הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
המחלקה להנדסת תעשיה
אוניברסיטת תל אביב
TEL AVIV UNIVERSITY

מבוא ללמידת מכונה – תש"ף (2019-20)

פרויקט מסכם

מרצה: ד"ר אייל קולמן

מתרגלת: שי עובר

מגישים: אולגה סולדטנקו – 342480308

יליזבטה גולדברג – 337762462

קבוצה 03

תקציר מנהלים:

בפרויקט זה עסקנו בבעיית Binary Classification בה סיווגנו רשומות לשתי קטגוריות על סמך מספר פיצ'רים בדאטה סט. ראשית חקרנו את הנתונים – אופי התפלגות הפיצ'רים, התנהגות קורלטיבית בין הפיצ'רים, נתונים סטטיסטים. כמו כך בנינו מספר גרפים שונים המשמשים לוויזואליזציה של נתונים ושיפור הבנתם. לאחר מכן ביצענו שלב של עיבוד מקדים – הסרת נתונים חריגים, נרמול נתונים, השלמת נתונים חסרים, הקטנת ממדיות של הבעיה ובניית פיצ'רים חדשים. כל זה על מנת להכין את הדאטה לשלב של הרצת מודלים. בנינו 4 מודלים – 2 בסיסיים (KNN, Logistic Regression) ו-2 מתקדמים (ANN, Adaptive Boosting). על חלק מהדאטה הרצנו את המודלים (train) ועל חלק שני (validation) ביצענו פרדיקציה וערכנו טיב המודלים. קיבלנו כי חודל Logistic Regression בעל מדדים הכי טובים ולכן השתמשנו בו לסיווג הנתונים הסופי.

חלק ראשון – אקספלורציה

בחלק זה ביצענו בדיקת הנתונים. בעיה בעלת 22161 דגימות ו-25 פיצ'רים. מהם 6 קטגוריאליים והשער מספריים. משתנים קטגוריאליים – מופעים בפיצ'רים מס' 5, 6, 9, 14, 18, 19. פיצ'ר 13 על פניו נראה קטגוריאלי אך בפועל הוא בשלמים. פיצ'ר 14 גם הפיך לפיצ'ר נומרי בקלות על ידי הסרת 'mm' מכל הדגימות.

ראינו שאין העתקים של אותה דגימה.

בנינו את Heat map וראינו שקיימת קורלציה חיובית גבוהה (יותר מ-0.85) ביון פיצ'רים מסוימים: [0,1,2], [7,16], [7,17], [8,17], [16,17]. לא קיימים פיצ'רים בעלי קורלציה שלילית גבוהה. בנוסף לחלק מהפיצ'רים בעלי קורלציה גבוהה בנינו scatter plots כדי לראות את האופן תלות ביניהם.

עבור הפיצ'רים הנומריים בוצעו היסטוגרמות ו-Boxplots לפי כל קלס כדי להבין איך הערכים מתפלגים. לפי גרפים ניתן לראות כי הנתונים לא מנורמלים - עבור פיצ'רים נדרש לשנות את הסקלה של הנתונים (טווחי ערכים שונים בפיצ'רים שונים) כמו כן קיימים פיצ'רים עם נתונים חריגים.

עבור פיצ'רים קטגוריאליים בנינו גרפים שכיחויות.

דוגמאות של הגרפים ניתן לראות בנספח 1.

חלק שני – עיבוד מקדים:

- <u>נתונים חריגים</u> – באקספלורציה ראינו שרוב הפיצ'רים הנומריים הם בעלי פילוג שיש בו מרכז וההסתברות יורדת ככל שמתרחקים ממנו וקיימים נתונים חריגים, לכן בחרנו להסיר כל הדגימות אשר עולות על 3 סטיות התקן.

- <u>נרמול נתונים</u> – לנרמול השתמשנו ב z-score normalization. נרמול חשוב לעבודה טובה של רוב האלגוריתמים של למידת מכונה - הם משתמשים במרחקים אוקלידיים.
- <u>טיפול בנתונים חסרים</u> – החלטנו בכלל לא להשתמש בפיצ'רים שחסרים להם יותר מ 5% מהנתונים. כבר בשלב זה הסרנו פיצ'רים 5 ו-15. לכל שאר הפיצ'רים השלמנו דגימות חסרות בממוצה (mean). בפיצ'ר 13 מכיוון שערכיו שלמים הוחלט להסיר את הדגימות הלא ידועות (פחות מ 5% מה dataset).
- <u>הקטנת ממדיות</u> – ממדיות הבעיה לא גדולה מידי, אחרי הורדת פיצ'רים קטגוריאליים נשארו רק 19. זה טוב לגודל הדאטה סט שיש לנו. אין סכנה של curse of dimentionality. בכל זאת ניסינו קצת להקטין את הממדיות לריצה יותר מהירה וחסכון של זיכרון. הורדנו פיצ'רים בעלי קורלציה גבוהה אשר גילינו בשלב האקספלורציה (פיצ'ר 1 ירד).
בזמן אימון המודלים וניסיונות השתמשנו ב-PCA, לקחנו רק את הפיצ'רים הראשונים אשר ביחד מסבירים את 85% מהנתונים (expected variance), אבל ראינו ששימוש ב-PCA מוריד את הביצועים לכן במודלים סופיים הוחלט לא להשתמש בו.
- <u>בניית פיצ'רים חדשים</u> – הוחלט לבנות פיצ'רים חדשים מפיצ'רים שכבר קיימים. עשינו סכום של פיצ'רים 8 ו-17 ופיצ'רים 0 ו-2 (פיצ'רים בעלי קורלציה גבוהה בניהם). בנוסף השתמשנו ב-KFold Clustering כדי לשייך דגימות לקבוצות שונות. ראינו שהדאטה מתחלק הכי טוב ל 3 קלסתרים (לפי שיטת Elbow plot). הוספנו לדאטה של train שורה עם מספר קלסתר שאליו דגימה השתייכה ('clustering').
- <u>החלת העיבוד המקדים על סט ה-test</u> – בשביל סיווג נכון סט הטסט צריך להיראות כמו סט שעליו אומנו המודלים. בשביל זה עושים נרמול, PCA (אם מחליטים להשתמש בו), מסירים את הפיצ'רים ובונים את הפיצ'רים החדשים (קומבינציות ו-Clustering) בדיוק כמו בסט האימון.

לכל הבדיקות והרצות השתמשנו באובייקט Pipeline שבנינו בעצמנו. הוא מאפשר להריץ גרסאות שונות של הבעיה ללא הרצה ידנית ולשמור את כל הנתונים בטבלאות נוחות. הקוד אפשר לראות בנספח 2.

<u>חלק שלישי ורביעי – הרצת המודלים והערכתם:</u>

עבור מודלים ראשוניים בחרנו לא להשתמש ב- Naïve Bayes Classifier משום שמודל זה מניח אי-תלות של הפיצ'רים, אשר ככל הנראה לא נכון במקרה שלנו. לכן ממשנו את KNN ו- Logistic Regression.
במודלים מתקדמים בחרנו לממש Adaptive Boosting ורשתות נוירונים (ANN) אשר לדעתנו מתאימות הכי טוב למבנה הנתונים שלנו.
הערכת מודלים התבצעה על ידי השוות מדד AUC ובדיקת overfitting. על מנת להגדיל יכולת הכללה של המודלים השתמשנו ב-K-Fold Cross Validation כאשר מספר K-Folds האופטימלי הוא 5. הגדנו כי מודל נחשב overfitted כאשר פערי ביצוע על ה-Train ועל ה-Validation עולים מעל 5%. גרפים של K-Fold ניתן לראות בנספח 3.

בנוסף עבור כל מודל חשבנו מדד דיוק (Accuracy) משוקלל. מדד זה צריך להתייחס לעלות סיווג שגוי אשר חמורה פי 5 לדגימה בעלת תיוג "1":

$$(w_{tp}, w_{fp}, w_{tn}, w_{fn}) = (1,1,1,5)$$

$$\frac{w_{tp}TP + w_{tn}TN}{w_{tp}TP + w_{tn}TN + w_{fp}FP + w_{fn}FN}$$

במקרים קצה (כל הנבואות נכונות, כל הנבואות שגויות) מדד זה מקבל ערכים 1 ו-0 בדומה למדד הדיוק רגיל. במקרה ויש יותר סיווגים שגויים של מדדים בעלי סיווג 1 מדד חדש מקבל ארך יותר קטן.

- <u>KNN</u> - למציאת פרמטרים למודל השתמשנו ב-Grid Search.
  פרמטרים שהתקבלו: 'metric= 'euclidean', n_neighbors = 100, weights='uniform
  מדד AUC שהתקבל: 0.84. קיבלנו הבדל של 1.5% בין Train ל-Validation ולכן ניתן להסיג שלא קיים overfitting. מדד דיוק משוקלל – 0.54. מדד זה יצאה נמוך מכיוון שכמות הטעויות מסוג false negative יותר גדולה מ-false positive.

- <u>Logistic Regression</u> - למציאת פרמטרים למודל השתמשנו ב-Grid Search.
  הפרמטרים שהתקבלו: C=0.5, max_iter=300, penalty='l1',
  tol=0.0001 'solver='liblinear.
  מדד AUC שהתקבל: 0.862. קיבלנו הבדל של 0.09% בין Train ל-Validation ולכן ניתן להסיג שלא קיים overfitting. מדד דיוק משוקלל – 0.6. מדד זה יצאה נמוך מכיוון שכמות הטעויות מסוג false negative יותר גדולה מ-false positive.

- <u>ANN</u> - למציאת פרמטרים למודל השתמשנו ב-Random Search.

  הפרמטרים שהתקבלו: max_iter = 1000, learning_rate_init = 0.01,
  'hidden_layer_sizes = (100,),batch_size = 10, activation = 'relu

  מדד AUC שהתקבל:0.856. קיבלנו הבדל של 2.2% בין Train ל-Validation ולכן ניתן להסיג שלא קיים overfitting. מדד דיוק משוקלל - 0.6. מדד זה יצאה נמוך מכיוון שכמות הטעויות מסוג false negative יותר גדולה מ-false positive

- <u>Adaptive Boosting</u> - למציאת פרמטרים למודל השתמשנו ב-Grid Search.

  הפרמטרים שהתקבלו: learning_rate=0.1 ,n_estimators=300, random_state=0.

  מדד AUC שהתקבל:0.857. קיבלנו הבדל של 0.41% בין Train ל-Validation ולכן ניתן להסיג שלא קיים overfitting. מדד דיוק משוקלל - 0.6. מדד זה יצאה נמוך מכיוון שכמות הטעויות מסוג false negative יותר גדולה מ-false positive.

- <u>Confusion Matrix</u> – בעזרתה ניתן לעריך איכות הפלט של המודל.

לדוגמה מופיע מטריצה עבור מודל Logistic Regression בנספח 3.
לאלן הסבר על מרכיבי המטריצה:

TP 379 תצפיות היו 1 ומודל חזה 1	FP 159 תצפיות היו 0 ומודל חזה 1
FN 517 תצפיות היו 1 ומודל חזה 0	TN 3918 תצפיות היו 0 ומודל חזה 0

מהמטריצה ניתן להסיג כביצועי המודל טובים אך יש מספר די גדול של טעויות מסוג false negative.

<u>חלק חמישי – ביצוע פרדיקציה:</u>

בחרנו להשתמש במודל Logistic Regression לפרדיקציה מכיוון שהיא בעלת AUC הכי גבוה.

<u>סיכום:</u>

בפרויקט זה בנינו 4 מודלים. ראינו שכל המודלים נותנות תוצאות טובות, אין overfitting. למרות זאת מדד דיוק משוקלל יחסית נמוך, מודלים מחזירים הרבה שגיאות false negative. מכיוון שנדרשנו לבחור מודל לפי ROC החלטנו לא לנסות לשפר מדד זה.

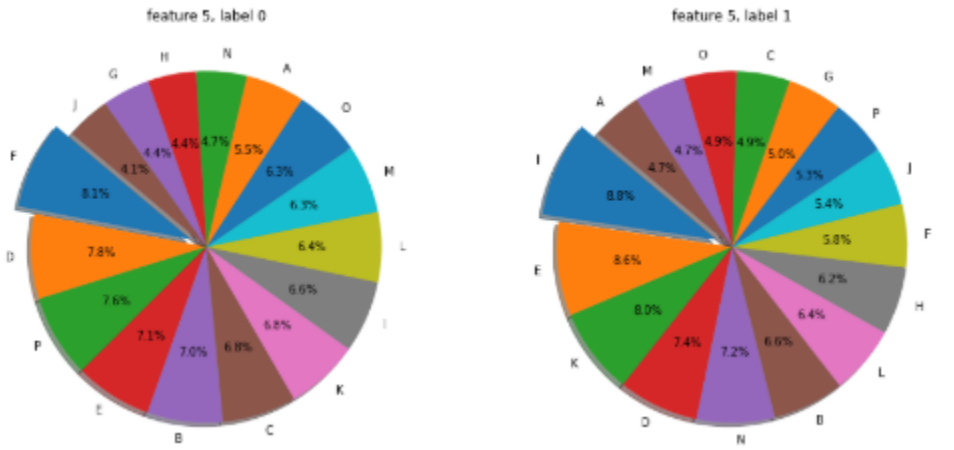
במהלך ניתוח נתונים ראינו כי היה נירש לבצע הרבה פעולות בעיבוד מקדים, אך לא קיבלנו ידע מספק לגבי פיצ'רים ודאטה ולכן ניתן לומר שלא נעשה עיבוד מקדים בצורה המיטבית ולכן תוצאות המודלים ניתנים לשיפור.

קיבלנו 3 מודלים (ANN, Logistic Regression, Adaptive Boosting) בעלי מדדי ביצוע מאוד דומים: AUC קצת יותר טוב ל-Logistic Regression , אך שווה בערך ל-0.86 עבור שלושת המודלים; מדד דיוק משוקלל אומד על 0.6 עבור שלושת המודלים; פערי ביצוע בין test ו-train הקטן ביותר עבור מודל Logistic Regressio.

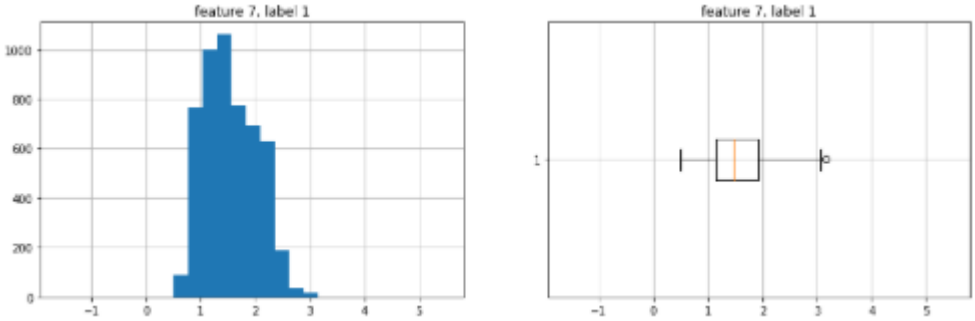
לבסוף בחרנו במודל Logistic Regression כמסווג הכי טוב לפי מטריקת AUC ובשכלול מדדים נוספים.

נספחים

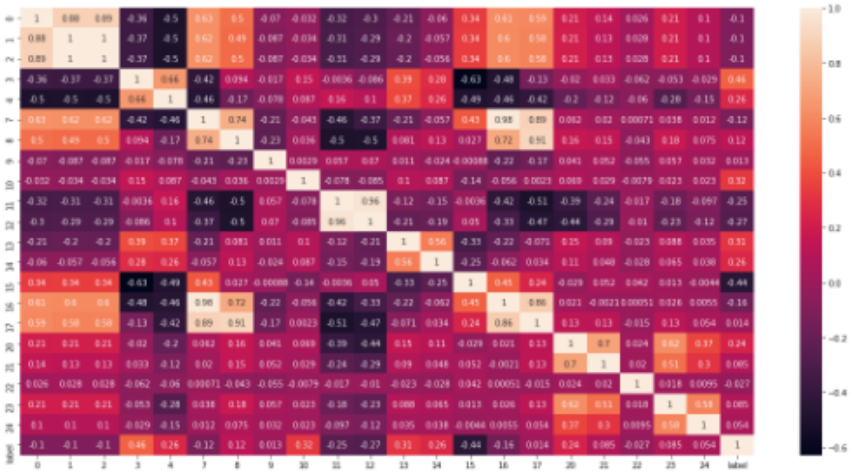
נספח 1: אקספלורציה – דוגמאות לגרפים שונים



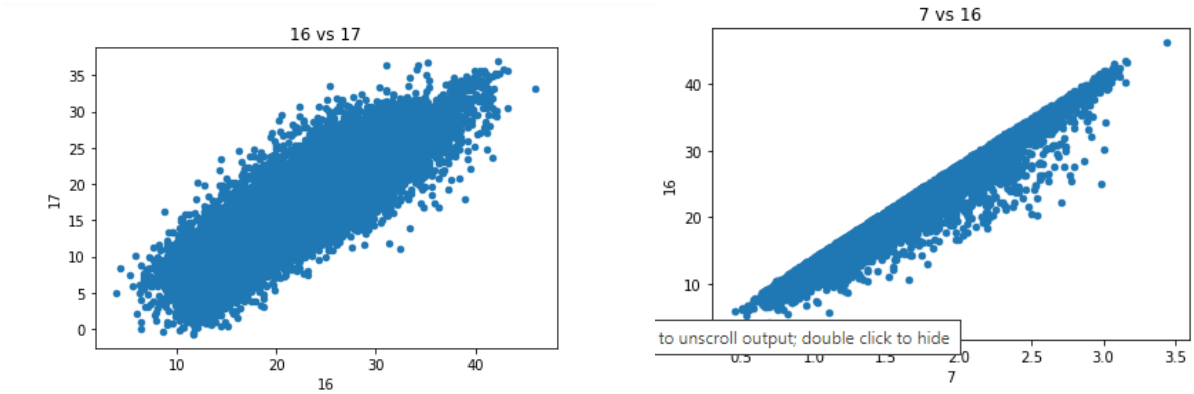
איור 1: pieplot עבור פיצ'ר קטגוריאלי (5)



איור 2: boxplot ו-hisrogram עבור פיצ'ר נומרי (7)



איור 3: Heatmap עבור סט נתונים train



איור 4: גרף עבור פיצ'רים בעלי קורלציה גבוהה (7 ו-16, 16 ו-17)

נספח 2: עיבוד מקדים

אובייקט Pipeline

```
class Pipeline:
    def __init__(self, normalization = 1):
        self.one_hot_spot = 0
        self.combine_features = 0
        self.drop_features = 0
        self.anova = 0
        self.normalization =  normalization
        self.model = 'KNN'
        self.outliers = 3
        self.pca_n_components = 0
        self.combine_features = 0
        self.clustering = 0

    def set_combine_features(self, features):
        temp = copy.copy(self)
        temp.combine_features = features
        return temp

    def set_drop_features(self, features):
        temp = copy.copy(self)
        temp.drop_features = features
        return temp

    def set_remove_outliers(self, z):
        temp = copy.copy(self)
        temp.outliers = z
        return temp

    def set_clustering(self, n_clusters = 3):
        temp = copy.copy(self)
        temp.clustering = n_clusters
        return temp

```

```
    def set_PCA(self, n_components):
        temp = copy.copy(self)
        temp.pca_n_components = n_components
        return temp

    def set_model(self, model):
        temp = copy.copy(self)
        temp.model = model
        return temp


    def run_pipeline(self, X,y,i,lenth, start):
        print('Starting ', i ,'out of ',lenth)

        if self.combine_features:
            for feature in self.combine_features:
                X = combine_features(X,feature[0],feature[1])
            print('Combining features - done')

        if self.drop_features:
            X = X.drop(self.drop_features,axis = 1)
            print('Dropping features - done')

        if self.outliers:
            X,y  = Outliers(X,y)
            print('Outliers - done')

        X = Filling_the_missing_data(X)
        print('Filling_the_missing_data - done')

        if self.clustering:
            X = K_means(X)
            print('Clustering - done')

        if self.normalization:
            X, std = Normalization(X)
            print('Normalization - done')

        if self.pca_n_components:
            X, pca_clf = our_PCA(X,self.pca_n_components)
            print('PCA - done')

        X_train, X_valid, y_train, y_valid = split_test(X,y)

        if self.model == 'KNN':
            print('Entered KNN')
            res_KNN = KNN(X_train,y_train)
            self.best_estimator = res_KNN.best_estimator_
            self.auc = res_KNN.best_score_

        elif self.model == 'LogReg':
            print('Entered LogReg')
            res_LogReg = LogReg(X_train,y_train)
            self.best_estimator = res_LogReg.best_estimator_
            self.auc = res_LogReg.best_score_
```

```
89.
90.         elif self.model == 'AdaBoost':
91.             print('EnteredAdaBoost')
92.             res_AdaBoost = AdaBoost(X_train,y_train)
93.             self.best_estimator = res_AdaBoost.best_estimator_
94.             self.auc = res_AdaBoost.best_score_
95.
96.         elif self.model ==='AdaBoost_SVC':
97.             print('EnteredAdaBoost_SVC')
98.             res_AdaBoost_SVC = AdaBoost_SVC(X_train,y_train)
99.             self.best_estimator = res_AdaBoost_SVC.best_estimator_
100.                self.auc = res_AdaBoost_SVC.best_score_
101.
102.             elif self.model == 'ANN':
103.                print('ANN')
104.                res_ANN = ANN(X_train,y_train)
105.                self.best_estimator = res_ANN.best_estimator_
106.                self.auc = res_ANN.best_score_
107.
108.            stop = timeit.default_timer()
109.            print('Time: ', (stop - start)/60)
110.            self.execution_differences,self.auc_train =
    execution_differences(X_train, y_train,X_valid, y_valid,
    self.best_estimator)
```

דוגמה של שימוש ב Pipeline:

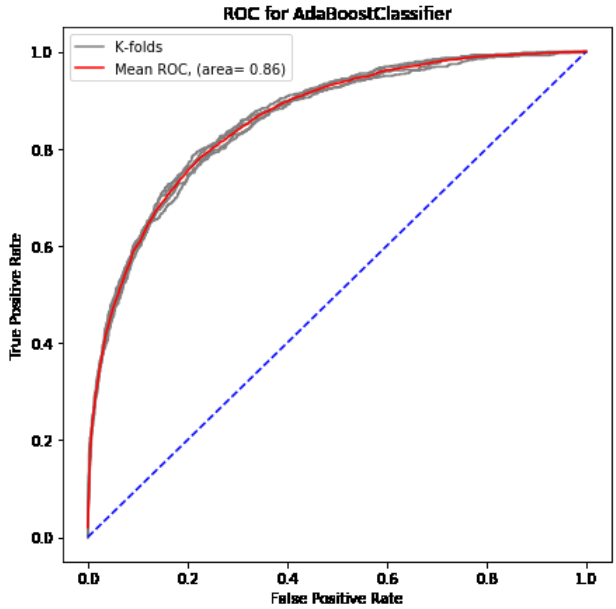
```
111.      #defining parameters
112.      Params_combine_features_1 = [[('8','17'),('0','2')],[('0','2')]]
113.      Params_drop_features_1 = [['16'],['12']]
114.      Params_PCA = [0,0.9]
115.
116.      #creating list of pipelines
117.      Pipelines = [Pipeline().set_model('LogReg')]
118.      Pipelines = [P.set_combine_features(Param) for P,Param in
    product(Pipelines,Params_combine_features_1) ]
119.      Pipelines = [P.set_drop_features(Param) for P,Param in
    product(Pipelines,Params_drop_features_1) ]
120.      Pipelines = [P.set_PCA(Param) for P,Param in
    product(Pipelines,Params_PCA) ]
121.
122.      #control time
123.      start = timeit.default_timer()
124.      i=1
125.      lenth = len(Pipelines)
126.
127.      #run the pipelines
128.      for P in Pipelines:
129.          dat = P.run_pipeline(X,y,i,lenth,start)
130.          i+=1
131.          Results_df = Results_df.append(P.__dict__, ignore_index=True)
132.
133.      stop = timeit.default_timer()
```

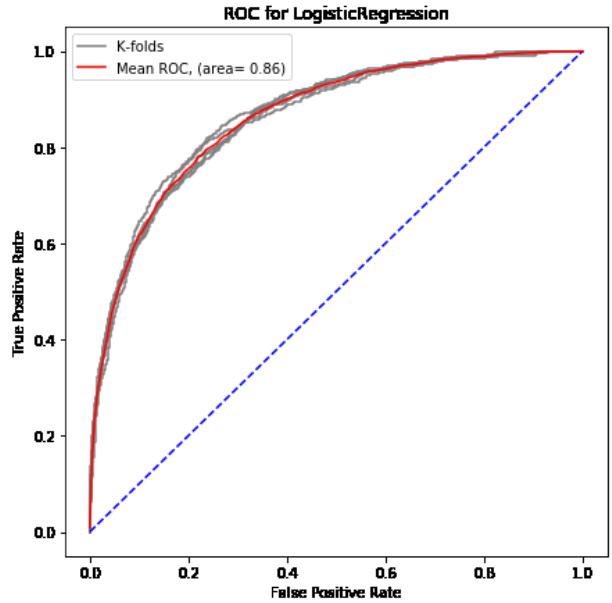
```
print('Time: ', (stop - start)/60)
```

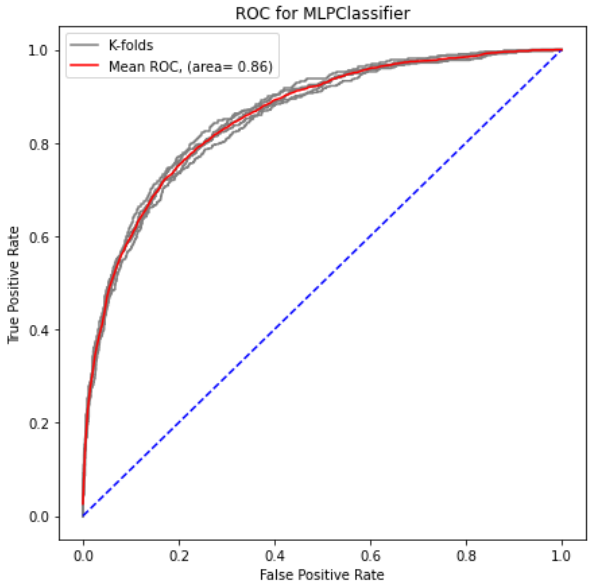
דוגמה של טבלה עם תוצאות של Pipeline:

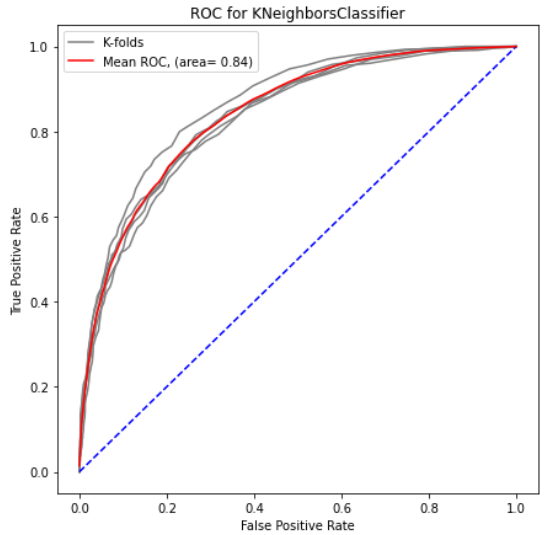
	drop_features	Combine_features	normalization	model	outliers	pca	clustering	auc
0	['16']	[('2' ,'0') ,('17' ,'8')]	1	LogReg	3	0	0	0.8635
1	['16']	[('2' ,'0') ,('17' ,'8')]	1	LogReg	3	0.9	0	0.8313
2	['16']	[('2' ,'0') ,('17' ,'8')]	1	LogReg	3	0	3	0.8634
3	['16']	[('2' ,'0') ,('17' ,'8')]	1	LogReg	3	0.9	3	0.8310
4	['12']	[('2' ,'0')]	1	LogReg	3	0	0	0.8601
5	['12']	[('2' ,'0')]	1	LogReg	3	0.9	0	0.8389
6	['12']	[('2' ,'0') ,('17' ,'8')]	1	LogReg	3	0	3	0.8601
7	['12']	[('2' ,'0') ,('17' ,'8')]	1	LogReg	3	0.9	3	0.8388

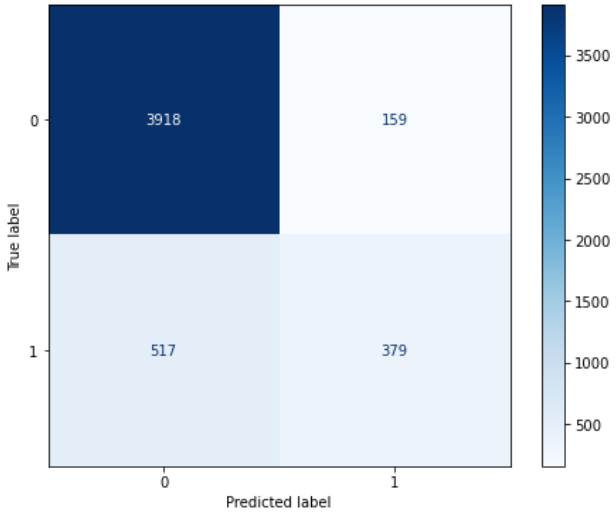
נספח 3: הערכת המודלים











איור 5: Confusion matrix for logistic regression