כריית מידע וייצוג מידע (83676)
דו"ח מסכם - פרויקט חלק 2

עידו שר שלום 21240146, תומר גריבה 325105625

הקדמה:

בחלק זה של הפרויקט, אנו מתבקשים ליצור מודל סיווג אשר ישמש את מנהלי השיווק כדי להחליט לאילו לקוחות פוטנציאליים חדשים לפנות.
ערך המטרה אותו נרצה לחזות הוא Response .כלומר, בהינתן פרטים אודות צרכן מסוים נרצה לדעת האם הוא יקבל את ההצעה לקמפיין או לא.
כאמור, נעשה זאת על ידי אימון מודל סיווג בפרט, המודל ילמד בעזרת מידע על לקוחות עבר של החברה כך יוכל לשערך בהינתן מידע על לקוח חדש אם יענה בחיוב להצעה למוצר או לא.
חלק זה מסתמך על שלב הכנת הנתונים (preprocessing) אשר ביצענו בחלק הקודם.

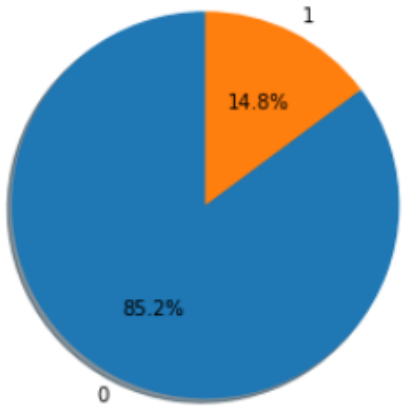
בחלק זה של הפרויקט:

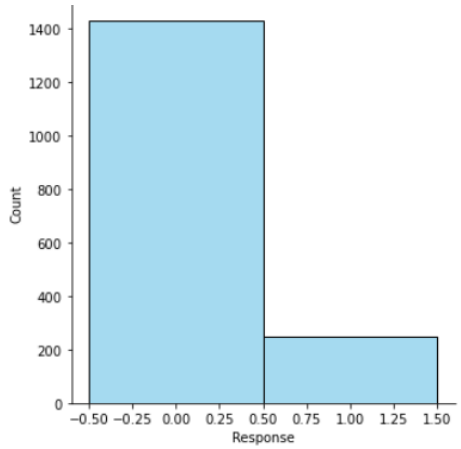
- נתאר את שלושת מודלי הסיווג בהם נשתמש
  1. Decision Tree
  2. SVM
  3. Neural Network

- ניישם את תהליך ה- preprocess עבור סט המבחן תוך טיפול בדאטא לא מאוזן.
- נפצל את הנתונים לסט אימון ו- וולידציה.
- נבצע hyperparameters tuning עבור כל מודל סיווג.
- על ידי cross-validation נאמוד את טיב המסווגים.
- נאמן את המסווגים השונים ונציג מדדי הערכה כדי להשוות ביניהם.
- לבסוף, נבחר את המודל הטוב ביותר, במונחים של מובהקות סטטיסטית, אשר יחול על מערך נתוני הבדיקה (test-set) המצורף.

טיפול ב data לא מאוזן

במשימה אותה קיבלנו, ערך ה (Response) target הוא 0 או 1.
ההתפלגות היא:





כלומר, הרוב המוחלט של האנשים סירבו להצעת הקמפיין.
הדאטא לא מאוזן שכן יש הטיה עבור 0, סירוב להצעת הקמפיין. נרצה לטפל בבעיה.

הסיבה לטפל בבעיה זו היא כי אין הפרט מעיד על הכלל. כלומר, התפלגות קלאסים זו של דאטא האימון אינה מעידה על דאטא המבחן.
במקרה זה, המודל ילמד כי רוב האנשים מסרבים להצעת הקמפיין. מספיק כי יוציא 0 עבור כל הרשומות והוא יצליח ב- 85% מהן.
דבר זה אינו מעיד על דאטא המבחן לדוגמה, ב- test-set במקרה שהקמפיין היה מאוד מוצלח, יכולה להיות הטיה ל- 1 המודל יסווג את רוב הרשומות ל- 0, יתקבלו תוצאות סיווג גרועות.

כדי למנוע בעיה זו נצטרך לאזן את ה- data נעשה זאת בדרכים הבאות:

- Oversampling - עבור ערך ה target ממנו יש פחות. כלומר, ייצור דוגמאות חדשות, בהן ערך ה target הוא.
- Undersampling - עבור ערך ה target ממנו יש יותר. כלומר, תת דגימה של רשומות אשר ערך ה- target שלהן הוא 0.

עשינו זאת בעזרת שימוש ב SMOTE, RandomUnderSampler, כך נקבל איזון של ה- data (יחס 1).
כמתואר מטה.

```
# Balance the data
over = SMOTE(sampling_strategy=0.3, random_state = 42)
under = RandomUnderSampler(sampling_strategy=1, random_state = 42)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
X_train, y_train = pipeline.fit_resample(X_train, y_train)
print(len(y_train))
print(y_train.value_counts())
smote2_data = pd.concat([pd.DataFrame(X_train),pd.DataFrame(y_train, columns =['Response'])], axis =1)
print(smote2_data['Response'].value_counts())
```

לאחר יישום שיטות אלו על הדאטא, ערך ה response מתפלג בצורה הבאה:

```
604
0.0    302
1.0    302
Name: Response, dtype: int64
0.0    302
1.0    302
Name: Response, dtype: int64
```

אכן קיבלנו איזון עבור ערך ה- target.
מצד אחד, קיבלנו צמצום עבור רשומות דאטא האימון, 672 רשומות לעומת 1576. כלומר, למידת המודל מתבצעת על פחות דוגמאות דבר אשר יכול לפגוע בביצועיו. מצד שני, הדאטא מאוזן ואימונו על המודל אמנם יניב תוצאות פחות טובות על ה- trainset אבל, ההכללה של המודל ל- testset במקרים רבים תהיה יותר טובה.
ולכן, נעדיף להשתמש באימון המודל על הדאטא המאוזן.

חלוקת ה- dataset ל- train ו- validation.

על ידי הפונקציה train_test_split ביצענו חלוקה לכלל דאטא האימון לסט וואלידאציה וסט אימון:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, shuffle=True, random_state=42)
```

גודל דאטא הוולידציה מהווה כרבע מכלל דאטא האימון (לפני החלוקה), החלוקה מתבצעת בהתאם לרשומות ולייבלים.

תהליך ה- hyperparameters tuning

את תהליך ה- hyperparameters tuning נרצה לעשות לכל מודל שאנו בוחנים לפני ההשוואה בין המודלים.
כלומר, קודם כל נאפטם כל מודל בפני עצמו וזאת על ידי בחירה של ה-hyperparameters אשר יתנו לנו את התוצאות הכי טובות. אחר כך, באמצעות שיטות cross-validation נשווה בין מודלי הסיווג ונבחר את הטוב ביותר.
נציין כי נשתמש גם ב- cross-validation להערכה של מודלים עבור ערכי היפר פרמטרים שונים כלומר, ערכי הגרפים התקבלו על סמך מיצוע ה- accuracy על פני K=10 folds, באלגוריתם האימון K-fold cross validation.

אופטימיזציית hyperparameters נעשתה על ידי RandomizedSearchCV ו- GridSearchCV.
בכל אחת משיטות אלו הגדרנו רשת טווחים של ערכי היפר פרמטרים של המודל.

- שיטת RandomizedSearchCV - טווח ערכי היפר-פרמטרים נדגם רנדומלית n_iter פעמים.
- שיטת GridSearchCV - נבדקות **כל** הקומבינציות האפשריות של ערכי hyperparameter.

בשתי השיטות האופטימיזציה הינה על פי קומבינציית הערכים אשר הניבה תוצאות דיוק טובות ביותר.

משום מה כאשר הרצנו עם Grid Search, על מציאת כל הקומבינציות השונות התוצאות היו פחות טובות מאשר Randomized Search

סיבה אפשרית לכך הינה, כיוון שהמודל מחפש את כל אפשרויות הפרמטרים אשר יניבו תוצאות דיוק מרביות עבור ה- trainset, אזי הוא ממקסם את ערך הדיוק על ה- train אך, במובן מסוים ביצע Overfitting עליו.לכן, תוצאותיו על סט המבחן תהיינה פחות טובות.

<u>שמירת וטעינת פרמטרי מודל סיווג לתוך/מתוך הדיסק (זיכרון המחשב)</u>

בחלק זה של הפרויקט היו ריצות רבות ושונות על סמך פרמטרים מגוונים להשגת אופטימיזציה של hyperparameter. לא מעט מהריצות אשר הרצנו קודם הניבו תוצאות טובות מאשר הריצות הנוכחיות אך, לא ניתן היה לשחזר בדיוק את הגדרות המערכת אשר נתנו תוצאות אלו. לכן, כדאי לשמור פרמטרי מודל אשר הניבו תוצאות טובות.

להלן הקוד אשר מבצע שמירה וטעינה של מודל לתוך/מתוך הדיסק:

```
# Save model parameters
path = './saved_models/best_random_model.joblib'
dump(best_random, path)
# Load the model from disk
loaded_model = load(path)
```

Model evaluation

הערכת מודלים של למידת מכונה הינה דבר חשוב עבור כל פרויקט. ישנם סוגים רבים ושונים של מדדי הערכה לבדיקת מודלים

מטריצת confusion מתארת קשר בין classes בבעיית חיזוי כלומר, איבר i,j במטריצה מתאר כמות רשומות בקלאס i אשר סווגו על ידי המודל כקלאס j.
בפרט עבור בעיית קלסיפיקציה בינארית ה- confusion matrix היא:

	predicted negative	predicted positive
actual negative	TN	FP
actual positive	FN	TP

בדומה, גם הבעיה איתה אנו מתעסקים היא קלסיפיקציה בינארית (נענה בחיוב לקמפיין/סירב לקמפיין).
ולכן, בעזרת מטריצת confusion ובפרט ממדי השוואות אשר היא מגדירה נוכל ללמוד אודות ביצועי המודל.

ראינו מדדי השוואת ביצועים אשר נבעו מ- confusion matrix, ביניהם:

evaluation metric	sensitivity = recall	specificity	precision	f1-score
formula	$\frac{TP}{TP+FN}$	$\frac{TN}{TN+FP}$	$\frac{TP}{TP+FP}$	$\frac{2 \cdot precision \cdot recall}{precision+recall}$

נשאלת השאלה, מהו מדד ההערכה שעלינו להשתמש בו בבעיה שלנו?

נדון בשאלה זו ופתרונה.

ראשית, הבעיה שאנו עוסקים בה היא חיזוי לקוח אשר יענה בחיוב להצעה למוצר או שירות.
נשים לב כי גם אם הייתה שגיאה בחיזוי ההפסד אינו רב, שיחת טלפון ללקוח אשר סירב להצעה במקרה של false-positive אי ביצוע שיחת טלפון ללקוח אשר היה מקבל את ההצעה במקרה של false-negative.
לעומת בעיות אחרות למשל, חיזוי גידול ממאיר או שפיר במצב כזה כל החלטת חיזוי היא משמעותית לחיי המטופל.

ולכן, כיוון שהשלכות חיזוי שגוי (false-positive/negative) במודל שלנו אינן רבות מאוד נרצה למקסם את החיזוי הנכון (true-positive/negative) תוך מזעור אמנם לא אפסי של השגיאות.

שיקולים אלו, מתאימים לנו בדיוק למדד סנסיטיביות (sensitivity=recall) שכן נרצה את מירב הדיוק בפרדיקציית הלקוחות הפוטנציאליים תוך מזעור ערך ה- "פספוס" לקוחות אשר בדיעבד היו נענים בחיוב לקמפיין.

נשווה למדדים אחרים, specificity ממקסם את כמות הלקוחות אשר יענו בשלילה לקמפיין (true-negative) דבר אשר לא נחוץ עבורינו.

precision שואף להגדיל את דיוק הלקוחות אשר יענו בחיוב בדומה ל- recall אבל ממזער את FP,
בשונה מ- recall אשר ממזער את FN.

ההבדל המשמעותי הוא:
נעדיף למזער את ה- FN שכן אני אמנם אתקשר ליותר לקוחות אבל לא אפספס לקוחות פוטנציאליים.
לעומת מזעור FP אשר יגרור התקשרות מיותרת ללקוחות אשר יענו בסירוב, דבר אשר לא לוקח ממני משאבים רבים **אבל**, גורם לי להחמצה של לקוחות אשר כן היו יכולים להיענות בחיוב.

לכן, נעדיף את מטריקת sensitivity=recall על פני precision.

מדד f1-score משלב את precision ו- recall על ידי לקיחת ממוצע הרמוני שלהם.
כפי שנאמר מדד precision פחות משמעותי עבור הבעיה הנתונה, נרצה להתחשב במדד ה- recall בבחירת המודלים.

נציין כי בתעשייה היינו רוצים למקסם את ה- recall מכיוון שאנו לא נרצה לפספס לקוחות. יחס עלות-תועלת משתלם שכן משאב השיחה ללקוח הוא יחסית זניח לעומת התועלת המתקבלת מלקוח אשר נענה בחיוב.
אבל, מטרתינו בפרויקט זה הינה accuracy מירבי.

<u>השוואה בין מודלים</u>
על מנת להשוות בין מודלי סיווג שונים נשתמש באלגוריתם cross-validation עבור K=10 folds.
אלגוריתם זה מבטיח לנו כי כל חלק מתוך הדאטא משמש כ- testset דבר אשר גורר חוסר הטיה כלפי מודל מסוים אשר מצליח ב- testset מסוים בצורה טובה יותר על פני מודלי סיווג אחרים.
שכן בתעשייה היינו רוצים למקסם את ה- recall אבל, מטרתינו בפרויקט זה הינה accuracy מירבי.

<u>מובהקות סטטיסטית</u>
מובהקות סטטיסטית הינה כלי אשר יעזור לנו לאמוד מודל סיווג ביחס למודל סיווג אחר.
באמצעות t-test נוכל לקבוע האם השערת האפס מתקיימת או לא.
כלומר, תוצאת t-test הינה ה- p value אם ערכו של p value גדול מרמת המובהקות הסטטיסטית α אזי נדחה את השערת האפס אחרת, לא נדחה אותה.
ה- null hypothesis אשר נעבוד לפיה היא: אם שתי דגימות עבור שני מודלי סיווג קשורות או חוזרות אז ישנם ערכים ממוצעים זהים.
מטרתינו היא לקבל את השערת האפס וזאת כדי לקרב בין המודלים השונים כלומר, עבור דגימות "דומות" בין שני מודלי סיווג נרצה לקבל כי המודלים באופן ממוצע זהים.

מודל ראשון: עץ החלטה - Decision Tree

עץ החלטה הינו מודל סטטיסטי המשמש לסיווג מידע.
עץ ההחלטה הוא עץ בינארי, בו בכל צומת יש תנאי. מסלול בעץ בין שורש לעלה מתאר החלטה עבור רשומה ספציפית כאשר הסיווג בכל צומת הוא בינארי כלומר, רשומה אשר מקיימת תנאי ספציפי ממשיכה ימינה בעץ, ורשומה שלא מקיימת את התנאי ממשיכה שמאלה.
מודל עץ החלטה לומד להפריד את הדאטא על פי המובהקות הסטטיסטית שלו. כלומר, בכל פיצול נבחר מאפיין אשר ישרה קבלת מידע מירבי אודות הרשומה, באופן כזה אשר יביא לחיזוי אופטימלי של הלייבל שלה.
כמו כן, אחד מיתרונות עצי ההחלטה הוא הצגתו הויזואלית לכן, קל להבין ולפרש אותם.

כעת, נבנה את המודל כלומר, נבחר את ה- hyperparameters אשר יניבו תוצאות טובות ביותר.

בדקנו את הפרמטרים הבסיסיים של Decision tree, והחלטנו לעשות Hyperparameters Tuning עבור הפרמטרים:

- max_features - מספר מקסימלי של מאפיינים הנדרש לפיצול צומת.
- max_depth - העומק המרבי של העץ.
- min_samples_split - מספר מינימלי של רשומות הנדרש לפיצול צומת.
- min_samples_leaf - מספר מינימלי של רשומות הנדרש להיות בצומת עלה.

הגדרנו טווח ערכים עבור ה- hyperparameters השתמשנו בשני אלגוריתמי hyperparameter tuning לקבלת אופטימיזצית דיוק המודל.
שיחקנו עם טווח ערכי ה- hyperparameters, אפשרנו ערכים אשר יותר קרובים לאופטימליות המודל, בדקנו טווח ערכים רחב, נמנענו מ- Overfitting, התחשבנו בערך ה- recall ולא רק ב- accuracy.
לבסוף אחרי הרבה מאוד ריצות, אופטימיזציות בין ריצה לריצה. כאשר לקחנו בחשבון את הריצות אשר הניבו תוצאות טובות ביותר, שמרנו אותן בדיסק והשוונו אותן על פי שיפורן למודל הבסיסי. הגענו למסקנה עבור המודל סיווג הטוב ביותר בעל היפר פרמטרים הבאים:

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 17,
 'max_features': 10,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 2,
 'min_samples_split': 19,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 42,
 'splitter': 'best'}
```

כאשר היפר פרמטרים המסומנים אלו הם ההיפר פרמטרים עבורם ביצענו tuning.

תוצאות דיוק של סט הוולידציה והאימון עבור מודל סיווג בסיסי, מודל סיווג טוב ביותר עבור ריצת RandomizedSearchCV נוכחית, מודל סיווג אשר נטען מהמחשב לאחר שמירתו ו- יחס שיפור ביניהם לבסיסי:

```
train-set base model accuracy = 99.50%
validation-set base model accuracy = 74.11%
-----------------------------------------------------------------
train-set random model accuracy = 88.74%
validation-set random model accuracy = 76.65%
-----------------------------------------------------------------
train-set loaded from disk model accuracy = 90.73%
validation-set loaded from disk model accuracy = 77.41%
-----------------------------------------------------------------
validation-set improvement of random model compared to base model 3.42%.
validation-set improvement of loaded from disk model compared to base model 4.45%.
```

ניתן לראות Overfitting של המודל הבסיסי על ה- training data, דיוק של 99.5%.

כמו כן נשים לב כי ישנו trade-off בין ערך ה- accuracy של המודל ל- validation-set לבין ה- accuracy של ה- trainset. ככל שהמודל מתאמן על ה- trainset נקבל עליו תוצאות יותר טובות (עד ל- Overfitting מלא). ולכן, הכללת המודל לסט המבחן תהיה פחות טובה. נרצה לכוון תוצאות אלו כדי לקבל דיוק מקסימלי על סט הוולידציה.

קיבלנו כי תוצאות המודל אשר נטען מהזיכרון הן טובות יותר הן על סט הוולידציה והן על סט האימון. זאת ביחס למודל הרנדומי בריצה הנוכחית. כמו כן, שיפור יחסי של כ- 4.45% מהמודל הבסיסי, לא זניח בכלל.

נציג report מתומצת של מודל הסיווג עבור דיוק סט הוולידציה:

	0	1
0	263	73
1	16	42

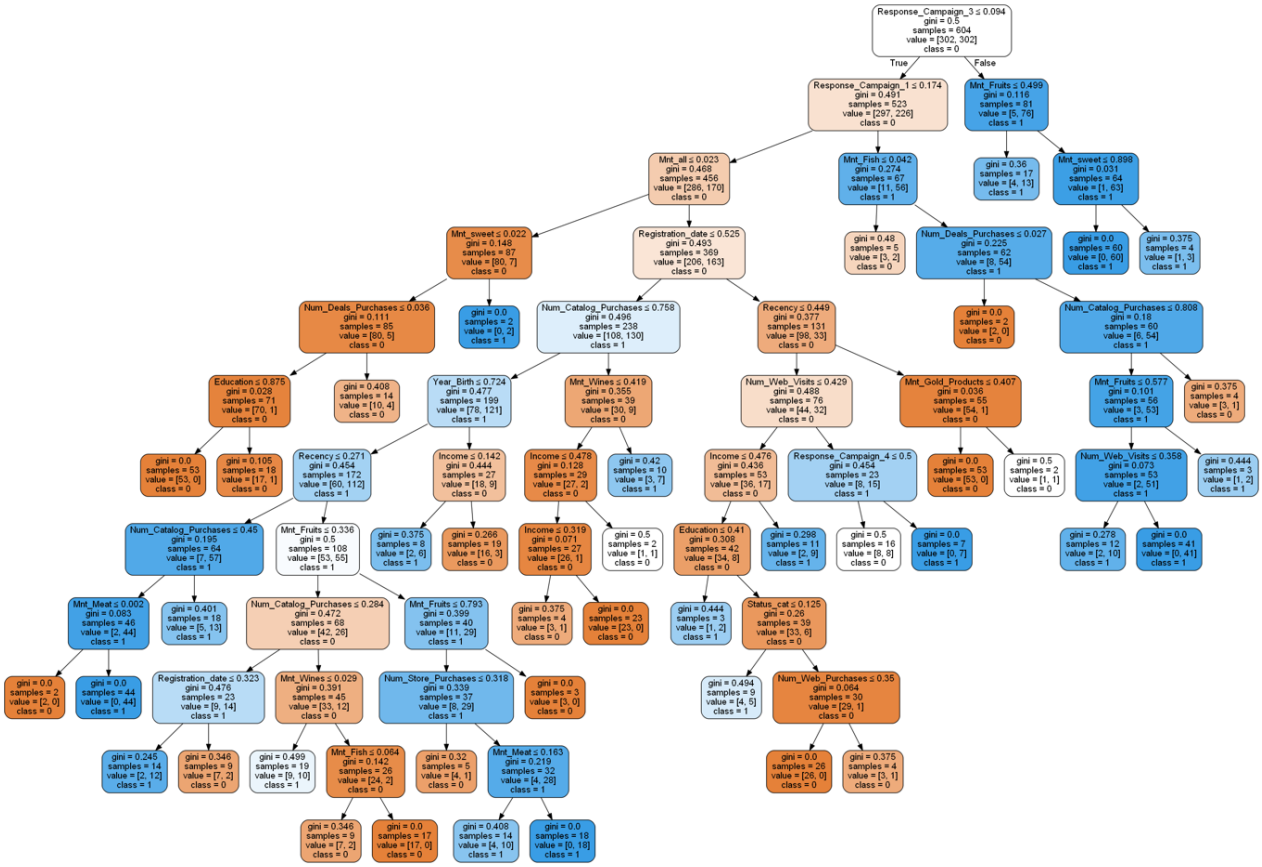
	precision	recall	f1-score	support
0.0	0.94	0.78	0.86	336
1.0	0.37	0.72	0.49	58
accuracy			0.77	394
macro avg	0.65	0.75	0.67	394
weighted avg	0.86	0.77	0.80	394

כאשר הערכים המסומנים הם recall ו- accuracy. ניתן לראות כי ערך ה- recall יחסית טוב כלומר, אין הטיה משמעותית של המודל לסיווג 0 (על אף הטיית הדאטא ל-0). כמו כן, ניתן לראות זאת ב- confusion matrix.

ערך מטריקת ה- recall הוא: 0.72, יחסית נמוך שאיפתינו היא למקסם מטריקה זו. למרות זאת, הדיוק הוא 0.77 accuracy טוב ביחס למודלים אחרים שאימנו, בהתחשב כי הדאטא מאוזן.

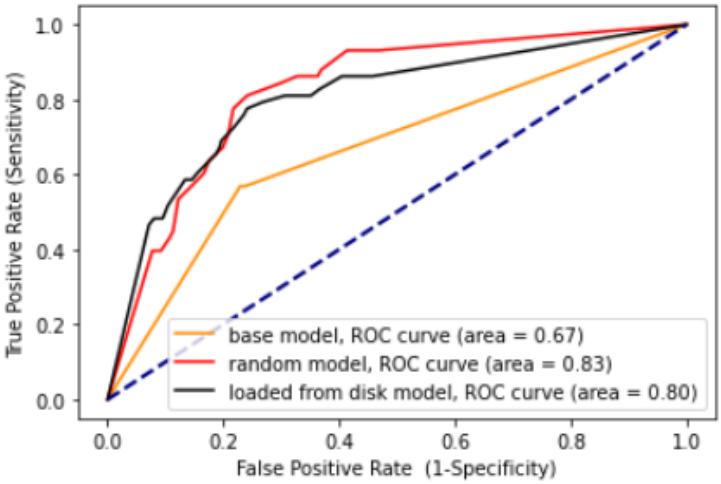
אפשר לשים לב כי אין Overfitting עבור מודל הסיווג הרנדומלי המתקבל מ- RandomizedSearchCV, וכן מודל הסיווג הנטען מהזיכרון. כלומר, המודל לומד מעבר לסט האימון.
ניתן לראות זאת מתוך דיוק המודל. דיוק סט האימון אינו מושלם (~90%) על כן אין התאמת יתר לנתוני האימון.
כמו כן, דיוק סט המבחן לא רע (~77%) כלומר, המודל לומד נכון, צולח בחיזוי נתונים אשר לא נראו מראש.
דבר נוסף המעיד על אי התאמת יתר הוא הפרש נמוך בין אחוזי דיוק אלו. יתר על כן, דיוק סט המבחן נמוך ביחס לדיוק סט האימון דבר הגיוני לנו שכן המודל מתאמן על נתוני האימון לעומת דוגמאות המבחן אשר לא נראות מראש.

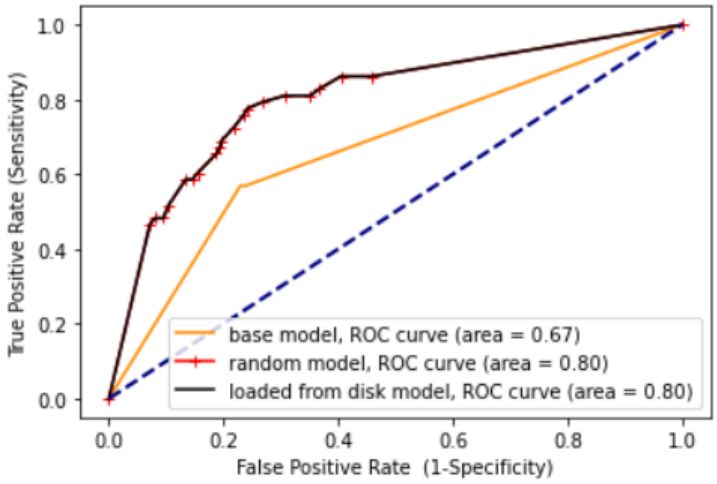
ויזואליזציית עץ החלטות אשר נטען מהזיכרון מתוארת מטה.



נשים לב כי אכן העומק המקסימלי של העץ הינו max_depth=17. כמו כן, כאשר נדרש פיצול בעץ עקב הפרמטר min_samples_split, הפיצול לא יבוצע אם עומק העץ יעבור את max_depth, ניתן לראות זאת בחלק מהעלים.
נציין כי במחברת Jupyter ישנה ויזואליזציה עבור כל מודל מהשלושה.

עקומות ROC עבור המודלים השונים מתוארות מטה (עבור שתי ריצות שונות).





ניתן לראות שיפור ביחס למודל הבסיסי. שטח העקומה של המודלים אשר עברו hyperparameter tuning גדל, שיפור של כ- 0.13 בקירוב.
עם זאת כתלות בריצות שונות לא ראינו שינוי משמעותי של הגדלה/הקטנה עבור השטח מתחת לעקומה יותר מהמודל השמור (0.8).
באופן כללי קיבלנו עקומה זהה יחסית לזו.
כפי שניתן לראות העקומות של המודלים אשר עברו hyperparameter tuning כמעט זהות זו לזו ו- השטח מתחת לגרף גם הוא.

מודל שני: רשת נוירונים - Neural Network

המודל השני בו בחרנו, הוא מודל של רשת נוירונים. מודל זה לא נלמד בכיתה.
ברשת נוירונים, המידע (features) נכנס לרשת, ומה שיוצא זה החלטת סיווג,0 או 1.
הרשת מתאמנת על פי ה train_data, בשיטה שנקראת backpropagation.
את הרשת בנינו בעזרת sklearn, כך שלא יצרנו רשת גדולה ועמוקה (כמו שניתן לעשות ב- pytorch).

לרשת יש 5 שכבות ליניאריות, כאשר יש 26 נוירונים בכל שכבה.
מאפייני הרשת הינם:

```
'activation': 'relu',

'alpha': 1e-05,

'batch_size': 'auto',

'beta_1': 0.9,

'beta_2': 0.999,

'early_stopping': False,

'epsilon': 1e-08,

'hidden_layer_sizes': (5,),

'learning_rate': 'constant',

'learning_rate_init': 0.001,

'max_fun': 15000,

'max_iter': 200,

'momentum': 0.9,

'n_iter_no_change': 10,

'nesterovs_momentum': True,

'power_t': 0.5,

'random_state': 1,

'shuffle': True,

'solver': 'lbfgs',

'tol': 0.0001,
```

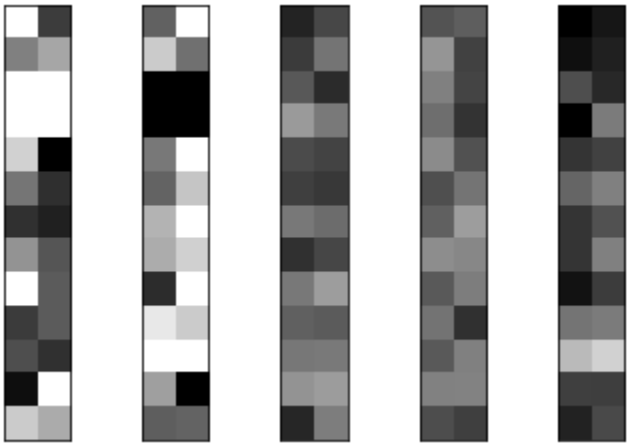
```
'validation_fraction': 0.1,

'verbose': False,

'warm_start': False
```

כלומר, ניתן לראות כי אנחנו משתמשים בפונקציית activation של relu, קצב למידה $learning\ rate\ =\ 0.001$ וכו'...

כעת, נראה את המשקולות של הרשת:



מכיוון שה data הוא טבלאי ולא תמונה, לא ניתן לראות צורות מעניינות במשקולות.

הרצנו את הרשת וביצענו fit על y_train , x_train.

לאחר מכן, בדקנו עבור ה validation set, וקיבלנו:

```
val acc:  0.8197969543147208
```

	0	1
0	277	59
1	12	46

עבור ה train, קיבלנו:

```
train acc:  0.8320251177394035
```

	0	1
0	277	58
1	49	253

כלומר, אין overfitting, ועבור ה validation set, יש דיוק של כ 82%.

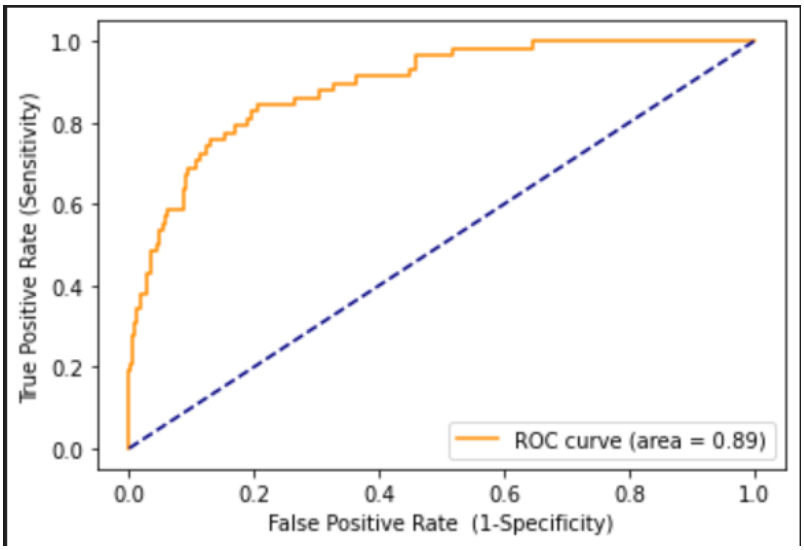
כפי שאמרנו קודם, המדד בו אנו מעוניינים, הוא sensitivity=recall, כלומר $\frac{TP}{TP+FN}$

אצלנו, מתקיים:

$$TP = 46,\ FN = 12$$

לכן, מתקיים $recall = \frac{TP}{TP+FN} = \frac{46}{46+12} = 0.793$

כמו כן, שרטטנו עקומת ROC:



ניתן לראות כי השטח מתחת לגרף ה ROC גדול יחסית, כפי שאנו צריכים שיהיה.

כעת, נציג report מתומצת של המודל שבנינו:

	precision	recall	f1-score	support
0.0	0.96	0.82	0.89	336
1.0	0.44	0.79	0.56	58
accuracy			0.82	394
macro avg	0.70	0.81	0.73	394
weighted avg	0.88	0.82	0.84	394

ניתן לראות כי למודל יש recall, precision יחסית גבוהים.

כמו כן, ביצענו cross validation למודל, כלומר, אימנו אותו כמה פעמים, כאשר בכל פעם חלק אחר ב data משמש כ validation והחלק האחר משמש לאימון המודל.

השתמשנו ב under sample על ה data שמסוג ל-0, ו over sample על ה data שמסוג ל-1.

קיבלנו:

```
Mean ROC AUC: 0.874
```

תוצאות טובות עבור ממוצע שטח מתחת לעקומות ה- ROC.

כעת נבצע Hyperparameter tuning עבור המודל.
לאחר שיצרנו את מודל רשת נוירונים בסיסית, בדקנו תחילה את הפרמטרים שיש לה, על מנת שנדע אילו פרמטרים אנו יכולים לשנות ב GridSearch.

מצאנו שהפרמטרים שיש לרשת הנוירונים הבסיסית הם:

```
{'activation': 'relu',
 'alpha': 1e-05,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 200,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': 1,
 'shuffle': True,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
```

```
 'warm_start': False}
```

בחרנו ליצור את ה- grid מהפרמטרים הבאים:

```
hidden_layer_sizes = [(3, ), (4, ), (5, ), (6, )]

max_iter = [200, 500, 1000]

learning_rate_init = [0.001, 0.01]

random_grid = {

    'hidden_layer_sizes': hidden_layer_sizes,

    'learning_rate_init': learning_rate_init,

    'max_iter': max_iter

}
```

הסיבה לשימוש בפרמטרים אלו הינה כי הם הפרמטרים המרכזיים ביותר עבור מודל סיווג רשת נוירונים.
לדוגמה, עומק הרשת משפיע על סיבוכיות הרשת, ככל שנגדיל את העומק המודל יותר ויותר יבצע Overfit לדאטא האימון, דבר דומה גם עבור מספר איטרציות גבוה.

השתמשנו ב GridSearch כדי למצוא את הפרמטרים האופטימליים לרשת (מתוך הפרמטרים שבחרנו), וקיבלנו:

```
{'hidden_layer_sizes': (3,), 'learning_rate_init': 0.01, 'max_iter': 500}
```

כלומר, רשת לא ממש עמוקה.

קיבלנו שהדיוק של הרשת החדשה מול הישנה הוא:

```
base model accuracy = 81.47%

random model accuracy = 81.73%

Improvement of 0.31%.
```

כלומר, כן יש שיפור, אך הוא לא גדול.

כמו כן, בדקנו את ה confusion matrix על ה validation set וקיבלנו:

	0	1
0	276	60
1	13	45

כלומר, מתקבל recall של $recall = \frac{45}{45+13} = 0.775$

מודל שלישי: SVM

המודל השלישי בו בחרנו, הוא מודל של SVM.
את הרשת בנינו בעזרת sklearn.svm.svc.

מאפייני ה SVM הינם:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': True,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

אימנו את המודל על X_train, y_train.

בדקנו את המודל על ה train ועל ה validation וקיבלנו:

עבור ה train:

```
train acc:  0.859271523178808
```

	0	1
0	245	57
1	28	274

עבור ה val:

```
val acc:  0.799492385786802
```

	0	1
0	273	63
1	16	42

ניתן לראות כי מתקבל recall של:

$$recall = \frac{42}{42+16} = 0.724$$

אבל, קיימת דרך להעלות את ה recall על חשבון ה acc.

כעת, המודל מסווג לפי מה שמעל 0.5 כ- 1, ומה שמתחת ל- 0.

ניתן לשנות זאת בצורה הבאה:

```
y_val_pred = (clf_1_svm.predict_proba(X_val)[:,1] >= 0.3).astype(bool)
```

כעת, נקבל:

```
val acc:  0.7385786802030457
```

	0	1
0	238	98
1	5	53

כלומר, ה acc ירד בכ 6 אחוזים, אבל ה recall השתנה ל- $recall = \frac{53}{53+5} = 0.913$

כלומר, קיבלנו שיפור משמעותי ב recall.

במשימה שקיבלנו רשום שבין היתר אנו נבדקים על סמך ה acc שקיבלנו על ה test. לכן, לא נגיש את המודל המוטה, אלא נגיש מודל הרגיל (מסווג לפי מה שמעל 0.5 כ- 1, ומה שמתחת ל- 0).

כעת נבצע Hyperparameter tuning עבור המודל.
לאחר שיצרנו את מודל ה SVM הבסיסי, בדקנו תחילה את הפרמטרים שיש לו, על מנת שנדע אילו פרמטרים אנו יכולים לשנות ב GridSearch.

מצאנו שהפרמטרים שיש ל SVM הבסיסי הם:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': True,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

לאחר חיפוש באינטרנט (אתר sklearn), החלטנו שיהיה הכי טוב ליצור את ה grid מהפרמטרים הבאים:

```
param_grid= {'kernel': ('linear', 'rbf'),
             'C': [1, 10, 100],
             'tol':[0.00001,0.001,0.0003,0.01]}
```

לאחר שהגדרנו את ה grid, ביצענו GridSearch על מנת לקבל את המודל הטוב ביותר מבחינת roc_auc_ovr_weighted.

כלומר, לפי השטח שמתחת לגרף ה- ROC.

לאחר ריצות ובדיקות רבות, מצאנו כי הפרמטרים של המודל הטוב ביותר, מתוך אלו שנבדקו הם:

```
{'C': 10, 'kernel': 'rbf', 'tol': 1e-05}
```

כעת, בדקנו מהי מידת השיפור של המודל החדש על המודל המקורי ביחס לדיוק עבור ה- validation set.

הגענו לתוצאות הבאות:

```
grid model accuracy = 79.95%

Improvement of 1.29%.
```

כלומר, התקבל שיפור, אבל הוא לא שיפור גדול מאוד.

כעת, בדקנו פרמטרים נוספים והם:

	precision	recall	f1-score	support
0.0	0.93	0.83	0.88	336
1.0	0.39	0.62	0.48	58
accuracy			0.80	394
macro avg	0.66	0.73	0.68	394
weighted avg	0.85	0.80	0.82	394

כלומר, ה recall שקיבלנו עבור target=1 הוא לא גבוה.

השוואה בין המודלים

ביצענו השוואה של שלושת המודלים, על ידי שימוש באלגוריתם K-Folds cross-validation.
בכל פעם ביצענו אימון של המודל על K-1 סטים מתוך הדאטא, ובדקנו את התוצאות עבור הסט האחר.

קיבלנו:

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5,), random_state=1,
              solver='lbfgs')
mean accuracy:  0.8191364992340564
mean recall:  0.7755126214365344
mean AUC:  0.8773873793709285
DecisionTreeClassifier(max_depth=17, max_features=10, min_samples_leaf=2,
                       min_samples_split=19, random_state=42)
mean accuracy:  0.7379908086753206
mean recall:  0.5722728539576367
mean AUC:  0.7340515718793751
SVC(C=10, probability=True, tol=1e-05)
mean accuracy:  0.8280738531000565
mean recall:  0.7119035010882836
mean AUC:  0.8874663592011934
```

כלומר מסווג ה- SVM קצת יותר טוב ב- accuracy מאשר רשת הנוירונים. עם זאת, ערך ה- recall שלו פחות טוב ביחס לרשת הנוירונים.

לאחר מכן, ביצענו t-test בין ה accuracy של מודלי הסיווג SVM ו- Neural Network.
קיבלנו p-value: 0.158, כלומר השערת ה 0 התקבלה.

כמו כן, ביצענו t-test על ה- recall וה AUC של שני המודלים, וקיבלנו בהתאמה:

p-value: 0.998

p-value: 0.08

לכן, בכל המקרים, התקבלה השערת ה 0.

מתוך תוצאותינו של קבלת השערת האפס נסיק כי המובהקות הסטטיסטית של מודלי הסיווג הינה טובה. וזאת מכיוון שעבור סט הוולידציה, אין סתירה בין המודלים, כלומר בממוצע, המודלים הוציאו תוצאות חיזוי דומות עבור סט וואלידציה זהה. כלומר, המובהקות הסטטיסטית גבוהה. מכאן נוכל להסיק כי נקבל ערכי תוצאות חיזוי דומות גם על ה- testset.

תוצאות

מכיוון שמבין כל המודלים שבנינו, המודל עם ה accuracy הכי טוב עבור ה- validation set הוא רשת הנוירונים, סיווגנו את ה test בעזרתה.
בהשוואה למודל ה- SVM בו קיבלנו תוצאות קצת יותר טובות, אבל recall פחות טוב ולכן החלטנו לבחור ברשת הנוירונים.
עבור עץ ההחלטות קיבלנו תוצאות recall ו- accuracy פחות טובות ולכן, החלטנו שלא להשתמש בו.

לפני הכנסת ה- testset לתוך רשת הנוירונים, ביצענו עליו תהליך preprocess בדומה לזה של ה- trainset.
ביצענו זאת תוך התחשבות בכך ש- כמות ה features, טווח הערכים שלהם, וסוגם יהיו זהים לאלו של ה- train.

לאחר שביצענו את ה preprocess, הכנסנו את המידע לרשת.

קיבלנו חזרה את וקטור הפרדיקציה אשר מכיל אפסים ואחדות.

כדי להבחין בין חיזוי רשומה מסוימת לרשומה אחרת בסט המבחן קובץ ה- csv של הפרדיקציות מכיל את מאפיין ID (זאת בשונה מסט האימון שם הסרנו feature זה) אשר ייחודי עבור כל רשומה ורשומה על פי פרמטר זה ניתן יהיה לראות את אחוז הדיוק ביחס לתוצאות.

כמו כן, מצורפים קבצי ה- csv לאחר preprocess של ה- train/test dataset בתוך תיקיית data.
בנוסף, צירפנו בנפרד את מחברת ה- preprocess של ה- testset בתיקייה test_preprocess בתוך תיקיית data.