Προπτυχιακό μάθημα: Μηχανική Μάθηση

Αναφορά 1ης σειράς ασκήσεων

Ομάδα :

Αθανάσιος Τόμπρας 3345

Δήμητρα Μαχαιρίδου 4108

Εαρινό Εξάμηνο 2022

ΓΕΝΙΚΑ

Για την υλοποίηση των αλγορίθμων χρησιμοποιήθηκαν οι συναρτήσεις της βιβλιοθήκης sklearn ενώ η δημιουργία των αντίστοιχων προγραμμάτων έγινε με χρήση της εφαρμογής Jupyter notebook μέσω του Anaconda Navigator.

Λόγω της απουσίας της κατηγορίας type από το αρχείο test.csv αποφασίστηκε να γίνει διαχωρισμός των δεδομένων του train.csv σε train set και test set σε ποσοστό 25% με σκοπό την εκπαίδευση του μοντέλου και τη σύγκριση με το test set ώστε να βρεθούν τα accuracy score και f1 score. Επομένως το σύνολο εκπαίδευσης έχει 278 σειρές και 5 κατηγορίες στη συνιστώσα X_train αφού εξαιρούμε την κατηγορία 'type' και 278 σειρές και μόνο την κατηγορία 'type' στη συνιστώσα y_train ενώ το σύνολο ελέγχου περιέχει 93 σειρές στις αντίστοιχες συνιστώσες X_test και y_test με τις ανάλογες κατηγορίες όπως στο σύνολο ελέγχου.

Όσον αφορά το χειρισμό του πεδίου color με την βοήθεια της βιβλιοθήκης sklearn χρησιμοποιήθηκε η συνάρτηση preprocessing.LabelEncoder() ώστε να δωθεί μια ακέραια τιμή σε κάθε διαφορετική μεταβλητή τύπου color(κάθε χρώμα) και στη συνέχεια διαιρέθηκε ο κάθε ακέραιος με το 6 για να είναι οι τιμές τους στο διάστημα [0,1], όπως φαίνεται παρακάτω.

```
color = preprocessing.LabelEncoder()
color.fit(train_data['color'])
train_data['color'] = color.transform(train_data['color'])
train_data['color'] = train_data['color']/6
train_data.head()
```

	bone_length	rotting_flesh	hair_length	has_soul	color	type
0	0.354512	0.350839	0.465761	0.781142	0.500000	Ghoul
1	0.575560	0.425868	0.531401	0.439899	0.666667	Goblin
2	0.467875	0.354330	0.811616	0.791225	0.000000	Ghoul
3	0.776652	0.508723	0.636766	0.884464	0.000000	Ghoul
4	0.566117	0.875862	0.418594	0.636438	0.666667	Ghost

Το διάβασμα του αρχείου train.csv γίνεται με τον ίδιο τρόπο σε κάθε υλοποίηση του αλγορίθμου ως εξής:

```
: traindata=pd.read_csv('C:/Users/Thanos/Desktop/12TH SEMESTER/MHXANIKH MAΘHΣH/PROJECT/train.csv')
```

Επιπλέον αφαιρέθηκε η κατηγορία 'id' διότι δεν προσφέρει κάποια βοήθεια στην κατηγοριοποίηση.

```
#exairw to id apo ta dedomena giati prosthetei epipleon thoryvo
train_data = traindata.drop(['id'], axis = 1)
```

ΜΕΘΟΔΟΣ 1 : Κ-ΚΟΝΤΙΝΟΤΕΡΟΣ ΓΕΙΤΟΝΑΣ

Ο Αλγόριθμος Κ κοντινότεροι γείτονες (K Nearest Neighbors - KNN) είναι μία πολύ γνωστή και ευρεία χρησιμοποιούμενη τεχνική κατηγοριοποίησης που στηρίζεται στη χρήση μέτρων βασισμένων στην απόσταση. Η κεντρική ιδέα είναι πως η τιμή της συνάρτησης-στόχου για ένα νέο στιγμιότυπο βασίζεται αποκλειστικά και μόνο στις αντίστοιχες τιμές των k πιο «κοντινών» στιγμιότυπων εκπαίδευσης, τα οποία αποτελούν τους «γείτονες» του. Δύο ζητήματα πρέπει να αποφασιστούν προκειμένου να καθοριστεί πλήρως ο αλγόριθμος:

1. Ο ορισμός της απόστασης μεταξύ δύο στιγμιότυπων.

2. Η τιμή του k.

Για τον προσδιορισμό της απόστασης χρησιμοποιήθηκε η Ευκλείδια απόσταση('p' = 2):

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_n - x_n)^2} = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}.$$

Ενώ για την τιμή του κ δοκιμάστηκαν οι τιμές 1, 3, 5, 10 και επιλέχθηκε αυτή που έδινε την καλύτερη ακρίβεια.

Για την υλοποίηση του αλγορίθμου KNN χρησιμοποιήθηκε ο αντίστοιχος αλγόριθμος KNeighborsClassifier() της βιβλιοθήκης Sk learn. Επιπλέον χρησιμοποιήθηκε η συνάρτηση GridSearchCV() μέσω της οποίας υπολογίστηκε ο αριθμός γειτόνων που δίνει την καλύτερη ακρίβεια χωρίς να χρεαστεί να επαναληφθεί ο αλγόριθμος για κάθε τιμή του κ και αυτός είναι για κ = 10.

```
#KNeighborsclassifier
params = {'n_neighbors':[1, 3, 5, 10], 'leaf_size': [30], 'p': [2], 'weights': ['distance']}
knc = KNeighborsClassifier()
#Using GridSearch to find the number of neighbors that gives the best score
clf = GridSearchCV(knc, param_grid = params, scoring = accuracy_scorer, cv = 5, n_jobs = -1)
clf.fit(X_train, y_train)
print('Best score: {}'.format(clf.best_score_))
print('Best parameters: {}'.format(clf.best_params_))

knc_best = KNeighborsClassifier(n_neighbors = 10)
```

```
Best score: 0.6905194805194805
Best parameters: {'leaf_size': 30, 'n_neighbors': 10, 'p': 2, 'weights': 'distance'}
```

Τέλος, μετά την εκπαίδευση και την εφαρμογή του μοντέλου υπολογίζονται το Accuracy Score καθώς και το F1 Score μέσω των αντίστοιχων συναρτήσεων της βιβλιοθήκης sk learn όπως φαίνεται παρακάτω.

```
y_pred = clf.predict(X_test)
print("\nF1 Score is: " + str(metrics.f1_score(y_test, y_pred, average= "weighted")))
print("\nAccuracy Score is: " + str(metrics.accuracy_score(y_test, y_pred)))
```

```
F1 Score is: 0.7957496398413566

Accuracy Score is: 0.7956989247311828
```

ΜΕΘΟΔΟΣ 2 – ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Ένα Τεχνητό Νευρωνικό Δίκτυο MLP είναι ένα δίκτυο εισόδου-εξόδου αποτελούμενο από εσωτερικές κρυμμένες (*hidden*)μονάδες perceptrons οι οποίες είναι διασυνδεδεμένες σε επίπεδα (*layers*). Στην άσκηση θα γίνει υλοποίηση: **α.)** με 1 επίπεδο και K κρυμμένους νευρώνες και **β.)** με 2 επίπεδα και K1, K2 κρυμμένους νευρώνες αντίστοιχα. Οι τιμές των K, K1, K2 είναι 200, 200, 100.

Για να γίνει η κατηγοριοποίηση της κλάσης *'type'* μετατράπηκε σε numeric με τη χρήση της συνάρτησης LabelEncoder() ώστε να είναι συμβατό με το νευρωνικό δίκτυο.

```
type = preprocessing.LabelEncoder()
type.fit(train_data['type'])
train_data['type'] = type.transform(train_data['type'])
train_data.head()
```

	bone_length	rotting_flesh	hair_length	has_soul	color	type
0	0.354512	0.350839	0.465761	0.781142	0.500000	1
1	0.575560	0.425868	0.531401	0.439899	0.666667	2
2	0.467875	0.354330	0.811616	0.791225	0.000000	1
3	0.776652	0.508723	0.636766	0.884464	0.000000	1
4	0.566117	0.875862	0.418594	0.636438	0.666667	0

Για να γίνει η κατηγοριοποίηση της κλάσης *'type'* δίνουμε τυχαίες τιμές σε κάθε ένα από τα 3 είδη της χρησιμοποιώντας τη συνάρτηση *get_dummies()* από τη βιβλιοθήκη pandas.

```
X = train_data.drop(['type'], axis=1)
y = pd.get_dummies(train_data['type'])
y.head()
```

	0	1	2
0	0	1	0
1	0	0	1
2	0	1	0
3	0	1	0
4	1	0	0

Για την κατασκευή του νευρωνικού δικτύου χρησιμοποιήθηκε το API Keras του *tensorflow.*

Α.) Στην πρώτη περίπτωση το νευρωνικό δίκτυο αποτελείται από 1 κρυμμένο επίπεδο με 200 νευρώνες και 1 επίπεδο εξόδου με 3 νευρώνες όσες και οι κατηγορίες της κλάσης *'type'.*
Με τη χρήση της συνάρτησης *Sequential()* διευκρινίζεται ότι το μοντέλο δημιουργείται σειριακά και το output κάθε επιπέδου τροφοδοτεί το επόμενο, ενώ με τη συνάρτηση *add()* κατασκευάζονται όλα τα

επίπεδα. Η συνάρτηση *Dense* χρησιμοποιείται για να δείξει πως πρόκειται για πλήρως συνδεδεμένο δίκτυο ενώ δέχεται ως παραμέτρους τη συνάρτηση sigmoid για είσοδο και την *softmax* ως έξοδο. Τέλος για τη διαδικασία εκπαίδευσης του συστήματος χρησιμοποιήθηκε η μέθοδος SGD μέσω του *keras.optimizers* με learning rate = 0.5(έδινε καλύτερο accuracy) ενώ η χρήση της *categorical_crossentropy* γίνεται διότι έχουμε κατηγοριοποίηση πολλαπλών κλάσεων.

```
#NeuralNetworkClassifier
model = Sequential()
model.add(Dense(200, input_shape=(X_train.shape[1],))) #1o layer
model.add(Dense(3, activation='softmax')) #output layer

opt = keras.optimizers.SGD(learning_rate=0.5) #Stochastic Gradient Descend

model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

Παρακάτω φαίνεται η εκπαίδευση του μοντέλου με 10 εποχές. Το βέλτιστο accuracy σύμφωνα με τις δοκιμές που πραγματοποιήθηκαν επετεύχθη με 30 εποχές.

```
model_data = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10) # ekpaideysi toy neuronikou diktyou
```

```
Epoch 1/10
9/9 [==============================] - 1s 61ms/step - loss: 1.1187 - accuracy: 0.3381 - val_loss: 1.0150 - val_accuracy: 0.3763
Epoch 2/10
9/9 [==============================] - 0s 20ms/step - loss: 1.0040 - accuracy: 0.4353 - val_loss: 0.9567 - val_accuracy: 0.4731
Epoch 3/10
9/9 [==============================] - 0s 18ms/step - loss: 0.8507 - accuracy: 0.6547 - val_loss: 0.8522 - val_accuracy: 0.5699
Epoch 4/10
9/9 [==============================] - 0s 17ms/step - loss: 0.8433 - accuracy: 0.6079 - val_loss: 0.8344 - val_accuracy: 0.5484
Epoch 5/10
9/9 [==============================] - 0s 20ms/step - loss: 0.7820 - accuracy: 0.6223 - val_loss: 0.7633 - val_accuracy: 0.5914
Epoch 6/10
9/9 [==============================] - 0s 15ms/step - loss: 0.7257 - accuracy: 0.6439 - val_loss: 0.6494 - val_accuracy: 0.6882
Epoch 7/10
9/9 [==============================] - 0s 7ms/step - loss: 0.6665 - accuracy: 0.6978 - val_loss: 0.6676 - val_accuracy: 0.6989
Epoch 8/10
9/9 [==============================] - 0s 9ms/step - loss: 0.6804 - accuracy: 0.6942 - val_loss: 0.6229 - val_accuracy: 0.6559
Epoch 9/10
9/9 [==============================] - 0s 7ms/step - loss: 0.7264 - accuracy: 0.6259 - val_loss: 0.6775 - val_accuracy: 0.6344
Epoch 10/10
9/9 [==============================] - 0s 7ms/step - loss: 0.7205 - accuracy: 0.6727 - val_loss: 0.5870 - val_accuracy: 0.7634
```

Τέλος υπολογίζεται το Accuracy rate καθώς και το loss για τις 30 εποχές.

```
Accuracy = model.evaluate(X_test,  y_test)

print('\nTest accuracy:', Accuracy)
```

```
3/3 [==============================] - 0s 3ms/step - loss: 0.5118 - accuracy: 0.8172

Test accuracy: [0.5117911696434021, 0.8172042965888977]
```

Β.) Στη δεύτερη περίπτωση το νευρωνικό δίκτυο αποτελείται από δύο κρυμμένα επίπεδα αποτελούμενα από 200 και 100 κρυμμένους νευρώνες και 3 επίπεδα εξόδου.

Ο τρόπος κατασκευής είναι όμοιος με το προηγούμενο ερώτημα με μόνη διαφοροποίηση την προσθήκη ενός ακόμη επιπέδου εισόδου.

```
#NeuralNetworkClassifier
model2 = Sequential()
model2.add(Dense(200, input_shape=(X_train.shape[1],))) #1o layer
model2.add(Dense(100, activation='sigmoid')) #2o layer
model2.add(Dense(3, activation='softmax')) #output layer

opt2 = keras.optimizers.SGD(learning_rate=0.5) #Stochastic Gradient Descend

model2.compile(optimizer=opt2, loss='categorical_crossentropy', metrics=['accuracy'])
```

Η βέλτιστη ακρίβεια επετεύχθη με 40 εποχές, ωστόσο παρατηρείται πως με ένα νευρώνα το δίκτυο αποδίδει καλύτερα αφού έχει καλύτερη ακρίβεια.

```
Accuracy2 = model2.evaluate(X_test,  y_test)

print('\nTest accuracy:', Accuracy2)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.5354 - accuracy: 0.7849

Test accuracy: [0.5354016423225403, 0.7849462628364563]
```

ΜΕΘΟΔΟΣ 3 – Μηχανές Διανυσματικής Υποστήριξης (ΜΔΥ)

Ανήκουν στην οικογένεια των μεθόδων ταξινόμησης βασιζόμενων σε πυρήνες. Οι ΜΔΥ μεταφέρουν τα δεδομένα στο «χώρο χαρακτηριστικών», στον οποίο τα δεδομένα είναι γραμμικά διαχωριζόμενα και έχουν μεγαλύτερες διαστάσεις, όπου γίνεται ο υπολογισμός ενός πολυεπίπεδου μέγιστου διαχωρισμού των δύο τάξεων. Η μεταφορά αυτή επιτυγχάνεται με τη χρήση πυρήνων. Στην άσκηση έγινε χρήση:

α.) γραμμικής συνάρτησης πυρήνα (linear kernel) και **β.)** Gaussian συνάρτηση πυρήνα (RBF kernel)

Στην υλοποίηση των συναρτήσεων χρησιμοποιήθηκε η τεχνική *one-versus-all* καθώς πρόκειται για κατασκευή δυαδικών ταξινομητών έναν για κάθε κατηγορία(multiclass classification).

Όσον αφορά την γραμμική συνάρτηση πυρήνα χρησιμοποιήθηκε η συνάρτηση της Sklearn SVC() με παράμετρο *'kernel=linear'* ενώ η τεχνική one versus all είναι η default τεχνική γι αυτό και δεν χρειάζεται επιπλέον πέρασμα παραμέτρων. Στη συνέχεια γίνεται η διαδικασία της εκπαίδευσης μέσω της συνάρτησης fit() της sklearn και τέλος βρίσκουμε το y_pred το οποίο αποτελεί την πρόβλεψη της κατηγορίας για κάθε δεδομένο. Συγκρίνοντας το με το y_test υπολογίζονται τα accuracy και f1score όπως φαίνονται παρακάτω.

```
#SVCclassifier(kernel)
clf = SVC(kernel='linear', probability=True)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("\nF1 Score is: " + str(metrics.f1_score(y_test, y_pred, average= "weighted")))
print("\nAccuracy Score is: " + str(metrics.accuracy_score(y_test, y_pred)))
```

```
F1 Score is: 0.7089093701996928

Accuracy Score is: 0.7204301075268817
```

Ομοίως εργαζόμαστε για την Gaussian kernel ωστόσο γίνεται χρήση της GridSearch() συνάρτησης για να βρεθεί η πιο αποδοτική παράμετρος που δίνει καλύτερο accuracy score. Δόθηκαν οι τιμές 1,2,3,5,8,10 και επιλέχθηκε η τιμή C=1 ως βέλτιστη.

```
#SVCclassifier(Gaussian)
params = {'kernel':['rbf'], 'C':[ 1,2,3,5,8,10]}
svc = SVC(probability = True, random_state = 0)
#Using GridSearch to find the best regularization parameter (C) that gives the best score
clf = GridSearchCV(svc, param_grid = params, scoring = accuracy_scorer, cv = 5, n_jobs = -1)
clf.fit(X_train, y_train)
print('Best score: {}'.format(clf.best_score_))
print('Best parameters: {}'.format(clf.best_params_))

svc_best = SVC(C = 1,kernel = 'rbf', probability = True, random_state = 0)
```

```
Best score: 0.7083766233766233
Best parameters: {'C': 1, 'kernel': 'rbf'}
```

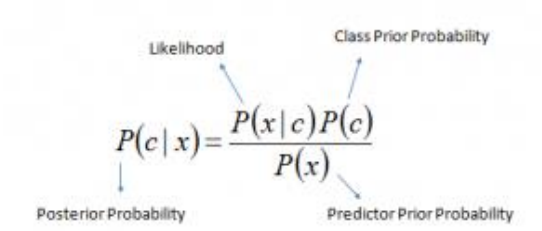
```
y_pred = clf.predict(X_test)
print("\nF1 Score is: " + str(metrics.f1_score(y_test, y_pred, average= "weighted")))
print("\nAccuracy Score is: " + str(metrics.accuracy_score(y_test, y_pred)))
```

```
F1 Score is: 0.7957496398413566

Accuracy Score is: 0.7956989247311828
```

ΜΕΘΟΔΟΣ 4 – Naive Bayes

Στην άσκηση υλοποιήθηκαν 2 μορφές του αλγορίθμου ταξινόμησης Naive Bayes. Αρχικά χρησιμοποιήθηκε η κανονική κατανομή (Gaussian) για τα 4 πρώτα χαρακτηριστικά συνεχούς κατανομής. Εξαιρέθηκε η κατηγορία 'color' η οποία χρησιμοποιήθηκε για κατηγοριοποίηση με διαφορετική μέθοδο που θα αναλυθεί στη συνέχεια. Ενώ υπήρχε η δυνατότητα χρήσης της έτοιμης συνάρτησης GaussianNB() της βιβλιοθήκης sklearn προτιμήθηκε η κατασκευαστική υλοποίηση της. Δημιουργήθηκαν 3 συναρτήσεις η οποίες απαρτίζουν τον τύπο της κανονικής κατανομής.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Α.) Η prior_prob() η οποία υπολογίζει την πιθανότητα P(y) για όλα τα πιθανά y που πρέπει να κατηγοριοποιηθούν και επιστρέφει έναν πίνακα με όλες τις προγενέστερες πιθανότητες.

```
def prior_prob(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

Β.) Η likelihood_Gaussian() η οποία υπολογίζει τη δεσμευμένη πιθανότητα της πρόβλεψης της κατηγορίας δεδομένου της κλάσης που κατηγοριοποιείται P(x|y) χρησιμοποιώντας ως βάση την Gaussian απόσταση. Για να συμβεί αυτό υπολογίζονται ο μέσος όρος (mean) και η διακύμανση (std) για κάθε μια κατηγορία j και συνιστώσα i της διανυσματικής έκφρασης. Επομένως υπολογίζεται η παρακάτω έκφραση:

$$p(x|\omega_j) = \prod_{i=1}^{d} p(x_i|\omega_j) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{-\frac{(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

Παρατίθεται ο αντίστοιχος κώδικας.

```
def likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_val-mean)**2 / (2 * std**2 )))
    return p_x_y
```

Γ.) Η naive_bayes_gaussian() η οποία υπολογίζει το γινόμενο όλων των πιθανοτήτων των προηγούμενων δύο συναρτήσεων για κάθε X και εν συνεχεία βρίσκει το μέγιστο και το αποθηκεύει στον τελικό πίνακα Y_pred() που θα χρησιμοποιηθεί για έλεγχο του set.

```
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = prior_prob(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= likelihood_gaussian(df, features[i], x[i], Y, labels[j])

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

Τέλος, γίνεται ο διαχωρισμός του set σε test και train, εφαρμόζεται ο αλγόριθμος naïve_bayes_gaussian και ελέγχεται το μοντέλο εκτυπώνοντας το Accuracy score και f1 score αντίστοιχα. Ο κώδικας καθώς και τα αποτελέσματα φαίνονται παρακάτω:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(traindata, test_size=0.25, random_state=0)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_gaussian(train, X=X_test, Y="type")

F1 = (metrics.f1_score(Y_test, Y_pred, average= "weighted"))
ACC1 = (metrics.accuracy_score(Y_test, Y_pred))

print("\nF1 Score is: " + str(F1))
print("\nAccuracy Score is: " + str(ACC1))
```

```
F1 Score is: 0.7857547093540302

Accuracy Score is: 0.7849462365591398
```

Β.) Η κατηγοριοποίηση με βάση το 5^{ο} χαρακτηριστικό (‘color’) έγινε με χρήση της πολυωνυμικής κατανομής (multinomial distribution). Για την υλοποίηση της χρησιμοποιήθηκε η έτοιμη συνάρτηση MultinomialNB() της βιβλιοθήκης sklearn καθώς θεωρήθηκε πιο εύκολη στην εκπαίδευση του μοντέλου αφού πρόκειται για μία μόνο κατηγορία.

```
#multinomial distribution for the 5th field ('color')
X2 = train_data.drop(['bone_length','rotting_flesh', 'hair_length','has_soul','type'], axis=1)
```

```
X2_train, X2_test, y_train, y_test = train_test_split(X2, train_data['type'], test_size = 0.25, random_state = 0)
```

```
#Naive-Bayes(Multinomial)
clf = MultinomialNB()
clf.fit(X2_train, y_train)
y2_pred = clf.predict(X2_test)
F2 = (metrics.f1_score(y_test, y2_pred, average= "weighted"))
ACC2 = (metrics.accuracy_score(y_test, y2_pred))

print("\nF1 Score is: " + str(F2))
print("\nAccuracy Score is: " + str(ACC2))
```

```
F1 Score is: 0.1306451612903226

Accuracy Score is: 0.2903225806451613
```

Όπως παρατηρείται τα f1 score και Accuracy score είναι αρκετά χαμηλά κάτι που ήταν αναμενώμενο καθώς η κατηγοριοποίηση με μία μόνο κλάση δεν μπορεί να προσεγγίσει την πραγματική τιμή λόγω έλλειψης δεδομένων και επιπλέον κατηγοριών που συμβάλουν στην καλύτερη εκπαίδευση του μοντέλου.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Συγκρίνοντας τα f1 scores και Accuracy scores των 4 μεθόδων που υλοποιήθηκαν διαπιστώθηκε ότι η βέλτιστη μέθοδος ήταν η χρήση του Νευρωνικού δικτύου με ένα κρυμμένο επίπεδο και 200 κρυμμένους νευρώνες καθώς το accuracy score του πλησίασε το 82% έχοντας μια μικρή διαφορά από τις υπόλοιπες μεθόδους.