קורס בריית מידע - 20595

ממ"ן 22

תאריך הגשה: 9.6.2023

קבוצה: 82

למרצה: רועי רחמני

מגיש: עידן קוגן

ת.ז: 316375591

שאלה 1

```python
from sklearn.naive_bayes import GaussianNB
# Search for the best parameters for Naive Bayes Classifier
nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=nb_classifier,
                     param_grid=params_NB,
                     cv=10,
                     verbose=1,
                     scoring='accuracy')
gs_NB.fit(X_train, y_train)
gs_NB.best_params_
```

```python
# Initialize the Naive Bayes Classifier
naive_bayes_model = GaussianNB(var_smoothing=0.004328761281083057)

# Perform cross validation with 10 folds with sklearn class
naive_bayes_k_fold = KFold(n_splits=10, shuffle=True, random_state=42)
naive_bayes_k_fold.get_n_splits(X_train)
folds_y_tests, predictions_of_all_folds = [], []
train_scores, test_scores = [], []
confusion_matrix_list = []
for train_index, test_index in naive_bayes_k_fold.split(X_train):
    # Define the train and test sets for each fold
    X_train_k_fold, X_test_k_fold = X_train[train_index], X_train[test_index]
    y_train_k_fold, y_test_k_fold = y_train.iloc[train_index], y_train.iloc[test_index]
    # Fit the model and predict of current fold
    naive_bayes_model.fit(X_train_k_fold, y_train_k_fold)
    # Calculate the scores on test set
    y_pred = naive_bayes_model.predict(X_test_k_fold)
    # Append the y_test and y_pred to the lists for the classification report
    folds_y_tests.extend(y_test_k_fold)
    predictions_of_all_folds.extend(y_pred)
    # Append the scores to the lists
    train_scores.append(naive_bayes_model.score(X_train_k_fold, y_train_k_fold))
    test_scores.append(naive_bayes_model.score(X_test_k_fold, y_test_k_fold))
    confusion_matrix_list.append(confusion_matrix(y_test_k_fold, y_pred))

# Print the train and test scores with a column of the number of fold
print('Naive Bayes 10-Cross Validation Results:')
print(tabulate.tabulate({'Fold': range(1, 11),
                         'Train scores': train_scores,
                         'Test scores': test_scores}, headers='keys', tablefmt='psql'))

# Print the mean of the train and test scores
print(f'Mean train score: {np.mean(train_scores)}')
print(f'Mean test score: {np.mean(test_scores)}')
```

א. בחרתי באלגוריתם Naïve Bayes מכיוון שהוא קל למימוש בפייתון וכי אחוזי ההצלחה שלו גבוהים מספיק. הוא עמיד לנקודות רועשות ומבודדות - מאחר שנקודות אלה ימוצעו (מלשון ממוצע) בעת שערוך ההסתברויות המותנות מהנתונים.
עם זאת אציין שקיים חסרון משמעותי בשיטה: הנחת אי תלות מותנת - במידה וכן מתקיימת תלות מסוימת בין המשתנים אז הסיווג יהיה פחות מדויק. באופן מעשי אפשר למצוא הרבה תלויות בין משתנים, ובמצב זה הנאייב בייס לא יעבוד טוב.

ראשית אמצא את הפרמטר הטוב ביותר עבור נאייב בייס בעזרת grid search המוצא את הפרמטר הטוב ביותר עבור בסיס הנתונים שלנו.
הפרמטר var_smoothing שולט בכמות ההחלקה המוחלת על התפלגויות גאוסיאנית של מודל הנתונים. ערך גבוה יותר של var_smoothing יביא להחלקה רבה יותר, מה שיכול לעזור במניעת התאמת יתר (Overfitting) ולשפר את ביצועי ההכללה של המודל. עם זאת, ערך גבוה מדי של var_smoothing יכול גם להוביל לחוסר התאמה (underfitting) של המודל לנתונים.
אאתחל את המודל ואבצע 10 cross validation כדי לקבל את ההצלחה שלו.

<u>שלב האימון:</u>
א. אסוף מערך נתונים מסומן המכיל תצפיות והסיווגים התואמים להן.
ב. חשב את ההסתברות הפריורית של כל מחלקה על ידי חלוקת מספר התצפיות בכל מחלקה במספר הכולל של התצפיות.
ג. חשב את ההסתברויות המותנות של כל תכונה בהינתן כל סיווג על ידי ספירת המופעים של ערכי תכונה בתוך כל מחלקה.

<u>שלב הסיווג:</u>
א. קבל תצפית ללא תווית.
ב. חשב את ההסתברות של התצפית להיות שייכת לסיווג כלשהו על ידי הכפלת ההסתברויות המותנות של התכונות השונות.
ג. הכפל את ההסתברויות בהסתברויות הקודמות שלהם.

ב. אציג את הפתרון לבעיה הנתונה תוך שימוש באלגוריתם K-NN. אסביר בקצרה על השיטה:
בשיטת KNN במקום לעבד את הנתונים - שומרים את הנתונים כמו שהם. זו שיטה עצלנית lazy evaluation. כאשר מגיעה תצפית חדשה, משתמשים באותן תצפיות ששמרנו על מנת לסווג את התצפית החדשה. התצפיות מיוצגות כנקודות במרחב אוקלידי בעל n מימדים ודמיון השכנים מתצפית כלשהי מוגדר במרחק אוקלידי.
פונקציית המטרה בדידה או רציפה ואפשר לחזות גם ערכים בדידים וגם ערכים רציפים.

```
############# K Nearest Neighbors #############
from sklearn.neighbors import KNeighborsClassifier
# Search for the best parameters for KNN Classifier
knn_classifier = KNeighborsClassifier()

params_KNN = {'n_neighbors': np.arange(1, 25)}
gs_KNN = GridSearchCV(estimator=knn_classifier,
                      param_grid=params_KNN,
                      cv=10,
                      verbose=1,
                      scoring='accuracy')
gs_KNN.fit(X_train, y_train)
gs_KNN.best_params_
```

ראשית, אבצע grid search כדי למצוא את הפרמטר n_neighbors שימצא את מספר השכנים האופטימלי כדי למזער את השגיאה / למקסם את דיוק המודל על בסיס הנתונים שלנו.

אאתחל את המודל ואבצע 10 cross validation כדי לקבל את ההצלחה שלו:

```
# Initialize the KNN Classifier
knn_model = KNeighborsClassifier(n_neighbors=9)

# Perform cross validation with 10 folds with sklearn class
knn_k_fold = KFold(n_splits=10, shuffle=True, random_state=42)
knn_k_fold.get_n_splits(X_train)
folds_y_tests, predictions_of_all_folds = [], []
train_scores, test_scores = [], []
confusion_matrix_list = []
for train_index, test_index in knn_k_fold.split(X_train):
    # Define the train and test sets for each fold
    X_train_k_fold, X_test_k_fold = X_train[train_index], X_train[test_index]
    y_train_k_fold, y_test_k_fold = y_train.iloc[train_index], y_train.iloc[test_index]
    # Fit the model and predict of current fold
    knn_model.fit(X_train_k_fold, y_train_k_fold)
    # Calculate the scores on test set
    y_pred = knn_model.predict(X_test_k_fold)
    # Append the y_test and y_pred to the lists for the classification report
    folds_y_tests.extend(y_test_k_fold)
    predictions_of_all_folds.extend(y_pred)
    # Append the scores to the lists
    train_scores.append(knn_model.score(X_train_k_fold, y_train_k_fold))
    test_scores.append(knn_model.score(X_test_k_fold, y_test_k_fold))
    confusion_matrix_list.append(confusion_matrix(y_test_k_fold, y_pred))
```

ג. אשווה את דיוק המודלים:

	Model	Mean Squared Error	Test Score
0	Naive Bayes	0	1
1	KNN	0.0125	0.9875

ניתן לראות שדיוק Naïve Bayes גבוה יותר מאלגוריתם KNN ולכן אבחר בו.
שגיאה נמוכה משמעותה – מודל מדויק יותר.
שגיאה גבוהה משמעותה – מודל פחות מדויק.
Test Score == 1 משמעותו 100% הצלחה על נתוני המבחן.
Test Score < 1 משמעותו שהיו כמה טעויות סיווג.

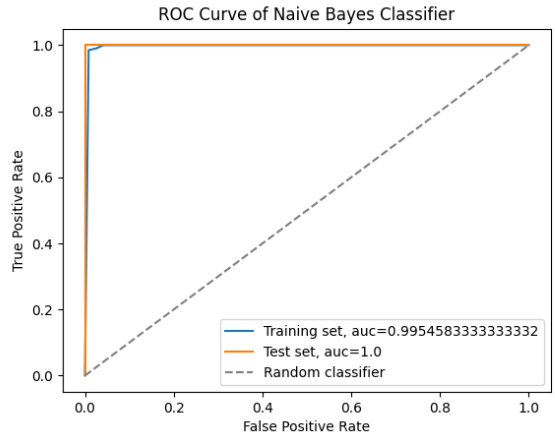
ד. <u>תוצאות נאייב בייס</u>

אציג את תוצאות כל FOLD בנפרד. ולאחר מכן אבצע ממוצע שלהם. תוצאות הדיוק יהיו גם עבור נתוני האימון הנוכחית של הFOLD הנוכחי וגם עבור קבוצת הוואלידציה של הFOLD הנוכחי.
בנוסף אציג את ה-confusion matrix.
ואת הROC curve.

```python
# Print the train and test scores with a column of the number of fold
print('Naive Bayes 10-Cross Validation Results:')
print(tabulate.tabulate({'Fold': range(1, 11),
                         'Train scores': train_scores,
                         'Test scores': test_scores}, headers='keys', tablefmt='psql'))

# Print the mean of the train and test scores
print(f'Mean train score: {np.mean(train_scores)}')
print(f'Mean test score: {np.mean(test_scores)}')

# Sum all the confusion matrices from the cross validation
confusion_matrix_sum = np.sum(confusion_matrix_list, axis=0)

# Print the confusion matrix with tabulate
print('Confusion Matrix:')
print(tabulate.tabulate({'': ['Actual Positive', 'Actual Negative'],
                         'Predicted Positive': [confusion_matrix_sum[0][0], confusion_matrix_sum[0][1]],
                         'Predicted Negative': [confusion_matrix_sum[1][0], confusion_matrix_sum[1][1]]},
                         headers='keys',
                         tablefmt='psql'))

# Print the classification report for the sum of the folds predictions
print('Classification Report of all folds:')
print(classification_report(folds_y_tests, predictions_of_all_folds))
```

```python
# Print the classification report for the sum of the folds predictions
print('Classification Report of all folds:')
print(classification_report(folds_y_tests, predictions_of_all_folds))

# Display the ROC curve of the Naive Bayes Classifier for comparing between the train and test sets
naive_bayes_model.fit(X_train, y_train)
y_pred_proba_train = naive_bayes_model.predict_proba(X_train)[:, 1]
y_pred_proba_test = naive_bayes_model.predict_proba(X_test)[:, 1]
plot_roc_curve(y_pred_proba_train, y_pred_proba_test, y_train, y_test, 'Naive Bayes Classifier')

# Predict the test set
y_pred = naive_bayes_model.predict(X_test)

# Calculate the Mean Squared Error of the cross validation
mse_naive_bayes = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mean_squared_error(y_test, naive_bayes_model.predict(X_test))}')

# print test score of the test set (not cross validation)
print(f'Test score: {naive_bayes_model.score(X_test, y_test)}')
```

```
Naive Bayes 10-Cross Validation Results:
+--------+----------------+---------------+
|   Fold |   Train scores |   Test scores |
|--------+----------------+---------------|
|      1 |       0.986111 |       1       |
|      2 |       0.986111 |       1       |
|      3 |       0.989583 |       0.96875 |
|      4 |       0.986111 |       1       |
|      5 |       0.986111 |       1       |
|      6 |       0.986111 |       1       |
|      7 |       0.989583 |       0.96875 |
|      8 |       0.989583 |       0.96875 |
|      9 |       0.986111 |       1       |
|     10 |       0.989583 |       0.96875 |
+--------+----------------+---------------+
Mean train score: 0.9875000000000002
Mean test score: 0.9875
Confusion Matrix:
+-----------------+---------------------+---------------------+
|                 |   Predicted Positive |   Predicted Negative |
|-----------------+---------------------+---------------------|
| Actual Positive |                 119 |                   3 |
| Actual Negative |                   1 |                 197 |
+-----------------+---------------------+---------------------+
Classification Report of all folds:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       120
           1       0.99      0.98      0.99       200

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320

Mean Squared Error: 0.0
Test score: 1.0
```




```
KNN 10-Cross Validation Results:
+--------+----------------+---------------+
|   Fold |   Train scores |   Test scores |
|--------+----------------+---------------|
|      1 |       0.989583 |       1       |
|      2 |       0.993056 |       0.9375  |
|      3 |       0.989583 |       0.96875 |
|      4 |       0.993056 |       0.96875 |
|      5 |       0.989583 |       1       |
|      6 |       0.989583 |       0.96875 |
|      7 |       0.993056 |       0.96875 |
|      8 |       0.989583 |       1       |
|      9 |       0.989583 |       1       |
|     10 |       0.989583 |       1       |
+--------+----------------+---------------+
Mean train score: 0.990625
Mean test score: 0.98125
Confusion Matrix:
+-----------------+---------------------+---------------------+
|                 |   Predicted Positive |   Predicted Negative |
|-----------------+---------------------+---------------------|
| Actual Positive |                 118 |                   4 |
| Actual Negative |                   2 |                 196 |
+-----------------+---------------------+---------------------+
Classification Report of all folds:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       120
           1       0.99      0.98      0.98       200

    accuracy                           0.98       320
   macro avg       0.98      0.98      0.98       320
weighted avg       0.98      0.98      0.98       320

Mean Squared Error: 0.0375
Test score: 0.9625
```

לשם השוואה אציג גם את תוצאות K-NN.

ה. מ-confusion matrix של Naïve Bayes ניתן לראות שיש 3 נבדקים שסווגו כבריאים אך הם חולים במחלה כלייתית. ואילו יש נבדק אחד שסווג כחולה אך היה בריא.
כמו כן אפשר לראות שהצלחת המודל בfolds השונים היא גבוה ועומדת על 99% ~ 98.6%.
בנוסף, השטח שמתחת לעקומת ROC עומד על 99.54% עבור נתוני האימון ו100% עבור נתוני המבחן.
לאחר הסתכלות על גם נתוני התוצאות של KNN אפשר לראות שהמסווג הביסיאני מתאים יותר עבור בסיס הנתונים הזה.

שאלה 2

א. מדדי איכות לאשכולות

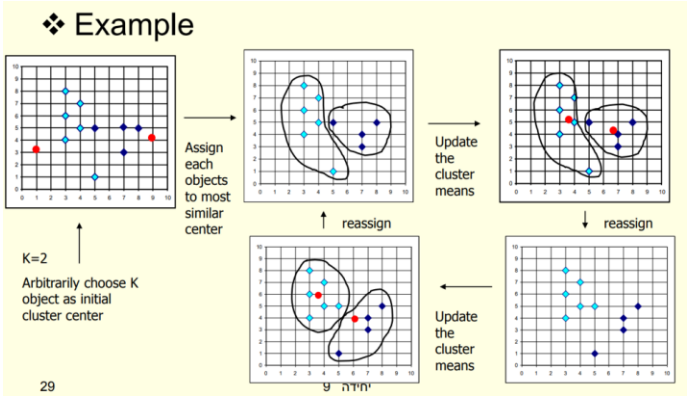
1. אפשר למדוד אשכולות בעזרת מדד דמיון של פונקציית דמיון/מרחק - similarity / distance function metric. הפונקציה מקבלת 2 אובייקטים ומחשבת כמה הם קרובים או רחוקים זה מזה.
לכל סוג משתנה יש פונקציית מרחק אחרת.
דוגמא לפונקציית מרחק הוא שימוש במרחק "מינקובסקי" המקבלת פרמטר q:
כאשר q=1 מכנים את המרחק "מרחק מנהטן" - מרחק בין 2 צמתים כאשר כל הרחובות בנויים בצורת שטיב הערב (מנהטן) - זוויות ישרות בין כל רחוב.
כאשר q=2 זהו מרחק אוקלידי ומשמעותו גאומטרית.
אלגוריתם קלאסטרינג יודע להתמודד גם עם מאגר נתונים בצורת מטריצה (טבלה רגילה) וגם בצורת ייצוג נוספת: מטריצת אי דמיון. במטריצת אי דמיון יש ערכים רק מתחת או רק מעל האלכסון והיא מתייחסת למרחק בין אובייקט i לאובייקט j. המרחק בין אובייקט לבין עצמו הוא 0, אז יש אפסים לאורך האלכסון.
ובגלל תכונת הסימטריות אין צורך להציג את הערכים שמעל האלכסון אם יש ערכים מתחת לאלכסון.
2. מדד איכות נוסף הוא פונקציה נוספת הבודקת את איכות החלוקה הנוכחית לאשכולות – quality function.
איכות או טיב האשכול תלויה במדד הדמיון בנתונים עצמם. האיכות של שיטת חלוקה לאשכולות נמדדת ביכולת שלה לגלות תבניות נסתרות במאגר הנתונים.

ב. לאחר הרצה בפייתון של 10 אלגוריתמי חלוקה לאשכולות בחרתי באלגוריתם K-Means, שכן הוא חילק את הנתונים בצורה הטובה ביותר – בצורה הנראית הטובה ביותר לעין. האלגוריתמים האחרים ששקלתי והרצתי בפייתון הם: Affinity Propagation, Agglomerative Clustering, BIRCH, DBSCAN,
K-Means Mini-Batch, Mean Shift, OPTICS, Spectral Clustering, Gaussian Mixture Model
שיטת חלוקה.
K-means הוא אלגוריתם פופולרי, יעיל ומהיר. הוא לינארי במספר האיטרציות שלו, במספר האוביקטים שלו ובמספר האשכולות שלו. הוא אלגוריתם היוריסטי והוא עוצר באופטימום מקומי ולא באופטימום גלובלי וקיימות שיטות שעוזרות לאלגוריתם זה להתקרב לאופטימום הגלובלי למרות שזה לא תמיד עוזר. כלומר לא בטוח שנמצא בעזרתו את השיוך הטוב ביותר של אובייקטים.

אפרט על אופן הפעולה של האלגוריתם.

אלגוריתם K means

1. מחלקים את האובייקטים באופן אקראי ל-K קבוצות. אם יש ידע מוקדם על החלוקה אז אפשר לחלק לפי ידע זה.
2. מחשבים את הסנטרואידים (מרכז הכובד של האשכול, זהו וקטור המכיל מספרים שכל אחת מתאים לממוצע של משתנה/פיצ'ר של כל אובייקט באשכול) של האשכולות על פי החלוקה שביצענו.
3. מנסים לשנות את השיוך של כל אובייקט לאשכול - נחשב את המרחקים של כל אובייקט למרכז הכובד של האשכולות הראשוניים, במידה והמרחק בין אובייקט למרכז כובד של אשכול שאינו נמצא בו קצר יותר מהמרחק בין האובייקט לבין מרכז הכובד של האשכול שהוא כן נמצא בו אז נעביר את האובייקט מאשכול לאשכול.
4. נחזור לשלב 2. נפסיק את האיטרציות כאשר אין יותר מעברים של אובייקטים בין אשכולות.



ניתן לבחור 2 נקודות רחוקות בתור מרכזים ראשוניים כדי להבטיח התכנסות מהירה יותר וגם הגעה לתוצאות טובות יותר. כמו בציור הראשון כאן למעלה ^ .

<u>חולשות k-means</u>

1. אי אפשר להריץ את האלגוריתם אם לא יודעים לחשב את מרכז הכובד של אשכול. ובמידה והאובייקטים מאופיינים על ידי משתנים קטגוריים אז החישוב של מרכז כובד אינו טריוואלי ולעיתים לא אפשרי.
2. האלגוריתם מחייב הגדרה של K לפי הרצו. וייתכן שהמשתמש לא יודע מה K. ולכן יש להריץ את האלגוריתם עבור טווח של ערכים לK.
3. לא מתפקד היטב כשהנתונים רועשים ובעלי חריגים. מכיוון שמרכז כובד מתחשב בכל האובייקטים וממוצע/מרכז הכובד רגיש לזה ולכן מרכזי הכובד לא יהיו מדויקים.
4. הנחה סמויה של האלגוריתם - האשכולות קונבקסים (לא מעגליים) - לדוגמא אי אפשר למצוא אשכולות טובים עבור נתונים עם "מפרצים" או עם "חורים" בעזרת האלגוריתם לk-means.

<u>ווריאציות k-means</u>

1. אפשר לבחור סנטרואידים ראשוניים בכמה צורות שונות - לדוגמא לשייך באקראי או לקחת נקודות רחוקות.
2. חישוב מרחקים בין נקודות באופן שונה.
3. התייחסות שונה לדאטה קטגורי:
   א. החלפת ערך ממוצע בערך שכיח. נקרא K-modes.
   ב. שימוש מדדי אי דמיון נוספים
   ג. שימוש frequency based method כדי לעדכן את ערך השכיח בקלאסטרים.
   ד. שילוב של משתנים קטגוריים ונומינליים. נקרא: K - prototype

הבעיה עם אלגוריתם k-means היא יש רגישות גבוהה מדי לערכים חריגים.
לכן ניתן להשתמש ב"מדויד" - במקום להשתמש במרכז כובד (ממוצע) משתמשים באובייקט שהוא הכי "באמצע" של כל הנקודות. באופן זה נקודה חריגה לא תשפיע על בחירת המדויד.
מכיוון שבהכנת הנתונים ניקיתי ערכים חסרים אז אין את הבעיה הזו.

ג. <u>הפרמטרים המרכזיים לאלגוריתם K-Means:</u>

n_clusters – מספר הקלאסטרים שיש לשייך אליהם תצפיות / ליצור. כמו כן זהו מספר הסנטרואידים שיש ליצור.

Init – הבחירה של המרכזים הראשוניים של כל קלאסטר. ניתן לבחור random אשר בוחר באופן אקראי מרכזים ראשוניים. אפשר לבחור נקודות ספציפיות. בנוסף אפשר לבחור באופציית ברירת המחדל k-means++ אשר יעילה ובוחרת את הנקודות בהתבסס על תרומת כל נקודה (תצפית) לאיכות האשכול שנמדדת בעזרת מדד איכות של אנרציה(שימוש בסכום המרחקים בריבוע בין נקודות הנתונים).

n_init – מספר הפעמים שהאלגוריתם פועל עם מרכזים שונים.

ד. ראשית אני אריץ PCA שמצמצם את ממדים מ15 ל-2 כדי שאוכל להציג וויזואלית בגרף את הנתונים.

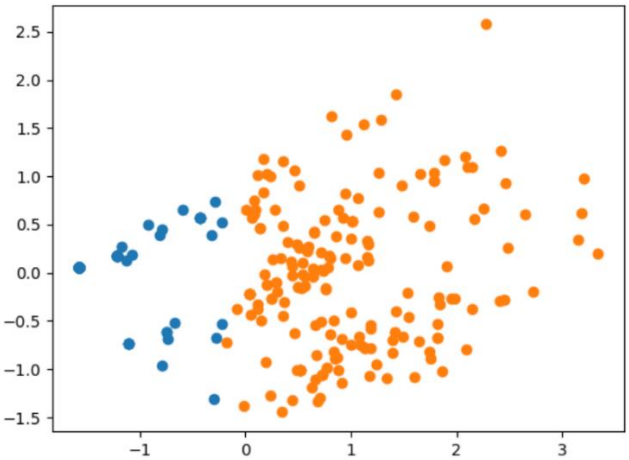
```
# PCA
pca = PCA(n_components=2)  pca: PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)  X_train_pca: [[ 0.20762611 -0.12765105], [ 2.25776597  0.66137249], [ 1.8165794  -0.67258799],
```

לאחר מכן אריץ את אלגוריתם K-means:

```
# After testing 10 clusters, I found k means is the best cluster.
# define the kmeans model
kmeans_model = KMeans(n_clusters=2)  kmeans_model: KMeans(n_clusters=2)
# fit the kmeans model
yhat = kmeans_model.fit_predict(X_train_pca)  yhat: [1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 1 0 0 1 1 1 0 1
# retrieve unique clusters
clusters = np.unique(yhat)  clusters: [0 1]

# create scatter plot for samples from each cluster
for cluster in clusters:  cluster: 1
    # get row indexes for samples with this cluster
    row_ix = np.where(yhat == cluster)  row_ix: ([  0   1   2   3   4   5   6   7   8  10  11  12  17  18
    # create scatter of these samples
    plt.scatter(X_train_pca[row_ix, 0], X_train_pca[row_ix, 1])
# show the plot
plt.show()
unique, counts = np.unique(kmeans_model.labels_, return_counts=True)  counts: [120 200]  unique: [0 1]
print(f'clustering: {dict(zip(unique, counts))}')
# print the real labels counts
unique, counts = np.unique(y_train, return_counts=True)
print(f'real labels: {dict(zip(unique, counts))}')
```

```
# validate that the clustering is good by comparing the real labels with the clustering labels
# compare the real labels with the clustering labels
df = pd.DataFrame({'clustering': kmeans_model.labels_, 'real': y_train.to_numpy()})
# count number of not matching labels
print(f'number of not matching labels: {df[df["clustering"] != df["real"]].shape[0]}')
```

התוצאות



	clustering	real
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	0	1
10	1	1
11	1	1
12	1	1
13	0	0

```
number of not matching labels: 16
```

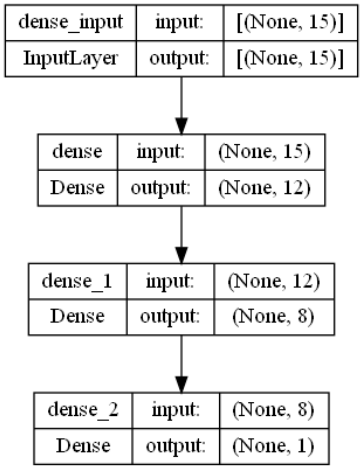
```
clustering: {0: 136, 1: 184}
real labels: {0: 120, 1: 200}
```

ה. באיור ניתן לראות ייצוג של הנקודות במרחב דו ממדי הנוצר על ידי PCA וk-means.
בכתום אלו הנבדקים ללא אי ספיקת כליות ובכחול אלו החולים באי ספיקת כליות.
ניתן לראות שהתבצעה חלוקה ל2 קבוצות של הנתונים. בהחלט יש כמות גדולה יותר של נבדקים ללא אי ספיקת כליות במאגר הנתונים שלנו.

בטבלה מוצג ה"סיווג" של אלגוריתם הקלאסטרינג ולצידו הסיווג המקורי (real).
יש 136 נבדקים ללא אי ספיקת כליות בסיווג אלגוריתם הקלאסטרינג ו120 נבדקים ללא אי ספיקת כליות במאגר הנתונים המקורי.

יש 16 תצפיות שאלגוריתם הקלאסטרינג לא סיווג נכון – כלומר שאין התאמה בין הערך בעמודת real לבין הערך בעמודת clustering.

שאלה 3



א. אציג את ארכיטקטורת הרשת שבחרתי.
שכבת קלט – בעלת 15 נוירונים, אחד עבור כל תכונה/פיצ'ר בבסיס הנתונים. שכבה זו מקבלת את הנתונים הגולמיים ומעבירה אותם לשכבה החבויה הראשונה.
שכבה חבויה 1 – בעלת 12 נוירונים ומשתמשת בפונקציית אקטיבציה relu. הפלט של שכבת הקלט מועבר לשכבה זו, בשכבה זו מתבצעת פונקציית הrelu ולאחר מכן מעבירים לשכבה החבויה הבאה את תוצאות פונקציית הrelu.
שכבה חבויה 2 – בעלת 8 נוירונים ומשתמשת בפונקציית אקטיבציה relu. שכבה זו מקבלת את תוצאות פונקציית הrelu של השכבה החבויה הראשונה, וגם כאן מתבצעת עוד פונקציית relu של השכבה הנוכחית. לבסוף הפלט עובר לשכבת הפלט.
שכבת פלט – בעלת פלט אחד ומשתמשת בפונקציית האקטיבציה sigmoid. הפלט של השכבה החבויה השנייה שהוא תוצאת חישוב של 2 שכבות חבויות מגיע אל שכבת הפלט ובעזרת פונקציית sigmoid ניתן לבצע את הסיווג הסופי למחלקות השונות.

אתחול משקולות - ביצעתי אתחול למשקולות מסוג he_uniform אשר מתאים לפונקציית האקטיבציה relu. זה עוזר לrelu להתגבר על משקלים קטנים מאוד ולהמשיך לשפר את דיוק המודל.
אתחול ההטיה - לא הוספתי bias כי אין צורך והמודל עובד טוב גם ללא.

להלן המימוש:

```
model = Sequential()
model.add(Dense(12, input_shape=(15,), activation='relu', bias_initializer='zeros', kernel_initializer='he_uniform'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

ב. הפרמטרים של תהליך האופטימיזציה
אופטימייזר - השתמשתי באופטימייזר הנפוץ adam, אשר הינו גרסה משופרת של gradient descent שכן הוא מכוון את עצמו אוטומטית לקבל תוצאות טובות עבור בעיה נתונה. בחרתי בקצב הלמידה learning rate של 0.001 לאחר כמה ניסיונות. קצב הלמידה קובע את גודל הצעד בכל איטרציה, ומשפיע עד כמה האופטימייזר

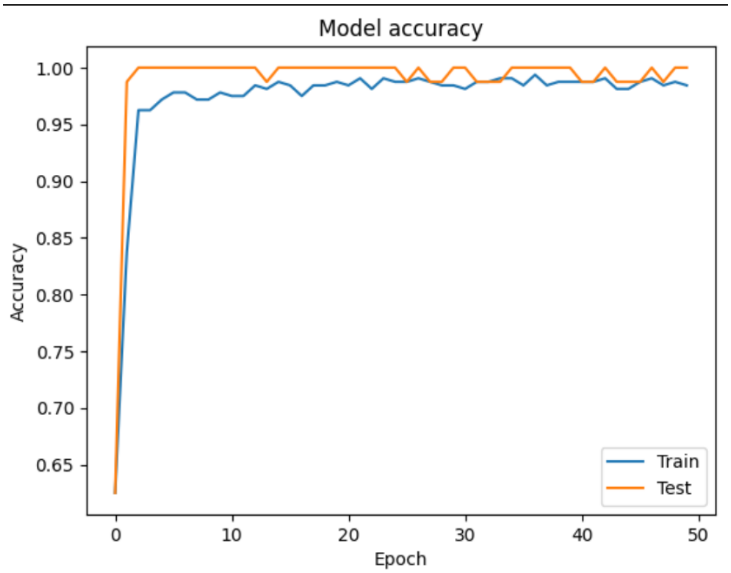
```
# configure the adam optimizer
adam_optimizer = Adam(learning_rate=0.001)
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer=adam_optimizer, metrics=['accuracy'])
```

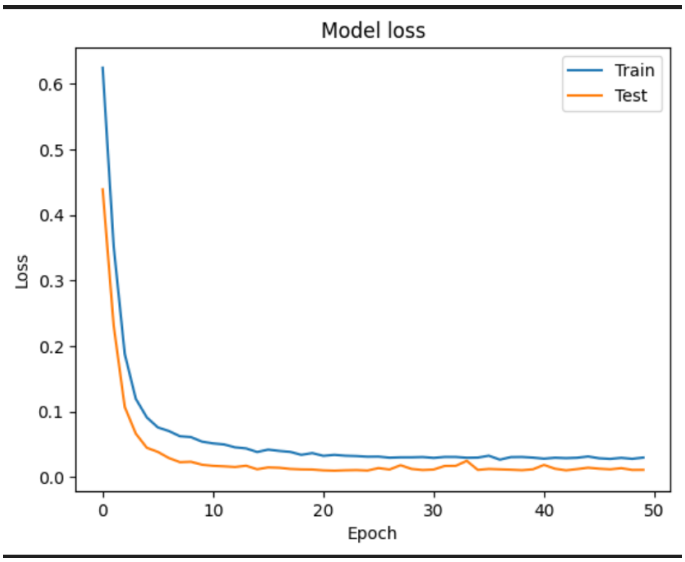
מתאים את המשקולות וההטיות של המודל בהתבסס על הגרדיאנט.
פונקציית הפסד - אני משתמש בפונקציית ההפסד binary_crossentropy שכן היא מתאימה עבור פלט סיווג בינרי.

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=5, verbose=1)
```

מדד הערכה – בחרתי במדד accuracy להערכת הצלחת המודל .
Batch – בחרתי ב5 לגודל הbatch. כלומר יהיו 5 תצפיות "שיפעפעו" ברשת בכל איטרציה של המודל.
לאחר כמה בדיקות מצאתי ש50 epochs עם גודל batch של 5 תצפיות מביא להתכנסות מהירה עם התוצאות הטובות ביותר.

ג. להלן ביצועי הרשת מבחינת דיוק ופונקציית השגיאה עבור כל epoch:





הקוד:

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```

ד. לפני שעשיתי ניסויים על גדלי batch ו-epochs שונים, קיבלתי לדוגמא כי תצפית מספר 27 לא סווגה נכון.

```
Misclassified samples: [27]
```

```
y_true = y_test.values
# Predicting the Test set results
y_pred = model.predict(X_test)
# y_pred = (y_pred > 0.5)
y_pred = np.round(y_pred)
# Find the misclassified samples
misclassified_indices = np.where(y_true != y_pred.T[0])[0]

# Printing the indices of the misclassified samples
print('Misclassified samples:', misclassified_indices)
```

לאחר משחק עם הפרמטרים הגעתי למצב בו המודל הצליח לסווג נכון את כל תצפיות המבחן:

```
Misclassified samples: []
```

ה. ניתן לראות בגרפים שציירתי שכבר לאחר מספר מועט מאוד של איטרציות המודל מגיע להצלחה גבוהה. בפרט עבור נתוני המבחן המעטים.
כמו כן ניתן לראות שאם אמשיך לאמן את המודל אז אקבל התאמת יתר על נתוני האימון ודיוק המודל יפחת עבור נתוני המבחן.
כמו כן, עבור גרף השגיאה loss ניתן לראות את דעיכת העקומה עד לכדי קרוב ל-0, מה שמראה על שגיאה נמוכה ומודל איכותי.
אציג את הצלחת המודל הכללית:

```
# print the model results
print(f'Train Accuracy: {model.evaluate(X_train, y_train, verbose=0)[1]}')
print(f'Test Accuracy: {model.evaluate(X_test, y_test, verbose=0)[1]}')
```

```
Train Accuracy: 0.987500011920929
Test Accuracy: 1.0
```

המודל מקבל 98.75% הצלחה על נתוני האימון ו-100% על נתוני המבחן. כלומר המודל למד להכליל בצורה טובה את הנתונים ולא ביצע overfitting על נתוני האימון. ההגעה ל100% היא הגיונית שכן סט הנתונים קטן יחסית.

שאלה 4

בפרויקט הזה (ממן 21 וממן 22) יצרתי מודלים שונים אשר לומדים את התנהגות הנתונים ומצליחים לסווג בהצלחה גבוהה נתונים חדשים שלא נראו בעבר.
שלב הכנת הנתונים היה זהה לכל המודלים – השלמת הנתונים, ניקוי רעשים ועוד.
בשלב הכנת הנתונים ציירתי בגרפים את הנתונים וראיתי איך הם מתפלגים ומצאתי ערכים חריגים.
בשלב הסיווג, ראיתי שאלגוריתם Random Forest הוא המוצלח ביותר מאלגוריתמי הmachine learning הקלאסיים. לאחר מכן ראיתי שבעזרת אלגוריתמים מתקדמים יותר, כמו רשת נוירונים (פשוטה), אפשר לקבל אפילו תוצאה טובה יותר במעט.
השתמשתי באלגוריתם קלאסטרינג k-means כדי לראות את הנתונים בצורה ברורה, להבין אותם בבהירות ולהבדיל ביניהם במרחב דו ממדי.
מבחינת השוואתית אפשר לראות ששימוש בגרף המציג את הצלחת/שגיאת המודל על פני הepochs השונים מראה בצורה ברורה ומובנת את תהליך הלמידה של המודל לאורך זמן, דבר שלא מתאפשר בROC curve.