

גלאי חריגות פשוט ל time-series data

הקדמה

סטודנטים יקרים שלום רב,

בימנו הרבה פרויקטים מכילים אלמנטים של data science / ML בצד השרת ואפליקציות שונות בצד הלקוח. גם קיימים כלים מאד מתקדמים שמאפשרים לנו לעבוד ברמת אבסטרקציה מאד גבוהה. גבוהה עד כדי כך שאנו עלולים שלא להבין כיצד הדברים עובדים מאחורי הקלעים, וכתוצאה להפעיל אותם לא נכון. לכן, כדי ליצור הבנה עמוקה, בקורס זה אנו נממש דוגמאות פשוטות כמעט מאפס ונבנה אותן שכבה מעל שכבה בפרויקט מתגלגל עד שנגיע לסיום מכובד של מערכת שלמה.

למעשה פרויקט זה יכול לשמש כחלון הראווה שלכם כשתרצו להציג את הניסיון התכנותי שצברתם.

פרויקט זה מכיל את האלמנטים הבאים:

- שימוש בתבניות עיצוב וארכיטקטורה
- תקשורת וארכיטקטורת שרת-לקוח
- שימוש במבני נתונים \ במסד נתונים
- הזרמת נתונים (קבצים ותקשורת)
- הטמעה של אלגוריתמים מבוססי data science בתוך המערכת שניצור
- תכנות מקבילי באמצעות ת'רדים
- תכנות מוכוון אירועים, אפליקציית desktop עם GUI

בכל סמסטר תגישו מספר אבני דרך:

פת"מ 1

1. מטלת "חימום" – ספריית קוד
2. גלאי חריגות פשוט
3. שרת גילוי חריגות

וב-פת"מ 2 אפליקציות desktop לתחקור טיסה וגילוי חריגות

במסמך מקוון שיפורסם באתר המודול של הקורס יופיעו כל פרטי ההגשה – מה מגישים, לאן, כיצד תתבצע הבדיקה וכדומה.

בהצלחה!

אבן דרך 1 – ספריית קוד לצורך גילוי חריגות

מיני-משימה – לימוד עצמי – GIT

לפני שנתחיל, לימדו לעבוד עם version control. זהו לא חומר לימוד אקדמי, אלא כלי. עם כלים של version control אתם יכולים לשמור את הקוד שלכם בענן, לעבוד במקביל עם שותפים ולמזג את הקוד שכתבו, לחזור לגרסאות קודמות ועוד. מערכת GIT הפכה בשנים האחרונות לפופולרית ביותר, ולכן עליכם להכיר אותה (<https://git-scm.com/>). אחד משרתי האחסון הפופולריים נקרא GitHub.

קיימים לא מעט tutorials לשימוש ב-GIT. אולם, הדרך הטובה ביותר לדעתי ללמוד את הכלי היא להתנסות איתו. צרו עם השותפים פרויקט קטן ב C++ בסגנון "hello world" ותשתפשו עליו בכל הקשור להעלאה, הורדה, עדכון, מיזוג של קוד וכדומה. כאשר אתם מרגישים שהשימוש בכלי כבר טבעי לכם, תוכלו להתחיל את הפרויקט. לאורך השנה, עליכם להשתמש בכלי זה.

ספריית קוד לצורך גילוי חריגות

בפרויקט זה נממש גלאי חריגות (Anomaly Detector) פשוט המבוסס על שיטות סטטיסטיות פשוטות. לשם כך נצטרך להיעזר בספריית קוד שאותה נממש באבן דרך זו. עליכם לממש את הפונקציות הבאות בקובץ StatLib.java:

```
// simple average
public static float avg(float[] x);
// returns the variance of X and Y
public static float var(float[] x);
// returns the covariance of X and Y
public static float cov(float[] x, float[] y);

// returns the Pearson correlation coefficient of X and Y
public static float pearson(float[] x, float[] y);

// performs a linear regression and returns the line equation
public static Line linear_reg(Point[] points);
// returns the deviation between point p and the line equation of the
points

public static float dev(Point p, Point[] points);
// returns the deviation between point p and the line
public static float dev(Point p, Line l);
```

כאשר נותנות לכם המחלקות Line ו Point:

```
public class Point {
    public final float x,y;
    //...
}
public class Line {
    public final float a,b;
    //...
}
```

הפונקציה var צריכה להחזיר את השונות (variance) עבור מדגם סופי של משתנה בדיד X .

בהינתן אוכלוסייה בגודל N נוכל לחשב את השונות על ידי לקיחת הסתברות אחידה:

$$\text{Var}(X) = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2 = \left(\frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \mu^2$$

כאשר

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i \text{ הוא ערך התוחלת.}$$

הפונקציה cov צריכה להחזיר את השונות המשותפת (covariance) של המשתנים X ו- Y .

$$\text{cov}(X, Y) = E(XY) - E(X)E(Y) = E((X - E(X))(Y - E(Y)))$$

כאשר גם כאן מדובר על מדגם סופי ולכן $E(X)$ כמו μ לעיל, הוא ממוצע פשוט.

הפונקציה Pearson היא מדד לקורלציה ליניארית. ככל שהערך קרוב יותר ל 1 כך שני המשתנים מתנהגים דומה יותר (כשהאחד עולה או יורד השני עולה או יורד בהתאמה). ככל שהערך קרוב יותר ל -1 כך המשתנים מתנהגים הפוך זה מזה (כשהאחד עולה, השני יורד). בשני המקרים מדובר בקורלציה מאד חזקה. לעומת זאת, ככל שהערך קרוב יותר ל 0 כך גובר חוסר הקשר בין המשתנים.

יש לממש את Pearson כך:

$$\frac{\text{cov}(X, Y)}{\sigma_x \sigma_y}$$

כאשר σ היא סטיית התקן (השורש של variance)

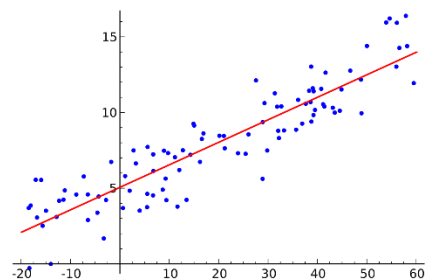
כעת נתונה המחלקה Line המייצגת משוואת ישר ע"י $Y = a \cdot X + b$ והמחלקה Point המייצגת נקודה דו-ממדית.

הפונקציה linear_reg צריכה בהינתן מערך של נקודות להחזיר את משוואת הישר ע"י טכניקה שנקראת רגרסיה ליניארית:

$$a = \frac{\text{COV}(x, y)}{\text{VAR}(x)} \text{ ואילו } b = \bar{y} - a\bar{x} \text{ הם הממוצעים של } (Y \mid X)$$

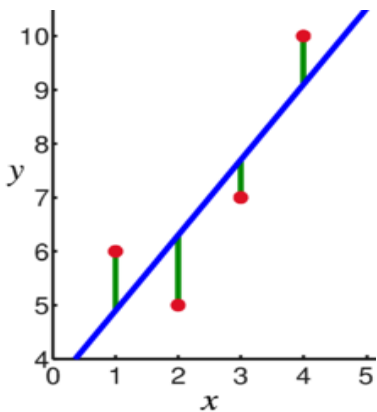
כך מתקבלת משוואת ישר $Y = a \cdot X + b$ המתארת את הנק' אם אכן יש ביניהן קורלציה ליניארית.

למשל:



לקריאה נוספת (ולאפשרויות חישוב נוספות):

https://en.wikipedia.org/wiki/Simple_linear_regression



כעת, שתי הפונקציות האחרונות בספרייה יעזרו לנו למדוד את הסטייה בין נק' כלשהי, לבין שאר הנקודות בהתפלגות. יש המון דרכים למדוד זאת. אנו נבחר בדרך פשוטה והיא המרחק בין הנק' לבין הישר שיצרו הנקודות. ראו את התרשים הבא. לכל נק' (x, y) יש את הנק' המתארת היכן צפינו שהיא תהיה על הישר

$(x, f(x))$ וההפרש בערך מוחלט $|f(x) - y|$ מתאר לנו את המרחק מהצפייה הזו. בתרשים אלו הקווים הירוקים. ככל שאורך הקו הירוק גדול יותר כך הנק' נראת לנו חריגה יותר ביחס לכל האחרות.

(הערת אגב: אין צורך למדוד את אורך הקו בין נק' אדומה וניצב לקו הכחול כי בגלל כלל פיתגורס החישוב הפשוט שלנו מספיק כדי לתאר עד כמה רחוקה הנק' האדומה מהקו הכחול)

במסמך ההגשות מפורט תאריך היעד להגשה, כיצד להגיש ולאן.

הבדיקה בודקת את הפונקציות לעיל על קלט סטנדרטי ואקראי וכן כל מיני מקרי קצה.

בהצלחה

אבן דרך 2 – גלאי חריגות פשוט

באבן דרך זו נרצה לממש גלאי חריגות פשוט עבור time-series data. ה data שלנו מאופיין בכך שהוא מגיע (בדרך כלל) בצורה של טבלה ע"י ערכים מופרדים בפסיק (csv). בשורה הראשונה ישנה כותרת לכל עמודה, אלו הם שמות המאפיינים (features) שנמדדו. לאחר מכן כל שורה מתארת את הערכים שהיו למאפיינים אלו בכל איטרציית דגימה (time step). למשל מידע שנדגם משעוני טיסה:

Time (seconds)	Altitude (feet)	Air Speed (knots)	Heading (deg)
0.1	12000.2	250	45.0
0.2	12001.0	250	45.3
0.3	12000.5	250	45.8

הערה: הטבלה לעיל היא רק דוגמה

- אלו לא בהכרח העמודות שיבדקו בקלט בדיקת המטלה
- ייתכן מס' גבוה של עמודות
- לא בהכרח העמודה הראשונה מייצגת את הזמן
- ייתכן קלט ללא עמודות זמן, עם הגדרה של קצב דגימה קבוע (למשל 10Hz).

המטרה שלנו היא בהינתן דגימה של שורה, להכריע מהר ככל הניתן האם היא חריגה ביחס לשורות הקודמות שראינו.

ה domain עבורו נפעיל את גלאי החריגות שלנו הוא סימולטור טיסה בשם FlightGear. כשנזריק תקלה לטיסה נרצה שהיא תתגלה כחריגה ע"י גלאי החריגות שלנו.

נניח כמה הנחות עבור גלאי החריגות שלנו:

- טיסות מתחילות כתקינות, ולאחר נק' זמן כלשהי עלולה להתפתח תקלה
- נתוני הטיסה שנדגמים מכילים לא מעט מאפיינים קורלטיביים זה לזה
- נצפה ממאפיינים שנמצאו כקורלטיביים על פני טיסה תקינה שלמה להישאר קורלטיביים גם בטיסות אחרות. אם צפייה זו תישבר נכריז על גילוי חריגה.

לכן, גלאי החריגות שלנו יפעל בשני שלבים – שלב מקדים (offline learning) ושלב גילוי (online anomaly detection).

בשלב המקדים ניקח קובץ של טיסה תקינה ונבדוק לכל מאפיין מי מהמאפיינים האחרים הכי קורלטיבי אליו ע"פ Pearson. כלומר לכל מאפיין f_i נמדוד באמצעות Pearson את הקורלציה בין וקטור הערכים של f_i לבין וקטור הערכים של כל f_j . כך לכל מאפיין f_i נחזיר את המאפיין f_j שעבורו קבלנו את תוצאת ה Pearson **בערך מוחלט** הגבוהה ביותר.

נשים לב ש:

- אין טעם לבדוק קורלציה בין עמודה כלשהי לעצמה (הקורלציה תמיד תהיה 1 ולכן חסרת משמעות)
- הקורלציה בין עמודה A ל B זהה הקורלציה בין B ל A כאשר מדובר ב Pearson. לכן ניתן לוותר על חצי מהחישובים.

לכן הפסאודו קוד למציאת הזוגות הקורלטיביים ביותר יהיה:

FOR $i \leftarrow 1$ TO N

$m \leftarrow 0, c \leftarrow (-1)$

FOR $j \leftarrow i + 1$ TO N

IF $(\rho \leftarrow |pearson(f_i, f_j)|) > m$ THEN $m \leftarrow \rho, c \leftarrow j$

IF $c \neq (-1)$ THEN associate f_i and f_j as correlated features

נשים לב למשמעויות הבאות:

- ייתכנו עמודות שלא נמצא להן בת זוג קורלטיבית – וזה בסדר.
- נניח ששמות העמודות הם A-Z. ייתכן מצב שעמודה A הכי קורלטיבית ל C, אך כשמגיע תורה של C להיסרק היא נסרקה מול D-Z ולכן ייתכן שאין לה קורלציה כלל או שהקורלציה הכי גבוהה שלה היא לעמודה אחרת מ A (נניח S) – וגם זה בסדר.
- במקרה כזה פשוט יהיו לנו הזוגות A,C וגם C,S.
 - לאלגוריתמים יותר מורכבים (שמבצעים גם דיאגנוזה) זה מידע בעל ערך.
 - לנו זה לא משנה. פשוט יהיו לנו שני הזוגות הללו.
- אם עמודה כלשהי לא נמצאה קורלטיבית ל Z, אז Z לא נסרק מול אף אחד – וגם זה בסדר.

נמשיך.

לא די במציאת העמודה הקורלטיבית המקסימלית בלבד, נצטרך גם להגדיר איזשהו סף (threshold) שמעליו המאפיינים ייחשבו בכלל כקורלטיביים. לדוגמה אם נמצא שהקורלציה המקסימלית בין שני מאפיינים כלשהם היתה 0.2 זה לא יכול לעזור לנו לגלות חריגה שכן מראש שני המאפיינים הללו בלתי תלויים. לעומת זאת על קורלציה של 0.9 אפשר לסמוך הרבה יותר. הסף הזה הוא פרמטר של האלגוריתם וככזה הוא ישפיע מאד על הדיוק שלו.

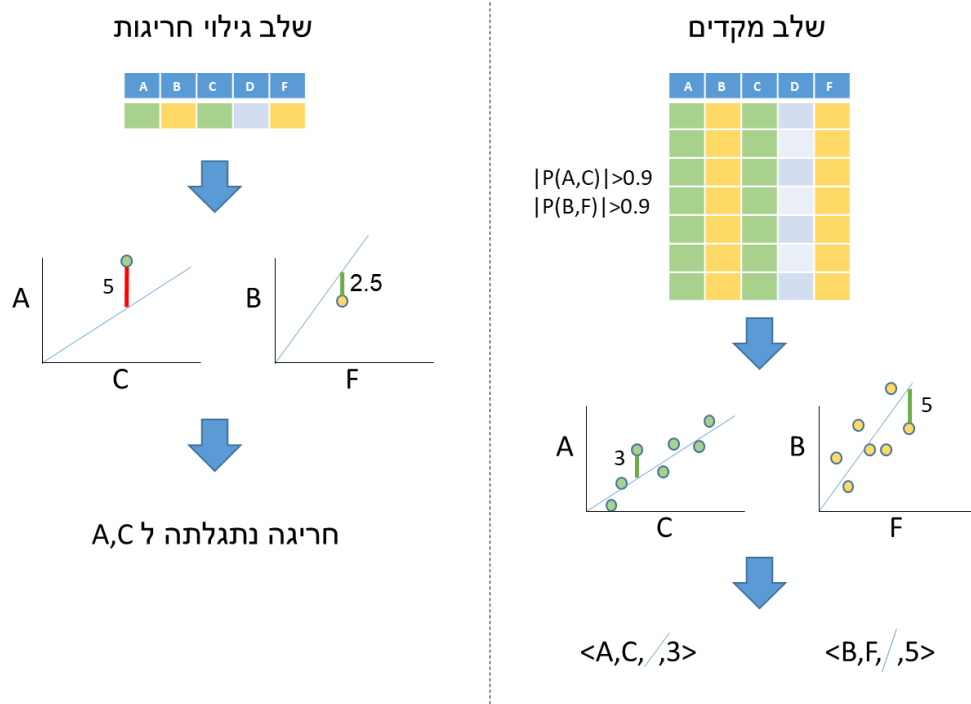
לכל זוג מאפיינים שימצאו קורלטיביים דיים (קורלציה ישירה או הפוכה) נלמד את קו הרגרסיה שהנתונים שלהם מייצרים.

כעת נעבור על הטיסה התקינה שוב, ולכל זוג מאפיינים, ולכל נקודה דו-ממדית שלהם ב data נמדוד את ההיסט המקסימאלי שראינו ביחס לקו הרגרסיה שלהם. מכיוון שזו טיסה תקינה, אז נניח שהמרחקים שראינו מותרים וצפויים. אם בטיסת המבחן נראה נק' שמייצרת היסט גדול יותר, נתריע על כך כחריגה ואף נוכל לומר מיהם המאפיינים המעורבים בחריגה.

הערה: כדאי להכפיל את ההיסט הנורמלי המקסימאלי שנמצא ב 1.1 כדי לא להחריג נק' גבוליות וליצור אזעקות שווא.

בשלב גילוי החריגות נקבל את הקלט תוך כדי טיסה – כלומר שורה אחר שורה. מכל שורה שכזו ניצור אוסף של נקודות דו-ממדיות ע"פ המאפיינים שלמדנו שהם קורלטיביים בשלב המקדים. לכל נק' דו-ממדית שכזו נמדוד את המרחק שלה מקו הרגרסיה שלמדנו עבור המאפיינים שיצרו אותה. אם המרחק הזה גדול דיו מהמרחק המקסימאלי שראינו עבור המאפיינים האלה נתריע על חריגה ונכלול בדיווח את המאפיינים המעורבים.

התרשים הבא מתאר אילוסטרציה של גלאי החריגות הפשוט שלנו:



מימוש:

כרגע אנו יודעים על מימוש אחד לגלאי החריגות שלנו. אולם, יתכן ובעתיד נרצה לשחק עם הקוד ולבחון מימושים שונים לאלגוריתם הזה או אפילו לנסות אלגוריתמים שונים לחלוטין לגילוי חריגות.

המשמעות היא שלא נרצה להטמיע את האלגוריתם בצורה קבועה בפרויקט, אלא לשמור על היכולת להחליף ולשבץ אלגוריתמים שונים בלי שזה ישפיע על שאר החלקים של הפרויקט.

במילים אחרות, עלינו לשמור על עקרון חשוב שנקרא **Open / Closed** – שמשמעותו העיצוב שלנו צריך להיות **פתוח להרחבה** אך **סגור לשינויים**. נבין מתוך דוגמה. אם נממש את האלגוריתם לעיל במחלקה כלשהי, מן הסתם יהיו לה מתודות ספציפיות שמתאימות לדרישות האלגוריתם הזה. כשנרצה לנסות כל מיני וריאציות למימוש נצטרך לשנות קוד קיים. אם נרצה לכתוב אלגוריתם אחר או לייבא אחד, אז המתודות שלו יהיו שונות, והטמעתו בפרויקט לא תהיה פשוטה.

לעומת זאת, אילו נתאמץ להגדיר ממשק שמתאר את הפונקציונאליות הכללית של כל גלאי חריגות, ובכל שאר הפרויקט יעבדו עם ממשק זה כטיפוס, אז נוכל להחליף אפילו בזמן ריצה מימושים שונים לממשק מבלי שזה ישפיע על שאר הקוד. כלומר העיצוב סגור לשינויים. ואם נרצה להוסיף עוד מימוש פשוט נוסיף מחלקה שמממשת את הממשק וכך היא תתחבר לפרויקט, או במילים אחרות, פתוח להרחבה.

נגדיר את הממשק שלנו ואת התלויות שלו:

```
public class TimeSeries{
    //...
}

public class AnomalyReport {
    public final String description;
    public final long timeStep;
    public AnomalyReport(String description, long timeStep){
        this.description=description;
        this.timeStep=timeStep;
    }
}

public interface TimeSeriesAnomalyDetector {
    void learnNormal(TimeSeries ts);
    List<AnomalyReport> detect(TimeSeries ts);
}
```

הגדרנו את הממשק TimeSeriesAnomalyDector.

תחילה אנו נותנים את האפשרות לכל גלאי חריגות ללמוד ולמדל data שנחשב נורמלי – במתודה learnNormal. מתודה זו תקבל אובייקט מסוג TimeSeries – טיפוס שאותו עליכם להגדיר בכוחות עצמכם.

בסופו של דבר TimeSeries צריך להחזיק טבלה כמתואר לעיל. רצוי שיהיו לו מתודות שיעשו את החיים שלנו קלים יותר, כגון הזנה או החזרה של רשימת שמות המאפיינים, הוספת שורה, איזה ערך יש למאפיין j בזמן i, טעינה מתוך קובץ csv וכדומה.

אם גלאי החריגות שלנו מבוסס על למידה מקדימה, אז באמצעות המתודה learnNormal נוכל להזין לו TimeSeries (פעם אחד או יותר) מבלי שיהיה תלוי מהיכן הגיע המידע (למשל קובץ או stream של תקשורת וכדומה).

המתודה העיקרית של גלאי החריגות היא המתודה detect. בהינתן TimeSeries (למשל סדרה המאפיינת טיסה חדשה) נרצה להחזיר רשימה של דיווחים. לכל דיווח (AnomalyReport) יש תיאור ונק' זמן. כך נוכל למדוד את הדיוק של גלאי החריגות.

כעת נוכל להגדיר את גלאי החריגות הפשוט שלנו כסוג של `TimeSeriesAnomalyDetector`.

```
public class CorrelatedFeatures {
    String feature1, feature2;
    float correlation;
    Line lin_reg;
    float threshold;
    //...
}

public class SimpleAnomalyDetector implements TimeSeriesAnomalyDetector {

    @Override
    public void learnNormal(TimeSeries ts) {
        // TODO Auto-generated method stub
    }

    @Override
    public List<AnomalyReport> detect(TimeSeries ts) {
        // TODO Auto-generated method stub
        return null;
    }

    public List<CorrelatedFeatures> getNormalModel(){
        // TODO implement
        return null;
    }
}
```

במתודה `learnNormal` עליכם לממש את השלב המקדים של האלגוריתם שתיארנו, ואילו במתודה `detect` את תהליך גילוי החריגות.

בנוסף, הגדרנו את המתודה `getNormalModel` למען הטסטביליות של המחלקה. במתודה זו תחזירו רשימה של מאפיינים קורלטיביים כמתואר בקוד. כך, נוכל לבדוק שהלמידה היתה תקינה.

הערות:

- Feature1 מהווה את עמודת הקלט (x) שבאמצעות $\text{line_reg.f}(x)$ חוזים את הערך של Feature2 (y). יש להקפיד ש Feature1 תמיד תהיה העמודה השמאלית יותר מבין 2 העמודות הקורלטיביות. לדוגמה, אם עמודות A ו C קורלטיביות, אז A תהיה feature1 ובאמצעותה נחזה את C.
- ה description ב `AnomalyReport` צריך פשוט לכלול את שמות המאפיינים מופרדים ב "-". הסדר הוא feature1 ואז feature2. לדוגמה, אם נתגלתה חריגה ביחס של עמודות A ו C אז התיאור יהיה פשוט "A-C". כמו כן שימו לב שספירת ה time-steps (שורות בקובץ) מתחילה מ 1.

במסמך ההגשות מפורט תאריך היעד להגשה, כיצד להגיש ולאו.

הבדיקה בודקת את המודל הנורמלי ואת רמת הדיוק של הגלאי. תשחקו עם הפרמטרים של סף הקורלציה או סף המרחק שתוארו לעיל כדי לקבל דיוק מיטבי.

בהצלחה!

אבן דרך 3 – שרת גילוי חריגות

כיום ישנם שרתים מוכנים שמאד נוח להתממשק אליהם. אולם, כדי שההבנה שלנו תהיה עמוקה, אנו נממש שרת פשוט בכוחות עצמנו. מאוחר יותר תוכלו להשתמש בשרתים מוכנים וכבר יהיה לכם מושג טוב כיצד הם עובדים מאחורי הקלעים.

בשלב זה נרצה שלשרת שלנו יהיה Command Line Interface (CLI). כלומר, כאשר לקוח יתחבר, יופיעו לו תפריטים טקסטואליים שהשרת שלח ובאמצעותם תתבצע האינטראקציה בין השרת ללקוח.

הלקוח יוכל להעלות לשרת קובץ csv, לעדכן פרמטרים של האלגוריתם ולקבל בחזרה דו"ח חריגות שנתגלו. בנוסף הלקוח יוכל להזין היכן התרחשו החריגות בפועל ולקבל ניתוח של דיוק האלגוריתם על ה data ששלח.

השרת שלנו יצטרך לטפל במספר לקוחות במקביל ונצטרך אף להגביל את הכמות הזו כדי שהשרת לא יקרוס כתוצאה מעומס.

ייתכן ובשלב מאוחר יותר (למשל פת"מ 2) נרצה להחליף את ה CLI ולהפוך את השרת שלנו ל Service שניתן להפעלה ישירות מקוד הלקוח לפי סטנדרטים מודרניים.

הערה חשובה: משלב זה של הפרויקט אתם מקבלים בהדרגה יותר חופש ואחריות על ההחלטות בפרויקט. המשמעות היא שאגדיר לכם מה עליכם לממש ופחות אגדיר לכם איך לממש זאת. נתחיל בכך שתקבלו את העיצוב בלבד, ועליכם לנתח ולגזור משמעויות לאימפלמנטציה, ולממש בהתאם.

משימה א' CLI

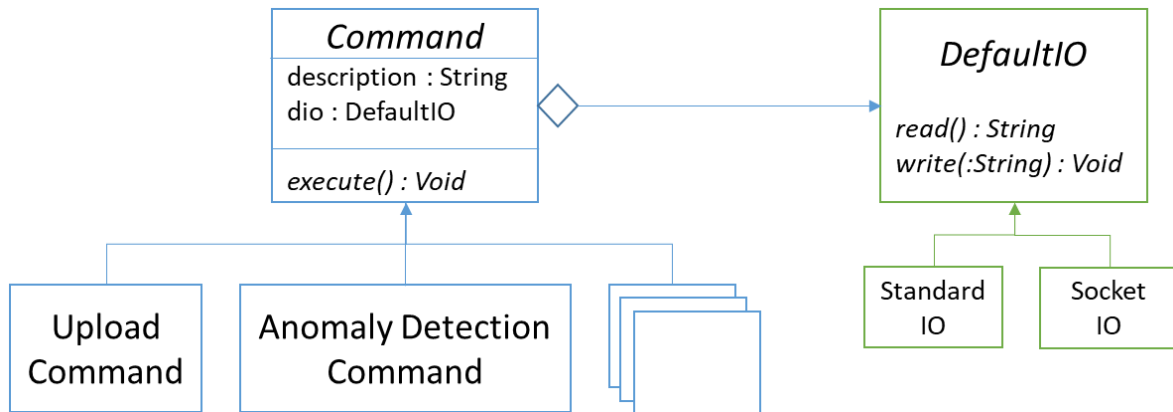
לשם מימוש ה CLI נשתמש בתבנית עיצוב שנקראת Command Pattern בה לכל פקודה במערכת שלנו יש מחלקה משלה מסוג Command. הטיפוס Command יכול להגדיר כל מה שרלוונטי לכל הפקודות במערכת שלנו, ובפרט פקודת execute אבסטרקטית עבור הפעלה. היתרונות הם:

- מכנה משותף פולימורפי לכל הפקודות
 - ניתן למשל להכניס את כל הפקודות למבנה נתונים
 - כגון מפה או מערך, ובהינתן ה key (או אינדקס) מיד לשלוף את הפקודה ולהפעילה
 - לרכז את כל הבקשות לפקודות המגיעות במקביל בתור \ תור עדיפויות
- פתוח להרחבה – ניתן להוסיף עוד מחלקות Command ע"פ הצורך \ לרשת פקודות קיימות
- ליצור הרכבות עם תבניות עיצוב אחרות, למשל עם Composite. אם למשל המחלקה MacroCommand תחזיק מערך פולימורפי של Commands אז כשנקרא ל execute שלה היא בתורה תקרא ל execute של כל פקודה במערך, שבתורן יכולות להיות פקודות רגילות (עלים) או גם MacroCommands.

אך היתרון העיצובי הוא החשוב ביותר – והוא ההפרדה בין יוזם הפקודה (invoker) לבין מי שהולך להיות מופעל (receiver). למשל אם יש 5 דרכים שונות ליזום את אותו הדבר (נניח פעולת הדבק) אז מכולן יהיה קישור לאותו אובייקט פקודה ויצרנו מקור אחד של אמת אם נרצה לשנות בעתיד משהו.

לעוד מידע תוכלו לצפות בסרטון 13.1

נגדיר את העיצוב הבא:



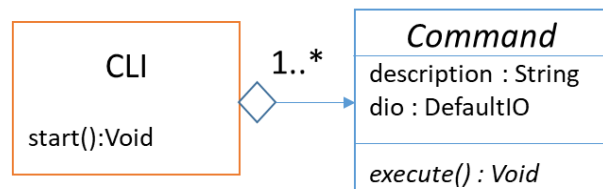
כפי שניתן לראות יש לנו מחלקה אבסטרקטית Command עם מתודה אבסטרקטית execute(). את המתודה הזו יצטרכו לממש כל היורשים – הפקודות השונות בתוכנית שלנו.

בנוסף לכל Command יש מחרוזת description. כדי ליצור תפריט למשל, נוכל להשתמש במערך של Command-ים ולעבור על כל Command ולהדפיס את ה description שלו. כאשר המשתמש יבחר אופציה i נוכל ללכת ל Command ה i במערך ולקרוא ל execute שלה. הפקודה הזו בתורה תמשיך את האינטראקציה עם המשתמש לפי הצורך. היא אפילו יכולה בתורה להפעיל פקודות אחרות.

אולם, לא תמיד נרצה להדפיס למסך או לקרוא מהמקלדת... בהקשר שלנו נרצה להשתמש בעיצוב הזה בתוך שרת, כאשר את הקלט והפלט אנו מבצעים דרך socket-ים של תקשורת. לשם כך הגדרנו את הטיפוס המופשט DefaultIO שהיורשים שלו יצטרכו לממש בדרכם את המתודות הקריאה והכתיבה. כך נוכל להזין בזמן ריצה ל Command מימושים שונים של DefaultIO. בדומה לעיל, אם נרצה קלט-פלט סטנדרטי אז נזין לו את StandardIO ואילו אם נרצה באמצעי תקשורת אז נזין SocketIO. נשים לב שזה גם פתוח להרחבה, כי אם נרצה למשל לקרוא ולכתוב לקבצים אז נוכל להוסיף מימוש ל DefaultIO וכל ה Command-ים לא יצטרכו להשתנות ויעבדו אותו הדבר.

זכרו! תרשים מחלקות ב UML אינו מהווה תחליף לקוד; הוא מכיל רק את מה שרלוונטי להבנת העיצוב. עליכם לגזור משמעויות נוספות שקשורות למימוש.

קעת צרו את המחלקה CLI עם המתודה void start().



טיפ: תחילה תעבדו עם standardIO. זה יהיה הרבה יותר נוח ונכון לדיבאג. כשהכל יעבוד ב CLI, תוכלו לממש את socketIO ואז ה CLI יזין אותו ל Command, ולראות שהכל עובד גם דרך ערוצי התקשורת.

כאשר מתודה זו תופעל יודפס ללקוח התפריט הבא:

```
Welcome to the Anomaly Detection Server.
Please choose an option:
1. upload a time series csv file
2. algorithm settings
3. detect anomalies
4. display results
5. upload anomalies and analyze results
6. exit
```

הערה: עבור כל הדפסה בפרויקט, ירידת שורה היא '\n' בלבד, ולא '\r\n' כמו במערכת ההפעלה חלונות.

על כל אופציה שהלקוח בוחר יופעל אובייקט Command מתאים שימשיך ע"פ הצורך את האינטראקציה עם הלקוח.

זכרו שהאינטראקציה צריכה להיעשות ע"י אובייקט DeafultIO כדי שמאוחר יותר תוכלו להחליף אותו עם קלט-פלט מבוסס תקשורת. כשתעשו את זה, מן הסתם תצטרכו לבנות גם צד לקוח, לעת עתה ניתן להניח לזה.

אם הלקוח הקליד 1 ו enter, תינתן האפשרות ללקוח להקליד נתיב לקובץ csv לוקאלי אצלו במחשב, ולאחר לחיצה על enter הלקוח ישלח את הקובץ לשרת. בסיום ההעלאה השרת יכתוב חזרה ללקוח הודעת "upload complete".

זה צריך להיראות כך:

<pre>Please upload your local train CSV file. C:\data\flightgear\flight1.csv Upload complete.</pre>	<p>השרת מדפיס הלקוח מקליד שם קובץ השרת מדפיס</p>
---	--

תהליך זה יחזור על עצמו פעמיים, כאשר בפעם הראשונה מקבלים קובץ עבור אימון גלאי החריגות ובפעם השנייה קובץ עבור הבחינה שלו. בהתאמה, בפעם הראשונה תופיע המילה train ובשנייה test.

עבור צד השרת נגדיר את הפרוטוקול כך:

לאחר בחירה של 1 ע"י הלקוח, השרת כותב ללקוח "Please type ... CSV file" ואז יצפה לקרוא מהלקוח time series – כלומר, כותרות מופרדות בפסיק בשורה הראשונה, ואז בכל שורה לאחר מכן ערכים המופרדים בפסיק, בדיוק כפי שנראה קובץ ה csv. כאשר תופיע שורה ובה המילה "done" בלבד, ידע השרת שהסתיימה שליחת הקובץ ע"י הלקוח. השרת שומר בצד שלו את הקלט שקיבל מהלקוח כקובץ anomalyTrain.csv \ anomalyTest.csv בהתאמה לקובץ שהועלה ע"י הלקוח. השרת כותב ללקוח "Upload complete".

עבור צד הלקוח נגדיר את הפרוטוקול בהתאמה:

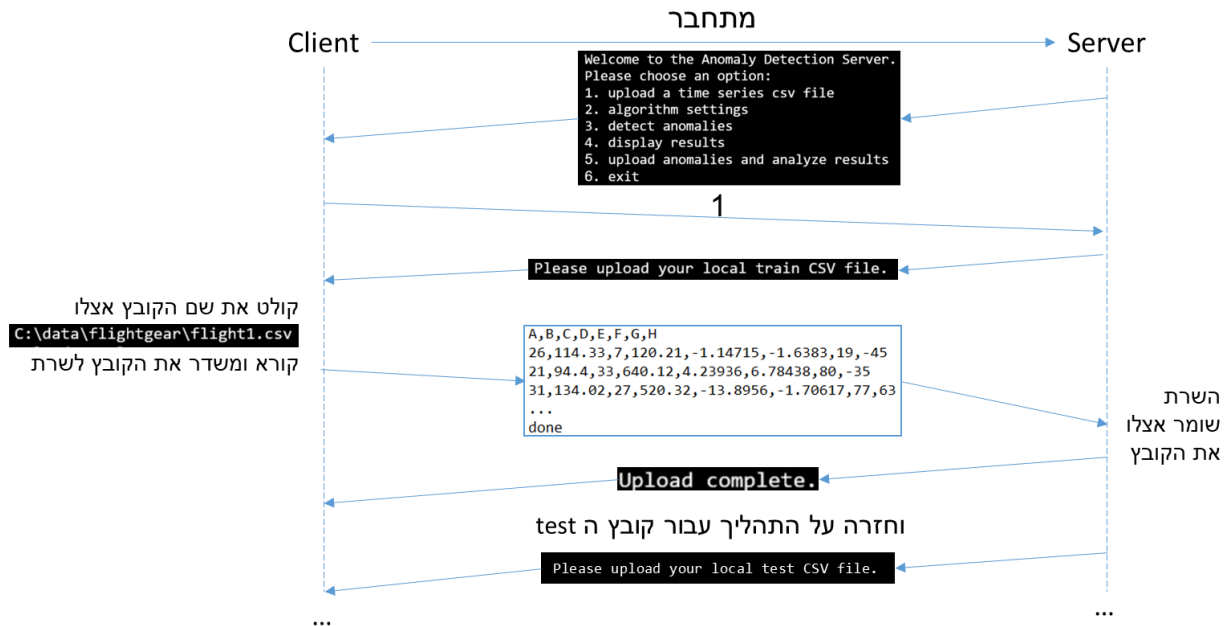
הלקוח בוחר 1 (ו enter), ומקבל מהשרת את השורה "Please ..CSV file". אז הלקוח קולט מהמשתמש (האדם שנמצא בצד הלקוח) נתיב ושם קובץ מלא עבור קובץ csv. קורא את קובץ ה csv ועל כל שורה שקרא הלקוח מהקובץ הוא שולח אותה לשרת. לאחר סיום קריאת הקובץ הלקוח כותב לשרת שורה ובה המילה "done" בלבד. הלקוח יקרא מהשרת את השורה "Upload complete".

למען הסר ספק, הקלט של נתיב הקובץ בצד הלקוח לא עובר בתקשורת אל השרת, אלא רק תוכן הקובץ.

טיפ: זה יכול להיות רעיון טוב להוסיף ל DefaultIO אפשרות להעלות \ להוריד קבצים (כמתודות דיפולטביות התלויות במתודות הבסיסיות שהוגדרו).

לאחר סיום העלאת הקובץ השני יוצג שוב התפריט הראשי.

תרשים התהליך עד כה:



נשים לב שמנקודת מבטו של השרת הוא רואה:

1

A,B,C,D,E,F,G,H

26,114.33,7,120.21,-1.14715,-1.6383,19,-45

31,134.02,27,520.32,-13.8956,-1.70617,77,63

...

done

כלומר הוא אינו רואה את שם הקובץ שהלקוח הקליד לוקאלית אצלו.

הלקוח שלח 1 לשרת, קרא מהשרת מחרוזת. Please upload... file, קלט מהמשתמש אצלו את שם הקובץ, והתחיל לשלוח את תוכן הקובץ לשרת ולבסוף שלח את המחרוזת done.

אז נא לא להתבלבל, קלט הבדיקה האוטומטית מדמה בדיוק את זה, כלומר כשהשרת נבדק נשלח לו מה שהוא מצפה לראות כמו בדוגמה לעיל – כלומר ללא קלט שם הקובץ.

לעומת זאת המימוש של צד הלקוח (שאותו אינכם נדרשים לממש) ידרוש את קלט שם הקובץ, קריאתו ושליחתו.

אם הלקוח בחר 2, השרת יציג לו את סף הקורלציה והאפשרות להחליפו באופן הבא:

```
The current correlation threshold is 0.9
Type a new threshold
```

אם הוא בחר ערך תקין ולחץ enter אז הסף ישתנה. אם הוא לא בחר ערך בין 0 ל 1 אז יש לכתוב לו את ההודעה הבאה:

"please choose a value between 0 and 1."

ולחזור ולהציג שוב את האפשרות לשנות את הסף.

לאחר בחירה של סף תקין יש לחזור לתפריט הראשי.

אם הלקוח בחר 3, השרת יריץ את האלגוריתם על קובץ ה CSV שהועלה קודם לכן. בסוף הריצה השרת יכתוב "anomaly detection complete." ונחזור לתפריט הראשי.

אם הלקוח בחר 4, השרת ידפיס את דיווחי החריגות. לכל דיווח ההדפסה תהיה כך: ה time step, טאב, ה description ואז ירידת שורה. לבסוף יודפס "Done". לאחר מכן יש לחזור לתפריט הראשי. לדוגמה:

```
73      A-B
75      C-D
8       E-F
Done.
```

אם הלקוח בחר 5, הוא יקליד שם מלא של קובץ חריגות. לאחר enter הלקוח יעלה את הקובץ לשרת, השרת ינתח את תוצאות האלגוריתם ויציג אותן ללקוח.

בדומה לפרוטוקול ההעלאה הקודם, המשתמש בצד הלקוח מקליד את שם הקובץ ואז הלקוח שולח לשרת שורה אחר שורה מהקובץ כאשר בסוף הוא כותב לשרת שורה ובה המילה "done" בלבד.

קובץ החריגות הוא קובץ טקסט בו בכל שורה יופיעו הזמנים שבהם אכן הופיעו חריגות (לפי סדר כרונולוגי). הפורמט הוא time step של תחילת החריגה, פסיק, time step של סיום החריגה. לדוגמה:

```
122,150
180,185
1001,1020
done
```

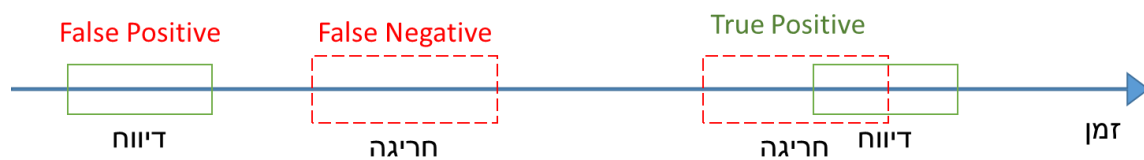
חריגה כוללת את זמן סיום החריגה, לדוגמה כל השורות 122 עד 150 כולל 150 מהוות שורות שבהן היתה חריגה. השרת ישווה את זמני הדיווחים של גלאי החריגות לזמנים הללו (השרת לא חייב לשמור את קובץ החריגות כקובץ בצד השרת).

תחילה, עלינו לאחד את דיווחי הגלאי שיש להם את אותו ה description ורציפות ב timestep. לדוגמה אם גלאי החריגות דיווח 15 דיווחים רציפים מ x time step ועד x+15 time step ולכולם אותו התיאור, אז נחשיב את זה כדיווח אחד רציף שהתחיל ב x והסתיים ב x+15 (כולל). את כמות החריגות (בקובץ החריגות) נסמן כ P (עבור positive). בדוגמה לעיל P=3. את כמות ה time steps שבהם לא היתה כל חריגה (בקובץ החריגות) נסמן כ N (עבור negative). בדוגמה לעיל, אם n זה מס' השורות בקלט (לא כולל הכותרת) אז $N=n-29-6-20$

כעת נוכל למדוד את האלגוריתם:

- כל דיווח מחוץ לאחת ממסגרות הזמן בהן היו חריגות ייחשב כ false positive. נסמן כ FP.
- כל מסגרת זמן שבה היתה חריגה, ויש לה חיתוך גדול מ 0 עם זמן של דיווח חריגה ייחשב כ true positive. נסמן כ TP.
- ישנם מדדים נוספים שאין לנו בהם צורך כרגע, אך כדי להשלים את התמונה:
 - כל מסגרת זמן שבה היתה חריגה, ואין לי חיתוך עם אף דיווח – תיחשב כ false negative. נסמן כ FN.
 - כל זמן בו לא היתה חריגה ולא היה דיווח ייחשב כ true negative. נסמן כ TN.

המחשה ויזואלית:



נרצה לדווח שני מדדים:

כמה מתוך כלל הדיווחים היו נכונים – TP / P True positive rate

יחס אזעקות השווא – FP / N False alarm rate

האינטראקציה בין השרת ללקוח תיראה כך:

<pre>Please upload your local anomalies file. C:\data\flightgear\flight1_anomalies.txt Upload complete. Analyzing... True Positive Rate: 0.753 False Positive Rate: 0.12</pre>	<p>השרת מדפיס הלוקח מקליד השרת מדפיס</p>
--	--

את התוצאות יש להדפיס עם חיתוך של 3 ספרות אחרי הנקודה.

לאחר מכן נחזור לתפריט הראשי.

אם הלקוח בחר 6, תסתיים האינטראקציה בין השרת ללקוח.

כאמור, מומלץ לבדוק את כל ה CLI לוקאלית עם standardIO לפני שממשיכים למשימות הבאות.

משימה ב' צד שרת

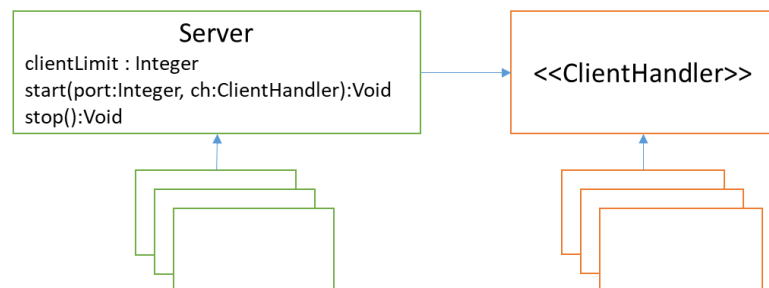
בדומה לאלמנטים הקודמים בפרויקט, גם כאן נרצה לשמור כלליות הקוד כך שיתאפשר reuse בעת הצורך. כשאנו כותבים צד שרת יש חלקים שחוזרים על עצמם מפרויקט לפרויקט ויש כמובן דברים שמשתנים. זה חשוב להפריד בין אלו כדי שנוכל לעשות reuse לקוד שאינו משתנה. לשם כך אסור לו להיות תלוי בקוד שכן צפוי להשתנות. צריך לנתק בין חלקים אלו – כלומר לעשות decoupling.

בצד השרת נרצה להפריד בין המנגנון שמפעיל את השרת, דהיינו מאזין על port כלשהו וממתין ללקוח, פותח ת'רד כדי להתנהל איתו במקביל, מגביל את כמות הת'רדים וכדומה, לבין השיחה עם הלקוח שתשתנה כמובן מפרויקט לפרויקט.

נשתמש בתבנית עיצוב בשם Bridge pattern כדי להפריד בין מימושים שונים של השרת לבין מימושים שונים לשיחה עם הלקוח.

צפו בהרצאה 9, וכן בסרטון 11.3 אודות התבנית.

כעת אתם מקבלים יותר חופש. הביטו בתרשים הכללי הבא, והפכו אותו לעיצוב שמתאים למימוש צד השרת.



בצד אחד של הגשר יש לנו את המחלקה Server. בצד השני יש לנו ממשק בשם ClientHandler שצריך להגדיר את הפונקציות של טיפול בלקוח שהתחבר. במתודה start() של Server נקבל מופע מהסוג של ClientHandler. כך, ניתן להרחיב את Server בצד אחד של הגשר בלי להיות תלויים בהרחבות השונות של ClientHandler. הלכה למעשה, נוכל לממש את Server פעם אחת, ובכל פרויקט להחליף לו ClientHandler.

בנוסף, הערך של clientLimit יגביל את כמות הלקוחות שנטפל בהם במקביל. המתודה start תפעל את השרת על port כלשהו, ותפעיל את ה client handler שקבלה כדי להתכתב עם הלקוח שהתחבר. המתודה stop() תסגור את השרת בצורה מסודרת. כלומר, לא יתקבלו לקוחות חדשים. כל הלקוחות שהתחברו יסיימו את הטיפול, ולאחר מכן כל המשאבים (ת'רדים, socket-ים) ייסגרו.

- עליכם להגדיר בכוחות עצמכם מה צריך להיות בממשק של Client Handler.
- עליכם לממש את Client Handler במחלקה בשם AnomalyDetectionHandler
- עליכם לשבץ בצורה נכונה במחלקה זו את השימוש ב CLI, ה Command-ים שלו, ובפרט את ה DefaultIO שלו שצריך עתה להתכתב באמצעות ה Socket-ים שהשרת פתח.
- השרת שלכם צריך לטפל בלקוחות אחד אחרי השני ולא במקביל. אולם,
 - המתודה start כן צריכה לרוץ ברקע כפי שראינו בשיעור
 - בפת"מ 2 תוסיפו מימוש ל Server שמטפל בכמה לקוחות במקביל.
- הפעם, כאשר הלקוח בחר exit השרת יכתוב לו את המילה "bye" בשורה משלה.

במסמך ההגשות מפורט תאריך היעד להגשה, כיצד להגיש ולאו.

הבדיקה בודקת את ה CLI בנפרד, ולאחר מכן באמצעות לקוח שיתחבר לשרת שלכם.

בהצלחה!

ד"ר אליהו חלסצ'י

פרויקט פיתוח תוכנה מתקדם 1, סמסטר א' תשפ"ב

פרומו לאבן דרך 4 בסמסטר ב: ☺

