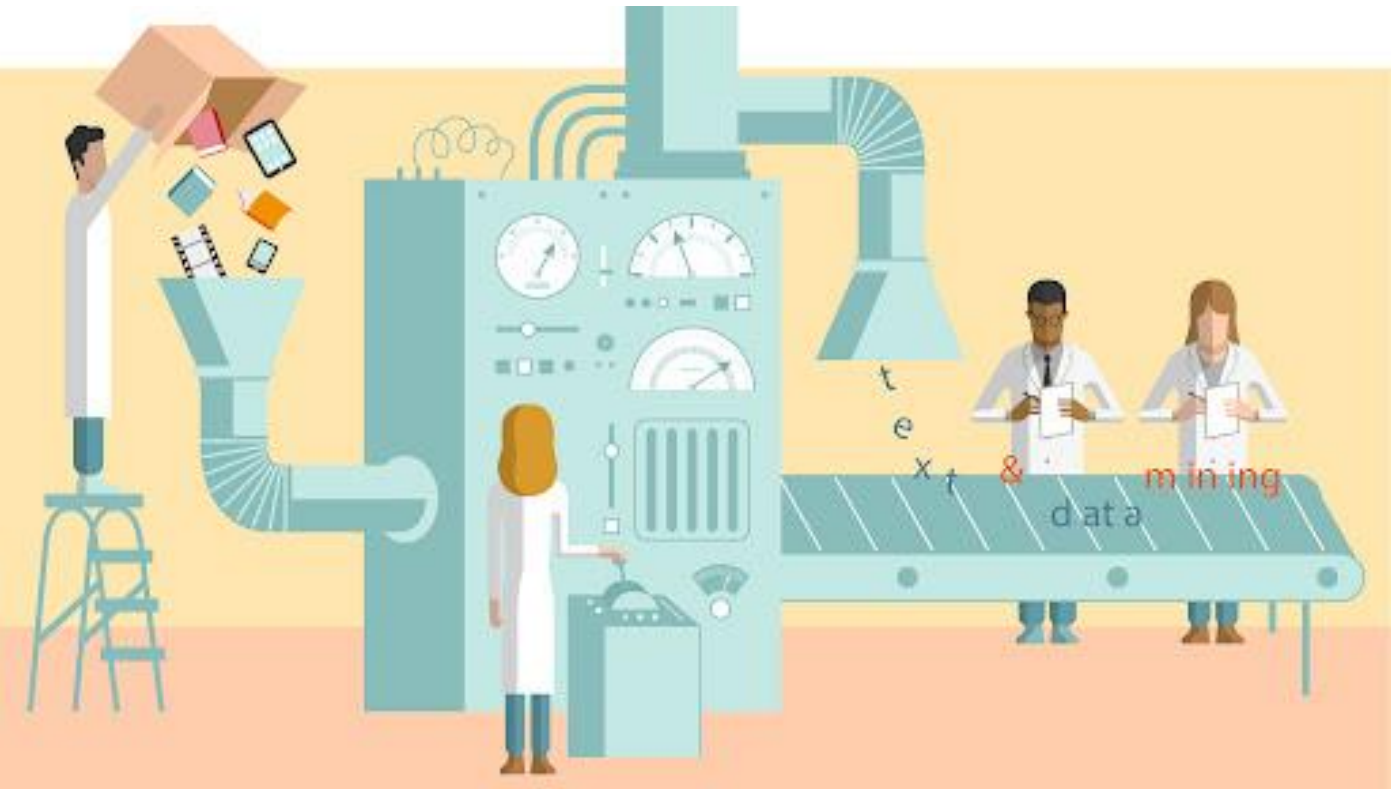
פרוייקט כריית מידע

חלק 3



מגישים:

יעל מוכתר 208606079

טום אשקורי 205702608

בחלק זה של הפרוייקט התבקשנו ליצור מודל סופי שידע לבצע חיזויים על לקוחות פוטנציאלים באופן שתמזער את מטריצת העלויות שקיבלנו ממחלקת ניהול סיכונים החביבה.

$Predicted$ / $Actual$	False	True
$False$	0	$C(T\|F) = 1$
$True$	$C(F\|T) = 15$	0

בעצם לחלקו הזה של הפרוייקט מתנקזת כל העבודה שעשינו במהלך הסמסטר, משלב העיבוד המקדים ועד של האימון ונתינת החיזויים. כמובן עכשיו יש לנו ערך מטרה שאותו אנחנו רוצים למזער והוא פונקציית המחיר:

$$loss = false\_negative * 15 + false\_positive * 1$$

למעשה אם נמזער את מדד הprecision נוכל למזער את הלוס.

ביצענו תחילה עיבוד מקדים משותף ל- test ול- train בעזרת ה Pre-processing שעשינו בחלקים הקודמים ובנוסף השתמשנו במסווגים ישנים וחדשים על מנת להגיע לתוצאות הטובות ביותר.

*בדוח זה אציג את המסווגים החדשים שהכנסנו ואת האופי שבו התייחסנו למשקלים שקיבלנו.

המסווגים בהם בחרנו להשתמש:

- Decision Tree
- Random Forest
- Stochastic Gradient Decent
- Gradient Boosting
- Naive Bayes

*נשים לב שSVM לא נמצא ברשימה מכמה סיבות קיבלנו תוצאות לא טובות עבור המסווג הזה ובנוסף זמן הריצה הארוך ולכן לא המשכנו להשתמש בו ולהכניס אותו בהשוואות.

Decision Tree

עץ החלטות הוא מסווג מאוד טוב, מהיר ולומד את ה- Train בצורה מאוד מדויקת ואף יותר מדי מדוייקת.

במסווג זה בתחילה השתמשנו כמו בחלק הקודם ללא מישקול, ואח"כ עשינו אופטימיזציה לפרמטרי העץ שוב ללא התחשבות במשקלים. לאחר מכן יצרנו עץ ממשוקל כלומר גרמנו לנטייה בסיווג עבור ערך אחד על פני השני ובנוסף עץ החלטות שמשתמש באופטימיזציה ומסווג לפי threshold. בסוף נשווה בין כל המסווגים.

להלן העץ ללא אופטימיזציה וללא משקול:

```
# Create Decision Tree classifer object
clf_T = DecisionTreeClassifier(criterion='gini', splitter='best', random_state=42)

# Train Decision Tree Classifer
clf_T = clf_T.fit(X,Y)

#cross - validation
acc_simple_tree, loss_simple_tree, x_tree, y_tree=k_folds(clf_T,X,Y)
```

```
Accuracy:  0.8663044860353459
loss per entry:  1.011760909879364
```

הפרמטרים אותם כוונו:

```
max_features = ['auto', 'log2', 2, 5, 8]
max_depth = [int(x) for x in np.linspace(5, 100, num = 8)]
max_depth.append(None)

min_samples_split = [int(x) for x in np.linspace(math.floor(X.shape[0]/100),(5*math.floor(X.shape[0]/100)),num = 8)]

min_samples_leaf = [int(x) for x in np.linspace(math.floor(X.shape[0]/100),(2*math.floor(X.shape[0]/100)),num = 8)]

splitter='best'
random_grid = {'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
pprint(random_grid)
```

```
{'max_depth': [5, 18, 32, 45, 59, 72, 86, 100, None],
 'max_features': ['auto', 'log2', 2, 5, 8],
 'min_samples_leaf': [305, 348, 392, 435, 479, 522, 566, 610],
 'min_samples_split': [305, 479, 653, 827, 1002, 1176, 1350, 1525]}
```

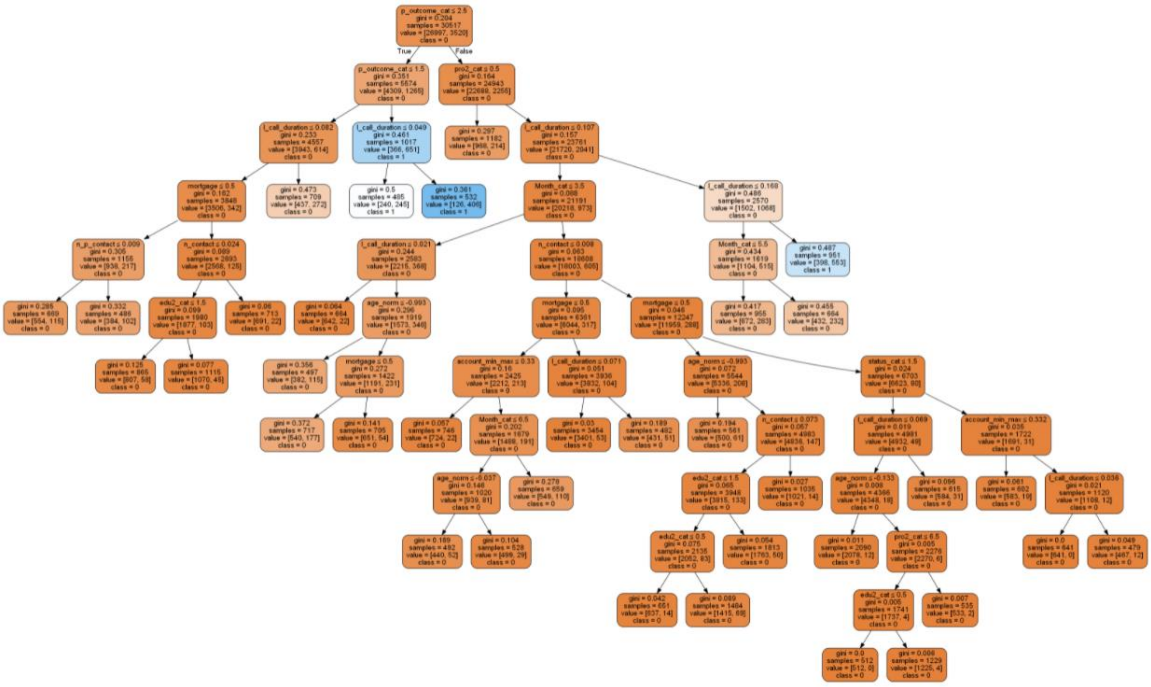
בעזרת טבלה זו אנו מוצאים עמק מקסימלי אופטימלי מוצאים כמות פיצולים אופטימלית ודורשים כמות מסויימת של דגימות כדי לאשר פיצול או סיווג.

```
best_random_simple_tree=Tree_random.best_estimator_
# Predict the label
Tree_random.best_params_
```

```
{'min_samples_split': 305,
 'min_samples_leaf': 479,
 'max_features': 2,
 'max_depth': 72}
```

העץ לאחר אופטימיזציה ללא משקול:

Out[12]:



העץ המקורי במחברת מייצג בצורה טובה בעיה של אובר פיטינג (בגלל שאנחנו לא מגבילים אותו בשום פרמטר ולכן הוא פרוש עד ההחלטה האחרונה) וכאן אפשר לראות שינוי משמעותי במספר הפיצולים. בנוסף ניתן לראות כי המון עלים צבועים בכתום מה שאומר שיש נטויה לסווג כnegative נוכל לראות בעקבות זאת עלייה בערך הדיוק.

עץ פשוט

```
Accuracy:  0.8663044860353459
loss per entry:  1.011760909879364
```

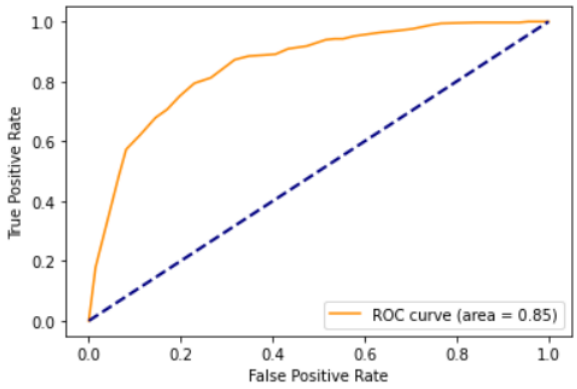
עץ משודרג

```
Accuracy:  0.894190289757392
loss per entry:  1.3462977138750458
auc:  0.8527442193713691
```

שיפור בדיוק אבל ירידה בלוס.

```
Improvement of 3.22%.
```

העקום המשוייך לעץ המשודרג.



לאחר מכן הוספנו משקולות לעץ וקיבלנו:

```
clf_T_weight = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=7 ,class_weight={0:1,1:15}, random_state=42)

# Train Decision Tree Classifer
clf_T_weight = clf_T_weight.fit(X,Y)

#cross - validation
acc_simple_tree_weight, loss_simple_tree_weight,x_simple_tree_weight, y_simple_tree_weight=k_folds(clf_T,X,Y)
```

```
Accuracy:  0.8663044860353459
loss per entry:  1.011760909879364
```

העץ שקיבלנו נמצא במחברת אבל ניתן לראות שם שבחירת הקלאס יותר מאוזנת ואף נוטה להיות 1, אך עדיין הפונקציית לוס גבוהה.

לסיום, יצרנו עץ נוסף ושינינו את ערך הthreshold להיות $\text{threshold} = \frac{1}{1+15}$ ולא 0.5 (דיפולטיבי)

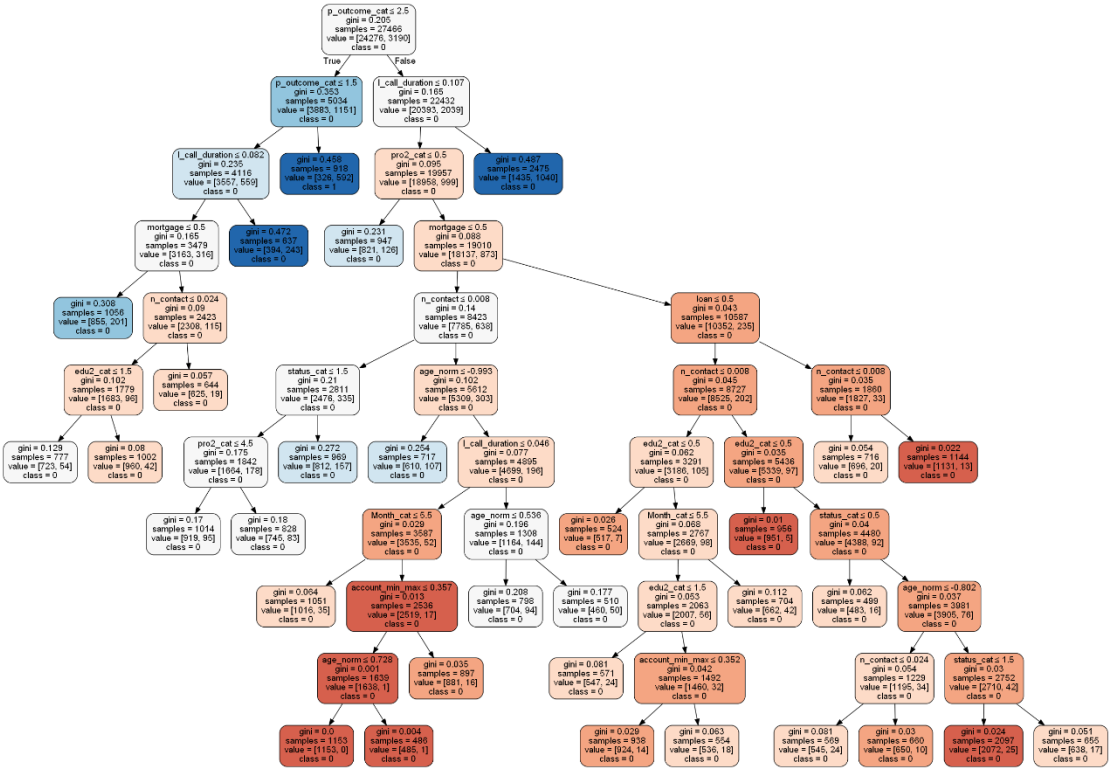
ובאמצעות זאת להתייחס לכך הדאטה לא מאוזן. את זה עשינו על העץ עם הפרמטרים של האופטימיזציה.

להלן התוצאות:

```
Accuracy:  0.6763112173865603
loss per entry:  0.5218721124887399
```

כפי שניתן לראות ישנה ירידה בדיוק, Accuracy, אבל גם ירידה משמעותית בלוס -כפי שרצינו.

להלן העץ לאחר אופטימיזציה ושינוי הthreshold:



ניתן לראות חלק נכבד מהקליסטור נהפך לצבעי כחול וזה התרגום של המסווג לthreshold שהכנסנו.

סה"כ ניתן לראות כי המסווג שנותן לנו עד כה את הLOSS הקטן ביותר הוא האחרון.

Random Forest

האלגוריתם של Random Forest מבוסס על Ensemble Methods , תפיסה שטוענת שלא טוב להסתכל על מודל אחד ספציפי אלא צריך לשקלל נתונים מכמה מודלים.

גם במודל זה עשינו את אותו תהליך שעשינו בעץ הפשוט והתחלנו עם מסווג בסיסי ואז מסווג עם אופטימיזציה של נתונים ולאחר מכן בסיסי עם מישקול ומסווג אופטימלי עם threshold.

מסווג בסיסי:

```
rf = RandomForestClassifier(random_state = 42,n_jobs=-1)
rf.fit(X, Y)
acc_rf, loss_rf,X_test_cm_rf, Y_test_cm_rf=k_folds(rf,X,Y)
```

```
Accuracy:  0.9035622250487882
loss per entry:  1.0914887283158778
```

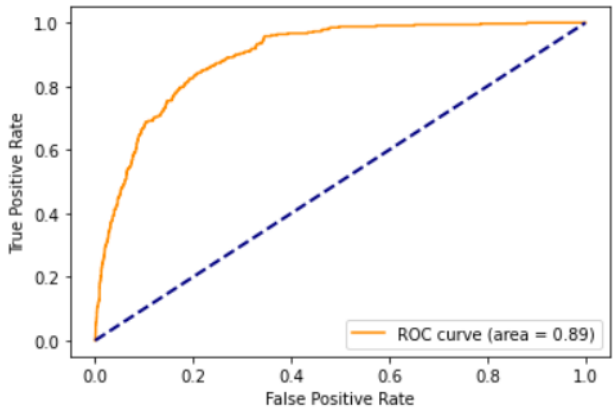
הפרמטרים שכיוונו:

```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, num = 10)]

max_features = ['auto', 'log2', 2, 5, 8]

max_depth = [int(x) for x in np.linspace(5, 100, num = 10)]
max_depth.append(None)


min_samples_split = [int(x) for x in np.linspace(math.floor(X.shape[0]/100),(5*math.floor(X.shape[0]/100)),num = 8)]

min_samples_leaf = [int(x) for x in np.linspace(math.floor(X.shape[0]/100),(2*math.floor(X.shape[0]/100)),num = 8)]



max_samples = [0.1, 0.3, 0.5, 0.7, 0.9]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'max_samples': max_samples}
pprint(random_grid)
```

```
{'max_depth': [5, 15, 26, 36, 47, 57, 68, 78, 89, 100, None],
 'max_features': ['auto', 'log2', 2, 5, 8],
 'max_samples': [0.1, 0.3, 0.5, 0.7, 0.9],
 'min_samples_leaf': [305, 348, 392, 435, 479, 522, 566, 610],
 'min_samples_split': [305, 479, 653, 827, 1002, 1176, 1350, 1525],
 'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]}
```

לאחר היפר פרמטרים:

```
Accuracy:  0.8979913446078097
loss per entry:  1.3273594953935135
Improvement of -0.62%.
```



מסווג בסיסי עם משקול:

```
rf_weight = RandomForestClassifier(random_state = 42,n_jobs=-1,class_weight={0:1,1:15})
rf_weight.fit(X, Y)
acc_rf_weight, loss_rf_weight,X_test_cm_rf_weight, Y_test_cm_rf_weight=k_folds(rf_weight,X,Y)
```

```
Accuracy:  0.9012028370476044
loss per entry:  1.2250529551576885
```

מסווג אופטימלי עם threshold:

```
Accuracy:  0.6408893502463365
loss per entry:  0.4279240031736581
```

סה"כ ניתן לראות כי המסווג שנותן לנו עד כה את הLOSS הקטן ביותר הוא האחרון.

Naïve Bayes

מסווג זה נותן לכל component בדאטה הסתברות מסוימת להיות משויך לכל אחד מה- classes.

הרצנו את המסווג הבייסיוני הבסיסי רגיל ואחד עם threshold :

```
clf_bg = GaussianNB()
clf_bg.fit(X, Y)
acc_clf_bg, loss_clf_bg, X_clf_bg, Y_clf_bg=k_folds(clf_bg,X,Y)
print('Improvement of threshold')
acc_clf_bg, loss_clf_bg, X_clf_bg, Y_clf_bg=k_folds(clf_bg,X,Y,threshold=threshold)
#bg_Y_pred = clf_bg.predict(X_test)
```

```
Accuracy:  0.8368128340706891
loss per entry:  1.00409449365161
Improvement of threshold
Accuracy:  0.5292779090112045
loss per entry:  0.6413809815916659
```

ניתן לראות כי הthreshold אכן שיפר את תוצאת הLOSS אך עדין המסווג הקודם טוב יותר.

Gradient Boosting

מסווג חדש- לפי שמו נוכל לנחש שאלגוריתם זה מתעסק במזעור של הפרש בין ערך נצפה לבין ערך חזוי (שאותם הוא מחשב תחילה לפי לוג הסיכוי להיות TRUE) כלשהו והוא משתמש בשיטה של בוסטינג בנוסף.

$$\log\big(P(Target = TRUE)\big)$$

ואז הוא מחשב את ההפרשים:

$$Residual = Observed - Predicted(in\ the\ first\ round\ obsered = 0\ IF\ False\ and\ 1\ o.w.)$$

לאחר מכן בונים עץ לפי הפיצרים ועמודת השאריות וכדי לסווג מחשבים ערך לכל עלה בעץ לפי

▼

$$\frac{\sum \text{Residual}_i}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]}$$

ולבסוף מחשבים

$$\log(odds) = first_{val} + learning\_rate * \Sigma tree_i val$$

ואז מחשבים הסתברות כך:

$$Probability = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

חלק מההבדלים בין אדבוסט לגרדיאנט זה שבאדאבוסט נוצרים בדרך כלל עצים בעלי שני עלים ובעוד שכאן העץ הראשון הוא עלה שמהווה סוג של ניחוש עבור סיווג על בסיס השגיאה נוצרים עצים חדשים.

הפרמטרים שאותם כיוונו

```
gboosting_clf = GradientBoostingClassifier(n_estimators=100, random_state=0)

parameters_gboost = {
    "n_estimators":[30,70,100,150,200,250],
    "max_depth":[5,6,7,8,9],
    "learning_rate":[0.1,0.15,0.2,0.25,0.3,0.35,0.4,1]
}
```

התוצאות עם ה threshold:

```
Accuracy:  0.7601670358814956
loss per entry:  0.3531456394633305
```

סה"כ ניתן לראות כי המסווג שנותן לנו עד כה את הLOSS הקטן ביותר הוא זה.

Stochastic Gradient Decent

מסווג חדש- אלגוריתם gradient decent אשר ביצועיו מהירים יותר.

בדומה לגרדיאנט דיסנט האלגוריתם מייצר קו מגמה כדי לנסות על פיו לסווג והוא מחשב שגיאה לפי מרחק מהקו(ריבועים מינימלים) ומנסה ליצור קו מגמה כך שהסכום של השגיאות הוא מינימלי.

העיניים שקיימות המון השוואות שצריך לעשות אז זמן הריצה הוא ארוך לעומת זאת האלגוריתם שבחרנו לוקח צעדים גדולים כשהוא רחוק מהמינימום וצעדים קטנים כשהוא קרוב למינימום כדי לשפר דיוק וזמני ריצה.

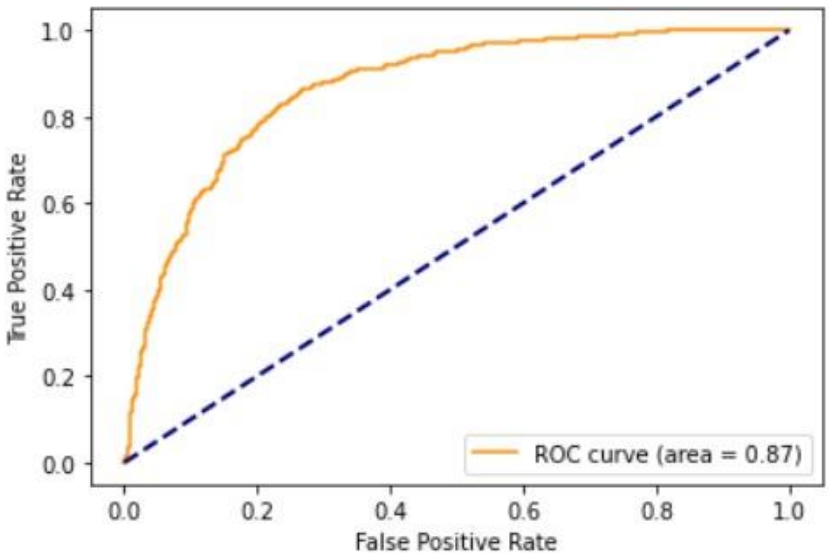
יש הסברים ונוסחאות אבל זה מרגיש טו מאץ להסבר פשוט בקצרה.

לפני אופטימיזציה וללא משקול:

```
Accuracy:  0.8858020789436718
loss per entry:  1.4969024937787623
```

לפני אופטימיזציה עם משקול:

```
Accuracy:  0.662447941568263
loss per entry:  0.6696800417369551
```



פרמטרים שכיוונו:

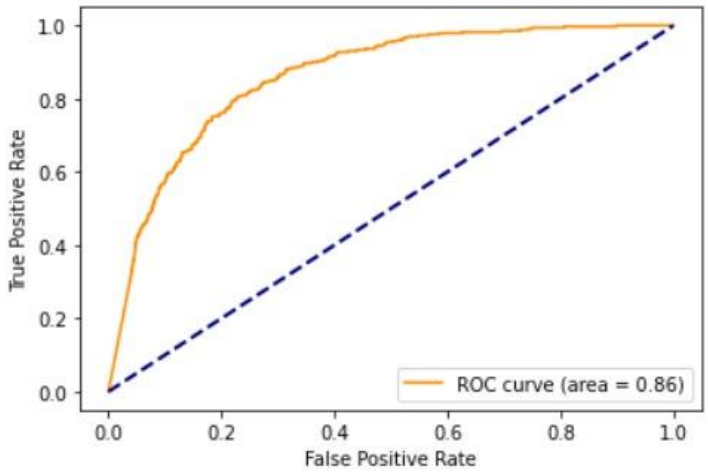
```
loss = ['log', 'modified_huber', 'squared_hinge']
max_iter=[500, 1000]
penalty = ['l1', 'l2', 'elasticnet']
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
learning_rate = ['constant', 'optimal', 'invscaling']
class_weight = [{0:1, 1:15}]
eta0 = [0.3, 0.5, 1, 10, 100]
param_distributions = dict(loss=loss,
    penalty=penalty,
    alpha=alpha,
    learning_rate=learning_rate,
    class_weight=class_weight,
    eta0=eta0)
```

לאחר אופטימיזציה +משקול:

```
Best Score:  0.8474104448003839
Best Params:  {'penalty': 'elasticnet', 'loss': 'modified_huber', 'learning_rate': 'optimal', 'eta0': 0.5, 'class_weight': {0:
1, 1: 15}, 'alpha': 0.001}
Accuracy:  0.5987514567769501
loss per entry:  0.6379621789989575
```

ההשוואה בין המודל הבסיסי ביותר למודל לאחר אופטימיזציה + משקול:

```
Improvement of -32.41%.
```



מודל לאחר אופטימיזציה + threshold :

```
Accuracy:  0.3089990334126546
loss per entry:  0.7387079899041982
```

סה"כ ניתן לראות כי המסווג שנותן לנו את הLOSS הקטן ביותר הוא Gradient Boosting.

כעת נשווה בין כל המסווגים שנתנו תוצאות LOSS נמוכות:

*במחברת ניתן לראות מפורט

להלן הממוצעים:

```
DecisionTreeClassifier(class_weight={0: 1, 1: 15}, max_depth=7, random_state=42)
mean loss:  0.43274189155694387
SGDClassifier(class_weight={0: 1, 1: 15}, loss='log')
mean loss:  0.6249984535504548
SGDClassifier(alpha=0.001, class_weight={0: 1, 1: 15}, eta0=0.5,
              loss='modified_huber', penalty='elasticnet')
mean loss:  0.5556224180199175
DecisionTreeClassifier(max_depth=72, max_features=2, min_samples_leaf=479,
                       min_samples_split=305, random_state=42)
mean loss:  0.5218721124887399
RandomForestClassifier(max_depth=15, max_features=8, max_samples=0.9,
                       min_samples_leaf=392, min_samples_split=827,
                       n_estimators=200, n_jobs=-1, random_state=42)
mean loss:  0.4279240031736581
GaussianNB()
mean loss:  0.6413809815916659
GradientBoostingClassifier(max_depth=6, n_estimators=30, random_state=0)
mean loss:  0.3531456394633305
```

שוב ניתן לראות כי המסווג הטוב ביותר הוא **Gradient Boosting**

לאחר מכן עשינו את מבחן ה T-test ובכל המסווגים ביחס למסווג זה יצא

```
reject null hypothesis
```

*ניתן לראות במחברת.

ולסיכום בחרנו Gradient Boosting וחזינו את הפרדיקציה בעזרתו.

תוצאות הפרדיקציה בתקיית דרייב.