Πρώτη Σειρά Ασκήσεων

Στυλιανός Σύρρος: 4805

Διάβασμα Αρχείων

Στην παρακάτω εικόνα φαίνεται ο τρόπος με τον οποίο διαβάζουμε τα δεδομένα των δύο αρχείων.

```
24  airlines_delay = pd.read_csv("airlines_delay.csv")
25  sample_size = int(len(airlines_delay) * 0.01)
26  random_indices = random.sample(range(len(airlines_delay)), sample_size)
27  airlines_delay = airlines_delay.iloc[random_indices]
28  mobile_data = pd.read_csv("train.csv")
29
30  le = LabelEncoder()
31  airlines_delay['Airline'] = le.fit_transform(airlines_delay['Airline'])
32  airlines_delay['AirportFrom'] = le.fit_transform(airlines_delay['AirportFrom'])
33  airlines_delay['AirportTo'] = le.fit_transform(airlines_delay['AirportTo'])
34  airlines_delay = airlines_delay.drop('Flight', axis=1)
35
36  X_airline = airlines_delay.iloc[:, :-1].values
37  Y_airline = airlines_delay.iloc[:, -1].values
38  X_mobile = mobile_data.iloc[:, :-1].values
39  Y_mobile = mobile_data.iloc[:, -1].values
40
41  X_train_airline, X_test_airline, y_train_airline, y_test_airline = train_test_split(X_airline, Y_airline, test_size=0.3, random_state=42)
42  X_train_mobile, X_test_mobile, y_train_mobile, y_test_mobile = train_test_split(X_mobile, Y_mobile, test_size=0.3, random_state=42)
```

- Στις γραμμές 24-27 διαβάζουμε το αρχείο airlines_delay.csv και επιλέγουμε μόνο το 1% των δεδομένων τυχαία επειδή ο όγκος πληροφορία ήταν μεγάλος. Στην γραμμή 34 διαβάζουμε το άλλο αρχείο
- Στις γραμμές 31-33 μετατρέπουμε τα αλφαριθμητικά σε ακέραιες τιμές
- Στην γραμμή 34 διαγράφουμε την στήλη Flight
- Στις γραμμές 36-39 διαχωρίζουμε τις διαστάσεις από τις κατηγορίες
- Στις γραμμές 41-42 χωρίζουμε το σύνολο σε σύνολο trainning και σύνολο test και για τα δύο αρχεία.

Μέθοδος K – Nearest Neighbors

Αρχικά εφόσον θέλουμε να χρησιμοποιήσουμε ευκλείδεια απόσταση για τις συνεχείς μεταβλητές και Hamming απόσταση για τις διακριτές μεταβλητές , φτιάξαμε την συνάρτηση custom_distance_airline (για το αρχείο airlines_delay.csv) και custom_distance_train (για το

αρχείο train.csv) υπολογίζει την απόσταση με τον τρόπο αυτό. Παρακάτω φαίνονται αυτές οι συναρτήσεις.

```
dist_cont = DistanceMetric.get_metric('euclidean')
dist_disc = DistanceMetric.get_metric('hamming')

def custom_distance_train(x1, x2):
    x1_disc = x1[:, [1, 3, 5, 17, 18, 19]]
    x1_cont = x1[:, [i for i in range(x1.shape[1]) if i not in [1, 3, 5, 17, 18, 19]]]
    x2_disc = x2[:, [1, 3, 5, 17, 18, 19]]
    x2_cont = x2[:, [i for i in range(x2.shape[1]) if i not in [1, 3, 5, 17, 18, 19]]]


    dist_cont_mat = dist_cont.pairwise(x1_cont, x2_cont)
    dist_disc_mat = dist_disc.pairwise(x1_disc, x2_disc)

    combined_dist = dist_cont_mat + dist_disc_mat
    return combined_dist

def custom_distance_airlines(x1, x2):

    dist_cont_mat = dist_cont.pairwise(x1, x2)

    combined_dist = dist_cont_mat
    return combined_dist
```

Ακολούθως φτιάξαμε την μέθοδο KNN η οποία επιστρέφει το y_pred που είναι η εκτίμηση που θα συγκρίνουμε με τα πραγματικά δεδομένα για να βρούμε το scores των μετρικών που χρησιμοποιούμε. Παρακάτω φαίνεται η συνάρτηση knn.

```
def knn(k, distance_fn, X_train, y_train, X_test):
    dist_mat = distance_fn(X_train, X_test)

    nn_indices = np.argpartition(dist_mat, k, axis=0)[:k, :]

    nn_labels = y_train[nn_indices]

    y_pred = np.empty(X_test.shape[0])
    for i in range(X_test.shape[0]):
        counter = Counter(nn_labels[:,i])
        y_pred[i] = counter.most_common(1)[0][0]

    return y_pred
```

Ακόμα φτιάξαμε την κλάση KNN_Custom για να μπορέσουμε να προσδιορίσουμε ποιες μετρικές να χρησιμοποιήσει για να υπολογίσει τις αποστάσεις , καθώς άμα χρησιμοποιήσουμε την έτοιμη KNeighborsClassifier δεν μπορούσαμε να προσδιορίσουμε ποια απόσταση να χρησιμοποιήσει(και πότε) οπότε χρησιμοποιούσε μόνο την default απόσταση που είναι η ευκλείδεια . Μετά ορίζουμε τις μετρικές του score οι οποίες είναι το accuracy και το f1_score και το αντικείμενο τύπου KNN_Custom το οποίο παίρνει ως παραμέτρους το k που είναι ο αριθμός των Nearest Neighbors και το distance_fn που είναι η συνάρτηση που θα χρησιμοποιήσει για τις αποστάσεις(π.χ. euclidean , hamming , custom_distance . Στην δικιά μας περίπτωση custom_distance) . Αν θέλουμε 10-fold cross validation χρησιμοποιούμε την έτοιμη συνάρτηση cross_val_score για κάθε μετρική ενώ αν

θέλουμε 70 – 30 (train – test) split κάνουμε split τα δεδομένα μας , fit το KNN_Custom μοντέλο και υπολογίζουμε το y_pred . Παρακάτω φαίνεται η κλάση KNN_Custom.

```
class KNN_Custom:
    def __init__(self, k=5, distance_fn='euclidean'):
        self.k = k
        self.distance_fn = distance_fn

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
        return self

    def predict(self, X):
        return knn(self.k, self.distance_fn, self.X_train, self.y_train, X)

    def score(self, X, y, scoring='f1'):
        y_pred = self.predict(X)
        if scoring == 'f1':
            return f1_score(y, y_pred, average='weighted')
        elif scoring == 'accuracy':
            return accuracy_score(y, y_pred)
        else:
            raise ValueError("Invalid scoring metric")

    def get_params(self, deep=True):
        return {'k': self.k, 'distance_fn': self.distance_fn}
```

Τέλος υλοποιήσαμε την συνάρτηση NearestNeighbors η οποία χρησιμοποιεί ότι φτιάξαμε προηγουμένως και μας επιστρέφει τοaccuracy και το f1-score. Παρακάτω φαίνεται η συνάρτηση.

```
def NearestNeighbors(k,distance_fun,X_train=None,y_train=None,X_test=None,y_test=None,X=None,Y=None,type="test_set"):
    knn_custom = KNN_Custom(k=k,distance_fn=distance_fun)

    if type == 'test_set':
        knn_custom.fit(X_train,y_train)
        y_pred = knn_custom.predict(X_test)
        f1 = f1_score(y_test, y_pred, average='weighted')
        accuracy = accuracy_score(y_test, y_pred)
    elif type == 'cross_val':
        scoring = {'accuracy': 'accuracy',
                   'f1_weighted': 'f1_weighted'}
        cv_results = cross_validate(estimator=knn_custom, X=X, y=Y, cv=10, n_jobs=-1, scoring=scoring)
        f1 = cross_val_score(knn_custom, X, Y, cv=10, scoring='f1_macro')
        accuracy = cross_val_score(knn_custom, X, Y, cv=10, scoring='accuracy')
        accuracy = accuracy.mean()


    return accuracy,f1
```

Εφαρμογή K – Nearest Neighbors στο αρχείο airlines_delay.csv

	70-30 TRAIN TEST SPLIT		10-FOLD CROSS VALIDATION	
K	Accuracy	F1	Accuracy	F1
1	0.5293072824156305	0.5284898750179129	0.54483519918723	0.5383769943831483
3	0.5642391947898164	0.5628137769831237	0.5471927062836153	0.5381736912246786
5	0.5571343990526939	0.5535934246333295	0.5587697924254984	0.5482768527111548

10	0.5695677915926584	0.5654972134484836	0.5669217736915996	0.5531071370148148

Εφαρμογή K – Nearest Neighbors στο αρχείο train.csv

	70-30 TRAIN TEST SPLIT		10-FOLD CROSS VALIDATION	
K	Accuracy	F1	Accuracy	F1
1	0.9066666666666666	0.9064125159447122	0.9075	0.9072466650137544
3	0.9133333333333333	0.9132589281334315	0.9215	0.9218870035640446
5	0.9183333333333333	0.9182611006703343	0.9195	0.9195988287304587
10	0.9283333333333333	0.928544091571694	0.9350000000000002	0.9347527903583339

Μέθοδος Naïve Bayes

Για τη μέθοδο Naïve Bayes θέλουμε να χρησιμοποιήσουμε κανονική (Gaussian) κατανομή για κάθε ένα από τα χαρακτηριστικά συνεχούς τιμής και πολυωνυμική (Multinomial) κατανομή για τα διακριτά χαρακτηριστικά . Έτσι φτιάξαμε δύο μεταβλητές που κρατάν τους αριθμούς από τις στήλες του αρχείου που διαβάζουμε , όπου η μία έχει τους αριθμούς των στηλών με χαρακτηριστικά συνεχούς τιμής και η άλλη τους αριθμούς των στηλών με διακριτά χαρακτηριστικά . Ακολούθως φτιάχνουμε 2 pipelines , ένα για κάθε είδος χαρακτηριστικών , και χρησιμοποιούμε ένα preprocessor όπου συνδυάζει τα δυο pipelines . Μετατρέπουμε τα δεδομένα στην μορφή που θέλουμε με την βοήθεια του preprocessor , τα χωρίζουμε σε continuous_data(συνεχή χαρακτηριστικά) και discrete_data(διακριτά χαρακτηριστικά) και εκτελούμε Gaussian κατανομή στα continuous_data και Multinomial κατανομή στα discrete_data . Τέλος αν θέλουμε 10-fold cross validation παίρνουμε τα scores χρησιμοποιώντας την συνάρτηση cross_validate δύο φορές (μία για κάθε score ,accuracy και f1) , ενώ αν θέλουμε 70 – 30 (train – test) split ,κάνουμε split τα δεδομένα και χρησιμοποιούμε τις συναρτήσεις accuracy_score και f1_score για να υπολογίσουμε τα scores. Παρακάτω φαίνεται η συνάρτηση NaiveBayes η οποία υλοποιεί ότι αναφέραμε προηγουμένως.

```python
def NaiveBayes(X_train=None,y_train=None,X_test=None,y_test=None,X=None,Y=None,file=None,type="test_set"):
    if file == 'train.csv':
        continuous_indices = [2,7]
        discrete_indices = [i for i in range(X.shape[1]) if i not in continuous_indices]

        continuous_pipeline = Pipeline([
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ])

        discrete_pipeline = Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('encoder', OrdinalEncoder())
        ])


        preprocessor = ColumnTransformer([
            ('continuous', continuous_pipeline, continuous_indices),
            ('discrete', discrete_pipeline, discrete_indices),
        ])

        preprocessor.fit(X)

        X_transformed = preprocessor.transform(X)

        continuous_data = X_transformed[:, :len(continuous_indices)]
        discrete_data = X_transformed[:, len(continuous_indices):]

        gaussian_nb = GaussianNB()

        multinomial_nb = MultinomialNB()

        gaussian_nb.fit(continuous_data, Y)
        multinomial_nb.fit(discrete_data, Y)
        if type == "test_set":
            accuracyG = accuracy_score(Y, gaussian_nb.predict(continuous_data))
            f1G = f1_score(Y, gaussian_nb.predict(continuous_data), average='weighted')

            accuracyM = accuracy_score(Y, multinomial_nb.predict(discrete_data))
            f1M = f1_score(Y, multinomial_nb.predict(discrete_data), average='weighted')

        elif type == "cross_val":
            scoring = {'accuracy': 'accuracy', 'f1_weighted': 'f1_weighted'}
            gaussian_nb_scores = cross_validate(gaussian_nb, continuous_data, Y, cv=10, scoring=scoring)
            accuracyG = np.mean(gaussian_nb_scores['test_accuracy'])
            f1G = np.mean(gaussian_nb_scores['test_f1_weighted'])
            multinomial_nb_scores = cross_validate(multinomial_nb, discrete_data, Y, cv=10, scoring=scoring)
            accuracyM = np.mean(multinomial_nb_scores['test_accuracy'])
            f1M = np.mean(multinomial_nb_scores['test_f1_weighted'])

        return accuracyG,accuracyM,f1G,f1M
```

```python
    elif file == "airlines_delay.csv":
        continuous_indices = []
        discrete_indices = [i for i in range(X.shape[1]) if i not in continuous_indices]

        continuous_pipeline = Pipeline([
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ])

        discrete_pipeline = Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('encoder', OrdinalEncoder())
        ])


        preprocessor = ColumnTransformer([
            ('continuous', continuous_pipeline, continuous_indices),
            ('discrete', discrete_pipeline, discrete_indices),
        ])

        preprocessor.fit(X)
        X_transformed = preprocessor.transform(X)

        continuous_data = X_transformed[:, :len(continuous_indices)]
        discrete_data = X_transformed[:, len(continuous_indices):]

        multinomial_nb = MultinomialNB()
        multinomial_nb.fit(discrete_data, Y)
        if type == "test_set":
            accuracy = accuracy_score(Y, multinomial_nb.predict(discrete_data))
            f1 = f1_score(Y, multinomial_nb.predict(discrete_data), average='weighted')
        elif type == "cross_val":
            scoring = {'accuracy': 'accuracy', 'f1_weighted': 'f1_weighted'}
            multinomial_nb_scores = cross_validate(multinomial_nb, discrete_data, Y, cv=10, scoring=scoring)
            accuracy = np.mean(multinomial_nb_scores['test_accuracy'])
            f1 = np.mean(multinomial_nb_scores['test_f1_weighted'])

        return accuracy,f1
```

Εφαρμογή Naïve Bayes στο αρχείο airlines_delay.csv

70-30 TRAIN TEST SPLIT		10-FOLD CROSS VALIDATION	
Accuracy	F1	Accuracy	F1

0.5499289267945985	0.5512481475761554	0.5493988735991099	0.550482182756145

Εφαρμογή Naïve Bayes στο αρχείο train.csv

70-30 TRAIN TEST SPLIT		10-FOLD CROSS VALIDATION	
Accuracy	F1	Accuracy	F1
0.4567	0.4419826850041815	0.4557	0.4427529950384546

Μέθοδος Neural Network

Στην παρακάτω εικόνα φαίνεται η συνάρτηση την οποία φτιάξαμε και υλοποιεί το νευρωνικό δίκτυο.

```
def NeuralNetwork(hidden_layers, n_neurons1, n_neurons2=None,X_train=None,y_train=None,X_test=None,y_test=None,X=None,Y=None, activation_hidden=None,type="test_set"):
    if hidden_layers == 1:
        hidden_layer_sizes = (n_neurons1)
    elif hidden_layers == 2:
        if n_neurons2 is not None:
            hidden_layer_sizes = (n_neurons1, n_neurons2)
        else:
            raise ValueError("Invalid number of neurons for the second hidden layer.")
    else:
        raise ValueError("Invalid number of hidden layers. Must be 1 or 2.")


    nn = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
                       activation=activation_hidden,
                       solver='sgd',
                       learning_rate='constant',
                       learning_rate_init=0.01,
                       max_iter=10000,
                       tol=0.001)

    nn.out_activation_ = 'softmax'

    if(type == "test_set"):
        nn.fit(X_train, y_train)
        mean_accuracy = nn.score(X_test, y_test)
    elif type == "cross_val":
        accuracy = cross_val_score(nn, X, Y, cv=10)
        mean_accuracy = accuracy.mean()

    return mean_accuracy
```

Τα ορίσματα που χρησιμοποιούμε στην συνάρτηση είναι κατανοητά. Ανάλογα με το όρισμα type θα εφαρμόζεται είτε η μέθοδος 10-fold cross_vallidation είτε ο τυχαίος διαχωρισμός δεδομένων σε 70% για trainning και 30% για test. Όπως φαίνεται στην συνάρτηση χρησιμοποιούμε την κλάση MLPClassifier της βιβλιοθήκης scikit-learn Θέσαμε τα παρακάτω ορίσματα της κλάσης ως εξής:

- Hidden_layer_sizes: Μία tuple όπου περιέχει τον αριθμό νευρώνων κάθε κρυμμένου επιπέδου
- Activation: Συνάρτηση ενεργοποίησης την οποία δίνει ο χρήστης
- Solver: Θέσαμε sgd για να εφαρμοστεί η μέθοδος Gradient Descent
- Learning_rate_init: Θέσαμε στο 0.01 το οποίο είναι για το πόσο θα αλλάζουν τα βάρη κατά την εκπαίδευση
- Max_iter: Αριθμός εποχών που πρέπει να γίνουν για να τελειώσει η εκπαίδευση
- Tol: Θέσαμε στο 0.001 προκειμένου να σταματήσει η εκπαίδευση όταν το τετραγωνικό σφάλμα είναι μικρότερο από 0.001

Ο αριθμός νευρώνων στο επίπεδο εξόδου θέτεται αυτόματα από την βιβλιοθήκη όταν εκτελείται η μέθοδο fit.

Εφαρμογή Neural Network στο αρχείο airlines_delay.csv

LOGISTIC ACTIVATION FUNCTION					
70-30 TRAIN TEST				10-FOLD CROSS VAL	
K1	K2	ACCURACY	F1-Score	MEAN ACCURACY	F1-Score
50	-	0.5530945833204586	0.3940187385535758	0.5546032312250949	0.39613355832765634
100	-	0.5530822235268671	0.39427464015827424	0.5490061677038968	0.39628996666387567
200	-	0.5139140376355715	0.55310694311405	0.5546810979590988	0.3943572111427122
50	25	0.5529400859005654	0.3937598641075988	0.5545717136254044	0.3957441829208744
100	50	0.5529400859005654	0.3937598641075988	0.5545828374052657	0.39591876039749974
200	100	0.552995704971727	0.39397981334157545	0.5546236246324625	0.39588738339186685

TANH ACTIVATION FUNCTION					
70-30 TRAIN TEST				10-FOLD CROSS VAL	
K1	K2	ACCURACY	F1-Score	MEAN ACCURACY	F1-Score
50	-	0.553038964249297	0.3938790072142997	0.5545828377833555	0.39634361077354574
100	-	0.5530266044557056	0.39443527622434343	0.54370388168521	0.45372012051971133
200	-	0.5530327843525014	0.3947608040579752	0.5110515765000174	0.5030395731325239
50	25	0.5529400859005654	0.3944371778494296	0.5546551424040119	0.39622206113602
100	50	0.5531193029076414	0.3940019811490781	0.5546644121002618	0.39911356118101327
200	100	0.5531254828044372	0.3944924641465437	0.5546644122033773	0.3963123410355083

LOGISTIC ACTIVATION FUNCTION					
70-30 TRAIN TEST				10-FOLD CROSS VAL	
K1	K2	ACCURACY	F1-Score	MEAN ACCURACY	F1-Score
50	-	0.5183333333333333	0.5065879720058647	0.5065	0.4889103408485215
100	-	0.5066666666666667	0.5492516066232078	0.512	0.4786526800899906
200	-	0.5383333333333333	0.4477563868131476 7	0.5010000000000001	0.4896011891046367
50	25	0.5183333333333333	0.4782324345724049 4	0.5105000000000001	0.4954778878559075
100	50	0.5316666666666666	0.4990119498646232	0.5255000000000001	0.5100223851584835
200	100	0.5466666666666666	0.4833069167466721 4	0.52	0.5065890452794952

Εφαρμογή Neural Network στο αρχείο train.csv

TANH ACTIVATION FUNCTION					
70-30 TRAIN TEST				10-FOLD CROSS VAL	
K1	K2	ACCURACY	F1-Score	MEAN ACCURACY	F1-Score
50	-	0.5116666666666667	0.2806473490750729 4	0.446	0.4082959136445737 4
100	-	0.4833333333333334	0.3275744807605582 3	0.4955	0.4343977039485316 7
200	-	0.475	0.4623167403054556 6	0.4615	0.4565905892933267 4
50	25	0.3183333333333336	0.3846023940562459 7	0.3934999999999996	0.3877049841648801
100	50	0.495	0.3319912235011353 6	0.4395000000000006	0.4152471061079445
200	100	0.5066666666666667	0.1324006972850757	0.4514999999999996	0.436754358002402

Μέθοδος SVM (Support Vector Machine)

Στην παρακάτω εικόνα φαίνεται η συνάρτηση SVM η οποία υλοποιεί της λειτουργίες που ζητήθηκαν.

```
airlines_delay = pd.read_csv("airlines_delay.csv")
sample_size = int(len(airlines_delay) * 0.01)
random_indices = random.sample(range(len(airlines_delay)), sample_size)
airlines_delay = airlines_delay.iloc[random_indices]
mobile_data = pd.read_csv("train.csv")

le = LabelEncoder()
airlines_delay['Airline'] = le.fit_transform(airlines_delay['Airline'])
airlines_delay['AirportFrom'] = le.fit_transform(airlines_delay['AirportFrom'])
airlines_delay['AirportTo'] = le.fit_transform(airlines_delay['AirportTo'])
airlines_delay = airlines_delay.drop('Flight', axis=1)

X_airline = airlines_delay.iloc[:, :-1].values
Y_airline = airlines_delay.iloc[:, -1].values
X_mobile = mobile_data.iloc[:, :-1].values
Y_mobile = mobile_data.iloc[:, -1].values

X_train_airline, X_test_airline, y_train_airline, y_test_airline = train_test_split(X_airline, Y_airline, test_size=0.3, random_state=42)
X_train_mobile, X_test_mobile, y_train_mobile, y_test_mobile = train_test_split(X_mobile, Y_mobile, test_size=0.3, random_state=42)
```

Όταν έχουμε 2 κατηγορίες (αρχείο airlines_delay) χρησιμοποιούμε την την στρατηγική ‘ovo’ και για πυρήνα rbf και για πυρήνα linear, διότι ο default είναι ο ovr. Η κλάση SVC που χρησιμοποιούμε έχει τα παρακάτω πεδία

- Kernel: ο πυρήνας που θα χρησιμοποιήσουμε (είτε linear είτε rbf)
  - Σε περίπτωση που χρησιμοποιήσουμε rbf έχουμε και την παράμετρο gamma
- Decision_function_shape: για να θέσουμε την αρχιτεκτονική one-versus-all
- C: επηρεάζει το margin

Επίσης προτού χρησιμοποιήσουμε την συνάρτηση που φτιάξαμε πρέπει να κάνουμε Scale τα δεδομένα επειδή υπολογίζονται αποστάσεις.

Στις γραμμές 305-314 ανάλογα ποια μέθοδο χρησιμοποιούμε για εκπαίδευση και test υπολογίζουμε επιπλέον και το accuracy και f1-score.

Εφαρμογή SVM στο αρχείο airlines_delay.csv

70-30 TRAIN TEST			10-FOLD CROSS VALIDATION	
C	ACCURACY	F1-SCORE	MEAN ACCURACY	F1-SCORE
0.01	0.5777276519482125	0.5584483725741101	0.5804161091781331	0.5613008784843746
0.1	0.5777338318450082	0.5584458475878042	0.5804086932561485	0.5612993169890511
1	0.5777276519482125	0.5584535721683915	0.5804105472366448	0.5612946324987503
4	0.5777214720514168	0.5584381229250552	0.5804161091437614	0.5613001245938432

10	0.5783580014213763	0.5567893700565896	0.5802492511053409	0.5603415639925611

RBF KERNEL					
70-30 TRAIN TEST				10-FOLD CROSS VAL	
C	gamma	ACCURACY	F1-Scope	MEAN ACCURACY	F1-Scope
0.01	0.01	0.24333333333333335	0.09524575513851653	0.7084999999999999	0.7124918722279729
0.01	0.1	0.24333333333333335	0.09524575513851653	0.6825	0.6924207876406397
0.01	1	0.09524575513851653	0.09524575513851653	0.253	0.11654717168600941
0.01	5	0.24333333333333335	0.09524575513851653	0.2514999999999999 5	0.10569977361889911
0.01	10	0.24333333333333335	0.09524575513851653	0.296	0.2469459128953125
0.1	0.01	0.49333333333333335	0.48035402885908707	0.7194999999999999	0.7240657586871548
0.1	0.1	0.24333333333333335	0.09537360178970918	0.6825	0.6922505922634586
0.1	1	0.24333333333333335	0.09524575513851653	0.253	0.11654717168600941
0.1	5	0.24333333333333335	0.09524575513851653	0.2514999999999999 5	0.10569977361889911
0.1	10	0.24333333333333335	0.09524575513851653	0.2784999999999999 7	0.21356397497482954
1	0.01	0.9033333333333333	0.9040679733914101	0.931	0.931229334525816
1	0.1	0.8066666666666666	0.8088533366245211	0.8404999999999999	0.8419432768070635
1	1	0.245	0.10459418921183626	0.312	0.28864869013597005
1	5	0.24333333333333335	0.09524575513851653	0.2505	0.1699222457993575
1	10	0.24333333333333335	0.09524575513851653	0.251	0.1038157582173359
4	0.01	0.925	0.9250807867874232	0.9215	0.9212349628515061
4	0.1	0.8183333333333334	0.8198184353864177	0.8370000000000001	0.838081800437288

4	1	0.24666666666666667	0.10796795304760423	0.3259999999999996	0.30890427755387534
4	5	0.24333333333333335	0.09524575513851653	0.253	0.14303245851311913
4	10	0.24333333333333335	0.09524575513851653	0.2505	0.10368737071397402
10	0.01	0.9083333333333333	0.9084269500120744	0.9319999999999998	0.9319455992868556

Εφαρμογή SVM στο αρχείο train.csv

10	0.1	0.8183333333333334	0.8198184353864177	0.8375	0.8385303466874743
10	1	0.24666666666666667	0.10796795304760423	0.3259999999999996	0.30890427755387534
10	5	0.24333333333333335	0.09524575513851653	0.253	0.14303245851311913
10	10	0.24333333333333335	0.09524575513851653	0.2505	0.10368737071397402

LINEAR KERNEL				
70-30 TRAIN TEST			10-FOLD CROSS VAL	
C	ACCURACY	F1-Score	MEAN ACCURACY	F1-Score
0.01	0.905	0.9061022053037766	0.9259999999999999	0.9263723415292042
0.1	0.9383333333333334	0.9384500710825442	0.9514999999999999	0.9514233821942482
1	0.9533333333333334	0.9534558763219786	0.9604999999999999	0.9604319918792089
4	0.9583333333333334	0.9583630559238058	0.9649999999999999	0.9649875638272258
10	0.9666666666666667	0.9667887792152499	0.9700000000000001	0.9699870632199403

Βέλτιστη Μέθοδος

Όπως φαίνεται και από τις μετρήσεις που κάναμε η βέλτιστη μέθοδο είναι SVM με πυρήνα linear, όπου μας έδινε accuracy > 0,90 και f1-score > 0,90 σε αντίθεση με τις άλλες μεθόδους.