למידת מכונה תרגיל 2

Report

רונלי ויגננסקי 211545892

אלגוריתם KNN:

הפרמטרים שמצאתי: k = 9

Normalize = z_score

Metric = Canberra distance

זו הפונקציה שכתבתי כדי למצוא את הk האופטימלי:

```
def find_k_in_knn(rands, k_range):
    counters = [0] * k_range

    for j in range(rands):
        validation, new_train, new_labels, new_labels_validation = split_k_fold(rands, j)

        dictionary_test = dict()
        # putting the test data in a dictionary and normalize it
        put_data_in_dict(dictionary_test, new_train)
        normalization(dictionary_test, validation, "z_score")

        dictionary = dict()
        # putting the train data in a dictionary and normalize it
        put_data_in_dict(dictionary, new_train)
        normalization(dictionary, new_train, "z_score")

        for k in range(1, k_range):
            labels = knn(validation, new_train, new_labels, k, "z_score", canberra_distance)
            for i in range(len(labels)):
                if labels[i] == new_labels_validation[i]:
                    counters[k] = counters[k] + 1

    for x in range(len(counters)):
        counters[x] = counters[x] / rands
    print("k: ", counters.index(max(counters)))
    print("accuracy: ", max(counters))
    print("success rate: ", (max(counters) / len(new_labels_validation)) * 100, "%")
    return counters.index(max(counters)), (max(counters) / len(new_labels_validation)) * 100, 48 - max(counters)
```

המטרה היא למצוא את k השכנים הקרובים ביותר, לפיהם נקטלג את הנקודות למחלקות 0 1 או 2. כתבתי לולאה שרצה מספר פעמים כך שבכל איטרציה רצה על כל הkים. זאת כדי למצוא את הk האופטימלי שנותן לנו את ה loss המינימלי. כפי שניתן לראות אנחנו מחלקים את הtrain המקורי שלנו לtrain ול validation (20% validation) בפונקציה split_k_fold, פונקציה אשר מממשת את הרעיון של חלוקה לKFold שהוסבר בכיתה, והיא מצורפת להלן. לאחר הדפסת הkים האופטיונליים וה lossים בהתאמה, קיבלתי שהk עבורו יוצאים לנו אחוזי הצלחה גבוהים ביותר הינו 9 ולכן שלחתי אותו כפרמטר.

```
avg k:  9.0
avg rates:  92.91666666666666
avg losss:  3.3999999999999986
```

פונקציית הsplit:

```python
def split_k_fold(k, slice):
    validation = []
    new_train = []
    new_labels_validation = []
    new_labels = []

    size = len(train_x)
    size_for_part = int(size / k)

    # create the part of validation
    for i in range(0 + size_for_part * slice, 0 + size_for_part * slice + size_for_part):
        validation.append(train_x[i])
        new_labels_validation.append(int(train_y[i]))
    # create the pare of the train
    for j in range(len(train_x)):
        if j in range(0 + size_for_part * slice, 0 + size_for_part * slice + size_for_part):
            continue
        new_train.append(train_x[j])
        new_labels.append(train_y[j])

    return np.asarray(validation), np.asarray(new_train), np.asarray(new_labels), np.asarray(new_labels_validation)
```

לגבי הנרמול, הרצתי את שתי האופציות של הנרמול, min_max וz_score ועבור z_score יצאו תוצאות טובות יותר, כלומר loss נמוך יותר ולכן שלחתי את זה. כנ"ל לגבי סוג המטריקה. מימשתי סוגים שונים של מטריקות: Euclidean_diastance, manhattan_distance, Canberra_distance, ועבור Canberra קיבלתי תוצאות ביותר. מצורפים הממימושים לפונקציות המטריקה השונות שעשיתי:

```python
def euclidean_distance(first_point, second_point):
    return np.linalg.norm(first_point - second_point, ord=2)


def manhattan_distance(first_point, second_point):
    sum = 0
    for i, j in zip(first_point, second_point):
        num = np.abs(i - j)
        sum = sum + num

    return sum


def canberra_distance(first_point, second_point):
    sum = 0
    for i, j in zip(first_point, second_point):
        numerator = np.abs(i - j)
        denominator = np.abs(i) + np.abs(j)
        if denominator == 0:
            continue
        sum = sum + numerator / denominator

    return sum
```

אלגוריתמי perceptron, passive aggressive, SVM :

המכנה המשותף בין אלגוריתמים אלו הוא שבכולם מצאתי את מספר ה epochs הטוב ביותר. כמו כן, עשיתי shuffle על המידע כדי שהאלגוריתם לא יזכור את המידע אלא ילמד אותו. אבל, כיוון שרציתי שזו תהיה הרצה קבועה בסבמיט, קבעתי SEED עבור כל אלגוריתם, SEED שיהפוך את הריצה לטובה יותר ע"י סידור מסוים של האינדקסים בdata באמצעות הshuffle.

הקוד שכתבתי עבור אלגוריתמים אלה הוא זהה פרט לקריאה לפונקציה המתאימה- perceptron, pa, svm בהתאמה, ופרט לSEED הטוב שמצאתי עבור כל אחד.

רעיון הפונקציה הוא לרוץ על כל האופציות לחלוקה ל train ול validation ע"י KFold (מצורף הקוד שלי לזה). בתוך זה לרוץ על מספר גדול של epochs, לאמן את המודל ואז לשלוח לפונקציית predict שמימשתי, ולהשוות בין התיוגים האמיתיים לתיוגים שיצאו אצלי באלגוריתם. לפי זה מחשבים את מספר losso ים ואת אחוז ההצלחה. בסופו של דבר בחרתי את הepoch עבורו losso יצא מינימלי.

בנוסף, כפי שניתן לראות בקוד, את כל החישוב הזה שמתי בלולאה שרצה על it, כלומר מספר איטרציות, כדי לרוץ יותר פעמים ושהתוצאה תהיה נכונה יותר לכלל הריצות. גם זה, נמצא בלולאה

עבור מציאת הSEED הטוב. גם הSEED נבחר לפי אחוז ההצלחה הגבוה ביותר, כלומר הepoch האופטימלי.

הפונקציה למציאת הפרמטרים שתארתי לעיל: צירפתי את הקוד של מציאת הפרמטרים ל perceptron, ושמתי בהערה את הקריאות ל svm ול pa, אשר מתאימות במציאת הפרמטרים עבור אגוריתמים אלו. בנוסף, במקום PER_SEED, יש לי בקוד של האלגוריתמים האחרים PA_SEED ו SVM_SEED ל passive aggressive ו svm בהתאמה.

```
def find_epochs_perceptron(epoch, it, parts=5):
    list_of_loss_and_epochs = [0] * epoch
    list_of_loss_and_epochs[0] = sys.maxsize

    per_seeds = []
    good_epoch_and_seed = []
    # random seeds
    for i in range(10):
        PER_SEED = np.random.randint(1000)
        per_seeds.append(PER_SEED)
    for s in per_seeds:

        for times in range(it):

            for k in range(parts):
                validation, new_train, new_labels, new_labels_validation = split_k_fold(parts, k)
                dictionary_test = dict()
                # putting the test data in a dictionary and normalize it
                put_data_in_dict(dictionary_test, new_train)
                normalization(dictionary_test, validation, "z_score")

                dictionary = dict()
                # putting the train data in a dictionary and normalize it
                put_data_in_dict(dictionary, new_train)
                normalization(dictionary, new_train, "z_score")

                # add bias
                validation = np.hstack((np.ones((validation.shape[0], 1)), validation))
                for i in range(1, epoch):
                    loss = 0
                    weights = perceptron(i, new_labels, new_train, validation, "z_score")
                    # weights = svm(i, new_labels, new_train, validation, "z_score")
                    # weights = pa(i, new_labels, new_train, validation, "z_score")
                    y_hats = predict(weights, validation)
                    for j in range(len(y_hats)):

                        if y_hats[j] != new_labels_validation[j]:
                            loss += 1
                    list_of_loss_and_epochs[i] += loss
        for index in range(len(list_of_loss_and_epochs)):
            list_of_loss_and_epochs[index] /= parts * it

        good_epoch_and_seed.append((list_of_loss_and_epochs.index(min(list_of_loss_and_epochs)), s))
        sorted_ep_seed = sorted(good_epoch_and_seed)
        best_seed = sorted_ep_seed[0][1]
    return list_of_loss_and_epochs.index(min(list_of_loss_and_epochs)), min(list_of_loss_and_epochs), best_seed
```

אלגוריתם perceptron:

הפרמטרים שמצאתי: number of epochs = 15

Learning rate = 0.1

PER_SEED = 859400

הפונקציה שכתבתי כדי למצוא את מספר הepochs: מצורפת לעיל.

לגבי הlearning rate, ניסיתי ערכים בכפולות 10 וקבעתי לבסוף את התוצאה שהביאה לי את ה loss המינימלי. פרמטר זה קובע את קצב המציאה של המינימום, אם נמוך מידי ייתכן ויקח המון זמן עד למציאה, ואם גדול ייתכן ונפספס את הערך הרצוי. ולכן הערך 0.1 גרם לי לתוצאה רצויה הן מבחינת מציאת המינימום והן מבחינת זמן הריצה ומספר ה epochs הנחוצים.

אלגוריתם passive aggressive:

הפרמטרים שמצאתי: number of epochs = 16

PA_SEED = 5357

הפונקציה שכתבתי כדי למצוא את מספר הepochs: זהה למצורפת לעיל, רק שבמקום החץ האפור, החץ הכתום שבהערה צריך להופיע. ובמקום PER_SEED יהיה את PA_SEED.

אלגוריתם SVM:

הפרמטרים שמצאתי: number of epochs = 17

Learning rate = 0.1

Lambda = 0.01

SVM_SEED = 3481

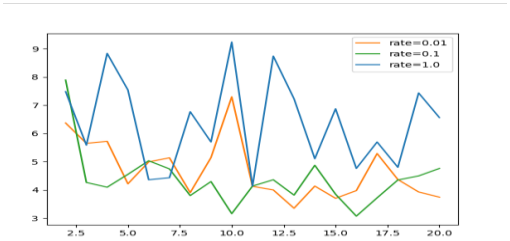
הפונקציה שכתבתי כדי למצוא את מספר הepochs: זהה למצורפת לעיל, רק שבמקום החץ האפור, החץ הירוק שבהערה צריך להופיע. ובמקום PER_SEED יהיה את SVM_SEED.

מציאת ה learning rate זהה לתיאור שכתבתי בperceptron.

מציאת הlambda – ניסיתי להריץ עם ערכי lambda שונים, גם פה בכפולות של 10, ולקחתי את הערך שעבורו הloss יצא לי מינימלי. המשמעות של lambda זה כמה "להעניש" על טעות בסיווג. גם על טעות שהינה בmargin של 1. ככל שערך הlambda גדול יותר מענישים יותר ואז יהיו יותר loss ים. וככל שקטן יותר פחות loss ים. אבל צריך לשים לב שלא מקטינים מידי את ה lambda באופן כזה שבvalidation שלנו יצא שהloss באמת יורד ויורד, אך בtest האמיתי נראה עליה במספר הloss ים בגלל overfitting. לכן, ניסיתי לבחור ערך שיהיה טוב גם בvalidation ובשאיפה גם בtest.

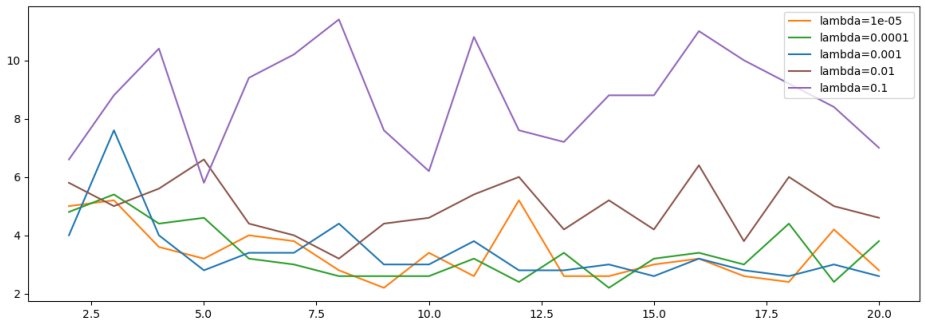
ניסיתי להמחיש את בחירת ה learning rate וה lamda ע"י גרפים בSVM:

Learning rate:



ניתן לראות ש0.1 זה הערך שנותן loss מינימלי בסביבות 17 epochs שזה מה שיצא לי הטוב ביותר בsvm.

:Lambda



ניתן לראות ש0.01 זה ערך טוב עבורו יש loss קטן. רואים בגרף שעבור lamda גבוה יותר, מענישים יותר, כלומר loss גבוה יותר, ולכן לא ניקח את ערך ה lamda של הקו הסגול בהיר.

אמנם יש ערכים שהביאו lossים נמוכים יותר מהערך של lamda השווה ל0.01 אך ייתכן שאלו ערכים שיגרמו לoverfitting בtest האמיתי כפי שהסברתי לעיל, ולכן בחרתי בערך ה lamda של 0.01.

בנוסף, בשלושת האלגוריתמים של perceptron, passive aggressive, svm הוספתי bias, עמודה של אחדות. הביצועים עם הוספת ה bias היו טובים יותר.

דבר נוסף שעשיתי זה **feature selection**, כלומר הרצתי את כל המתואר לעיל על כל אלגוריתם בצורה כזו שבחרתי להוריד feature מסוים כל פעם, ובדקתי את התוצאות עבור הdata ללא feature זה. גיליתי שהביצועים הטובים ביותר היו כאשר הורדתי את העמודה האחרונה, כלומר את ה feature האחרון. לפיכך, הורדתי עמודה זו וזה ה train ששלחתי למודל הלמידה בכל אחד מהאלגוריתמים השונים.