



Documentation

Αναγνώριση Προτύπων

Πέτρος-Ευστάθιος Φατούρος Π18164

----ΠΕΡΙΕΧΟΜΕΝΑ----

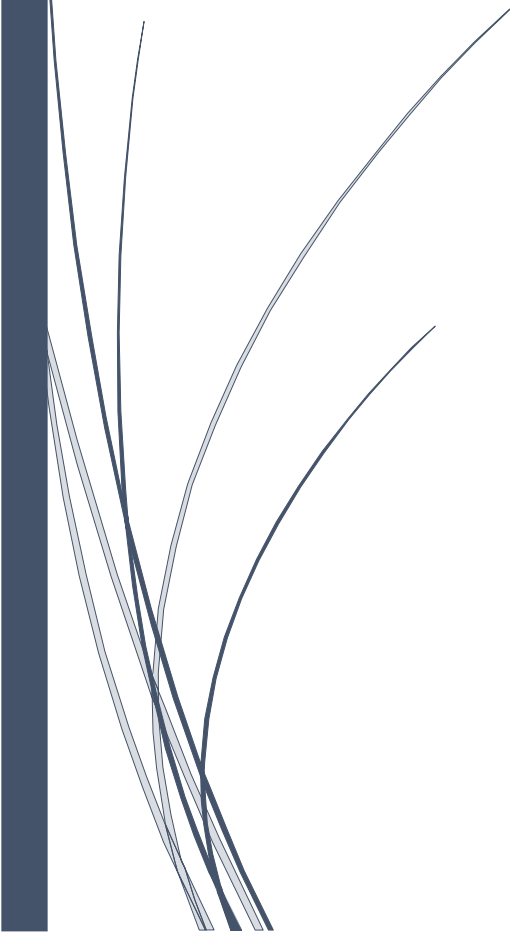
Επεξεργασία δεδομένων → σελίδες 2-6

1° ερώτημα (αλγόριθμος ελαχίστων μέσων τετραγώνων) → σελίδες 7-13

2° ερώτημα (αλγόριθμος ελαχίστων τετραγώνων) → σελίδες 14-15

3° ερώτημα (πολυστρωματικό νευρωνικό δίκτυο) → σελίδες 16-18

Οδηγίες και παράδειγμα εκτέλεσης του προγράμματος → σελίδες 19-25



Επεξεργασία δεδομένων

Ακολουθεί η επεξήγηση του κώδικα που επεξεργάζεται τα δεδομένα από το αρχείο *database.sqlite*

Το αρχείο που περιέχει τον κώδικα για την επεξεργασία των δεδομένων ονομάζεται *data_processing.py*

Οι βιβλιοθήκες που χρησιμοποιήθηκαν:

```
import pandas as pd
import sqlite3
```

Συνάρτηση *filter_data()* :

Επιλέγει κάθε στήλη από τον πίνακα *Match* του αρχείου *database.sqlite*. Αφαιρεί εγγραφές που δεν περιέχουν τιμές ή περιέχουν μηδενικές τιμές σε συγκεκριμένες στήλες, αποθηκεύει τον πίνακα σε ένα pandas DataFrame και τον εξάγει σε αρχείο *csv*.

```
def filter_data():
    # Read sqlite query results into a pandas DataFrame (Match)
    connection = sqlite3.connect('database.sqlite')
    match_df = pd.read_sql_query('SELECT * from Match', connection)

    # Remove rows that contain odds with NaN values
    match_df = match_df.dropna(subset=['B365H', 'B365D', 'B365A',
                                       'BWH', 'BWD', 'BWA',
                                       'IWH', 'IWD', 'IWA',
                                       'LBH', 'LBD', 'LBA'])

    # Export Match dataframe to csv
    match_df.to_csv("Match.csv", index=False)
```

Αποθηκεύει τα αποτελέσματα των αγώνων σε ένα pandas DataFrame και τα εξάγει σε αρχείο **csv**.

```
# Create a DataFrame with the results of each match
# 0 -> draw (D)
# 1 -> home team wins (H)
# 2 -> away team wins (A)
match_results_df = match_df[['match_api_id', 'home_team_api_id', 'away_team_api_id']]
match_results_df['result'] = match_df['home_team_goal'] - match_df['away_team_goal']
match_results_df.loc[match_results_df['result'] == 0, 'result'] = 0
match_results_df.loc[match_results_df['result'] > 0, 'result'] = 1
match_results_df.loc[match_results_df['result'] < 0, 'result'] = 2

# Export dataframe to csv
match_results_df.to_csv('match_results.csv', index=False)
```

Αποθηκεύει τις αποδόσεις κάθε στοιχηματικής εταιρίας σε 4 διαφορετικά pandas DataFrames και τα εξάγει σε αρχεία **csv**.

```
# Create 4 DataFrames with the odds of each betting company
B365_odds_df = match_df[['match_api_id', 'home_team_api_id', 'away_team_api_id', 'B365H', 'B365D', 'B365A']]
BW_odds_df = match_df[['match_api_id', 'home_team_api_id', 'away_team_api_id', 'BWH', 'BWD', 'BWA']]
IW_odds_df = match_df[['match_api_id', 'home_team_api_id', 'away_team_api_id', 'IWH', 'IWD', 'IWA']]
LB_odds_df = match_df[['match_api_id', 'home_team_api_id', 'away_team_api_id', 'LBH', 'LBD', 'LBA']]

# Export dataframes to csv
B365_odds_df.to_csv('B365_odds.csv', index=False)
BW_odds_df.to_csv('BW_odds.csv', index=False)
IW_odds_df.to_csv('IW_odds.csv', index=False)
LB_odds_df.to_csv('LB_odds.csv', index=False)
```

Επιλέγει κάθε στήλη από τον πίνακα *Team_Attributes* του αρχείου *database.sqlite*. Αφαιρεί εγγραφές που δεν περιέχουν τιμές ή περιέχουν μηδενικές τιμές σε συγκεκριμένες στήλες, αποθηκεύει τον πίνακα σε ένα pandas DataFrame και τον εξάγει σε αρχείο **csv**.

```
# Read sqlite query results into a pandas DataFrame (Team_Attributes)
connection = sqlite3.connect('database.sqlite')
team_attributes_df = pd.read_sql_query('SELECT * from Team_Attributes', connection)

# Remove rows that contain odds with NaN values
team_attributes_df = team_attributes_df.dropna(subset=['buildUpPlaySpeed', 'buildUpPlayPassing',
                                                         'chanceCreationPassing', 'chanceCreationCrossing',
                                                         'chanceCreationShooting',
                                                         'defencePressure', 'defenceAggression', 'defenceTeamWidth'])

# Export Team_Attributes dataframe to csv
team_attributes_df.to_csv('Team_Attributes.csv', index=False)
```

Αποθηκεύει τα 8 χαρακτηριστικά κάθε ομάδας σε ένα pandas DataFrames. Πολλαπλές εγγραφές περιέχουν χαρακτηριστικά της ίδιας ομάδας. Ενσωματώνει τις εγγραφές αυτές υπολογίζοντας το μέσο όρο των χαρακτηριστικών της εκάστοτε ομάδας. Εξάγει το DataFrame σε αρχείο csv.

```
# Create a DataFrame with each team's 8 characteristics
team_characteristics_df = team_attributes_df[['team_api_id', 'buildUpPlaySpeed', 'buildUpPlayPassing',
                                              'chanceCreationPassing', 'chanceCreationCrossing',
                                              'chanceCreationShooting',
                                              'defencePressure', 'defenceAggression', 'defenceTeamWidth']]

# Multiple records correspond to the same team (based on different dates)
# Merge these records by calculating the average value of each team's attributes
column_names = ['buildUpPlaySpeed', 'buildUpPlayPassing',
                'chanceCreationPassing', 'chanceCreationCrossing', 'chanceCreationShooting',
                'defencePressure', 'defenceAggression', 'defenceTeamWidth']
team_characteristics_df = team_characteristics_df.groupby('team_api_id', as_index=False)[column_names].mean()

# Export dataframe to csv
team_characteristics_df.to_csv('team_characteristics.csv', index=False)
```

Δημιουργεί ένα pandas DataFrame στο οποίο συγκεντρώνει τα δεδομένα για την είσοδο του πολυστρωματικού νευρωνικού δικτύου και το εξάγει σε αρχείο csv.

Πρώτα ορίζει λίστα με τα ονόματα των στηλών.

```
# Create a DataFrame which represents the input layer of the multi-layer neural network

# Define column names
column_names = ['H_buildUpPlaySpeed', 'H_buildUpPlayPassing', # home team characteristics
                'H_chanceCreationPassing', 'H_chanceCreationCrossing', 'H_chanceCreationShooting',
                'H_defencePressure', 'H_defenceAggression', 'H_defenceTeamWidth',
                'A_buildUpPlaySpeed', 'A_buildUpPlayPassing', # away team characteristics
                'A_chanceCreationPassing', 'A_chanceCreationCrossing', 'A_chanceCreationShooting',
                'A_defencePressure', 'A_defenceAggression', 'A_defenceTeamWidth',
                'B365H', 'B365D', 'B365A', # B365 odds
                'BWH', 'BWD', 'BWA', # BWH odds
                'IWH', 'IWD', 'IWA', # IW odds
                'LBH', 'LBD', 'LBA'] # LBH odds
```

Έπειτα συγκεντρώνει τα δεδομένα και τα φέρνει στην επιθυμητή τους μορφή.

```
# Gather the data of the DataFrame
data = []
for index, row in match_results_df.iterrows():
    record = []

    # home team attributes
    home_team_id = row['home_team_api_id']
    home_team_attributes_df = team_characteristics_df.loc[team_characteristics_df['team_api_id'] == home_team_id]
    record.append(home_team_attributes_df.values.tolist())

    # away team attributes
    away_team_id = row['away_team_api_id']
    away_team_attributes_df = team_characteristics_df.loc[team_characteristics_df['team_api_id'] == away_team_id]
    record.append(away_team_attributes_df.values.tolist())

    match_id = row['match_api_id']
    # B365 odds
    match_B365_odds_df = B365_odds_df.loc[B365_odds_df['match_api_id'] == match_id]
    record.append(match_B365_odds_df.values.tolist())

    # BW odds
    match_BW_odds_df = BW_odds_df.loc[BW_odds_df['match_api_id'] == match_id]
    record.append(match_BW_odds_df.values.tolist())

    # IW odds
    match_IW_odds_df = IW_odds_df.loc[IW_odds_df['match_api_id'] == match_id]
    record.append(match_IW_odds_df.values.tolist())

    # LB odds
    match_LB_odds_df = LB_odds_df.loc[LB_odds_df['match_api_id'] == match_id]
    record.append(match_LB_odds_df.values.tolist())

    # Convert record (nested list) to a flat list
    flat_record = flatten_nested_list(record)

    # Remove unnecessary values ('home_team_id', 'away_team_id', 'match_id')
    flat_record = list(
        filter(lambda elem: elem != home_team_id and elem != away_team_id and elem != match_id, flat_record))

    data.append(flat_record)
```

Τέλος ορίζει το DataFrame, αφαιρεί εγγραφές που δεν περιέχουν τιμές και το εξάγει σε αρχείο **csv**.

```
# Create the DataFrame
MLNN_input_layer_df = pd.DataFrame(data=data, columns=column_names)

# Add label ('result') column to DataFrame
MLNN_input_layer_df['result'] = match_results_df['result']

# Team_Attributes table does not contain the attributes of all the teams in the Match table
# Remove missing values
MLNN_feature_layer_df = MLNN_input_layer_df.dropna()

# Export dataframe to csv
MLNN_feature_layer_df.to_csv('MLNN_input.csv', index=False)
```

Συνάρτηση *flatten_nested_list()* :

Παίρνει ως είσοδο λίστα με εμφωλευμένες λίστες και την επιστρέφει σε μονοδιάστατη μορφή.

```
def flatten_nested_list(nested_list):
    # Convert a nested list to a flat list
    flatList = []

    for element in nested_list:
        if isinstance(element, list):
            flatList.extend(flatten_nested_list(element))
        else:
            flatList.append(element)

    return flatList
```

Εκφώνηση (1^ο ερώτημα)

Να υλοποιήσετε τον Αλγόριθμο Ελάχιστου Μέσου Τετραγωνικού Σφάλματος (**Least Mean Squares**), ώστε ο εκπαιδευμένος ταξινομητής να υλοποιεί την συνάρτηση διάκρισης της μορφής

$g_k(\psi_k(\mathbf{m})) : \mathbb{R}^3 \rightarrow \{\mathbf{H}, \mathbf{D}, \mathbf{A}\}$ για κάθε στοιχηματική εταιρεία. Να αναγνωρίσετε την στοιχηματική εταιρεία τα προγνωστικά της οποίας οδηγούν σε μεγαλύτερη ακρίβεια ταξινόμησης.

Λύση

Σκοπός του αλγορίθμου LMS η προσέγγιση του βέλτιστου διανύσματος βαρών ώστε να ελαχιστοποιείται το μέσο τετραγωνικό σφάλμα ανάμεσα στην έξοδο του ταξινομητή (εκτίμηση της έκβασης του αγώνα) και την επιθυμητή έξοδο του ταξινομητή (πραγματική έκβαση του αγώνα)

Στον αλγόριθμο LMS χρησιμοποιείται ως συνάρτηση κόστους η στιγμιαία τιμή του σφάλματος, δηλαδή:

$$C(\vec{W}) : \text{η συνάρτηση κόστους} \quad C(\vec{W}) = \frac{1}{2} e^2(n)$$

\vec{W} : το διάνυσμα των βαρών

$\frac{1}{2}$: χρησιμοποιείται για την διευκόλυνση υπολογισμού της παραγώγου

$e(n)$: το σφάλμα

Διαφοροποιώντας την παραπάνω σχέση ως προς το διάνυσμα των βαρών προκύπτει:

$$\frac{\partial C(\vec{W})}{\partial \vec{W}} = e(n) \frac{\partial e(n)}{\partial \vec{W}}$$

Επειδή ο αλγόριθμος LMS λειτουργεί με ένα γραμμικό νευρώνα, το σφάλμα μπορεί να υπολογιστεί ως εξής:

$$e(n) = d(n) - \vec{X}^T(n) \vec{W}(n)$$

$d(n)$: η επιθυμητή έξοδος του ταξινομητή (πραγματική έκβαση του αγώνα)

$\vec{X}^T(n) \vec{W}(n)$: η έξοδος του ταξινομητή (εκτίμηση της έκβασης του αγώνα)

Διαφοροποιώντας την παραπάνω σχέση ως προς το διάνυσμα των βαρών προκύπτει:

$$\frac{\partial e(n)}{\partial \vec{W}} = -\vec{X}(n)$$

$\vec{X}(n)$: το διάνυσμα των χαρακτηριστικών

Με βάση τα παραπάνω προκύπτει ότι:

$$\frac{\partial C(\vec{W})}{\partial \vec{W}} = -e(n)\vec{X}(n)$$

$\frac{\partial C(\vec{W})}{\partial \vec{W}}$: η εκτίμηση της κλίσης (gradient)

Επομένως:

$$\hat{g} = -\vec{X}(n)e(n)$$

Στον αλγόριθμο LMS το διάνυσμα των βαρών ενημερώνεται για κάθε δείγμα στο διάνυσμα χαρακτηριστικών. Η ενημέρωση του διανύσματος των βαρών πραγματοποιείται σύμφωνα με την παρακάτω σχέση:

$$\vec{W}(n+1) = \vec{W}(n) + \eta \vec{X}(n)e(n)$$

$\vec{W}(n+1)$: η νέα εκτίμηση του διανύσματος των βαρών

$\vec{W}(n)$: η τρέχων εκτίμηση του διανύσματος των βαρών

η : ο ρυθμός μάθησης (learning rate) ή αλλιώς το μέγεθος του βήματος προς την αντίθετη κατεύθυνση της κλίσης (gradient)

Ακολουθεί η επεξήγηση του κώδικα που υλοποιεί τον αλγόριθμο LMS.

Το αρχείο που περιέχει τον κώδικα για τον αλγόριθμο LMS ονομάζεται ***least_mean_squares.py***

Οι βιβλιοθήκες που χρησιμοποιήθηκαν:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```


Συνάρτηση *run()* :

Αρχικά φορτώνει σε pandas DataFrames τις αποδόσεις των 4 στοιχηματικών εταιριών καθώς και τις πραγματικές εκβάσεις κάθε αγώνα. Διαχωρίζει τα διανύσματα των χαρακτηριστικών (αποδόσεις κάθε στοιχηματικής εταιρίας) από τις ετικέτες (πραγματικές εκβάσεις κάθε αγώνα). Συγκεντρώνει τα διανύσματα χαρακτηριστικών σε μια λίστα.

```
def run():  
    # Load 4 DataFrames with the odds of each betting company  
    B365_odds_df = pd.read_csv('B365_odds.csv')[['B365H', 'B365D', 'B365A']]  
    BW_odds_df = pd.read_csv('BW_odds.csv')[['BWH', 'BWD', 'BWA']]  
    IW_odds_df = pd.read_csv('IW_odds.csv')[['IWH', 'IWD', 'IWA']]  
    LB_odds_df = pd.read_csv('LB_odds.csv')[['LBH', 'LBD', 'LBA']]  
  
    # Load DataFrame with results  
    match_results_df = pd.read_csv('match_results.csv')[['result']]  
  
    # Separate features from label  
    B365_features = B365_odds_df.values  
    BW_features = BW_odds_df.values  
    IW_features = IW_odds_df.values  
    LB_features = LB_odds_df.values  
    labels = match_results_df.values  
  
    # Gather the features of all betting companies  
    features_list = [B365_features, BW_features, IW_features, LB_features]
```

Έπειτα δημιουργεί αντικείμενο το οποίο υλοποιεί 10-fold cross validation. Για τις αποδόσεις κάθε στοιχηματικής εταιρίας, υπολογίζει το μέσο όρο ακρίβειας ταξινόμησης κάθε στοιχηματικής εταιρίας. Για κάθε split του 10-fold cross validation, δημιουργεί training sets, testing sets, υπολογίζει την ακρίβεια ταξινόμησης του κάθε set (training και testing) και τυπώνει στην κονσόλα το μέσο όρο ακρίβειας ταξινόμησης της τρέχων στοιχηματικής εταιρίας. Για κάθε training set και test set υλοποιεί την μέθοδο *one vs all* (draw vs home, away / home vs draw, away / away vs home, draw). Το training πραγματοποιείται από τη συνάρτηση *train_lms* και το testing από τη συνάρτηση *test_lms*. Για κάθε στοιχηματική εταιρία, για κάθε split του 10-fold cross validation, για κάθε φάση του *one vs all* τυπώνει στην κονσόλα την ακρίβεια στο training και στο testing του ταξινομητή.

```

# 10-fold cross validation
# https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.KFold.html
K_fold = KFold(n_splits=10, shuffle=True, random_state=6)

# Iterate for the features of each betting company
average_train_accuracy_dict = {} # to store the train accuracy of each betting company
average_test_accuracy_dict = {} # to store the test accuracy of each betting company
current_betting_company = 'B365' # to print current betting company on console
for features in features_list:

    current_split = 1 # to print current split on console
    train_accuracy_list = [] # to store the train accuracy of each split
    test_accuracy_list = [] # to store the test accuracy of each split
    for train_index, test_index in K_fold.split(features, labels):
        # Split data
        x_train, x_test = features[train_index], features[test_index]
        y_train, y_test = labels[train_index], labels[test_index]

        current_one_vs_all = 'draw vs home, away' # to print current 'one vs all' on console
        for i in range(3): # one vs all (home, draw, away)
            # Train classifier
            weights = train_lms(x_train, y_train, i)

            # Get the most recent updated weights
            weights_final = weights[-1]

            # Evaluate classifier
            train_accuracy = test_lms(x_train, y_train, weights_final, i)
            train_accuracy_list.append(train_accuracy)
            test_accuracy = test_lms(x_test, y_test, weights_final, i)
            test_accuracy_list.append(test_accuracy)

            # Print information of current split on console
            print('---- Split', current_split, ':', current_betting_company, ':', current_one_vs_all, '----')
            print('Train accuracy --> ', train_accuracy)
            print('Test accuracy --> ', test_accuracy)
            print()

            # Update 'current_one_vs_all' variable
            if current_one_vs_all == 'draw vs home, away':
                current_one_vs_all = 'home vs draw, away'
            elif current_one_vs_all == 'home vs draw, away':
                current_one_vs_all = 'away vs home, draw'

        current_split += 1 # update variable

    # Print average train and test accuracy on console
    print('-- ', current_betting_company, ' --')
    print('-- Average train accuracy : ', np.average(train_accuracy_list), ' --')
    average_train_accuracy_dict[current_betting_company] = np.average(train_accuracy_list)
    print('-- Average test accuracy : ', np.average(test_accuracy_list), ' --')
    average_test_accuracy_dict[current_betting_company] = np.average(test_accuracy_list)
    print()

    # Update betting company
    if current_betting_company == 'B365':
        current_betting_company = 'BW'
    elif current_betting_company == 'BW':
        current_betting_company = 'IW'
    elif current_betting_company == 'IW':
        current_betting_company = 'LB'

```

Τέλος, τυπώνει την καλύτερη εταιρία στο training και στο testing αντίστοιχα (με βάση την ακρίβεια ταξινόμησης)

```
# Print company is better at training and testing on console
best_training_company = max(average_train_accuracy_dict, key=average_train_accuracy_dict.get)
print('Best at training --> ', best_training_company)
best_testing_company = max(average_test_accuracy_dict, key=average_test_accuracy_dict.get)
print('Best at testing --> ', best_testing_company)
print()
```

Συνάρτηση *train_lms(features, labels, code)* :

Για κάθε δείγμα εισόδου (διάνυσμα χαρακτηριστικών), υπολογίζει το σφάλμα και ενημερώνει το διάνυσμα των βαρών. Επιστρέφει το ιστορικό όλων των βαρών κατά τη διάρκεια της εκπαίδευσης.

```
def train_lms(features, labels, code):
    n = len(features) # number of samples
    lr = 0.001 # learning rate
    x = features # input matrix (2 dimensional array)
    desired = one_vs_all(labels, code) # desired values of each given sample
    errors = [] # history of all errors (2 dimensional array)
    weights_zero = [0, 0, 0] # array with initial weights
    weights = [weights_zero] # history of all weights (2 dimensional array)

    for i in range(n):
        current_desired = desired[i] # desired value of the current sample
        current_x = x[i] # current sample
        current_weights = weights[i] # array with current weights

        # Calculate error
        current_error = calculate_error(current_desired, current_x, current_weights) # current error
        errors.append(current_error)

        # Update coefficients
        new_weights = current_weights + lr * np.dot(current_x, current_error) # array with updated weights
        weights.append(new_weights)

    return weights
```

Συνάρτηση *test_lms(features, labels, w, code)* :

Χρησιμοποιώντας το διάνυσμα των βαρών που προέκυψε από το στάδιο της εκπαίδευσης, παράγεται μία έξοδος από τον ταξινομητή για κάθε δείγμα εισόδου. Έπειτα συγκρίνει τις εξόδους του ταξινομητή (εκτιμήσεις των εκβάσεων των αγώνων) με τις ετικέτες (πραγματικές εκβάσεις των αγώνων) και επιστρέφει την ακρίβεια ταξινόμησης του ταξινομητή.

```
def test_lms(features, labels, w, code):
    n = len(features) # number of samples
    x = features # input matrix (2 dimensional array)
    desired = one_vs_all(labels, code) # desired values of each given sample

    # Compute the output of the classifier (predict labels)
    y = [] # predicted labels
    for i in range(n):
        current_predicted_label = np.dot(x[i].transpose(), w)
        if current_predicted_label < 0:
            current_predicted_label = -1
        elif current_predicted_label > 0:
            current_predicted_label = 1
        y.append(current_predicted_label)

    # Compute accuracy
    # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy\_score.html
    accuracy = accuracy_score(y_true=desired, y_pred=y)

    return accuracy
```

Συνάρτηση *one_vs_all(labels, code)* :

Ανάλογα με τη μεταβλητή *code*, η οποία αντιπροσωπεύει 3 καταστάσεις (draw vs home, away / home vs draw, away / away vs home, draw), διαμορφώνει τις τιμές των ετικετών.

```
def one_vs_all(labels, code):
    # The output of the match that is represented by variable 'code' is labeled with value -1
    # The other two outputs are labeled with value 1
    # code = 0 -> draw (D)
    # code = 1 -> home team wins (H)
    # code = 2 -> away team wins (A)

    desired = []
    for value in labels:
        if code == 0:
            if value == 1: # home team wins
                desired.append(1)
            elif value == 0: # draw
                desired.append(-1)
            elif value == 2: # away team wins
                desired.append(1)
        elif code == 1:
            if value == 1: # home team wins
                desired.append(-1)
            elif value == 0: # draw
                desired.append(1)
            elif value == 2: # away team wins
                desired.append(1)
        elif code == 2:
            if value == 1: # home team wins
                desired.append(1)
            elif value == 0: # draw
                desired.append(1)
            elif value == 2: # away team wins
                desired.append(-1)

    return desired
```

Συνάρτηση *calculate_error(d, x, w)* :

Υπολογίζει και επιστρέφει το σφάλμα.

```
def calculate_error(d, x, w):
    error = d - np.dot(x.transpose(), w)
    return error
```

Εκφώνηση (2^ο ερώτημα)

Να υλοποιήσετε τον Αλγόριθμο Ελάχιστου Τετραγωνικού Σφάλματος (**Least Squares**), ώστε ο εκπαιδευμένος ταξινομητής να υλοποιεί την συνάρτηση διάκρισης της μορφής $g_k(\psi_k(\mathbf{m})) : \mathbb{R}^3 \rightarrow \{\mathbf{H}, \mathbf{D}, \mathbf{A}\}$ για κάθε στοιχηματική εταιρεία. Να αναγνωρίσετε την στοιχηματική εταιρεία τα προγνωστικά της οποίας οδηγούν σε μεγαλύτερη ακρίβεια ταξινόμησης.

Λύση

Σκοπός του αλγορίθμου LS η προσέγγιση του βέλτιστου διανύσματος βαρών ώστε να ελαχιστοποιείται το άθροισμα των τετραγώνων ανάμεσα στην έξοδο του ταξινομητή (εκτίμηση της έκβασης του αγώνα) και την επιθυμητή έξοδο του ταξινομητή (πραγματική έκβαση του αγώνα)

Δηλαδή:

$$\hat{w} = \underset{w}{\operatorname{argmin}} \left(\sum_{i=1}^n \left| y(i) - \sum_{j=1}^p X(i,j)w(j) \right|^2 \right) = \underset{w}{\operatorname{argmin}} (\|y - Xw\|^2)$$

\hat{w} : η προσέγγιση του βέλτιστου διανύσματος βαρών

y : η επιθυμητή έξοδος του ταξινομητή (πραγματική έκβαση του αγώνα)

X : η μήτρα των χαρακτηριστικών

n : το πλήθος των δειγμάτων

p : το πλήθος των στηλών της μήτρας των χαρακτηριστικών

Με την προϋπόθεση ότι οι στήλες της μήτρας των χαρακτηριστικών είναι γραμμικά ανεξάρτητες προκύπτει το εξής:

$$\hat{w} = (X^T X)^{-1} X^T y$$

Ακολουθεί η επεξήγηση του κώδικα που υλοποιεί τον αλγόριθμο LS.

Το αρχείο που περιέχει τον κώδικα για τον αλγόριθμο LS ονομάζεται *least_squares.py*

Οι βιβλιοθήκες που χρησιμοποιήθηκαν:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```

Συνάρτηση *run()* :

Χρησιμοποιείτε η ίδια συνάρτηση *run* από το πρώτο ερώτημα (βλ. σελίδα 9-11). Η μοναδική διαφορά μεταξύ τους είναι ότι το training πραγματοποιείται από τη συνάρτηση *train_ls* και το testing από τη συνάρτηση *test_ls*.

Συνάρτηση *train_ls(features, labels, code)* :

Υπολογίζει και επιστρέφει τη προσέγγιση του βέλτιστου διανύσματος βαρών.

```
def train_ls(features, labels, code):
    x = features # input matrix (2 dimensional array)
    desired = one_vs_all(labels, code) # desired values of each given sample

    weights = np.dot(np.linalg.inv(np.dot(x.transpose(), x)), np.dot(x.transpose(), desired))

    return weights
```

Συνάρτηση *test_ls(features, labels, w, code)* :

Χρησιμοποιείτε η ίδια συνάρτηση *test_lms* από το πρώτο ερώτημα (βλ. σελίδα 12).

Συνάρτηση *one_vs_all(labels, code)* :

Χρησιμοποιείτε η ίδια συνάρτηση *one_vs_all* από το πρώτο ερώτημα (βλ. σελίδα 13).

Εκφώνηση (3^ο ερώτημα)

Να υλοποιήσετε ένα πολυστρωματικό νευρωνικό δίκτυο, ώστε ο εκπαιδευμένος ταξινομητής να υλοποιεί μια συνάρτηση διάκρισης της μορφής $g(\Phi(\mathbf{m})) : \mathbb{R}^{28} \rightarrow \{\mathbf{H}, \mathbf{D}, \mathbf{A}\}$, όπου το $\Phi(\mathbf{m}) \in \mathbb{R}^{28}$ αντιστοιχεί στο πλήρες διάνυσμα χαρακτηριστικών του κάθε αγώνα που δίνεται από την σχέση:

$$\Phi(\mathbf{m}) = [\varphi(\mathbf{h}), \varphi(\alpha), \psi_{B365}(\mathbf{m}), \psi_{BW}(\mathbf{m}), \psi_{IW}(\mathbf{m}), \psi_{LW}(\mathbf{m})]$$

Λύση

Ακολουθεί η επεξήγηση του κώδικα που υλοποιεί το πολυστρωματικό νευρωνικό δίκτυο.

Το αρχείο που περιέχει τον κώδικα για το πολυστρωματικό νευρωνικό δίκτυο ονομάζεται ***multi_layer_neural_network.py***

Οι βιβλιοθήκες που χρησιμοποιήθηκαν:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import plot_confusion_matrix
```

Συνάρτηση ***run()*** :

Αρχικά φορτώνει σε pandas DataFrame τα δεδομένα της εισόδου του πολυστρωματικού νευρωνικού δικτύου. Διαχωρίζει τα διανύσματα των χαρακτηριστικών (χαρακτηριστικά των 2 ομάδων και αποδόσεις κάθε στοιχηματικής εταιρίας) από τις ετικέτες (πραγματικές εκβάσεις κάθε αγώνα). Κανονικοποιεί τα διανύσματα των χαρακτηριστικών (υπολογίζοντας τα Z-score τους)

```
def run():
    # Load input layer DataFrame
    MLNN_feature_layer_df = pd.read_csv('MLNN_input.csv')

    # Separate features from label
    labels = MLNN_feature_layer_df.pop('result').values
    features = MLNN_feature_layer_df.values

    # Normalize features' values by calculating the Z-scores
    features_mean = features.mean()
    features_std = features.std()
    features_norm = (features - features_mean) / features_std
```

Έπειτα δημιουργεί αντικείμενο το οποίο υλοποιεί 10-fold cross validation. Για κάθε split του 10-fold cross validation, δημιουργεί training sets, testing sets, υπολογίζει την ακρίβεια ταξινόμησης του κάθε set (training και testing) και τυπώνει στην κονσόλα τη μέση ακρίβεια ταξινόμησης. Το training και το testing πραγματοποιείται από ένα πολυστρωματικό ταξινομητή perceptron.

```
# 10-fold cross validation
# https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.KFold.html
K_fold = KFold(n_splits=10, shuffle=True, random_state=6)

current_split = 1 # to print current split on console
train_accuracy_list = [] # to store the train accuracy of each split
test_accuracy_list = [] # to store the test accuracy of each split
for train_index, test_index in K_fold.split(features_norm, labels):
    # Split data
    x_train, x_test = features_norm[train_index], features_norm[test_index]
    y_train, y_test = labels[train_index], labels[test_index]

    # Create a Multi-layer Perceptron classifier
    # https://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPClassifier.html
    MLP_clf = MLPClassifier(
        hidden_layer_sizes=(28, 28, 28), activation='relu', solver='adam', batch_size='auto',
        learning_rate='constant', learning_rate_init=0.001, max_iter=2000, shuffle=True, random_state=6,
        tol=0.0001, verbose=False, n_iter_no_change=10)

    # Train the model
    MLP_clf.fit(x_train, y_train)

    # Evaluate the model
    train_accuracy = MLP_clf.score(x_train, y_train)
    train_accuracy_list.append(train_accuracy)
    test_accuracy = MLP_clf.score(x_test, y_test)
    test_accuracy_list.append(test_accuracy)
```

Στο τέλος κάθε split, πέρα από τις πληροφορίες που τυπώνει στην κονσόλα, εμφανίζει και τις αντίστοιχες confusion matrices.

```
# Compute confusion matrix to evaluate the accuracy of the classification
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_confusion_matrix.html
CM_train = plot_confusion_matrix(estimator=MLP_clf, X=x_train, y_true=y_train,
                                display_labels=['Draw', 'Home', 'Away'],
                                cmap=plt.cm.Blues)
CM_test = plot_confusion_matrix(estimator=MLP_clf, X=x_test, y_true=y_test,
                                display_labels=['Draw', 'Home', 'Away'],
                                cmap=plt.cm.Blues)
CM_train.ax_.set_title(f'split {current_split} : train')
CM_test.ax_.set_title(f'split {current_split} : test')

# Print information of current split on console
print('---- Split ', current_split, ' ----')
current_split += 1
print('Train mean accuracy --> ', train_accuracy)
print('Test mean accuracy --> ', test_accuracy)
print()

plt.show()
```

Τέλος τυπώνει στη κονσόλα, τους μέσους όρους ακρίβειας ταξινόμησης (training και testing) των 10 split.

```
# Print average train and test accuracy on console
print('-- Average train accuracy : ', np.average(train_accuracy_list), ' --')
print('-- Average test accuracy : ', np.average(test_accuracy_list), ' --')
```

Οδηγίες και παράδειγμα εκτέλεσης του προγράμματος

Εκτελέστε το αρχείο *main.py*.

Αρχικά θα κληθεί η συνάρτηση *filter_data* από το αρχείο *data_processing.py* και θα επεξεργαστεί τα δεδομένα από το αρχείο *database.sqlite*

Στην κονσόλα θα εμφανιστεί ένα μενού, ανάλογα με την επιλογή σας θα κληθεί η συνάρτηση *run* από το αντίστοιχο αρχείο (*least_mean_squares.py*, *least_squares.py*, *multi_layer_neural_network.py*)

Ο κώδικας του αρχείου *main.py* είναι αυτοεπεξηγηματικός.

```
import data_processing
import least_mean_squares
import least_squares
import multi_layer_neural_network

def run():
    # Print program's menu
    print('---- Main Menu ----')
    print('1 --> Least Mean Squares')
    print('2 --> Least Squares')
    print('3 --> Multi-Layer Neural Network')

    # Prompt user to select an option from the menu
    option = input('Option: ')
    while option != '1' and option != '2' and option != '3':
        option = input('Option: ')
    print()

    # Execute option
    if option == '1':
        least_mean_squares.run()
    elif option == '2':
        least_squares.run()
    elif option == '3':
        multi_layer_neural_network.run()

if __name__ == '__main__':
    data_processing.filter_data()
    run()
```

Σημείωση: πριν εκτελέσετε το αρχείο *main.py*, θα πρέπει τα αρχεία *database.sqlite*, *main.py*, *data_processing.py*, *least_mean_squares.py*, *least_squares.py*, *multi_layer_neural_network.py* να βρίσκονται στον ίδιο κατάλογο και να έχετε εγκατεστημένες τις βιβλιοθήκες *sqlite3*, *pandas*, *numpy*, *matplotlib*, *sklearn*.

Ακολουθούν ενδεικτικά screenshots από την εκτέλεση του προγράμματος.

Εκτελώντας το αρχείο *main.py*:

```
---- Main Menu ----
1 --> Least Mean Squares
2 --> Least Squares
3 --> Multi-Layer Neural Network
Option: option
Option: 1
```

Για τον αλγόριθμο LMS:

Τυπώνεται στη κονσόλα η ακρίβεια του ταξινομητή (training και testing) για κάθε στοιχηματική εταιρία, για 10 splits, για κάθε μία από τις 3 καταστάσεις του one vs all.

Ενδεικτικά, στο παρακάτω screenshot φαίνεται η ακρίβεια του ταξινομητή (training και testing) , για την B365 στοιχηματική εταιρία, για το πρώτο split, για κάθε μία από τις 3 καταστάσεις του one vs all.

```
---- Split 1 : B365 : draw vs home, away ----
Train accuracy -->  0.7473788328387735
Test accuracy -->  0.744103248776146

---- Split 1 : B365 : home vs draw, away ----
Train accuracy -->  0.5969831849653808
Test accuracy -->  0.586559857587895

---- Split 1 : B365 : away vs home, draw ----
Train accuracy -->  0.7217111770524234
Test accuracy -->  0.7187360925678683
```

Για κάθε στοιχηματική εταιρία τυπώνεται στην κονσόλα ο τελικός μέσος όρος ακρίβειας του ταξινομητή (training και testing)

```
-- B365 --  
-- Average train accuracy : 0.6922284609010428 --  
-- Average test accuracy : 0.6915764537076777 --
```

```
-- BW --  
-- Average train accuracy : 0.6987202662638511 --  
-- Average test accuracy : 0.6988319111012832 --
```

```
-- IW --  
-- Average train accuracy : 0.7058879680388941 --  
-- Average test accuracy : 0.70604259391137 --
```

```
-- LB --  
-- Average train accuracy : 0.6892463323141141 --  
-- Average test accuracy : 0.6882382340729892 --
```

Τέλος τυπώνεται στην κονσόλα η καλύτερη στοιχηματική εταιρία στο training και στο testing αντίστοιχα (με βάση την ακρίβεια ταξινόμησης)

```
Best at training --> IW  
Best at training --> IW
```

```
Process finished with exit code 0
```

Παρομοίως για τον αλγόριθμο LS:

Τυπώνεται στη κονσόλα η ακρίβεια του ταξινομητή (training και testing) για κάθε στοιχηματική εταιρία, για 10 splits, για κάθε μία από τις 3 καταστάσεις του one vs all.

Για κάθε στοιχηματική εταιρία τυπώνεται στην κονσόλα ο τελικός μέσος όρος ακρίβειας του ταξινομητή (training και testing)

```
-- B365 --  
-- Average train accuracy : 0.7111780388037605 --  
-- Average test accuracy : 0.7111910633656459 --
```

```
-- BW --  
-- Average train accuracy : 0.7112785955547286 --  
-- Average test accuracy : 0.7110574793633357 --
```

```
-- IW --  
-- Average train accuracy : 0.7108549291456997 --  
-- Average test accuracy : 0.7110573934997001 --
```

```
-- LB --  
-- Average train accuracy : 0.710362031859199 --  
-- Average test accuracy : 0.7103899886699631 --
```

Τέλος τυπώνεται στην κονσόλα η καλύτερη στοιχηματική εταιρία στο training και στο testing αντίστοιχα (με βάση την ακρίβεια ταξινόμησης)

```
Best at training --> BW  
Best at training --> BW  
  
Process finished with exit code 0
```


Για το πολυστρωματικό νευρωνικό δίκτυο:

Τυπώνεται στη κονσόλα η μέση ακρίβεια του ταξινομητή (training και testing) για κάθε split.

```
---- Main Menu ----
1 --> Least Mean Squares
2 --> Least Squares
3 --> Multi-Layer Neural Network
Option: 3

---- Split 1 ----
Train mean accuracy --> 0.4774834073830289
Test mean accuracy --> 0.43139190523198423

---- Split 2 ----
Train mean accuracy --> 0.4741923098019856
Test mean accuracy --> 0.4452122408687068

---- Split 3 ----
Train mean accuracy --> 0.49404859854094674
Test mean accuracy --> 0.41510365251727543

---- Split 4 ----
Train mean accuracy --> 0.4781964785255883
Test mean accuracy --> 0.428923988153998

---- Split 5 ----
Train mean accuracy --> 0.4846689704349734
Test mean accuracy --> 0.4116485686080948
```

```

---- Split 6 ----
Train mean accuracy --> 0.4820909439964895
Test mean accuracy --> 0.4378084896347483

---- Split 7 ----
Train mean accuracy --> 0.4900444298173441
Test mean accuracy --> 0.41609081934846986

---- Split 8 ----
Train mean accuracy --> 0.4813514699429574
Test mean accuracy --> 0.4301234567901235

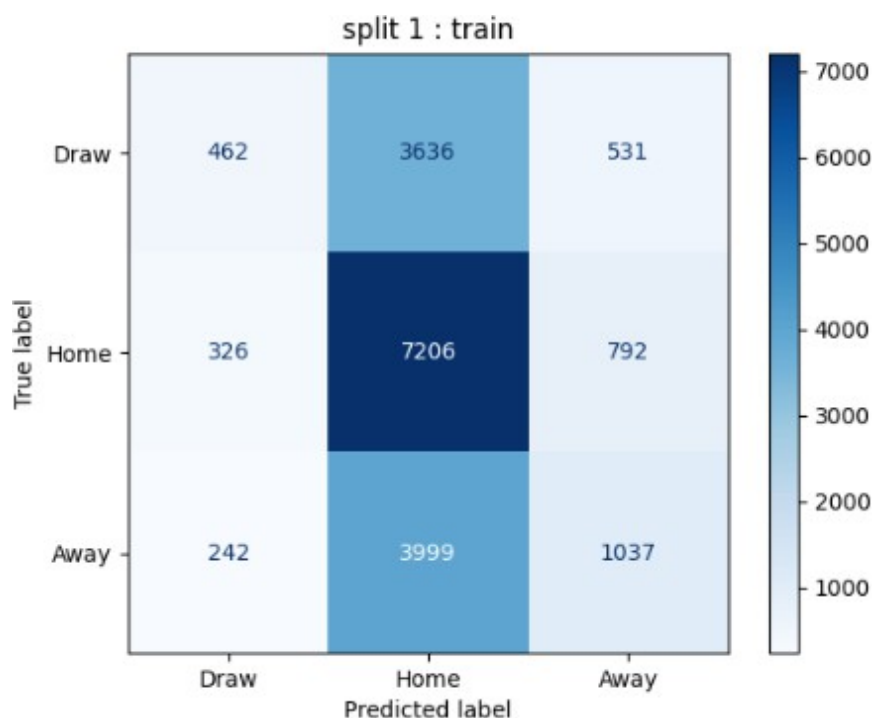
---- Split 9 ----
Train mean accuracy --> 0.47679903466432644
Test mean accuracy --> 0.4301234567901235

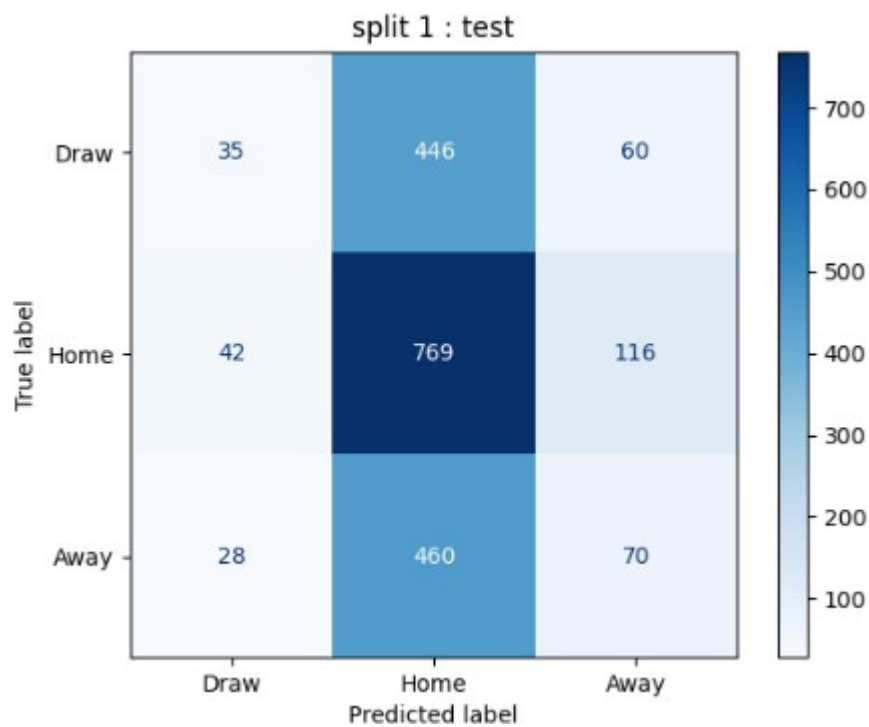
---- Split 10 ----
Train mean accuracy --> 0.474659938569548
Test mean accuracy --> 0.44098765432098763

```

Στο τέλος κάθε split, πέρα από τις πληροφορίες που τυπώνονται στην κονσόλα, εμφανίζονται και οι αντίστοιχες confusion matrices.

Ενδεικτικά, στο παρακάτω screenshot φαίνονται οι confusion matrices του split 1.





Τέλος τυπώνονται στη κονσόλα οι μέσοι όροι ακρίβειας ταξινόμησης (training και testing) των 10 split.

```
-- Average train accuracy : 0.4813535581677188 --  
-- Average test accuracy : 0.42874142322645126 --  
  
Process finished with exit code 0
```