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 ובמסד נתונים
- הזרמת נתונים (קבצים ותקשורת)
- הטמעה של אלגוריתמים מבוסס data בתוך המערכת שניצור
- תכנות מקבילי באמצעות ת'רדים
- תכנות מוכוון אירועים, אפליקציית desktop עם GUI
- תכנות מוכוון אירועים, אפליקציית Web בסגנון REST
- אפליקציית מובייל (אנדרואיד)

בכל סמסטר תגישו 3 אבני דרך:

תכנות מתקדם 1

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

תכנות מתקדם 2

4. אפליקציית desktop לתחקור טיסה וגילוי חריגות
5. אפליקציית רשת (SaaS) לגילוי חריגות
6. אפליקציית מובייל – שלט רחוק למטוס

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" ותשתפשפו 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 anomaly_detection_util.cpp עבור הקובץ anomaly_detection_util.h הבא:

```
// returns the variance of X and Y
float var(float* x, int size);

// returns the covariance of X and Y
float cov(float* x, float* y, int size);

// returns the Pearson correlation coefficient of X and Y
float pearson(float* x, float* y, int size);

class Line{
public:
	const float a,b;
	Line(float a, float b):a(a),b(b){}
	float f(float x){
		return a*x+b;
	}
};

class Point{
public:
	const float x,y;
	Point(float x, float y):x(x),y(x){}
};

// performs a linear regression and return    s the line equation
Line linear_reg(Point** points, int size);

// returns the deviation between point p and the line equation of the points
float dev(Point p,Point** points, int size);

// returns the deviation between point p and the line
float dev(Point p,Line l);
```

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:

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

כאשר

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

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

$$cov(X,Y) = E(XY) - E(X)E(Y) = E\big((X-E(X))(Y-E(Y))\big)$$

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 ביו המשתנים.

יש לממש את Pearson כך:

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

כאשר σ היא סטיית התקן (השורש של variance)

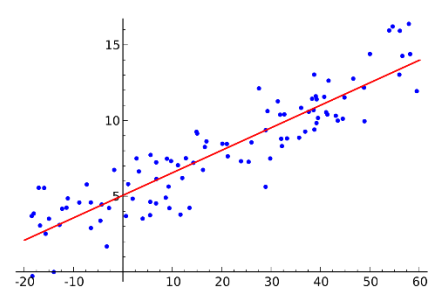
כעת נתונה המחלקה Line המייצגת משוואת ישר ע"י $Y = a\cdot X + b$ והמחלקה Point המייצגת נקודה דו-ממדית.

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

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

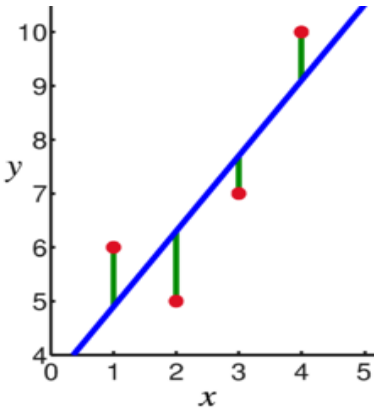
כך מתקבלת משוואת ישר $Y = a\cdot X + b$ המתארת את הנק' אם אכן יש ביניהן קורלציה לינארית.

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 את הנק' המתארת היכן ציפינו 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 (50%)

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

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:

א. טיסות מתחילות כתקינות, ולאחר נק' זמן כלשהי עלולה להתפתח תקלה
ב. נתוני הטיסה שנדגמים מכילים לא מעט מאפיינים קורלטיביים זה לזה
ג. נצפה ממאפיינם 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 $f_{j\neq i}$. כך לכל מאפיין f_i נחזיר את המאפיין f_j שעבורו קבלנו את תוצאת ה Pearson בערך מוחלט הגבוהה ביותר.

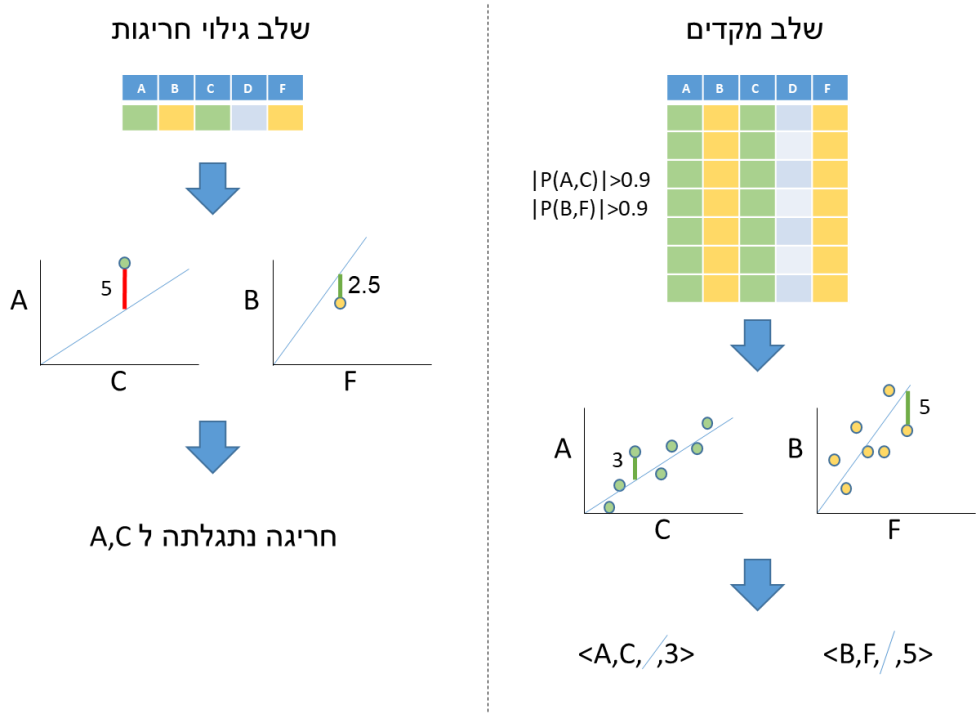
אך לא די בכך, נצטרך גם להגדיר איזשהו סף (threshold) שמעליו המאפיינים ייחשבו בכלל כקורלטיביים. לדוגמה אם נמצא שהקורלציה המקסימלית בין שני מאפיינים כלשהם היתה 0.2 זה לא יכול לעזור לנו לגלות חריגה שכן מראש שני המאפיינים הללו בלתי תלויים. לעומת זאת על קורלציה של 0.9 אפשר לסמוך הרבה יותר. הסף הזה הוא פרמטר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

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 בקובץ AnomalyDetector.h:

```
class TimeSeries{
	//...
};

class AnomalyReport{
public:
	const string description;
	const long timeStep;
	AnomalyReport(string description, long timeStep) :
		description(description),timeStep(timeStep){}
};

class TimeSeriesAnomalyDetector {
public:
	virtual void learnNormal(const TimeSeries& ts)=0;
	virtual vector<AnomalyReport> detect(const TimeSeries& ts)=0;
	virtual ~TimeSeriesAnomalyDetector(){}
};
```

הגדרנו את המחלקה TimeSeriesAnomalyDector כממשק – ישנן רק מתודות וכולן pure virtual.

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. נעשה זאת בקובץ SimpleAnomalyDetector.h.

```
struct correlatedFeatures{
	string feature1,feature2; // names of the correlated features
	float corrlation;
	Line lin_reg;
	float threshold;
};

class SimpleAnomalyDetector:public TimeSeriesAnomalyDetector{
public:
	SimpleAnomalyDetector();
	virtual ~SimpleAnomalyDetector();

	virtual void learnNormal(const TimeSeries& ts);
	virtual vector<AnomalyReport> detect(const TimeSeries& ts);
	vector<correlatedFeatures> getNormalModel();
};
```

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:

א. במחלקה Line עליכם להסיר את ה const מ a,b ולהוסיף בנאי דיפולטיבי שמאתחל אותם ל 0. זאת כדי ש line_reg לעיל יוכל להיות אובייקט בתוך correlatedFeatures. אחרת, הוא יצטרך להיות מצביע, וזה יגרור הרבה קוד מיותר: ב correlatedFeatures בנאי דיפולטיבי, copy CTOR, ו DTOR, וכל העתקה של Line תצטרך להיות עמוקה במקום דיפולטיבית.

ב. Feature1 מהווה את הקלט (x) שבאמצעות line_reg.f(x) חוזים את הערך של Feature2 (y). יש להקפיד ש Feature1 תמיד תהיה העמודה השמאלית יותר מבין 2 העמודות הקורלטיביות. לדוגמה ב mainTrain, אם עמודות A ו C קורלטיביות, אז A תהיה feature1 ובאמצעותה נחזה את C.

ג. 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.

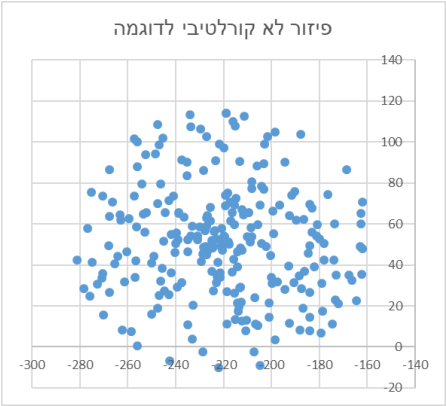
בעיית המעגל המינימאלי (50%)

עד כה המטלה היתה מאד טכנית מתוך מטרה לתרגל אתכם בתכנות ב *C++*. כתבתם אלגוריתם פשוט מבוסס *data* לגילוי חריגות ובפרט תרגלתם קבצים, חישובים, ו *STL*. בנוסף, דנו בעקרון חשוב שנקרא *open /closed*.

כעת הגיע הזמן גם לחשוב אלגוריתמית.

בגלאי הפשוט הנחנו שקיימות קורלציות ליניאריות בין משתנים, ובאמצעות רגרסיה ליניארית בחרנו לחזות את הערך הצפוי. כאשר הערך הנצפה היה שונה מהצפוי הכרזנו על חריגה.

אולם, לא בכל *domain* נוכל להניח הנחות אלו. למשל, כמו בתרשימים הבא.



במקרים כאלה, נרצה למצוא בצורה יעילה את המעגל בעל הרדיוס המינימאלי שמכיל את כל הנקודות (הנחשבות תקינות). כלומר למצוא את מרכז המעגל ואת הרדיוס שלו.

כך, בהינתן נק' חדשה נוכל למדוד את המרחק שלה ממרכז המעגל, ואם הוא גדול מהרדיוס שלו אז נכריז על חריגה.

זו כבר בעיה אלגוריתמית של מדעי המחשב (מתחום מגניב שנקרא גיאומטריה חישובית).

קחו בחשבון שהפיזורים יכולים להיות שונים ומגוונים ושמרכז המעגל הוא בד"כ אינו הממוצע של כל הנקודות, או אמצע שתי הנקודות הרחוקות ביותר...

בקובץ *minCircle.h* עליכם לממש אלגוריתם שהמצאתם או מצאתם בספרות למציאת המעגל המינימאלי שמכיל את כל הנקודות. למרות שניתן להרחיב את הבעיה ל *N* ממדים, נתמקד במרחב אוקלידי דו-ממדי.

משימה זו היא לזוגות (או ליחידים).

במסמך ההגשות מפורט תאריך היעד להגשה, כיצד להגיש ולאן.

עבור הגלאי הפשוט: 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.

עבור בעיית המעגל המינימלי: הבדיקה במוד האימון תכלול פיזור פשוט, אולם הבדיקה במוד ההגשה תכלול מספר פיזורים מגוונים. בנוסף האלגוריתם שלכם יצטרך לעמוד באילוצי זמן.

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.

בשלב מאוחר יותר (ת. מתקדם 2) נרצה להחליף את ה CLI ולהפוך את השרת שלנו ל Service שניתן להפעלה ישרות 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

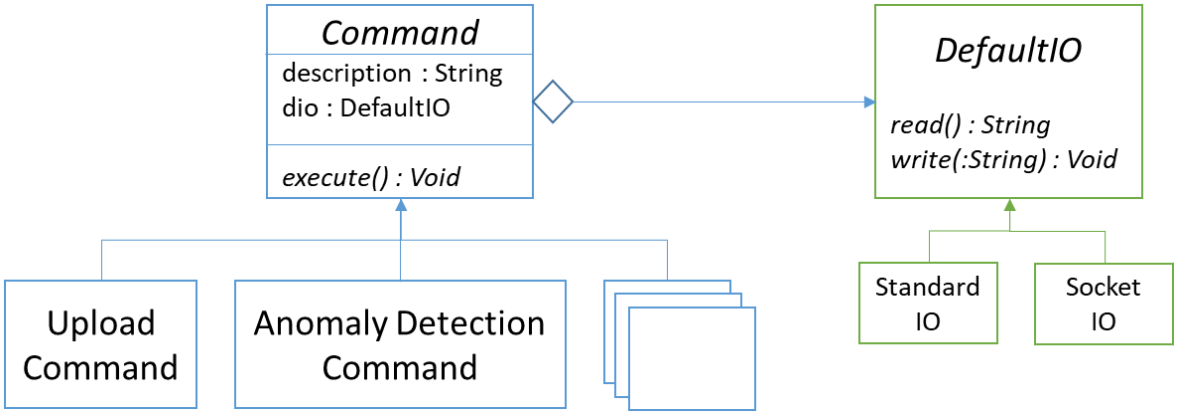
לשם מימוש ה CLI נשתמש בתבנית עיצוב שנקראת Command Pattern בה לכל פקודה במערכת שלנו יש מחלקה משלה מסוג Command. המחלקה Command יכולה להגדיר כל מה שרלוונטי לכל הפקודות במערכת שלנו, ובפרט פקודת execute אבסטרקטית עבור הפעלה. היתרונות הם:

- 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 14.1.

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.

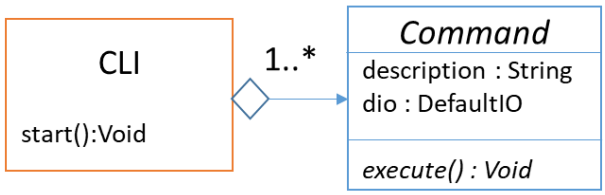
בנוסף לכל Command יש מחרוזת description. כדי ליצור תפריט למשל, נוכל להשתמש במערך של Command** ולעבור על כל *Command ולהדפיס את ה description שלו. כאשר המשתמש יבחר אופציה i נוכל ללכת ל Command ה i במערך ולקרוא ל execute שלה. הפקודה הזו בתורה תמשיך את האינטראקציה עם המשתמש לפי הצורך. היא אפילו יכולה בתורה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

אחד האתגרים למשל, הוא כיצד להעביר מידע בין אובייקטי ה Command בלי להשתמש במשתנים גלובאליים או סטטיים. עליכם לחשוב בכוחות עצמכם על פתרון מונחה עצמים לכך.

כעת 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
```

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

זכרו שהאינטראקציה צריכה להיעשות ע"י אובייקט ה 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:

Please upload your local train CSV file.	השרת מדפיס
C:\data\flightgear\flight1.csv	הלקוח מקליד שם קובץ
Upload complete.	השרת מדפיס

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".

טיפ: זה יכול להיות רעיון טוב להוסיף ל DefaultIO אפשרות להעלות \ להוריד קבצים.

לאחר סיום העלאת הקובץ השני יוצג שוב התפריט הראשי.

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

```
The current correlation threshold is 0.9
```

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."

ולאחר enter לחזור להציג שוב את האפשרות לעיל לשנות את הסף.

לאחר בחירה של סף תקין יש לחזור לתפריט הראשי.

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

אלגוריתם גילוי החריגות יהיה היברידי – שילוב של הגלאי הפשוט עם אלגוריתם הרדיוס המינימאלי הפועל כך: אם סף הקורלציה גדול או שווה לסף שנקבע (למשל 0.9) אז האלגוריתם יריץ את הגלאי הפשוט (רגרסיה). לעומת זאת, אם קיים מאפיין שהקורלציה המקסימאלית שלו עם מאפיין אחר גדולה מ 0.5 אך קטנה מסף הקורלציה אז לשני מאפיינים אלו ייקבע הרדיוס המינימלי שמקיף את כל נק' האימון. גם כאן יש להכפילו ב 1.1 כדי ליצור "מרווח נשימה". נק' שתימצא מחוץ למעגל תיחשב חריגה.

הערה עיצובית: עליכם לבצע reuse לקוד קיים במקום להעתיק אותו. לכן הגלאי ההיברידי **יירש** את הגלאי הפשוט. לשם כך מותר לכם לעשות שינויים מינוריים ככל הניתן במחלקה של הגלאי הפשוט. בפרט, מותר להוסיף (אך לא לשנות) שדות ל correlatedFeatures. **טיפ:** היעזרו בפונקציות עזר וירטואליות וחלוקה נכונה יותר ל single responsibility.

הערות טכניות:

א. יש למחוק את כפילות ההגדרה של המחלקה Point מ minCircle.h ובמקום זאת לבצע שם include ל anomaly_detection_util.h. כמו כן, יש למחוק את ה const מ x,y ב Point.
ב. את המימוש של הפונקציות של minCircle.h יש לממש מעתה ב minCircle.cpp

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.

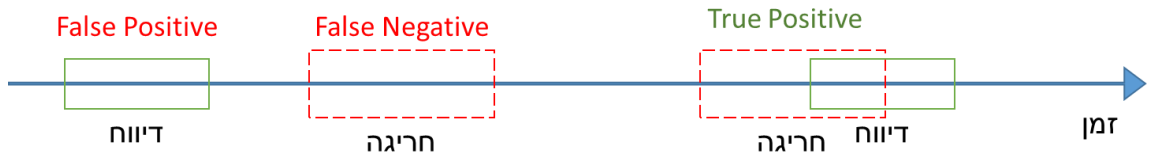
השרת ישווה את זמני הדיווחים של גלאי החריגות לזמנים הללו (השרת לא חייב לשמור את קובץ החריגות כקובץ בצד השרת).

תחילה, עלינו לאחד את דיווחי הגלאי שיש להם את אותו ה description ורציפות ב timestep. לדוגמה אם גלאי החריגות דיווח 15 דיווחים רציפים מ time step x ועד time step x+15 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:

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

False alarm rate = FP / N – יחס אזעקות השווא

האינטראקציה בין השרת ללקוח תיראה כך:

`Please upload your local anomalies file.`
`C:\data\flightgear\flight1_anomalies.txt`
`Upload complete.`
`True Positive Rate: 0.753`
`False Positive Rate: 0.12`
	השרת מדפיס
	הלקוח מקליד
	השרת מדפיס

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

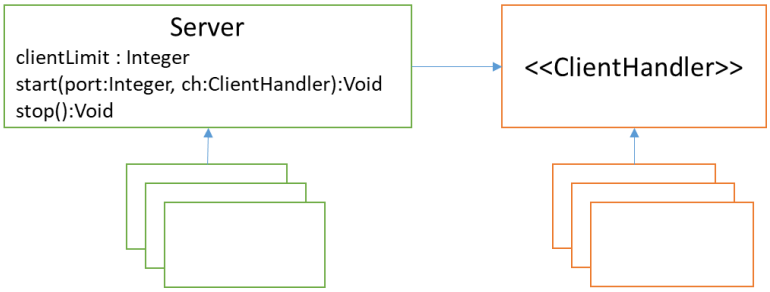
בדומה לאלמנטים הקודמים בפרויקט, גם כאן נרצה לשמור על כלליות הקוד כך שיתאפשר reuse בעת הצורך. כשאנו כותבים צד שרת יש חלקים שחוזרים על עצמם מפרויקט לפרויקט ויש כמובן דברים שמשתנים. זה חשוב להפריד בין אלו כדי שנוכל לעשות reuse לקוד שאינו משתנה. לשם כך אסור לו להיות תלוי בקוד שכן צפוי להשתנות. צריך לנתק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

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 11, וכן בסרטון 12.3 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 הפונקציונאליות של טיפול בלקוח שהתחבר. במתודה ()start של Serverנקבל 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 ClinetHandler.

המתודה start תפעיל 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, soket-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 DfaultIO שלו שצריך עתה להתכתב באמצעות ה Socket-ים שהשרת פתח.

הקלה: ניתן גם לכתוב את השרת כך שיטפל בלקוחות אחד אחרי השני במקום במקביל.

שימו לב: בדומה למפגש האחרון, המתודה start של Server צריכה לפעול בת'רד נפרד, ובמתודה stop יש לבצע join לת'רד הזה כדי לוודא שהוא אכן הסתיים ואינו רץ ברקע.

טיפ: עליכם להגדיר time out ל accept (למשל ע"י alarm), אחרת השרת יישאר פתוח כשהוא מחכה ללקוח הבא.

<u>טיפ נוסף:</u> מומלץ לבדוק את הלקוח מול שרת פשוט תחילה, למשל כזה שמטפל בלקוח אחד בלבד ושולח מחרוזות הפוכות. אחרי זה מול שרת שמטפל בכמה לקוחות במקביל. לאחר שזה עובד, עברו לבדוק את הלקוח מול השרת הראשי.

במסמך ההגשות מפורט תאריך היעד להגשה, כיצד להגיש ולאן.

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

בהצלחה!

פרומו לאבן דרך 4 בסמסטר ב: ☺