מסומן כ־$Outlier$. החלטנו לא להפטר מרשומה, אלא להפוך את הערך החורג שלה ל־NaN שבשלב ה־$Imputation$ יוחלף ע"י ערך הגיוני יותר.
- $Imputation$ - בסט ה־$Train$ לפי $Mode$ (עבור ערכים נומינליים) או $Mean$ פר לייבל, ובסט ה־$Validate$ וה־$Test$ בלי להתייחס ללייבל.
- $Scaling$ - לפי $Standard\ Scaler$.
- $Feature\ Type$ - זיהינו פיצ'רים קטגוריאליים. עבור פיצ'רים קטגוריאליים שבהם הסדר משנה, וידאנו שהקידוד הוא לפי הסדר, והשתמשנו בקידוד. עבור פיצ'רים קטגוריאליים שאינם בינאריים, והסדר לא משנה בהם, המרנו ל־$One-Hot$, כדי לא להשרות סדר.
- $Feature\ Set$ - במשימה זו ניתן לנו סט הפיצ'רים ה"נכון", ולכן השתמשנו בו. כן נתון לשיקולינו האם להשתמש בכל הפיצ'רים שנוצרו מ־Most_Important_Issue (שהומר ל־$One-Hot$). דירגנו את הפיצ'רים ב־3 שיטות. לפי MI, לפי $Wrapper$ עם SVC, ולפי $Embedded$ עם $ExtraTreesClassifier$ - ועדת עצים רנדומיים שבוחרים לפי הממוצע וכחלק מאלגוריתם האימון, מדרגים את הפיצ'רים לפי חשיבות. על שלושת השיטות האלה לקחנו ממוצע, וכך יצרנו דירוג משלנו לפיצ'רים. ניסינו ללא הפיצ'רים שדורגו נמוך (רק מבין הפיצ'רים של Most_Important_Issue, כי הם היחידים שהיו נתונים לשיקולינו) - הרצנו מספר $Classifiers$, וראינו שהתוצאות נוטות לרדת, ולכן החלטנו להשאר עם כל סט הפיצ'רים שניתן לנו, כולל כל הפיצ'רים של Most_Important_Issue.

המשימות שניתנו לנו:

- לכל מצביע, לנבא לאיזו מפלגה יצביע
- לנבא את התפלגות ההצבעות למפלגות
- לנבא את המפלגה הזוכה

ניתן לענות על 2 המשימות שניתנו לנו ע"י בדיקה של סט האימון, הרי הוא כבר מתויג, ומייצג את התפלגות הבוחרים, ולא צריך קלאסיפייר בשביל זה, כי אפשר להשתמש בסטטיסטיקה - על סט האימון המתויג - ולנבא התפלגות של מפלגה i ע"י האומד $\frac{\#\ of\ votes\ for\ i}{\#\ of\ total\ votes}$ (ולפי זה גם לנבא את המפלגה הצפויה לנצח). הסברים בחלק הבונוס.

אנחנו משתמשים בקלאסיפייר, ומנבאים על סט ה־$Test$. ההצדקה לזה היא שסט ה־$Test$ נבחר רנדומלית, ולכן גם הוא מייצג את התפלגות הבוחרים. לכן ניבוי שלנו פר בוחר בסט ה־$Test$ יתן קירוב להתפלגות המפלגות (וכך נוכל גם להחליט מי המפלגה הצפויה לזכות).

אופטימיזצית ה־$Hyperparameters$:

מטריקת $Weighted\ F1\ Score$:

לאחר ה־$Data\ Preparation$, עברנו לבדוק מודלים ע"י $\boldsymbol{Grid\ Search}$ עם $\boldsymbol{Cross-Validation}$.

מטריקת המטרה שלנו ב־$\boldsymbol{GridSearchCV}$ וגם בהמשך על סט ה־$Validation$ היא $\boldsymbol{Weighted\ F1\ Score}$ מ־2 סיבות:

- כי $Weighted$. במקרה של Multilabel Classification (המקרה שלנו - 11 מפלגות), אם התפגות ה־$Labels$ לא אחידה (בדיוק המקרה שלנו - מפלגות - חלקן קטנות וחלקן גדולות) שימוש ב־$Accuracy$ יכול להוות בעיה, מכיוון שאחוז טעות גדול על מפלגות קטנות יתכן ולא יבוא לידי ביטוי. עדיין ניתן להשיג תוצאות $Accuracy$ גבוהות, כי בסה"כ סופרים את מספר הטעויות, ולכן יתכן מספר טעויות נמוך, אבל על לייבל קטן מסוים אחוז טעויות גבוה. כלומר $Accuracy$ יכול להטות אותנו לטובת לייבל נפוץ. עבור $Recall,\ Precision$ ו־$F1$ שהוא שילוב שלהם, ניתן לבחור במצב $average='weighted'$ ולקבל שלכל לייבל משקל שווה, וכך נמנעים מההטיה הזאת.
- כי שילוב של $Precision$ ו־$Recall$ (הרי להתרכז באחד מהם היא החלטה ספציפית לבעיה, וכאן אין השלכות מיוחדות לטעות מסוימת, ולכן לקחנו $F1$ שהוא ממוצע הרמוני של שניהם).

המודלים:

החלטנו לבדוק מספר מודלים פשוטים - SVC, KNN, Decision Tree ומספר מודלים יותר מורכבים Random Forest, Gradient Boosting, Multi Layer Preceptron.

בהתחלה בדקנו הרבה מהפרמטרים, וראינו שחלקם נוטים פחות להשפיע, ובנוסף עם קריאה על הפרמטרים של כל אחד מהמודלים, וההבנה שחלקם יותר משמעותיים מאחרים, החלטנו לבדוק לכל מודל 3־2 פרמטרים ומספר ערכים יחסית קטן לכל פרמטר, על מנת שנוכל להציג את התוצאות בצורה סבירה, ועדיין לקבל תוצאות לא רעות.

בכל אחד מהפרמטרים השתדלנו להראות נקודה אופטימלית. לפני שמגיעים אליה, יש $Underfitting$, כלומר ככל שעולים לכיוון הנקודה האופטימלית, מקבלים שיפור בביצועים, ואחרי שעוברים אותה, ככל שמתרחקים ממנה מקבלים הרעה בביצועים שנובעת מ־$Overfitting$. לדוגמא עבור עצים והפרמטר $Max-Depth$. עץ מעומק קטן מדי יפגע ביכולות ההכללה. עץ מעומק גדול מדי יגרום ללמידה טובה מדי של סט האימון, ולכן נקבל $Overfitting$.

כמובן שהשתמשנו ב־$Grid\ Search$ - כלומר, נבדקו כל השילובים האפשריים של כל הפרמטרים.
לכל מודל נציג את התוצאות. מאופי הבעיה (מספר פרמטרים בין 3־2, ושמות הערכים הגדולים) קשה להציג את התוצאות בגרף. לכן נציג בטקסט, ונשתדל להסביר את התוצאות.

המודלים מוצגים מהדף הבא.

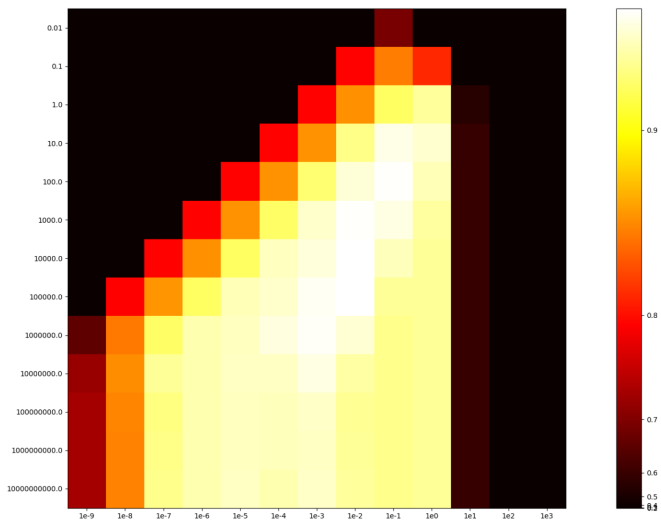
- SVC:

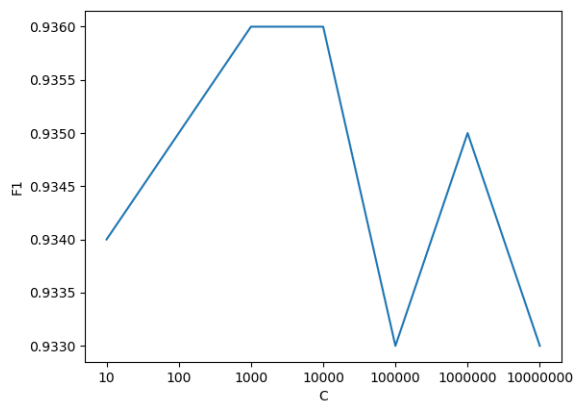
בדקנו $Kernel$ לינארי או RBF. קיבלנו באופן גורף ש־RBF טוב יותר. התוצאות עבור קרנל לינארי כצפוי, עד נקודה מסוימת מקבלים שיפור, והחל ממנה ירידה.

בדקנו $Gamma$ בין $1e-9$ לבין $1e2$ (בקפיצות כפול 10 בכל פעם, כלומר 12 ערכים).

בדקנו C בין $1e-2$ לבין $1e10$ (שוב בקפיצות כפול 10 בכל פעם, כלומר 13 ערכים).

תוצאות הניסוי (בצד ימין עבור קרנל לינארי, בצד שמאל עבור קרנל RBF):



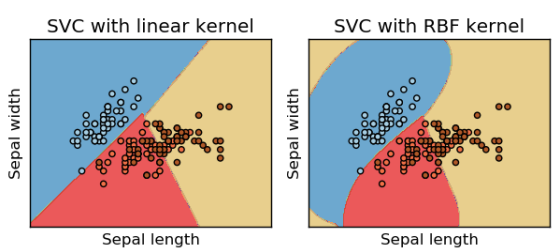


מבחינת הצבעים ב־RBF, השתמשנו ב־$PowerNorm$ כדי שיהיה קל יותר להבדיל בהבדלים המינוריים לקראת הערכים האופטימליים, שנמצאים על האלכסון (נסביר בהמשך).

המודל בעל הפרמטרים האופטימליים מבין אלה שבדקנו היה בעל $kernel =' rbf'$, $C = 100000$ ו־$gamma = 0.01$.

הסבר:

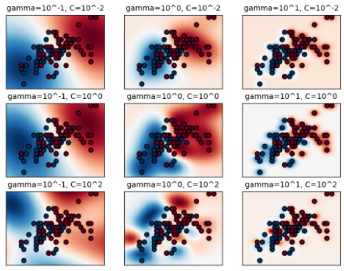
- $Kernel$ - לינארי או RBF:

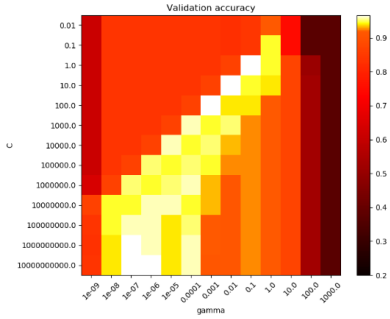


קרנל לינארי מוגבל יותר. RBF מאפשר ליצור מושגים מורכבים יותר.

- C ו־$Gamma$:

סדרת ניסוים שהתבצעה על ה־$Iris\ Dataset$ מה־$Documentation$ של $sklearn$, דומה מאוד לתוצאות שקיבלנו (למרות שכאן בודקים $Accuracy$):

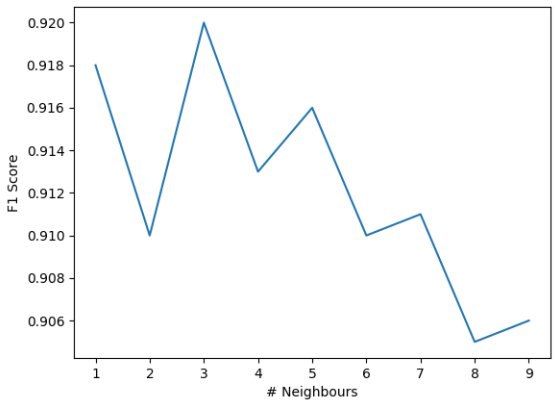




* C: מספר ה־$Support\ Vectors$ שנבחרים בזמן האימון. מייצג $Trade\ off$ בין טעות על סט האימון לבין פשטות המודל. ככל ש־C גדול יותר, רמת הדיוק על סט האימון עולה (עד גבול מסוים שגורם ל־$Overfitting$).
* $Gamma$: גודל איזור ההשפעה של כל $Support\ Vector$. $Gamma$ קטנה מאוד אומרת שאיזור ההשפעה של כל וקטור יהיה כל סט האימון, וככל שנגדיל איזור ההשפעה יקטן, וכך יאפשר יצירת צורת מורכבות יותר, כלומר נוכל ללמוד מושגים מורכבים יותר, עד גבול מסוים של למידה טובה מדי של סט האימון, כלומר $Overfitting$.

גם בתוצאות שלנו, וגם בתוצאות שהצגנו עכשיו (נשים לב שציר ה־y מסודר בסדר יורד), ניתן לראות את השפעות ה־$Overfitting$ של שני הפרמטרים.
לכן אם מגדילים אחד, מקטינים את האחר. התוצאות האופטימליות על האלכסון - המקום בו אכן מתבצע האיזון הזה.

- KNN:



המודל האופטימלי שקיבלנו בעל $k = 3$.

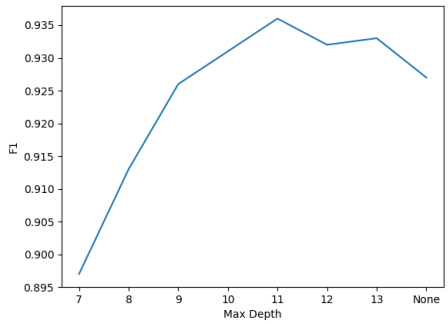
- k: עבור דגימה שנרצה לסווג, k הוא מספר השכנים הקרובים ביותר שניקח, ונסתכל על ההחלטה שלהם.

ככל ש־k קטן יותר, אנו לומדים את סט האימון טוב יותר (עבור $k = 1$ דיוק של 100% על האימון), מה שפוגע ביכולות ההכללה, וגורם ל־$Overfitting$. הגדלת K תמנע זאת (תוריד את הדיוק על סט האימון בתמורה לשיפור יכולות הכללה), אבל החל משלב מסוים תתחיל להקטין את יכולת ההכללה (כאשר מצב הקיצון הוא k שווה למספר הדגימות בסט האימון - כלומר כל הסיווגים יהיו אותו הדבר). בחירת k אי זוגי מונעת בעיות של תיקו. לפי התוצאות נראה ש־k אי זוגי אכן נותן תוצאות טובות יותר.

- $Tree\ Based$ - $Decision\ Tree$, RF & GBC:
  - $n\_estimators$ כמה שיותר יותר דיוק, על חשבון ביצועים (מהירות) - ניסינו 500 ו־1000, אין הבדל גדול מספיק כדי להצדיק את זמן הניסוים. נשארנו עם הערך הדיפולטי.
  - max_depth ככל שהעץ עמוק יותר - לומד את סט האימון טוב יותר. עמוק מדי - $Overfitting$. לא עמוק מספיק - $Underfitting$.
  - $max\_features$ - עבור RF ו־GBC - מספר הפיצ'רים שנבחרים באופן רנדומלי בכל פיצול.

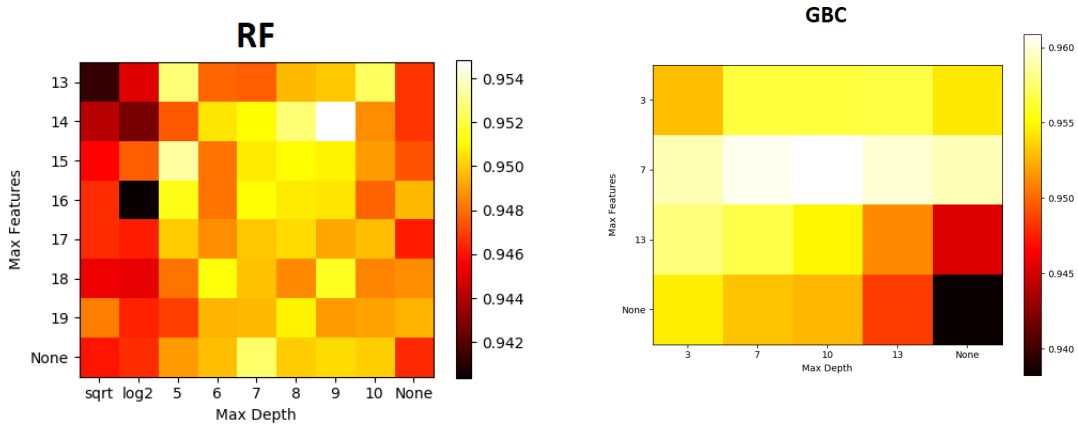
המודלים:

- $Decision\ Tree$:



המודל האופטימלי שקיבלנו בעל $Max\ Depth = 11$.
כצפוי, הערך האופטימלי באמצע, וניתן לראות את העליה עד ה־$Overfitting$ ולאחר מכן ירידה.

- RF ו־GBC:



ב־RF המודל האופטימלי שקיבלנו בעל $max\_depth = 9$, ו־$max\_features = 14$.
ב־GBC המודל האופטימלי שקיבלנו בעל $max\_depth = 7$, ו־$max\_features = 10$.
כצפוי, הערכים האופטימליים באמצע ובאלכסון, וניתן לראות את העליה עד ה־$Overfitting$ (כאשר מגדילים כל פרמטר בנפרד) ולאחר מכן ירידה, כלומר ככל שמתרחקים מהאמצע, מקבלים ירידה בביצועים.

- $Multi-layer\ Perceptron$:

```
MLP
Best parameters set found on development set:
{'alpha': 0.00015, 'hidden_layer_sizes': (500, 500)}

Grid scores on development set:
0.934 (+/-0.012) for {'alpha': 0.0001, 'hidden_layer_sizes': (15,)}
0.940 (+/-0.008) for {'alpha': 0.0001, 'hidden_layer_sizes': (15, 15)}
0.950 (+/-0.008) for {'alpha': 0.0001, 'hidden_layer_sizes': (100, 100)}
0.949 (+/-0.010) for {'alpha': 0.0001, 'hidden_layer_sizes': (500, 500)}
0.937 (+/-0.010) for {'alpha': 0.00015, 'hidden_layer_sizes': (15,)}
0.939 (+/-0.009) for {'alpha': 0.00015, 'hidden_layer_sizes': (15, 15)}
0.947 (+/-0.010) for {'alpha': 0.00015, 'hidden_layer_sizes': (100, 100)}
0.954 (+/-0.006) for {'alpha': 0.00015, 'hidden_layer_sizes': (500, 500)}
```

- **פקטור רגולריזציה:**

אנו לומדים את סט האימון במטרה לייצר מודל בעל יכולות הכללה טובות. כלומר ביצועים (במקרה זה דיוק) על מידע חדש (שלא ראינו בסט האימון).
הכללה חשובה מכיוון שסט האימון שלנו הוא בסה"כ סט סופי של דגימות - הוא אינו מכיל את כל המידע, אלא רק מהווה חלון שדרכו אנו מסתכלים על ההתפלגות האמיתית. בנוסף יתכן ומכיל רעש.
2 סיבות אלה מסבירות למה אנחנו רוצים להמנע מ־$Overfitting$, כלומר למידת סט האימון שלנו כל כך טוב, שאנחנו נותנים משקל גבוה לדגימות שראינו וגם לומדים את הרעש, וכך פוגעים בביצועים על דגימות חדשות (כלומר פוגעים ביכולות ההכללה).
כמובן ש־$Underfitting$ (לא לומדים מספיק את סט האימון) גם פוגעת בנו, מכיוון שסט האימון מכיל המון מידע על ההתפלגות האמיתית.
אם לא נלמד את סט האימון מספיק טוב, לא נלמד מספיק על ההתפלגות האמיתית, ולכן לא נוכל לקוות לביצועים טובים על דגימות חדשות.

המטרה שלנו היא כמובן למצוא נקודה אופטימלית המפשרת בין שני מצבי הקיצון האלה.

בתהליך האימון, המטרה שלנו היא למזער את פונקצית ה־$loss$.
כאשר ערכי המשקלים גדלים, אנו באופן פוטנציאלי, נותנים משקל גדול לרעש, מה שיכול לפגוע ביכולות ההכללה שלנו.
לכן אנו מוסיפים פקטור רגולריזציה, בכדי באופן מלאכותי להגדיר מחיר גבוה יותר עבור משקולות גבוהים.
כי עתה עוצמת המשקולות היא חלק מפונקצית המטרה שאותה אנו ממזערים.
ולכן נכניס משקלים גדולים יותר, רק כאשר השגיאה יורדת בהרבה.

לא אכפת לנו קצת להגדיל את השגיאה (על סט האימון), אם אפשר לקבל משקולות קטנים יותר.
מכיוון שאנו מבצעים אופטימיזציה, המשקולות שיקטנו הם גם אלה שהיו מוסיפים לנו רעש, לו היו גדולים יותר.

וככה אנחנו מגדילים טיפה את השגיאה על סט האימון, אבל מרוויחים יכולות הכללה יותר טובות.

- **מבנה הארכיטקטורה (מימדים של השכבות הנסתרות):**

יתרונות וחסרונות עבור עומק, רוחב וכמות הנוירונים גבוהים:

  - **חסרונות:**
    - **זמן אימון:**
      ככל שיש יותר נוירונים, יש יותר חישובים. זמן אימון וגם זמן שלוקח לתייג דגימות, שניהם עולים.
    - **פקטור רגולריזציה:**
      ככל שיש יותר נוירונים, המשקל שלהם בפונקצית ה־$Loss$ גדל, עד שבשלב מסוים נהיה דומיננטי יותר מפונקצית ה־$Loss$ ללא הרגולריזציה.
      לכן דרוש פקטור רגולריזציה חזק יותר (כלומר קטן יותר) כדי למנוע $Overfitting$.
  - **יתרונות:**
    ככל שהרשת יותר רחבה, עמוקה, ובעלת כמות נוירונים גדולה יותר, הביצועים שלה טובים יותר.
    - **מרחב היפותזות גדול יותר:**
      אפשר לחשוב על רשת עם פחות נוירונים, או רשת פחות עמוקה או פחות רחבה, כמקרה פרטי של העמוקה\רחבה או בעלת יותר

נוירונים ממנה. ע"י קביעת 0 עבור חלק מהפרמטרים נוכל לקבל רשת המתנהגת באופן דומה.
כאשר אנחנו מגבילים את הרשת לכמות נוירונים קטנה, אנחנו יוצרים $Bias$ מסוים, כי במרחב ההיפתוזות הגדול יותר, המכיל את הקטן ממנו, יתכן ויש היפותזות טובות יותר, שלא נמצאות בקטן.

- **סה"כ:**

להריץ ניסוי שוב עם כמות נוירונים משוגעת במקסימום. להציג גרף כאן. להגיד שקיבלנו בדיוק מה שדיברנו עליו.

השוואת המודלים לאחר אופטימיזציית ה־$Hyperparameters$:

לאחר שביצענו אופטימיזצית פרמטרים לכל אחד מהמודלים, אנו מקבלים את רשימת המודלים כאשר כל מודל בעל פרמטרים אופטימליים (מבין אלה שבדקנו), ומאומן על **כל** סט ה־$Train$.
לכל אחד מ־6 המודלים שלנו, אנו מריצים $Predict$ על סט ה־$Validation$. לאחר מכן ממיינים את המודלים לפי $F1\ Score$ (שוב זו המטריקה שלנו, מאותן הסיבות מקודם) שקיבלנו על סט ה־$Validation$, ובוחרים את המקסימלי.

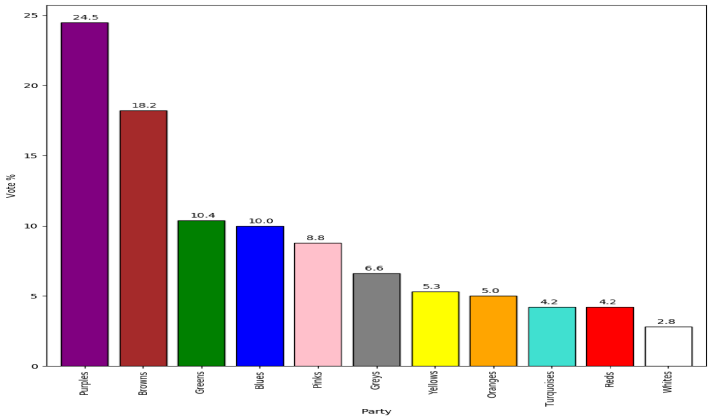
התוצאות שקיבלנו:

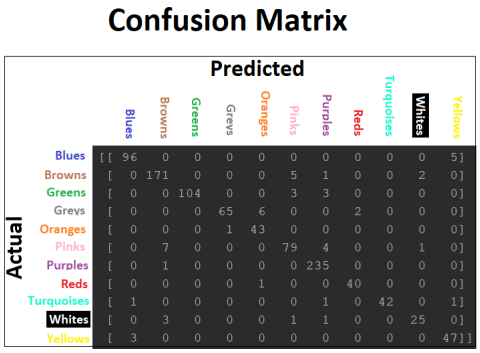
```
Models Evaluated F1 Score:
      Model Name            F1 Score
0            GBC  0.9669128064615438
1  RANDOM_FOREST  0.9558482944924336
2            MLP  0.9555412309840482
3            SVC  0.9520369037194467
4  DECISION_TREE  0.9362408051017167
5            KNN  0.9300284186768094

Best Model Is:
GBC 0.9669128064615438
```

המודל הטוב ביותר שקיבלנו הוא GBC.

שימוש במודל האופטימלי:

מההסברים בהתחלה, מספיק לנו להריץ את המודל שלנו על סט ה־$Test$, וכך נענה על 3 המשימות.
לרשימת המצביעים וההצבעה שניבאנו להם, צירפנו קובץ $results.csv$.
כפי שהסברנו בהתחלה, רשימה זו מספיקה לנו כדי להציג את התפלגות הבוחרים והמפלגה הצפויה לנצח - מפלגת ה־ **Purples**:





ה־$Test\ Error$ (כלומר $1 - Accuracy$) שלנו הוא 0.053, וה־$Accuracy$ היא 0.947.
כפי שניתן לראות, בזכות כך שהשתמשנו ב־$Weighted\ F1$ (מההסברים הקודמים שלנו), גם על הקלאסים שמופיעים פחות, יש אחוז דיוק טוב, ולא קיבלנו הטיה עבור קלאס גדול.

Non-Mandatory Assignment:

A

```python
def get_best_model(validate, models, names):
    validate_x, validate_y = split_label(validate)
    evaluated_models = [
        [model, name, f1_score(validate_y, model.predict(validate_x), average='weighted')]
        for model, name in zip(models, names)
    ]

    evaluated_models = sorted(evaluated_models, key=lambda t: t[2], reverse=True)

    print('=' * 100)
    print('Models Evaluated F1 Score:')

    # Print results in a nice format using pd.Dataframe
    print(pd.DataFrame(
        np.matrix([[name, f1] for _, name, f1 in evaluated_models]).transpose(), ['Model Name', 'F1 Score']
    ).transpose())

    print()
    best = evaluated_models[0]
    print('Best Model Is:')
    print(best[1], best[2])
    print('=' * 100)

    return best[0], best[1]
```

הקוד נמצא בקובץ הראשי - $Models.py$ והוא חלק מכל תהליך בחירת המודל (כלומר לא קובץ נפרד).
לאחר שביצענו בשלבים הקודמים אופטימיזצית פרמטרים לכל אחד מהמודלים, אנו מקבלים את רשימת המודלים כאשר כל מודל בעל פרמטרים אופטימליים (מבין אלה שבדקנו), ומאומן על **כל** סט ה־$Train$.
לכל אחד מ־6 המודלים שלנו, אנו מריצים $Predict$ על סט ה־$Validation$. לאחר מכן ממיינים את המודלים לפי $F1\ Score$ (שוב זו המטריקה שלנו, מאותן הסיבות מקודם) שקיבלנו על סט ה־$Validation$, ובוחרים את המקסימלי מביניהם.
לא בטוח אילו מסקנות יכולות להיות בחלק זה חוץ משלכתוב קוד זה יותר נחמד מלבחור באופן ידני...

B

כפי שכתבנו (ועתה נרחיב) אפשר גם להשתמש בכלים סטטיסטיים כדי לנבא את התפלגות ההצבעה ואת המפלגה הצפויה לנצח.

מכיוון שסט ה־$Train$ וה־$Validation$ הנתונים לנו מתויגים (בניגוד לסט ה־$Test$ שאמור לייצג מצב של מידע חדש שאינו מתויג, ולכן איננו מתייחסים לתיוגים שלו), וכן המידע שלנו מהווה מדגם מייצג של האוכלוסיה הכללית, על מנת להעריך עבור מפלגה את הסיכוי להצבעה אליה, נוכל להשתמש באומד.

נתייחס לכל מפלגה בנפרד. כלומר נוכל להתייחס להצבעה למפלגה כהצלחה בניסוי, ולהצבעה לכל מפלגה אחרת ככשלון.
עבור המצביע ה־$i \in [n]$ (כאשר n גודל הסט) בסט ה־$Train$ וה־$Validation$ נגדיר אינדיקטור I_i=האם הצביע למפלגה או לא.

נוכל לאמוד את אחוז ההצבעות למפלגה שלנו - p, ע"י $\bar{I} = \frac{1}{n}\sum_{i=1}^{n} I_i$.

- **חסר הטיה**:

$$E\left(\frac{1}{n}\sum_{i=1}^{n} I_i\right) = \frac{1}{n}\sum_{i=1}^{n} E\left(I_i\right) = \frac{n}{n}E\left(I\right) = E\left(I\right) = p$$

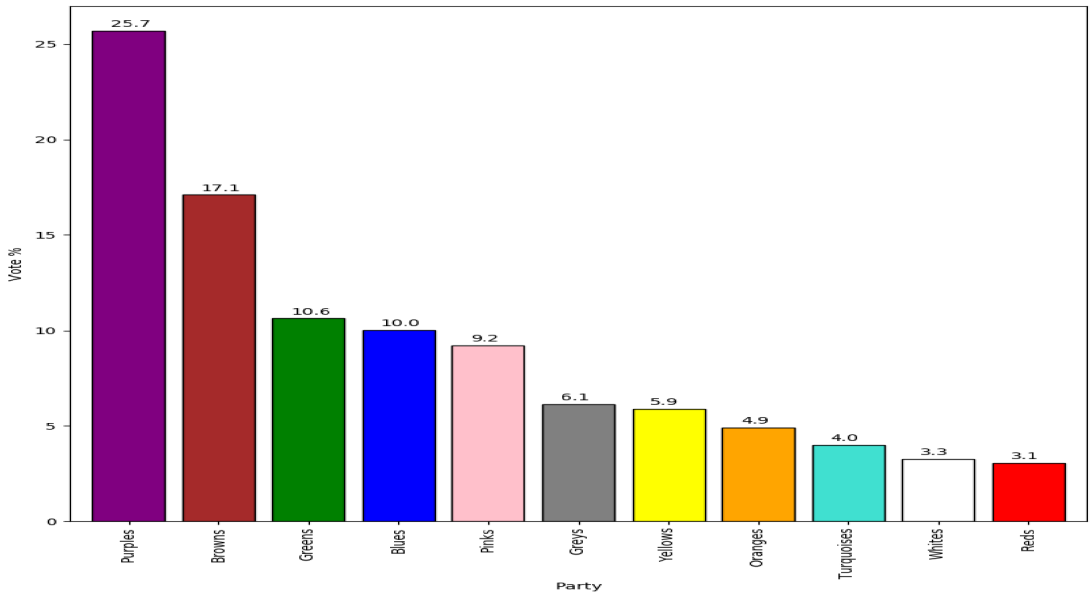
וכמובן $E\left(I\right)$ זה בדיוק מה שאנחנו מחפשים כי זה שווה לסיכוי להצביע למפלגה עבור אדם כלשהו - p.

- **עקיב**:

$$Var\left(\frac{1}{n}\sum_{i=1}^{n} I_i\right) = \frac{1}{n^2}\sum_{i=1}^{n} Var\left(I_i\right) = \frac{n}{n^2}Var\left(I\right) = \frac{1}{n}Var\left(I\right) = \frac{1}{n}p\left(1-p\right)$$

ולכן $MSE_{\bar{I}}(p) = Var\left(\bar{I}\right) + \underbrace{\left[E\left(\bar{I}\right) - p\right]^2}_{=0} = \frac{1}{n}p\left(1-p\right) \to 0$ ולכן $\bar{I}$ עקיב ל־p.

הראנו עבור מפלגה כלשהי, לכן נכון עבור כל מפלגה. כלומר על מנת לנבא את התפלגות ההצבעות לכל מפלגה, נרצה לספור את מספר המצביעים לה במדגם המייצג שנתון לנו ($Train + Validation$). הגרף בדף הבא:



זה מאוד קרוב לתוצאות שקיבלנו. יתכן שאפילו יהיה מדויק יותר, כי מרחב המדגם גדול יותר במקרה הזה ($Train + Validation$ לעומת $Test$).