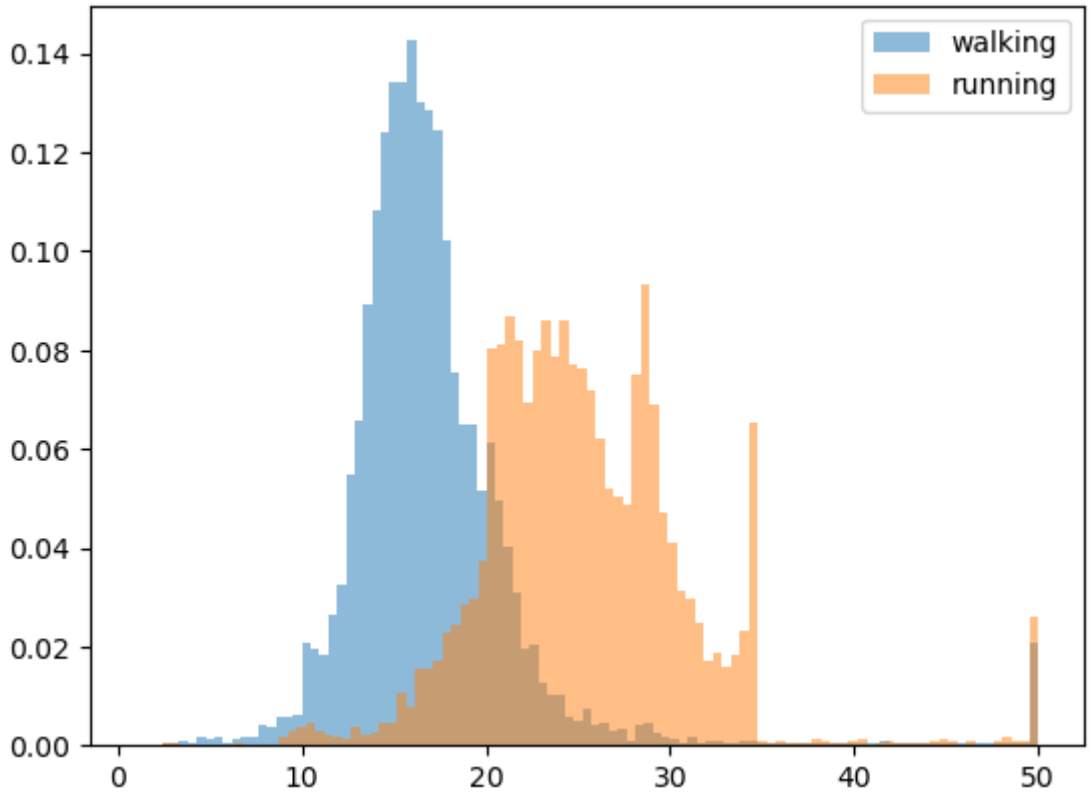
דוח סופי - אינטרנט של הדברים

חלק 1 - ניתוח נתונים משותפים.

זיהוי הליכה או ריצה:

חשבנו והגענו למסקנה שבריצה הערכים המוחלטים של התאוצה יהיו גבוהים יותר מבהליכה. לכן, על מנת להבדיל בין הליכה לריצה החלטנו לעשות מבחן סטטיסטי להבדל בין התפלגויות באופן הבא:
ניקח מדגמים בגודל 20, ונבדוק מהוא הערך המקסימלי על החלון הנל. מתוך ההנחה שהערכים המוחלטים של התאוצה יהיו גבוהים יותר בריצה מבהליכה, אם ניקח את המקסימום, תחת ההנחה שהמדגמים מתפגים נורמאלית, המובהקות הסטטיסטית להבדל תהיה גדולה יותר. לכל חלון סיווגנו לריצה או הליכה באופן הבא: אם המקסימום גדול מערך כלשהו אז זה ריצה, אחרת הליכה. עבור סיווג למקטע שלם בדקנו על כל חלון בגודל 20 האם הסיווג הוא ריצה או הליכה, ולאחר מכן לקחנו את החלטת הרוב, שיטה זו מאוד עמידה לערכים חריגים שמופיעים הרבה בדאטה שלנו ועובדת דיי טוב. ACC = 0.9.

בגרף הבא אנו רואים את התפלגות הערך המקסימלי בחלון של 20:



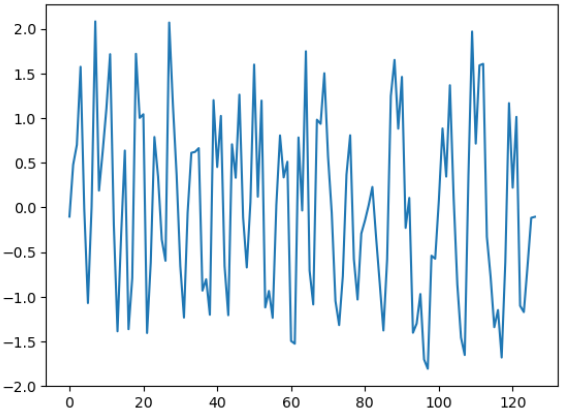
הערך שבחרנו כדי להבדיל היה 20.

ספירת צעדים:

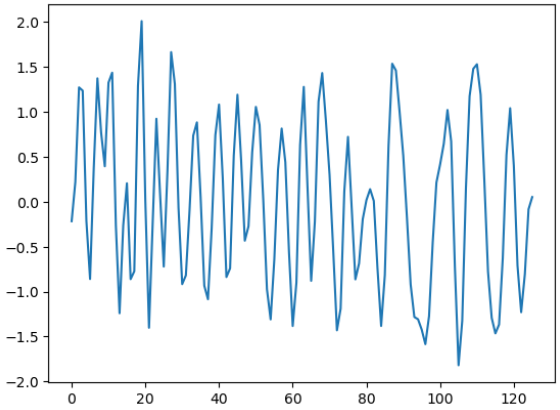
על מנת לספור כמות צעדים עשינו שני שלבים:

ניקוי והחלקת הדאטה: גם כאן הסתכלנו על הנורמה של התאוצה. החלטנו שכל ערך גדול מ35 הוא ערך חריג, זאת לפי מה שראינו בDATA. החלקנו את הדאטה עם החלקה אקספוננציאלית עדינה יחסית. לאחר מכן החלטנו לנרמל את הנתונים שכן כל חיישן ברגישות מעט שונה וכל אדם הולך קצת חזק או חלש יותר, מה שיצר שוני בדגימות. על מנת לעשות זאת חילקנו את הנתונים בשונות שלהם. לבסוף השתמשנו בהתמרת פוריה כדי לעשות low pass filter על הדאטה, המטרה הייתה להחליק את הדאטה באופן כזה שישמר את הפיקים והמחזוריות הכללית.

בלי LOW PASS FILTER



עם LOW PASS FILTER



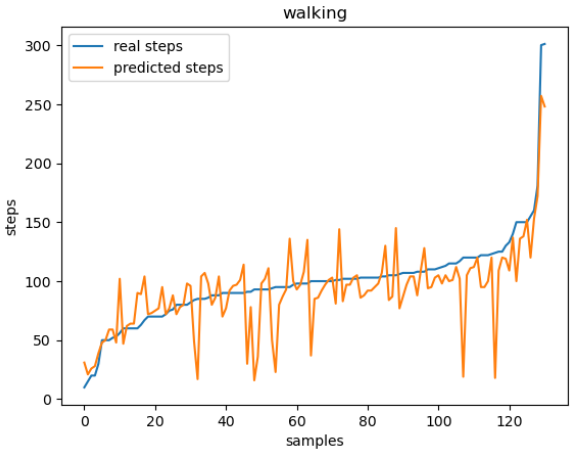
חיזוי מספר צעדים: על מנת לעשות את זה החלטנו להשתמש בפונקציה find_peaks. חילקנו את הדאטה לריצה והליכה, ועבור כל אחד מהם מצאנו את ההיפר פרמטרים האידיאלים, הן לחהלקה, והן לfind_peaks.

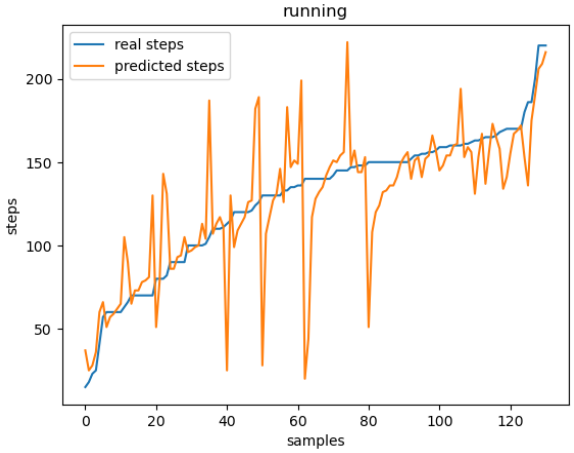
לבסוף קיבלנו את התוצאות הבאות:

walking: 0.128 Median Absolute Relative Difference
running: 0.078 Median Absolute Relative Difference

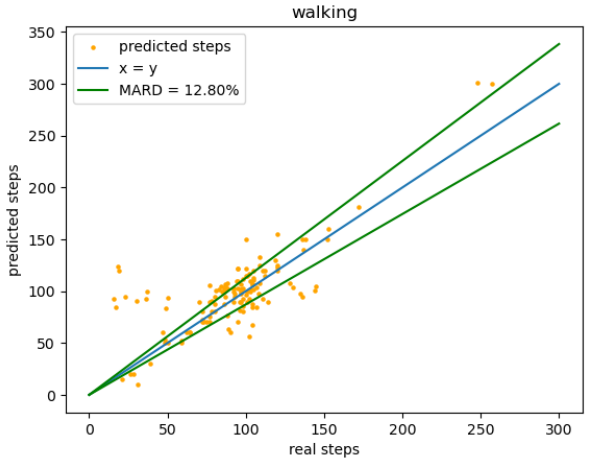
Mean Absolute Relative Difference -> Median(Abs(1 - (Pred_Labels/True_Labels)))

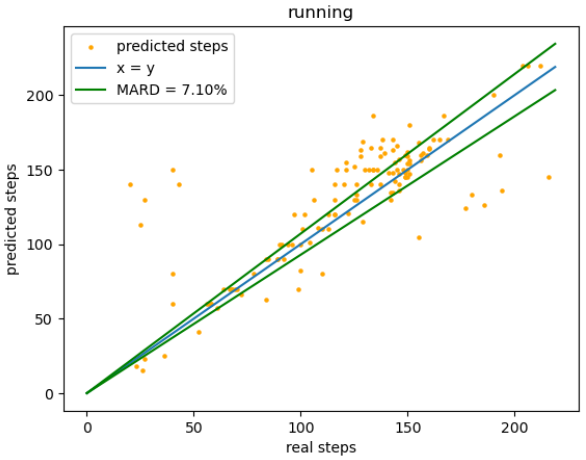
בגרפים הבאים הדגימות (ממוינות לפי מספר צעדים) הם ציר הX והמספר שחזינו הוא ציר הY:





בגרפים הבאים, הנקודות הן מספר הצעדים הנחזה של דוגמה לעומת האמיתית, והקוים הם y = x, וההפרש הממוצע ממנו של הנקודות.





<u>הסבר על הקוד:</u>

על מנת להריץ את הקוד ולבדוק אותו על קבצים יש לעשות את הדברים הבאים:

בתחילת הקובץ יש משתנה train, יש לשנות אותו לtrue אם רוצים לאמן על דאטה חדש.

את הדאטה החדש\הקובץ שרוצים לבדוק עליו יש לשים בתיקיה data_set באותו המיקום של קובץ הקוד, על מנת שהקוד יוכל לגשת אליו.

עבור כל חלק (הליכה\ריצה וספירת הצעדים) יש בתוך הסקופ שלו כותרת predictions. שם ניתן לראות מהי התחזית לכל קובץ בתיקיה הנ"ל.

קטעי קוד נבחרים:

החלטה על חלון האם מדובר בהליכה או ריצה:

```
def classify(data):
    # max l2 norm of acceleration
    norms = np.sqrt(data['acc_x']**2 + data['acc_y']**2 + data['acc_z']**2)

    # max acceleration
    max_acc = np.max(norms)

    max_acc = min(max_acc, 50)

    return max_acc > 20, max_acc
✓ 0.0s
```

החלטה על קובץ האם מדובר בהליכה או ריצה:

```python
import tqdm
# lets calculate the accuarcy of the classifier
true_labels = []
predicted_labels = []
my_dict = {"walking":[], "running":[]}
for d in tqdm.tqdm(data):
    frame_labels = []
    for i in range(20, len(d[0])):

        # take the last 10 values
        last_20 = d[0].iloc[i-20:i]
        # classify
        classified, max_acc = classify(last_20)
        # append predicted label
        frame_labels.append(classified)

        # append to dict
        if d[1]:
            my_dict["walking"].append(max_acc)
        else:
            my_dict["running"].append(max_acc)

    pred = np.mean(frame_labels) < 0.5

    true_labels.append(d[1])
    predicted_labels.append(pred)

predicted_labels = np.array(predicted_labels)
true_labels = np.array(true_labels)
```

✓ 1m 38.4s

פונקציה הסופרת את הצעדים על prossesed data:

```python
# classic method for counting steps
from scipy.signal import find_peaks

def count_steps(data, threshold=1.8, distance=3, low_pass=350):
    # Apply Fast Fourier Transform (FFT)
    fft_values = np.fft.rfft(data)

    # Low pass filter
    fft_values[low_pass:] = 0

    # Apply Inverse Fast Fourier Transform (IFFT) to get the filtered signal
    data = np.fft.irfft(fft_values)

    peaks, _ = find_peaks(data, threshold=threshold, distance=distance)
    return str(len(peaks))
```

✓ 0.1s

ספירת הצעדים עבור קובץ בודד:

```
if d[1]:
    category = "walking"
else:
    category = "running"

# calc l2 norm of acceleration
norms = np.sqrt(d[0]['acc_x']**2 + d[0]['acc_y']**2 + d[0]['acc_z']**2)

# remove nan values
norms = norms[~np.isnan(norms)]

m = np.mean(norms)

if m > 35:
    continue

# if norm > 35  then norm = m
norms[norms > 35] = m

# normalize
norms = (norms - m) / np.std(norms)

# smooth
norms = np.array(norms)
#norms = smooth_signal_fast(norms, window_size=best_window_size[category])
norms = exp_smooth_signal(norms, alpha=best_window_size[category])

# norms to float array
lst = np.array(norms)

threshold=best_treshold[category]
distance=best_distance[category]

cs = count_steps(lst, threshold=threshold, distance=distance, low_pass=best_low_pass[category])

real[category].append(d[2])
pred[category].append(int(cs))

pred_steps[idx] = int(cs)
```

חלק 2 - פיתוח מערכת ניטור פעילות גופנית

סיפור מוצר: bADAM-tss

עבור אדם סילברברייך, חובב טכנולוגיה וחובב מוזיקה, הנסיעה לעבודה הייתה תמיד החלק הכי משעמם ביום. למרות התשוקה לתיפוף, לא ניתן לשאת ערכת תופים ברכבת ואדם המסכן נותר לא מרוצה. כל שנשאר לו זה להקיש באצבעותיו על ברכיו בקצב השירים האהובים עליו.

ואז הגיעה bADAM tss, מערכת התופים החשמליים החדישה של Adam bois - אבן דרך במסעו המוזיקלי של אדם. נייד, ידידותי למשתמש וחדשני, bADAM tss הפך את נסיעת הסרק שלו לסשן תיפוף תוסס, וכמובן שהוא מאוד התלהב מהשם שהרגיש כאילו הונדס במיוחד אישית בשבילו.

עוצב עבור חובבי תופים כמו אדם, bADAM tss מורכב מחישן תאוצה המולבש על היד, והופך כל שיא אנרגטי לצליל תוף דינמי. עם חיישן מרחק שמודד את גובה היד, bADAM tss יכול לחקות שלושה צלילי תופים שונים. מושב הרכבת הפך לבמה של אדם, וידיו, מקלות התיפוף.

כעת, אדם יכול מצד אחד לנגן עם האוזניות שלו כאוות נפשו מבלי להפריע לעמיתיו הנוסעים, ולנגן באצטדיון בהופעה עם חיבור מהטלפון לרמקולים ענקיים מצד שני.

קלת משקל וקלה לנשיאה, bADAM tss עוקבת אחרי אדם לכל מקום. אהבתו לגאדג'טים טכנולוגיים הועצמה עוד יותר. bADAM tss בישבילו היא לא רק פיסת טכנולוגיה מסודרת, אלא כלי שנתן כנפיים לתשוקה שלו לתיפוף. לא סתם עוד גאדג'ט, אלה מערכת התופים אישית וניידת, שתמיד תהיה מוכנה לתופף בקצב התוסס של היום יום.

<u>קהל היעד:</u>
המשתמשים בכלי הנגינה יכולים להיות מוזיקאים, במיוחד אלו המתעניינים במוזיקה ניסיונית ואלקטרונית. מתופפים שרוצים לחקור דרכים חדשות ליצירת קצב ללא ערכות תופים מסורתיות יכולים למצוא את הכלי הזה מושך וכן גם אלו שהיו מעוניינים לתופף במרחבים ציבוריים חופשיים הן מהמימדים הגדולים של מערכת תופים רגילות והן מהצליל החזק שהן משמיעות בפומבי.

עבור מוזיקאים המחפשים כלים מוזיקליים חדשניים ולא שגרתיים, הכלי שלנו מציע גישה רעננה לביטוי מוזיקלי. כלי זה מאפשר למשתמשים להתנסות בהפקה מוזיקלית מבוססת תנועה, ולגשר על הפער בין כלים מסורתיים לממשקים אלקטרוניים.

<u>חיישנים:</u>
השתמשנו בחיישן מרחק, זאת על מנת לחשב את המיקום של היד בציר הגובה שממנה נוכל להפיק את סוג התוף שהמשתמש רוצה לנגן בו, והשתמשנו בחיישן תאוצה שהיה מחובר לכף היד כדי להחליט מתי להשמיע את קול התוף בעזרת זיהוי פיקים. בעזרת שני החיישנים אנחנו יכולים לדמות מערכת תופים שכוללת שלושה סוגי תופים.

<u>הצורך:</u>
הצורך היזמי בפרויקט הוא לספק כלי נגינה רב תכליתי וחדשני המציע חווית משתמש ייחודית. הבעיה שאנו מתמקדים בה היא הכמות המוגבלת של אפשרויות הזמינות למוזיקאים הרוצים לבטא את עצמם באמצעות כלי נגינה נשלטי תנועה. בעיה נוספת שאנו רוצים לפתור היא שמערכות תופים היא כלי מאסיבי, מגושם ולא נייד שאי אפשר לנגן בו בכל מקום וזמן עקב היותו ידוע לשמצע בתור כלי רועש.

<u>הסבר מפורט על המכשיר והליך העבודה:</u>

המוצר מורכב משני חיישנים, לוח esp ואפליקציה שמתקשרת איתם. החיישנים קולטים את תנועות היד והגובה שלה, הESP משדר את זה בbluetooth לאפליקציה שמנגנת צלילי תופים בהתאם לסיגנלים שהיא קולטת.
חיישן המרחק: קבענו כי כל 20 סנטימטרים מחליפים סוג תוף. מדדנו את מרחק היד מהחיישן ושינינו בהתאם.

חיישן התאוצה: כאן המטרה הייתה לקבוע מתי לנגן את הצליל של התוף. זאת עשינו בעזרת זיהוי פיקים בפייתון על הסיגנל של התאוצה בציר הגובה. כאשר זהינו שיש פיק שעונה לדרישות שלנו בדגימה לפני האחרונה ניגנו את הצליל של התוף, כך שיש דילי אבל עקב קצב הדגימה הגבוה הוא לא מורגש. את חיישן התאוצה היינו חייבים לחבר בחוטים ארוכים במיוחד שכן הוא אמור לזוז הרבה ביחס לESP שלא זז בכלל ולכן צריך להיות לו תווח תנועה עצמאי גדול.

השתמשנו בscipy.signal.find_peaks כדי למצוא האם היה peak. רצינו שנשמיע את התוף רק ברגע אחרי שהפיק היה. לא ניתן לעשות ברגע שבו היה מאחר שfind_peaks לא אומר שיש peak בכניסה האחרונה בוקטור. לכן, "התפשרנו" על הרגע אחרי (כלומר בפעם הבאה שבה נקבל מדידה של תאוצה)
מבחינת הקוד:

```
def add_accel(a):
    lst.append(-float(a))
    if len(lst)>5:
        idx,_ = find_peaks(lst[-5:], height=0.0, threshold=0)
        if len(idx) >=1 and idx[-1] == 3:
            return str(lst[3])
    return "False"
```

```
is_peak = pyobj.callAttr("add_accel", String.valueOf(z)).toString();
```

הוספנו את מינוס התאוצה מאחר שיש תיפוף כאשר מורידים את המד תאוצה.

אפליקציה: האפליקציה מתקשרת עם הESP, ומנגנת מהרמקול של הטלפון את צלילי התופים. עשינו שצלילי התופים יכולים לחפוף שכן סאונד אחד לא אמור לקטוע את השני. הוספנו כפתורים להקלטה והשמעת הקלטה, כלומר כאשר לוחצים על כפתור ההקלטה מתחילים לשמור את הנתונים, ובסופה שומרים לקובץ dtg.
כמו כן יש לנו כפתור שפותח את מנהל הקבצים של המחשיר וכאשר בוחרים קובץ dtg האפליקציה מנגנת אותו מהרמקולים של הטלפון כמו שהוקלט, באופן זה ניתן להקליט לשמוע וגם לשתף את הקבצים האלו עם אנשים אחרים שיש להם את האפליקציה.