מערכות לומדות – 236756

תרגיל מס' 3

שם מגיש: מיכאל נוביצקי

מס' ת"ז: 311773915

שם מגיש: אלון קווארט

מס' ת"ז: 201025228

הקבצים שהוגשו:

1. main.py – קובץ מריץ את כל התרגיל.
2. ElectionsDataPreperation.py – מודל האחראי על הכנת המידע.
3. scale_data.py – מודל המנרמל מידע.
4. modeling.py – מימוש שלב המידול.
5. CompareModels.py
6. Consts.py – קובץ המרכז רשימות, מבני נתונים, כתובות של קבצי מידע וטיפוסים שהגדרנו.
7. ElectionsData.csv – המידע הראשוני.
8. LMS.py
9. Datasets – תיקיה עם קבצי המידע כפי שנשמרו לאחר כל שלב במודל ElectionsDataPreperation
10. EX3_data – תיקיה עם קבצי הפלט כפי שנתבקשנו לחשב.

תכלס:

1. קבצי המידע הרלוונטיים (ערוכים לשלב המידול) נמצאים ב- datasets\1\filtered_and_scaled\
2. מימוש שלב המידול המצא בקובץ - modeling.py.
3. תוצאות החיזוי (על ידי מסווג יחיד לכל המשימות) נמצא בתיקיה – EX3_data\single_estimator\

חלק ראשון – Mandatory assignment

פיצול הדאטה:

פיצלנו את הדאטה לפי Stratified sampling עבור קבוצת האימון וקבוצת הולידציה, את קב' המבחן פיצלנו רגיל על מנת שהתוצאות בדיקה על קב' המבחן לא יהיו מוטות. היחסים שלפיהם פיצלנו הם: train = 70%, validation = 15%, test = 15%.

בחירת תכונות: לאחר הפיצול, בחרנו בכל אחת מהקבוצות רק את רשימת התכונות הרלוונטיות – על פי הנתון בתרגיל. ביצענו את צעד זה מוקדם ככל האפשר בכדי לחסוך חישובים מיותרים.

הפיכת תכונות נומינאליות לנומריות:

הבחנו בין מספר מקרים אפשריים:

1. תכונות עם סדר מסויים – Will_vote_only_large_party התאמנו את הערכים 1, 0, -1
2. תכונות ללא סדר – Most_Important_Issue מילאנו בעזרת שיטת Hot Spot.

מילוי מידע חסר – Imputation:

עבור מילוי התאים החסרים השתמשנו בשיטת Closest Fit על מנת למלא ערכים לפי השכן הכי קרוב בעזרת מתודה שכתבנו. את מילוי המידע החסר בכל סט (אימון, ולדיציה ומבחן) עשינו רק על סמך המידע שנמצא בסט האימון.

ניקוי רעש וערכים חריגים – Data Cleansing:

הפכנו ערכים שליליים לאותם ערכים בערך מוחלט בתכונה הבאה שבה לא חשבנו שיש הגיון בערכים שליליים:

Avg_Satisfaction_with_previous_vote

השווינו בין הקורלציה של התכונה עם תכונות אחרות לפני השינוי ואחרי השינוי והשינוי בקורלציה שקיבלנו היה מאוד נמוך ולכן השינוי כנראה לא משפיע כמעט ובכלל על הדאטאסט.

נורמליזציה – Scaling:

הבחנו בין שלושה מקרים אפשריים:

1. תכונה תכונה טרינארית – אין צורך לנרמל.
2. תכונה שההיסטוגרמה שלה נראית גאוסית – הפכנו למשתנה גאוסי בעזרת Z-Score. רשימת התכונות הללו נמצאת בקובץ Consts.setGaussianGeatures.

3. תכונה שההיסטוגרמה שלנה נראית אחידה – הפכנו למשתנה אחיד בתחום [-1,1] בעזרת Min-Max scaling. רשימת התכונות הללו נמצאת בקובץ Consts.setUniformFeatures.

בתרגיל הקודם פירטנו בצורה מורחבת על הנורמלזציה וכל שאר ההכנות שביצענו על הדאטה ולכן רק ציינו את השינויים ללא פירוט נרחב.

אימון ובחירת מודלים:

השלב כולו נמצא בקובץ modeling.py, כאשר הרצנו גם את הקובץ main.py לביצוע כל השלבים הקודמים של הכנת הדאטה. המודלים שבחרנו לאמן הם: Decision Tree, Random Forest, SVM, KNN. רצינו גם לבדוק את Naïve-Bayes, אבל לא כל התכונות הן עם התפלגות זהה ולכן לא השתמשנו בו.

הערה- בקובץ Modeling.py שמנו בהערה את SVC מכיוון ששלב מציאת ההיפר-פרמטרים עבורו לוקח הרבה זמן והביצועים שקיבלנו עבורו נמוכים משמעותית מהביצועים עבור יער אקראי.

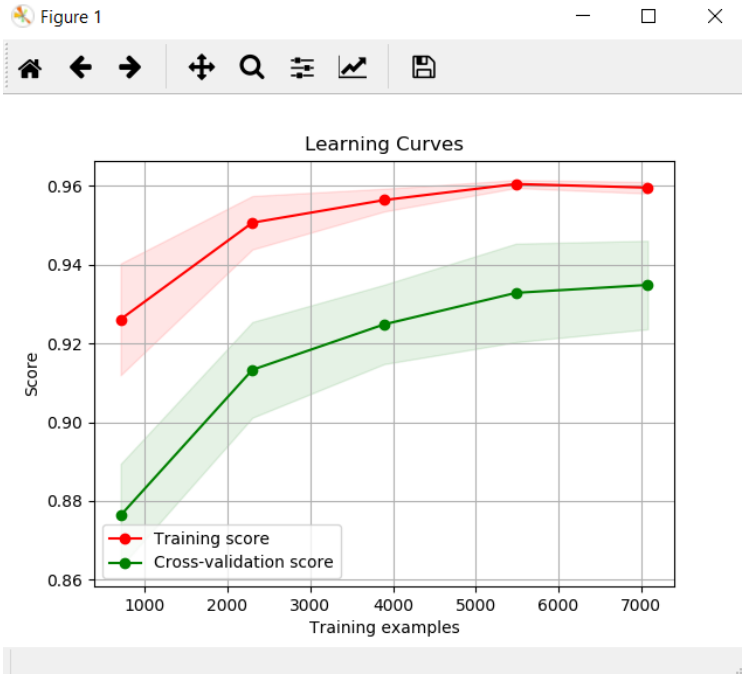
נתאר את השלבים שהתבצעו (בחלק הבא יש פירוט מורחב יותר לגבי הקוד):

1. טעינת הדאטהסט כפי שתואר בסוף שלב הנרמול.

2. בניית מודלים לחיפוש היפר-פרמטרים עבור כל אחד מהמסווגים הנבחרים. מכיוון שמספר האפשרויות הוא גדול מאוד, בלתי אפשרי לבדוק את כל האופציות עבור הפרמטרים ולכן השתמשנו במחלקה RandomizedSearchCV על מנת לבדוק בצורה אקראית חלק מה-grid שהגדרנו (ה-grid עבור כל מסווג מוגדר בתוך הקובץ Consts.py בתוך מחלקה בשם RandomGrid). לכל מסווג בנינו מודל חיפוש והרצנו למשך 30 איטרציות, כאשר סוג הציון אותו אנו מנסים למקסם הוא Accuracy ובכל פעם בודקים score על ידי ביצוע 10-Fold Cross Validation. נפרט את המניעים ונקודות אליהן התייחסנו בבחירת מודל Accuracy:

a. נדרשנו לענות על כלל המשימות באמצעות אותו המסווג, ולכן חיפשנו מודל ציון בעל תאימות רבה ככל הניתן לכלל המשימות. ראינו כאן יתרון עבור המודל בכך שהוא מתייחס לכל התיוגים בצורה שווה.
b. בקבוצת האימון אין רוב מוחלט (מעל 50%) של הצבעות, ולכן לא ראינו מצב בו אנו מטים בצורה גסה את התוצאות רק וזאת רק על פי מודל הציון. זהו חסרון ידוע של מודל זה, אשר כאן איננו מקבל ביטוי.

3. לאחר בחירת הפרמטרים למסווגים, בדקנו האם כדאי לאמן את המסווג על פי קבוצת הולידאציה וקבוצת האימון יחדיו. להלן הדפסת גרף המראה את אחוז הדיוק כפונקציה של גודל קב' האימון – כפי שניתן לראות קיבלנו גרף מונוטוני עולה ולכן העדפנו לאמן את המסווג הסופי על קב' האימון וקב' הולידציה.



4. בחירת המסווג היעיל ביותר על פי מודל המטריקה הנבחר – במקרה שלנו יער אקראי.
מסווג זה אומן שוב על ידי סט האימון והולידציה יחדיו. בשלב הבא בחנו את תוצאות המסווג על ידי חיזוי על סט המבחן. אלו הן תוצאות המסווג:
   a. המפלגה המנצחת היא הסגולים והיא נמצאת תחת
   EX3_data/single_estimator/the_winner.csv
   b. התפלגות הבוחרים נמצאת תחת
   EX3_data/single_estimator/predicted_distribution.txt
   והפורמט שהשתמשנו בו בקובץ הוא כמו בפיאצה:
   <Color>: <% of the voters in the test data>
   c. סיווג הבוחרים נמצא בקובץ
   EX3_data/single_estimator/most_likely_to_vote.csv
   והפורמט שהשתמשנו בו בקובץ הוא כמו בפיאצה:
   <Color>: <List of row indexes of people that voted for Color >

כאשר האינדקס המודפס הינו אינדקס המצביע במאגר המקורי.

בחלק זה החלטנו שהתפלגות הבוחרים שווה בדיוק לרשימת האנשים הסבירים ביותר להצביע עבור כל מפלגה, כלומר כל מפלגה תבחר את האנשים שעבורם היא תארגן הסעות לפי האנשים שהמודל חזה שיצביעו לה.

d. מטריצת הבלבול:

```
[[ 62   0   5   0   0   0   0   0   0   0  25]
 [  0  70   0   0   0   0   1   0   0   0   0]
 [  2   0  50   0   0   0   0   0   0   0   0]
 [  0   1   0  78   0   0  13   0   0   0   0]
 [  0   0   0   0 240   0   0   3   0   0   0]
 [  0   0   0   0   6  26   0  10   2   0   0]
 [  0   5   0   6   0   0 128   0   0   0   0]
 [  1   0   0   0  14   1   0 375   7   0   0]
 [  0   0   0   0  14   0   0   5 115   0   0]
 [  0   0   0   0   0   0   0   0   1 162   0]
 [  3   0   6   0   0   0   0   1   0   0  62]]
```

דיוק ממוצע:

```
0.912
```

כפי שניתן לראות מהמטריצה, הסגולים קיבלו את רוב הקולות. בנוסף, הדיוק הממוצע הוא יחסית גבוה ולכן הבחירה במדד Accuracy היא מוצדקת.

- בחלק הבא נשתמש בעץ החלטה על מנת לראות ויזואלית את המסלולים בעץ ולבחור את התכונות שעל ידי מניפולציה עליהם, נוכל לשנות את תוצאות הבחירות.
-

<u>חלק שני – Non-Mandatory Assignments</u>

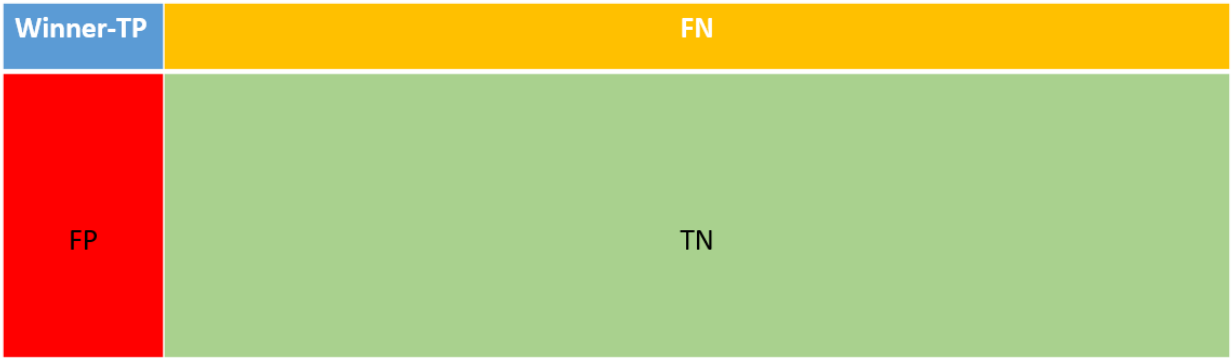
<u>שלב ראשון -</u> אוטומציית בחירת המודל נמצא כולו בקובץ modeling.py והשתמשנו בו גם בחלק הראשון. פירוט השלבים במהלך הריצה:

- במתודה allocate_rand_search_classifiers מקצים את כל המסווגים שבהם רוצים להשתמש (במקרה שלנו יער ועץ החלטה) ועבור כל אחד מהם מאתחלים אובייקט מסוג RandomizedSearchCV על מנת לבצע חיפוש של היפר-פרמטרים לפי grid שמוגדר בקובץ Consts.py ופונקציית scoring שאותה רוצים למקסם המתקבלת כפרמטר.
- מבצעים חיפוש של היפר-פרמטרים במתודה parameter_search_classifiers, עבור כל מודל נשמרים הפרמטרים הטובים ביותר שנמצאו ולאחר מכן בוחרים את המודל הכי טוב על ידי חיזוי על קב' הולידציה במתודה best_trained_model_by_validation.
- מדפיסים את הגרף של עקומת הלמידה של המודל כתלות בגודל קב' האימון.
- קוראים למתודה predict_the_winner שכותבת לקובץ את המפלגה המנצחת על פי מסווג שאומן למשימה זאת.
- קוראים למתודה predict_voters_distribution שכותבת לקובץ את התפלגות הבוחרים על פי מסווג שאומן למשימה זאת.
- קוראים למתודה print_test_confusion_matrix_and_test_error שמדפיסה את מטריצת הבלבול ואת השגיאה הממוצעת על פי מסווג שאומן לקבוצה זאת.

על מנת להוסיף מסווג חדש למודל, צריך להוסיף לפונקציה allocate_rand_search_classifiers את המסווג והאובייקט המתאים של RandomizedSearchCV ובנוסף להוסיף לקובץ Consts.py את ה-grid של ההיפר-פרמטרים שעליהם רוצים לבצע את החיפוש, לא נדרש כל שינוי נוסף. שלב בחירת המודל נעשה כולו בצורה אוטומטית.

<u>שלב שני-</u> נתבקשנו לפתור כל בעיית סיווג באמצעות מודל ייעודי לבעיה. מאחר ואנו מאמין מספר מודלים ובוחרים את הרלוונטי ביותר מתוכו, ראינו לנכון לא לבחור מראש את סוג המסווג (עץ או יער וכדומה), אלא להתאים מטריקה לכל בעיית סיווג.

- למפלגה המנצחת יש רוב קולות במדגם בעל התפלגות רלוונטית (מדגם שאיננו מוטה). מאחר וקל להטות מסווג לבחירת הסיווג בעל הרוב בקבוצת האימון, נרצה לתת ציון למסווג על פי סיווגים אמתיים (TP) תוך התייחסות לסיווגים שהוטו(FP) כלומר נשתמש במדד $precision = \frac{TP}{TP+FP}$.
בכדי לחשב את המשוואה הנ"ל נגדיר, בהינתן כי המנצח הוא i ומטריצת הבלבול היא mat (באיור מטה ניתן לראות את החלוקה במידה ו - $i = 0$).
1. $TP = mat[i][i]$.
2. $TPFP = (sum\ of\ column\ i)$. כלומר כל מי שנובא כי יצביע למנצח אך בפעול לא הצביע למנצח.
3. $FN = (sum\ of\ row\ i) - TP$. כלומר כל מי שנובא כי לא יצביע למנצח אך בפועל הצביע למנצח.
4. TN – סכום על ההצבעות פחות הערכים לעיל. עבור המנצח אין הבדל אם קיים בלבול בין הצבעה כלשהי כל עוד הבלבול אינו קשור למנצח (נגרע או נוסף) ולא נקבע מנצח חדש.



- בכדי לאמן מסווג הקובע את ההתפלגות המדויקת ביותר בין המפלגות, לא נצטרך להבחין בין מצביעים אלא בין ההתפלגות הסופית שהמסווג מוציא. לכן, במקרה זה בחרנו להשוות את סכומי העמודות עם סכומי השורות, ציון כל שורה\עמודה:

$$f(i) = \frac{\min(sum\ row\ i, sum\ column\ i)}{\max(sum\ row\ i, sum\ column\ i)}$$

במידה ואחד מהערכים הנ"ל שונה מ-0 (במדגם רלוונטי), פונקציה זאת חסומה בקטע $[0,1]$.
בנוסף היא מקבל את המקסימום, 1, כאשר 2 הערכים זהים, כלומר למפלגה סווגו כמות זהה של מצביעים ביחס לערכי האמת.
ציון מסווג יחשב כממוצע הפונקציה הנ"ל על פני כל השורות:

$$score(confusion\ matrix) = \frac{1}{amount\ of\ rows} \cdot \sum_{i=0}^{amount\ of\ rows} f(i)$$

במשוואה המתוארת, טעות בתא i,j תשפיע על חישובי 2 המפלגות, לאחת נגרעת הצבעות ולשנייה מתווספות הצבעות. ניתן לחשוב כי ערך טעות זאת מחושב בציון הכולל פעמיים. מנגד ראינו את היתרונות הבאים:

1. השפעת זאת לכל מפלגה אינה בהכרח שווה, לדוגמה טעות עבור מלגה בעלת מספר רב של קולות תשפיע מעט, בעוד טעות עבור מפלגה בעל קול יחיד, תאפס את הציון.
2. סכמת ההשפעה בשני המקרים תניב השפעה גדולה יותר על תוצאת המסווג.
3. לבסוף, אנו רוצים לתת לכל מפלגה השפעה זהה על תוצאת המסווג ולכן אנו ממצעים את כלל הנקודות.

- בכדי לאמן מסווג הקובע לכל אדם את המפלגה עבורה הוא יצביע, תוך שמירה על זהותו ואחוז דיוק גבוהה ככל הניתן בחרנו להשתמש בשיטת הדיוק המוכרת $accuracy$. אומנם ישנה מפלגה בעלת רוב קולות, אך אחוז הקולות הנ"ל איננו עולה על 50%. בכך אנו מבטיחים כי המסווג לא יהיה מוטה ויתחשב בערכים אלו.
  עבור המטלה השלישית אנו דורשים מהמסווג להיות המדויק ביותר. עליו לתת חיזוי נכון ככל האפשר בכל חיזוי. מנגד, במשימות הראשונות אנו דורשים מהמסווגים לתת חיזויים בעלי התפלגות כזאת המתאימה לקבוצת האימון.

מתוך כך, נצפה כי הערך המוסף במסווגים הראשונים הינו הפשטות שלהם. בעוד עבור המסווג האחרון נצפה לאחוז דיוק גבוהה עבור קבוצת סיווגים קטנה.

<u>בשלב השלישי</u> נדרשנו לזהות את התכונות שעל ידי מניפולציה עליהם ניתן לשנות את המפלגה המנצחת. בחרנו להשתמש בעץ סיווג על מנת לזהות את התכונות הללו, כלומר הדפסנו את עץ הסיווג וראינו את התכונות שעל ידי שינוי שלהם ניתן לשנות את המפלגה המנצחת.

<u>הערה:</u> בציור אין שמות של תכונות, אלא מספרים. מילון עבור מספר-תכונה:

0 – AVG_satisfaction_previous_vote, 1- Garden_sqr_meter_per_person_in_residancy_area,

2- Yearly_IncomeK, 3- Weighted_education_rank, 4- Number_of_valued_Kneset_members,

5- Overall_happiness_score, 6- Will_vote_only_large_party_int,

7-Most_Important_Issue_Environment, 8- Most_Important_Issue_Social,

9- Most_Important_Issue_Education, 10- Most_Important_Issue_Healthcare,

11 - Most_Important_Issue_Financial, 12- Most_Important_Issue_Foreign_Affairs,

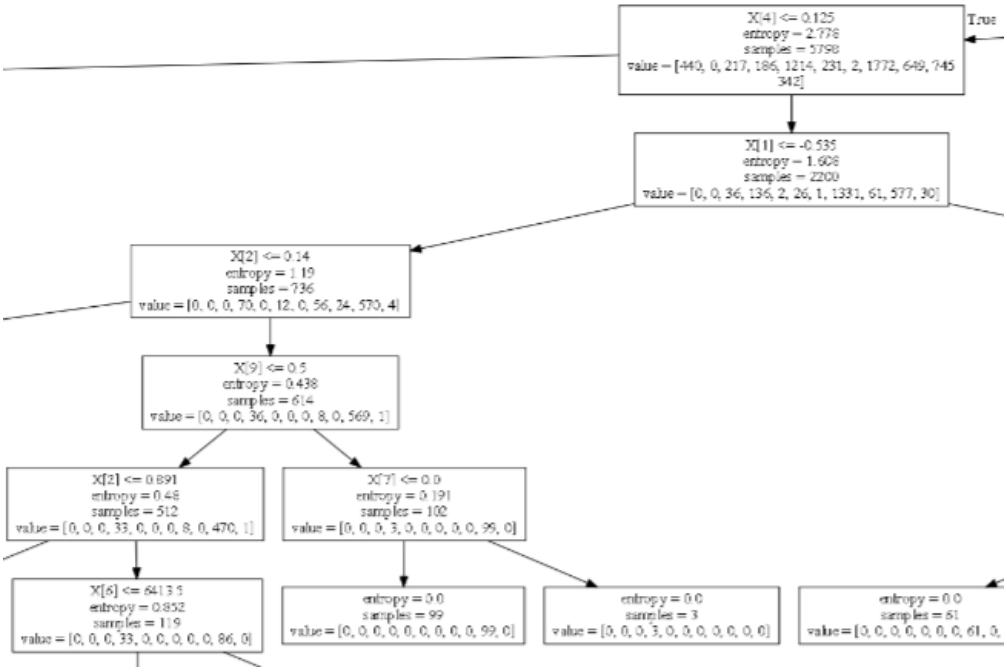
13- Most_Important_Issue_Other, 14- Most_Important_Issue_Military

מבנה מערך ה-Values:

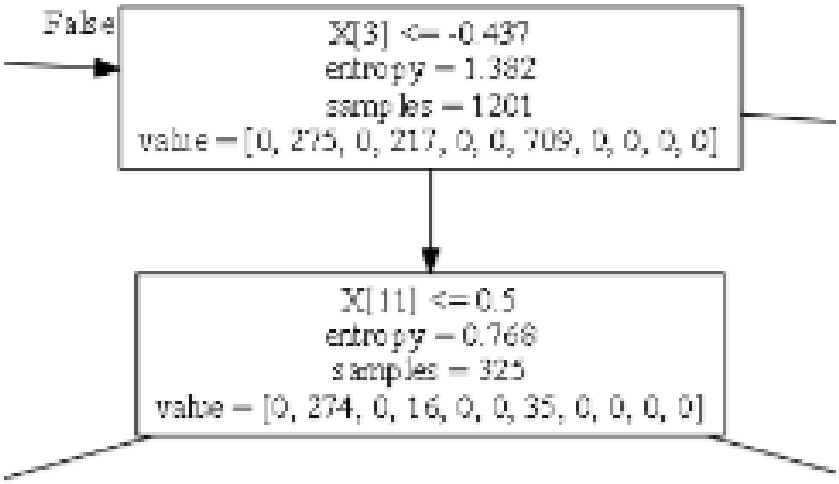
[Greys, Turquoise, Reds, Yellows, Browns, Whites, Blues, Purples, Pinks, Greens, Oranges]

העץ יצא רחב מאוד ולכן התמונות הם חלקיות, צירפנו את הקובץ שמכיל את כל העץ סיווג - 30_iter.png להגשה.

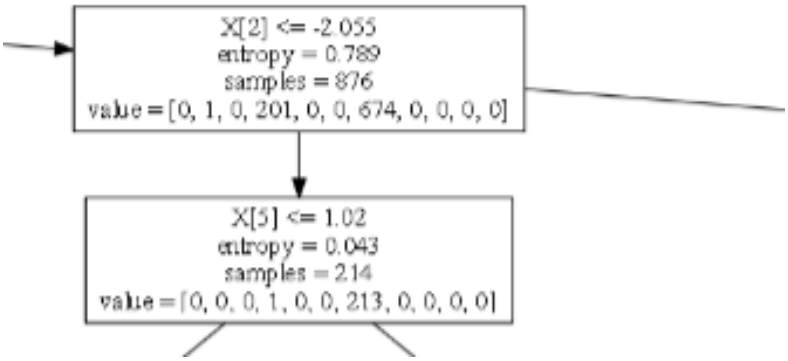
מקרה 1: נעלה את AVG_satisfaction_with_previous_vote, נוריד את Weighted_education_rank
רוב הדוגמאות הם של טורקיז (274) ורוב העלים שיוצאים מהצומת התחתון בתמונה הם של טורקיז ולכן טורקיז ינצח.



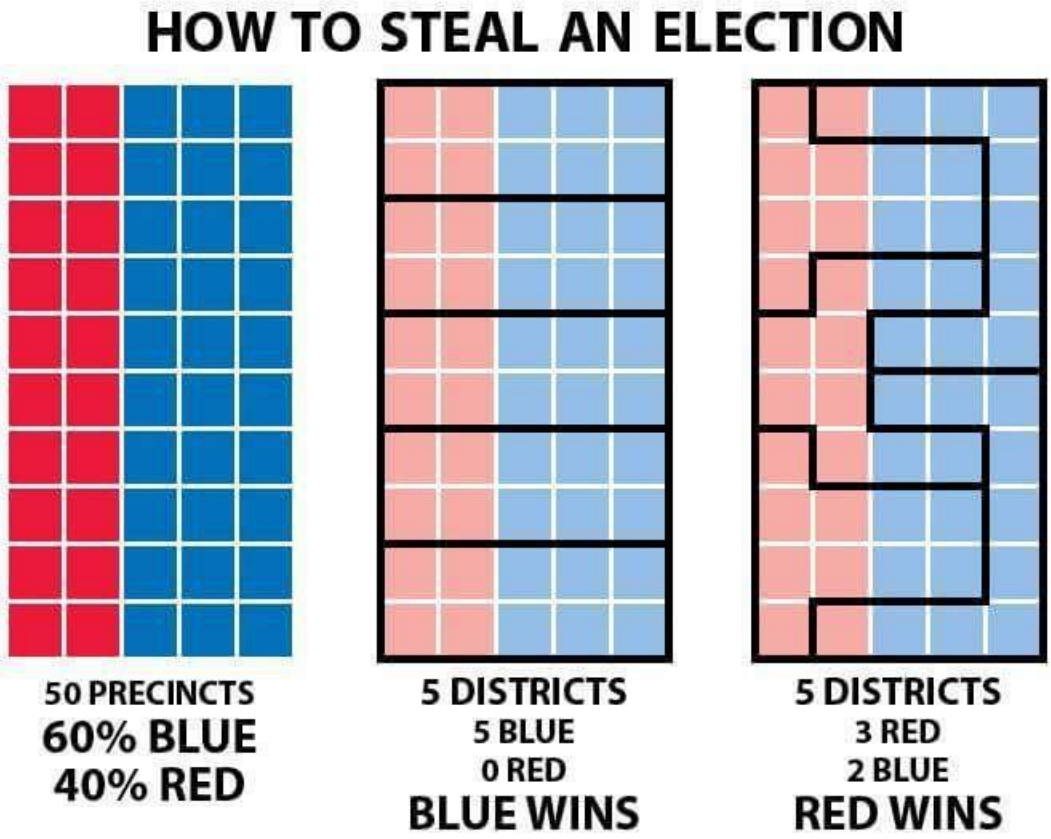
מקרה 2: נוריד את AVG_satisfaction_with_previous_vote, נעלה את Weighted_education_rank,
נוריד את Garden_sqr_meter_per_person_in_residancy_area, נעלה את
Garden_sqr_meter_per_person_in_residancy_area, נעלה את Yearly_IncomeK, נבחר
Most_Important_Issue_Environmen מגיעים לעלה טהור שבו יש 99 דוגמאות של ירוק ולכן ירוק ינצח.



מקרה 3: נעלה את AVG_satisfaction_with_previous_vote, נעלה את Weighted_education_rank, נוריד את Garden_sqr_meter_per_person_in_residancy_area, נוריד את Yearly_IncomeK רוב הדוגמאות בצומת התחתון הם של כחול ולכן כחול ינצח.



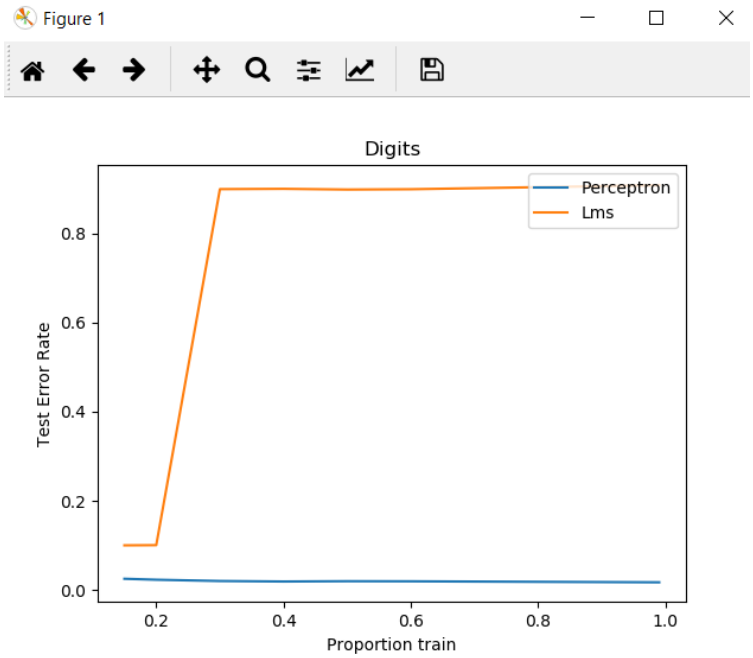
השתמשנו בסעיף זה בעץ החלטה מכיוון וניתן להבין ממנו בצורה ויזואלית את התכונות וההשפעה שלהם על הבוחרים. חיפשנו מסלולים שהם יחסית קצרים, שמסתיימים בצומת עם רוב מוחלט של דוגמאות מזוג מסויים ווידאנו שהעלים שמתקבלים מצומת זה מסווגים לפי המפלגה עם הרוב.



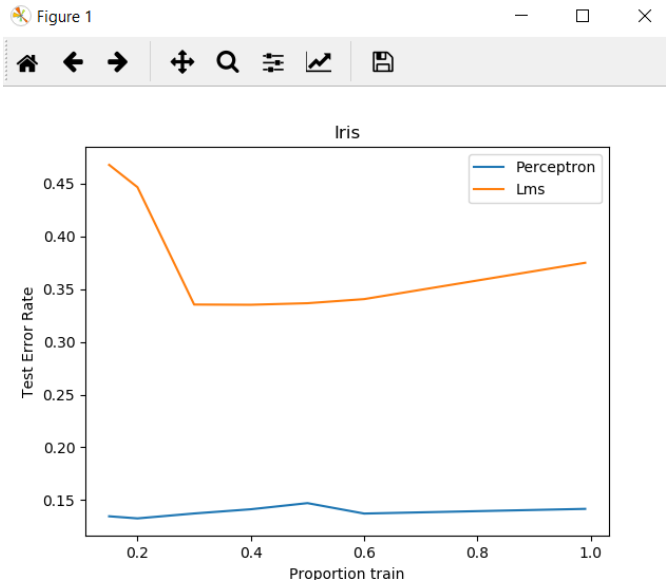
חלק שלישי – Bonus for Pairs

מימשנו את אלגוריתם LMS בקובץ LMS.py ואת החלק שמשווה בין המסווגים בקובץ CompareModels.py.

a. הפרמטרים שבחרנו עבור Perceptron – num_of_iter=100, eta=0.001
הפרמטרים שבחרנו עבור LMS - num_of_iter=100, eta=0.001
התוצאות עבור Digits:

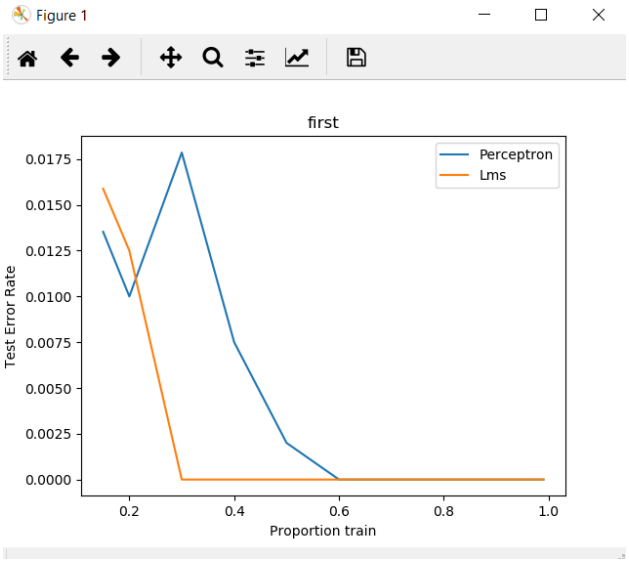


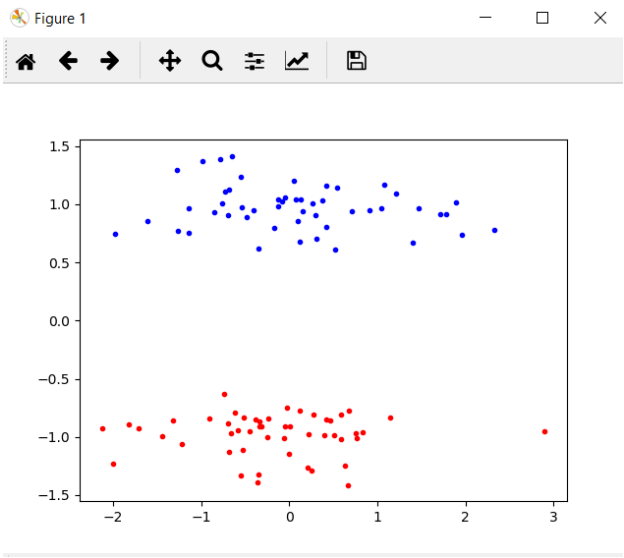
התוצאות עבור Iris:



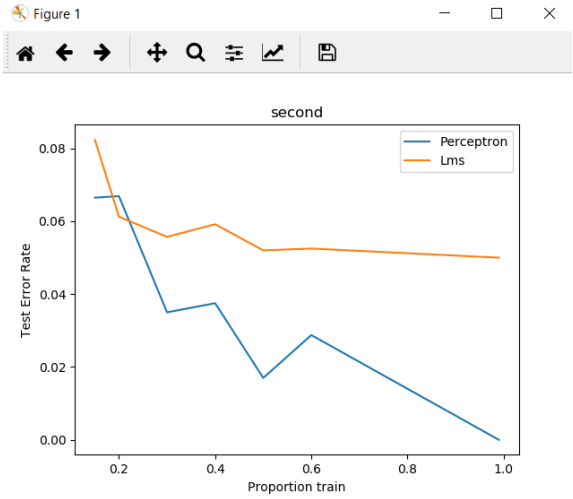
בשני הדטהסטס האלגוריתם של Perceptron התכנס מהר יותר מ-LMS.

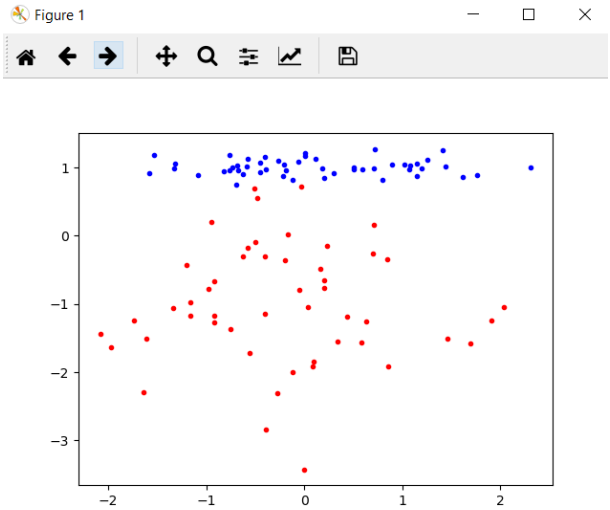
b. דטהסט שעבורו LMS מתכנס מהר יותר:





דטהסט שעבורו Perceptron מתכנס מהר יותר:





בשני המקרים השתמשנו במתודה createLinearySeperableDataset שנמצאת בקובץ CompareModels.py על מנת ליצור דטהסטס רנדומליים שיש בהם הפרדה לינארית.

ראינו שבמקרים שבהם יש הפרדה גדולה בין הסיווגים (אין דוגמאות עם סיווגים שונים במרחק קרוב) אלגוריתם LMS מתכנס מהר יותר. עבור מקרים שבהם ההפרדה היא לא גדולה וקיימות דומאות קרובות עם סיווגים שונים,אלגוריתם Perceptron מתכנס מהר יותר.

c. נעזרנו במימוש של "Comparing various online solvers" כשמימשנו את CompareModels.py כאשר התאמנו אותו כך שיעבוד עם 1-vs-all. התוצאות עבור כל דטהסט מתוארות בגרף כשא ציר ה-x מייצג את אחוז הדאטה ששימש לקבוצת האימון וציר ה-y מייצג את אחוז השגיאה על קב' המבחן. ניתן להריץ את CompareModels.py על מנת לקבל את התוצאות והגרפים שצירפנו לדו"ח.