Κείμενο Τεκμηρίωσης Υπολογιστικής Εργασίας

Αναγνώριση Προτύπων

Ακαδημαϊκό Έτος 2023 – 2024

Λεωνίδας Πάστρας
π20155

Περιεχόμενα

Προλογος

Η υπολογιστικη εργασία υλοποιήθηκε σε γλώσσα python και χρισημοποιήθηκαν οι βιβλιοθήκες

- *numpy*
- *matplotlib*
- *pandas*
- *sklearn*

Επιπλέον το αρχείο **dataVisualisation.ipynb** είναι μορφής *jupyter* notebook

Προ-επεξεργασία Δεδομένων

Η προ-επεξεργασία των δεδομένων γίνεται όλη μέσω του αρχείου **data.py**. Μέσα στο αρχείο αυτό υπάρχει η κλάση HousingData η οποία παίρνει ως όρισμα το *path* του αρχείου **housing.csv**. Μέσα στην κλάση υπάρχη η μέθοδος Standardize_Housing_Data() η οπόια επιστρέφει σε μορφή πίνακα κλιμακωμένα τα δεδομένα του αρχείου **housing.csv**.

1. *Θα πρέπει να αναγνωρίσετε τα υποσύνολα των αριθμητικών και των κατηγορικών χαρακτηριστικών*

   Όλα τα χαρακτηριστικά είναι αριθμιτικά εκτός απο ocean_proximity το οπόιο είναι κατηγορικό. Συγκεκριμένα χωρίζεται σε 5 κατηγορίες *NEAR BAY, 1<H OCEAN, INLNAD, NEAR OCEAN & ISLAND*

2. *Για το υποσύνολο των αριθμητικών χαρακτηριστικών θα πρέπει να πειραματιστείτε με διαφορετικές τεχνικές κλιμάκωσης (scaling) των δεδομένων ώστε όλα τα αριθμητικά χαρακτηριστικά να αναπαρίστανται στην ίδια κλίμακα.*

   Η κλιμάκωση των αριθμιτικών δεδομένων πραγματοποιήθηκε με τον αλγόριθμο *MinMaxScaler*. Δηλαδή ο μετασχηματισμός των δεδομένων γίνεται με βάση το ελάχιστο και το μέγιστο των αρχικών τιμών τους, δίνοντάς τους νέες τιμές που να είναι ανάλογες μεταξύ τους στο διάστημα [0, 1].

```
# scale features
scaler = MinMaxScaler()
model=scaler.fit(temp_rows)
scaled_data=model.transform(temp_rows)
```

Η temp_rows περιέχει όλα τα αριθμητικά χαρακτηριστικά και μετά τον μετασχηματιμό σώζονται στην scaled_data

3. *Για το υποσύνολο των κατηγορικών χαρακτηριστικών μπορείτε να χρησιμοποιήσετε την One Hot Vector κωδικοποίηση ώστε τα εν λόγω δεδομένα να λάβουν διανυσματική αναπαράσταση.*

```
# ocean_proximity options: ['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']
# one hot vector
def Standardize_ocean_proximit(ocean_proximity):
    if ocean_proximity == "NEAR BAY":
        return [1, 0, 0, 0 ,0]
    elif ocean_proximity == "<1H OCEAN":
        return [0, 1, 0, 0 ,0]
    elif ocean_proximity == "INLAND":
        return [0, 0, 1, 0 ,0]
    elif ocean_proximity == "NEAR OCEAN":
        return [0, 0, 0 ,1 ,0]
    else:
        return [0, 0, 0 ,0 ,1]
```

Η κωδικοκοποίηση *One Hot Vector* χρησιμοποιήθηκε στα κατηγορικά δεδομένα ocean_proximity. Συγκεκριμένα έγινε μέσω της μεθόδου Standardize_ocean_proximity() η οποία παίρνει ως όρισμα μια κατηγορική τιμή και επιστρέφει το αντίστοιχο διάνυσμα.

4. *Θα πρέπει να αναγνωρίσετε αν υπάρχουν αριθμητικά χαρακτηριστικά με ελλιπείς τιμές. Για τις συγκεκριμένες εγγραφές μπορείτε να συμπληρώσετε τις τιμές που απουσιάζουν με την διάμεση τιμή του χαρακτηριστικού.*

Υπήρχαν ελλιπείς δεδομένα μόνο στην στήλη total_bedrooms. Αρχικά σώζουμε τα index των άδειων κελιών σε έναν πίνακα, στην συνέχεια υπολογίζουμε την μέση τιμή όλων των μη-κενών κελιών και τέλος συμπληρώνουμε τα κενά κελία με την μέση τιμή που υπολογίσουμε.

Οπτικοποίηση Δεδομένων

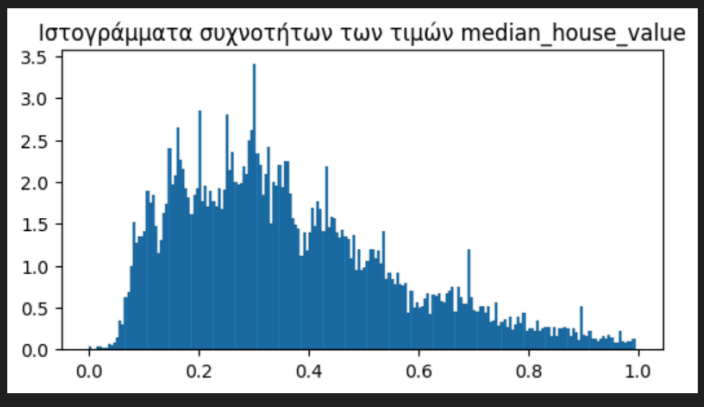
Η οπτικοποίηση των δεδομένων γίνεται στο αρχείο **dataVisualisation.ipynb** με την βοήθεια της βιβλιοθηκης *matplotlib*.

1. *Να αναπαραστήσετε γραφικά τα ιστογράμματα συχνοτήτων (που αντιστοιχούν στις συναρτήσεις πυκνότητας πιθανότητας) για κάθε μία από τις 10 μεταβλητές που εμπλέκονται στο πρόβλημα.*

Οι γραφικές αναπαραστήσεις των ιστογράμματων συχνότητας πυκνότητας πιθανότητας είναι τα 10 πρώτα ιστογράμματα που φένονται στο αρχείο και δημιουργούνται με ανάλογο τρόπο:

```
array = dataMethods.Median_house_value_column()
fig, axs = plt.subplots(figsize=(6, 3))
hist = axs.hist(array, np.arange(0, 1, 0.005), edgecolor='black', linewidth=0.1, density=True)
axs.set_title('Ιστογράμματα συχνοτήτων των τιμών median_house_value')
plt.show()
```

✓ 0.6s



Παράδειγμα δημιουργείας ιστογράμματος των δεδομένων Median_House_Value

I. Σώζουμε τα δεδομένα που θέλουμε να αναπαραστήσουμε γραφικά στην λίστα array. Για να πάρουμε τα συγκεκριμένα δεδομένα κάθε χαρακτηριστηκού καλούμε την ανάλογη μέθοδο απο το αρχείο **dataMethods.py**

II. Ορίζουμε το μέγεθος του

III. Δημιουργούμε το ιστόγραμμα και ορίζουμε το χαρακτηριστηκό του density=True. Με αυτόν τον τρόπο γίνεται αναπαράσταση της συνάρτησης πυκνότητας πιθανότητας των δεδομένων της λίστας array.

2. *Προσπαθήστε να δημιουργήσετε δισδιάστατα γραφήματα των δεδομένων στα οποία να αναπαρίστανται με ευδιάκριτο τρόπο συνδυασμοί 2, 3 ή και 4 μεταβλητών*.

Σε αυτό το ερώτημα έφτιαξα γραφήματα με σκοπό να καταλάβω συχέτιση μεταξύ τον διαφόρων δεδομένων. Εκτός απο ιστογράμματα έφτιαξα γραφήματα και διαγράμματα διασποράς.

Παλινδρόμηση Δεδομένων

1. Perceptron algorithm

Ο κώδικας που υλοποιεί τον αλγόριθμο perceptron χωρίζεται σε τρία αρχεία.

1.1. Class Perceptron_Dataset

Η κλάση Perceptron_Dataset βρίσκεται μέσα στο αρχείο **dataMethods.py** και σκοπός της είναι να φέρει τα δεδομένα της κλάσης HousingData στην κατάλληλη μορφή έτσι ώστε να μπορεί να τα χρησιμοποιήσει ο *Perceptron*.

I. Αρχικά χωρίζει τις ετικέτες (labels) των δεδομένων απο τα υπόλοιπα δεδομένα. Και τις σώζει στην λίστα labels αφού πρώτα τις φέρει σε δυαδική μορφή ανάλογα με το αν ξεπερνούν το μέγεθος της μεταβλητής threshold

```
def Create_labels(this):
    for i in range(num_of_data):
        label = 1.0 if mydata[i].pop(8) < this.threshold else 0.0
        Perceptron_Dataset.labels.append(label)
```

II. Στην συνέχεια χώρίζει τα δεδομένα σε δύο ομάδες. Η Πρώτη ομάδα περιέχει το ένα δέκατο (1/10) των δεδομένων και χρησιμοποιήται κατα την φάση του έλεγχου του αλγορίθμου. Τα δεδομένα και οι ετικέτες αυτης της ομάδας σώζονται στους πίνακες testing_data[] & testing_labels[] αντίστοιχα. Η Δεύτερη ομάδα περιέχει τα υπόλιπα εννέα δέκατα (9/10) των δεδομένων, χρησιμοποιείται για την εκπαίδευση του αλγορίθμου τα οπόια σώζονται επίσης στους πίνακες training_data[] & training_labels[]. Τα κομμάτια των δεδομένων που χρησιμοποιούται στην φάση του ελέγχου και στην φάση της εκπάιδευσης αλλάζουν ανάλογα με το όρισμα fold της κλάσης Perceptron_Dataset.

III. Κατα την εισαγωγή των δεδομένων στους πίνακες testing_data[] και training_data[] τα διανυσματικά δεδομένα της κατηγορίας ocean_proximity σώζονται ως πέντε χαρακτηριστηκά, το καθένα αντιπροσωπεύοντας μία διάσταση του διανύσματος.

IV. Επιπλέον στην αρχή κάθε εγγραφής των πινάκων testing_data[] και training_data[] εισάγεται το *bias* (1.0)

Στο τέλος αυτής της διαδικασίας υπάρχουν δύο δισδιάστατοι πίνακες training_data[] και testing_data[] διαστάσεων **n x 14** και **(n – n * 1/10) x 14** αντίστοιχα (οπου n ο αριθμός των δεδομένων και 14 ο αριθμός των χαρακτηριστηκών). Επίσης υπάρχουν και οι δύο μονοδιάστατοι πίνακες training_labels[] & testing_labels[] διαστάσεων **n & n – n * 1/10** αντίστοιχα.

1.2. Class Perceptron

Η κλάση Perceptron βρίσκεται μέσα στο αρχείο **perceptron.py** και υλοποιεί τον αλγόριθμο Perceptron.

1.2.1. def Train_weights()

Τα βήματα εκπάιδευσης του αλγορίθμου είναι τα εξής

I. Αρχικοποιούμε τα βάρη μας με πολύ μικρές τυχαίες τιμές

```
dimensions = len(data[0]) # get the number of features/dimensions of our data
this.weights = np.random.normal(0, 0.1, dimensions) * 0.1#np.zeros(dimensions)
```

II. Και μετά για κάθε δεδομένο του πίνακα training_data[]

a) Υπολογίζουμε την ετικέτα του

```
for indx, data_i in enumerate(data):
    linear_output = np.dot(this.weights, data_i) # f(w, x) = <w, x>
    class_predicted = this.Activation_function(linear_output)
```

b) Ενημερώνουμε τα βάρη

```
# Perceptron update rule
update = this.lr * (data_labels[indx] - class_predicted)
this.weights = this.weights + (update * data_i)
```

1.2.2. def Activation_function()

Ως συνάρτηση ενεργοποίησης χρησιμοποιούμε την *Heaviside step function* η οπόια παίρνει σαν όρισμα το εσωτερικό γινόμενο των βαρώων με των αντίστοιχων δεδομένων και επίστρέφη 1 ή 0 ανάλογα με το πρόσιμό του εσωτερικού γινόμενου.

```
def Activation_function(this, linear_output):
    unit_step_function = np.where(linear_output >= 0, 1, 0)
    return unit_step_function
```

1.2.3. def Predict()

Η μέθοδος πρόβλεψης Predict παίρνει ως όρισμα ένα πίνακα data_i που αντιπροσωπεύει την i-οστή εγγραφή του πίνακα των δεδομένων, και στην συνέχεια υπολογίζει το εσωτερικό γινόμενο της με τα βάρη. Έπειτα περνάει το αποτέλεσμα του εσωτερικού γινόμενου στην Activation_function() η οποία με την σειρά της επιστρέφει 1 η 0 ανάλογα με την κλάση στην οποία κατηγοροποίησε το δεδομένο.

```
def Predict(this, data_i):
    linear_output = np.dot(data_i, this.weights)
    label_predicted = this.Activation_function(linear_output)
    return label_predicted
```

1.2.4. def Mean_absolute_error() & def Mean_squared_error()

Η MAE παίρνει ως όρισμα έναν πίνακα data, που περιέχει τα δεδομένα στα οποία θέλουμε να υπολογίζουμε το σφάλμα, και έναν πίνακα data_labels που περιέχει της αντίστοιχες κλάσεις των δεδομένων του data. Αρχικοποιεί την τιμή του αθροισματός sum με το 0. Στην συνέχεια για κάθε εγγραφή της λίστας data βρίσκει την αναμενόμενη κλάση της μέσω της μεθόδου predict() και μετα υπολογίζει και αθροίζει με το sum την απόλυτη τιμή της διαφοράς της αναμενόμενης κλάσης και της πραγματικής κλάσης. Τέλος επιστρέφει τον λόγο του αθροίσματος sum με τον αριθμο n των δεδομένων.

Οι μέθοδοι MAE & MSE λειτουργόυν με ακριβώς τον ίδιο τρόπο εκός απο τον υπολογισμό του αθρίσματος όπου η MAE υπολογίζει την απόλυτη τιμή της διαφοράς της αναμενόμενης κλάσης και της πραγματικής κλάσης ενώ η MSE το τετράγων της διαφοράς τους. Στον δυαδικό πρόβλημα που εξετάζουμε αυτή η διαφορά είναι πάντα ή -1 ή 0 ή 1, συνεπώς η απόλυτη τιμή και το τετράγωνο του αποτελέσματος είναι σε κάθε περίπτωση ίδιο. Συνεπώς σε αυτο το ερώτημα η ύπαρξη της MSE είναι περιττή και δεν χρησιμοποιείται.

1.3. Perceptron_testing.py

Εκτελώντας αυτό το αρχείο υλοποιείται ο αλγόριθμος Perceptron και εμφανίζονται στο αρχείο **log.txt** τα αποτελέσματα. Τα αποτελέσματα αναφέρουν την ακρίβεια παλινδρόμησης που επιτυγχάνει σε όρους Μέσου Τετραγωνικού Σφάλματος και Μέσου Απόλυτου Σφάλματος τόσο κατά την φάση της εκπαίδευσης όσο και κατά την φάση του ελέγχου σύμφωνα με την μέθοδο της 10- πλής διεπικύρωσης (10 fold cross validation).

Όπως αναφέρθηκε προηγουμένος Τα κομμάτια των δεδομένων που χρησιμοποιούται στην φάση του ελέγχου και στην φάση της εκπάιδευσης αλλάζουν ανάλογα με το όρισμα fold της κλάσης Perceptron_Dataset. Έτσι βρίσκοντας τον μέσω όρο του σφάλαμτος μετά απο 10 κλήσεις της Perceptron_Dataset με το όριμσα fold να αυξάνεται κατα ένα μετά απο κάθε κλήσει, επιτυγχάνουμε τον υπολογισμό του σφάλματος σύμφωνα με την μέθοδο της 10- πλής διεπικύρωσης.

2. Least Squares Algorithm

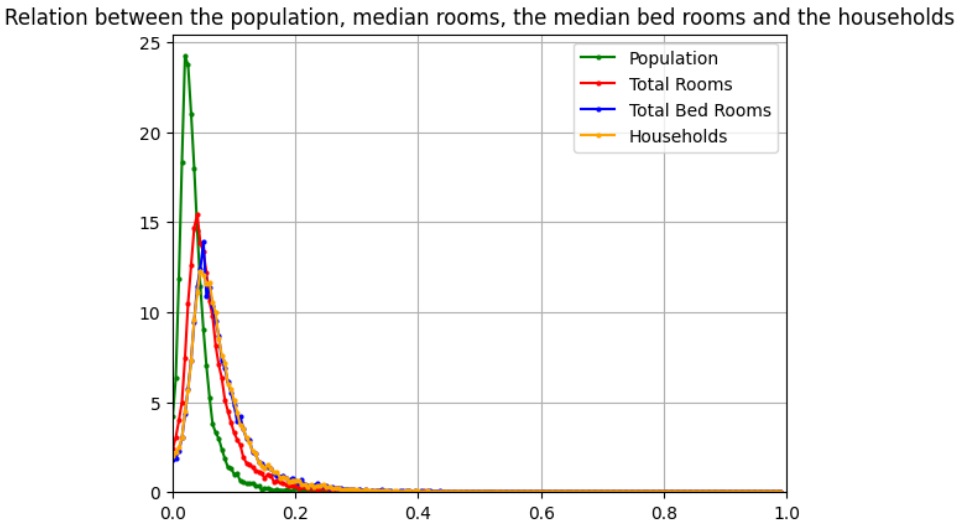
Ο κώδικας που υλοποιεί τον Αλγόριθμο Ελάχιστου Τετραγωνικού Σφάλματος χωρίζεται σε τρία αρχεία.

2.1. Class Linear_Regression_Dataset

Η κλάση Linear_Regression_Dataset βρίσκεται μέσα στο αρχείο **dataMethods.py** και σκοπός της είναι να φέρει τα δεδομένα της κλάσης HousingData στην κατάλληλη μορφή έτσι ώστε να μπορεί να τα χρησιμοποιήσει ο Αλγόριθμος Ελάχιστου Τετραγωνικού Σφάλματος.

I. Αρχικά χωρίζει την εξαρτημένη μεταβλητή (median_house_value) απο τις μεταβλητές εισόδου. Οι τιμές τις εξαρτημένης μεταβλητής σώζωνται στην λίστα labels

II. Στο δεύτερο βήμα παραμετροποιούνται οι μεταβλητές εισόδου έτσι ώστε να αποφευχθεί η πολυσυγγραμικότητα (Multicollinearity).
Παρατηρούμε απο την οπτικοποίηση των δεδομένων την παρακάτω σχέση:



Αφαιρούμε τα δεδομένα Households & Total Bed Rooms απο το σύνολο των δεδομένων μας.

```
def Fix_Multicollinearity(this):
        mydata = np.transpose(mydata)
        mydata.pop(6) # Throw away households
        mydata.pop(4) # Throw away total_bedrooms
        mydata = np.transpose(mydata)
```

III. Τα υπόλοιπα βήματα είναι όμοια με τα βήματα II, III & IV της κλάσης Perceptron_Dataset.

2.2. Class Least_Squares_Method

Η κλάση Least_Squares_Method βρίσκεται μέσα στο αρχείο **linear_regression.py** και υλοποιεί τον αλγόριθμο τον Αλγόριθμο Ελάχιστου Τετραγωνικού Σφάλματος.

2.2.1. def Train()

Η μέθοδος Train() πέρνει ως όρισμα τα δεδομένα με τις αντίστοιχες ετικέτες και ενημερώνει τα βάρει με βασή την παρακάτω εξήσωση:

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

Όπου:

- **w** = Βάρη
- **X** = Πίνακας δεδομένων
- **y** = Πίνακας με ετικέτες

```
def Train(this, data, data_labels):
        X = data
        y = data_labels

        X_T = np.transpose(X)
        XTX = np.float64(np.dot(X_T, X))
        XTX_inv = np.linalg.inv(XTX)
        XTy = np.dot(X_T, y)

        this.weights = np.dot(XTX_inv, XTy)
```

2.2.2. def Predict()

Για να κάνει προβλέψεις υπολογίζει το εσωτερικό γινόμενω του διανύσματος βάρους με του διανύσματος των δεδομένων

2.2.3. def Mean_absolute_error() & def Mean_squared_error()

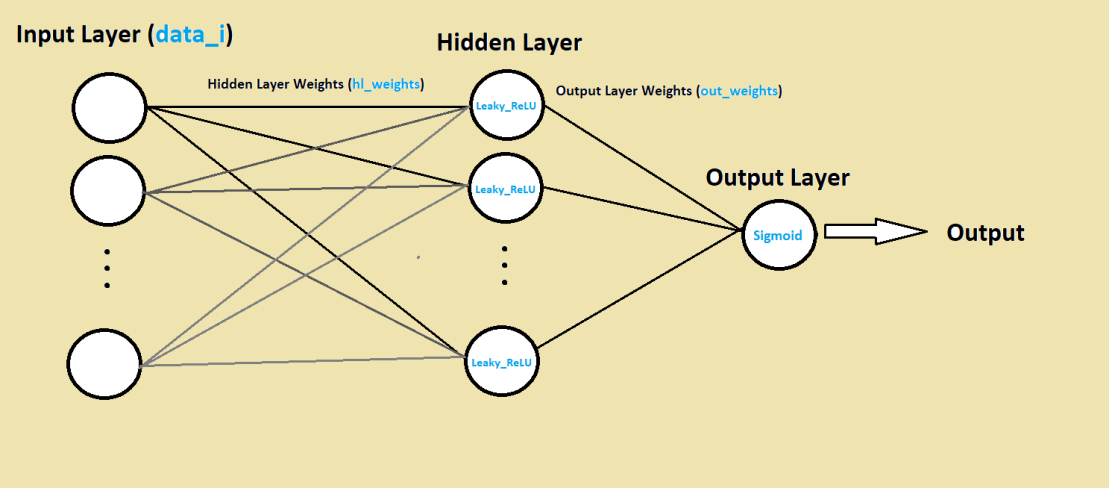
Όπως λειτουργούν και στην κλάση Perceptron.

2.3. Least_squares_testing.py

Εκτελώντας αυτό το αρχείο υλοποιείται ο Αλγόριθμος Ελάχιστου Τετραγωνικού Σφάλματος. και εμφανίζονται στο αρχείο **log.txt** τα αποτελέσματα. Η λειτουργία είναι όμοια με την λειτουργία του **perceptron.py.**

3. Πολυστρωματικό νευρωνικό δίκτυο

Για αυτό το ερώτημα υλοποίησα ενα Multilayer Perceptron (MLP) με σκοπό να χωρίζει τα δοδεμένα σε δύο κλάσεις (Binary Classification) ανάλογα με την προβλεπόμενη τιμή του σπιτιού.



Η αρχιτεκτονική του MLP που υλοποιείται στο αρχείο MLP.py

Όπως φένεται στο σχήμα το MLP αποτελείται απο τρία στρώματα (Input, Hidden, Output), η επιλογή να έχει μόνο ένα Hidden Layer έγινε για λόγους ευκολίας.

Το πολυστρωματικό νευρωνικό δίκτυο υλοποιείται μέσω της κλάσης MultilayerPerceptron που βρίσκεται στο αρχείο **MLP.py**

Στον constructor της κλάσης __init__ ορίζεται ο αριθμός τον κόμβων του Hidden Layer απο την μεταβλητή n_nodes=10. Διάλεξα το 10 διότι είναι ανάμεσα στον αριθμό τον εισόδων (14) και τον αριθμώ των εξόδων (1).

Μέσα στην κλάση υπάρχουν επίσης τέσσερεις μέθοδοι που υλοποιούν τις συναρτήσεις που χρησιμοποίούνται κατα την λειτουργία του νευρωνικού δίκτυου. Οι συναρτήσεις αυτές είναι:

- Leaky ReLU

Η Leaky ReLU (Rectified Linear Unit) χρησιμοποιείται στους κόμβους του hidden layer διότι είναι μια απλή και υπολογιστικά αποδοτική συνάρτηση ενεργοποίησης που ειναι εύκολη στην υλοποίηση και στην χρήση της. Ένα πρόβλημα με την παραδοσιακή συνάρτηση ενεργοποίησης ReLU είναι ότι οι νευρώνες μερικές φορές μπορεί να γίνουν "νεκροί" κατά τη διάρκεια της εκπαίδευσης, πράγμα που σημαίνει ότι πάντα εξάγουν μηδέν για οποιαδήποτε είσοδο και δεν συνεισφέρουν στη διαδικασία μάθησης. Η Leaky ReLU αντιμετωπίζει αυτό το πρόβλημα επιτρέποντας μια μικρή, μη μηδενική κλίση για αρνητικές εισόδους, εμποδίζοντας τους νευρώνες από το να γίνουν εντελώς αδρανείς.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

Leaky ReLU

- Sigmoid

Η συνάρτηση Sigmoid περιορίζει τις τιμές της εισόδου στο εύρος (0, 1). Στο πλαίσιο της δυαδικής ταξινόμησης, αυτό είναι ευνοϊκό, διότι μπορεί να ερμηνευτεί ως η πιθανότητα να ανήκει στη θετική κατηγορία. Τιμές κοντά στο 0 υποδηλώνουν χαμηλή πιθανότητα, ενώ τιμές κοντά στο 1 υποδηλώνουν υψηλή πιθανότητα. Για αυτόν τον λόγο χρησιμοποιείται ως συνάρτηση ενεργοποίησης για τον κόμβο στο στρώματος εξόδου.

$$S(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid

- Binary Cross-Entropy Loss

Η Συνάρτηση Binary Cross-Entropy Loss, είναι η συνάρτηση σφάλματος που χρησιμοποιείται στο πρόβλημα. Μετρά την απόδοση ενός μοντέλου ταξινόμησης, το οποίο εξάγει μια πιθανοτική τιμή μεταξύ 0 και 1, που αντιπροσωπεύει την πιθανότητα για το αν ένα συγκεκριμένο παράδειγμα ανήκει στη θετική κατηγορία. Ο σκοπός αυτής της συνάρτησης σφάλματος είναι να τιμωρήσει τα μοντέλα ανάλογα με το μέγεθος της απόκλησης των προβλεπόμενων ετικέτων από τις πραγματικές ετικέτες.

$$Log\_Loss(y, p) = - (y * \log_2(p) + (1 - y) * \log_2(1 - p))$$

όπου το 'y' αντιπροσωπεύει την πραγματική ετικέτα και το 'p' είναι η προβλεπόμενη πιθανότητα για το σημείο δεδομένων να ανήκει στη θετική κατηγορία.

- Παράγωγος της Leaky ReLU

  Η παράγωγος της Leaky ReLU χρησιμοποιείται κατα την διαδικασία του backpropagation.

$$f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0.01 & \text{otherwise.} \end{cases}$$

Leaky ReLU Derivative

Όλες αυτες οι συναρτήσεις χρησιμοποιούνται στη μέθοδο Train() που εκτελεί την διαδικασία μέσα απο την οποία εκπαιδέυονται τα βάρη του νευρωνικού δυκτίου.

I. Κατα την κλήση της η Train() ορίζει την τιμή ρυθμού εκμάθησης (learning rate) και αρχικοποίει τους πίνακες με τα βάρη του hidden layer και του output layer με τυχάιες τιμές. Συγκεκριμένα ο πίνακας για τα βάρη του hidden layer είναι δυσδιάστατος της μορφής [10×14] (10 hidden layer nodes, 14 data features), και ο πίνακας για τα βάρη του output layer ειναι ένας μονοδιάστατος πίνακας μήκους 10 (τα αποτελέσματα των συναρτήσεων ενεργοπόισης των 10 hidden nodes)

```
def Train(this, data, data_labels):
    this.lr = 0.01
    dimensions = len(data[0]) # get the number of features/dimensions of our d
    for _ in range(this.n_nodes):
        this.hl_weights.append(np.random.normal(0, 0.1, dimensions) * 1) # ini
    this.out_weights = np.random.normal(0, 0.1, this.n_nodes) * 1 # init the o
```

II. Στην συνέχεια υπολογίζει την προβλεπόμενη κλάση για κάθε δεδομένο . Δηλαδή για κάθε δεδομένο (data_i) υπολογίζει την τιμή του κάθε κόμβου στο hidden layer και μετά την αθροίζει στην μεταβλητή output (πάντα πολλαπλάσιάζοντας την κάθε τιμή με το αντίστοιχο βάρος). Η προβλεπόμενη κλάση υπολογίζεται στην μεταβλητή class_predicted καλώντας την συνάρτηση sigmoid() με όρισμα την μεταβλητή output.

III. Στο επόμενο βήμα γίνεται η διόρθωση των βαρών, η αλλίως backpropagation. Αφού υπολογιστεί το σφάλμα μέσω της συναρτησης log_loss(), μετά με την βοήθεια της μεταβλητής delta υπολογίζεται η καινούρια τιμή των out_weights και των hl_weights. Η ακριβής λογική της παραπάνω διαδικασίας περιγράφεται εδω.

IV. Τέλος υπολίζεται η καινούρια τιμή του ρυθμού εκμάθησης με βάσει το *Exponential Decay Leaning Rate Scheduler*

$$learning\ rate = learning\ rate * e^{-k * indx}$$

Η Εκτέλεση του MLP γίνεται στο αρχείο **MLP_testing.py** και ακολουθεί τα ίδια βήματα με την εκτέλεση του Perceptron.