ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Τελική Εργασία Μαθήματος ***Αναγνώριση Προτύπων***

Όνομα φοιτητή – Αρ. Μητρώου	Βραχνής Ιωάννης – Π16016 Σέγκος Νικόλαος - Π16125 Σμύρης Βασίλειος - Π16131
Ημερομηνία παράδοσης	20/9/2021

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1 Εισαγωγή 3
2 Τα δεδομένα και η εξαγωγή τους από την βάση 3
3 Αλγόριθμος Ελάχιστου Μέσου Τετραγωνικού Σφάλματος (Least Mean Squares) 6
3.1 Συνάρτηση ανανέωσης βαρών 6
3.2 Υλοποίηση ταξινομητή 6
4 Αλγόριθμος Ελάχιστου Τετραγωνικού Σφάλματος (Least Squares) 10
4.1 Συνάρτηση ανανέωσης βαρών 10
4.2 Υλοποίηση ταξινομητή 10
5 Πολυστρωματικό Νευρωνικό Δίκτυο 13
6 Επιπλέον Συναρτήσεις 17
6.1 Συνάρτηση fix_outliers 17
6.2 Συνάρτηση k_fold_validate 18
7 Εύρεση ακριβέστερης εταιρείας 24

1 Εισαγωγή

Η υλοποίηση της εργασίας πραγματοποιήθηκε με τη χρήση της γλώσσας προγραμματισμού Python. Για τα ερωτήματα I και II, οι αλγόριθμοι υλοποιήθηκαν εξ ολοκλήρου από εμάς, εφαρμόζοντας του τύπους γραμμικής άλγεβρας για τον υπολογισμό των βαρών των ταξινομητών, οι οποίοι παρουσιάζονται στις αντίστοιχες ενότητες. Για το ερώτημα III, χρησιμοποιήθηκε η κλάση MLPClassifier της βιβλιοθήκης sci-kit learn, ώστε να υλοποιηθεί ένας Multi-Layer Perceptron ταξινομητής. Υλοποιήθηκαν, επίσης, μία συνάρτηση για 10-fold validation των ταξινομητών και μία για τον καθαρισμό υπερβολικά μεγάλων τιμών από τα δεδομένα. Οι βιβλιοθήκες που χρησιμοποιήθηκαν για την εργασία είναι οι παρακάτω:

```
import sqlite3
import math
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
```

2 Τα δεδομένα και η εξαγωγή τους από την βάση

Σε αυτή την ενότητα, παρουσιάζονται τα δεδομένα για την εκπαίδευση και για την αξιολόγηση των ταξινομητών, καθώς και τα SQL queries που χρησιμοποιούνται για την ανάκτησή τους από την βάση. Τα δεδομένα προέρχονται από την ακόλουθη βάση δεδομένων: https://www.kaggle.com/hugomathien/soccer. Η βάση είναι αποθηκευμένη σε ένα αρχείο SQLite.

Για την εργασία, χρησιμοποιούνται δύο από τους πίνακες που περιέχει η βάση, ο πίνακας Match, ο οποίος περιέχει δεδομένα από ποδοσφαιρικούς αγώνες, και ο πίνακας Team_Attributes, ο οποίος περιέχει δεδομένα για τις ομάδες που συμμετείχαν στους αγώνες. Η χρήσιμη πληροφορία για τα πρώτα δύο ερωτήματα, βρίσκεται στις στήλες **B365H**, **B365D**, **B365A**,**BWH**, **BWD**, **BWA**, **IWH**, **IWD**, **IWA**, **LBH**, **LBD**, **LBA** του πίνακα Match. Οι στήλες περιέχουν προγνωστικά για την έκβαση του κάθε αγώνα από τέσσερις στοιχηματικές εταιρίες. Για κάθε εταιρεία υπάρχουν τρία προγνωστικά, για τη νίκη της ομάδας που παίζει εντός έδρας, της ομάδας που παίζει εκτός έδρας και για ισοπαλία. Για το τρίτο ερώτημα, χρησιμοποιούνται οι προηγούμενες στήλες, καθώς και οι στήλες **buildUpPlaySpeed**, **buildUpPlaySpeed**, **buildUpPlayPassing**, **chanceCreationPassing**, **chanceCreationCrossing**, **chanceCreationShooting**, **defencePressure**, **defenceAggregation**, **defenceTeamWidth** του πίνακα Team_Attributes, οι οποίες περιέχουν κάποια στατιστικά για κάθε ομάδα. Όλες αυτές

οι στήλες χρησιμοποιούνται ως χαρακτηριστικά για την εκπαίδευση και την αξιολόγηση των ταξινομητών.

Η διαδικασία εκπαίδευσης όλων των ταξινομητών είναι supervised, που σημαίνει ότι ξέρουμε από πριν όλες τις πιθανές κλάσεις στις οποίες μπορούν να ταξινομηθούν τα διανύσματα χαρακτηριστικών, καθώς και την κλάση του κάθε διανύσματος. Αυτές οι ταμπέλες εκπαίδευσης(training labels), αλλά και οι κλάσεις των διανυσμάτων αξιολόγησης, είναι στην περίπτωση μας τρεις, μία για κάθε πιθανή έκβαση του αγώνα, και προκύπτουν από τις στήλες **home_team_goals** και **away_team_goals**. Οι κλάσεις είναι οι H, A και D, και προκύπτουν από την παρακάτω σχέση:

$$r = \begin{cases} H, \Delta G_m(h,a) > 0; \\ D, \Delta G_m(h,a) = 0; \\ A, \Delta G_m(h,a) < 0. \end{cases}$$

όπου $\Delta G_m(h,a)$ = home_team_goals – away_team_goals.

Η ανάκτηση των δεδομένων γίνεται με τη χρήση SQL query. Ο κώδικας που χρησιμοποιεί τα queries παρουσιάζεται σε παρακάτω ενότητα. Για τα πρώτα δύο ερωτήματα, όπου χρησιμοποιούν μόνο τα προγνωστικά των εταιρειών και τα αποτελέσματα έκβασης των αγώνων, χρησιμοποιείται το παρακάτω query:

```
SELECT CASE
WHEN (home_team_goal - away_team_goal) > 0 THEN "H"
WHEN (home_team_goal away_team_goal) < 0 THEN "A"
ELSE "D" END AS result,
' + company + 'H,
' + company + 'A,
' + company + 'D
FROM Match
WHERE NOT (
B365H IS NULL
OR B365A IS NULL
OR B365D IS NULL
OR BWH IS NULL
OR BWA IS NULL
OR BWD IS NULL
OR IWH IS NULL
OR IWA IS NULL
OR IWD IS NULL
OR LBH IS NULL
OR LBA IS NULL
OR LBD IS NULL
```

```
)
```

Η πρώτη στήλη που επιλέγει το query είναι η στήλη result, η οποία προκύπτει από τις στήλες **home_team_goal** και **away_team_goal**. Για κάθε εγγραφή στον πίνακα Match, η διαφορά τους υπολογίζεται και στήλη result για την εγγραφή παίρνει μία από τις τρεις τιμές έκβασης του αγώνα. Τα δεδομένα από αυτό το query χρησιμοποιούνται από ταξινομητές που δουλεύουν για κάθε εταιρεία ξεχωριστά, επομένως το query ανακτά τις στήλες με τα προγνωστικά μόνο για μία εταιρεία, σύμφωνα με τη μεταβλητή company, η οποία παίρνει τιμές κατά την εκτέλεση του προγράμματος. Το query, επίσης, εξαιρεί τις εγγραφές του πίνακα όπου οι τιμές των προγνωστικών τον εταιρειών είναι κενές. Για το τρίτο ερώτημα, το query είναι το παρακάτω:

```
SELECT CASE
WHEN (home_team_goal - away_team_goal) > 0 THEN "H"
WHEN (home_team_goal -away_team_goal) < 0 THEN "A"
ELSE "D" END AS result,
Home_Team.buildUpPlaySpeed, Home_Team.buildUpPlayPassing,
Home_Team.chanceCreationPassing, Home_Team.chanceCreationCrossing,
Home_Team.chanceCreationShooting, Home_Team.defencePressure,
Home_Team.defenceAggression, Home_Team.defenceTeamWidth,
Away_Team.buildUpPlaySpeed, Away_Team.buildUpPlayPassing,
Away_Team.chanceCreationPassing, Away_Team.chanceCreationCrossing,
Away_Team.chanceCreationShooting, Away_Team.defencePressure,
Away_Team.defenceAggression, Away_Team.defenceTeamWidth,
B365H, B365A, B365D,
BWH, BWA, BWD,
IWH, IWA, IWD,
LBH, LBA, LBD
FROM Match
INNER JOIN Team_Attributes Home_Team
ON Home_Team.team_api_id = Match.home_team_api_id
INNER JOIN Team_Attributes Away_Team
ON Away_Team.team_api_id = Match.away_team_api_id
WHERE NOT (
B365H IS NULL
OR B365A IS NULL
OR B365D IS NULL
OR BWH IS NULL
OR BWA IS NULL
OR BWD IS NULL
OR IWH IS NULL
OR IWA IS NULL
OR IWD IS NULL
```

```
OR LBH IS NULL
OR LBA IS NULL
OR LBD IS NULL
)
```

Η διαφορά του με το προηγούμενο query, είναι ότι αυτό τραβά τα προγνωστικά όλων των εταιρειών, καθώς και τα στατιστικά των δύο ομάδων που συμμετέχουν σε κάθε αγώνα. Η αντιστοίχιση των ομάδων με τον αγώνα γίνεται με βάση τις στήλες **home_team_api_id** και **away_team_api_id**, οι οποίες αποτελούν ξένα κλειδιά στη στήλη **team_api_id** της κάθε ομάδας.

3 Αλγόριθμος Ελάχιστου Μέσου Τετραγωνικού Σφάλματος (Least Mean Squares)

Σε αυτή την ενότητα παρουσιάζεται η υλοποίησή μας για τον ο αλγόριθμο Least Mean Squares(LMS).

3.1 Συνάρτηση ανανέωσης βαρών

Ο αλγόριθμος LMS, έχει φτιαχτεί για να λειτουργεί σε σενάρια όπου δεν είναι όλα τα δεδομένα εκπαίδευσης γνωστά από την αρχή, αλλά λαμβάνονται ένα-ένα σε πραγματικό χρόνο. Ο ταξινομητής ξεκινά με ένα αρχικό διάνυσμα βαρών, το οποίο ανανεώνει κατά τη λήψη κάθε δείγματος. Η συνάρτηση ανανέωσης των βαρών είναι η εξής:

$$w(n+1) = w(n) + \rho * \text{error}^{\text{T}}(i)x(i)$$

Το **w(n)** είναι το τρέχον διάνυσμα βαρών του ταξινομητή και το **w(n+1)** είναι το νέο διάνυσμα «διορθωμένο» διάνυσμα βαρών. Το ρ είναι ο ρυθμός εκμάθησης και το **x(i)** είναι το τελευταίο δείγμα εκπαίδευσης που έχει λάβει ο αλγόριθμος, με βάση το οποίο ανανεώνονται τα βάρη. Το **error(i)**, υπολογίζεται από τον τύπο **error(i)** = **y(i)** – $\mathbf{w^T(n)x(i)}$, όπου **y(i)** είναι η πραγματική κλάση του τρέχοντος δείγματος. Το **error(i)** είναι το σφάλμα ταξινόμησης του ταξινομητή για το δείγμα με τα τρέχον διάνυσμα βαρών του. Είναι η διαφορά της πραγματικής κλάσης του δείγματος και της κλάσης που το ταξινομεί ο ταξινομητής με τα τρέχοντα βάρη.

3.2 Υλοποίηση ταξινομητή

Η συνάρτηση calculate_accuracy_lms εκπαιδεύει έναν ταξινομητή με τη χρήση του αλγόριθμου LMS κι έπειτα δοκιμάζει την ακρίβειά του. Όλες οι συναρτήσεις που υλοποιούν τους ταξινομητές έχουν σχεδιαστεί για να εκτελούνται stand-alone, αλλά και ως μέρη του 10-fold validation. Η συνάρτηση λαμβάνει τέσσερις παραμέτρους. Η μεταβλητή company ορίζει για το ποιας εταιρείας τις προβλέψεις θα τρέξει ο αλγόριθμος. Η k_fold_val ορίζει εάν η συνάρτηση τρέχει στα πλαίσια k-fold validation. Οι μεταβλητές train_set και test_set,

περιέχουν τα δεδομένα εκπαίδευσης και αξιολόγησης αντίστοιχα. Οι δύο τελευταίες μεταβλητές χρησιμοποιούνται όταν η συνάρτηση καλείται από τη συνάρτηση για το k-fold validation(παρουσιάζεται σε παρακάτω ενότητα) για να περαστούν στον αλγόριθμο τα δεδομένα που έχουν χωριστεί για το κάθε fold. Εάν η συνάρτηση καλείται stand-alone, δηλαδή η μεταβλητή k_fold_val έχει τιμή false, οι μεταβλητές train_set και test_set γεμίζουν δεδομένα κατά την εκτέλεση της συνάρτησης. Όλες οι συναρτήσεις ταξινομητών, κατά την stand-alone εκτέλεσή τους, ανακτούν δεδομένα από την SQLite βάση με τη χρήση SQL query. Η συνάρτηση για τον αλγόριθμο LMS ανακτά τα προγνωστικά της εταιρείας που ορίζεται από τη μεταβλητή company και τα αποτελέσματα των αγώνων.

```
def calculate_accuracy_lms(company, k_fold_val = False, train_set = None, tes
t_set = None):
    if k_fold_val is False:
        if company is None:
            print ("\nNo company name supplied. Exiting.")
            quit()

        print ('\nUsing the Least Mean Squares algorithm.')

        print ('\nCompany: ', company)

        conn = sqlite3.connect('database.sqlite')

        query = 'SELECT CASE WHEN (home_team_goal -
 away_team_goal) > 0 THEN "H" WHEN (home_team_goal -
 away_team_goal) < 0 THEN "A" ELSE "D" END AS result, ' + company + 'H, ' + c
ompany + 'A, ' + company + 'D FROM Match WHERE NOT (B365H IS NULL OR B365A IS
 NULL OR B365D IS NULL OR BWH IS NULL OR BWA IS NULL OR BWD IS NULL OR IWH IS
 NULL OR IWA IS NULL OR IWD IS NULL OR LBH IS NULL OR LBA IS NULL OR LBD IS N
ULL)'
        df = pd.read_sql_query(query, conn)
        conn.close()
        df = fix_outliers(df)
        df_shuffled=df.sample(frac=1).reset_index(drop=True)

        rowcount = math.floor(len(df) * training_set_size)
        train_set = df_shuffled.iloc[:rowcount, :]
        test_set = df_shuffled.iloc[(rowcount + 1 ):,:]
```

Τα δεδομένα αποθηκεύονται στη μεταβλητή df σε ένα pandas dataframe. Έπειτα, περνούν από τη συνάρτηση fix_outilers(παρουσιάζεται σε παρακάτω ενότητα), για να αντιμετωπιστούν οι διάφορες πολύ μεγάλες τιμές, και μετά ανακατεύονται ώστε να έχουν μια τυχαία σειρά, ώστε

τα σενάρια εκπαίδευσης και αξιολόγησης να είναι πιο δυναμικά. Ένα ποσοστό των δεδομένων, που ορίζεται από τη μεταβλητή training_set_size (0 < μεταβλητή training_set_size < 1), επιλέγεται ως το σετ δεδομένων εκπαίδευσης, ενώ τα υπόλοιπα δεδομένα ορίζονται ως το σετ δεδομένων αξιολόγησης. Τα δεδομένα αποθηκεύονται στα dataframes train_set και test_set αντίστοιχα, αφού πρώτα ανακατευτούν ξανά.

Στο επόμενο στάδιο, τα δείγματα εκπαίδευσης ανακτώνται από το train_set και αποθηκεύονται σε numpy arrays, ώστε να γίνει χρησιμοποιηθούν από τη συνάρτηση ανανέωσης βαρών σε πράξεις γραμμικής άλγεβρας. Δημιουργούνται τα κενά arrays X και y. Στο array X θα αποθηκεύονται τα δείγματα εκπαίδευσης, και έχει τρεις στήλες, όσα και τα features του κάθε δείγματος. Στο array y θα αποθηκεύονται τα labels των δειγμάτων, οι κλάσεις στις οποίες ανήκουν. Έχει και αυτό τρεις στήλες, καθώς οι κλάσεις θα κωδικοποιούνται σε αλληλουχίες -1 και 1, μήκους 3. Η μεταβλητή w θα λειτουργεί ως το διάνυσμα βαρών του ταξινομητή. Όλα τα βάρη αρχικοποιούνται ως 0.

```
X = np.empty((0,3))
y = np.empty((0,3))
w = np.array([0., 0., 0.])

for i, row in train_set.iterrows():
    X = np.append(X, [[row[1], row[2], row[3]]], axis=0)
    if row[0] == 'H':
        y = np.append(y, [[1., -1., -1.]], axis=0)
    elif row[0] == 'D':
        y = np.append(y, [[-1., 1., -1.]], axis=0)
    else:
        y = np.append(y, [[-1., -1., 1.]], axis=0)

    error = y[i] - (w.T).dot(X[i])
    w_delta = 0.001 * np.dot(np.array([error]).T, np.array([X[i]]))


    w += w_delta
```

Για κάθε εγγραφή στο dataset εκπαίδευσης, αποθηκεύονται στο array X οι τρεις τιμές της, η οποίες αποτελούν τα χαρακτηριστικά του κάθε διανύσματος εκπαίδευσης. Η πρώτη, στήλη, που περιέχει την κλάση του δείγματος, κωδικοποιείται και αποθηκεύεται στο array y. Αφού αποθηκευτεί το δείγμα και το label του στα αντίστοιχα arrays, ξεκινά η διαδικασία ανανέωσης του βάρους. Αρχικά, υπολογίζεται το σφάλμα. Το δείγμα πολλαπλασιάζεται με την ανάστροφη μήτρα του διανύσματος βαρών και ταξινομείται σε μια κλάση σύμφωνα με τα τρέχοντα βάρη του ταξινομητή. Αυτή η «εικασία» αφαιρείται από την πραγματική κλάση του δείγματος. Από αυτές τις πράξεις προκύπτει το σφάλμα της εικασίας και αποθηκεύεται στη μεταβλητή error. Έπειτα, υπολογίζεται η ποσότητα κατά την οποία θα πρέπει να μεταβληθεί το διάνυσμα βαρών του αλγορίθμου. Το δείγμα πολλαπλασιάζεται με το σφάλμα και με το συντελεστή

ρυθμού εκμάθησης, ο οποίος στην προκειμένη περίπτωση είναι 0.01. Το αποτέλεσμα αποθηκεύεται στη μεταβλητή w_delta και προστίθεται στο διάνυσμα βαρών.

Μετά την επεξεργασία όλων των δειγμάτων, ο ταξινομητής έχει ένα διάνυσμα βαρών το οποίο μπορεί να ταξινομήσει με κάποια επαρκή ακρίβεια δείγματα. Το παρακάτω κομμάτι κώδικα, αναλαμβάνει την αξιολόγηση του ταξινομητή. Οι μεταβλητές hit και miss, θα κρατάνε τον αριθμό των επιτυχών και ανεπιτυχών ταξινομήσεων αντίστοιχα. Τα χαρακτηριστικά κάθε εγγραφής του συνόλου test_set πολλαπλασιάζονται με το διάνυσμα βαρών. Αυτό που προκύπτει, είναι ένα διάνυσμα τριών τιμών, όπου η κάθε μία αντιπροσωπεύει την πιθανότητα το δείγμα να ανήκει σε κάθε κλάση.

```
hit = 0
miss = 0

for i, row in test_set.iterrows():
    r = np.array([row[1], row[2], row[3], 1]).dot(w)
    c = np.where(r == r.max())[0][0]
    if c == 0 and row[0] == 'H':
        hit += 1
    elif c == 1 and row[0] == 'D':
        hit += 1
    elif c == 2 and row[0] == 'A':
        hit += 1
    else:
        miss += 1

if k_fold_val is False:
    print("hit: "+str(round((hit / len(test_set) * 100),3))+"%.")
    print("miss: "+str(round( (miss / len(test_set) * 100),3))+"%.")

return (hit / len(test_set) * 100)
```

Από αυτές τις τιμές επιλέγεται η μεγαλύτερη και ελέγχεται αν η θέση της στο διάνυσμα αντιστοιχεί στην πραγματική κλάση του δείγματος. Αν ναι, τότε αυξάνεται η μεταβλητή hit κατά ένα, αλλιώς αυξάνεται η miss. Εάν η συνάρτηση δεν τρέχει ως μέρος 10-fold validation, τότε εκτυπώνονται στατιστικά επιτυχίας και αποτυχίας. Η συνάρτηση επιστρέφει το ποσοστό επιτυχίας του ταξινομητή. Ακολουθεί παράδειγμα της stand-alone εκτέλεσης της συνάρτησης για την εταιρεία B365. Η συνάρτηση καλείται με την εντολή **calculate_accuracy_lms('B365')**.

```
Using the Least Mean Squares algorithm.

Company:  B365
hit: 45.805%.
miss: 54.195%.
```

4 Αλγόριθμος Ελάχιστου Τετραγωνικού Σφάλματος (Least Squares)

Σε αυτή την ενότητα παρουσιάζεται η υλοποίησή μας για τον ο αλγόριθμο Least Squares(LS).

4.1 Συνάρτηση ανανέωσης βαρών

Για τον υπολογισμό των βαρών του ταξινομητή, χρησιμοποιήθηκε η συνάρτηση γραμμικής άλγεβρας του αλγορίθμου LS. Η συνάρτηση είναι η παρακάτω:

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

Το διάνυσμα βαρών του ταξινομητή είναι το $\mathbf{w}$. Το $\mathbf{X}$ είναι η μήτρα διανυσμάτων εκπαίδευσης και το $\mathbf{y}$ είναι η μήτρα των κλάσεων των διανυσμάτων εκπαίδευσης. Σε αντίθεση με τον LMS, στον LS προστίθεται σε κάθε διάνυσμα χαρακτηριστικών ένα επιπλέον μοναδιαίο χαρακτηριστικό και υπολογίζεται ένα επιπλέον βάρος γι' αυτό. Αυτό συμβαίνει διότι ο ταξινομητής που εκπαιδεύει ο LS είναι γραμμικός, δηλαδή η συνάρτηση ταξινόμηση είναι η εξίσωση μιας ευθείας, οι οποίες είναι της μορφής $\mathbf{y = w_1x_1 + w_2x_2 + \ldots + w_{n-1}x_{n-1} + w_n}$. Το επιπλέον βάρος είναι το $\mathbf{w_n}$, το οποίο πολλαπλασιάζεται με τη μονάδα(το μοναδιαίο χαρακτηριστικό).

4.2 Υλοποίηση ταξινομητή

Η συνάρτηση calculate_accuracy_ls εκπαιδεύει έναν ταξινομητή με τη χρήση του αλγόριθμου LS κι έπειτα δοκιμάζει την ακρίβειά του. Όπως και οι συνάρτηση calculate_accuracy_lms, έχει τη δυνατότητα να λειτουργήσει και ως stand-alone συνάρτηση, αλλά και ως μέρος k-fold validation. Λαμβάνει τις ίδιες παραμέτρους με τη συνάρτηση calculate_accuracy_lms και ανακτά και προετοιμάζει τα δεδομένα εκπαίδευσης με παρόμοιο τρόπο.

```
def calculate_accuracy_ls(company, k_fold_val = False, train_set = None, test
_set = None):

    if k_fold_val is False:
        if company is None:
            print ("\nNo company name supplied. Exiting.")
            quit()

        print ('\nUsing the Least Squares algorithm.')

        print('\nCompany: ', company)

        conn = sqlite3.connect('database.sqlite')
```

```
        query = 'SELECT CASE WHEN (home_team_goal -
away_team_goal) > 0 THEN "H" WHEN (home_team_goal -
 away_team_goal) < 0 THEN "A" ELSE "D" END AS result, ' + company + 'H, ' + c
ompany + 'A, ' + company + 'D FROM Match WHERE NOT (B365H IS NULL OR B365A IS
 NULL OR B365D IS NULL OR BWH IS NULL OR BWA IS NULL OR BWD IS NULL OR IWH IS
 NULL OR IWA IS NULL OR IWD IS NULL OR LBH IS NULL OR LBA IS NULL OR LBD IS N
ULL)'
        df = pd.read_sql_query(query, conn)
        conn.close()
        df = fix_outliers(df)
        df_shuffled=df.sample(frac=1).reset_index(drop=True)

        rowcount = math.floor(len(df) * training_set_size)
        train_set = df_shuffled.iloc[:rowcount, :]
        test_set = df_shuffled.iloc[(rowcount + 1 ):,:]

    X = np.array(np.zeros((len(train_set),4)))
    y = np.array(np.zeros((len(train_set),3)))

    for i, row in train_set.iterrows():
        X[i] = [row[1], row[2], row[3], 1]
        if row[0] == 'H':
            y[i] = [1, -1, -1]
        elif row[0] == 'D':
            y[i] = [-1, 1, -1]
        else:
            y[i] = [-1, -1, 1]
```

Παρ' όλα αυτά, υπάρχουν κάποιες διαφοροποιήσεις στη διαδικασία προετοιμασίας δεδομένων, ώστε να εξυπηρετηθούν οι ανάγκες του αλγορίθμου LS. Τα δεδομένα εκπαίδευσης αποθηκεύονται στο array **X**. Πριν την προσθήκη του στο array, προστίθεται σε κάθε δείγμα ένα τέταρτο μοναδιαίο χαρακτηριστικό. Οι κλάσεις των δειγμάτων εκπαίδευσης, αποθηκεύονται κωδικοποιημένες στο array **y.**

Η εκπαίδευσης στον αλγόριθμο LS δε γίνεται τμηματικά με την προσπέλαση κάθε ξεχωριστού δείγματος, αλλά γίνεται αφού έχει ανακτηθεί ολόκληρο το σύνολο εκπαίδευσης, καθώς ο αλγόριθμος LS έχει φτιαχτεί για να χρησιμοποιείται όταν όλα τα δεδομένα εκπαίδευσης είναι γνωστά. Δηλαδή, σε αντίθεση με τον LMS, δε μπορεί να χρησιμοποιηθεί σε ένα περιβάλλον real-time ανάκτησης δεδομένων. Παρακάτω φαίνεται ο υπολογισμός του διανύσματος βαρών.

```
    Xtranspose = X.T
    dotProduct = Xtranspose.dot(X)
```

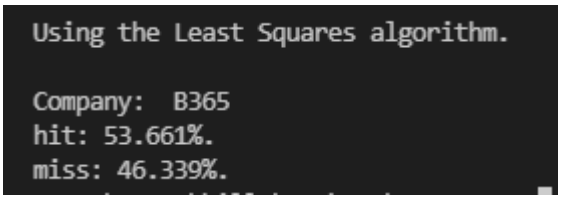
```
    inverse = np.linalg.pinv(dotProduct)
    A = inverse.dot(Xtranspose)
    w = A.dot(y)
```

Το σύνολο εκπαίδευσης πολλαπλασιάζεται με τον ανάστροφο πίνακά του. Από αυτόν τον πολλαπλασιασμό, προκύπτει η μήτρα συσχετισμού των δεδομένων εκπαίδευσης. Η αντίστροφη της μήτρας συσχετισμού αποθηκεύεται στη μεταβλητή **inverse** και πολλαπλασιάζεται με την ανάστροφη μήτρα του συνόλου εκπαίδευσης. Το αποτέλεσμα αποθηκεύεται στη μεταβλητή **A** και πολλαπλασιάζεται με τον πίνακα που περιέχει τις κλάσεις των δειγμάτων εκπαίδευσης για να προκύψει το διάνυσμα βαρών, το οποίο θα περιέχει τέσσερα τρισδιάστατα βάρη.

Η αξιολόγηση του ταξινομητή γίνεται με τον ίδιο τρόπο που γίνεται και στη συνάρτηση **calculate_accuracy_lms,** με τη διαφορά ότι σε κάθε δείγμα αξιολόγησης προστίθεται ένα τέταρτο μοναδιαίο χαρακτηριστικό.

```
    hit = 0
    miss = 0

    for i, row in test_set.iterrows():
        r = np.array([row[1], row[2], row[3], 1]).dot(w)
        c = np.where(r == r.max())[0][0]
        if c == 0 and row[0] == 'H':
            hit += 1
        elif c == 1 and row[0] == 'D':
            hit += 1
        elif c == 2 and row[0] == 'A':
            hit += 1
        else:
            miss += 1

    if k_fold_val is False:
        print("hit: "+str(round((hit / len(test_set) * 100),3))+"%.")
        print("miss: "+str(round( (miss / len(test_set) * 100),3))+"%.")

    return (hit / len(test_set) * 100)
```

Στην παρακάτω εικόνα, φαίνεται ένα παράδειγμα stand-alone εκτέλεσης της συνάρτησης **calculate_accuracy_ls** για την εταιρεία B365. Η συνάρτηση καλείται με την εντολή **calculate_accuracy_ls('B365')**.



5 Πολυστρωματικό Νευρωνικό Δίκτυο

Για την υλοποίηση του πολυστρωματικού νευρωνικού δικτύου, υλοποιήθηκε ένα δίκτυο τύπου Multi-Layer Perceptron(MLP) με τη χρήση της κλάσης MLPClassifier της βιβλιοθήκης scikit-learn. Η συνάρτηση MLP αναλαμβάνει την εκπαίδευση και αξιολόγηση του ταξινομητή με τη χρήση MLP. Η συνάρτηση μπορεί να τρέξει και stand-alone και ως μέρος k-fold validation. Σε αντίθεση με τις συναρτήσεις των άλλων ταξινομητών, δε λαμβάνει την παράμετρο company, καθώς τα δεδομένα εκπαίδευσης και αξιολόγησης περιλαμβάνουν τα προγνωστικά όλων των εταιρειών, καθώς και τα στατιστικά των ομάδων που παίζουν σε κάθε αγώνα. Γι' αυτό το λόγο, χρησιμοποιείται το δεύτερο query που παρουσιάστηκε στη δεύτερη ενότητα. Η ανάκτηση των δεδομένων γίνεται με παρόμοιο τρόπο με τις υπόλοιπες συναρτήσεις.

```
def mlp(k_fold_val = False, train_set = None, test_set = None):
    if k_fold_val is False:
        print('\nRetrieving data from database...\n')
        conn = sqlite3.connect('database.sqlite')

        query = 'SELECT CASE WHEN (home_team_goal -
 away_team_goal) > 0 THEN "H" WHEN (home_team_goal -
 away_team_goal) < 0 THEN "A" ELSE "D" END AS result,  Home_Team.buildUpPlayS
peed, Home_Team.buildUpPlayPassing, Home_Team.chanceCreationPassing, Home_Tea
m.chanceCreationCrossing, Home_Team.chanceCreationShooting, Home_Team.defence
Pressure, Home_Team.defenceAggression, Home_Team.defenceTeamWidth, Away_Team.
buildUpPlaySpeed, Away_Team.buildUpPlayPassing, Away_Team.chanceCreationPassi
ng, Away_Team.chanceCreationCrossing, Away_Team.chanceCreationShooting, Away_
Team.defencePressure, Away_Team.defenceAggression, Away_Team.defenceTeamWidth
, B365H, B365A, B365D, BWH, BWA, BWD, IWH, IWA, IWD, LBH, LBA, LBD FROM Match
 INNER JOIN Team_Attributes Home_Team ON Home_Team.team_api_id = Match.home_t
eam_api_id INNER JOIN Team_Attributes Away_Team ON Away_Team.team_api_id = Ma
tch.away_team_api_id  WHERE NOT (B365H IS NULL OR B365A IS NULL OR B365D IS N
ULL OR BWH IS NULL OR BWA IS NULL OR BWD IS NULL OR IWH IS NULL OR IWA IS NUL
L OR IWD IS NULL OR LBH IS NULL OR LBA IS NULL OR LBD IS NULL)'
        df = pd.read_sql_query(query, conn)
        conn.close()
        df = fix_outliers(df)
        df_shuffled=df.sample(frac=1).reset_index(drop=True)
```

```
    rowcount = math.floor(len(df) * training_set_size)
    train_set = df_shuffled.iloc[:rowcount, :]
    test_set = df_shuffled.iloc[(rowcount + 1 ):,:]
```

Αφού ανακτηθούν τα δεδομένα από τη βάση, τα διανύσματα χαρακτηριστικών εκπαίδευσης μαζεύονται στον πίνακα **X** και οι κλάσεις στον πίνακα **y**. Τα διανύσματα χαρακτηριστικών για την παρούσα συναρτηση περιέχουν 28 χαρακτηριστικά, τα οποία αποτελούνται από τα οκτώ στατιστικά στοιχεία της ομάδας που παίζει εντός έδρας, τα οκτώ στατιστικά στοιχεία της ομάδας που παίζει εκτός έδρας και 12 προγνοστικά για το αποτέλεσμα του αγώνα, τρία για κάθε μία από τις τέσσερις στοιχηματικές εταιρείες.

```
X = np.array(np.zeros((len(train_set),28)))
y = np.array(np.zeros((len(train_set))))

print('Preparing training data...\n')

for i, row in train_set.iterrows():
    for j in range(1, len(row)):
        X[i][j - 1] = row[j]
    if row[0] == 'H':
        y[i] = 1
    elif row[0] == 'D':
        y[i] = 2
    else:
        y[i] = 3
```

Οι κλάσεις των δειγμάτων εκπαίδευσης αποθηκεύονται ως αριθμητικές τιμές, καθώς το δίκτυο MLP μπορεί να διαχειριστεί μόνο δύο μη αριθμητικές κλάσεις.

Αρχικοποιείται ένα αντικείμενο MLPCassifier στη μεταβλητή **clf**. Ορίζοντας την παράμετρο **verbose** ως True, θα εκτυπώνονται μηνύματα που θα μας ενημερώνουν για την πρόοδο της διαδικασίας εκπαίδευσης. Η συνάρτηση ενεργοποίησης που χρησιμοποιείται by default είναι η ReL. Ο ταξινομητής εκπαιδεύεται με την συνάρτηση **fit,** όπως φαίνεται στον παρακάτω κώδικα. Παίρνει ως ορίσματα το σύνολο εκπαίδευσης και τις ταμπέλες εκπαίδευσης. Η συνάρτηση πραγματοποιεί 100 εποχές εκπαίδευσης.

```
print('Training classifier...\n')

clf = MLPClassifier(random_state=1, verbose = True)
clf.out_activation_ = 'softmax'
clf.fit(X, y)
```

Τα δεδομένα αξιολόγησης προετοιμάζονται όπως τα δεδομένα εκπαίδευσης. Τα διανύσματα χαρακτηριστικών αποθηκεύονται στον πίνακα **testX** και οι κλάσεις τους στον πίνακα **testy**. Οι κλάσεις μετατρέπονται και πάλι σε αριθμητικές τιμές.

```
testX = np.array(np.zeros((len(test_set),28)))
testy = np.array(np.zeros((len(test_set))))

print('Preparing test data...\n')

i = 0
for w, row in test_set.iterrows():
    for j in range(1, len(row)):
        testX[i][j - 1] = row[j]
    if row[0] == 'H':
        testy[i] = 1
    elif row[0] == 'D':
        testy[i] = 2
    else:
        testy[i] = 3
    i += 1
```

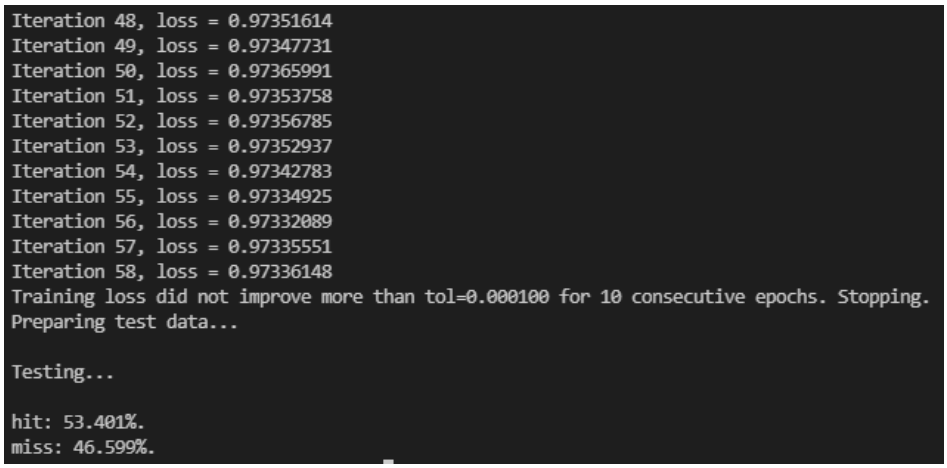
Στον παρακάτω κώδικα, φαίνεται η διαδικασία αξιολόγησης. Κάθε δείγμα από το σύνολο αξιολόγησης ταξινομείται με τη χρήση της συνάρτησης **predict_proba**, η οποία δεν επιστρέφει την τελική κλάση του δείγματος, αλλά το softmax επίπεδο του νευρωνικού δικτύου. Αυτό το επίπεδο περιέχει τρεις τιμές, οι οποίες είναι η πιθανότητα που έχει το δείγμα να ανήκει σε κάθε μία από τις τρεις κλάσεις. Η κλάση του ορίζεται σύμφωνα με το ποια από τις τρις τιμές είναι μεγαλύτερη.

```
print('Testing...\n')

hit = 0
miss = 0

i = 0
for row in testX:
    result = clf.predict_proba([row])
    c = np.where(result[0] == result[0].max())[0][0]
    if c + 1 ==  testy[i]:
        hit += 1
    else:
        miss += 1
    i += 1
```

```
if k_fold_val is False:
    print("hit: "+str(round((hit / len(test_set) * 100),3))+"%.")
    print("miss: "+str(round( (miss / len(test_set) * 100),3))+"%.")

return (hit / len(test_set) * 100)
```

Παρακάτω, φαίνεται ένα παράδειγμα stand-alone εκτέλεσης της συνάρτησης mlp. Η συνάρτηση καλείται με την εντολή **mlp()**.

```
Retrieving data from database...

Preparing training data...

Training classifier...

Iteration 1, loss = 1.11151275
Iteration 2, loss = 1.02308592
Iteration 3, loss = 1.01657306
Iteration 4, loss = 1.01475928
Iteration 5, loss = 1.01504306
Iteration 6, loss = 1.00609401
Iteration 7, loss = 1.00205875
Iteration 8, loss = 0.99558257
Iteration 9, loss = 0.99174729
Iteration 10, loss = 0.98890035
Iteration 11, loss = 0.98478942
Iteration 12, loss = 0.98241338
Iteration 13, loss = 0.98037816
Iteration 14, loss = 0.97883661
Iteration 15, loss = 0.97755182
Iteration 16, loss = 0.97688915
Iteration 17, loss = 0.97686179
Iteration 18, loss = 0.97620173
Iteration 19, loss = 0.97587570
Iteration 20, loss = 0.97565240
Iteration 21, loss = 0.97560568
Iteration 22, loss = 0.97533554
Iteration 23, loss = 0.97532567
Iteration 24, loss = 0.97497959
Iteration 25, loss = 0.97496904
Iteration 26, loss = 0.97470174
Iteration 27, loss = 0.97469549
Iteration 28, loss = 0.97458775
Iteration 29, loss = 0.97434870
Iteration 30, loss = 0.97442597
Iteration 31, loss = 0.97419447
Iteration 32, loss = 0.97404327
Iteration 33, loss = 0.97405605
Iteration 34, loss = 0.97399375
Iteration 35, loss = 0.97398017
Iteration 36, loss = 0.97389887
Iteration 37, loss = 0.97387373
Iteration 38, loss = 0.97378957
Iteration 39, loss = 0.97371536
Iteration 40, loss = 0.97377175
Iteration 41, loss = 0.97384718
Iteration 42, loss = 0.97374587
Iteration 43, loss = 0.97352513
Iteration 44, loss = 0.97357363
Iteration 45, loss = 0.97362873
Iteration 46, loss = 0.97357332
Iteration 47, loss = 0.97342332
```



6 Επιπλέον Συναρτήσεις

Σε αυτή την ενότητα παρουσιάζεται η συνάρτηση fix_outliers, η οποία αντιμετωπίζει τις υπερβολικά μεγάλες και μικρές τιμές στα σύνολα δεδομένων, και η συνάρτηση k_fold_validate, η οποία αναλαμβάνει να τρέχει τους διάφορους αλγόριθμους με k-fold validation.

6.1 Συνάρτηση fix_outliers

Η συνάρτηση **fix_outliers**, κάνει χρήση της στατιστικής μέτρησης Interquartile Range(IQR), με σκοπό την κανονικοποίηση τιμών στο σύνολο οι οποίες εξέχουν των φυσιολογικών ορίων. Ο τρόπος με τον οποίο υλοποιείται αυτή η μέθοδος, είναι απλός: Για κάθε στήλη στο σύνολο, λαμβάνουμε τη μέση τιμή των 25%(Q1) μικρότερων τιμών και των 25% μεγαλύτερων τιμών(Q3). Στη συνέχεια, μπορούμε να υπολογίσουμε το interquartile range αφαιρόντας το Q1 από το Q3. Έχοντας βρει αυτή την τιμή, μπορούμε να θέσουμε ότι η μικρότερη αποδεκτή τιμή στο σύνολό μας αντιστοιχεί σε Q1 μειωμένο κατά 1.5*IQR και αντίστοιχα η μεγαλύτερη αποδεκτή τιμή στο σύνολό μας αντιστοιχεί σε Q3 αυξημένο κατά 1.5*IQR.

```
def fix_outliers(dframe):
    ColumnNames=dframe.columns

    for j in ColumnNames:
        try:
            xy=dframe[j]
            mydata=pd.DataFrame()
            updated=[]
            Q1,Q3=np.percentile(xy,[25,75])
            IQR=Q3-Q1
            minimum=Q1-1.5*IQR
```

```
            maximum=Q3+1.5*IQR
            for i in xy:
                if(i>maximum):
                    i=maximum
                    updated.append(i)
                elif(i<minimum):
                    i=minimum
                    updated.append(i)
                else:
                    updated.append(i)
            dframe[j]=updated
        except:
            continue
    return dframe
```

Με τις μέγιστες και ελάχιστες επιτρεπτές τιμές υπολογισμένες, ελέγχουμε κάθε εγγραφή της εκάστοτε στήλης και σε περίπτωση που η τιμή της εγγραφής ξεπερνάει τα επιτρεπτά όρια, η τιμή μεταβάλλεται ανάλογα, ώστε να συγλίνει προς το όριό της.

6.2 Συνάρτηση k_fold_validate

Η συνάρτηση **k_fold_validate** τρέχει τις διάφορες συναρτήσεις των ταξινομητών με k-fold validation. Στην προκειμένη περίπτωση, πραγματοποιείται 10-fold validation. Λαμβάνει δύο ορίσματα. Το **company** είναι η εταιρεία για την οποία θα πρέπει να τρέξουν οι αγόριθμοι LS και LMS, στην περίπτωση που θα πρέπει να γίνει k-fold validation με αυτούς. Το **method** ορίζει για ποιόν αλγόριθμο θα πρέπει να τρέξει το k-fold validation.

```
def k_fold_validate(company = None, method = 'LMS'):
    if method == 'LMS':
        print ('\nUsing the Least Mean Squares algorithm.')
    elif method == 'LS':
        print ('\nUsing the Least Squares algorithm.')
    else:
        print ('\nUsing Multi-Layer Perceptron.')


    folds=10
    conn = sqlite3.connect('database.sqlite')

    if(method == 'MLP'):
        query = 'SELECT CASE WHEN (home_team_goal -
 away_team_goal) > 0 THEN "H" WHEN (home_team_goal -
 away_team_goal) < 0 THEN "A" ELSE "D" END AS result,  Home_Team.buildUpPlayS
```

```
peed, Home_Team.buildUpPlayPassing, Home_Team.chanceCreationPassing, Home_Tea
m.chanceCreationCrossing, Home_Team.chanceCreationShooting, Home_Team.defence
Pressure, Home_Team.defenceAggression, Home_Team.defenceTeamWidth, Away_Team.
buildUpPlaySpeed, Away_Team.buildUpPlayPassing, Away_Team.chanceCreationPassi
ng, Away_Team.chanceCreationCrossing, Away_Team.chanceCreationShooting, Away_
Team.defencePressure, Away_Team.defenceAggression, Away_Team.defenceTeamWidth
, B365H, B365A, B365D, BWH, BWA, BWD, IWH, IWA, IWD, LBH, LBA, LBD FROM Match
 INNER JOIN Team_Attributes Home_Team ON Home_Team.team_api_id = Match.home_t
eam_api_id INNER JOIN Team_Attributes Away_Team ON Away_Team.team_api_id = Ma
tch.away_team_api_id  WHERE NOT (B365H IS NULL OR B365A IS NULL OR B365D IS N
ULL OR BWH IS NULL OR BWA IS NULL OR BWD IS NULL OR IWH IS NULL OR IWA IS NUL
L OR IWD IS NULL OR LBH IS NULL OR LBA IS NULL OR LBD IS NULL)'
    else:
        query = 'SELECT CASE WHEN (home_team_goal -
 away_team_goal) > 0 THEN "H" WHEN (home_team_goal -
 away_team_goal) < 0 THEN "A" ELSE "D" END AS result, ' + company + 'H, ' + c
ompany + 'A, ' + company + 'D FROM Match WHERE NOT (B365H IS NULL OR B365A IS
 NULL OR B365D IS NULL OR BWH IS NULL OR BWA IS NULL OR BWD IS NULL OR IWH IS
 NULL OR IWA IS NULL OR IWD IS NULL OR LBH IS NULL OR LBA IS NULL OR LBD IS N
ULL)'
    df = pd.read_sql_query(query, conn)
    conn.close()
    df = fix_outliers(df)
    df_shuffled=df.sample(frac=1).reset_index(drop=True)
```

Ανάλογα με το ποιος αλγόριθμος θα χρησιμοποιηθεί, ανακτούνται από τη βάση τα αντίστοιχα δεδομέναμε τη χρήση του κατάλληλου query. Τα δεδομένα αποθηκεύονται σε ένα dataset, φιλτράρονται οι υπερβολικά μεγάλες τιμές και ανακατεύονται ώστε να είναι πιο δυναμικά.

Μετά την ανάκτησή τους, τα δεδομένα χωρίζονται σε υποσύνολα, τόσα όσα και τα folds. Σε αυτή την περίπτωση, τα δεδομένα της βάσης θα χωριστούν σε δέκα κομμάτια. Τα υποσύνολα ανακατεύονται περαιτέρω και αποθηκεύονται στον πίνακα sets.

```
    hit_results=[]

    sets = {}
    key = 0
    step = round(1/folds,2)
    total = round(1/folds,2)
    prev = 0
    while total <= 1:
        rowcount = math.floor(len(df) * total)
        sets[key] = df_shuffled.iloc[prev:rowcount, :]
```

```
        total += step
        prev = rowcount + 1
        key += 1
```

Αφού ολοκληρωθεί η πρετοιμασία των δεδομένων, ξεκινά η διαδικασία του k-fold validation. Για κάθε fold, δημιουργείται ένα dataframe εκπαίδευσης με τις κατάλληλες στήλες για κάθε αλγόριθμο. Το υποσύνολο δεδομένων που αντιστοιχεί στο fold, αφαιρείται από τον πίνακ με τα υποσύνολα. Όλα τα υπόλοιπα δεδομένα περνούν ως δεδομένα εκπαίδευσης στον αλγόριθμο που χρησιμοποιείται, ενώ το αφαιρεμένο υποσύνολο περνά ως τα δεδομένα αξιολόγησης.

```
for dict_key in range(0,folds):
        if method == 'MLP':
            train_set = pd.DataFrame(columns=df.columns)
        else:
            train_set = pd.DataFrame(columns=['result',company+'H',company+'D
',company+'A'])
        leftover_sets = dict(sets)
        leftover_sets.pop(dict_key)
        for frame in leftover_sets.values():
            train_set = train_set.append(frame, ignore_index=True)
        if method == 'MLP':
            result = mlp(True, train_set, sets[dict_key])
        elif method == 'LMS':
            result = calculate_accuracy_lms(company, True, train_set, sets[di
ct_key])
        else:
            result = calculate_accuracy_ls(company, True, train_set, sets[dic
t_key])
        hit_results.append(result)
        print('Result for fold', str(dict_key + 1) + ':', result)
```

Το αποτέλεσμα του fold αποθηκεύεται στη μεταβλητή **result**, προστίθεται στον πίνακα hit_results και εκτυπώνεται.

Η μέση ακρίβεια υπολογίζεται ως το άθροισμα των ποσοστών ακριβείας από τις επαναλήψεις δια τον αριθμό των επαναλήψεων.

```
    if method == 'MLP':
        print ("\n"+str(folds)+"-fold cross-validated accuracy for Multi-
Layer Perceptron:",str(round(sum(hit_results)/len(hit_results),2))+"%.\n")
    else:
```

```
        print ("\n"+str(folds)+"-fold cross-
validated accuracy for "+company+" using "+str(method)+":",str(round(sum(hit_
results)/len(hit_results),2))+"%.\n")
    return(round(sum(hit_results)/len(hit_results),2))
```

Παρακάτω φαίνονται παραδείγματα εκτέλεσης των αλγορίθμων ταξινόμησης με τη χρήση 10-fold validation. Ο LMS και ο LS εκτελούνται για την εταιρεία B365.

LMS:

Εντολή: **k_fold_validate('B365', 'LMS')**

```
Using the Least Mean Squares algorithm.
Result for fold 1: 46.79430097951914
Result for fold 2: 45.36954585930543
Result for fold 3: 45.68121104185218
Result for fold 4: 46.6815144766147
Result for fold 5: 45.59216384683882
Result for fold 6: 46.215494211932324
Result for fold 7: 45.96881959910913
Result for fold 8: 44.25645592163846
Result for fold 9: 46.260017809439006
Result for fold 10: 46.50334075723831

10-fold cross-validated accuracy for B365 using LMS: 45.93%.
```

LS:

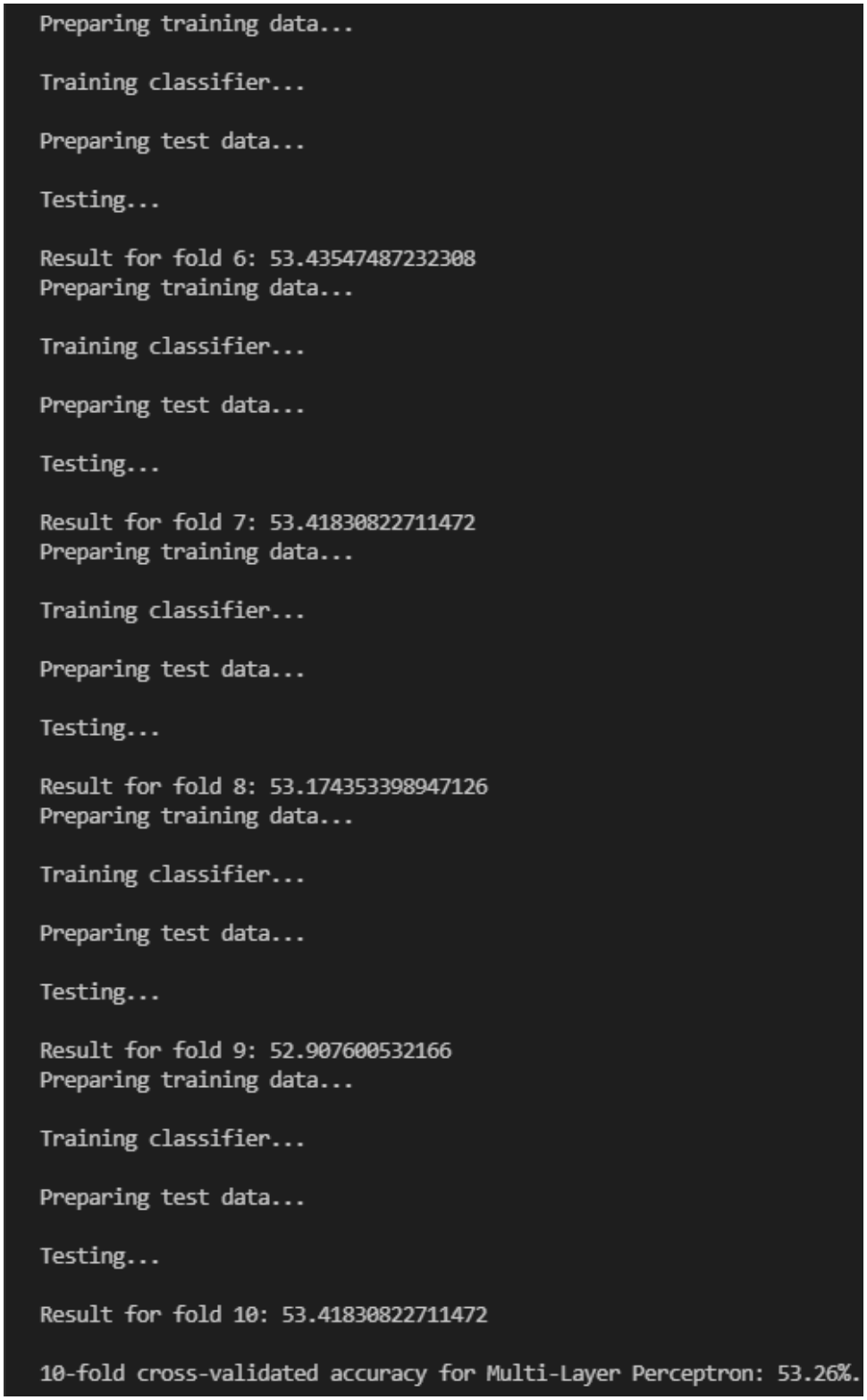
Εντολή: **k_fold_validate('B365', 'LS')**

```
Using the Least Squares algorithm.
Result for fold 1: 49.55476402493321
Result for fold 2: 49.73285841495993
Result for fold 3: 47.01691896705254
Result for fold 4: 49.487750556792875
Result for fold 5: 49.95547640249332
Result for fold 6: 47.46215494211932
Result for fold 7: 48.195991091314035
Result for fold 8: 48.619768477292965
Result for fold 9: 46.52715939447907
Result for fold 10: 48.685968819599104

10-fold cross-validated accuracy for B365 using LS: 48.52%.
```

MLP:

Εντολή: **k_fold_validate(None, 'MLP')**

```
Using Multi-Layer Perceptron.
Preparing training data...

Training classifier...

Preparing test data...

Testing...

Result for fold 1: 53.19151979858091
Preparing training data...

Training classifier...

Preparing test data...

Testing...

Result for fold 2: 53.02776704862452
Preparing training data...

Training classifier...

Preparing test data...

Testing...

Result for fold 3: 53.46475165941863
Preparing training data...

Training classifier...

Preparing test data...

Testing...

Result for fold 4: 53.44405819492726
Preparing training data...

Training classifier...

Preparing test data...

Testing...

Result for fold 5: 53.1342984664683
```


Preparing training data...
Training classifier...
Preparing test data...
Testing...
Result for fold 6: 53.43547487232308
Preparing training data...
Training classifier...
Preparing test data...
Testing...
Result for fold 7: 53.41830822711472
Preparing training data...
Training classifier...
Preparing test data...
Testing...
Result for fold 8: 53.174353398947126
Preparing training data...
Training classifier...
Preparing test data...
Testing...
Result for fold 9: 52.907600532166
Preparing training data...
Training classifier...
Preparing test data...
Testing...
Result for fold 10: 53.41830822711472
10-fold cross-validated accuracy for Multi-Layer Perceptron: 53.26%.

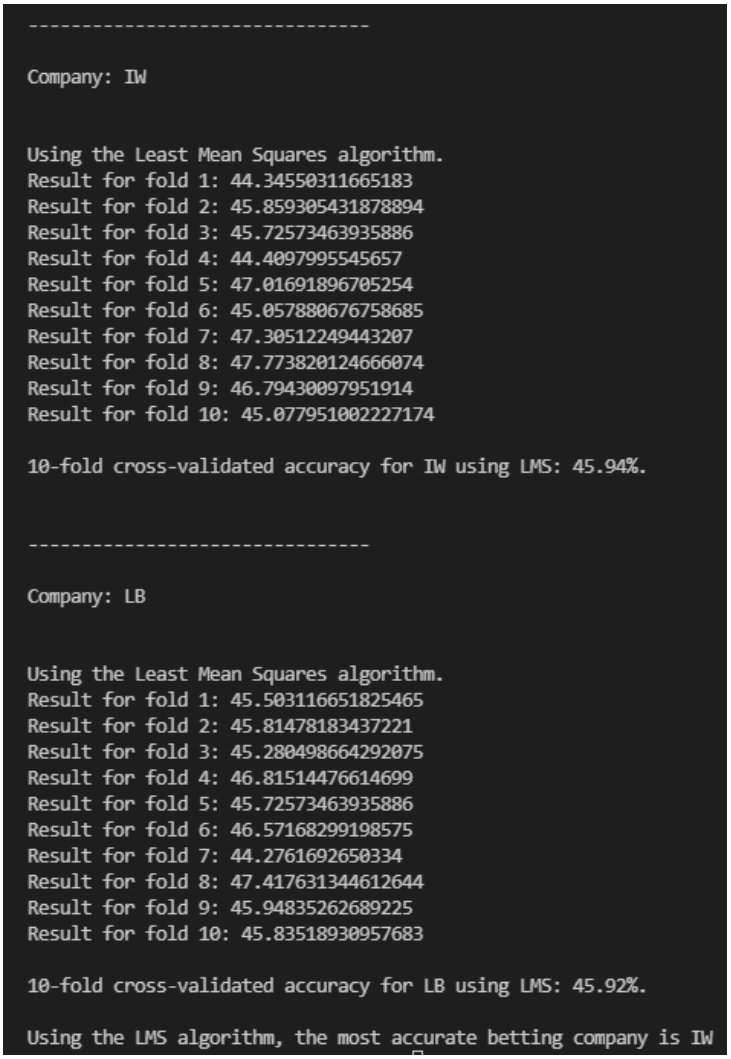
7 Εύρεση ακριβέστερης εταιρείας

Για την εύρεση της εταιρείας τις οποίας τα προγνωστικά είναι τα πιο ακριβή, δημιουργήθηκε η συνάρτηση **most_accurate_betting_company**. Η συνάρτηση λαμβάνει σαν όρισμα τον αλγόριθμο τον οποίο θα χρησιμοποιήσει, τον LMS ή τον LS. Η συνάρτηση τρέχει τον ορισμένο αλγόριθμο για κάθε μία από τις τέσσερις εταιρείες με τη χρήση k-fold validation.

```
def most_accurate_betting_company(algorithm):
    print('\nFinding most accurate betting company using the', algorithm, 'al
gorithm...')
    companies = ['B365', 'BW', 'IW', 'LB']
    results = []
    for company in companies:
        print('\n---------------------------------\n')
        print('Company:', company,'\n')
        results.append(k_fold_validate(company,algorithm))
    print('Using the', algorithm, 'algorithm, the most accurate betting compa
ny is', companies[results.index(max(results))])
```

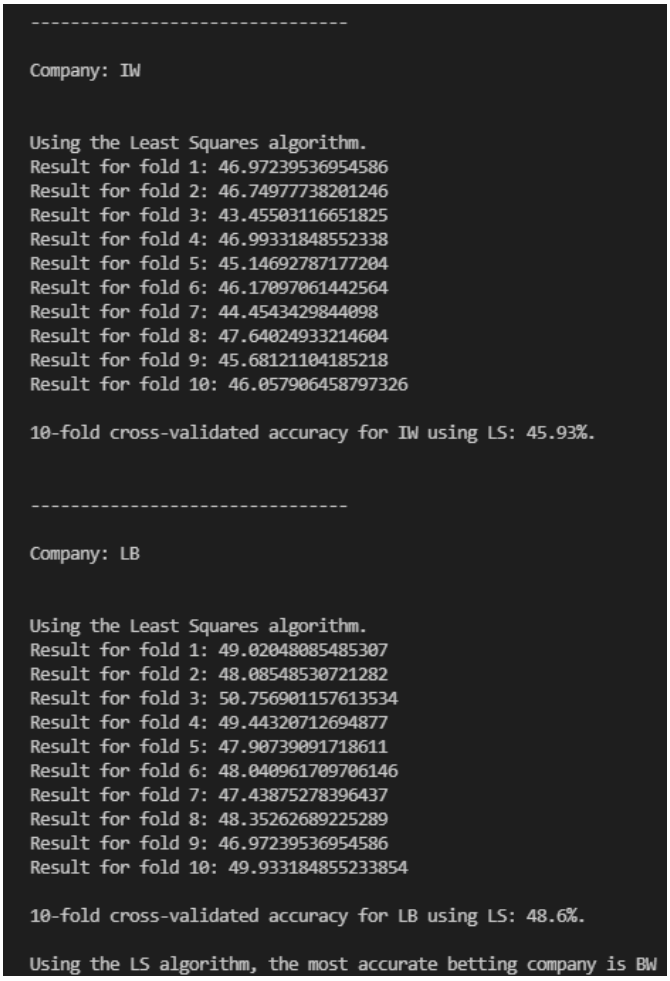
Η συνάρτηση μαζεύει τα ποσοστά ακριβείας στον πίνακα **results** και τυπώνει τη εταιρεία της οποίας το ποσοστό ακριβείας είναι το μεγαλύτερο, καθώς και την ακρίβειά της. Παρακάτω φαίνεται η εκτέλεση της συνάρτησης με τη χρήση του αλγορίθμου LMS. Η συνάρτηση καλείται με την εντολή **most_accurate_betting_company('LMS')**.

```
Finding most accurate betting company using the LMS algorithm...

--------------------------------

Company: B365


Using the Least Mean Squares algorithm.
Result for fold 1: 46.438112199465714
Result for fold 2: 44.83526268922529
Result for fold 3: 45.54764024933215
Result for fold 4: 46.6815144766147
Result for fold 5: 46.126447016918966
Result for fold 6: 45.859305431878894
Result for fold 7: 48.15144766146993
Result for fold 8: 44.701691896705256
Result for fold 9: 45.503116651825465
Result for fold 10: 45.345211581291764

10-fold cross-validated accuracy for B365 using LMS: 45.92%.


--------------------------------

Company: BW


Using the Least Mean Squares algorithm.
Result for fold 1: 28.450578806767584
Result for fold 2: 28.40605520926091
Result for fold 3: 29.741763134461262
Result for fold 4: 28.418708240534524
Result for fold 5: 27.337488869100625
Result for fold 6: 28.584149599287624
Result for fold 7: 29.265033407572382
Result for fold 8: 28.36153161175423
Result for fold 9: 29.207479964381122
Result for fold 10: 30.111358574610247

10-fold cross-validated accuracy for BW using LMS: 28.79%.
```



Παρακάτω φαίνεται η εκτέλεση της συνάρτησης με τη χρήση του αλγορίθμου LS. Η συνάρτηση καλείται με την εντολή **most_accurate_betting_company('LS')**.

```
Finding most accurate betting company using the LS algorithm...

--------------------------------

Company: B365


Using the Least Squares algorithm.
Result for fold 1: 50.08904719501336
Result for fold 2: 49.77738201246661
Result for fold 3: 49.33214603739982
Result for fold 4: 48.41870824053452
Result for fold 5: 46.260017809439006
Result for fold 6: 46.838824577025825
Result for fold 7: 48.8195991091314
Result for fold 8: 47.81834372217275
Result for fold 9: 48.48619768477293
Result for fold 10: 47.97327394209354

10-fold cross-validated accuracy for B365 using LS: 48.38%.


--------------------------------

Company: BW


Using the Least Squares algorithm.
Result for fold 1: 49.73285841495993
Result for fold 2: 47.72929652715939
Result for fold 3: 48.1300089047195
Result for fold 4: 50.33407572383074
Result for fold 5: 50.35618878005342
Result for fold 6: 50.756901157613534
Result for fold 7: 49.79955456570156
Result for fold 8: 51.11308993766697
Result for fold 9: 51.33570792520036
Result for fold 10: 51.937639198218264

10-fold cross-validated accuracy for BW using LS: 50.12%.
```



```
--------------------------------

Company: IW


Using the Least Squares algorithm.
Result for fold 1: 46.97239536954586
Result for fold 2: 46.74977738201246
Result for fold 3: 43.45503116651825
Result for fold 4: 46.99331848552338
Result for fold 5: 45.14692787177204
Result for fold 6: 46.17097061442564
Result for fold 7: 44.4543429844098
Result for fold 8: 47.64024933214604
Result for fold 9: 45.68121104185218
Result for fold 10: 46.057906458797326

10-fold cross-validated accuracy for IW using LS: 45.93%.


--------------------------------

Company: LB


Using the Least Squares algorithm.
Result for fold 1: 49.02048085485307
Result for fold 2: 48.08548530721282
Result for fold 3: 50.756901157613534
Result for fold 4: 49.44320712694877
Result for fold 5: 47.90739091718611
Result for fold 6: 48.040961709706146
Result for fold 7: 47.43875278396437
Result for fold 8: 48.35262689225289
Result for fold 9: 46.97239536954586
Result for fold 10: 49.933184855233854

10-fold cross-validated accuracy for LB using LS: 48.6%.

Using the LS algorithm, the most accurate betting company is BW
```