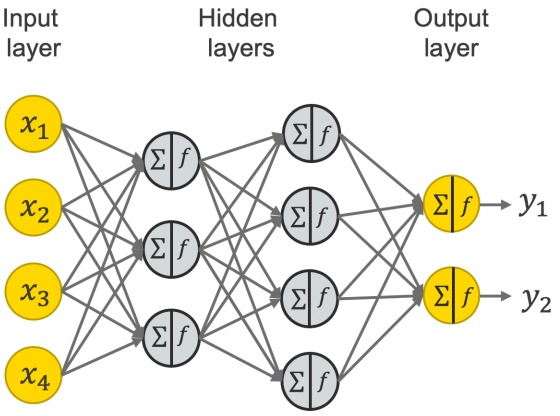
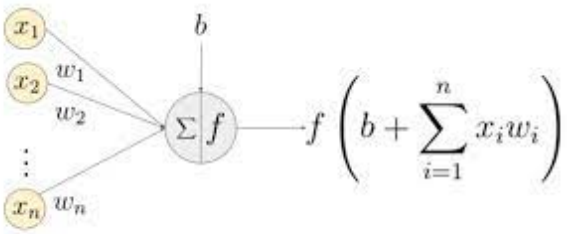
Theory

Τα πολυστρωματικά νευρωνικά δίκτυα είναι αλγόριθμοι μηχανικής μάθησης που μπορούν να χρησιμοποιηθούν για γραμμικά αλλά και για μη γραμμικά μοντέλα και για καταστάσεις παλινδρόμησης(regression task) αλλά και ταξινόμησης(classification task).Βασίζονται στον συνδυασμό πολλών απλών εμπρός-τροφοδοτούμενων νευρώνων τα οποία 'στιβάζονται' σε σειρά και παράλληλα,δημιουργώντας ένα δίκτυο (network of neurons) ,στην έξοδο των οποίων υπάρχει μία συνάρτηση ενεργοποίησης(activation functions),που δημιουργεί τις μη γραμμικότητες του αλγορίθμου. Παρακάτω φαίνεται μία εικόνα ενός τετοιου νευρωνικού δικτύου με μόνο 2 εσωτερικά στρώματα(hidden layer):





An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Τα βελάκια αντιπροσωπεύουν τη ροή της πληροφορίας ,η οποία πολλαπλασιάζεται με ένα βάρος(άγνωστο εξ αρχής) και καταλλήγει στον εκάστοτε νευρώνα του επόμενου στρώματος.Στη συνέχεια υπάρχει το αθροισμα/συνδιασμός των πληροφοριών που συμβολίζεται με το γράμμα 'Σ' ,και ολο αυτό 'περνάει' μέσα από μία συνάρτηση ,την συνάρτηση ενεργοποίησης.Είπαμε ότι στον απλό αλγόριθμο Perceptron ενός νευρώνα ,η τελικλή συνάρτηση ενεργοποίησης ειναι η συνάρτησ **heaviside** που δίνει είτε 1 είτε -1 ως έξοδο.Τετοιες ΜΗ ΓΡΑΜΜΙΚΕΣ συναρτήσεις που χρησιμοπουούνται στην πράξη είναι η συνάρτήσεις 'relu','tanh','logistic'.Αξίζει να σημειωθεί πως μια συνάρτηση της μορφής f(x)=x,το μοντέλο είναι γραμμικό και η είσοδο του επόμενου νευρώνα είναι η έξοδος του προηγούμενου,δίκτυο το οποίο μπορεί να εκφυλιστεί σε ένα μόνο νευρώνα.

Μόλις η πληροφορία ,επεξεργασμένη φτάσει στον/στους τελευταίο κόμβο πρέπει με κάποιον τρόπο να μετρηθεί σε σχέση με την πραγματική έξοδο που αναμένουμε.Εδώ εισάγεται η έννοια της συνάρτησεις κόστους ή συνάρτηση απώλειας(cost function , loss function).Με βάση αυτό μετράει τη διαφορά της πραγματικής από την έξοδο του δικτύου μας.Αυτή η συνάρτηση θα χρησιμοποιηθεί για να γίνει ,αναβάθμισει των βαρών του δικτύου ,έτσι το δίκτυο αρχίζει να εκπαιδεύεται.Ένας από τους συνήθης τρόπους που γίνεται αυτό είναι μέσω της παραγώγου της συνάρτησης απώλειας(loss function gradient) και χρήση του αναπτύγματος Taylor των βαρών όπου και συνδυάζονται στην τελική σχέση,του αλγόριθμου Gradient Descent:

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

οπου, **α:learning rate,L:loss function, wt+1=updated weight at next iteration,wt=known weight previous iteration**

Συνήθως για πολλά δεδομένα χρησιμοποιείται ο λεγόμενος αλγόριθμος **stochastic gradient descent** οπου τα βάρη ανα ανάβαθμιζονται σταδιακά και τα δεδομένα χωρίζονται σε υπολσύνολα οπου αυτο συμβαίνει ,ταλεγόμενα mini batches.Οπως γινεται αντιληπτό από το όνομα του είναι στοχαστικός (και όχι ντετερμινιστικός) αλγόριθμος βελτιστοποίησης. **Γενικά η εκπαίδευση των νευρωνικών δικτύων είναι ένα πρόβλημα βελτιστοποίησης όπου θέλουμε ελάχιστη συνάρτηση κόστους ως προς τα βάρη των νευρώνων**

Όπως γίνεται εύκολα καταννοητό το νευρωνικό δίκτυο μπορεί να έχει πολυδιάστατες εισόδους και εξόδους οπότε εκ φύσεως μπορεί να λύνει προβλήματα με πολλές εισόδους και να τα μεταφράζει σε πολλές εξόδους,οπως για παράδειγμα το δικό μας πρόβλημα με 3 κλάσεις εξόδου.

Γενικά ,για γραμμικά προβλήματα υπάρχουν άλλοι αλγοριθμοι απλοί,που δίνουν γρήγορα και πολύ καλά αποτελέσματα. Εκεί που φαίνεται η πραγματική τους αξία των νευρωνικών δικτύων ειναι σε προβλήματα πολύπλοκα που οι συμβατικοί αλγοριθμοι αστοχούν.Σε συνδιασμό μαζί με άλλες αρχιτεκτονικές βοηθούν σε τομείς όπως η ρομποτική όραση,αναγνωριση εικόνας,ανάλυση ήχου κ.α.Μαζί με την μεγάλη εξέλιξη των υπολογιστών ,σήμερα τα πολύπλοκα αυτά μοντέλα είναι σε θέση να εκπαιδευτούν και να δώσουν πολύ καλα αποτελέσματα,ακόμα και σε έναν προσωπικό υπολογιστή μπορεί να γίνει επίδειξη των ικανοτήτων τους.Φυσικά, για πολύ μεγάλα δεδομένα χρειάζονται υπολογιστές ισχυρότεροι τοσο για την αποθηκευση των δεδομένω οσο και για την εκπαιδευση των νευρώνων ενός τέτοιου δικτύου.

Αρνητικά των νευρωνικών δικτύων είναι ότι έχουν πολλές hyperparameters που πρέπει να επιλεγούν,μπορεί να κάνουν overfitting στα δεδομένα,μπορεί να παρουν ώρα και για εκπαίδευση αλλά και για συντονισμό των υπερπαραμέτρων,δεν ειναι εύκολα επεξηγήσιμα(interpretable).

Στο δικό μας πρόβλημα,έχουμε 28 εισόδους δηλαδη 28 χαρακτηριστικά,τα bet-odds της κάθε στοιχηματικής και τα χαρακτηριστικά των ομάδων του κάθε αγώνα.Η διαδικασία που ακολουθείται παρουσιάζεται συνοπτικά και στη συνέχεια φαίνεται αναλυτικά με τον κατάλληλο κώδικα:

- Διαβάζονται τα δεδομένα από την sqlite database
- Χωρίζονται σε match και Attributes dataframes,και μέσα από αυτά κρατώνται οι στήλες που χρειάζονται(και εχουν δοθεί από την εκφώνηση).Προφανώς η κάθε στήλη αποτελεί ένα Χαρακτηριστικό εισόδου δλδ για N στήλες(χαρακτηριστικά) και M γραμμές το διάνυσμα X εχει διαστάσεις M x N
- Επεξεργαζόμαστε αυτά τα δεδομένα ,δηλαδη αφαιρούμε τις γραμμές με NaN values.Σημειώνεται ότι αυτό θα μπορούσε να αποφευχθεί ώστε να μην χαθούν δεδομένα,και οι NaN values να αντικατασταθουν από την μέση ή την ενδιάμεση τιμη της στήλης (mean or median).Επίσης θα μπορούσε να γινει και περεταίρω επεξεργασία όπως η αφαίρεση των outliers,οπως πχ αποδόσεις που προκύπτουν εξαιρετικά σπάνια
- Προκειμένου να συνδυάσουμε τα 2 dataframes σε 1 ,πρώτα βλέπουμε ποια match λείπουν από τα Attributes με βάση το match_api_id(match)και τα home_api_id και away_api_id (Attributes) ,Όσα δεν υπάρχουν, προσθέτουμε το id στα Attributes και γεμίζουμε τα κελιά με την median τιμή της κάθε στήλης.Επίσης για τις ομάδες που εμφανίζονται αρκετές φορές διαφορετικές χρονιές ,αντικαθιστούμε όλες τις τιμές στους αγώνες με την μεση τιμή όλων των χρόνων(mean).Οι ομάδες είναι λογικό να μην έχουν αποδόσεις σε χαρακτηριστικά πολύ διαφορετικά μέσα σε λίγα χρόνια.
- Με βάση τα παραπάνω 3 id στα 2 dataframe ,τα συνδυάζουμε και παίρνουμε το τελικό 'final_merged' με τις απαραίτητες στήλες
- Πριν την εκπαίδευση του αλγόριθμου χωρίζουμε τα δεδομένα σε train και test set,μια στανταρ διαδικασία σε προβλήματα μηχανικής μάθησης.Μετά επίσης χρησιμοποιούμε ένας StandarScaler και τα κανονικοποιούμε.(κανονική κατανομή)
- Φτιάχνουμε τον αλγοριθμο MLPClassifier,ορίζοντας τα απαραίτητα arguments οπως τον λύτη για την βελτιστοποίηση των βαρών,τον ρυθμό μάθησης(learning rate),τον μεγεθος των minibatches κ.α.
- Κάνουμε fit τον αλγόριθμο στα δεδομένα εκπαιδευσης(training data),πλοτάρουμε την loss function
- Τελος κανουμε evaluate κυρίως με χρηση της πολλαπλής διεπικύρωσης(k fold cross validation) με 10 υποομάδες,και βλεπουμε πόσο διαφέρει ο αλγόριθμος στα δεδομένα του test set.

Σημειώνουμε πως τα μοντέλα πρέπει να διερευνηθούν περισσότερο είτε με διάφορους συνδυασμούς χαρακτηριστικών(feature engineering) όπως και ψάξιμο των υπερ-παραμέτρων όπως τα ενδιάμεσα κρυμένα στρωματα(hidden layers),ο ρυθμός μάθησης κ.α ώστε να πετύχουμε καλύτερα cross-validation scores.Γενικα το ποσοστό 46% της κλασσης νίκης HOME(δλδ ένας αλγοριθμος που θα προβλέπει μόνο κλάση 2 θα έχει επιτυχία ποσοστού εμφάνισης των κλασσεων όπως εδω το HOME WIN)

Χρησιμοποιήθηκαν τα μοντέλα από την ανοιχτή κοινότητα και συγκεκριμένα για τα μοντέλα των νευρωνικών οι βιβλιοθήκες του sklearn σε python,,περισσότερες

πληροφορίες στο https://scikit-learn.org

Practise

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

In [2]:
```
import sqlite3
import pandas as pd
import os
```

In [3]:
```
file_name="database.sqlite"

stream= os.popen('pwd') #put path of file_name
working_dir_path=stream.read().split('\n')[0] #read the command pwd from
absolute_path=working_dir_path + "/" + file_name
print(absolute_path)



#connection to database

connection=sqlite3.connect(absolute_path)

match =pd.read_sql_query("SELECT * FROM Match", connection)

Team_Attributes=pd.read_sql_query("SELECT * FROM Team_Attributes", connec


match.head()
print("-----")
```

```
/home/nickthegreek/Desktop/match_classification_newest/database.sqlite
-----
```

In [4]:
```
match.head()
```

Out[4]:

	id	country_id	league_id	season	stage	date	match_api_id	home_team_api_id
0	1	1	1	2008/2009	1	2008-08-17 00:00:00	492473	9987
1	2	1	1	2008/2009	1	2008-08-16 00:00:00	492474	10000
2	3	1	1	2008/2009	1	2008-08-16 00:00:00	492475	9984
3	4	1	1	2008/2009	1	2008-08-17 00:00:00	492476	9991
4	5	1	1	2008/2009	1	2008-08-16 00:00:00	492477	7947

5 rows × 115 columns

In [5]:
```
Team_Attributes.head()
```

Out[5]:

	id	team_fifa_api_id	team_api_id	date	buildUpPlaySpeed	buildUpPlaySpeedClass	b
0	1	434	9930	2010-02-22 00:00:00	60	Balanced	
1	2	434	9930	2014-09-19 00:00:00	52	Balanced	
2	3	434	9930	2015-09-10 00:00:00	47	Balanced	
3	4	77	8485	2010-02-22 00:00:00	70	Fast	
4	5	77	8485	2011-02-22 00:00:00	47	Balanced	

5 rows × 25 columns

In [6]:
```
match_cols=['date','home_team_api_id','away_team_api_id','B365H', 'B365D'
match_2=match[match_cols]
match_2.head()
```

Out[6]:

	date	home_team_api_id	away_team_api_id	B365H	B365D	B365A	BWH	BWD	BV
0	2008-08-17 00:00:00	9987	9993	1.73	3.40	5.00	1.75	3.35	4.
1	2008-08-16 00:00:00	10000	9994	1.95	3.20	3.60	1.80	3.30	3.
2	2008-08-16 00:00:00	9984	8635	2.38	3.30	2.75	2.40	3.30	2.
3	2008-08-17 00:00:00	9991	9998	1.44	3.75	7.50	1.40	4.00	6.
4	2008-08-16 00:00:00	7947	9985	5.00	3.50	1.65	5.00	3.50	1.

In [7]:
```
useful_cols=['date','team_api_id','buildUpPlaySpeed', 'buildUpPlayPassing
             , 'defencePressure', 'defenceAggression','defenceTeamWidth']
main_df=Team_Attributes[useful_cols]

#main_df.head()
main_df.info()
main_df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1458 entries, 0 to 1457
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   date                    1458 non-null   object
 1   team_api_id             1458 non-null   int64
 2   buildUpPlaySpeed        1458 non-null   int64
 3   buildUpPlayPassing      1458 non-null   int64
 4   chanceCreationPassing   1458 non-null   int64
 5   chanceCreationCrossing  1458 non-null   int64
 6   chanceCreationShooting  1458 non-null   int64
 7   defencePressure         1458 non-null   int64
 8   defenceAggression       1458 non-null   int64
 9   defenceTeamWidth        1458 non-null   int64
dtypes: int64(9), object(1)
memory usage: 114.0+ KB
```

Out[7]:

	team_api_id	buildUpPlaySpeed	buildUpPlayPassing	chanceCreationPassing	chance
count	1458.000000	1458.000000	1458.000000	1458.000000	
mean	9995.727023	52.462277	48.490398	52.165295	
std	13264.869900	11.545869	10.896101	10.360793	
min	1601.000000	20.000000	20.000000	21.000000	
25%	8457.750000	45.000000	40.000000	46.000000	
50%	8674.000000	52.000000	50.000000	52.000000	
75%	9904.000000	62.000000	55.000000	59.000000	
max	274581.000000	80.000000	80.000000	80.000000	

In [8]:
```
match.tail()
```

Out[8]:

	id	country_id	league_id	season	stage	date	match_api_id	home_team
25974	25975	24558	24558	2015/2016	9	2015-09-22 00:00:00	1992091	
25975	25976	24558	24558	2015/2016	9	2015-09-23 00:00:00	1992092	
25976	25977	24558	24558	2015/2016	9	2015-09-23 00:00:00	1992093	
25977	25978	24558	24558	2015/2016	9	2015-09-22 00:00:00	1992094	
25978	25979	24558	24558	2015/2016	9	2015-09-23 00:00:00	1992095	

5 rows × 115 columns

In [9]:
```
s1=set(match['home_team_api_id'].unique())

s3=set(match['away_team_api_id'].unique())
```

In [10]:
```
s2=set(Team_Attributes['team_api_id'].unique())
```

In [11]:
```
len(s2),len(s1),len(s3)
```

Out[11]: (288, 299, 299)

In [12]:
```
s1.difference(s2),s1.difference(s3)
```

Out[12]:
```
({4049, 4064, 6367, 6601, 7896, 7947, 7992, 9765, 10213, 177361, 188163},
 set())
```

For some teams we dont have the team_api_id so we must remove them or impute them!We will choose the second!

In [13]:
```
median_attributes=main_df.describe().loc[['50%']]
median_attributes
```

Out[13]:

	team_api_id	buildUpPlaySpeed	buildUpPlayPassing	chanceCreationPassing	chanceCre
50%	8674.0	52.0	50.0	52.0	

In [14]:
```
table=[]
for idx in set(main_df['team_api_id']):
    #print(main_df[main_df['team_api_id']==idx].mean())
    table.append(main_df[main_df['team_api_id']==idx].mean())
```

```
/tmp/ipykernel_12209/2818235614.py:4: FutureWarning: Dropping of nuisance
columns in DataFrame reductions (with 'numeric_only=None') is deprecated;
in a future version this will raise TypeError.  Select only valid columns
before calling the reduction.
  table.append(main_df[main_df['team_api_id']==idx].mean())
```

In [15]:
```
table=pd.DataFrame(table)
table.tail()
```

Out[15]:

	team_api_id	buildUpPlaySpeed	buildUpPlayPassing	chanceCreationPassing	chanceCrea
283	10233.0	62.833333	51.500000	61.333333	
284	10228.0	50.833333	45.000000	53.833333	
285	10235.0	43.833333	44.833333	48.500000	
286	10238.0	51.000000	41.333333	54.333333	
287	8191.0	51.666667	59.500000	62.500000	

In [16]:
```
table.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 288 entries, 0 to 287
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   team_api_id            288 non-null    float64
 1   buildUpPlaySpeed       288 non-null    float64
 2   buildUpPlayPassing     288 non-null    float64
 3   chanceCreationPassing  288 non-null    float64
 4   chanceCreationCrossing 288 non-null    float64
 5   chanceCreationShooting 288 non-null    float64
 6   defencePressure        288 non-null    float64
 7   defenceAggression      288 non-null    float64
 8   defenceTeamWidth       288 non-null    float64
dtypes: float64(9)
memory usage: 20.4 KB
```

In [17]:
```
table['team_api_id']=table['team_api_id'].astype(int)
```

In [18]:
```
for id_not_in_table in s1.difference(s2):
    print(id_not_in_table)
    median_attributes['team_api_id']=id_not_in_table
    #print(pd.concat([table,median_attributes]).iloc[-1])
    table=pd.concat([table,median_attributes])
```

```
4064
188163
9765
10213
6601
7947
177361
7992
4049
7896
6367
```

In [19]:
```
table.tail(15)
```

Out[19]:

	team_api_id	buildUpPlaySpeed	buildUpPlayPassing	chanceCreationPassing	chanceCre
284	10228	50.833333	45.000000	53.833333	
285	10235	43.833333	44.833333	48.500000	
286	10238	51.000000	41.333333	54.333333	
287	8191	51.666667	59.500000	62.500000	
50%	4064	52.000000	50.000000	52.000000	
50%	188163	52.000000	50.000000	52.000000	
50%	9765	52.000000	50.000000	52.000000	
50%	10213	52.000000	50.000000	52.000000	
50%	6601	52.000000	50.000000	52.000000	
50%	7947	52.000000	50.000000	52.000000	
50%	177361	52.000000	50.000000	52.000000	
50%	7992	52.000000	50.000000	52.000000	
50%	4049	52.000000	50.000000	52.000000	
50%	7896	52.000000	50.000000	52.000000	
50%	6367	52.000000	50.000000	52.000000	

In [20]:
```
table=table.reset_index()
```

In [21]:
```
table=table.drop('index',axis=1)
```

In [22]:
```
m1=pd.merge(left=match_2,right=table,how='inner',left_on=['home_team_api_
m2=pd.merge(left=match_2[['date', 'home_team_api_id', 'away_team_api_id']]
```

In [23]:
```
m2.columns=['date', 'home_team_api_id', 'away_team_api_id', 'team_api_id'
       'buildUpPlaySpeed_away', 'buildUpPlayPassing_away', 'chanceCreatio
       'chanceCreationCrossing_away', 'chanceCreationShooting_away', 'def
       'defenceAggression_away', 'defenceTeamWidth_away']
```

In [24]:
```
m1.head(2)
```

Out[24]:

	date	home_team_api_id	away_team_api_id	B365H	B365D	B365A	BWH	BWD	BV
0	2008-08-17 00:00:00	9987	9993	1.73	3.40	5.0	1.75	3.35	4
1	2008-11-15 00:00:00	9987	9999	1.25	5.25	10.0	1.23	5.00	1(

2 rows × 24 columns

In [25]:
```
m2.head(2)
```

Out[25]:

	date	home_team_api_id	away_team_api_id	team_api_id	buildUpPlaySpeed_away	b
0	2008-08-17 00:00:00	9987	9993	9993	46.0	
1	2008-11-15 00:00:00	10000	9993	9993	46.0	

In [26]:
```
final_merged=pd.concat([m1,m2],axis=1)
final_merged.head(2)
```

Out[26]:

	date	home_team_api_id	away_team_api_id	B365H	B365D	B365A	BWH	BWD	BV
0	2008-08-17 00:00:00	9987	9993	1.73	3.40	5.0	1.75	3.35	4
1	2008-11-15 00:00:00	9987	9999	1.25	5.25	10.0	1.23	5.00	1(

2 rows × 36 columns

In [27]:
```
final_merged=final_merged.loc[:,~final_merged.columns.duplicated(keep='fi
```

In [28]:
```
final_merged.columns
```

Out[28]:
```
Index(['date', 'home_team_api_id', 'away_team_api_id', 'B365H', 'B365D',
       'B365A', 'BWH', 'BWD', 'BWA', 'IWH', 'IWD', 'IWA', 'LBH', 'LBD', '
LBA',
       'team_api_id', 'buildUpPlaySpeed', 'buildUpPlayPassing',
       'chanceCreationPassing', 'chanceCreationCrossing',
       'chanceCreationShooting', 'defencePressure', 'defenceAggression',
       'defenceTeamWidth', 'buildUpPlaySpeed_away', 'buildUpPlayPassing_a
way',
       'chanceCreationPassing_away', 'chanceCreationCrossing_away',
       'chanceCreationShooting_away', 'defencePressure_away',
       'defenceAggression_away', 'defenceTeamWidth_away'],
      dtype='object')
```

columns of 'final_merged'are 32 = 28 + 4['date', 'home_team_api_id', 'away_team_api_id','team_api_id']

In [29]:
```
final_merged.head()
```

Out[29]:

	date	home_team_api_id	away_team_api_id	B365H	B365D	B365A	BWH	BWD	BV
0	2008-08-17 00:00:00	9987	9993	1.73	3.40	5.0	1.75	3.35	4
1	2008-11-15 00:00:00	9987	9999	1.25	5.25	10.0	1.23	5.00	10
2	2008-11-29 00:00:00	9987	9984	1.73	3.40	4.5	1.65	3.45	4
3	2008-12-13 00:00:00	9987	9986	1.53	4.00	6.0	1.55	3.55	5
4	2009-01-24 00:00:00	9987	9998	1.44	4.00	6.5	1.40	3.85	7

5 rows × 32 columns

In [30]:
```
final_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25979 entries, 0 to 25978
Data columns (total 32 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   date                         25979 non-null  object 
 1   home_team_api_id             25979 non-null  int64  
 2   away_team_api_id             25979 non-null  int64  
 3   B365H                        22592 non-null  float64
 4   B365D                        22592 non-null  float64
 5   B365A                        22592 non-null  float64
 6   BWH                          22575 non-null  float64
 7   BWD                          22575 non-null  float64
 8   BWA                          22575 non-null  float64
 9   IWH                          22520 non-null  float64
 10  IWD                          22520 non-null  float64
 11  IWA                          22520 non-null  float64
 12  LBH                          22556 non-null  float64
 13  LBD                          22556 non-null  float64
 14  LBA                          22556 non-null  float64
 15  team_api_id                  25979 non-null  int64  
 16  buildUpPlaySpeed             25979 non-null  float64
 17  buildUpPlayPassing           25979 non-null  float64
 18  chanceCreationPassing        25979 non-null  float64
 19  chanceCreationCrossing       25979 non-null  float64
 20  chanceCreationShooting       25979 non-null  float64
 21  defencePressure              25979 non-null  float64
 22  defenceAggression            25979 non-null  float64
 23  defenceTeamWidth             25979 non-null  float64
 24  buildUpPlaySpeed_away        25979 non-null  float64
 25  buildUpPlayPassing_away      25979 non-null  float64
 26  chanceCreationPassing_away   25979 non-null  float64
 27  chanceCreationCrossing_away  25979 non-null  float64
 28  chanceCreationShooting_away  25979 non-null  float64
 29  defencePressure_away         25979 non-null  float64
 30  defenceAggression_away       25979 non-null  float64
 31  defenceTeamWidth_away        25979 non-null  float64
dtypes: float64(28), int64(3), object(1)
memory usage: 6.5+ MB
```

we can see from above,that there are some null values

In [31]:
```
#final_merged.to_csv('final_merged_table_features+odds.csv')
```

fix dataset before the model

In [32]:
```
df_result=pd.read_csv('./results.csv')
```

In [33]:
```
match['Result']=pd.Series(['D' for i in range(len(match['home_team_goal']
```

In [34]:
```
match['Result'][match['home_team_goal']-match['away_team_goal']>0]='W'
match['Result'][match['home_team_goal']-match['away_team_goal']<0]='L'
#match['Result'][match['home_team_goal']-match['away_team_goal']==0]='D'
```

```
/tmp/ipykernel_12209/2977273242.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  match['Result'][match['home_team_goal']-match['away_team_goal']>0]='W'
/tmp/ipykernel_12209/2977273242.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  match['Result'][match['home_team_goal']-match['away_team_goal']<0]='L'
```

In [35]:
```
match['Result'].head()
```

Out[35]:
```
0    D
1    D
2    L
3    W
4    L
Name: Result, dtype: object
```

In [36]:
```
final_merged=pd.concat([final_merged,match['Result']],axis=1)
```

In [37]:
```
final_merged.head()
```

Out[37]:

	date	home_team_api_id	away_team_api_id	B365H	B365D	B365A	BWH	BWD	BV
0	2008-08-17 00:00:00	9987	9993	1.73	3.40	5.0	1.75	3.35	4
1	2008-11-15 00:00:00	9987	9999	1.25	5.25	10.0	1.23	5.00	10
2	2008-11-29 00:00:00	9987	9984	1.73	3.40	4.5	1.65	3.45	4
3	2008-12-13 00:00:00	9987	9986	1.53	4.00	6.0	1.55	3.55	5
4	2009-01-24 00:00:00	9987	9998	1.44	4.00	6.5	1.40	3.85	7

5 rows × 33 columns

In [38]:
```
#final_merged.to_csv('final_merged_table_features+odds+result.csv')
```

In [39]:
```
print('ratio of dataset rows after droping nan values:')
len(final_merged.dropna())/len(final_merged)
```

Out[39]:
```
ratio of dataset rows after droping nan values:
0.8648138881404211
```

In [40]:
```
final_merged_dropna=final_merged.dropna()
final_merged_dropna.head()
```

Out[40]:

	date	home_team_api_id	away_team_api_id	B365H	B365D	B365A	BWH	BWD	BV
0	2008-08-17 00:00:00	9987	9993	1.73	3.40	5.0	1.75	3.35	4
1	2008-11-15 00:00:00	9987	9999	1.25	5.25	10.0	1.23	5.00	10
2	2008-11-29 00:00:00	9987	9984	1.73	3.40	4.5	1.65	3.45	4
3	2008-12-13 00:00:00	9987	9986	1.53	4.00	6.0	1.55	3.55	5
4	2009-01-24 00:00:00	9987	9998	1.44	4.00	6.5	1.40	3.85	7

5 rows × 33 columns

MODEL

In [41]:
```
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPClassifier
```

In [42]:
```
y=final_merged_dropna['Result']
```

In [43]:
```
columns_not_x=['date', 'home_team_api_id', 'away_team_api_id','team_api_i
final_merged_dropna.drop(columns_not_x,axis=1)
```

Out[43]:

	B365H	B365D	B365A	BWH	BWD	BWA	IWH	IWD	IWA	LBH	...	defenceAggress
0	1.73	3.40	5.00	1.75	3.35	4.20	1.85	3.2	3.5	1.80	...	50.666
1	1.25	5.25	10.00	1.23	5.00	10.00	1.30	4.2	8.0	1.25	...	50.666
2	1.73	3.40	4.50	1.65	3.45	4.90	1.65	3.4	4.2	1.72	...	50.666
3	1.53	4.00	6.00	1.55	3.55	5.65	1.55	3.5	4.8	1.50	...	50.666
4	1.44	4.00	6.50	1.40	3.85	7.10	1.45	3.8	5.4	1.40	...	50.666
...	...	...	...	...	...	...	...	...	...	...	...	
24552	4.00	3.60	1.91	4.00	3.50	1.87	3.60	3.3	2.0	3.80	...	42.833
24553	1.91	3.50	4.00	1.90	3.50	3.90	2.00	3.3	3.6	1.91	...	42.833
24554	3.30	3.40	2.20	3.40	3.30	2.20	2.90	3.3	2.3	3.30	...	42.833
24555	2.10	3.30	3.75	2.10	3.00	3.80	2.10	3.3	3.3	2.00	...	42.833
24556	3.50	3.25	2.20	3.25	3.25	2.20	3.30	3.3	2.1	3.50	...	42.833

22467 rows × 28 columns

In [44]:
```
X=final_merged_dropna.drop(columns_not_x,axis=1)
```

In [45]:
```
encoder=LabelEncoder()
scaler=StandardScaler()
```

In [46]:
```
encoder.fit(y)
y_encoded=encoder.transform(y)
y_encoded[:8]
```

Out[46]:
```
array([0, 0, 1, 2, 1, 0, 0, 1])
```

In [47]:
```
X_scaled=scaler.fit_transform(X)
```

In [48]:
```
#sns.kdeplot(x=X_scaled[:,0]),sns.kdeplot(x=X_scaled[:,1]),sns.kdeplot(x=
#plt.title('after standarization')
```

split data to train,test

In [49]:
```
X_train, X_test, y_train, y_test=train_test_split(X_scaled,y_encoded,
                                                  test_size=0.1,
                                                  train_size=None,
                                                  random_state=1,
                                                  shuffle=True,
                                                  stratify=y_encode
```

In [50]:
```
(y_train==0).sum(),(y_train==1).sum(),(y_train==2).sum()
```

Out[50]:
```
(5110, 5820, 9290)
```

In [51]:
```
(y_train==0).sum(),(y_train==1).sum(),(y_train==2).sum()
```

Out[51]:
```
(5110, 5820, 9290)
```

In [52]:
```
#max_it=200
#max_it=400
max_it=800
```

In [53]:
```
MLP=MLPClassifier(hidden_layer_sizes=(50,10),activation='relu',solver='ad
                  max_iter=max_it,
                  shuffle=True,
                  random_state=1)
```

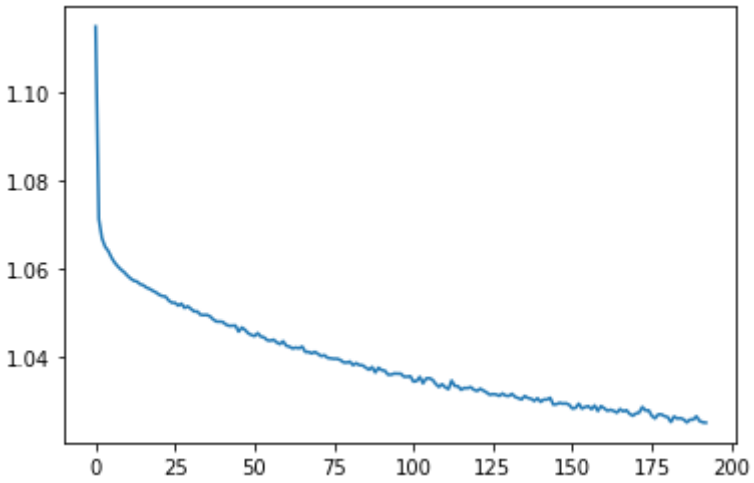
In [54]:
```
%%time
MLP.fit(X_train,y_train)
```

```
CPU times: user 1min 44s, sys: 57.7 s, total: 2min 42s
Wall time: 1min 26s
```

Out[54]:
```
MLPClassifier(hidden_layer_sizes=(50, 10), max_iter=800, random_state=1)
```

In [55]:
```
plt.plot(MLP.loss_curve_)
```

Out[55]:
```
[<matplotlib.lines.Line2D at 0x7f4579790520>]
```



In [ ]:
```
```

In [56]:
```
train_pred=MLP.predict(X_train)
```

In [57]:
```
MLP.score(X_train,y_train)
```

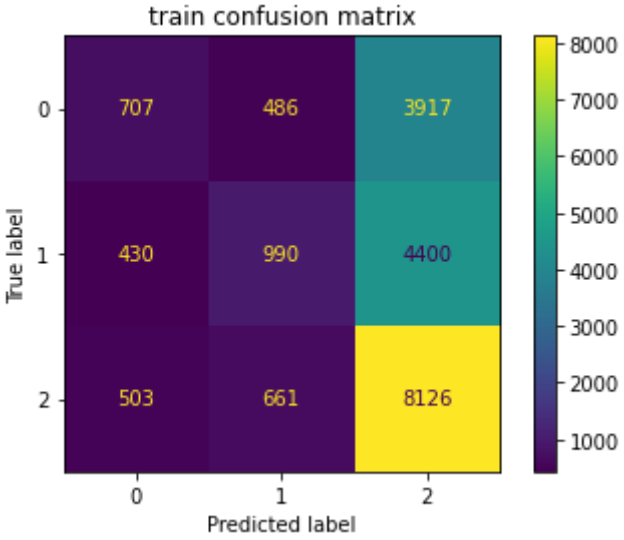
Out[57]:
```
0.4858061325420376
```

In [58]:
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

In [59]:
```
cmd=ConfusionMatrixDisplay(confusion_matrix(y_train,train_pred))
```

In [60]:
```
cmd.plot()
plt.title('train confusion matrix')
```

Out[60]:
```
Text(0.5, 1.0, 'train confusion matrix')
```



In [61]:
```
MLP.score(X_test,y_test)
```

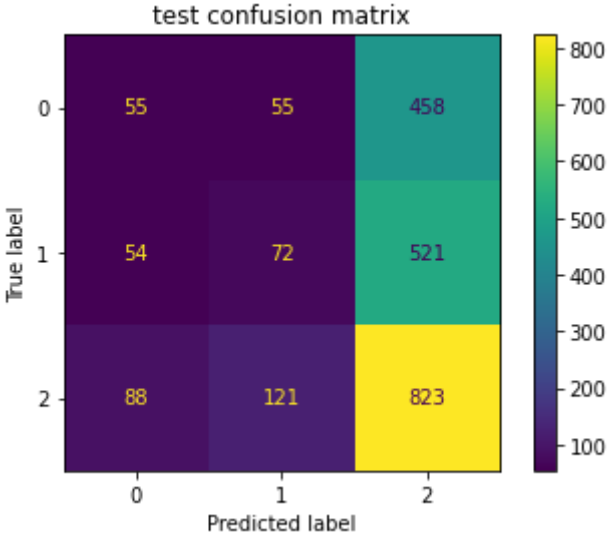
Out[61]:
```
0.4227859368046284
```

In [62]:
```
test_pred=MLP.predict(X_test)
```

In [63]:
```
cmd_test=ConfusionMatrixDisplay(confusion_matrix(y_test,test_pred))
```

In [64]:
```
cmd_test.plot()
plt.title('test confusion matrix')
```

Out[64]:
```
Text(0.5, 1.0, 'test confusion matrix')
```



cross validation

In [65]:
```
from sklearn.model_selection import cross_val_score
```

In [66]:
```
%%time
cv_scores=cross_val_score(estimator=MLP,X=X_train,y=y_train,cv=10,n_jobs=
```

```
CPU times: user 104 ms, sys: 224 ms, total: 328 ms
Wall time: 12min 12s
```

In [82]:
```
print('cross validation scores:',cv_scores,'\n')

print('cross validation mean and std for 1st model:')
cv_scores.mean(),cv_scores.std()
```

```
cross validation scores: [0.41295747 0.4223541  0.42185955 0.41988131 0.4
1493571 0.41048467
 0.40504451 0.41246291 0.42878338 0.43669634]

cross validation mean and std for 1st model:
```

Out[82]:
```
(0.4185459940652819, 0.008888466350593573)
```

In [67]:

Out[67]:
```
(array([0.41295747, 0.4223541 , 0.42185955, 0.41988131, 0.41493571,
        0.41048467, 0.40504451, 0.41246291, 0.42878338, 0.43669634]),
 0.4185459940652819,
 0.008888466350593573)
```

In [ ]:

In [68]:
```
#### more complicated MLP
```

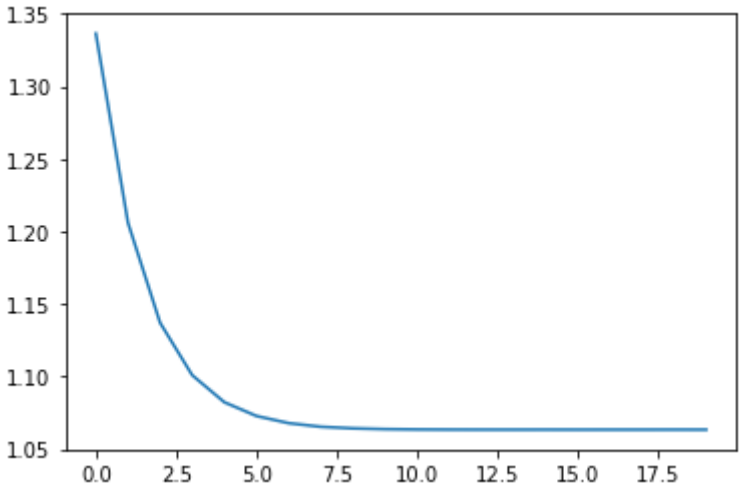
In [69]:
```
MLP2=MLPClassifier(hidden_layer_sizes=(2,2,2),activation='logistic',solve
                   max_iter=800,tol=0.001,early_stopping=False,
                   shuffle=True,
                   random_state=1)
```

In [70]:
```
MLP2.fit(X_train,y_train)
```

Out[70]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=(2, 2, 2), max_it
er=800,
              random_state=1, tol=0.001)
```

In [71]:
```
plt.plot(MLP2.loss_curve_)
```

Out[71]:
```
[<matplotlib.lines.Line2D at 0x7f457729e640>]
```



In [72]:
```
MLP2.score(X_train,y_train)
```

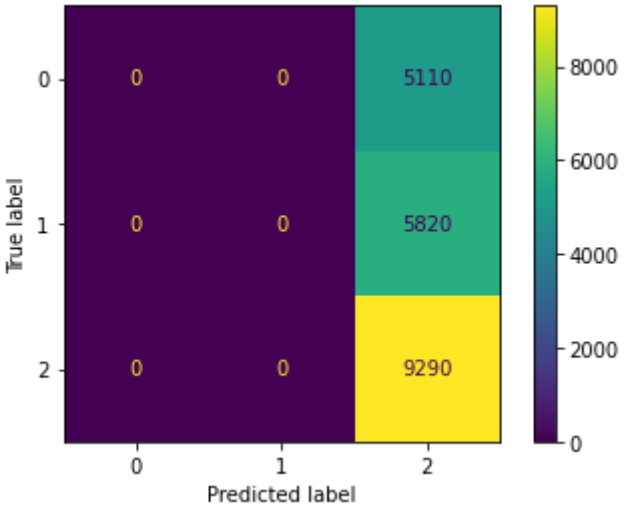
Out[72]:
```
0.45944609297725025
```

In [73]:
```
train_pred_2=MLP2.predict(X_train)
```

In [74]:
```
cmd2=ConfusionMatrixDisplay(confusion_matrix(y_train,train_pred_2))
```

In [75]:
```
cmd2.plot()
```

Out[75]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f457
7278c70>
```

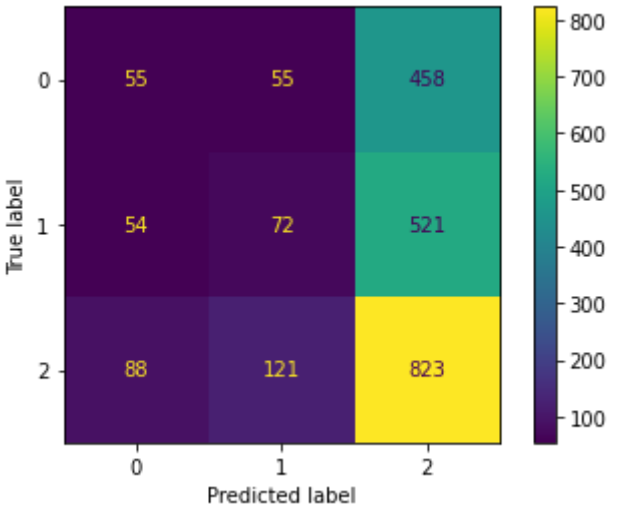


In [76]:
```
test_pred_2=MLP2.predict(X_test)
```

In [77]:
```
cmd_test_2=ConfusionMatrixDisplay(confusion_matrix(y_test,test_pred))
```

In [78]:
```
cmd_test_2.plot()
```

Out[78]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f457
723b970>
```



In [79]:
```
MLP2.score(X_test,y_test)
```

Out[79]:
```
0.45927903871829107
```

In [83]:
```
cv_scores=cross_val_score(estimator=MLP2,X=X_train,y=y_train,cv=10,n_jobs
```

In [84]:
```
print('cross validation scores:',cv_scores,'\n')

print('cross validation mean and std for 1st model:')
cv_scores.mean(),cv_scores.std()
```

```
cross validation scores: [0.45944609 0.45944609 0.45944609 0.45944609 0.4
5944609 0.45944609
 0.45944609 0.45944609 0.45944609 0.45944609]

cross validation mean and std for 1st model:
(0.4594460929772503, 5.551115123125783e-17)
```

Out[84]:

Για τα μοντέλα που επιλέχθηκαν πάλι το πιο απλό μοντέλο εμφανίζει καλύτερα σκορ(ακρίβεια ταξινόμησης-accuracy),περίπου κατά 3.5%,και πολύ μεγαλύτερη ταχύτητα εκπαίδευσης

In [ ]: