למידת מכונה מטלה 3

מגישים: רוני הר-טוב 207199282, ידידיה אבן-חן 207404997

הצהרת חלוקת עבודה: עבדנו ביחד על התהליך של כתיבת הקוד וה $debugging$. לאחר שהשלמנו את הקוד הבסיסי רצינו להרחיב ולבדוק מקרים נוספים, התעמקנו בתוצאות וחשבנו ביחד על הניתוח, ההסברים, והמסקנות. כל תהליך הניתוח והמסקנות נעשה במשותף, חלקנו רעיונות והתייעצנו וכתבנו אותן. במהלך העבודה ביצענו שעת קבלה משותפת עם המרצה.

שאלה 1 – K-NN

קובץ קוד (knn.py) וגרפים של התוצאות נמצאים בתיקייה question_1.

להלן דוגמת הרצה (ממוצעים של 100 הרצות):

p	k	avg emp	avg true	avg diff
1	1	0.0148	0.1282	0.1134
1	3	0.0549	0.0972	0.0422
1	5	0.0623	0.0892	0.0269
1	7	0.0648	0.0854	0.0205
1	9	0.0668	0.0876	0.0208
2	1	0.0148	0.1251	0.1103
2	3	0.0516	0.0977	0.0461
2	5	0.0603	0.0869	0.0266
2	7	0.0620	0.0832	0.0213
2	9	0.0635	0.0824	0.0189
inf	1	0.0148	0.1338	0.1190
inf	3	0.0538	0.0932	0.0395
inf	5	0.0610	0.0893	0.0283
inf	7	0.0646	0.0867	0.0221
inf	9	0.0643	0.0899	0.0256

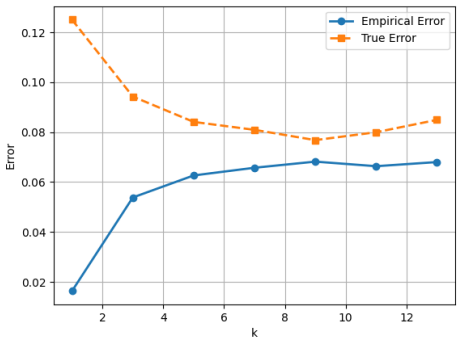
```
min true error: 0.0824 with p=2, k=9
```

ניתוח מילולי של התוצאות:

ניתוח של k:

הדוגמה הכי ברורה ל-$overfitting$ היא עבור $k = 1$, בכל הערכים של p. זה לא מפתיע. בקבוצת ה-$train$, כל נקודה היא הכי קרובה לעצמה, ולכן התיוג שלה יהיה נכון – למעט מקרים ספציפיים של שתי נקודות עם אותם ערכי x, y ותיוג שונה, שבהן הבחירה תהיה אקראית. אבל יש מעט מקרים כאלה, ובסך הכל הטעות האמפירית היא כמעט 0. מנגד, הטעות האמיתית גבוהה יחסית – מעל 10%. הנקודות הבעייתיות הן אלה בחלק ה"מעורבב" בין הקבוצות, או בחלק שקרוב לקבוצה השנייה.

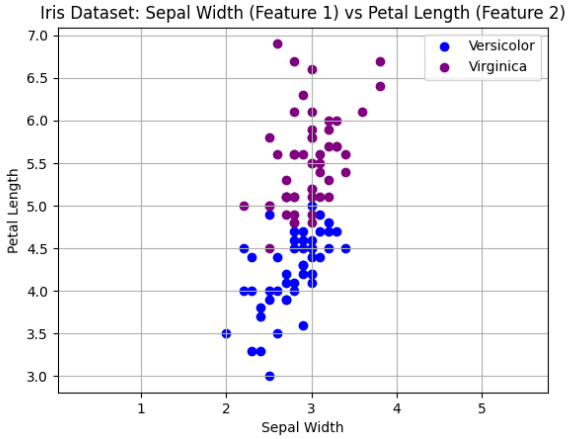
כצפוי, עבור ערכי k גדולים יותר, הטעות האמפירית עולה (כי לוקחים בחשבון יותר נקודות, והנקודה עצמה פחות משפיעה. אז הנקודות בחלק המעורבב, אם במקרה יש יותר נקודות מהצבע השני לידן, הן יקבלו תיוג הפוך) והטעות האמיתית יורדת (כי הנקודות שהיו בעייתיות, יש יותר סיכוי שנתחשב במספיק נקודות מהצבע הנכון). עד גבול מסוים (ה k האופטימלי), שאחריו הטעות האמיתית עולה שוב. לדוגמה, עבור $p = 2$, עם ערכי k שונים:



כפי שאמרנו, $k = 9$ הוא אופטימלי עבור המדגם שלנו. לא בגלל שהוא הכי גדול – אם ניקח ערכים גדולים יותר של k, הטעות תגדל. להשערתנו, הסיבה היא הפיזור הספציפי של הנקודות במדגם שלנו. אין נוסחה ל-k אופטימלי עבור מדגם כללי.

ניתוח של p:

ניזכר במדגם שלנו:



ניתן לראות, שהפיזור העיקרי הוא לאורך ציר ה-y. וגם, שאפשר להפריד (אינטואיטיבית) בין הקבוצות על סמך ערך y בלבד. אם "נדחף" את כל הנקודות לציר ה-y, רוב המידע יישמר.

עבור $p = 1$, "מרחק מנהטן", בודקים את סכום המרחקים בציר ה-x ו-y. עבור $p = 2$, בודקים את האורך של הקו המחבר בין הנקודות. כאשר $p = \infty$, המרחק מחושב רק לפי הקורדינטה שיש בה את ההבדל המקסימלי.

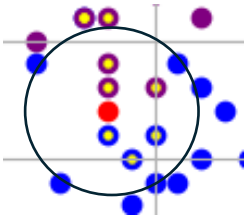
בגלל שהפיזור העיקרי הוא לאורך ציר ה-y, המרחק בין שתי נקודות מושפע בעיקר מהקורדינטה y. גם עבור $p = 1$, המרחק בציר x קטן יחסית, וגם עבור $p = 2$, המרחק בציר y משמעותי יותר מהמרחק בציר x.

ולכן, במדגם הזה אין משמעות רבה לערך הספציפי של p. ואכן בתוצאות שלנו, ההבדלים בין התוצאות היו קטנים מאד עבור ערכי p שונים (הבדל של 0.005 בערך).

ובכל זאת, יש עדיפות ל $p = 2$.

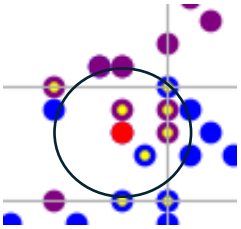
הסבר אינטואיטיבי לתוצאות:

ב $p = 1$, בגלל שמרחק מנהטן הוא לא ייצוג מדויק של המרחקים בין הנקודות, יש חסרון כלשהו. לדוגמה:



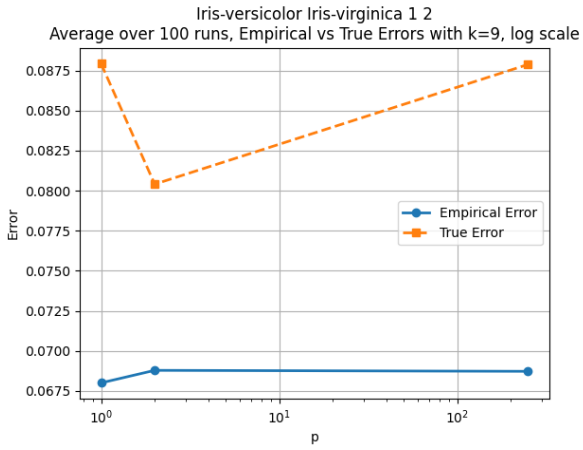
הנקודה האדומה היא נקודה כחולה שקיבלה תיוג סגול. הנקודות עם הצהוב הם השכנים הקרובים. אפשר לראות שארבעת הנקודות שעל המעגל, יותר קרובות לנקודה האדומה מאשר שתי הנקודות שלמעלה. אבל בגלל שאנחנו משתמשים כאן ב $p = 1$, הן נחשבות רחוקות יותר. אם היינו משתמשים ב $p = 2$, הנקודה הייתה מקבלת תיוג נכון.

ב $p = \infty$, למרות שרוב המידע נשמר, יש מידע שהולך לאיבוד (כי בכל נקודה, מתייחסים רק לאחד הפרמטרים). לדוגמה:



גם פה אם היינו משתמשים ב $p = 2$, היינו מקבלים יותר נקודות סגולות, והתיוג היה נכון.

ואכן, אפשר לראות בגרף הבא: ($p = 250$ מייצג את $p = \infty$)



האופטימלי הוא $p = 2$, אבל בפער מאוד קטן.

ניסויים נוספים:

מתוך סקרנות וכדי להבין יותר לעומק את משמעות הערכים השונים, הרצנו את הבדיקות עם ערכי k נוספים, עד $k = 49$. וראינו שאחרי $k = 11$ בערך, גם הטעות האמפירית וגם הטעות האמיתית עולות יחד. ההערכה שלנו היא, שעבור ערכים גדולים מדי של k, כל תיוג נעשה לפי יותר מדי נקודות, וכל קבוצה משפיעה יותר על השנייה. אם נחשוב על המקרה הקיצוני של $k = 49$, כבר מתחשבים בכל הנקודות חוץ מאחת, ואם במקרה הנקודה הזו היא מאותו צבע – אז התיוג היה שגוי. ואכן עבור $k = 49$, הטעות האמפירית קרובה ל-40%, והטעות האמיתית קרובה ל-50%.

בדקנו גם עבור קבוצות שונות – לדוגמה את $setosa$ עם $virginica$, ועם מאפיינים שונים (לדוגמה המאפיינים הראשון והשני במקום השני והשלישי). וראינו שעבור קבוצות שונות, הערכים האופטימליים יכולים להשתנות – אפילו עד $k = 30$. אבל המגמה הכללית זהה: עד ערך ספציפי של k, הטעות האמפירית עולה והטעות האמיתית יורדת בחדות, עד שהן כמעט נפגשות. ואחרי הערך הזה, שתי הטעויות עולות שוב, בשיפוע שהולך וגדל. ולקראת $k = 50$ העלייה הכי קיצונית.

כדי להגיע לתוצאות כמה שיותר אמינות, בנינו את המבחן הבא: ביצענו 100 הרצות של האלגוריתם (כלומר 100 הרצות שבכל אחת לוקחים את הערכים האופטימלים אחרי ממוצע של 100 פעמים KNN – סה"כ 10,000 פעמים KNN). וראינו שה-k האופטימלי יוצא (בערך): חצי מהפעמים 9, שליש מהפעמים 7, ושאר הפעמים 5:

```
k count: {5: 15, 7: 34, 9: 51}
```

כלומר, הערך שיוצא אופטימלי הכי הרבה פעמים הוא 9. ולכן נבחר אותו.

באותו מבחן בדקנו גם את p:

```
p count: {1: 15, 2: 71, inf: 14}
```

ראינו, שברוב הפעמים יוצא שהאופטימלי הוא $p = 2$, אבל המסקנה הזו פחות חד משמעית מאשר המסקנה לגבי ה-k.

שאלה 2 – *Decision-Tree*

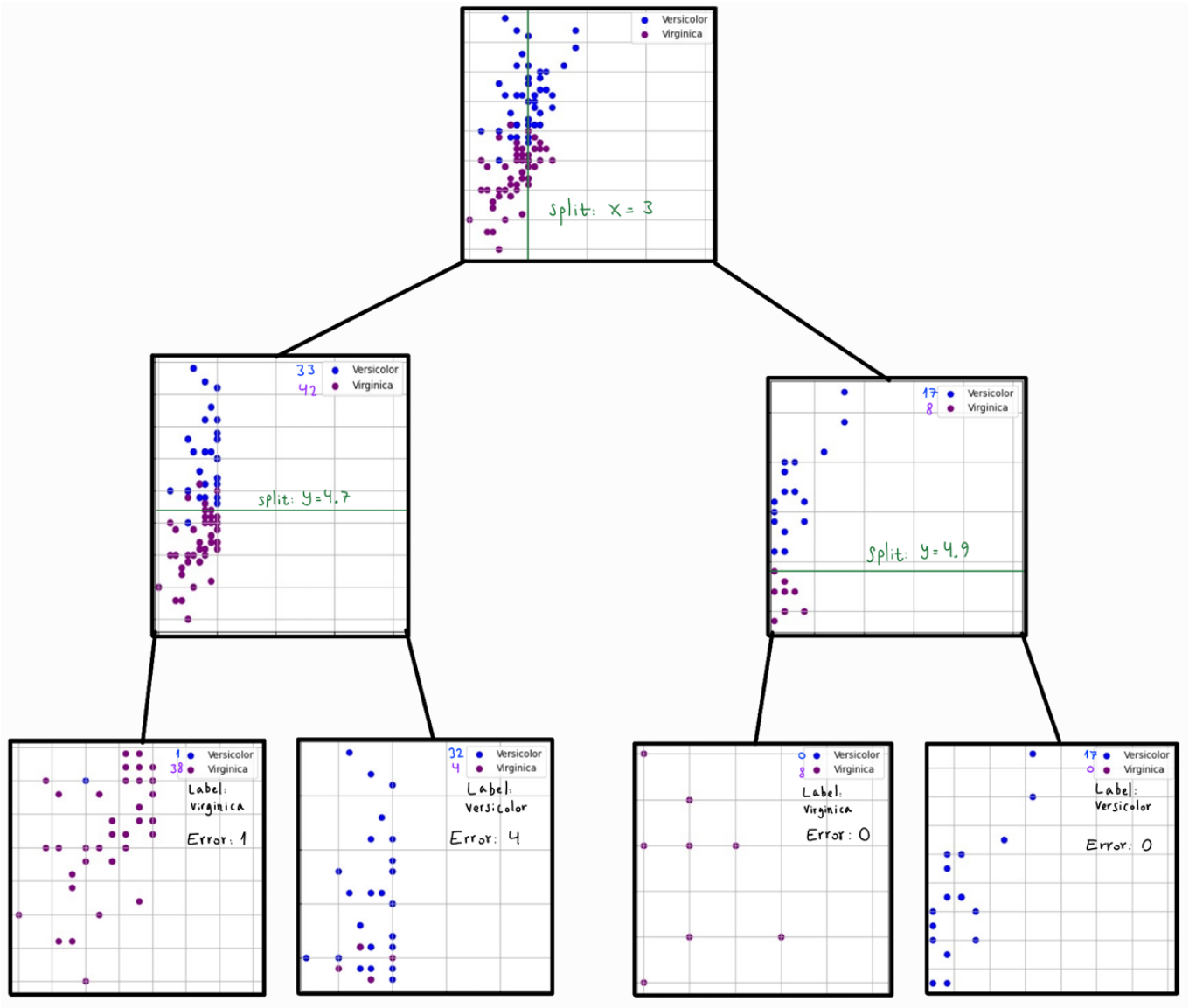
קובץ קוד (decision_tree.py) ותמונות של התוצאות נמצאים בתיקייה question_2.

סעיף א - *brute-force*:

להלן הפלט של הקוד:

```
best tree with error = 5 found with ((9, 46, 48)) at try 40367/274560
Node(split_feature=0, split_value=3.0, points=100)
    Node(split_feature=1, split_value=4.7, points=75)
        Leaf(label=1, points=39, error=1)
        Leaf(label=0, points=36, error=4)
    Node(split_feature=1, split_value=4.9, points=25)
        Leaf(label=1, points=8, error=0)
        Leaf(label=0, points=17, error=0)
```

העץ האופטימלי הוא:



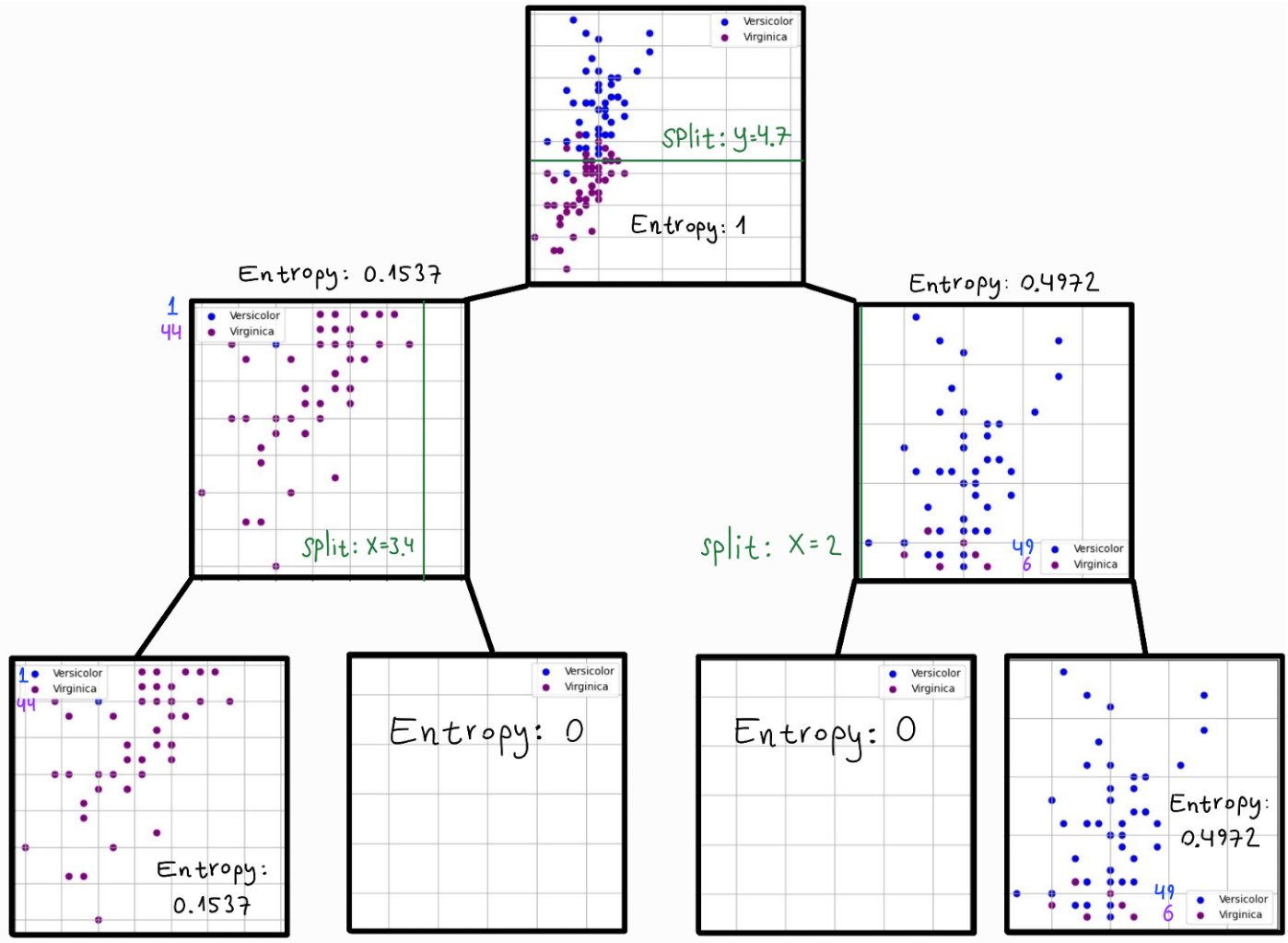
סה"כ טעות: 5%.

סעיף ב – *Binary entropy*:

להלן הפלט של הקוד:

```
best tree with error: 7
Node(split_feature=1, split_value=4.7, points=100)
    Node(split_feature=0, split_value=3.4, points=45)
        Leaf(label=1, points=45, error=1)
        Leaf(label=1, points=None)
    Node(split_feature=0, split_value=2.0, points=55)
        Leaf(label=1, points=None)
        Leaf(label=0, points=55, error=6)
```

העץ האופטימלי הוא:



סה"כ טעות: 7%.

בפיצול הראשון, ניתן לראות שהיה אפשר להגדיל קצת את הערך שלפיו מפצלים, ו"להרוויח" עוד 2 נקודות סגולות במחיר של רק נקודה אחת כחולה. הטעות הכללית הייתה יורדת. אבל האנטרופיה הייתה עולה (כי הוספת כחול אחד הרבה יותר משמעותית מהוספת 2 סגולים). כלומר, אנטרופיה היא קירוב טוב לאופטימלי, אך לא מושלם. בנוסף, הפיצולים הם כך שיש שני עלים ריקים. אפשר למצוא פיצול טוב יותר אם ננצל את כל העלים, אבל עלה ריק נותן אנטרופיה אפס, ומאוד הגיוני שאלגוריתם ששואף למזער אנטרופיה "ירצה" לייצר עלים ריקים.

נתמקד בהבדלים בין שיטה א ($brute\ force$) לשיטה ב ($entropy$): ניתן לראות שבסעיף א הפיצול הראשון הוא לפי ערך x, וזה הפיצול שנותן בסוף את הטעות המינימלית. בסעיף ב, הפיצול הראשון הוא לפי ערך y, כי זה מאפשר אנטרופיה נמוכה. כל פיצול לפי ערך x (בשלב הראשון) ייתן אנטרופיה גבוהה יחסית. והפיצול הראשון משפיע על שאר הפיצולים ועל התוצאה הסופית.

עם זאת, הביצועים של השיטה השנייה מרשימים, במיוחד לאור השיפור המשמעותי בזמן ריצה: עבור n נקודות, $O(n)$ במקום $O(n^3)$ במקרה שלנו של 3 רמות. או במקרה הכללי, עבור h רמות:

יש $2^{h-1} - 1$ קודקודים פנימיים, כלומר זמן ריצה $O(2^{h-1} \cdot n)$, לעומת $O\left(n^{2^{h-1}}\right)$.

קל לראות שהגישה של $brute\text{-}force$ לא ישימה עבור מדגמים גדולים.