

מטלה 2

אילן מאיר סופיר : 342615648

שי משה : 318160108

המטלה נעשתה ביחד והתרומה שווה

Problem 1.


- What is the VC-dimension of the infinite set of uni-directional balls on three dimensional points? Prove your result.
- Hypothesis class \mathcal{C} contains both the infinite set of uni-directional balls and the infinite set of hyperplanes on three dimensional points. What is its VC-dimension? Prove your result.
- Hypothesis class \mathcal{D} contains the infinite set of half-balls on three dimension points. Give an upper bound on its VC-dimension.
- How many different labels can \mathcal{D} give to 100 points? Give an upper bound.

עלמה 1:

א) נניח שה-VC dimension של כדור 3D הוא $\frac{4}{2}$.
נניח שיש לנו קבוצה של נקודות S הניתנת ל-shattering על ידי כדור כזה.
יהי \vec{p} בנקודה כלשהי המציינת סיווג אדום (1) או כחול (-1) לכל נקודות בקבוצה S .
מאחר ו- S ניתנת ל-shattering על ידי כדור 3D, אזי יש לנו כדור שמכיל את כל הנקודות שצבען
אדום או כחול על ידי \vec{p} . נגדיר \vec{p}' להיות בנקודה שמסווגת את הנקודות הפוך מ- \vec{p} (כלומר
אם \vec{p} מסדרה נקודה להיות אדומה, \vec{p}' מסדרה אותה להיות כחולה). אז עבור \vec{p}' נוסף לייצר כדור 3D
שיכיל את כל הנקודות האדומות.

נקח את שני הכדורים שיצרנו, ונמצא את החיתוך שלהם שהוא למעשה מעגל 2D. מכאן נוכל להסיק
מייד כי כל נקודה במעגל, תהיה גם במישור ולכן קיבלנו שהקבוצה S ניתנת ל-shattering על ידי
מישור. זה הכיוון הנכון, אבל קבוצה S ניתנת ל-shattering על ידי מישור, אז נוסף לייצר כדור 3D
גדול מאחר של המישור.

לכן ה-VC dimension של מישור 2D שזהו מעגל של כדור 3D.
מאחר ומישור יש 2 מימדים חופפים, המימד של מישור הוא 2. האינדיקס הכיתה שמימד ה-VC
של מישור בעל $d-1$ מימדים הוא $d+1$.

מכאן שמימד ה-VC dimension של מישור 2D ושל כדור 3D הוא 4.  \square d.e.n

(b)

כדי לדעת את ה VC-dimension של קבוצת C נקיר את היכולת של כל רכיב לעצב נקודות ואז ננסה על השלמה שלהם.

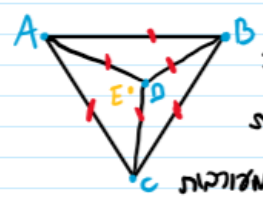
① כמו שהיו בסעיף (a), ה VC-dimension של כדור 3D הוא 4.

② לפי משפט שראינו הכיתה: "The set of $d-1$ dimensional hyperplanes has VC-dimension $d+1$ on \mathbb{R}^d ".
אז במקרה שלנו $d=3$ כי אנחנו ב 3D, כל' ה VC-dimension של קבוצה מ'נספות של מישור ב 3D הוא 4.

עכשיו ננסה שלמה של פיהם:

שני הקבוצות ניתנות ל-shattering של 4 נקודות.

אבל האם בן 5 נקודות? הנוסחה היא 5! על מנת להוכיח שאי אפשר ה VC הינו הידיוק 4, צריך להוכיח שכל קבוצה של יותר נקודות (25=5!) לא ניתן לבצע shattering למסביב רק עבור כל קבוצה מסוג 5 נקודות:



יהי 5 נקודות ב-3D כגון 4 נקודות יוצרים קובקדים של הקטל הזה: והנקודה החמישית יהיה מחוץ למחצית (centroid) של הקטל הזה, מאתגר את ההפרדה לניאורית ולא לניאורית באחד. הקוים מתחלקים מהפרדת שני קבוצות המעורבות.

\mathcal{H} עם קובקדים לא סמוכים של ה-shape, אם לא מישור ולא כדור חד-כיווני יכולים לעמוד כמבויק נפרד-קבוצה בן 5 לא מצליחה. קונפיגורציה זו מבטיחה את המבטאות האחריות של יכולת shattering של קבוצה C . מה שמביא שאפילו עם הכלים המשופרים של קיימים הסדרים (קובעיים שהם) לא יכול לעמוד בצורה מושלמת, שם צריך על אילוצים מדויקים של מידה ה-VC של כל ה מידה האל של הקבוצה C היא 4.

(c)

ההרצאה, ראוי שיהיה S חוקים מתוך קבוצה H בעלת מידה VC של d הוא בדקמן:

$$VC(H') \leq 2ds \log_2(3s)$$

כמו שאמרנו הכיתה הישם המעלה 2 שלן, חצי כדור ב-3D זה בעצם חיתוך של כדור ב 3D היה עם מישור.

אז בסעיפים הקודמים הוכחנו ש-VC-dimension של כדור ב-3D ומישור הם שווים 4.

יהי H_1, H_2 קבוצות כדור ב-3D ו- H_2 קבוצת מישור ב-3D.

$H_1, H_2 \subseteq H$. היא קבוצת הונויות עם מידה VC של 4.

$H' = \{H_1 \cup H_2\}$ (כמו שאמרנו הכיתה משתמש בחיתוך)

H' היא קבוצה לחצי כדור ב-3D
: s/c

החם העלון לחיגוק של 2 צורות מקבוצת חוקים בעלת מידה VC 4 הינו:

$$VC(H') \leq 2 \cdot 4 \cdot 2 \log_2(3 \cdot 2)$$

$$\leq 16 \log_2(6)$$

$$VC(H') \leq 41,36$$

(d) כדי לפתור למסדית הלה, נשתמש במשפט Samer שלמקנן הכיתה: המספט אומר שערור חוק H עם מימד ה VC של d, החסם על מספר הסיווגים האופקים שהוא יכול ליצור ערור קטנה S

ומכיל m נקודות הלא:

$$\Pi(H, S) \leq \left(\frac{e \cdot m}{d}\right)^d$$

מימד ה VC של חצי-כדור 3D לבי ההכלה מעבאן הלא 4. (כדור או חצי-כדור ה- 3D על אלו עס) נביה את הערכים הבאים: d=4, m=100 ינקבם:

$$\Pi(H, S) \leq \left(\frac{e \cdot 100}{4}\right)^4 \approx \underline{21\,327\,402}$$

ענה הרבה יותר טוב מאשר סויג שזהו מספר הסיווגים האפשרי המקסימלי.

Problem 2

תוצאות הריצה עבור קו:

הדפסה של של ה errors-מעל 50 איטרציות של adaboost כאשר התוצאות מוצגות משמאל לימין כך שהשגיאה בצד שמאל מסמנת חוק שמורכב מהקו הראשון, לאחר מכן, מוצגת התוצאה של השגיאה עבור חיבור של שני קווים וכן הלאה.

Results for method: line

Average empirical error: [0.25813333 0.26213333 0.172 0.22026667 0.1256 0.1416 0.08346667 0.10586667]

Average true error: [0.35226667 0.35946667 0.31866667 0.34746667 0.3008 0.30293333 0.25386667 0.28213333]

התוצאות מוצגות גם באופן גרפי:



(1) מהתוצאות ניתן לראות שככל שחיברנו יותר חוקים שאותם ה-adaboost, הטעות האמפירית והטעות האמיתית פחתה. כשמגיעים ל 7 קווים, הירידה בטעות נעצרת. זאת אכן התוצאה הרצויה מאחר והדאטא שלנו מורכב ואינו ניתן להפרדה ליניארית פשוטה, לכן היינו מצפים שיהיו מספר קווים שיפרידו בין המעגלים בדאטא. היכולת של Adaboost לשפר את התוצאה היא מוגבלת.

(2) לא קיבלנו overfitting מאחר ויש ירידה בטעות הנסיונית במקביל לירידה בטעות האמיתית.

תוצאות הריצה עבור מעגל :

```
Results for method: circle
Average empirical error: [0.0184      0.02133333 0.00106667 0.0008      0.          0.
 0.          0.          ]
Average true error: [0.0688      0.06293333 0.04746667 0.0504      0.04533333 0.04613333
 0.04613333 0.04266667]
```



(1) מהתוצאות ניתן לראות שהחל מחיבור של שני מעגלים, הטעות המדגמית והטעות האמיתית יורדות בצורה משמעותית. ראויים שבטעות האמיתית החל ב 3 מעגלים אין יותר שגיה.

(2) לא קיבלנו overfitting מאחר ויש ירידה בטעות ככל שמחברים יותר מעגלים. הירידה בטעות של המודל קורית במקביל עבור הטעות המדגמית והטעות האמיתית.

(3) מעגלים יכולים ללמוד בצורה טובה יותר את הדאטא מאשר קווים כי היינו יכולים רק עם עיגול אחד לחלק את הנקודות לפי הצבע של הנקודות בעיגול.

קוד:

Line.py:

```
from typing import Union
import numpy as np
class Line:
    def __init__(self, p1: np.ndarray, p2: np.ndarray, color: {1, -1}):
        self.p1 = p1
        self.p2 = p2
        self.color = color
        self.v = p2 - p1
        self.norm_v = np.linalg.norm(self.v)

    def __distance_from_point(self, point: np.ndarray) -> float:
        # Return the distance from this line to a point
        return np.cross(self.v, point - self.p1) / self.norm_v

    def predict(self, point: np.ndarray) -> Union[int, np.ndarray]: # Return the class of
the point
        distances = self.__distance_from_point(point)
        if self.color == 1:
            return np.where(distances > 0, 1, -1)
        else:
            return np.where(distances < 0, 1, -1)

    def __str__(self):
        return "Line: " + str(self.p1) + " to " + str(self.p2)
```

Circle.py:

```
from typing import Union
import numpy as np
class Circle:
    def __init__(self, center: np.ndarray, outer: np.ndarray, color: {1, -1}):
        self.center = center
        self.outer = outer
        self.radius = np.linalg.norm(center - outer)
        self.color = color

    def __distance_from_point(self, point: np.ndarray) -> np.ndarray:
        # Return the distance from this line to a point
        return np.linalg.norm(point - self.center, axis=1) - self.radius

    def predict(self, point: np.ndarray) -> Union[int, np.ndarray]: # Return the class of
the point
        distances = self.__distance_from_point(point)
        if self.color == 1:
            return np.where(distances > 0, 1, -1)
        else:
            return np.where(distances < 0, 1, -1)

    def __str__(self):
        return "Circle: " + str(self.center) + " radius: " + str(self.radius)
```

AdaBoost.py :

```
import multiprocessing
from typing import Union

import numpy as np

def calculate_error(rule, data: np.ndarray, y: np.ndarray, weights: np.ndarray):
    predictions = rule.predict(data)
    errors = predictions != y # Calculate the errors
    error_sum = np.dot(errors, weights) # Compute the weighted sum of errors
    return error_sum

class AdaBoost:
    def __init__(self, rules: list):
        self.rules = rules
        self.alphas = None
        self.weights = None
        self.best_rule = []
        self.chosen_alphas = []

    def fit(self, data: np.ndarray, y: np.ndarray, iteration: int): # Train the model with
AdaBoost
        # initialize weights
        self.alpha = 0
        self.weights = np.ones(len(data)) / len(data)
        for it in range(iteration):
            err_lst = [np.dot((rule.predict(data) != y), self.weights) for rule in
self.rules]
            argmin_index = np.argmin(err_lst)
            if err_lst[argmin_index] == 0:
                self.alpha = 1
                self.best_rule.append(argmin_index)
                self.chosen_alphas.append(self.alpha)
                continue
            else:
                self.alpha = 0.5 * np.log((1 - err_lst[argmin_index]) /
err_lst[argmin_index])
                # update weights
                self.weights = self.weights * np.exp(-self.alpha * y *
self.rules[argmin_index].predict(data))
                # for i in range(len(self.weights)):
                #     self.weights[i] = self.weights[i]\
                #         * np.exp(-self.alpha * y[i] *
self.rules[argmin_index].predict(data[i]))
                # normalize weights
                self.weights = self.weights / np.sum(self.weights)
                # add the index of the best rule in this iteration to the list of best rules
                self.best_rule.append(argmin_index)
                self.chosen_alphas.append(self.alpha)

    def predict(self, data: np.ndarray, index: int) -> Union[int, np.ndarray]: # Return the
class of the point
        predictions = np.zeros(len(data))
        for i in range(index):
            predictions += self.chosen_alphas[i] *
self.rules[self.best_rule[i]].predict(data)
        return np.where(predictions >= 0, 1, -1)
```

Utility.py:

```
import numpy as np

FILE_NAME = "circle_separator.txt"

def read_data(filename: str) -> np.ndarray:
    with open(filename) as f:
        lines = f.readlines()
        data = np.zeros((len(lines), 3))
        for i in range(len(lines)):
            data[i] = np.array([float(x) for x in lines[i].split()])
        return data

def random_train_test_split(data: np.ndarray) -> (np.ndarray, np.ndarray):
    shuffled_data = np.random.permutation(data)
    train_data = shuffled_data[0: int(len(data) * 0.5)]
    test_data = shuffled_data[int(len(data) * 0.5):]
    return train_data, test_data
```

main.py:

```
import AdaBoost
import numpy as np
import Utility
import Line
from itertools import combinations, permutations
import Circle
import matplotlib.pyplot as plt

def __run_adaboost(data: np.ndarray, iteration: int, method: str = "line") -> None:
    rules = []
    if method == "line":
        for p1, p2 in combinations(data, 2):
            # add all possible lines to the list of rules with red color side and blue color
side
            rules.append(Line.Line(p1[0:2], p2[0:2], 1)) # red color side
            rules.append(Line.Line(p1[0:2], p2[0:2], -1)) # blue color side
    if method == "circle":
        for p1, p2 in permutations(data, 2):
            # add all possible circles to the list of rules with red color side and blue
color side
            rules.append(Circle.Circle(p1[0:2], p2[0:2], 1)) # red color side
            rules.append(Circle.Circle(p1[0:2], p2[0:2], -1)) # blue color side
    print("AdaBoost with", method+"s:")
    np.random.seed(42)
    avg_empirical_error = np.zeros(8)
    avg_true_error = np.zeros(8)
    for i in range(iteration):
        print("Iteration: ", i+1)
        train, test = Utility.random_train_test_split(data)
        ada_boost = AdaBoost.AdaBoost(rules)
        ada_boost.fit(train[:, 0:2], train[:, 2], 8) # run 8 iterations
        for j in range(8):
            empirical_error = np.sum(ada_boost.predict(train[:, 0:2], j + 1) != train[:, 2])
            true_error = np.sum(ada_boost.predict(test[:, 0:2], j + 1) != test[:, 2])
            avg_empirical_error[j] += empirical_error / len(train)
            avg_true_error[j] += true_error / len(test)
    avg_empirical_error /= iteration
    avg_true_error /= iteration
    print("Results for method:", method)
    print("Average empirical error: ", avg_empirical_error)
    print("Average true error: ", avg_true_error)
    # plot the average empirical error and average true error in same graph of Circle or Line
    plt.title("Average empirical error and average true error of " + method)
    # add title to axis
    plt.xlabel("Numbers of " + method + "s")
```



```
plt.ylabel("Error")
# color the graph of empirical error with red and true error with blue
plt.plot(range(1, 9), avg_empirical_error, 'r', label="Empirical error")
plt.plot(range(1, 9), avg_true_error, 'b', label="True error")
# add legend to the plot
plt.legend()
plt.show()

if __name__ == '__main__':
    data = Utility.read_data(Utility.FILE_NAME)
    #data[:, -1] = np.where(data[:, -1] == 0, -1, data[:, -1])
    __run_adaboost(data, 50, "line")
    # __run_adaboost(data, 50, "circle")
```