Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Εξόρυξη Δεδομένων και Αλγόριθμοι Μάθησης

Υλοποιητικό Project

Λάμπρος Ανδρούτσος, ΑΜ 1054396

Email: androutsos@ceid.upatras.gr

Υπεύθυνοι Καθηγητές:

Χρήστος Μακρής

Μεγαλοοικονόμου Βασίλειος

Περιβάλλον Υλοποίησης

Το παρόν πρότζεκτ έτρεξε στο IDE Pycharm Professional και έκδοχη 2020.1.2. Ακολουθούν οι βιβλιοθήκες λογισμικού που χρησιμοποιήθηκαν:

Scikit-learn: Ανοιχτού-κώδικα βιβλιοθήκη Μηχανικής Μάθησης που υποστηρίζει επιβλεπόμενη και μη επιβλεπόμενη μάθηση. Επίσης, παρέχει και διάφορα εργαλεία για model fitting, προεπεξεργασία δεδομένων, επιλογή μοντέλου και αξιολόγηση και άλλα πολλά. Ακολουθούν τα εργαλεία της βιβλιοθήκης αυτής που χρησιμοποιήθηκαν σε αυτό το πρότζεκτ:

1ο Ερώτημα:

- sklearn.svm.SVC,
- sklearn.preprocessing.LabelEncoder(),
- train_test_split() και GridSearchCV() του sklearn.model_selection,
- sklean.preprocessing.StandardScaler
- LogisticRegression() του sklearn.linear_model
- Kmeans() του sklearn.cluster
- confusion_matrix, classification_report του sklearn.metrics

2ο Ερώτημα:

- train_test_split() του sklearn.model_selection
- TfidfVectorizer του sklearn.feature_extraction.text
- MLPClassifier του sklearn.neural_network
- confusion_matrix, classification_report του sklearn.metrics

Pandas: Βιβλιοθήκη λογισμικού για χειρισμό και ανάλυση δεδομένων. Χρησιμοποιείται και στα δύο ερωτήματα με την εντολή **import pandas as pd**.

Numpy: Η βασική βιβλιοθήκη για επιστημονικούς υπολογισμούς της Python. Χρησιμοποιείται στο πρώτο ερώτημα με την εντολή **import numpy as np**.

NLTK: Εργαλειοθήκη για επεξεργασία φυσικής γλώσσας. Χρησιμοποιείται στο δεύτερο ερώτημα με τις εντολές: **PorterStemmer() του nltk.stem, stopwords του nltk.corpus, import nlt και nltk.download('stopwords').**

Regex: Βιβλιοθήκη για διεργασίες κανονικών εκφράσεων. Χρησιμοποιείται στο δεύτερο ερώτημα με την εντολή: **import regex as re.**

Matplotlib.pyplot : Για γραφικές παραστάσεις μέσω της εντολής **import matplotlib.pyplot as plt** για το 1ο ερώτημα.

Πρέπει να σημειωθεί πως για να τρέξουν οι παραπάνω βιβλιοθήκες πρέπει να τις εγκαταστήσουμε στο περιβάλλον που τρέχουμε τους κώδικες των ερωτημάτων με την εντολή **pip install [όνομα βιβλιοθήκης]** για κάθε βιβλιοθήκη ξεχωριστά.

1ο Ερώτημα

Α] Στο 1ο ερώτημα αυτού του πρότζεκτ μας δίνεται ένα dataset σχετικά με την ποιότητα κόκκινων κρασιών και εμείς πρέπει να μαντέψουμε την ποιότητά τους μέσω κατηγοριοποίησης SVM και με την πληροφορία ποιότητας από έναν γευσιγνώστη ως δεδομένο.

Για το Α ερώτημα της άσκησης μας ζητείται να χωρίσουμε τα dataset σε train και test και να μετρήσουμε την απόδοσή του SVM μοντέλου μέσω κάποιων μετρικών.

Η εξήγηση της όλης διαδικασίας θα πραγματοποιηθεί από μία σειρών από screenshots και μερική περιγραφή σχετικά με τον κώδικα.

```
# Importing the dataset
wq_ds = pd.read_csv('winequality-red.csv', sep=',')
y = wq_ds.quality
x = wq_ds.drop('quality', axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)
```

Ξεκινάμε εισάγοντας το dataset με sep=',' ώστε να χωρίζουμε τις στήλες μεταξύ τους. Μετά χωρίζουμε σε x και y το dataset, δηλαδή σε ένα DataFrame που δεν έχει την στήλη quality (ποιότητα) και σε ένα που έχει μόνο αυτήν. Αυτό συμβαίνει γιατί εμείς θέλουμε να προβλέψουμε την ποιότητα και την θεωρούμε το σημαντικότερο attribute ενός κόκκινου κρασιού. Τέλος, χωρίζουμε σε training και test τα δεδομένα μας αυτά ώστε να εκπαιδεύσουμε το SVC μοντέλο μας.

```
# Normalization
sc = StandardScaler()
x_train1 = sc.fit_transform(x_train)
x_test1 = sc.fit_transform(x_test)
```

Κανονικοποίηση των training και test μέσω του StandardScaler, ώστε να έχουμε καλύτερα αποτελέσματα στις προβλέψεις μας. Είναι ένα τρόπος προεπεξεργασίας δεδομένων.

```
# Implementing the Support Vector Machine model
svc_model_def = SVC()
svc_model_def.fit(x_train1, y_train)
svc_model_def_predict = svc_model_def.predict(x_test1)
print("Η απόδοση του μοντέλου μας για τις default παραμέτρους:")
print(classification_report(y_test, svc_model_def_predict, zero_division=0))
print(confusion_matrix(y_test, svc_model_def_predict))
```

Αρχικοποιήση του SVM μας και εκπαίδευση αυτού με τα κανονικοποιημένα πλέον δεδομένα μας. Χρησιμοποιούμε το SVM με τις default παραμέτρους του, δηλαδή Έπειτα, κάνουμε μια πρόβλεψη με βάση το test dataset και μέσω του classification_report εμφανίζουμε την απόδοση του μοντέλου μας μέσω των μετρικών f1 score, precision και recall. Συνήθως, χρησιμοποιούνται και μητρώα σύγχυσης για την περιγραφή της απόδοσης ενός μοντέλου, οπότε υλοποιήθηκε και αυτό ως έξτρα. Ακολουθεί η απόδοση του παραπάνω μοντέλου SVM:

```
Η απόδοση του μοντέλου μας για τις default παραμέτρους:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        14
           5       0.66      0.75      0.70       169
           6       0.61      0.67      0.64       170
           7       0.58      0.35      0.44        40
           8       0.00      0.00      0.00         5

    accuracy                           0.64       400
   macro avg       0.31      0.29      0.30       400
weighted avg       0.60      0.64      0.61       400

[[  0   0   1   1   0   0]
 [  0   0  11   3   0   0]
 [  0   0 126  42   1   0]
 [  0   0  50 114   6   0]
 [  0   0   2  24  14   0]
 [  0   0   0   2   3   0]]
```

Παρατηρούμε τα weighted averages των τριών μετρικών που μας ζητούνται και μέσω αυτών θα συγκρίνουμε τα υπόλοιπα μοντέλα μας με το αρχικό μοντέλο SVM με default παραμέτρους. Στο κάτω μέρος της εικόνας φαίνεται το μητρώο σύγχυσης.

```
# param_grid = {'C': [0.1, 1, 10, 100],
#                   'gamma': ['scale', 'auto', 1, 0.1, 0.01, 0.001],
#                   'kernel': ['rbf', 'sigmoid', 'linear']}
# grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3, return_train_score=True)
# grid.fit(x_train, y_train)
#
# print(" C gamma and kernel καλύτεροι παράμετρες: ", grid.best_params_)
# print("Σκορ του ταξινομητή για τις καλύτερες παραμέτρους: ", grid.best_score_)
# print("Ο καλύτερος ταξινομητής: ", grid.best_estimator_)
# best_svm_clf = grid.best_estimator_

# SVM with the best parameters
svc_model = SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
                      decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf',
                      max_iter=-1, probability=False, random_state=None, shrinking=True,
                      tol=0.001, verbose=False)
svc_model.fit(x_train1, y_train)
svc_model_predict = svc_model.predict(x_test1)

print("Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους:")
print(classification_report(y_test, svc_model_predict, zero_division=0))
print(confusion_matrix(y_test, svc_model_predict))
```

Εμείς ενδιαφερόμαστε για το καλύτερο πιθανό μοντέλο, άρα και για την καλύτερη πιθανή πρόβλεψη της ποιότητας που θέλουμε. Για αυτόν τον λόγο πρέπει να ερευνήσουμε ποιες από τις παραμέτρους του SVM πρέπει να αρχικοποιήσουμε με συγκεκριμένες τιμές για να προκύψει το βέλτιστο SVM μοντέλο. Αυτή την εύρεση των βέλτιστων παραμέτρων την κάνει το GridSearchCV(), το οποίο εκτελεί εξαντλητική αναζήτηση χρησιμοποιώντας τις παραμέτρους που ορίζουμε στο param_grid. Οι κυριότερες παράμετροι που μας νοιάζουν είναι το kernel, το C (παράμετρος ρύθμισης) και το gamma (kernel συντελεστής) . Για το kernel δεν θα βάλουμε την τιμή poly (polynomial), καθώς επιβραδύνει κατά πολύ τον κώδικα και τελικά δεν θα είναι και η καλύτερη παράμετρος, όπως ισχύει συνήθως. Για μεγαλύτερες τιμές C και gamma το μοντέλο μας καταλήγει σε Overfitting, το οποίο θέλουμε οπωσδήποτε να αποφύγουμε. Ο κώδικας σχετικά με το GridSearchCV() είναι commented out, καθώς δεν θέλουμε να τρέχει κάθε φορά. Το εκτελέσαμε την πρώτη φορά και προέκυψαν οι παράμετροι που εισάγουμε στο νέο μοντέλο SVM, το svc_model. Οπότε, ακολουθεί η εκπαίδευση του βέλτιστου μοντέλου SVM και η αντίστοιχη πρόβλεψη. Ακολουθούν τα αποτελέσματα:

```
Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       1.00      0.07      0.13        14
           5       0.66      0.80      0.73       169
           6       0.69      0.65      0.67       170
           7       0.63      0.47      0.54        40
           8       0.00      0.00      0.00         5

    accuracy                           0.67       400
   macro avg       0.50      0.33      0.35       400
weighted avg       0.67      0.67      0.65       400

[[  0   0   1   1   0   0]
 [  0   1   6   7   0   0]
 [  0   0 136  30   3   0]
 [  0   0  52 111   7   0]
 [  0   0   8  11  19   2]
 [  0   0   2   2   1   0]]
```

Παρατηρούμε καλύτερες αποδόσεις, ειδικά στα macro averages, όπως είναι λογικό, αφού πλέον το μοντέλο μας είναι "βέλτιστο".

Β] Στο 2ο ερώτημα αυτής της άσκησης ασχολούμαστε με τον χειρισμό ελλιπών τιμών (NaN) της στήλης pH, του training dataset που προέκυψε από το 1ο ερώτημα. Αρχικά, πρέπει να αφαιρέσουμε το 33% των τιμών της στήλης pH του training dataset.

```
wq_ds1 = x_train.pH
wq_ds2 = wq_ds1.sample(frac=.33, random_state=138)
wq_ds1 = wq_ds1.drop(wq_ds2.index)
wq_ds3 = x_train.drop('pH', axis=1)
wq_ds4 = wq_ds3.join(wq_ds1)
```

Παίρνουμε ένα τυχαίο δείγμα μεγέθους ίσου με το 33% του μεγέθους των γραμμών του pH. Μέσω των indexes των γραμμών αυτών που δείγματος, τα αφαιρούμε από το pH. Διαγράφουμε τελείως την στήλη pH από το dataset και ενώνουμε το DataFrame που έχει 33% λιγότερες τιμές με το DataFrame που δεν έχει πια pH στήλη. Έτσι, το νέο DataFrame **wq_ds4** είναι ένα DataFrame με τις διαστάσεις του **x_train** αλλά με 33% των τιμών του pH να λείπουν.

Εμείς λοιπόν θέλουμε να διαχειριστούμε αυτές τις τιμές που λείπουν με 4 διαφορετικούς τρόπους, να εκπαιδεύσουμε ξανά το βέλτιστο SVM μας με τα νέα μητρώα και να συγκρίνουμε τις αποδόσεις ξανά.

1ος τρόπος: Αφαίρεση στήλης pH|

```
# SVC with no ph column
x_train2 = wq_ds4.drop('pH', axis=1)
x_test2 = x_test.drop('pH', axis=1)
x_train2 = sc.fit_transform(x_train2)
x_test2 = sc.fit_transform(x_test2)
svc_model.fit(x_train2, y_train)
svc_model_predict_noph = svc_model.predict(x_test2)
print("Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους του SVM και χωρίς πλέον την στήλη pH:")
print(classification_report(y_test, svc_model_predict_noph, zero_division=0))
print(confusion_matrix(y_test, svc_model_predict_noph))
```

Αφαιρούμε την στήλη pH από τα δεδομένα μας, κάνουμε κανονικοποίηση και εκπαιδεύουμε πάλι το βέλτιστο SVM. Η μέθοδος αυτή διαχείρισης ελλιπών τιμών οδηγεί συνήθως σε ένα robust μοντέλο υψηλής ακρίβειας, με μειονέκτημα το χάσιμο πληροφορίας και δεδομένων και αν το ποσοστό των ελλιπών τιμών είναι υψηλό, δεν θα δουλέψει τόσο αποδοτικά. Ακολουθεί η απόδοση του μοντέλου αυτού με αυτή τη μέθοδο διαχείρισης:

```
Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους του SVM και χωρίς πλέον την στήλη pH:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       1.00      0.07      0.13        14
           5       0.68      0.79      0.73       169
           6       0.68      0.68      0.68       170
           7       0.69      0.50      0.58        40
           8       0.00      0.00      0.00         5

    accuracy                           0.68       400
   macro avg       0.51      0.34      0.35       400
weighted avg       0.68      0.68      0.66       400

[[  0   0   1   1   0   0]
 [  0   1   7   6   0   0]
 [  0   0 134  32   3   0]
 [  0   0  50 115   5   0]
 [  0   0   5  12  20   3]
 [  0   0   1   3   1   0]]
```

Μένει να κάνουμε και τις άλλες τρεις μεθόδους για να συγκρίνουμε ποια είναι η καλύτερη στην περίπτωση μας.

2ος τρόπος: Συμπλήρωση των ελλιπών τιμών με το μέσο όρο των στοιχείων της στήλης|

```
# Filling NaN values with the mean of the rest.
x_train3 = wq_ds4.fillna(x_train.pH.mean())
x_train3 = sc.fit_transform(x_train3)
svc_model.fit(x_train3, y_train)
svc_model_predict_avg = svc_model.predict(x_test1)
print("Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους του SVM και όπου NaN στο pH πλέον υπάρχει ο μέσος όρος των υπόλοιπων τιμών")
print(classification_report(y_test, svc_model_predict_avg, zero_division=0))
print(confusion_matrix(y_test, svc_model_predict_avg))
```

Όπως βλέπουμε, η μόνη διαφορά είναι ότι πλέον αντί να διαγράψουμε την στήλη, πλέον αλλάζουμε τις NaN τιμές με τον αριθμητικό μέσο όρο των τιμών της στήλης pH και μετά από κανονικοποίηση εκπαιδεύουμε πάλι το μοντέλο και προβλέπουμε.

```
Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους του SVM και όπου NaN στο pH πλέον υπάρχει ο μέσος όρος των υπόλοιπων τιμών
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        14
           5       0.49      0.70      0.57       169
           6       0.54      0.49      0.51       170
           7       0.00      0.00      0.00        40
           8       0.00      0.00      0.00         5

    accuracy                           0.51       400
   macro avg       0.17      0.20      0.18       400
weighted avg       0.43      0.51      0.46       400

[[  0   0   1   1   0   0]
 [  0   0   9   5   0   0]
 [  0   0 118  50   1   0]
 [  0   0  86  84   0   0]
 [  0   0  24  16   0   0]
 [  0   0   4   1   0   0]]
```

Παρατηρούμε πως προκύπτει χειρότερη απόδοση από ότι με τον 1ο τρόπο διαχείρισης, κάτι που είναι λογικό αφού πλέον η απόκλιση των τιμών στην στήλη αυτή είναι αρκετά μικρότερη.

3ος τρόπος: Συμπλήρωση των ελλιπών τιμών χρησιμοποιώντας Logistic Regression|

```python
# Filling NaN values using Logistic Regression.
x_train_nan = wq_ds4.isnull()
x_train_rows_nan = x_train_nan.any(axis=1)
x_train_rows_with_nan = wq_ds4[x_train_rows_nan]
wq_ds4_no_nan = wq_ds4.dropna()

# Logistic Regression Model
y1 = wq_ds4_no_nan.pH
lab_enc = preprocessing.LabelEncoder()
y1 = lab_enc.fit_transform(y1)
x1 = wq_ds4_no_nan.drop('pH', axis=1)
# Splitting to train and test
x_train_lr, x_test_lr, y_train_lr, y_test_lr = train_test_split(x1, y1, test_size=0.25, random_state=0)
lr = LogisticRegression(random_state=0, solver="liblinear")
lr.fit(x_train_lr, y_train_lr)
lr_predict = lr.predict(x_train_rows_with_nan.drop('pH', axis=1))
predictionPH = lab_enc.inverse_transform(lr_predict)
predictionPH = np.reshape(predictionPH, (predictionPH.shape[0], 1))

# nparray to DataFrame
predictionPH = pd.DataFrame(predictionPH, columns=['pH'])
x_train_rows_with_nan.append(predictionPH)

x_train_rows_with_nan['pH'] = predictionPH['pH'].values
frames = [wq_ds4_no_nan, x_train_rows_with_nan]
```

```python
result = pd.concat(frames)

result.sort_index(inplace=True)
y_train.sort_index(inplace=True)
result = sc.fit_transform(result)
svc_model.fit(result, y_train)
svc_model_predict = svc_model.predict(x_test1)

print("Η απόδοση του logistic regression μας για τις καλύτερες παραμέτρους:")
print(classification_report(y_test, svc_model_predict, zero_division=0))
print(confusion_matrix(y_test, svc_model_predict))
```

Ξεκινάμε χωρίζοντας το wq_ds4 σε δύο dataset, ένα το οποίο έχει όλα τα rows του wq_ds4 που έχουν κάπου NaN τιμή και το λέμε **x_train_rows_with_nan** και ένα το οποίο έχει όλες τα υπόλοιπα rows και το λέμε **wq_ds4_no_nan**. Χρησιμοποιώντας ως y, την στήλη pH του δεύτερου dataframe και ως x1 το δεύτερο dataframe χωρίς την στήλη pH, θα χωρίσουμε πάλι τα δεδομένα σε train και test και θα εκπαιδεύσουμε με αυτά το μοντέλο Logistic Regression. Όπως μπορεί να παρατηρήσει κάποιος στον κώδικα, χρησιμοποιούμε το **LabelEncoder()** το οποίο είναι μέθοδος προεπεξεργασίας και αλλάζει τα continuous δεδομέν του y1 σε τιμές ανάμεσα στο 0 και στον αριθμό των κλάσεων -1. Αφού κάνουμε την πρόβλεψη μας μέσω του μοντέλου αυτού, κάνουμε ανάστροφο μετασχηματισμό των δεδομένων πρόβλεψης που προκύπτουν για να έχουμε τιμές continuous ξανά. Πλέον αυτό που θέλουμε είναι εισάγουμε στο **x_train_rows_with_nan** τις τιμές pH που προέκυψαν από την πρόβλεψη, ώστε πλέον να μην έχουμε NaN τιμές. Τέλος, ενώνουμε τα **wq_ds4_no_nan** και **x_train_rows_with_nan** και χρησιμοποιώντας το νέο dataframe που το λέμε **result** ως το x_train του SVM μας συνδυασμό με το αρχικό y_train, εκπαιδεύουμε το μοντέλο SVM μας και προβλέπουμε ξανά. Ακολουθούν τα αποτελέσματα:

```
Η απόδοση του logistic regression μας για τις καλύτερες παραμέτρους:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        14
           5       0.49      0.74      0.59       169
           6       0.59      0.51      0.54       170
           7       0.00      0.00      0.00        40
           8       0.00      0.00      0.00         5

    accuracy                           0.53       400
   macro avg       0.18      0.21      0.19       400
weighted avg       0.46      0.53      0.48       400

[[  0   0   2   0   0   0]
 [  0   0  11   3   0   0]
 [  0   0 125  44   0   0]
 [  0   0  84  86   0   0]
 [  0   0  29  11   0   0]
 [  0   0   3   2   0   0]]
```

Παρατηρούμε απόδοση χειρότερη του 1ου τρόπου, αλλά καλύτερη του δεύτερου. Αυτό συμβαίνει γιατί ακολουθεί διαδικασία μηχανικής μάθησης και δίνει πιο αντιπροσωπευτικές τιμές για το χαρακτηριστικό pH.

4ος τρόπος: Εφαρμογή K-means και συμπλήρωση των τιμών που λείπουν με τον αριθμητικό μέσο όρο του cluster στο οποίο ανήκει το δείγμα|

```
# Filling NaN values with kmeans
# Finding the best #of clusters for our dataset
# distortion = []
# K = range(1,10)
# for k in K:
#     kmean_Model = KMeans(n_clusters=k).fit(x_train)
#     kmean_Model.fit(x_train)
#     distortion.append(sum(np.min(cdist(x_train, kmean_Model.cluster_centers_, 'euclidean'), axis=1)) / x_train.shape[0])
# # Plot the elbow
# plt.plot(K, distortion, 'bx-')
# plt.xlabel('k')
# plt.ylabel('Παραμόρφωση')
# plt.title('Η Elbow Method που δείχνει το καλύτερο k')
# plt.show()

# initializing kmeans
kmeans = KMeans(n_clusters=3, init="k-means++", max_iter=300)
kmeans.fit(wq_ds4_no_nan)
# labels of kmeans
labels = kmeans.labels_
# centroids of the clusters
centroids = kmeans.cluster_centers_
```

```
# True for NaN, False for actual value
nan_or_not = ~np.isfinite(wq_ds4)
# Mean of every column
mean = np.nanmean(wq_ds4, 0, keepdims=True)
# If False, choose mean. If True, choose wq_ds4 value
new_dataset = np.where(nan_or_not, mean, wq_ds4)
max_iter = 10
for i in range(max_iter):
    if i > 0:
        method = KMeans(3, init=centroids)
    else:

        method = KMeans(3, n_jobs=-1)

    # Clustering for the new data
    labels1 = method.fit_predict(new_dataset)
    # New centroids
    centroids = method.cluster_centers_
    # fill in the missing values based on their cluster centroids

    new_dataset[nan_or_not] = centroids[labels1][nan_or_not]

    # when the labels have stopped changing then we have converged
    if i > 0 and np.all(labels1 == labels):
```

```
        break

prev_labels = labels1
prev_centroids = method.cluster_centers_
new_dataset = pd.DataFrame(new_dataset, columns=['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
x_train_new = new_dataset
y_new = y_train
x_train, x_test, y_train_new, y_test_new = train_test_split(x_train_new, y_new, test_size=0.25, random_state=0)
x_train1 = sc.fit_transform(x_train)
x_test1 = sc.fit_transform(x_test)
svc_model.fit(x_train1, y_train_new)
svc_model_predict_Kmean = svc_model.predict(x_test1)
print("Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους του SVM και όπου NaN στο pH πλέον υπάρχει ο αριθμητικός μέσος όρος του cluster στο οποίο ανήκει κάθε δείγμα")
print(classification_report(y_test_new, svc_model_predict_Kmean, zero_division=0))
print(confusion_matrix(y_test_new, svc_model_predict_Kmean))
```

Στο new dataset συνεχίζεται : `, 'sulphates', 'alcohol', 'pH'])`

Ξεκινήσαμε ψάχνοντας τον βέλτιστο αριθμό clusters για τον K-means μας, εφαρμόζοντας την **Elbow Method**. Η **Elbow Method** τρέχει τον k-means για διάφορες τιμές του k και υπολογίζει ένα μέσο score για όλους τους cluster για κάθε k. Στη συνέχεια τυπώνουμε την γραφική παράσταση και αν σκεφτούμε ότι μοιάζει με χέρι, στο σημείο όπου εμφανίζεται ένας «αγκώνας» το αντίστοιχο k είναι το καλύτερο. Εμείς βρήκαμε k=3. Έπειτα, ξεκινάει η εφαρμογή του μοντέλου K-means.

Ξεκινάμε αρχικοποιώντας τον K-means και εκπαιδεύοντας τον με το **wq_ds4_no_nan**, βρίσκουμε τις ετικέτες των clusters και τα κεντροειδή των clusters. Αμέσως μετά, γεμίζουμε ένα νέο dataset, ανάλογο με το εάν η αντίστοιχη τιμή του nan_or_not είναι false ή true (false αν δεν είναι nan, true αν είναι). Αν είναι false, στην αντίστοιχη θέση του dataset θα μπει η αντίστοιχη τιμή του μέσου όρου (**mean = np.nanmean(wq_ds4, 0 keepdims=True**)), ενώ αν είναι true, θα μπει η αντίστοιχη τιμή του **wq_ds4.** Συνεχίζουμε κάνοντας το επαναληπτικό κομμάτι του kmeans, ώστε να υπολογίσει τα κεντροειδή που θέλουμε και παίρνουμε το νέο dataset που προκύπτει και αφού το μετατρέψουμε σε DataFrame, το χρησιμοποιούμε ως x_train για το νέο train test split μας. Κάνουμε κανονικοποίηση των αποτελεσμάτων, εκπαιδεύουμε τον SVM και προβλέπουμε. Ακολουθούν τα αποτελέσματα:

```
Η απόδοση του μοντέλου μας για τις καλύτερες παραμέτρους του SVM και όπου NaN στο pH πλέον υπάρχει ο αριθμητικός μέσος όρος του cluster στο οποίο ανήκει κάθε δείγμα
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         9
           5       0.45      0.65      0.53       130
           6       0.35      0.27      0.31       113
           7       0.07      0.03      0.04        39
           8       0.00      0.00      0.00         8

    accuracy                           0.39       300
   macro avg       0.14      0.16      0.15       300
weighted avg       0.34      0.39      0.35       300

[[ 0  0  1  0  0  0]
 [ 0  0  3  6  0  0]
 [ 1  3 85 35  5  1]
 [ 2  1 70 31  9  0]
 [ 0  0 23 15  1  0]
 [ 0  0  7  1  0  0]]
```

Παρατηρούμε πως αυτός ο τρόπος διαχείρισης των ελλιπών τιμών έδωσε τα χειρότερα αποτελέσματα, γιατί οι τιμές- κεντροειδή των clusters φαίνεται να μην αντιπροσωπεύουν καλά το χαρακτηριστικό pH. Αξίζει να σημειωθεί πως τα αποτελέσματα του kmeans μπορεί να είναι διαφορετικά κάποιες φορές, λόγω τυχαίας ανάθεσης κεντροειδών στην αρχή του μοντέλου.

2ου Ερώτημα

Στο δεύτερο ερώτημα μας δίνεται ένα dataset onion-or-not και εμείς καλούμαστε να μαντέψουμε αν οι τίτλοι είναι ψευδείς ειδήσεις ή όχι, μέσω ενός νευρωνικού δικτύου που θα δημιουργήσουμε.

Ξεκινάμε κάνοντας την βασική προεπεξεργασία που μπορούμε να κάνουμε στους τίτλους μας.

```
# Data import
ds = pd.read_csv('onion-or-not.csv')

def titles_to_words(raw_title):

    # Punctuation removal
    letters_only = re.sub("[^a-zA-Z]", " ", raw_title)

    # no Upper letters
    words = letters_only.lower().split()

    # Word stemming for every word
    stemmer = PorterStemmer()
    tokens_lem = [stemmer.stem(i) for i in words]

    # Stop words removal
    stops = set(stopwords.words('english'))
    meaningful_words = [w for w in words if not w in stops]

    # Join into sentence again
    return (" ".join(meaningful_words))
```

```
# total_titles = ds.shape[0]

# Titles after preprocessing
clean_titles = []

j = 0
for title in ds['text']:
    # Convert to words, then append to clean_train
    clean_titles.append(titles_to_words(title))
```

Αρχικά, φορτώνουμε το dataset μας και χωρίζουμε τους τίτλους από το **column 'text'** του dataset σε λέξεις, καλώντας την function **titles_to_words()**. Αυτή την συνάρτηση την τρέχουμε για κάθε τίτλο. Ουσιαστικά, ξεκινάμε αφαιρώντας όλα τα σημεία στίξης μέσω μίας κανονικής έκφρασης, ώστε να έχουμε έναν τίτλο χωρίς σημεία στίξης. Μετά, κάνουμε lowercase όλα τα γράμματα των λέξεων αυτών και split τις λέξεις, ώστε να καταλήξουμε σε ένα διάνυσμα λέξεων για κάθε τίτλο στο οποίο εφαρμόζουμε stemming μέσω του PorterStemmer(), δηλαδή αφαιρούμε όλες τις καταλήξεις των λέξεων, κρατώντας μόνο το θέμα τους. Επόμενο βήμα είναι η αφαίρεση των stopwords, καθώς είναι λέξεις που εμφανίζονται πολύ συχνά και δεν προσφέρουν ιδιαίτερη πληροφορία. Αυτό γίνεται αφαιρώντας όλες τις λέξεις που υπάρχουν στο stopwords('english') του NLTK από το διάνυσμα λέξεων κάθε τίτλου. Τέλος, για κάθε τίτλο ξαναενώνουμε τις λέξεις μεταξύ τους.

```
# Create Tf_Idf Bag of Words
vect = TfidfVectorizer()

# Fit the vectorizer on our corpus and transform
matrix_titles = vect.fit_transform(clean_titles)
matrix_titles = pd.DataFrame(matrix_titles.toarray(), columns=vect.get_feature_names())
matrix_labels = ds['label']

# Create Classifier Neural Network, Train it and run predictions
titles_train, titles_test, label_train, label_test = train_test_split(matrix_titles, matrix_labels, test_size=0.25, random_state=15)

classifier = MLPClassifier()
classifier.fit(titles_train, label_train)
predictions = classifier.predict(titles_test)
classification_report(label_test, predictions)
```

Αυτή η συνένωση έγινε για να αναθέσουμε τιμές tf-idf σε όλες τις λέξεις, ως βάρη. Ουσιαστικά, φτιάχνουμε το **matrix_titles** που πλέον έχει για κάθε λέξη κάθε τίτλου μια tf-idf τιμή, σύμφωνα με το πως δουλεύει ο **TfidfVectorizer()**. Το μετατρέπουμε σε DataFrame, όπου ως γραμμές θα έχει αυτές τις τιμές tf-idf για κάθε τίτλο και ως στήλες τις unique λέξεις και χωρίζουμε σε train test χρησιμοποιώντας το μητρώο αυτό που δεν έχει στήλη label, δηλαδή δεν έχει πληροφορία αν κάθε τίτλος είναι ψευδείς είδηση ή όχι και ένα μητρώο με μόνο την στήλη label. Τα titles_train και label_train που προκύπτουν τα χρησιμοποιούμε για την εκπαίδευση του μοντέλου του νευρωνικού μας δικτύου, δηλαδή

Multilayer Perceptron και προβλέπουμε αν οι τίτλοι είναι ψευδείς ειδήσεις ή όχι. Ο Multilayer Perceptron είναι η πιο βασική μορφή νευρωνικού δικτύου και μας επιτρέπει να ελέγξουμε πιο εύκολα τις παραμέτρους. Για NLTK χρησιμοποιείται συνήθως MLP και αποτελεί αλγόριθμος επιβλεπόμενης μάθησης. Ακολουθεί η απόδοση του μοντέλου μας:

```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83      3725
           1       0.72      0.73      0.73      2275

    accuracy                           0.79      6000
   macro avg       0.78      0.78      0.78      6000
weighted avg       0.79      0.79      0.79      6000

[[3091  634]
 [ 614 1661]]
```

Τα αποτελέσματα αυτά είναι αρκετά καλά, ειδικά αν σκεφτεί κανείς πως χρησιμοποιούμε τον Multilayer Perceptron με τις default παραμέτρους. Αν προσθέσουμε solver= ‘ lbfgs’ και κυρίως hidden_layer_size=[7] που αυξάνει τα κρυφά επίπεδα του νευρωνικού πχ, τα αποτελέσματα θα είναι ενδεχομένως πολύ καλύτερα, αλλά η λύση αυτή παίρνει πάρα πολύ χρόνο και πόρους για να τρέξει, με αποτέλεσμα να μην μπορεί να ολοκληρωθεί επιτυχώς στον προσωπικό μου υπολογιστή.