ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πτυχιακή Εργασία

Υλοποίηση μεθόδων εξόρυξης δεδομένων με χρήση Tensorflow

Απόστολος Παναγιωτόπουλος-Θωμάς

Α.Μ.: 71323845

Επιβλέπων καθηγητής: Πάρις Μαστοροκώστας

Αθήνα, Οκτώβριος 2022

UNIVERSITY OF WEST ATTICA

SCHOOL OF ENGINEERING

DEPARTMENT OF INORMATICS AND COMPUTER ENGINEERING

Diploma Thesis

Data Mining with Tensorflow

Apostolos Panagiotopoulos-Thomas

R.N.: 71323845

Supervisor καθηγητής: Paris Mastorocostas

Athens, October 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση μεθόδων εξόρυξης δεδομένων με χρήση Tensorflow

Η πτυχιακή εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

Α/Α	ΟΝΟΜΑ ΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
1	Πάρις Μαστοροκώστας	Καθηγητής ΠΑ.Δ.Α.	Paris Mastorokostas Digitally signed by Paris Mastorokostas Date: 2022.10.18 15:14:51 +03'00'
2	Χρήστος Τρούσσας	Επίκουρος Καθηγητής ΠΑ.Δ.Α.	Christos Troussas Digitally signed by Christos Troussas Date: 2022.10.18 15:57:46 +03'00'
3	Παναγιώτα Τσελέντη	Ε.ΔΙ.Π. ΠΑ.Δ.Α.	Panagiota Tselenti Digitally signed by Panagiota Tselenti Date: 2022.10.18 19:59:57 +03'00'

Αθήνα, Οκτώβριος 2022

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος **Απόστολος Παναγιωτόπουλος-Θωμάς** του **Παναγιώτη**, με αριθμό μητρώου **71323845** φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής **Μηχανικών** του Τμήματος **Μηχανικών Πληροφορικής και Υπολογιστών**, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

(Υπογραφή)

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή κ. Πάρι Μαστοροκώστα για την πολύτιμη βοήθεια και καθοδήγηση του στην εκπόνηση της πτυχιακής μου εργασίας.

Ευχαριστώ, τέλος, την οικογένεια μου για την απεριόριστη και αναντικατάστατη συμπαράσταση και βοήθεια της.

Περίληψη

Σκοπός της πτυχιακής αυτής εργασίας είναι αρχικά η εξοικείωση με το Tensorflow, κάνοντας χρήση του βιβλίου «Machine Learning with TensorFlow» του Nishant Shukla και στην συνέχεια η υλοποίησή μεθόδων μηχανικής μάθησης στην εξόρυξη δεδομένων (παλινδρόμηση, ταξινόμηση, συσταδοποίηση, αυτοκωδικοποιητές) ώστε οι μέθοδοι να καταρτίσουν μία εργαλειοθήκη (toolbox).

Λέξεις – Κλειδιά

Εξόρυξη Δεδομένων, Tensorflow, Παλινδρόμηση, Κατηγοριοποίηση, Συσταδοποίηση, Αυτοκωδικοποιητές.

Data Mining with Tensorflow

Apostolos Panagiotopoulos-Thomas

Abstract

The purpose of the thesis is initially to get familiar with Tensorflow, using the book «Machine Learning with TensorFlow» by Nishant Shukla and then to implement machine learning methods in data mining (regression, classification, clustering, autoencoders) thus to create a toolbox.

Keywords

Data Mining, Tensorflow, Regression, Classification, Clustering, Autoencoders.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Γραφημάτων

Κατάλογος Πινάκων

Συντομογραφίες & Ακρωνύμια

ΑΓΒΔ	Ανακάλυψης Γνώστης από Βάσεις Δεδομένων
ΕΔ	Εξόρυξη Δεδομένων
ΤΝΔ	Τεχνητό Νευρωνικό Δίκτυο
BMU	Best Matching Unit
DAE	Denoising Autoencoder
DM	Data Mining
KDD	Knowledge Discovery in Databases
MSE	Mean Squared Error
SOM	Self-Organizing Map

1 Θεωρία

1.1 Εισαγωγή στην Εξόρυξη Δεδομένων

Η ραγδαία πρόοδος στην τεχνολογία συλλογής και αποθήκευσης δεδομένων, η οποία ήταν αποτέλεσμα της καινοτομίας σε διάφορους τομείς, όπως το διαδίκτυο, το ηλεκτρονικό εμπόριο, οι ηλεκτρονικές συναλλαγές , οι αναγνώστες barcode, οι κινητές συσκευές και οι ευφυείς μηχανές, έχει επιτρέψει στους διάφορους οργανισμούς να συσσωρεύουν τεράστιες ποσότητες δεδομένων. Ωστόσο, η εξαγωγή χρήσιμων πληροφοριών έχει αποδειχθεί εξαιρετικά δύσκολη. Το πρόβλημα αυτό προσπαθεί να επιλύσει η Εξόρυξη Δεδομένων (Data Mining), ένας ταχέως αναπτυσσόμενος τομέας που ασχολείται με την ανάπτυξη τεχνικών οι οποίες έχουν σαν στόχο να βοηθήσουν τους κατόχους των δεδομένων να κάνουν έξυπνη χρήση αυτών των συλλογών.

Σύμφωνα με τους (Tan, Steinbach, Karpatne, & Kumar, 2020) η εξόρυξη δεδομένων (ΕΔ) είναι η διαδικασία της αυτόματης ανακάλυψης χρήσιμων πληροφοριών μέσα από μεγάλες δεξαμενές πληροφοριών .

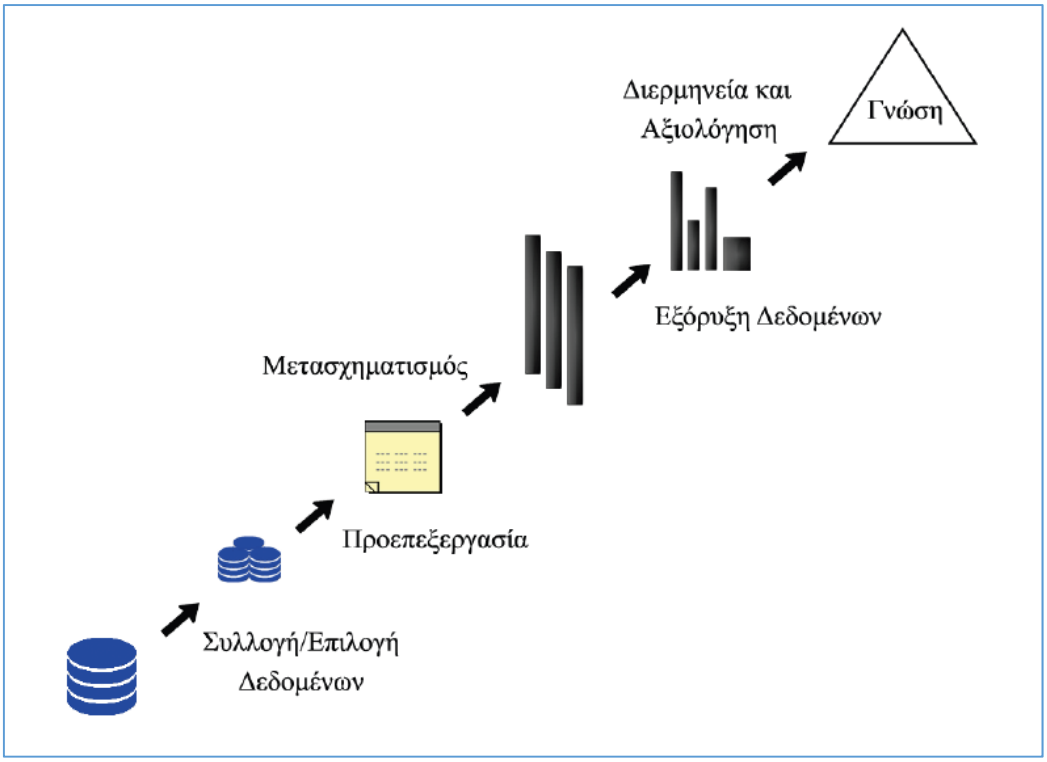
Η Εξόρυξη Δεδομένων αποτελεί ένα επιμέρους βήμα της διαδικασίας της Ανακάλυψης Γνώσης από Βάσεις Δεδομένων – ΑΓΒΔ (Knowledge Discovery in Databases – KDD), τα στάδια της οποίας είναι τα ακόλουθα (Βλαχάβας, Κεφαλάς, Βασιλειάδης, Κόκκορας, & Σακελλαρίου, 2000):

1. **Συλλογή Δεδομένων** (**Data Collection**), στο στάδιο αυτό δημιουργείται το σύνολο δεδομένων στο οποίο θα εφαρμοστεί η αναζήτηση.
2. **Προεπεξεργασία Δεδομένων** (**Preprocessing**), στο βήμα αυτό αντιμετωπίζονται περιπτώσεις ελλιπών δεδομένων, πεδίων με τιμές που ουσιαστικά τα καθιστούν κενά, πεδίων με τιμές που υπονοούν (κατά σύμβαση) κάτι άλλο, κλπ.. Το συγκεκριμένο στάδιο ονομάζεται και *στάδιο καθαρισμού των δεδομένων (data cleaning)*.
3. **Μετασχηματισμός Δεδομένων** (**Transformation**), στο στάδιο αυτό τα δεδομένα μετασχηματίζονται ώστε να διευκολυνθεί η ανακάλυψη της γνώσης. Τέτοιοι μετασχηματισμοί μπορεί να περιλαμβάνουν για παράδειγμα
    - την μείωση του αριθμού των υπό εξέταση χαρακτηριστικών (dimensionality reduction) με επιλογή ορισμένων εξ' αυτών (feature selection ή attribute selection),

- την ομοιόμορφη κωδικοποίηση της ποιοτικά ίδιας πληροφορίας,
- τη μετατροπή συνεχόμενων αριθμητικών τιμών σε διακριτές τιμές, (διακριτοποίηση), κλπ.

4. **Εξόρυξη Δεδομένων** (**Data Mining**), πρόκειται για καθαρά ένα υπολογιστικό στάδιο στο οποίο γίνεται η ουσιαστική αναζήτηση της γνώσης στα δεδομένα.
5. **Διερμηνεία και Αξιολόγηση** (**Interpretation/Evaluation**), στο βήμα αυτό γίνεται ερμηνεία και αξιολόγηση των ευρεθέντων προτύπων, πιθανώς με υποβοήθηση γραφικών απεικονίσεων των προτύπων ή/και των δεδομένων που περιγράφονται από το πρότυπο (pattern/data visualization).

Σε αυτό το στάδιο της ΕΔ της ΑΓΒΔ εφαρμόζεται κάποιος αλγόριθμος για την παραγωγή ενός μοντέλου. Έχοντας καθαρίσει και μετασχηματίσει τα δεδομένα, είναι έτοιμα να χρησιμοποιηθούν από κάποιον αλγόριθμο, ώστε να δημιουργηθεί κάποιο μοντέλο. Θέλουμε να χρησιμοποιήσουμε το μοντέλο αυτό, το οποίο δημιουργήθηκε με βάση κάποια γνωστά δεδομένα, έτσι ώστε να μπορεί να μας δώσει απάντηση για την τιμή ενός χαρακτηριστικού-μεταβλητής στόχου για νέα, άγνωστα δεδομένα (Βερύκιος, Καγκλής, & Σταυρόπουλος, 2015).



Εικόνα 1-1 Βασικά στάδια Ανακάλυψης Γνώσης από Βάσεις Δεδομένων (Βερύκιος, Καγκλής, & Σταυρόπουλος, 2015)

1.1.1 Ιστορική αναδρομή

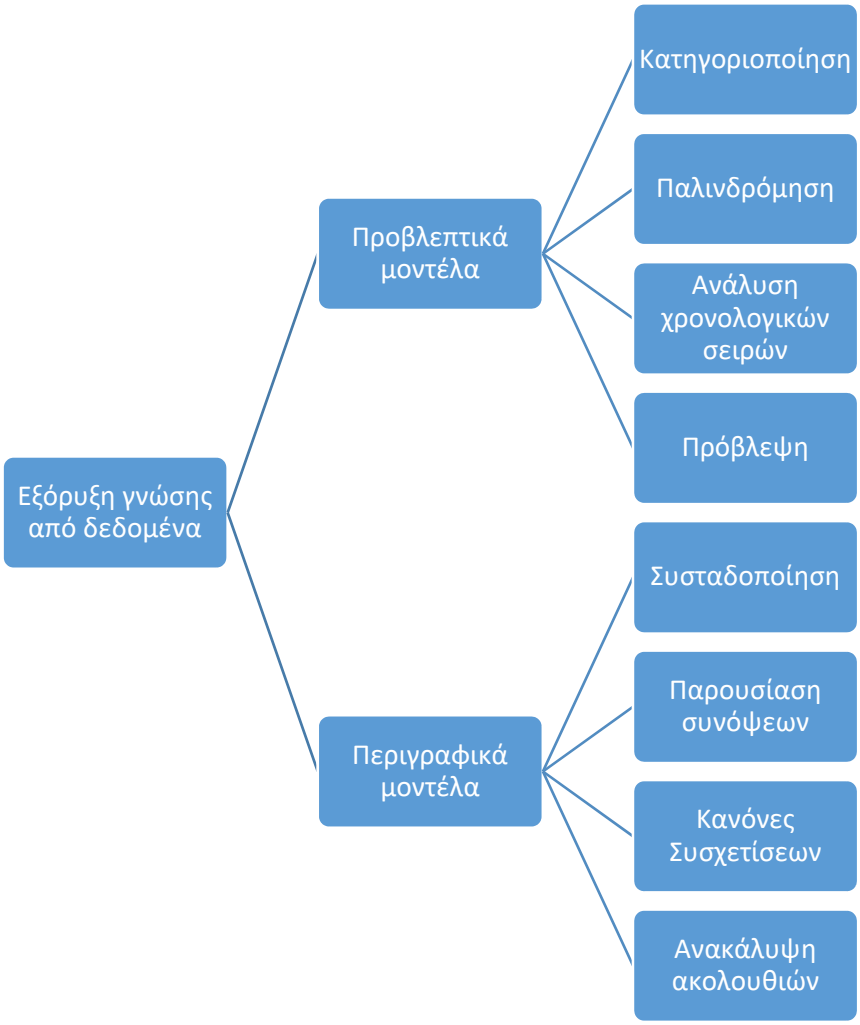
Η εξαγωγή προτύπων από δεδομένα συμβαίνει εδώ και αιώνες. Οι πρώτες μέθοδοι για τον προσδιορισμό προτύπων ήταν αυτές της θεωρίας Bayes και της ανάλυσης της παλινδρόμησης. Ο πολλαπλασιασμός, η ευρεία διαθεσιμότητα και η εξέλιξη της τεχνολογίας υπολογιστών έχουν αυξήσει τον όγκο των συγκεντρωμένων δεδομένων και την ζήτηση για αποδοτικούς και αποτελεσματικούς χειρισμούς. Καθώς οι συλλογές δεδομένων αυξήθηκαν τόσο σε όγκο όσο και σε πολυπλοκότητα, η «χειρωνακτική» ανάλυση των δεδομένων έχει αντικατασταθεί από την αυτόματη επεξεργασία δεδομένων. Σε αυτό συνέβαλαν άλλες ανακαλύψεις της επιστήμης των υπολογιστών, όπως τα νευρωνικά δίκτυα, η συσταδοποίηση, οι γενετικοί αλγόριθμοι (1950), τα δέντρα απόφασης (1960) και η μηχανή υποστήριξης διανυσμάτων (1990). Η εξόρυξη δεδομένων είναι η διαδικασία εφαρμογής αυτών των μεθόδων στα δεδομένα με σκοπό την αποκάλυψη άγνωστων προτύπων σε μεγάλα σύνολα δεδομένων. Αυτό γεφυρώνει το χάσμα της εφαρμοσμένης στατιστικής και της τεχνητής νοημοσύνης (τα οποία συνήθως παρέχουν το μαθηματικό υπόβαθρο) με την διαχείριση βάσης δεδομένων κάνοντας χρήση του τρόπο με τον οποίο αποθηκεύονται και κατατάσσονται στη βάση δεδομένων για να εκτελέσουν την θεωρία και τους διαθέσιμους αλγορίθμους περισσότερο αποτελεσματικά, επιτρέποντας σε τέτοιες μεθόδους να εφαρμόζονται σε μεγάλα σύνολα δεδομένων [6].

Η Εξόρυξη Δεδομένων αποτελεί ένα γεφύρωμα των επιστημονικών κλάδων της στατιστικής, τεχνικής νοημοσύνης, της μηχανικής μάθησης και τέλος των βάσεων δεδομένων. Η «καταγωγή» της ΕΔ έχει τις ρίζες της σε τρεις βασικούς επιστημονικούς κλάδους: της στατιστική (Statistics), της τεχνητής νοημοσύνης (Artificial Intelligence) και της μηχανικής μάθησης (Machine Learning).

Εικόνα 1-2 Σχέση Εξόρυξη Δεδομένων με άλλους επιστημονικούς κλάδους

1.1.2 Κατηγοριοποίηση Μεθόδων Εξόρυξης

Τα μοντέλα που παράγονται από το στάδιο της Εξόρυξης Δεδομένων διακρίνονται σε δυο βασικούς τύπους: τα μοντέλα πρόβλεψης (predictive) και τα περιγραφικά μοντέλα (descriptive).



Γράφημα 1-1 Μοντέλα εξόρυξης γνώσης από δεδομένα

Στόχος ενός προβλεπτικού μοντέλου (predictive model) είναι να προβλέψει τιμές για ένα συγκεκριμένο χαρακτηριστικό που παρουσιάζει ενδιαφέρον και που πιθανώς βασίζεται στη συμπεριφορά άλλων χαρακτηριστικών. Η υλοποίηση πρόβλεψης μπορεί να γίνει με βάση τη χρήση ιστορικών δεδομένων. Οι εργασίες εξόρυξης γνώσης από δεδομένα για το χτίσιμο ενός προβλεπτικού μοντέλου περιλαμβάνουν κατηγοριοποίηση, παλινδρόμηση, ανάλυση χρονολογικών σειρών και πρόβλεψη.

Ένα περιγραφικό μοντέλο (descriptive model) βρίσκει πρότυπα (patterns) ή σχέσεις (relations) που ενυπάρχουν στα δεδομένα και μελετά τις ιδιότητες τους, ώστε να δοθεί μια

αιτιολόγηση της συμπεριφοράς τους. Αντίθετα από το προβλεπτικό, το περιγραφικό μοντέλο λειτουργεί σαν ένα μέσο που διερευνά τις ιδιότητες των δεδομένων που εξετάζονται χωρίς να προβλέπει νέες ιδιότητες. Η συσταδοποίηση, η παρουσίαση συνόψεων, οι κανόνες συσχετίσεων και η ανακάλυψη ακολουθιών συνήθως θεωρούνται σαν περιγραφικές εργασίες από τη φύση τους (Μητσοπούλου, 2017).

Υπάρχει μια μεγάλη ποικιλία μεθόδων εξόρυξης δεδομένων. Ανάλογα με το είδος των δεδομένων και το είδος της γνώσης που εξάγεται, αυτές κατηγοριοποιούνται σε διαφορετική κατηγορία. Μερικές από τις βασικές μεθόδους της Εξόρυξης Δεδομένων παρουσιάζονται στις παρακάτω ενότητες (Βερύκιος, Καγκλής, & Σταυρόπουλος, 2015).

1.1.2.1 Κατηγοριοποίηση - Classification

Πρόκειται για μια προγνωστική μέθοδο που βασίζεται στην εξέταση των χαρακτηριστικών - Attributes ενός νέου αντικειμένου - Object (συνήθως αναπαρίσταται ως ένα διάνυσμα τιμών για τις χαρακτηριστικές του ιδιότητες) το οποίο με βάση αυτές τις τιμές ανατίθεται σε ένα προκαθορισμένο σύνολο κατηγοριών (Classes), χρησιμοποιώντας μεθόδους μάθησης με επίβλεψη (Supervised Learning Methods). Ο αλγόριθμος κατηγοριοποίησης μαθαίνει μέσα από ένα σύνολο εκπαίδευσης (Training Set) όπου όλα τα αντικείμενα είναι ήδη συνδεδεμένα με γνωστές κλάσεις. Στη συνέχεια, χρησιμοποιώντας τη μάθηση αυτή προχωρά στη κατασκευή ενός μοντέλου με βάση το οποίο ταξινομεί αντικείμενα στις κατάλληλες κλάσεις (Αλτιντζής, 2018).

Μερικές από μέτρα για την εκτίμηση της απόδοσης ενός μοντέλου είναι τα ακόλουθα:

$$accuracy\ (\pi\iota\sigma\tau\acute{o}\tau\eta\tau\alpha) = \frac{\#correct}{\#total} \quad \text{Εξίσωση 1}$$

$$precision\ (\alpha\kappa\rho\acute{\iota}\beta\epsilon\iota\alpha) = \frac{TP}{TP+FP} \quad \text{Εξίσωση 2}$$

$$recall\ (\alpha\nu\acute{\alpha}\kappa\lambda\eta\sigma\eta) = \frac{TP}{TP+FN} \quad \text{Εξίσωση 3}$$

1.1.2.2 Παλινδρόμηση - Regression

Μια σχετική διαδικασία με την κατηγοριοποίηση είναι αυτή της παλινδρόμησης (Regression), στόχος της οποίας είναι η μάθηση ή αλλιώς η εκπαίδευση (training) μιας συνάρτησης, η οποία απεικονίζει ένα αντικείμενο σε μία πραγματική μεταβλητή. Πρόκειται για μια, επίσης, προγνωστική μέθοδο. Στόχος είναι με βάση κάποιες ανεξάρτητες μεταβλητές (independent variables) να προβλεφθούν οι τιμές μιας εξαρτημένης μεταβλητής (dependent variable) (Βερύκιος, Καγκλής, & Σταυρόπουλος, 2015).

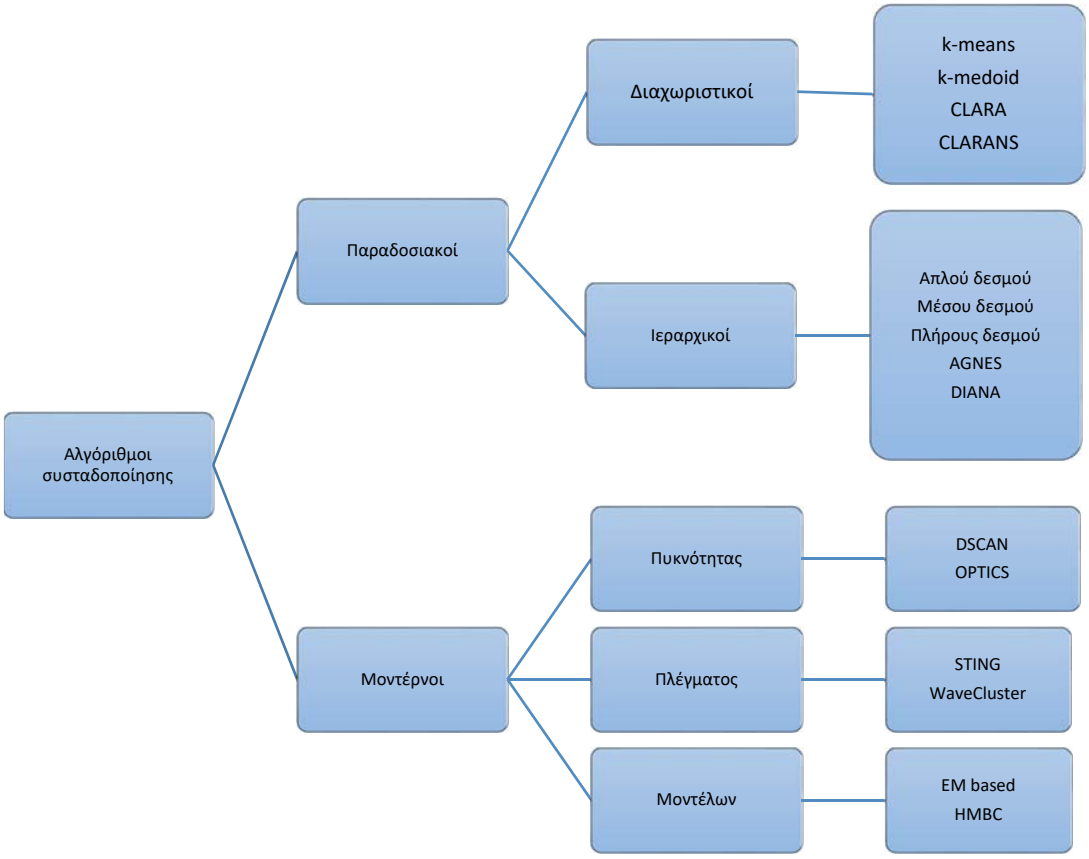
Έχουμε διάφορα είδη παλινδρόμησης, τα κυριότερα των οποίων είναι:

- Η Γραμμική Παλινδρόμηση, όπου το μοντέλο που θα παραχθεί είναι γραμμικό, δηλαδή η εξαρτημένη μεταβλητή είναι ένας γραμμικός συνδυασμός των ανεξαρτήτων μεταβλητών
- και η Λογιστική Παλινδρόμηση, όπου αφορά ένα μη γραμμικό μοντέλο, τα σφάλματα, του οποίου δεν υπακούν στην κανονική κατανομή και η μεταβλητή απόκρισης είναι διακριτή.

1.1.2.3 Συσταδοποίηση - Clustering

Η Συσταδοποίηση (Clustering) είναι μια περιγραφική μέθοδος και είναι η εργασία του καταμερισμού ενός ετερογενούς πληθυσμού σε ένα σύνολο συστάδων (clusters), δηλαδή ομάδων, οι οποίες θα περιέχουν όμοια ή παρεμφερή δείγματα. Οι βαθμοί ομοιότητας ή ανομοιότητας μεταξύ των αντικειμένων δεδομένων υπολογίζονται και αξιολογούνται με τη χρήση ενός μέτρου εγγύτητας που μπορεί να παρέχεται από τον χρήστη ή περιλαμβάνεται φυσικά στη συγκεκριμένη μέθοδο συσταδοποίησης. Δεν υπάρχει προηγούμενη γνώση των ετικετών κλάσης που σχετίζονται με τα αντικείμενα που πρόκειται να ομαδοποιηθούν σε μια εργασία ομαδοποίησης και για το λόγο αυτό, η ομαδοποίηση αναφέρεται μερικές φορές και ως ταξινόμηση χωρίς επίβλεψη, για να τονιστεί η διαφορά από την εργασία ταξινόμησης (με επίβλεψη), στην οποία είναι γνωστές οι ετικέτες κλάσης των αντικειμένων στο σύνολο εκπαίδευσης. Οι κατηγορίες μπορεί να είναι αμοιβαία αποκλειόμενες και εξαντλητικές ή να έχουν μία πιο σύνθετη αναπαράσταση, όπως για παράδειγμα ιεραρχικές και επικαλυπτόμενες (Βερύκιος, Καγκλής, & Σταυρόπουλος, 2015) [5].

Γενικά οι αλγόριθμοι συσταδοποίησης μπορούν να ομαδοποιηθούν στις ακόλουθες πέντε ομάδες:



Γράφημα 1-2 Κατηγορίες Αλγορίθμων συσταδοποίησης

Οι παραδοσιακοί αλγόριθμοι συσταδοποίησης χωρίζονται στις ακόλουθες δύο μεγάλες κατηγορίες:

- **Διαχωριστική Συσταδοποίηση** (Partitional Clustering), όπου πρόκειται για ένα διαμερισμό των αντικειμένων σε μη επικαλυπτόμενα (non-overlapping) υποσύνολα (συστάδες) τέτοιος ώστε κάθε αντικείμενο να ανήκει σε ακριβώς ένα υποσύνολο.
- **Ιεραρχική Συσταδοποίηση** (Hierarchical Clustering), όπου έχουμε ένα σύνολο από εμφωλευμένες (nested) ομάδες και επιτρέπουμε σε μια συστάδα να έχει υποσυστάδες οργανωμένες σε ένα ιεραρχικό δέντρο.

Οι μοντέρνοι αλγόριθμοι συσταδοποίησης μπορούν να χωριστούν στις ακόλουθες τρεις κατηγορίες:

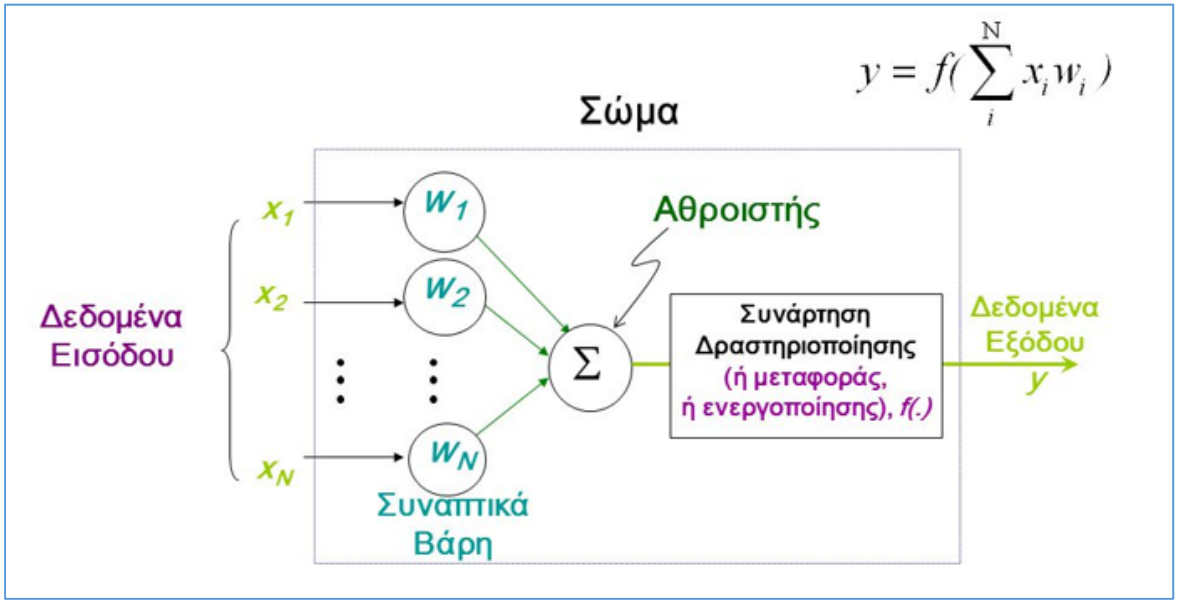
- **Συσταδοποίηση με βάση τη πυκνότητα** (Density-based), η οποία βασίζεται στην συνεκτικότητα και σε συναρτήσεις πυκνότητας των σημείων των δεδομένων, έτσι ώστε οι συστάδες να κατασκευάζονται σύμφωνα με κριτήρια πυκνότητας και συνεκτικότητας.
- **Συσταδοποίηση με βάση το πλέγμα** (Grid-based), όπου γίνεται χρήση μιας πολλαπλών επιπέδων υψηλής ανάλυσης κοκκοποιημένης δομής για να αναλυθούν οι συστάδες. Σε κάθε επίπεδο η ανάλυση μεγαλώνει.
- **Συσταδοποίηση με βάση τα μοντέλα** (Model-based), στην οποία υποθέτουμε ένα μοντέλο για κάθε συστάδα και βρίσκουμε το καλύτερο ταίριασμα των δεδομένων σε αυτό διανέμοντας κάθε σημείο δεδομένου στη συστάδα στην οποία αναμένεται να έχει τη μεγαλύτερη πιθανότητα να ανήκει. Η εκτίμηση γίνεται μέσω της μέγιστης πιθανοφάνειας (maximum likelihood).

1.1.2.4 Αυτοκωδικοποιητές - Autoencoders

Ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ) είναι ένα μαθηματικό μοντέλο επεξεργασίας πληροφορίας του οποίου η λειτουργία είναι εμπνευσμένη από τον τρόπο με τον οποίο βιολογικοί νευρώνες, όπως αυτοί του ανθρώπινου εγκεφάλου, επεξεργάζονται την πληροφορία. Συγκεκριμένα, είναι μια αρχιτεκτονική δομή (δικτύου) αποτελούμενη από ένα πλήθος διασυνδεδεμένων μονάδων επεξεργασίας (τεχνητοί νευρώνες). Κάθε μονάδα επεξεργασίας χαρακτηρίζεται από εισόδους και εξόδους. Υλοποιεί τοπικά έναν απλό υπολογισμό με βάση τις εισόδους που δέχεται και μεταδίδει το αποτέλεσμα (έξοδος) σε άλλες μονάδες επεξεργασίας με τις οποίες συνδέεται.

Τα τρία βασικά στοιχεία ενός τεχνητού νευρών, όπως φαίνονται και στην παρακάτω εικόνα, είναι:

- Ένα σύνολο από συνάψεις ή συνδετικούς κρίκους με άλλους νευρώνες.
- Ένας αθροιστής συνάψεων.
- Μια συνάρτηση ενεργοποίησης, μέσω της οποίας βγαίνει η έξοδος του νευρώνα. Οι τρεις ποιο κοινές συναρτήσεις ενεργοποίησης είναι η σιγμοειδείς (sig), η υπερβολική εφαπτομένη (tanh) και η συνάρτηση διορθωμένης γραμμικής μονάδας (relu).



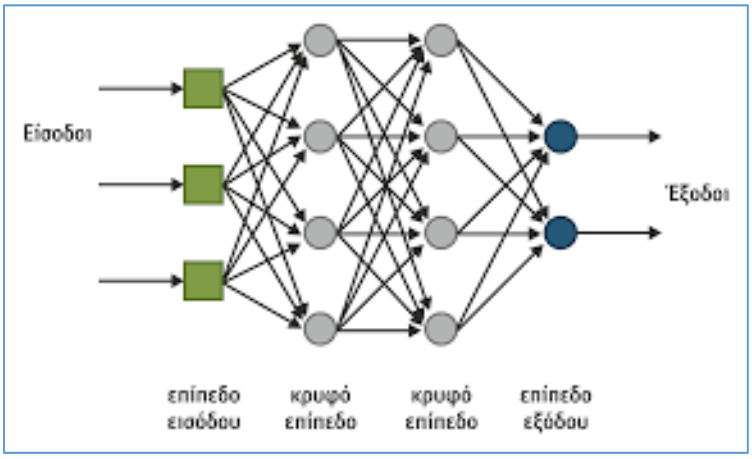
Εικόνα 1-3 Μοντέλο Τεχνητού Νευρώνα

Κάθε σύνδεση μεταξύ δύο μονάδων χαρακτηρίζεται από μια τιμή βάρους. Οι τιμές των βαρών των συνδέσεων αποτελούν τη γνώση που είναι αποθηκευμένη στο ΤΝΔ και καθορίζουν τη λειτουργικότητά του. Ένας τεχνητός νευρώνας που δεν είναι είσοδος ή έξοδος στο δίκτυο, καλείτε κρυφός. Οι κρυφοί τεχνητοί νευρώνες δεν έρχονται άμεσα σε επαφή με τις εισόδους και εξόδους του δικτύου και για αυτό δεν μπορούν να επηρεαστούν άμεσα από αυτούς. Κρυφό επίπεδο ενός ΤΝΔ καλούμε τώρα την συλλογή κρυφών νευρώνων οι οποίοι δεν συνδέονται μεταξύ τους. Όσο περισσότερα κρυφά πεδία έχει ένα ΤΝΔ τόσο περισσότερο βελτιώνεται.

Ένα ΤΝΔ αναπτύσσει μια συνολική λειτουργικότητα μέσω μιας μορφής εκπαίδευσης (μάθησης) και αποτελούν τεχνητά συστήματα μάθησης τα οποία εκπαιδεύονται από δεδομένα (παραδείγματα) για να μάθουν να υλοποιούν κάποια συγκεκριμένη λειτουργία. Το γεγονός αυτό τα καθιστά πολύ ισχυρά εργαλεία για την επίλυση ιδιαίτερα σύνθετων και δύσκολων προβλημάτων (Shukla, 2017) (Bishop, 1995).

Βασική ιδιότητα ενός ΤΝΔ είναι να μαθαίνει από το περιβάλλον του και να βελτιώνεται μέσου της εκπαίδευσης. Με τον όρο εκπαίδευση ενός ΤΝΔ ορίζουμε μια διαδικασία κατά την οποία οι ελεύθεροι παράμετροι ενός ΤΝΔ προσαρμόζονται μέσω μιας διαδικασίας διέγερσης από το περιβάλλον, την οποία το δίκτυο ενσωματώνει. Ο τύπος της εκπαίδευσης

είναι καθορισμένος από τον τρόπο με τον οποίο οι αλλαγές των παραμέτρων πραγματοποιούνται (Haykin, 1999).



Εικόνα 1-4 Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ)

Η εκπαίδευση ενός ΤΝΔ ακολουθεί τα παρακάτω βήματα:

- Το ΤΝΔ διεγείρεται από το περιβάλλον.
- Οι νευρώνες εισόδου ενεργοποιούνται και υπολογίζεται μια συνάρτηση από τα δεδομένα εισόδου. Η τιμή της συνάρτησης αυτής συγκρίνεται με την τιμή κατωφλίου του νευρώνα. Εάν η τιμή είναι μεγαλύτερη από την τιμή του κατωφλίου, τότε ο νευρώνας υπολογίζει την έξοδο την οποία προωθεί ως είσοδο στους επόμενους νευρώνες.
- Το μόνο που αλλάζει κατά την διάρκεια της εκπαίδευσης είναι η τιμές των βαρών των συνδέσεων των νευρώνων. Οι αλλαγές στις τιμές εξαρτάται από την μέθοδο εκπαίδευσης που χρησιμοποιούμε. Τρεις είναι οι βασικότεροι τρόποι με τους οποίους αλλάζουν τα βάρη: με εποπτευόμενο τρόπο, με μη-εποπτευόμενο τρόπο και τέλος με αυτό-εποπτευόμενο τρόπο.

Οι **Αυτοκωδικοποιητές (Autoencoders)** είναι νευρωνικά δίκτυα που εκπαιδεύονται με στόχο να αντιγράψουν την είσοδο στην έξοδο με τέτοιο τρόπο ώστε να κωδικοποιούν (encoding) χρήσιμες ιδιότητες των δεδομένων.

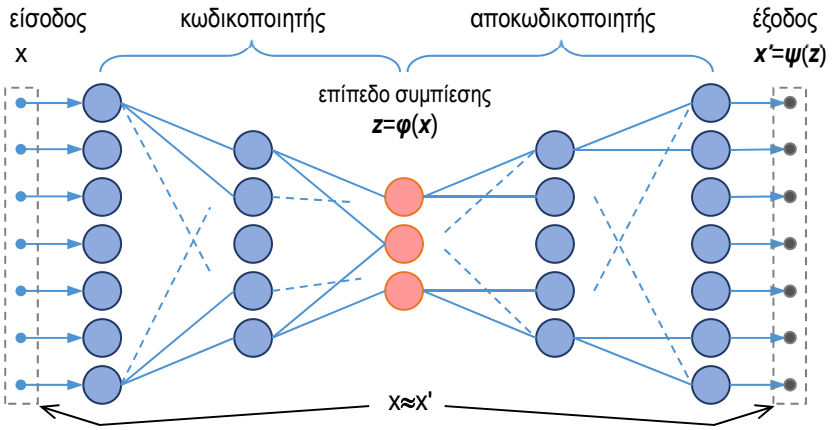
Σκοπός τους είναι να κωδικοποιήσουν αυτά τα δεδομένα σε ένα μικρότερο διάνυσμα χαρακτηριστικών (feature vector) και έπειτα με βάση αυτό να ανακατασκευάσουν την

αρχική είσοδο, μέσω ενός δεύτερου ΤΝΔ που επιτελεί την αποκωδικοποίηση του (decoding).

Η κύρια χρήση των αυτοκωδικοποιητών είναι σε προβλήματα **μείωσης διαστατικότητας** (dimensionality reduction) και **εξαγωγής χαρακτηριστικών** (feature extraction).

Η κλασική δομή αυτοκωδικοποιητή περιλαμβάνει δύο βασικά επιμέρους δίκτυα:

1. ένα δίκτυο κωδικοποίησης (encoder), το οποίο παίρνει τις μεταβλητές εισόδου και τις ενσωματώνει σε έναν code embedding (κωδικοποιημένη εμβύθιση) z στο επίπεδο συμπίεσης (bottleneck), $z = \varphi(x)$ Εξίσωση 4.
2. ένα δίκτυο αποκωδικοποίησης (decoder) που ως είσοδο παίρνει το κωδικοποιημένο (code) διάνυσμα z και το προβάλλει ξανά στο χώρο διαστάσεων της εισόδου. Αποτελεί κατά κάποιο τρόπο αντιστροφή της προηγούμενης διαδικασίας κωδικοποίησης, $x' = \psi(z)$ Εξίσωση 5. Ο στόχος εδώ είναι η x' να είναι όσο το δυνατό πιο παρόμοια με την x (Βαϊόπουλος, 2021).



Εικόνα 1-5 Διάγραμμα αυτοκωδικοποιητή

Το κύριο χαρακτηριστικό των αυτοκωδικοποιητών είναι το ενδιάμεσο επίπεδο στένωσης, οι διαστάσεις του οποίου πρέπει να είναι λιγότερες από αυτές τις εισόδου, ενώ η έξοδος έχει διαστάσεις που ταιριάζουν με τις διαστάσεις της εισόδου. Στην ουσία αυτά τα ΤΝΔ πρέπει να εκπαιδευτούν ώστε η έξοδος να μοιάζει όσο το δυνατόν περισσότερο στην είσοδο.

1.1.3 Μηχανική Μάθηση

Όπως είδαμε στην προηγούμενη μας ενότητα ο κύριος στόχος των διάφορων μεθόδων εξόρυξης είναι η δημιουργία ενός μοντέλου. Η δημιουργία των μοντέλων αυτών από ένα σύνολο δεδομένων και με την χρήση υπολογιστικού συστήματος ονομάζεται Μηχανική Μάθηση (Machine Learning). Έχουν αναπτυχθεί πολλές τεχνικές μηχανικής μάθησης που χρησιμοποιούνται ανάλογα με τη φύση του προβλήματος και εμπίπτουν σε ένα από τα παρακάτω τρία είδη:

- **μάθηση με επίβλεψη** ή εποπτευόμενη μάθησης (supervised learning)
- **μάθηση χωρίς επίβλεψη** ή μη εποπτευόμενη μάθηση (unsupervised learning)
- **μάθηση με ενίσχυση** ή ενισχυτική μάθηση (reinforcement learning)

Στη *μάθηση με επίβλεψη* τα στοιχεία του συνόλου των παραδειγμάτων είναι ζεύγη (όπου τα δεδομένα εισόδου απεικονίζονται σε γνωστές επιθυμητές εξόδους) και υλοποιεί συσχετίσεις εισόδου-εξόδου με απώτερο στόχο να εκπαιδευτεί ώστε όταν κάποιο δεδομένο εμφανίζεται ως είσοδος να μπορεί να παρέχει έξοδο την αντίστοιχη επιθυμητή τιμή. Στα προβλήματα μάθησης με επίβλεψη ανήκουν τα προβλήματα της ταξινόμησης ή κατηγοριοποίησης (classification) και τα προβλήματα συναρτησιακής προσέγγισης (regression ή function approximation).

Στη *μάθηση χωρίς επίβλεψη* τα δεδομένα εκπαίδευσης δεν περιλαμβάνουν την επιθυμητή έξοδο αλλά μόνο τα δεδομένα εισόδου. Σε αυτή την περίπτωση στόχος της εκπαίδευσης δεν είναι η εύρεση της απεικόνισης εισόδου – εξόδου, αλλά η εξαγωγή κάποιων βασικών δομικών ιδιοτήτων των δεδομένων του συνόλου εκπαίδευσης (π.χ. εύρεση ομάδων). Στην κατηγορία των προβλημάτων μάθησης χωρίς επίβλεψη ανήκουν τα ακόλουθα προβλήματα:

- Ομαδοποίηση (clustering)
- Εκτίμηση πυκνότητας πιθανότητας (probability density function estimation)
- Μείωσης της διάστασης των δεδομένων (dimensionality reduction)

Τέλος η *μάθηση με ενίσχυση* είναι μια παραλλαγή της μάθησης με επίβλεψη, κατά την οποία παρέχεται λιγότερη πληροφορία στο σύστημα μάθησης από τον εξωτερικό επιβλέποντα. Η διαφορά έγκειται στο ότι στο σύστημα μάθησης δεν παρέχεται η επιθυμητή έξοδος για κάθε είσοδο, αλλά μόνο η τιμή μιας ποσότητας (σήμα ενίσχυσης – reinforcement signal), η οποία δηλώνει εάν το σύστημα παρέχει σωστή ή λάθος απόκρισης χωρίς όμως να δίνει λεπτομέρειες για το ποια είναι η σωστής απόκριση (Γεωργούλη, 2015).

1.2 Tensorflow

Η Μηχανική Μάθηση ή Machine Learning είναι μια τεχνολογία υπολογισμού που εξελίσσεται συνεχώς τα τελευταία χρόνια. Η συγκεκριμένη τεχνολογία χρησιμοποιείται παντού γύρω μας, από την οδήγηση αυτοκινήτων μέχρι την πρόβλεψη της χρηματιστηριακής αγοράς. Το Tensorflow είναι ένα πρόγραμμα της Google που βασίζεται στη μηχανική μάθηση και τα νευρωνικά δίκτυα. Στις παρακάτω ενότητες θα δούμε τι είναι ακριβώς και πως μπορεί να χρησιμοποιηθεί στην Εξόρυξη Δεδομένων.

1.2.1 Ιστορική Εξέλιξη

Το Tensorflow είναι μια βιβλιοθήκη λογισμικού ανοιχτού κώδικα που χρησιμοποιείται για αριθμητικούς υπολογισμούς με την χρήση γράφους ροής δεδομένων (data flow graphs). Αναπτύχθηκε από την ομάδα Google Brain μέσα στα πλαίσια της έρευνας της Google, Machine Intelligence για την μηχανική μάθηση και την ανάλυση για τα «βαθιά νευρωνικά δίκτυα» (deep neural networks). Παρά την αρχική αιτία δημιουργίας του, το σύστημα πλέον είναι αρκετά γενικευμένο για να εφαρμοστεί σε μια πληθώρα άλλων χρήσεων. Το Φεβρουάριο του 2017 είχαμε την έκδοση 1.0 ενώ από το Σεπτέμβριο του 2019 έγινε επισήμως διαθέσιμη και η έκδοση 2.0. Στην πτυχιακή αυτή εργασίας θα χρησιμοποιήσουμε την έκδοση 1.0.

1.2.2 Βασικά Στοιχεία του Tensorflow

Βασικά συστατικά του Tensorflow αποτελούν οι Τανυστές (Tensors), από όπου πήρε και την ονομασία του, και οι Γράφοι. Οι Τανυστές είναι γεωμετρικά αντικείμενα που μπορούν να θεωρηθούν ως γενικευμένα διανύσματα, είναι δηλαδή στην ουσία πολυδιάστατοι πίνακες. Έτσι για παράδειγμα, ένα βαθμωτό μέγεθος είναι ένα τανυστείς $0^{ης}$ τάξης, ενώ ένα διάνυσμα είναι ένας τανυστείς $1^{ης}$ τάξης με 3 συνιστώσες.

Στο Tensorflow οι τανυστές προσδιορίζονται από τρία βασικά χαρακτηριστικά:

1. Βαθμό (rank): περιγράφει τη διάσταση κάθε τανυστή. Ένας δισδιάστατος πίνακας αποτελεί έναν tensor με βαθμό 2 ενώ ένα διάνυσμα έναν tensor με βαθμό 1.
2. Σχήμα (shape): περιγράφει το μέγεθος της κάθε διάτασης του τανυστή (σε ένα δισδιάστατο πίνακα περιγράφει τον αριθμό των γραμμών και στηλών).

3. Τύπο (type): περιγράφει τον τύπο των δεδομένων (int32, float κλπ.) του τανυστή.

Οι τύποι των tensors που χρησιμοποιούνται στο TensorFlow είναι οι ακόλουθοι:

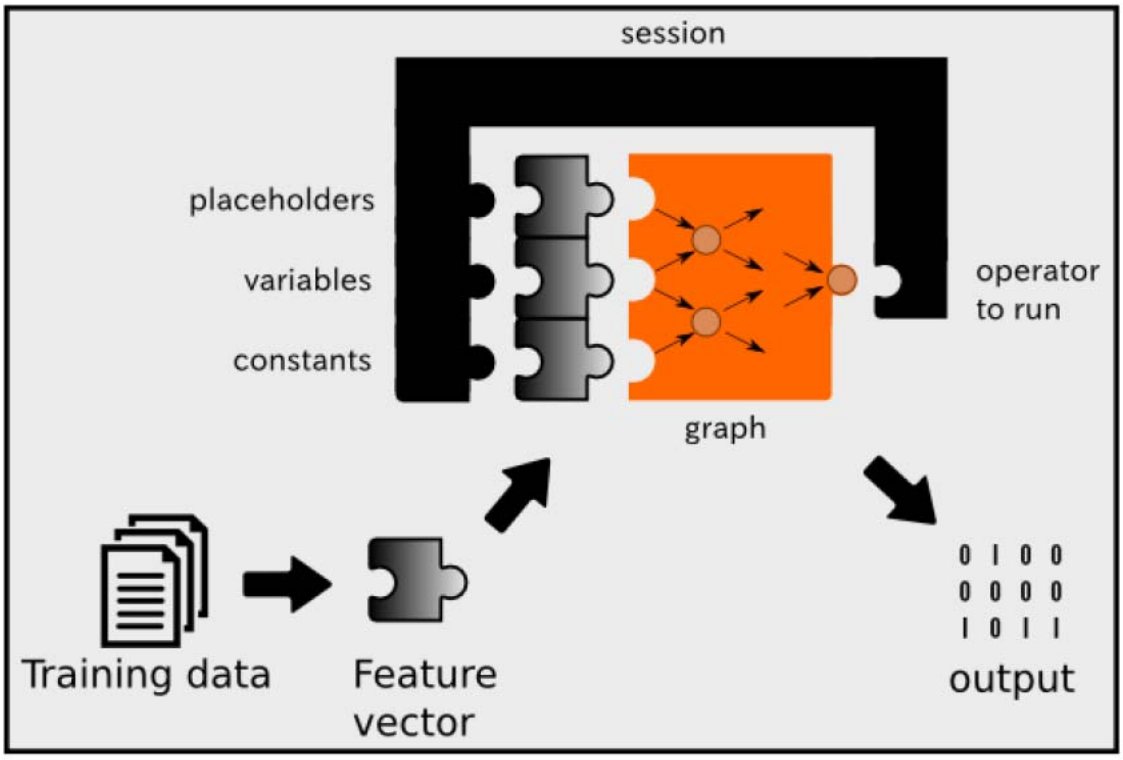
- **Constant**, πρόκειται για σταθερούς τανυστές που οι τιμές τους δεν αλλάζουν και δημιουργούνται με την κλάση tf.constant.
- **Variable**, δημιουργούνται με την κλάση tf.Variable και πρόκειται για τανυστές οι τιμές των οποίων μπορεί να αλλάξουν με την χρήση κάποιου operation κατά το στάδιο της εκτέλεσης (session).
- **Placeholder**, πρόκειται για τανυστές που δημιουργούνται με την κλάση tf.placeholder και αρχικά δεν έχουν τιμές αλλά τροφοδοτούνται με τιμές κατά το στάδιο της εκτέλεσης (session).

Η αρχιτεκτονική Tensorflow λειτουργεί σε τρία μέρη:

- Προ-επεξεργασία των δεδομένων
- Δημιουργία του μοντέλου
- Εκπαίδευση και αξιολόγηση του μοντέλου

Ένα μοντέλο Tensorflow, παίρνει ως είσοδο τους εν λόγω τανυστές, οι οποίοι εισέρχονται στο ένα άκρο και στη συνέχεια «ρέουν» στο σύστημα που αποτελείται από πολλαπλές μαθηματικές λειτουργίες και εν τέλει «εξέρχονται» από το άλλο άκρο ως έξοδος. Από την προαναφερθείσα λειτουργία, λοιπόν, προκύπτει η ονομασία του Tensorflow, καθώς ένας τανυστής (tensor) εισέρχεται σε αυτό, ρέει (flows) μέσω μιας σειράς λειτουργιών και στη συνέχεια εξέρχεται από την άλλη πλευρά.

Πιο αναλυτικά, το Tensorflow χρησιμοποιεί μια δομή που είναι γνωστή ως γράφος ροής δεδομένων (Data flow graph) όπου οι κύριες μονάδες είναι οι κόμβοι και οι ακμές. Οι κόμβοι (Nodes) σε έναν γράφο αναπαριστούν μαθηματικές λειτουργίες (πολλαπλασιασμός, άθροισμα, διαίρεση κλπ.) ενώ οι ακμές (Edges) αναπαριστούν τα δεδομένα που είναι συνήθως πολυδιάστατοι πίνακες ή τανυστές (tensors) οι οποίοι επικοινωνούν μεταξύ τους. Το Tensorflow χρησιμοποιεί ένα γράφο ροής δεδομένων για να αναπαραστήσει όλους τους υπολογισμούς σε έναν αλγόριθμο μηχανικής εκμάθησης, συμπεριλαμβανομένων των μεμονωμένων μαθηματικών λειτουργιών, των παραμέτρων και των κανόνων ενημέρωσής τους καθώς και της προεπεξεργασία που υφίστανται μια είσοδος (input) (Shukla, 2017).



Εικόνα 1-6 Αρχιτεκτονική Tensorflow (Shukla, 2017)

Ο γράφος έχει μια πληθώρα πλεονεκτημάτων (Μουμουλίδη & Τζούμα, 2018):

- Είναι φορητός, εφόσον μπορεί είτε να εκτελεστεί άμεσα είτε να αποθηκευτεί για μεταγενέστερη χρήση, ενώ μπορεί να λειτουργήσει σε πολλές πλατφόρμες: CPUs, GPUs, TPUs, κινητές και ενσωματωμένες συσκευές. Επίσης, μπορεί να χρησιμοποιηθεί σε στάδια παραγωγής, χωρίς καμία απαίτηση χρήσης του κώδικα που έφτιαξε τον γράφο, απαιτείται απλώς το κομμάτι εκτέλεσης για να χρησιμοποιηθεί.
- Είναι μεταβλητός και βελτιστοποιήσιμος, αφού μπορεί να μεταβληθεί κατάλληλα για να «τρέξει» σε μια διαφορετική πλατφόρμα. Επίσης μπορούν να γίνουν βελτιστοποιήσεις μνήμης και υπολογιστικής δύναμης για να χρησιμοποιηθεί ο ίδιος γράφος σε διαφορετικές πλατφόρμες και συνδυασμούς αυτών. Αυτό είναι χρήσιμο για παράδειγμα στην περίπτωση που χρησιμοποιούμε ένα ή περισσότερα ισχυρά μηχανήματα για να εκπαιδεύσουμε το μοντέλο και αρκετά χαμηλότερης ισχύος αλλά κινητές πλέον συσκευές για να κάνουν τις προβλέψεις.

- Υποστηρίζουν κατανεμημένη εκτέλεση. Τα υψηλού επιπέδου APIs του Tensorflow σε συνδυασμό με τους υπολογιστικούς γράφους, δημιουργούν ένα πλούσιο και ευέλικτο περιβάλλον ανάπτυξης και ισχυρές δυνατότητες παραγωγής στο ίδιο πλαίσιο.

1.2.3 Tensorflow 1.0 vs 2.0

Οι κυριότερες αλλαγές της έκδοσης TensorFlow 2.0 από την 1.0 είναι οι ακόλουθες:

- **Eager Execution (Πρόωρη εκτέλεση)**

Η επιλογή της πρόωρης εκτέλεσης (Eager execution) αφορά μια λειτουργία προγραμματισμού που αξιολογεί τις συναρτήσεις και τις εκτελεί αμέσως, χωρίς να δημιουργεί γράφους. Οι συναρτήσεις επιστρέφουν συγκεκριμένες τιμές αντί να κατασκευάσουν υπολογιστικούς γράφους για να τρέξουν αργότερα. Αυτό καθιστά εύκολο το ξεκίνημα με το Tensorflow και τα μοντέλα εντοπισμού σφαλμάτων (debugging), ενώ μειώνει και τις άσκοπες επαναλήψεις κώδικα (boilerplate). Οι πιο βασικοί λόγοι χρήσεις της πρόωρης εκτέλεσης είναι:

  - Γρηγορότερος έλεγχος κώδικα και του γράφου
  - Χρήση του ελέγχου ροής της Python μέσα από τα Tensorflow APIs (επαναλήψεις, συνθήκες, συναρτήσεις κτλ)
  - Πιο άμεση αποσφαλμάτωση
  - Η σημασία της εκχώρησης κατά την εκτέλεση (της πρόωρης εκτέλεσης) καθιστά εύκολη την κατασκευή δυναμικών γράφων

Όταν ο προγραμματιστής είναι ικανοποιημένος από την πρόωρη εκτέλεση, μπορεί να μετατρέψει αυτομάτως τον κώδικά του σε γράφο για να αποθηκευτεί και να χρησιμοποιηθεί σε άλλη πλατφόρμα ή σε μεταγενέστερη εκτέλεση.

- **Οι functions αντικαθιστούν τα sessions**
- **API cleanup και χρήση ως high-level API το tf.keras**

2 Πρακτικές Εφαρμογές

Σε αυτή την ενότητα θα εφαρμόσουμε τις μεθόδους εξόρυξης δεδομένων που αναφέραμε στην προηγούμενη ενότητα με χρήση Tensorflow και αξιοποιώντας μεθόδους και τεχνικές από το βιβλίο Machine Learning with TensorFlow (Shukla, 2017). Ως τελικό αποτέλεσμα της όλης διαδικασίας θα είναι η κατάρτιση μίας εργαλειοθήκη (toolbox), ο συνολικός κώδικας της οποίας παρατίθεται στο Παράρτημα Δ. Επίσης θα χρησιμοποιηθούν βοηθητικές συναρτήσεις και βιβλιοθήκες της Python, οι οποίες αναφέρονται αναλυτικά στο Παράρτημα Β.

Η εφαρμογή των συναρτήσεων σε διάφορα παραδείγματα (τα διάφορα Datasets που θα χρησιμοποιηθούν αναφέρονται αναλυτικά στο Παράρτημα Γ) θα γίνει στο Google Collab ή αλλιώς Συνεργασία. Πρόκειται για μια υπηρεσία cloud, που προσφέρεται από την Google δωρεάν και βασίζεται στο περιβάλλον Notebook Jupyter και προορίζεται για εκπαίδευση και έρευνα μηχανικής μάθησης. Αυτή η πλατφόρμα επιτρέπει την εκπαίδευση μοντέλων μηχανικής μάθησης απευθείας στο cloud και δεν είναι απαραίτητο να γίνει εγκατάσταση στον υπολογιστή.

2.1 Εφαρμογή Tensorflow σε Παλινδρόμηση (Regression)

2.1.1 Γενικά

Στη ενότητα αυτή θα ασχοληθούμε με την Μέθοδο Εξόρυξης Δεδομένων της Παλινδρόμησης (Regression). Συγκεκριμένα θα ασχοληθούμε με την δημιουργία ενός γραμμικού και ενός πολυωνυμικού μοντέλου. Ενώ τέλος θα δούμε και τον τρόπο υπολογισμού παραμέτρου εξομάλυνσης για το πολυωνομικό μοντέλο.

2.1.2 Γραμμικό μοντέλο

Όπως είπαμε και στην ενότητα 1.1.2.2 η μέθοδος της παλινδρόμησης προσπαθεί να πρόβλεψη μελλοντικές τιμές βασιζόμενοι σε προηγούμενες τιμές. Η γραμμική παλινδρόμηση είναι το πιο ευρέως διαδεδομένο μοντέλο που χρησιμοποιείται σε προβλήματα παλινδρόμησης, και το οποίο, στην πιο απλή του εκδοχή, μοντελοποιεί τη γραμμική σχέση μεταξύ δύο μεταβλητών, της εξαρτημένης μεταβλητής Y και της ανεξάρτητης X:

$$Y = w * X + b \qquad \text{Εξίσωση 6}$$

όπου w = weight (βάρος) και b = bias (μεροληψίας).

Η συνάρτηση που θα υλοποιεί το γραμμικό μοντέλο της μέθοδο της γραμμικής παλινδρόμησης είναι η ακόλουθη:

linear_regression(split, ratio, x, y, x_test, y_test, learning_rate, training_epochs)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **split**: 1 σε περίπτωση που κάνουμε διαχωρισμό των δεδομένων εισόδου σε training και σε test data, διαφορετικά 0.
- **ratio**: σε περίπτωση που το split έχει την τιμή 1 καταχωρούμε την αναλογία που θα έχουν τα training data σε σχέση με το σύνολο των δεδομένων.
- **x**: τα δεδομένα για την μεταβλητή X του μοντέλου.
- **y**: τα δεδομένα για την μεταβλητή Y του μοντέλου.
- **x_test**: εφόσον δεν έχουμε επιλέξει την λειτουργικότητα του διαχωρισμού, στη μεταβλητή αυτή καταχωρούμε τα δεδομένα της μεταβλητής X του μοντέλου που θα χρησιμοποιηθούν ως test data για την αξιολόγηση του μοντέλου.
- **y_test:** εφόσον δεν έχουμε επιλέξει την λειτουργικότητα του διαχωρισμού, στη μεταβλητή αυτή καταχωρούμε τα δεδομένα της μεταβλητής Y του μοντέλου που θα χρησιμοποιηθούν ως test data για την αξιολόγηση του μοντέλου.
- **learning_rate**: ρυθμός εκμάθησης του μοντέλου. Όσο ποιο μεγάλο είναι το learning rate, τόσο πιο γρήγορα ανανεώνονται τα βάρη του μοντέλου αλλά είναι πιθανό να χαθεί η ελάχιστη τιμή της συνάρτησης κόστους. Αντίθετα μικρές τιμές του ρυθμού εκπαίδευσης καθυστερούν πολύ την εκπαίδευση.
- **training_epochs**: σύνολο εποχών εκμάθησης του μοντέλου

ενώ οι παράμετροι εξόδου είναι οι ακόλουθοι:

- **weight**: με την τιμή της παραμέτρου w του μοντέλου.
- **bias**: με την τιμή της παραμέτρου b του μοντέλου.
- **cost_test**: με την τιμή του συνολικού κόστους, $MSE = \frac{\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2}{n}$

Η συνάρτηση αυτή θα δέχεται τα δεδομένα X και Y και θα υπολογίζει το γραμμικό μοντέλο της Εξίσωση 6. Τα δεδομένα αυτά θα μπορούν να γίνονται split σε training και testing data ή θα δίδονται ξεχωριστά ως δεδομένα εισόδου. Η συνάρτηση θα επιστρέφει τις

παραμέτρους του μοντέλου και πόσο καλή προσαρμογή έχει γίνει (MSE). Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά ανάλογα με την τιμή της μεταβλητής split έχουμε το διαχωρισμό των δεδομένων μας σε training και test data με χρήση της βοηθητική συνάρτησης **split_dataset(x, y, ratio)** (η οποία δίδεται στο παράρτημα Α).

```
# Linear regression
def linear_regression(split, ratio, x, y, x_test, y_test, learning_rate,
training_epochs):

    if split == 1:
        x_train, x_test, y_train, y_test = split_dataset(x, y, ratio)
        n = x_train.size
        n_test = x_test.size
    else:
        x_train = x
        y_train = y
        n = x.size
        n_test = x_test.size
```

Στην συνέχεια ορίζουμε τις μεταβλητές και το μοντέλο μας.

```
    X = tf.placeholder(tf.float32)
    Y = tf.placeholder(tf.float32)

    w = tf.Variable(np.random.randn(), name = "weights")
    b = tf.Variable(np.random.randn(), name = "bias")

    def model(X, w, b):
        return tf.add(tf.multiply(X, w), b)

    y_model = model(X, w, b)
```

Στην συνέχεια ορίζουμε την συνάρτηση κόστους (το οποίο είναι το Μέσο Άθροισμα Τετραγώνων του Σφάλματος - MSE) και το αλγόριθμο Gradient Descent ως μέθοδο της εκμάθησης του μοντέλου μας.

```
    # Mean Squared Error Cost Function
    cost = tf.reduce_sum(tf.square(Y-y_model)) / (n)
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για την εκμάθηση του μοντέλου μας, όπου ανά 50 εποχές εκμάθησής εκτυπώνουμε τις παραμέτρους του μοντέλου και το training cost που έχει υπολογισθεί μέχρι εκείνη την στιγμή.

```
    # Starting the Tensorflow Session
    with tf.Session() as sess:
        init = tf.global_variables_initializer()
        sess.run(init)

    # Iterating through all the epochs
        for epoch in range(training_epochs):
            # Feeding each data point into the optimizer using Feed Dictionary
            for (x, y) in zip(x_train, y_train):
                sess.run(train_op, feed_dict={X: x, Y: y})
            # Displaying the result after every 50 epochs
            if (epoch + 1) % 50 == 0:
                # Calculating the cost a every epoch
                c = sess.run(cost, feed_dict = {X : x, Y : y})
                print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(w),
"b =", sess.run(b))
        # Storing necessary values to be used outside the Session
        training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
        weight = sess.run(w)
        bias = sess.run(b)
    sess.close()
```

Αφού τελειώσει το Session και έχει υπολογιστεί το μοντέλο μας, εκτυπώνουμε τις μεταβλητές τους (W και b) και το τελικό training cost.

```
    # Print Weight and bias of the calculated model and cost from training data
    print("-------------------------------------------------------", '\n')
    print("Weight =", weight, "bias =", bias, '\n')
    print("Training cost =", training_cost, '\n')
```

Στη συνέχεια υπολογίζουμε τα prediction data τόσο για τα training data όσο και για τα testing data. Ενώ υπολογίζουμε και το συνολικό cost για τα testing data.

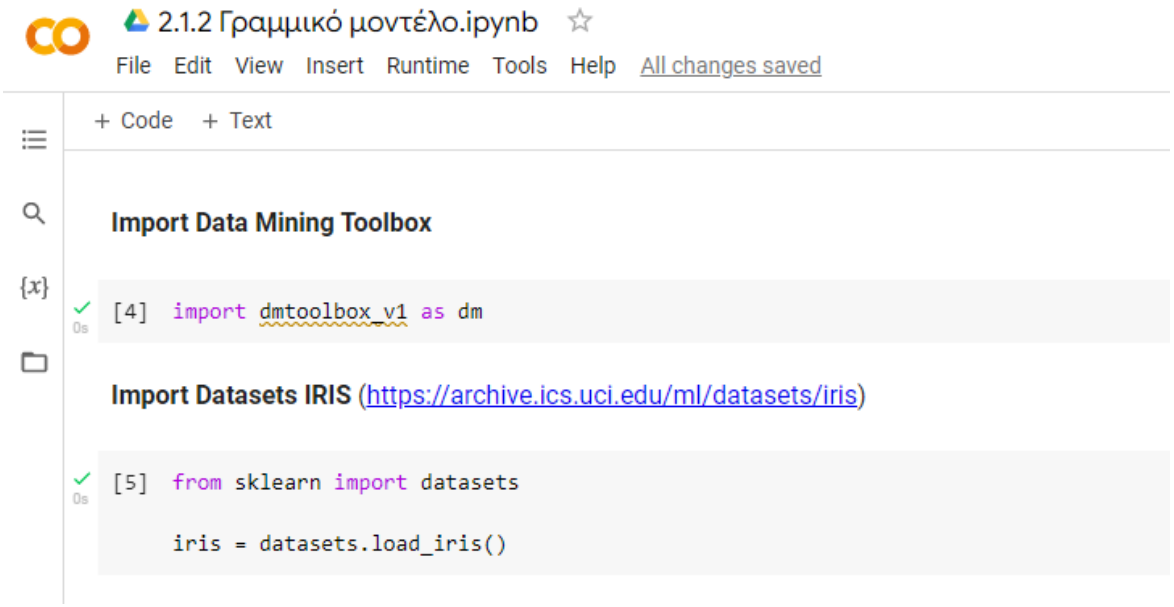
```
    # Calculating the predictions
    predictions = weight * x_train + bias
    # Calculating the predictions for test data
    predictions_test = weight * x_test + bias
    diff = y_test - predictions_test
    cost_test = 0.0
    for (z) in range(n_test):
```

```
    cost_test += math.pow(diff[z], 2)
cost_test = math.sqrt(cost_test)  / (n_test)
# Print cost from test data
print("Test cost =", cost_test, '\n')
```

Τέλος δημιουργούμε μια γραφική παράσταση με τα training, testing και prediction data και επιστρέφουμε τις τιμές του μοντέλου μας, δηλαδή τις μεταβλητές W και b και το συνολικό cost για τα testing data.

```
# Plotting the Results
plt.plot(x_train, y_train, 'ro', label ='Train data')
plt.plot(x_test, y_test, 'gx', label ='Test data')
plt.plot(x_train, predictions, label ='Fitted line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
return weight, bias, cost_test
```

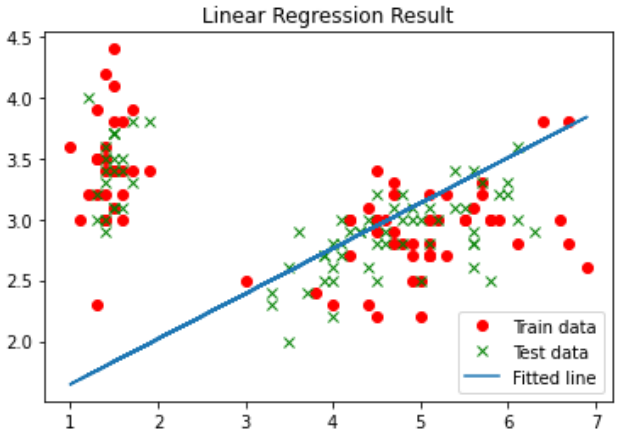
Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.1.2 Γραμμικό μοντέλο.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris και The California housing (Παράρτημα Γ).



Linear Regression

```
import numpy as np

X = np.transpose(iris.data[0:150, 2:3])
X = X.flatten()  # Return a copy of the array collapsed into one dimension.
Y = np.transpose(iris.data[0:150, 1:2])
Y = Y.flatten()
weight, bias, cost = dm.linear_regression(1, 0.5, X, Y, None, None, 0.01, 100)
```

```
Epoch 50 : cost = 0.00039502297 W = 0.47482544 b = 0.79619634
Epoch 100 : cost = 0.0004221364 W = 0.37172347 b = 1.2749666
----------------------------------------------------------

Weight = 0.37172347 bias = 1.2749666

Training cost = 0.0004221364

Test cost = 0.10526311671263254
```



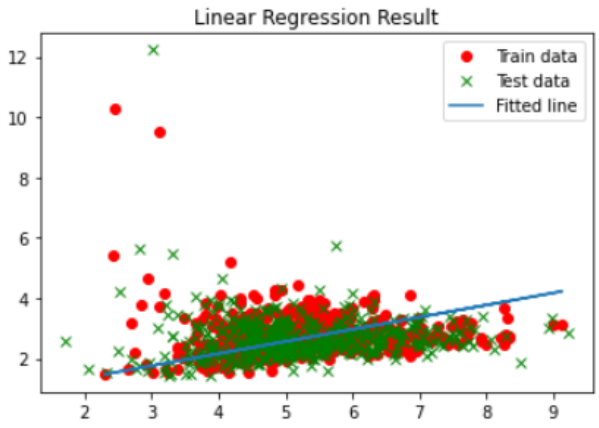
Αρχικά χρησιμοποιήσαμε το Dataset Iris και συγκεκριμένα τα χαρακτηριστικά 2 και 1. Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 150 δείγματα από τα οποία τα μισά ως δεδομένα εκπαίδευσης και τα υπόλοιπα μισά ως δεδομένα ελέγχου. Από την παραπάνω γραφική παράσταση των δύο χαρακτηριστικών παρατηρούμε ότι δεν έχουν απόλυτη γραμμική εξάρτηση γεγονός που επιβεβαιώνεται και από το κόστος του ελέγχου που είναι 0,1052. Οι τιμές του μοντέλου που υπολογίστηκε είναι $weight = 0{,}37172347$ και $bias = 1{,}2749666$.

Στην συνέχεια χρησιμοποιήσαμε το Dataset The California housing και συγκεκριμένα τα χαρακτηριστικά 2 «Μέσος Αριθμός Δωματίων» και 5 «Μέσος Αριθμός Ενοίκων». Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε τα 1.000 από τα οποία τα μισά ως δεδομένα εκπαίδευσης και τα υπόλοιπα μισά ως δεδομένα ελέγχου. Από την γραφική

παράσταση των δύο χαρακτηριστικών παρατηρούμε ότι έχουν γραμμική εξάρτηση γεγονός που αναμενόταν αφού όσο μεγαλύτερος είναι ο αριθμός των ενοίκων σε ένα οίκημα τόσο περισσότερα είναι και τα δωμάτια. Η γραμμική εξάρτηση επιβεβαιώνεται και από το μικρό κόστος ελέγχου 0,0376 (αρκετά μικρότερο από αυτό που βρήκαμε όταν κάναμε χρήση του Iris Dataset). Τέλος, τιμές του μοντέλου που υπολογίστηκε είναι $weight = 0{,}402414$ και $bias = 0{,}5541173$.

```
[4] housing = datasets.fetch_california_housing()

    X = np.transpose(housing.data[0:1000, 2:3]) # AveRooms
    X = X.flatten()  # Return a copy of the array collapsed into one dimension.
    Y = np.transpose(housing.data[0:1000, 5:6]) # AveOccup
    Y = Y.flatten()
    weight, bias, cost = dm.linear_regression(1, 0.5, X, Y, None, None, 0.01, 100)
```

```
Epoch 50 : cost = 3.2294334e-05 W = 0.4195088 b = 0.46060175
Epoch 100 : cost = 1.3946222e-05 W = 0.4024148 b = 0.5541173
----------------------------------------------------------

Weight = 0.4024148 bias = 0.5541173

Training cost = 1.3946222e-05

Test cost = 0.037620251164394373
```



2.1.3 Πολυωνυμικό μοντέλο

Η γραμμική παλινδρόμηση αποτελεί μερική περίπτωση της πολυωνυμικής παλινδρόμησης (multinomial regression), όπου το μοντέλο μας τώρα είναι ένα πολυώνυμο n βαθμού:

$$f(x) = w_n x^n + \cdots + w_1 x + w_0 \qquad \text{Εξίσωση 7}$$

Η συνάρτηση που θα υλοποιεί το πολυωνυμικό μοντέλο της μέθοδο της παλινδρόμησης είναι η ακόλουθη, όπου θα έχει τις ίδιες παραμέτρους εισόδου με αυτή του γραμμικού μοντέλου και επιπρόσθετα θα υπάρχει η μεταβλητή num_coeffs όπου εκεί ο χρήστη θα καταχωρεί το βαθμό του πολυωνύμου που θέλει να υπολογίσει:

polynomial_regression(split, ratio, x, y, x_test, y_test, learning_rate, training_epochs, num_coeffs)

Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά ανάλογα με την τιμή της μεταβλητής split έχουμε το διαχωρισμό των δεδομένων μας σε training και test data με χρήση της βοηθητική συνάρτησης **split_dataset(x, y, ratio)** (Παράρτημα Α).

```
# Polynomial regression
def polynomial_regression(split, ratio, x, y,  x_test, y_test,
learning_rate, training_epochs, num_coeffs):

    if split == 1:
        x_train, x_test, y_train, y_test = split_dataset(x, y, ratio)
        n = x_train.size
        n_test = x_test.size
    else:
        x_train = x
        y_train = y
        n = x.size
        n_test = x_test.size
```

Στην συνέχεια υπολογίζουμε τις τιμές του X που θα χρησιμοποιήσουμε για το σχεδιασμό της γραφικής παράστασης του μοντέλου μας. Παίρνουμε 100 τιμές μεταξύ της μικρότερης και μεγαλύτερης τιμής του X.

```
    # Calculating x values the predictions
    x_min = min(x)
    x_max = max(x)
    x_values = np.ndarray(100, dtype=float)
    for i in range(100):
        x_values[i] = x_min + i * (x_max - x_min)/100
```

Στην συνέχεια ορίζουμε τις μεταβλητές και τον μοντέλο μας.

```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

def model(X, w):
    terms = []
    for i in range(num_coeffs):
        term = tf.multiply(w[i], tf.pow(X, i))
        terms.append(term)
    return tf.add_n(terms)

w = tf.Variable([0.] * num_coeffs, name="parameters")
y_model = model(X, w)
```

Στην συνέχεια ορίζουμε την συνάρτηση κόστους (το οποίο είναι το Μέσο Άθροισμα Τετραγώνων του Σφάλματος - MSE) και το αλγόριθμο Gradient Descent ως μέθοδο της εκμάθησης του μοντέλου μας.

```
# Mean Squared Error Cost Function
cost = tf.reduce_sum(tf.square(Y-y_model)) / (n)
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Ακολούθως ξεκινάμε με την έναρξη του Tensorlfow Session για την εκμάθηση του μοντέλου μας, όπου ανά 50 εποχές εκμάθησής εκτυπώνουμε τις παραμέτρους του μοντέλου και το training cost που έχει υπολογισθεί μέχρι εκείνη την στιγμή.

```
# Starting the Tensorflow Session
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
# Iterating through all the epochs
    for epoch in range(training_epochs):

        # Feeding each data point into the optimizer using Feed Dictionary
        for (x, y) in zip(x_train, y_train):
            sess.run(train_op, feed_dict={X: x, Y: y})

        # Displaying the result after every 50 epochs
        if (epoch + 1) % 50 == 0:
            # Calculating the cost a every epoch
            c = sess.run(cost, feed_dict = {X : x, Y : y})
            print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(w))

    # Storing necessary values to be used outside the Session
```

```
        training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
        weight = sess.run(w)
    sess.close()
```

Αφού τελειώσει το Session και έχει υπολογιστεί το μοντέλο μας, εκτυπώνουμε τις μεταβλητές τους (W) και το τελικό training cost.

```
    # Print Weight and bias of the calculated model and cost from training data
    print("-------------------------------------------------------", '\n')
    print("Weight =", weight, '\n')
    print("Training cost =", training_cost, '\n')
```

Στη συνέχεια υπολογίζουμε τα prediction data τόσο για τα training data όσο και για τα testing data. Ενώ υπολογίζουμε και το συνολικό cost για τα testing data.

```
    # Calculating the predictions
    predictions = 0
    for i in range(num_coeffs):
        predictions += weight[i] * np.power(x_values, i)
    # Calculating the predictions for test data
    predictions_test = 0
    for i in range(num_coeffs):
        predictions_test += weight[i] * np.power(x_test, i)
    diff = y_test - predictions_test
    cost_test = 0.0
    for (z) in range(n_test):
      cost_test += math.pow(diff[z], 2)
    cost_test = math.sqrt(cost_test)  / (2 * n_test)
    # Print cost from test data
    print("Test cost =", cost_test, '\n')
```

Τέλος δημιουργούμε μια γραφική παράσταση με τα training, testing και prediction data και επιστρέφουμε τις τιμές του μοντέλου μας, δηλαδή τις μεταβλητές W και το συνολικό cost για τα testing data.

```
    # Plotting the Results
    plt.plot(x_train, y_train, 'ro', label ='Training data')
    plt.plot(x_test, y_test, 'gx', label ='Test data')
    plt.plot(x_values, predictions, label ='Fitted line')
    plt.title('Polynomial Regression Result')
    plt.legend()
    plt.show()
    return weight
```

Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.1.2 Πολυωνυμικό μοντέλο.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris (Παράρτημα Γ).

2.1.2 Πολυωνυμικό μοντέλο.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Import Data Mining Toolbox

```
[2]  import dmtoolbox_v1 as dm
```

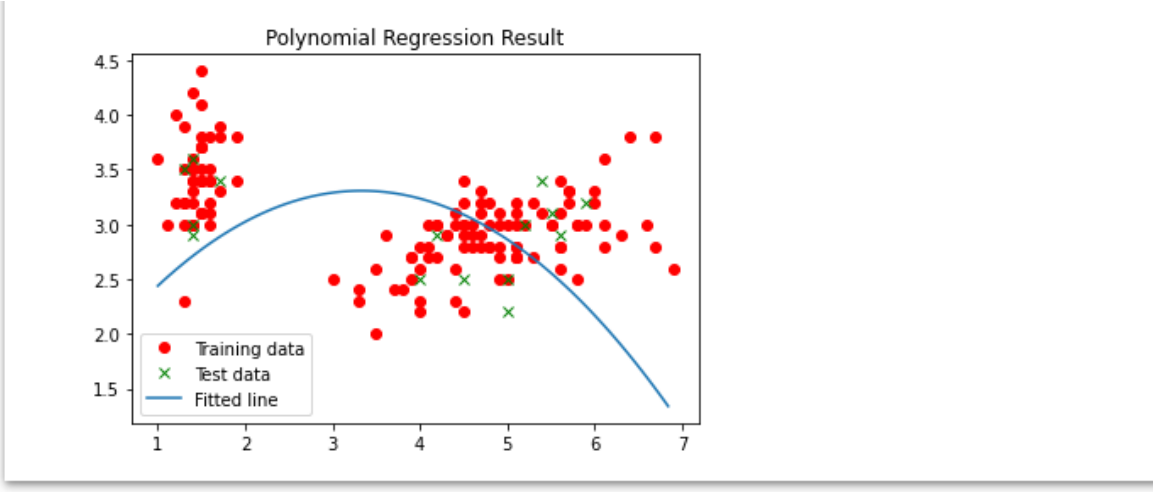
Import Datasets IRIS (https://archive.ics.uci.edu/ml/datasets/iris)

```
[3]  from sklearn import datasets
     import numpy as np

     iris = datasets.load_iris()
     X = np.transpose(iris.data[0:150, 2:3])
     X = X.flatten()  # Return a copy of the array collapsed into one dimension.
     Y = np.transpose(iris.data[0:150, 1:2])
     Y = Y.flatten()
```

Polynomial Regression

```
[4]  weight, cost = dm.polynomial_regression(1, 0.9, X, Y, None, None, 0.01, 300, 3)
```

```
Epoch 50 : cost = 0.015803132 W = [ 0.73355305  0.8872118  -0.08730913]
Epoch 100 : cost = 0.0063606603 W = [ 1.0493244   1.1415261  -0.15382436]
Epoch 150 : cost = 0.0038816186 W = [ 1.2211119   1.1877592  -0.17105909]
Epoch 200 : cost = 0.0029911147 W = [ 1.3414451   1.1638682  -0.17155422]
Epoch 250 : cost = 0.0025512052 W = [ 1.442154   1.1171709 -0.1664752]
Epoch 300 : cost = 0.0022615595 W = [ 1.5342295   1.0638684  -0.15964931]
----------------------------------------------------------

Weight = [ 1.5342295   1.0638684  -0.15964931]

Training cost = 0.0022615595

Test cost = 0.15597722515148416
```



Από το Dataset Iris χρησιμοποιήσαμε τα χαρακτηριστικά 2 και 1. Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 150 δείγματα από τα οποία τα μισά ως δεδομένα εκπαίδευσης και τα υπόλοιπα μισά ως δεδομένα ελέγχου.

Ο βαθμός του πολυωνύμου που υπολογίσαμε είναι 2, και οι μεταβλητές του μοντέλου που υπολογίστηκε είναι $w_0 = 1{,}5342295$, $w_2 = 1{,}0638684$και $w_3 = -0{,}15964931$. Όπως βλέπουμε και από την γραφική παράσταση τα δεδομένα δεν εφαρμόζουν αρκετά καλά από ένα πολυώνυμου δευτέρου βαθμού, γεγονός που επιβεβαιώνεται και από το κόστους ελέγχου που είναι 0, 1559 (μεγαλύτερο από το 0,1052 που βρήκαμε για τα ίδια δεδομένα με την γραμμική προσέγγιση).

2.1.4 Υπολογισμός παραμέτρου εξομάλυνσης

Η κανονικοποίηση (regularization) αποτελεί μία από τις σημαντικότερες τεχνικές της μηχανικής μάθησης, η οποία χρησιμοποιείται για την αποφυγή υπερεκπαίδευσης. Από μαθηματική σκοπιά, προσθέτει έναν επιπλέον όρο λ (regularization term) στη συνάρτηση κόστους, προκειμένου να αποτρέψει τους συντελεστές της εξίσωσης της παλινδρόμησης να ταιριάσουν (fit) τέλεια στα δεδομένα, γεγονός που οδηγεί σε υπερεκπαίδευση.

$$Cost(X, Y) = Loss(X, Y) + \lambda|w| \quad \text{Εξίσωση 8}$$

Υπάρχουν δύο βασικά είδη κανονικοποίησης, τα οποία συμβολίζονται με L1 και L2 αντίστοιχα, η διαφορά των οποίων έγκειται στο γεγονός ότι στην περίπτωση της L2 κανονικοποίησης, ο όρος regularization ισούται με το άθροισμα του τετραγώνου των συντελεστών, ενώ στην L1 αντιστοιχεί απλώς στο άθροισμα των συντελεστών της εξίσωσης της γραμμικής παλινδρόμησης.

Η συνάρτηση που θα υπολογίσει τις παραμέτρου εξομάλυνσης είναι η ακόλουθη:

regularized(ratio, x, y, learning_rate, training_epochs, num_coeffs)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **ratio**: η αναλογία με τη οποία θα γίνουν split τα δεδομένα εισόδου σε training και test data
- **x**: τα δεδομένα για την μεταβλητή X του μοντέλου
- **y**: τα δεδομένα για την μεταβλητή Y του μοντέλου
- **learing_rate**: ρυθμός εκμάθησης του μοντέλου
- **training_epochs**: σύνολο εποχών εκμάθησης του μοντέλου
- **num_coeffs**: βαθμός πολυωνύμου

ενώ οι παράμετροι εξόδου είναι οι ακόλουθοι:

- **l**: με τις δέκα τιμές του όρου λ που υπολόγισε η συνάρτηση.
- **fc**: με τα δέκα τελικά κόστη για τις αντίστοιχες τιμές του όρου λ που υπολογίσαμε.

Η συνάρτηση αυτή θα δέχεται τα δεδομένα X και Y και θα υπολογίζει για 10 διαφορετικές τιμές του λ (από 0.1 έως 1.0) το συνολικό κόστος για το πολυωνυμικό μοντέλο της εξίσωσης 5. Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά διαχωρίζουμε τα δεμένα εισόδου σε training και test data με χρήση της βοηθητικής συνάρτησης split_dataset.

```
# Regularized cost function
def regularized(ratio, x, y, learning_rate, training_epochs, num_coeffs):
    reg_lambda = 0.
    n = x.size
    x_data = x
    y_data = y
    (x_train, x_test, y_train, y_test) = split_dataset(x_data, y_data, ratio)
```

Στην συνέχεια ορίζουμε τις μεταβλητές και το μοντέλο μας. Στους πίνακες **l** και **fc** θα αποθηκεύσουμε τις 10 τελικές τιμές του όρου λ και του αντίστοιχου κόστους που υπολογίσαμε και τους οποίους επιστρέφει η συνάρτηση ως έξοδο.

```
    X = tf.placeholder(tf.float32)
    Y = tf.placeholder(tf.float32)
    l = np.array([])
    fc = np.array([])
    def model(X, w):
        terms = []
        for i in range(num_coeffs):
            term = tf.multiply(w[i], tf.pow(X, i))
            terms.append(term)
        return tf.add_n(terms)

    w = tf.Variable([0.] * num_coeffs, name="parameters")
    y_model = model(X, w)
```

Στην συνέχεια ορίζουμε την συνάρτηση κόστους (σύμφωνα με την Εξίσωση 6) και τον αλγόριθμο Gradient Descent ως μέθοδο της εκμάθησης του μοντέλου μας.

```
    cost = tf.div(tf.add(tf.reduce_sum(tf.square(Y-y_model)),
           tf.multiply(reg_lambda, tf.reduce_sum(tf.square(w)))),
           n)
    train_op =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για την εκμάθηση του μοντέλου μας, όπου εκτυπώνουμε τις 10 διαφορετικές τιμές του λ (από 0.1 έως 1.0) που υπολογίζουμε και το συνολικό κόστος για το πολυωνυμικό μοντέλο της εξίσωσης 5.

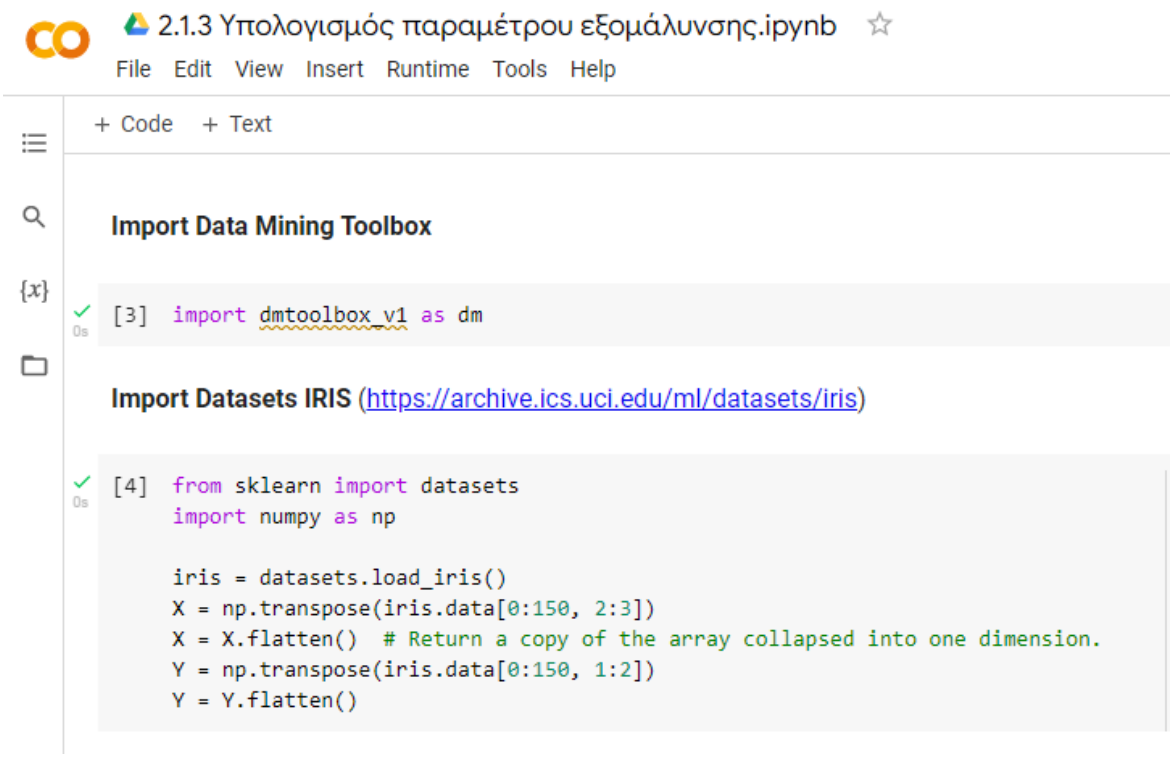
```
    sess = tf.Session()
    init = tf.global_variables_initializer()

    sess.run(init)
    for reg_lambda in np.linspace(0,1,10):
```

```
        for epoch in range(training_epochs):
            sess.run(train_op, feed_dict={X: x_train, Y: y_train})
        final_cost = sess.run(cost, feed_dict={X: x_test, Y:y_test})
        print('reg lambda', reg_lambda)
        print('final cost', final_cost)
        l = np.append(l, reg_lambda)
        fc = np.append(fc, final_cost)
    sess.close()
```

Τέλος η συνάρτηση επιστρέφει τους πίνακες l και fc.

```
    return l, fc
```

Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.1.3 Υπολογισμός παραμέτρου εξομάλυνσης.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris (Παράρτημα Γ).



Regularized

```
lamda, cost = dm.regularized(0.7, X, Y, 0.01, 300, 2)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
reg lambda 0.0
final cost 0.17730026
reg lambda 0.1111111111111111
final cost 0.07512642
reg lambda 0.2222222222222222
final cost 0.055553712
reg lambda 0.3333333333333333
final cost 0.053590342
reg lambda 0.4444444444444444
final cost 0.054548457
reg lambda 0.5555555555555556
final cost 0.055507407
reg lambda 0.6666666666666666
final cost 0.05610508
reg lambda 0.7777777777777777
final cost 0.056432724
reg lambda 0.8888888888888888
final cost 0.056603458
reg lambda 1.0
final cost 0.056690395
```

Από το Dataset Iris χρησιμοποιήσαμε τα χαρακτηριστικά 2 και 1. Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 150 δείγματα από τα οποία τα μισά ως δεδομένα εκπαίδευσης και τα υπόλοιπα μισά ως δεδομένα ελέγχου.

Από τα αποτελέσματα παρατηρούμε ότι αρχικά όσο αυξάνεται η τιμή του λ μειώνεται το τελικό κόστος γεγονός που οφείλεται στη μείωση της υπερεκπαίδευσης που προκαλεί ο όρος λ. Από την τιμή όπως 0,4 μέχρι την τιμή 1,0 έχουμε μια αύξηση, όπου πλέον η αύξηση του λ δεν επιφέρει κάποιο κέρδος στο μοντέλο μας.

2.2 Εφαρμογή Tensorflow σε Classification

2.2.1 Γενικά

Στη ενότητα αυτή θα ασχοληθούμε με την Μέθοδο Εξόρυξης Δεδομένων της Κατηγοριοποίησης (Classification). Συγκεκριμένα θα ασχοληθούμε με την δημιουργία μιας συνάρτησης που θα κάνει χρήση της γραμμικής παλινδρόμησης, που είδαμε στη προηγούμενη ενότητα, για την κατηγοριοποίηση σε ένα πρόβλημα binary classification. Στην συνέχεια θα δημιουργήσουμε μια συνάρτηση για την κατηγοριοποίησης ενός data set με χρήση της λογικής softmax regression. Τέλος θα εφαρμόσουμε την λογική softmax regression για την κατηγοριοποίηση εικόνων.

2.2.2 Binary Classification

Έστω ότι έχουμε δύο ανεξάρτητες μεταβλητές X1 και X2 και θέλουμε να κατηγοριοποιήσουμε τα δεδομένα μας σε δύο κατηγορίες Label1 και Label2. Ένας απλό μοντέλο για να αντιστοιχίσουμε τις εισόδους x και την έξοδο M(x), κάνοντας χρήση της παλινδρόμησης και συγκεκριμένα της σιγμοειδής συνάρτησης, είναι το ακόλουθο:

$$M(x, w) = sig(w_2 x_2 + w_1 x_1 + w_0) \quad \text{Εξίσωση 9}$$

Η συνάρτηση που θα υλοποιήσει το Binary Classification με χρήση της γραμμικής παλινδρόμησης είναι η ακόλουθη:

logistic2d(x1_label1, x2_label1, x1_label2, x2_label2, learning_rate, training_epochs)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **x1_label1**: τα δεδομένα της μεταβλητής X1 για την πρώτη κατηγορία Label1
- **x2_label1**: τα δεδομένα της μεταβλητής X2 για την πρώτη κατηγορία Label1
- **x1_label2**: τα δεδομένα της μεταβλητής X1 για της δεύτερης κατηγορία Label2
- **x2_label2**: τα δεδομένα της μεταβλητής X2 για την δεύτερης κατηγορία Label2
- **learning_rate**: ρυθμός εκμάθησης του μοντέλου
- **training_epochs**: σύνολο εποχών εκμάθησης του μοντέλου

ενώ η παράμετρος εξόδους της συνάρτησης είναι:

- **w_val**: με τις παραμέτρου του μοντέλου (Εξίσωση 9)

Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά ορίζουμε τις μεταβλητές, το μοντέλο μας και την συνάρτηση κόστους.

```
# Classification Logistic2d
def logistic2d(x1_label1, x2_label1, x1_label2, x2_label2, learning_rate,
training_epochs):

    x1s = np.append(x1_label1, x1_label2)
    x2s = np.append(x2_label1, x2_label2)
    ys = np.asarray([0.] * len(x1_label1) + [1.] * len(x1_label2))
    X1 = tf.placeholder(tf.float32, shape=(None,), name="x1")
    X2 = tf.placeholder(tf.float32, shape=(None,), name="x2")
    Y = tf.placeholder(tf.float32, shape=(None,), name="y")
    w = tf.Variable([0., 0., 0.], name="w", trainable=True)

    y_model = tf.sigmoid(-(w[2] * X2 + w[1] * X1 + w[0]))
    cost = tf.reduce_mean(-tf.log(y_model * Y + (1 - y_model) * (1 - Y)))
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για την εκμάθηση του μοντέλου μας, όπου ανά 100 εποχές εκμάθησής εκτυπώνουμε το training cost που έχει υπολογισθεί μέχρι εκείνη την στιγμή. Εάν η διαφορά από το προηγούμενο κόστος που υπολογίσαμε είναι μικρότερη από 0,0001 σταματάμε την εκπαίδευση ανεξάρτητα εάν έχουμε τελειώσει με όλες τις εποχές εκμάθησης.

```
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        prev_err = 0
        for epoch in range(training_epochs):
            err, _ = sess.run([cost, train_op], {X1: x1s, X2: x2s, Y: ys})
            if epoch % 100 == 0:
                print("Epoch" , epoch, ": cost:", err)
            if abs(prev_err - err) < 0.0001:
                break
            prev_err = err

        w_val = sess.run(w, {X1: x1s, X2: x2s, Y: ys})
```

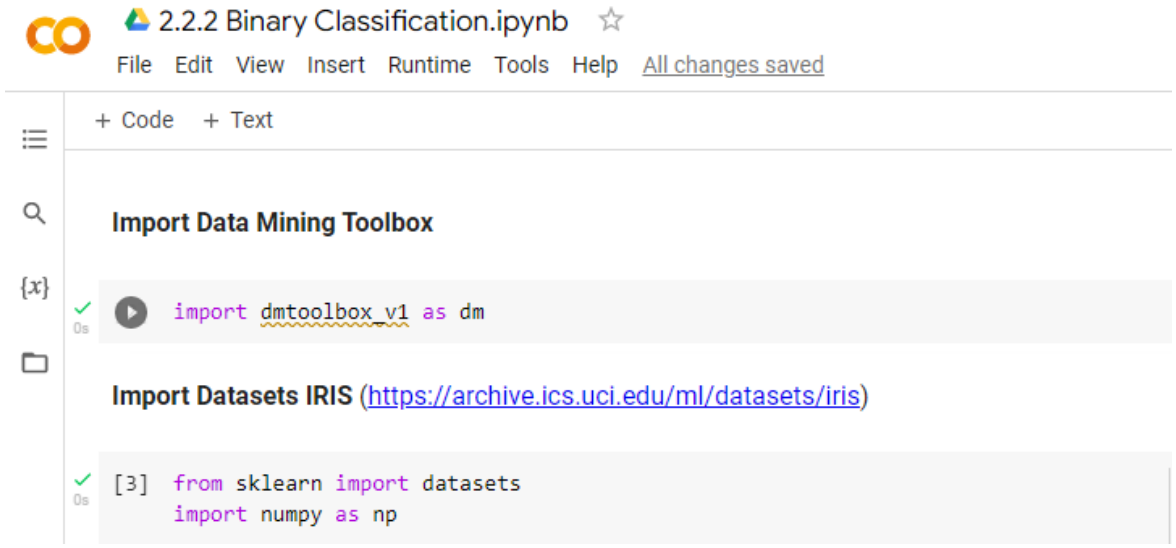
Με βάση το μοντέλο που υπολογίσαμε με την παραπάνω εκπαίδευση (οι παράμετροι του μοντέλου μας είναι αποθηκευμένοι στο πίνακα w_val) ξεκινάμε ένα νέο Tensorflow Session για να υπολογίσουμε την γραμμή διαχωρισμού των δύο κατηγοριών.

```
    x1_boundary, x2_boundary = [], []
    with tf.Session() as sess:
        for x1_test in np.linspace(0, 10, 20):
            for x2_test in np.linspace(0, 10, 20):
                z = sess.run(tf.sigmoid(-x2_test*w_val[2] - x1_test*w_val[1] -
w_val[0]))
                if abs(z - 0.5) < 0.05:
                    x1_boundary.append(x1_test)
                    x2_boundary.append(x2_test)
```

Τέλος εκτυπώνουμε την γραφική παράσταση των δεδομένων μας και την γραμμή διαχωρισμού των δύο κατηγοριών που υπολογίσαμε και επιστρέφουμε τις τιμές του μοντέλου μας.

```
    print('\n')
    plt.scatter(x1_boundary, x2_boundary, c='b', marker='o', s=20,
label='Boundary')
    plt.scatter(x1_label1, x2_label1, c='r', marker='x', s=20, label='Label1')
    plt.scatter(x1_label2, x2_label2, c='g', marker='1', s=20, label='Label2')
    plt.title('Binary Classification')
    plt.legend()
    plt.show()

    return w_val
```

Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.2.2 Binary Classification.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris (Παράρτημα Γ).

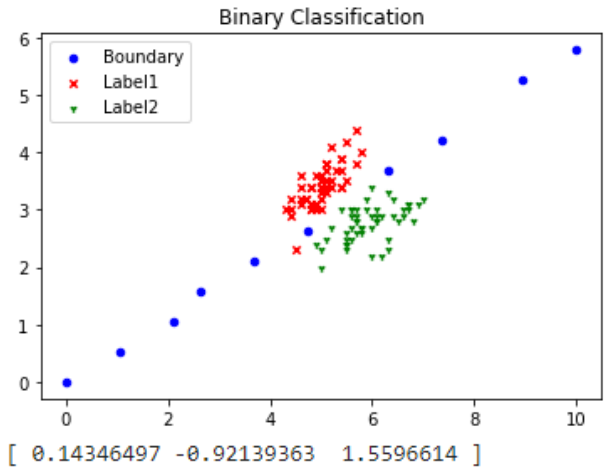


```
iris = datasets.load_iris()
X = np.transpose(iris.data[0:150, 2:3])
X = X.flatten()  # Return a copy of the array collapsed into one dimension.
Y = np.transpose(iris.data[0:150, 1:2])
Y = Y.flatten()
```

Classification Logistic2d

```
x1_label1 = iris.data[0:50,:1]  # 1st feature -- Iris Setosa
x1_label2 = iris.data[50:100,:1]# 1st feature-- Iris Versicolour

x2_label1 = iris.data[0:50,1:2]  # 2st feature -- Iris Setosa
x2_label2 = iris.data[50:100,1:2]# 2st feature-- Iris Versicolour


w_val = dm.logistic2d(x1_label1, x2_label1, x1_label2, x2_label2, 0.01, 1000)
print(w_val)
```

```
Epoch 0 : cost: 0.6931472
Epoch 100 : cost: 0.63091767
Epoch 200 : cost: 0.5785373
Epoch 300 : cost: 0.533782
Epoch 400 : cost: 0.4953506
Epoch 500 : cost: 0.46215522
Epoch 600 : cost: 0.43330115
Epoch 700 : cost: 0.40805954
Epoch 800 : cost: 0.3858385
Epoch 900 : cost: 0.36615726
```




```
[ 0.14346497 -0.92139363  1.5596614 ]
```

Από το Dataset Iris χρησιμοποιήσαμε την πρώτη και δεύτερη κατηγορία λουλουδιών Iris (Setosa και Versicolour αντίστοιχα). Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 50 δείγματα της κάθε κατηγορίας. Από την γραφική παράσταση παρατηρούμε ότι η ευθεία διαχωρισμού των δύο κατηγοριών είναι πάρα πολύ καλή έχει

γίνει σωστά ο διαχωρισμός. Τέλος οι παράμετροι του μοντέλου που υπολογίστηκε είναι $w_0 = 0{,}14346497$, $w_1 = -0{,}92139363$ και $w_0 = 1{,}5596614$

2.2.3 Classification με softmax regression

Στην συγκεκριμένη ενότητα θα ασχοληθούμε με την κατηγοριοποίηση σε τρεις κλάσεις κάνοντας χρήσης της λογικής softmax regression. Η softmax πήρε το όνομά της από την παραδοσιακή συνάρτηση max, η οποία παίρνει ένα διάνυσμα και επιστρέφει τη μέγιστη τιμή του. Ωστόσο, η softmax δεν είναι ακριβώς η συνάρτηση max επειδή έχει το πρόσθετο πλεονέκτημα ότι είναι συνεχής και διαφορίσιμη. Ως αποτέλεσμα, έχει τις χρήσιμες ιδιότητες για την αποτελεσματική λειτουργία της στοχαστικής gradient descent.

Σε αυτόν τον τύπο ταξινόμησης πολλαπλών κλάσεων, κάθε τάξη έχει μια βαθμολογία (ή πιθανότητα) για κάθε διάνυσμα εισόδου. Το βήμα softmax απλώς επιλέγει την έξοδο με την υψηλότερη βαθμολογία. Η συνάρτηση που θα υλοποιήσει το Classification με χρήση της της λογικής softmax regression είναι η ακόλουθη:

softmax(ratio, x1_label0, x2_label0, x1_label1, x2_label1, x1_label2, x2_label2, learning_rate, training_epochs)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **ratio**: η αναλογία με την οποία θα γίνει ο διαχωρισμός των δεδομένων σε training και test data σε σχέση με το σύνολο των δεδομένων.
- **x1_label0**: τα δεδομένα της μεταβλητής X1 για την πρώτη κατηγορία Label0
- **x2_label0**: τα δεδομένα της μεταβλητής X2 για την πρώτη κατηγορία Label0
- **x1_label1**: τα δεδομένα της μεταβλητής X1 για την δεύτερη κατηγορία Label1
- **x2_label1**: τα δεδομένα της μεταβλητής X2 για την δεύτερη κατηγορία Label1
- **x1_label2**: τα δεδομένα της μεταβλητής X1 για την τρίτη κατηγορία Label2
- **x2_label2**: τα δεδομένα της μεταβλητής X2 για την τρίτη κατηγορία Label2
- **learning_rate**: ρυθμός εκμάθησης του μοντέλου
- **training_epochs**: σύνολο εποχών εκμάθησης του μοντέλου

ενώ οι παράμετροι εξόδου είναι οι ακόλουθοι:

- **W_val**: οι παράμετροι weight του μοντέλου μας.
- **b_val**: οι παράμετροι bias του μοντέλου μας.

- **accuracy**: με την ακρίβεια της κατηγοριοποίησης που κάναμε.

Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά κάνουμε την γραφική παράσταση των τριών κατηγοριών με τα δύο χαρακτηριστικά.

```
# Classification Softmax
def softmax(ratio, x1_label0, x2_label0, x1_label1, x2_label1, x1_label2,
x2_label2, learning_rate, training_epochs):

    num_labels = 3
    batch_size = 50

    plt.scatter(x1_label0, x2_label0, c='r', marker='o', s=60, label='Label0')
    plt.scatter(x1_label1, x2_label1, c='g', marker='x', s=60, label='Label1')
    plt.scatter(x1_label2, x2_label2, c='b', marker='_', s=60, label='Label2')
    plt.title('Data')
    plt.legend()
    plt.show()
    print('\n')
```

Στη συνέχεια διαχωρίζουμε τα δεδομένα σε training και test data με χρήση της βοηθητικής συνάρτησης split_dataset. Στο πίνακα xs (training data) έχουμε όλες τις τιμές των δύο χαρακτηριστικών και για τις τρεις κατηγορίες πρώτα για την κατηγορία Label0, στην συνέχεια για την κατηγορία Label1 και τέλος για την Label2. Ενώ στο πίνακα labels (training data) έχουμε σε ποια κατηγορία ανήκει η συγκεκριμένη τιμή. Αυτό το επιτυγχάνουμε με ένα πίνακα τριών διαστάσεων όπου κάθε διάσταση αντιστοιχεί σε μια κατηγορία. Έτσι για παράδειγμα η τιμή [1., 0, 0] αντιστοιχεί στην κατηγορία Label0.

```
    x1_label0, test_x1_label0, x2_label0, test_x2_label0 =
split_dataset(x1_label0, x2_label0, ratio)
    x1_label1, test_x1_label1, x2_label1, test_x2_label1 =
split_dataset(x1_label1, x2_label1, ratio)
    x1_label2, test_x1_label2, x2_label2, test_x2_label2 =
split_dataset(x1_label2, x2_label2, ratio)

    xs_label0 = np.hstack((x1_label0, x2_label0))
    xs_label1 = np.hstack((x1_label1, x2_label1))
    xs_label2 = np.hstack((x1_label2, x2_label2))

    xs = np.vstack((xs_label0, xs_label1, xs_label2))

    labels = np.matrix([[1., 0., 0.]] * len(x1_label0) + [[0., 1., 0.]] *
len(x1_label1) + [[0., 0., 1.]] * len(x1_label2))
```

Στην συνέχεια ανακατεύομε τα training data ώστε οι κατηγορίες να μην είναι όλες μαζί για την καλύτερη εκπαίδευση του μοντέλου μας.

```
    arr = np.arange(xs.shape[0])
    np.random.shuffle(arr)
    xs = xs[arr, :]
    labels = labels[arr, :]
```

Αντίστοιχα δημιουργούμε τους πίνακες test_xs και test_labels με τις τιμές των χαρακτηριστικών και των κατηγοριών για τα test data.

```
    test_xs_label0 = np.hstack((test_x1_label0, test_x2_label0))
    test_xs_label1 = np.hstack((test_x1_label1, test_x2_label1))
    test_xs_label2 = np.hstack((test_x1_label2, test_x2_label2))

    test_xs = np.vstack((test_xs_label0, test_xs_label1, test_xs_label2))
    test_labels = np.matrix([[1., 0., 0.]] * len(test_x1_label0) + [[0., 1.,
0.]] * len(test_x1_label1) + [[0., 0., 1.]] * len(test_x1_label2))
```

Στην συνέχεια ορίζουμε τις παραμέτρους, το μοντέλο μας και την συνάρτηση κόστους. Επίσης ορίζουμε και το accuracy που θα χρησιμοποιήσουμε για τα test data.

```
    train_size, num_features = xs.shape

    X = tf.placeholder("float", shape=[None, num_features])
    Y = tf.placeholder("float", shape=[None, num_labels])

    W = tf.Variable(tf.zeros([num_features, num_labels]))
    b = tf.Variable(tf.zeros([num_labels]))
    y_model = tf.nn.softmax(tf.matmul(X, W) + b)

    cost = -tf.reduce_sum(Y * tf.log(y_model))
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

    correct_prediction = tf.equal(tf.argmax(y_model, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για την εκπαίδευση του μοντέλου μας, όπου ανά 100 βήματα εκμάθησής εκτυπώνουμε το training cost που έχει υπολογισθεί εκείνη την στιγμή.

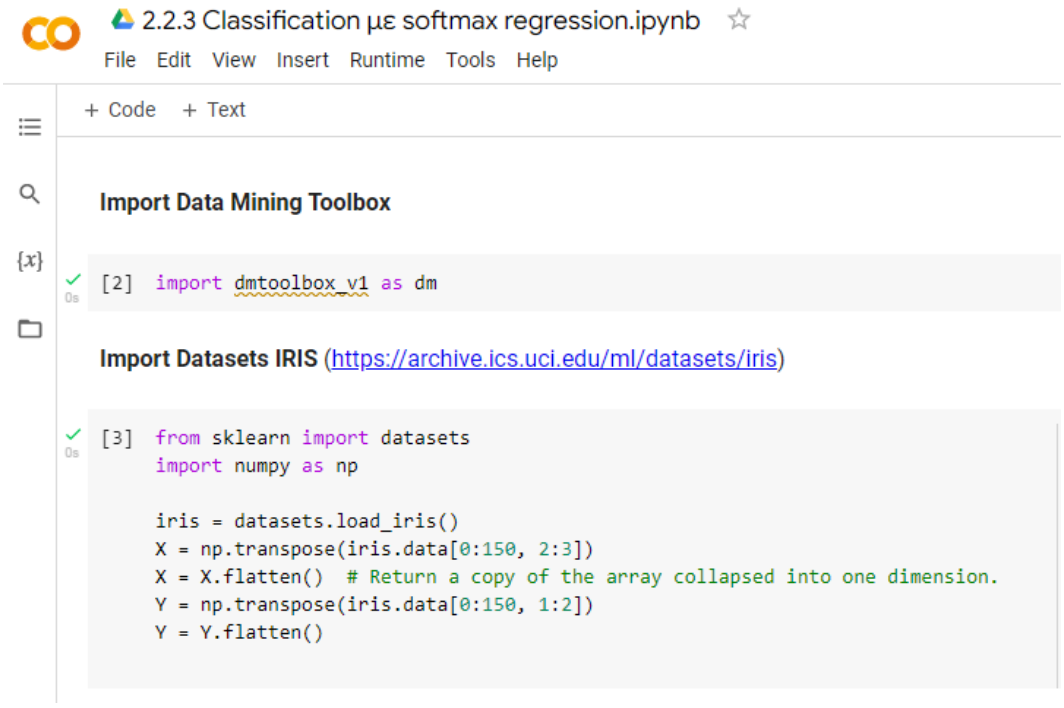
```
    with tf.Session() as sess:
      tf.global_variables_initializer().run()
```

```
      for step in range(training_epochs * train_size // batch_size):
        offset = (step * batch_size) % train_size
        batch_xs = xs[offset:(offset + batch_size), :]
        batch_labels = labels[offset:(offset + batch_size)]
        err, _ = sess.run([cost, train_op], feed_dict={X: batch_xs, Y:
batch_labels})
        if step % 100 == 0:
          print ("Step: ", step, "Error: ", err)
```

Τέλος εκτυπώνουμε και επιστρέφουμε τις παραμέτρους του μοντέλου μας και το accuracy κάνοντας χρήση των test data.

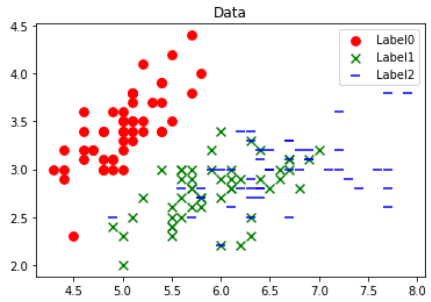
```
      W_val = sess.run(W)
      print('w', W_val)
      b_val = sess.run(b)
      print('b', b_val)
      print("accuracy", accuracy.eval(feed_dict={X: test_xs, Y: test_labels}))

    return W_val, b_val, accuracy
```

Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.2.3 Classification με softmax regression.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris (Παράρτημα Γ).



Classification Softmax

```
[4]  x1_label0 = iris.data[0:50,:1]  # 1st feature -- Iris Setosa
     x1_label1 = iris.data[50:100,:1]# 1st feature-- Iris Versicolour
     x1_label2 = iris.data[100:150,:1]# 1st feature-- Iris Virginica

     x2_label0 = iris.data[0:50,1:2]  # 2st feature -- Iris Setosa
     x2_label1 = iris.data[50:100,1:2]# 2st feature-- Iris Versicolour
     x2_label2 = iris.data[100:150,1:2]# 2st feature-- Iris Virginica


     w_val, b_val, accuracy = dm.softmax(0.5, x1_label0, x2_label0, x1_label1, x2_label1, x1_label2, x2_label2, 0.01, 1000)
```




```
Step:  0 Error:  54.930614
Step:  100 Error:  118.014175
Step:  200 Error:  43.468273
Step:  300 Error:  139.58897
Step:  400 Error:  65.624756
Step:  500 Error:  50.474644
Step:  600 Error:  152.15685
Step:  700 Error:  64.8489
Step:  800 Error:  48.85568
Step:  900 Error:  142.2327
Step:  1000 Error:  62.54216
Step:  1100 Error:  50.82129
Step:  1200 Error:  157.12473
Step:  1300 Error:  74.24551
Step:  1400 Error:  41.82476
w [[-5.536685   1.378844   4.157856 ]
 [ 9.721818  -5.4965587 -4.2252584]]
b [ 1.4731963  7.9018393 -9.375042 ]
accuracy 0.68
```

Από το Dataset Iris χρησιμοποιήσαμε και τις τρεις κατηγορίες των λουλουδιών Iris. Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 50 δείγματα της κάθε κατηγορίας. Από τα 50 αυτά δείγματα τα μισά χρησιμοποιήθηκαν ως δεδομένα εκπαίδευσης και τα υπόλοιπα μισά ως δεδομένα ελέγχου. Οι μεταβλητές του μοντέλου που υπολογίστηκε είναι w [[−5,536685 1,378844 4,157856] [9,721818 −5,4965587 −4,2252584]] και b [1,4731963 7,9018393 -9,375042]. Τέλος η τιμή του accuracy για το συγκεκριμένο μοντέλο έχει την τιμή 0,68.

2.2.4 Classification Εικόνων με softmax regression

Στην ενότητα αυτή θα χρησιμοποιήσουμε την λογική του softmax regression, που είδαμε στην προηγούμενη ενότητα, στην κατηγοριοποίηση εικόνων. Συγκεκριμένα η συνάρτηση που θα χρησιμοποιηθεί είναι η ακόλουθη:

image_classification(x_train, y_train, x_valid, y_valid, x_test, y_test, n_classes, img_h, img_w, epochs, batch_size, display_freq, learning_rate, n_test, color)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **x_train**: τα training δεδομένα της μεταβλητής X
- **y_train:** τα training δεδομένα της μεταβλητής Y
- **x_valid:** τα valid δεδομένα της μεταβλητής X
- **y_valid:** τα valid δεδομένα της μεταβλητής Y
- **x_test:** τα testing δεδομένα της μεταβλητής X
- **y_test:** τα testing δεδομένα της μεταβλητής Y
- **n_classes:** αριθμός των κλάσεων που θα κατηγοριοποιηθούν οι εικόνες
- **img_h:** ύψος εικόνας
- **img_w:** πλάτος εικόνας
- **epochs**: σύνολο εποχών εκμάθησης του μοντέλου
- **batch_size:** μέγεθος παρτίδας. Πρόκειται για ένα μέγεθος που χρησιμοποιείται για τον τεμαχισμό του δείγματος σε παρτίδες.
- **display_freq:** συχνότητα για την εμφάνιση δεδομένων κατά την εκπαίδευση του μοντέλου μας.
- **learning_rate:** ρυθμός εκμάθησης μοντέλου
- **n_test:** αριθμός δεδομένων για test
- **color:** χρώμα εικόνας

Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά ορίζουμε τις συναρτήσεις **randomize** για το ανακάτεμα το δεδομένων και **get_next_batch** για να λαμβάνουμε τα δεδομένα τις επόμενης παρτίδας, αφού η εκπαίδευση θα γίνει ανά παρτίδες, το μέγεθος της οποίας δίδεται από τον χρήστη στην μεταβλητή εισόδου batch_size.

```
# Image Classification
def image_classification(x_train, y_train, x_valid, y_valid, x_test, y_test,
n_classes, img_h, img_w, epochs, batch_size, display_freq, learning_rate,
n_test, color):

    img_size_flat = img_h * img_w * color  # hxwXc, the total number of pixels

    def randomize(x, y):
        """ Randomizes the order of data samples and their corresponding
labels"""
        permutation = np.random.permutation(y.shape[0])
        shuffled_x = x[permutation, :]
        shuffled_y = y[permutation]
        return shuffled_x, shuffled_y

    def get_next_batch(x, y, start, end):
        x_batch = x[start:end]
        y_batch = y[start:end]
        return x_batch, y_batch
```

Στην συνέχεια εκτυπώνουμε τα δεδομένα των εικόνων που φορτώθηκαν και πρόκειται να χρησιμοποιηθούν.

```
    # Print image data
    print("Size of:")
    print("- Training-set:\t\t{}".format(len(y_train)))
    print("- Validation-set:\t{}".format(len(y_valid)))

    print(y_train[1])
    print('x_train:\t{}'.format(x_train.shape))
    print('y_train:\t{}'.format(y_train.shape))
    print('x_valid:\t{}'.format(x_valid.shape))
    print('y_valid:\t{}'.format(y_valid.shape))

    y_valid[:5, :]
```

Στην συνέχεια ορίζουμε τις μεταβλητές του μοντέλου μας, το ίδιο το μοντέλο μας και την συνάρτηση κόστους, καθώς και την πρόβλεψη της κατηγορίας από το μοντέλο μας.

```
    def weight_variable(shape):
        """
        Create a weight variable with appropriate initialization
        :param name: weight name
        :param shape: weight shape
```

```
        :return: initialized weight variable
        """
        initer = tf.truncated_normal_initializer(stddev=0.01)
        return tf.get_variable('W',
                               dtype=tf.float32,
                               shape=shape,
                               initializer=initer)

    def bias_variable(shape):
        """
        Create a bias variable with appropriate initialization
        :param name: bias variable name
        :param shape: bias variable shape
        :return: initialized bias variable
        """
        initial = tf.constant(0., shape=shape, dtype=tf.float32)
        return tf.get_variable('b',
                               dtype=tf.float32,
                               initializer=initial)

    # Create the graph for the linear model
    # Placeholders for inputs (x) and outputs(y)
    x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='X')
    y = tf.placeholder(tf.float32, shape=[None, n_classes], name='Y')

    # Create weight matrix initialized randomely from N~(0, 0.01)
    W = weight_variable(shape=[img_size_flat, n_classes])

    # Create bias vector initialized as zero
    b = bias_variable(shape=[n_classes])

    output_logits = tf.matmul(x, W) + b

    # Define the loss function, optimizer, and accuracy
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,
logits=output_logits), name='loss')
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate, name='Adam-
op').minimize(loss)
    correct_prediction = tf.equal(tf.argmax(output_logits, 1), tf.argmax(y, 1),
name='correct_pred')
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),
name='accuracy')

    # Model predictions
    cls_prediction = tf.argmax(output_logits, axis=1, name='predictions')
```

Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για την εκμάθηση του μοντέλου μας, όπου ανά εποχή και ανά συχνότητα που έχω δοθεί από το χρήστη στην μεταβλητή εισόδου **display_freq** εκτυπώνουμε το batch loss και το accuracy που έχει το μοντέλο μας την στιγμή αυτή.

```
    # Creating the op for initializing all variables
    init = tf.global_variables_initializer()

    # Create an interactive session (to keep the session in the other cells)
    sess = tf.InteractiveSession()
    # Initialize all variables
    sess.run(init)
    # Number of training iterations in each epoch
    num_tr_iter = int(len(y_train) / batch_size)
    for epoch in range(epochs):
        print('Training epoch: {}'.format(epoch + 1))
        # Randomly shuffle the training data at the beginning of each epoch
        x_train, y_train = randomize(x_train, y_train)
        for iteration in range(num_tr_iter):
            start = iteration * batch_size
            end = (iteration + 1) * batch_size
            x_batch, y_batch = get_next_batch(x_train, y_train, start, end)

            # Run optimization op (backprop)
            feed_dict_batch = {x: x_batch, y: y_batch}
            sess.run(optimizer, feed_dict=feed_dict_batch)

            if iteration % display_freq == 0:
                # Calculate and display the batch loss and accuracy
                loss_batch, acc_batch = sess.run([loss, accuracy],
                                                 feed_dict=feed_dict_batch)

                print("iter {0:3d}:\t Loss={1:.2f},\tTraining
Accuracy={2:.01%}".
                      format(iteration, loss_batch, acc_batch))

        # Run validation after every epoch
        feed_dict_valid = {x: x_valid[:n_test], y: y_valid[:n_test]}
        loss_valid, acc_valid = sess.run([loss, accuracy],
feed_dict=feed_dict_valid)
        print('---------------------------------------------------------')
        print("Epoch: {0}, validation loss: {1:.2f}, validation accuracy:
{2:.01%}".
              format(epoch + 1, loss_valid, acc_valid))
        print('---------------------------------------------------------')
```

Αφού γίνει η εκπαίδευση του μοντέλου μας και υπολογιστούν οι παράμετροι του, γίνεται η δοκιμή του με τις εικόνες δοκιμής και εκτυπώνουμε το τελικό loss και accuracy των δοκιμών.

```
    # Test the network after training
    # Accuracy
    feed_dict_test = {x: x_valid[:n_test], y: y_valid[:n_test]}
    loss_test, acc_test = sess.run([loss, accuracy], feed_dict=feed_dict_test)
    print('---------------------------------------------------------')
    print("Test loss: {0:.2f}, test accuracy: {1:.01%}".format(loss_test,
acc_test))
    print('---------------------------------------------------------')
```

Τέλος ορίζουμε της συναρτήσεις **plot_images**, **plot_example_errors**, **plot_example_correct** για την εκτύπωση εικόνων, την εκτύπωση 9 εικόνων που δεν έχουν κατηγοριοποιηθεί σωστά και την εκτύπωση 9 εικόνων που έχουν κατηγοριοποιηθεί σωστά αντίστοιχα. Τις συναρτήσεις αυτές θα τις χρησιμοποιήσουμε στο τέλος για να κάνουμε τις ακόλουθες εκτυπώσεις:

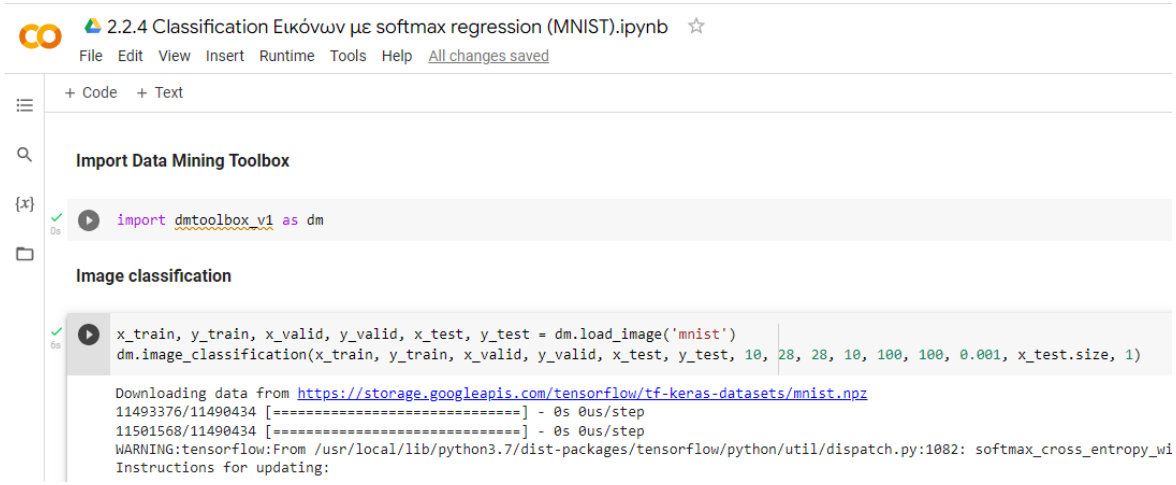
- εκτύπωση 9 εικόνων που έχουν κατηγοριοποιηθεί σωστά.
- εκτύπωση 9 εικόνων που δεν έχουν κατηγοριοποιηθεί σωστά.

```
    def plot_images(images, cls_true, cls_pred=None, title=None):
        """
        Create figure with 3x3 sub-plots.
        :param images: array of images to be plotted, (9, img_h*img_w)
        :param cls_true: corresponding true labels (9,)
        :param cls_pred: corresponding true labels (9,)
        """
        fig, axes = plt.subplots(3, 3, figsize=(9, 9))
        fig.subplots_adjust(hspace=0.3, wspace=0.3)
        for i, ax in enumerate(axes.flat):
            # Plot image.
            if color == 1:
                ax.imshow(images[i].reshape(img_h, img_w), cmap='binary')
            elif color == 3:
                ax.imshow(images[i].reshape(img_h, img_w, color),
cmap='binary')

            # Show true and predicted classes.
            if cls_pred is None:
                ax_title = "True: {0}".format(cls_true[i])
            else:
```

```
                ax_title = "True: {0}, Pred: {1}".format(cls_true[i],
cls_pred[i])

            ax.set_title(ax_title)

            # Remove ticks from the plot.
            ax.set_xticks([])
            ax.set_yticks([])

        if title:
            plt.suptitle(title, size=20)
        plt.show(block=False)

    def plot_example_errors(images, cls_true, cls_pred, title=None):
        """
        Function for plotting examples of images that have been mis-classified
        :param images: array of all images, (#imgs, img_h*img_w)
        :param cls_true: corresponding true labels, (#imgs,)
        :param cls_pred: corresponding predicted labels, (#imgs,)
        """
        # Negate the boolean array.
        incorrect = np.logical_not(np.equal(cls_pred, cls_true))

        # Get the images from the test-set that have been
        # incorrectly classified.
        incorrect_images = images[incorrect]

        # Get the true and predicted classes for those images.
        cls_pred = cls_pred[incorrect]
        cls_true = cls_true[incorrect]

        # Plot the first 9 images.
        plot_images(images=incorrect_images[0:9],
                    cls_true=cls_true[0:9],
                    cls_pred=cls_pred[0:9],
                    title=title)

    def plot_example_correct(images, cls_true, cls_pred, title=None):
        """
        Function for plotting examples of images that have been mis-classified
        :param images: array of all images, (#imgs, img_h*img_w)
        :param cls_true: corresponding true labels, (#imgs,)
        :param cls_pred: corresponding predicted labels, (#imgs,)
        """
        # Negate the boolean array.
        correct = np.equal(cls_pred, cls_true)
        # Get the images from the test-set that have been
```

```
        # incorrectly classified.
        correct_images = images[correct]

        # Get the true and predicted classes for those images.
        cls_pred = cls_pred[correct]
        cls_true = cls_true[correct]

        # Plot the first 9 images.
        plot_images(images=correct_images[0:9],
                    cls_true=cls_true[0:9],
                    cls_pred=cls_pred[0:9],
                    title=title)

    # Plot some of the correct and misclassified examples
    cls_pred = sess.run(cls_prediction, feed_dict=feed_dict_test)
    cls_true = np.argmax(y_valid[:n_test], axis=1)
    plot_example_correct(x_valid[:n_test], cls_true, cls_pred, title='Correct
Examples')
    plot_example_errors(x_valid[:n_test], cls_true, cls_pred,
title='Misclassified Examples')
    plt.show()
```

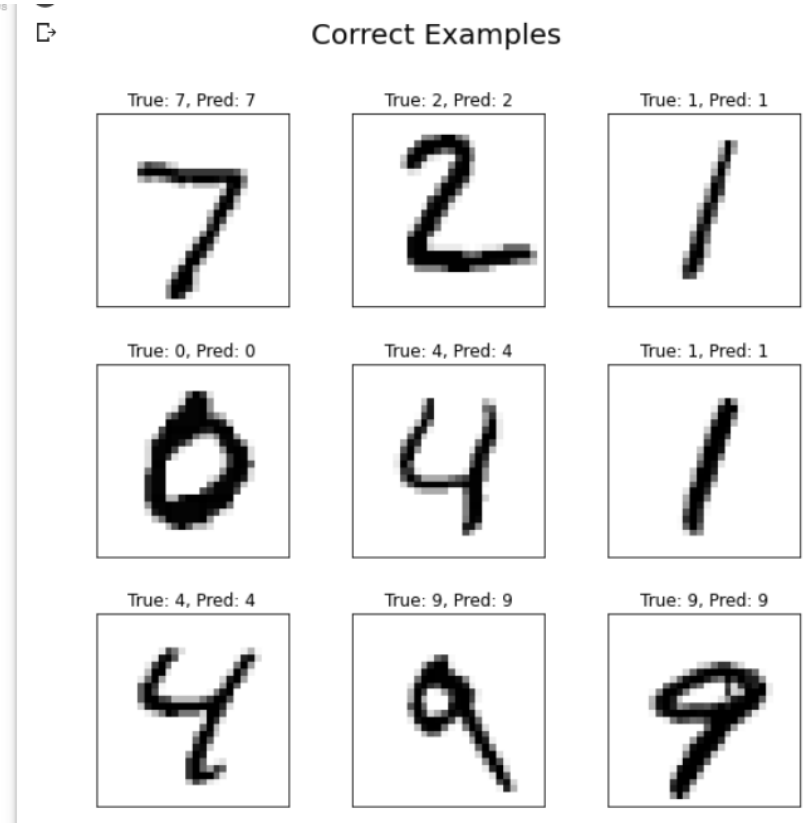
Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε τα Notebooks «2.2.4 Classification Εικόνων με softmax regression (MNIST)» και «2.2.4 Classification Εικόνων με softmax regression (CIFAR10)» (Παράρτημα Β). Στο πρώτο Notebook χρησιμοποιήσαμε το dataset MNIST (Παράρτημα Γ), που αφορά χειρόγραφα ψηφία, ενώ στο δεύτερο το CIFAR10 (Παράρτημα Γ), αποτελείται από έγχρωμες εικόνες. Για το φόρτωμα των εικόνων χρησιμοποιήσαμε την βοηθητική συνάρτηση **load_image**, η οποία δίνεται αναλυτικά στο Παράρτημα Α. Τα αποτελέσματα από το τρέξιμο των δύο Notebooks φαίνονται παρακάτω.

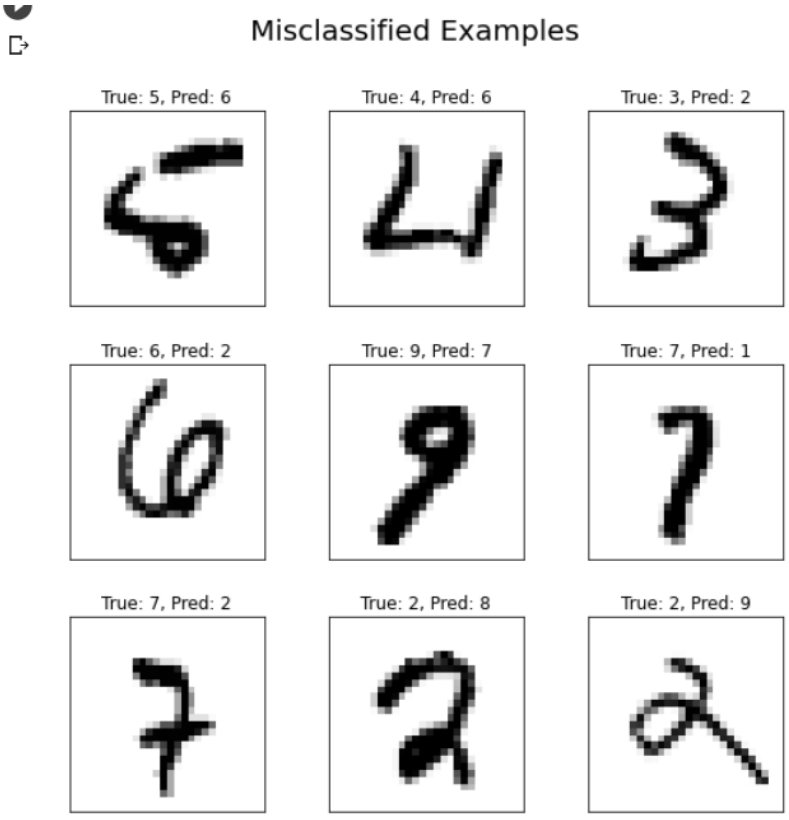


```
Size of:
- Training-set:         60000
- Validation-set:       10000
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
x_train:        (60000, 784)
y_train:        (60000, 10)
x_valid:        (10000, 784)
y_valid:        (10000, 10)
Training epoch: 1
iter   0:        Loss=20.06,    Training Accuracy=20.0%
iter 100:        Loss=2.62,     Training Accuracy=83.0%
iter 200:        Loss=1.22,     Training Accuracy=91.0%
iter 300:        Loss=1.33,     Training Accuracy=90.0%
iter 400:        Loss=1.50,     Training Accuracy=89.0%
iter 500:        Loss=2.35,     Training Accuracy=87.0%
---------------------------------------------------------
Epoch: 1, validation loss: 2.64, validation accuracy: 87.0%
---------------------------------------------------------
Training epoch: 2
iter   0:        Loss=1.09,     Training Accuracy=92.0%
iter 100:        Loss=0.76,     Training Accuracy=94.0%
iter 200:        Loss=2.05,     Training Accuracy=87.0%
iter 300:        Loss=3.87,     Training Accuracy=88.0%
iter 400:        Loss=2.65,     Training Accuracy=91.0%
iter 500:        Loss=1.79,     Training Accuracy=89.0%
---------------------------------------------------------
Epoch: 2, validation loss: 2.71, validation accuracy: 88.1%
---------------------------------------------------------
```

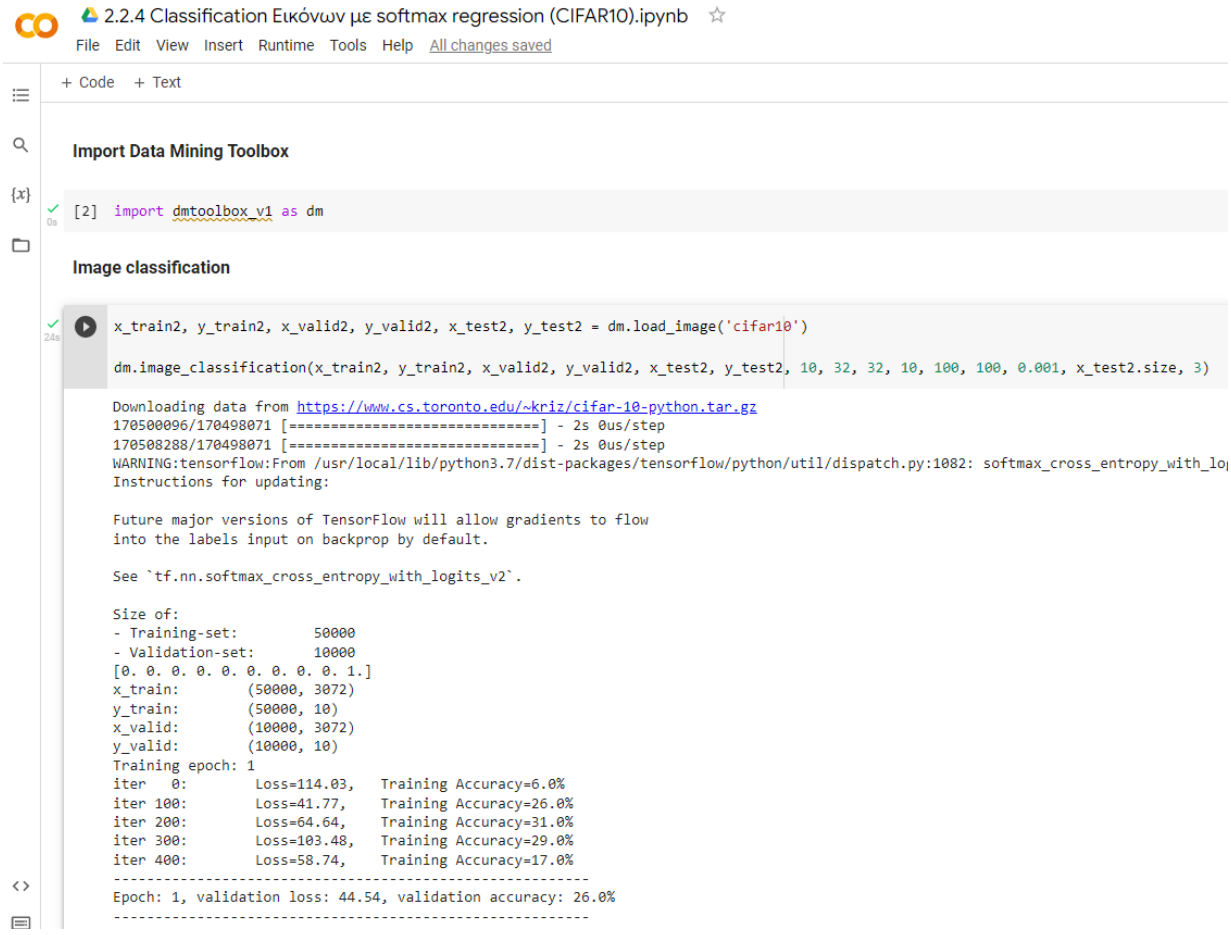
...

```
---------------------------------------------------------
Epoch: 8, validation loss: 2.70, validation accuracy: 90.0%
---------------------------------------------------------
Training epoch: 9
iter   0:        Loss=0.25,     Training Accuracy=98.0%
iter 100:        Loss=1.34,     Training Accuracy=92.0%
iter 200:        Loss=1.25,     Training Accuracy=91.0%
iter 300:        Loss=0.26,     Training Accuracy=97.0%
iter 400:        Loss=0.56,     Training Accuracy=92.0%
iter 500:        Loss=2.43,     Training Accuracy=89.0%
---------------------------------------------------------
Epoch: 9, validation loss: 2.85, validation accuracy: 88.9%
---------------------------------------------------------
Training epoch: 10
iter   0:        Loss=1.55,     Training Accuracy=91.0%
iter 100:        Loss=1.62,     Training Accuracy=89.0%
iter 200:        Loss=0.97,     Training Accuracy=93.0%
iter 300:        Loss=2.06,     Training Accuracy=94.0%
iter 400:        Loss=1.62,     Training Accuracy=87.0%
iter 500:        Loss=3.17,     Training Accuracy=91.0%
---------------------------------------------------------
Epoch: 10, validation loss: 3.34, validation accuracy: 89.3%
---------------------------------------------------------
---------------------------------------------------------
Test loss: 3.34, test accuracy: 89.3%
---------------------------------------------------------
```


Correct Examples
True: 7, Pred: 7
True: 2, Pred: 2
True: 1, Pred: 1
True: 0, Pred: 0
True: 4, Pred: 4
True: 1, Pred: 1
True: 4, Pred: 4
True: 9, Pred: 9
True: 9, Pred: 9


Misclassified Examples
True: 5, Pred: 6
True: 4, Pred: 6
True: 3, Pred: 2
True: 6, Pred: 2
True: 9, Pred: 7
True: 7, Pred: 1
True: 7, Pred: 2
True: 2, Pred: 8
True: 2, Pred: 9

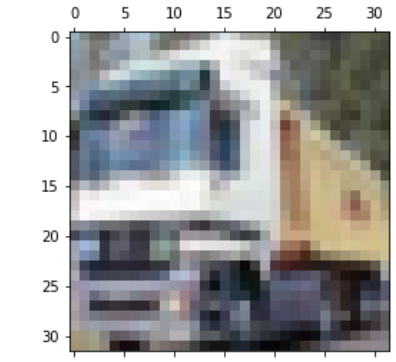
Από το Dataset MNIST έγινε χρήση και των 60.000 δειγμάτων εκπαίδευσης και των 10.000 δειγμάτων ελέγχου. Το τελικό κόστος ελέγχου είναι 3,34 ενώ το test accuracy είναι 89,3%, μια πολύ καλή τιμή.

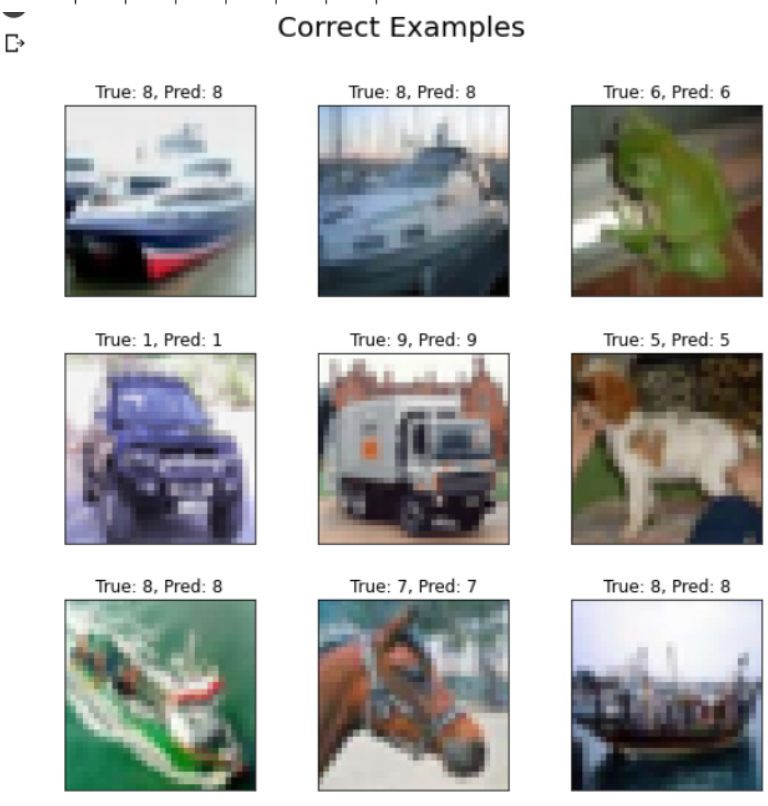


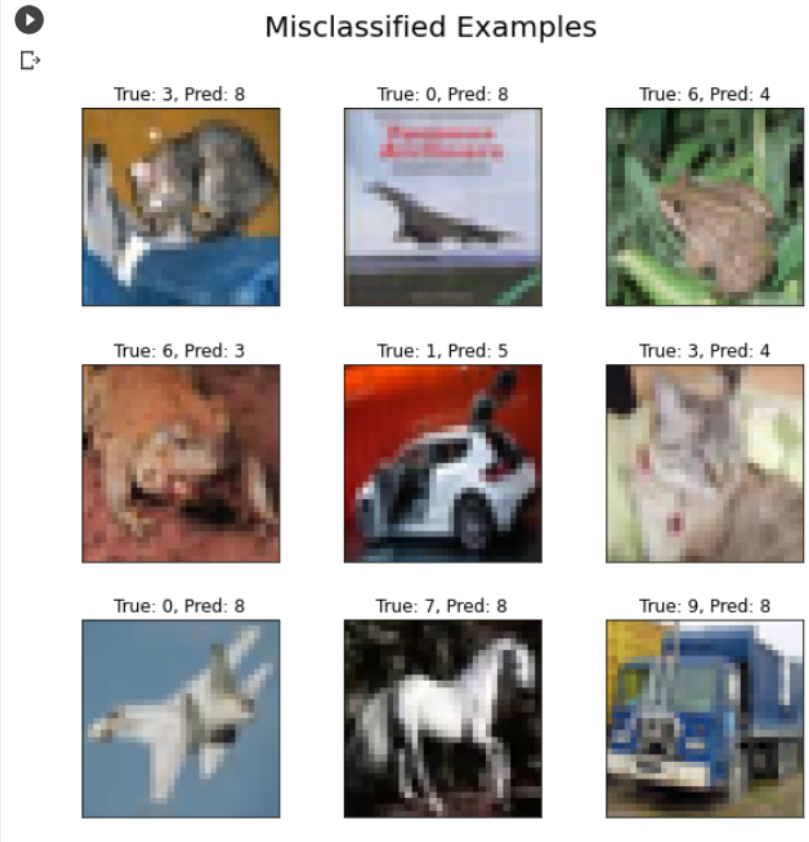

```
Training epoch: 2
iter   0:        Loss=40.22,    Training Accuracy=33.0%
iter 100:        Loss=66.22,    Training Accuracy=22.0%
iter 200:        Loss=76.30,    Training Accuracy=23.0%
iter 300:        Loss=70.28,    Training Accuracy=25.0%
iter 400:        Loss=48.78,    Training Accuracy=22.0%
---------------------------------------------------------
Epoch: 2, validation loss: 111.18, validation accuracy: 22.0%
---------------------------------------------------------
Training epoch: 3
iter   0:        Loss=124.59,   Training Accuracy=29.0%
iter 100:        Loss=110.83,   Training Accuracy=23.0%
iter 200:        Loss=75.72,    Training Accuracy=29.0%
iter 300:        Loss=37.26,    Training Accuracy=31.0%
iter 400:        Loss=61.93,    Training Accuracy=26.0%
---------------------------------------------------------
Epoch: 3, validation loss: 80.99, validation accuracy: 23.8%
---------------------------------------------------------
```

...

```
------------------------------------------------------------
Training epoch: 9
iter   0:        Loss=88.60,    Training Accuracy=31.0%
iter 100:        Loss=66.57,    Training Accuracy=25.0%
iter 200:        Loss=37.35,    Training Accuracy=34.0%
iter 300:        Loss=42.39,    Training Accuracy=31.0%
iter 400:        Loss=41.41,    Training Accuracy=31.0%
------------------------------------------------------------
Epoch: 9, validation loss: 50.00, validation accuracy: 26.1%
------------------------------------------------------------
Training epoch: 10
iter   0:        Loss=54.90,    Training Accuracy=32.0%
iter 100:        Loss=53.75,    Training Accuracy=28.0%
iter 200:        Loss=63.55,    Training Accuracy=32.0%
iter 300:        Loss=73.99,    Training Accuracy=24.0%
iter 400:        Loss=47.81,    Training Accuracy=27.0%
------------------------------------------------------------
Epoch: 10, validation loss: 47.27, validation accuracy: 28.9%
------------------------------------------------------------
------------------------------------------------------------
Test loss: 47.27, test accuracy: 28.9%
------------------------------------------------------------
```







Από το Dataset CIFAR10 έγινε χρήση και των 50.000 δειγμάτων εκπαίδευσης και των 10.000 δειγμάτων ελέγχου. Το τελικό κόστος ελέγχου είναι 47,27 ενώ το test accuracy είναι 28,9%, μια όχι τόσο καλή τιμή. Η πολύ καλύτερη απόδοση στο Dataset MNIST από ότι στο Dataset CIFAR10 οφείλεται στο γεγονός ότι οι εικόνες του Dataset MNIST είναι απλούστερες από αυτές του CIFAR10 και ασπρόμαυρες, οπότε πιο εύκολα διαχωρίσιμες.

2.3 Εφαρμογή Tensorflow σε Clustering

2.3.1 Γενικά

Στη ενότητα αυτή θα ασχοληθούμε με την Μέθοδο Εξόρυξης Δεδομένων της Συσταδοποίησης (Clustering) και συγκεκριμένα θα δούμε τις μεθόδους K-means, Self-Organizing Map και Subtractive clustering.

2.3.2 K-means

Ο συγκεκριμένος αλγόριθμος είναι από τους πιο πολυεφαρμοσμένους και είναι η ρίζα για πολλούς άλλους. Ανήκει στην κατηγορία της επίπεδης συσταδοποίησης διότι παράγει ένα σύνολο συσταδοποιήσεων οι οποίες δεν έχουν κάποια ιδιαίτερη δομή-σχέση μεταξύ τους. Ο αλγόριθμος έχει ως στόχο τη βελτιστοποίηση μίας συνάρτησης – της συνάρτησης κόστους. Αρχικά έχουμε K-ομάδες, με την κάθε ομάδα να αντιπροσωπεύεται από το μέσο διάνυσμα. Ο αλγόριθμος λειτουργεί σε δύο βήματα:

1. Για κάθε πρότυπο x_i, που θέλουμε να ομαδοποιήσουμε, υπολογίζουμε το κοντινότερο κέντρο K_j και αναθέτουμε το x_i στην συστάδα K_j.
2. Στη συνέχεια για κάθε συστάδα $K_1, K_2, \cdots, K_n$ υπολογίζομε τα νέα γεωμετρικά τους κέντρα ως τους μέσους όρους κάθε συστάδας με όλα τα σημεία x_i που προκύψαν από το προηγούμενο βήμα.

Σταματάμε την διαδικασία όταν δεν έχουμε μεταβολές στις συστάδες, δηλαδή τα στοιχεία τους παραμένουν ίδια.

Η συνάρτηση που θα υλοποιεί συσταδοποίηση με τον αλγόριθμο K-means σε ένα dataset είναι η ακόλουθη:

k_means(clusters_n, iteration_n, X)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **clusters_n:** αριθμός συστάδων
- **iteration_n:** αριθμός επαναλήψεων
- **X:** χαρακτηριστικά προτύπων

ενώ έχουμε την ακόλουθη παράμετρο εξόδου:

- **centroid_values:** με τις συντεταγμένες των κέντρων των συστάδων που υπολογίστηκαν από την συνάρτηση

Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά δημιουργώ το constant **points** με τα χαρακτηριστικά των προτύπων.

```
def k_means(clusters_n, iteration_n, X):

    points = tf.constant(X)
    print(points)
```

Στην συνέχεια αρχικά ανακατεύω τα δεδομένα του **points** και παίρνω τα K πρώτα πρότυπα τα οποία και θεωρώ ως αρχικά κέντρα των ομάδων και τα αποθηκεύω στο variable **centroids**.

```
    centroids = tf.Variable(tf.slice(tf.random_shuffle(points), [0, 0],
[clusters_n, -1]))
```

Στην συνέχεια κάνουμε επέκταση των tensors **points** και **centroids** κατά μια διάσταση ώστε να έχουν τις απαραίτητες διαστάσεις για να κάνουμε τους επόμενους υπολογισμούς για την εύρεση της απόσταση μεταξύ των προτύπων και των επιλεγμένων Κέντρων (αφού ο **points** είναι (N,2) ενώ ο **centroids** είναι (3,2)).

```
    points_expanded = tf.expand_dims(points, 0)
    centroids_expanded = tf.expand_dims(centroids, 1)
```

Στο tensor **distances** για κάθε πρότυπο θα υπολογίζουμε την απόσταση για τα τρία κέντρα που έχουμε επιλέξει, και στο **assignments** θα καταχωρούμε το δείκτη στο οποίο υπάρχει η μικρότερη απόσταση κάνοντας χρήση του operation **tf.argmin**, επομένως εδώ θα αποθηκεύονται σε ποιο Κέντρο ανήκει το κάθε πρότυπο.

```
    distances = tf.reduce_sum(tf.square(tf.subtract(points_expanded,
centroids_expanded)), 2)
    assignments = tf.argmin(distances, 0)
```

Στην συνέχεια ορίζουμε την λίστα **means** στην οποία αποθηκεύουμε τα νέα γεωμετρικά κέντρα ως τους μέσους όρους κάθε συστάδας που υπολογίσαμε παραπάνω.

```
    means = []
    for c in range(clusters_n):
```

```
        means.append(tf.reduce_mean(
          tf.gather(points,
                    tf.reshape(
                      tf.where(
                        tf.equal(assignments, c)
                      ),[1,-1])
                   ),reduction_indices=[1]))
```

Στην συνέχεια δημιουργούμε το tensor **new_centroids** από την συνένωση των τιμών της λίστας **means** (με χρήση του operation tf.concat) και ενημερώνουμε το tensor **centroids** (με χρήση του operation tf.assign)

```
    new_centroids = tf.concat(means, 0)

    update_centroids = tf.assign(centroids, new_centroids)
```

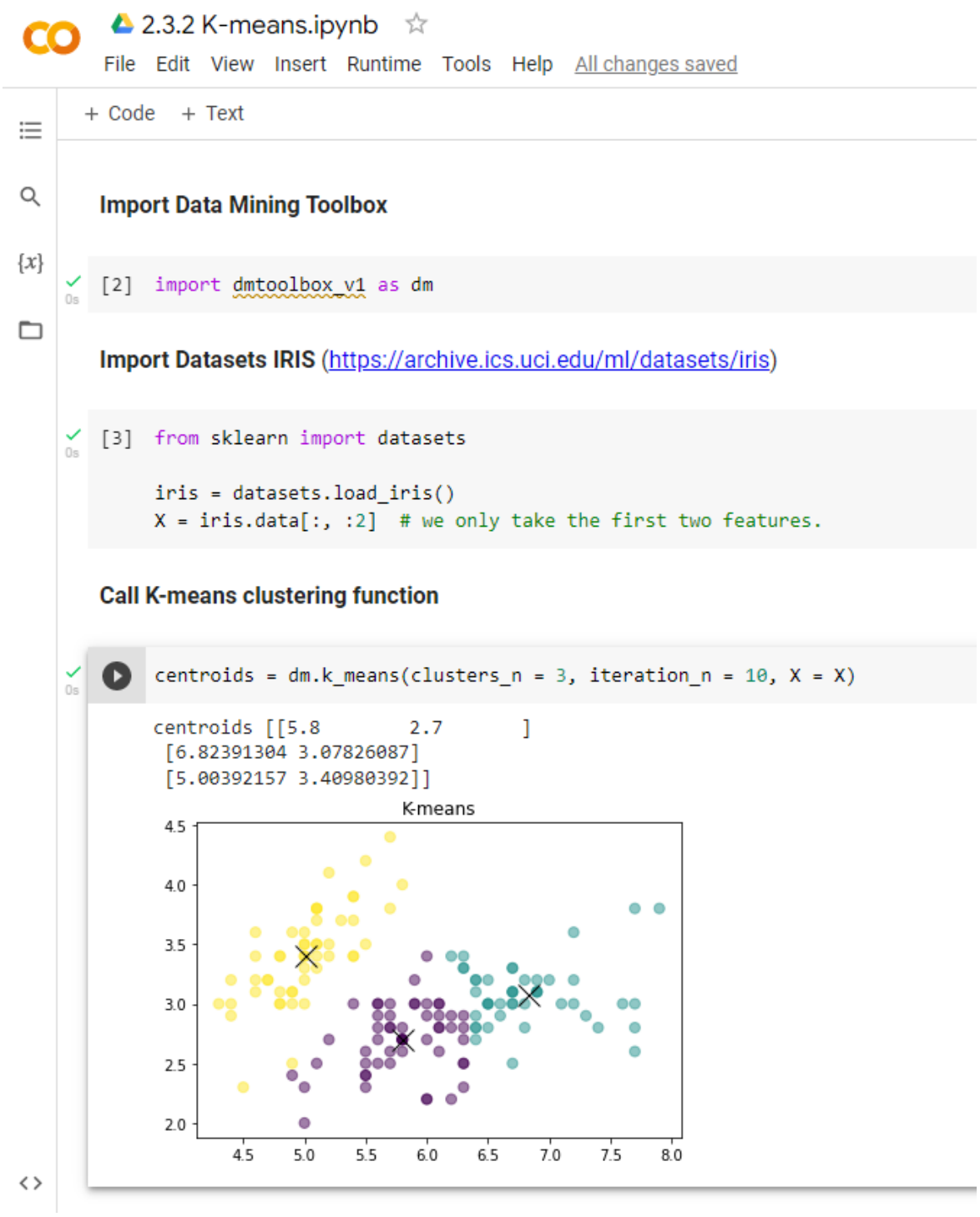
Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για το τρέξιμο του αλγορίθμου.

```
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
      sess.run(init)
      for step in range(iteration_n):
        [_, centroid_values, points_values, assignment_values] =
sess.run([update_centroids, centroids, points, assignments])
```

Αφού τελειώσει το session εκτυπώνουμε τις συντεταγμένες των Κέντρων που υπολογίσαμε, κάνουμε την γραφική παράσταση των δεδομένων μας και των Κέντρων των συστάδων και επιστρέφουμε τις συντεταγμένες αυτών.

```
      print("centroids", centroid_values)

    plt.scatter(points_values[:, 0], points_values[:, 1], c=assignment_values,
s=50, alpha=0.5)
    plt.plot(centroid_values[:, 0], centroid_values[:, 1], 'kx', markersize=15)
    plt.show()
    return centroid_values
```

Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.3.2 K-means.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris (Παράρτημα Γ).



Από το Dataset Iris χρησιμοποιήσαμε και τις τρεις κατηγορίες των λουλουδιών Iris και τα δύο πρώτα χαρακτηριστικά τους. Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 50 δείγματα της κάθε κατηγορίας. Οι θέσεις των τριών κέντρων των συστάδων που υπολογίστηκαν είναι:

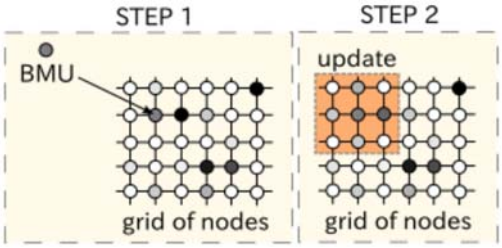
centroids [[5,8 2,7] [6,82391304 3,07826087] [5,00392157 3,40980392]], ενώ από την γραφική παράσταση βλέπουμε ότι τα κέντρα των τριών συστάδων είναι σωστά τοποθετημένα.

2.3.3 Self-organizing map (Αυτό-Οργανωμένος Χάρτης)

Ο αυτοοργανωμένος χάρτης χαρακτηριστικών του Kohonen (SOM), είναι μια τεχνική συσταδοποίησης και οπτικοποίησης των δεδομένων που βασίζεται σε μια άποψη των νευρωνικών δικτύων αλλά μπορεί να θεωρηθεί ως μια παραλλαγή της συσταδοποίησης βάσει προτύπων. Ο στόχος της τεχνικής DSOM είναι να βρει ένα σύνολο κέντρων βάρους και να εκχωρήσει κάθε αντικείμενο των δεδομένων στο κέντρο βάρους, που παρέχει την μεγαλύτερη εγγύτητα αυτού του αντικειμένου. Ένα ξεχωριστό χαρακτηριστικό του αλγόριθμου SOM, είναι ότι επιβάλει μια τοπογραφική (χωρική) οργάνωση των κέντρων βάρους (Tan, Steinbach, Karpatne, & Kumar, 2020).

Ο αλγόριθμος SOM λειτουργεί ως εξής (Shukla, 2017):

- Αρχικά σχεδιάζουμε ένα πλέγμα κόμβων, όπου κάθε κόμβος έχει ένα διάνυσμα βάρους της ίδιας διάστασης με ένα δεδομένο. Τα βάρη κάθε κόμβου αρχικοποιούνται σε τυχαίους αριθμούς, συνήθως από μια τυπική κανονική κατανομή.
- Στη συνέχεια εμφανίζουμε τα δεδομένα στο δίκτυο ένα προς ένα. Για κάθε δεδομένο, το δίκτυο προσδιορίζει τον κόμβο του οποίου το διάνυσμα βάρους ταιριάζει πιο κοντά σε αυτό. Αυτός ο κόμβος ονομάζεται καλύτερη μονάδα αντιστοίχισης (Best Matching Unit – BMU).
- Αφού το δίκτυο αναγνωρίσει το BMU, όλοι οι γείτονες του BMU ενημερώνονται έτσι ώστε τα διανύσματα βάρους τους να πλησιάζουν την τιμή του BMU. Οι πιο κοντινοί κόμβοι επηρεάζονται πιο έντονα από τους κόμβους που βρίσκονται πιο μακριά. Επιπλέον, ο αριθμός των γειτόνων γύρω από ένα BMU συρρικνώνεται με την πάροδο του χρόνου με ρυθμό που καθορίζεται συνήθως από δοκιμή και σφάλμα. Στο παρακάτω σχήμα φαίνεται η οπτικοποίησης του αλγόριθμου SOM.



Εικόνα 2-1 Οπτικοποίησης αλγορίθμου SOM (Shukla, 2017)

Αν και ο αλγόριθμος SOM μπορεί να χρειαστεί περισσότερο χρόνο για να συγκλίνει από αυτό των K-means, η προσέγγιση SOM δεν κάνει υποθέσεις σχετικά με τον αριθμό των συστάδων.

Για την χρήση του αλγορίθμου SOM δημιουργήσαμε την κλάση **SOM** με τις ακόλουθες μεθόδους:

- Μέθοδος **SOM(width, height, dim)** για την δημιουργία ενός αντικειμένου SOM όπου ο χρήστης δίδει το πλάτος, ύψος και την διάσταση του αυτοοργανωμένου χάρτη.
- Μέθοδος **get_bmu_loc(x)**, για την εύρεση του κόμβου με το καλύτερο ταίριασμα (BMU) για το δεδομένο x.
- Μέθοδος **get_propagation(bmu_loc, x, iter)**, για την ενημέρωση των βαρών των γειτονικών κόμβων του BMU, δίνοντας ως είσοδο το BMU, το δεδομένο x και την απόσταση της γειτονιάς.
- Μέθοδος **train(data)** όπου θα δέχεται τα δεδομένα και θα δημιουργεί τον αυτοοργανωμένο χάρτη.

Ακολουθεί αναλυτικά ο κώδικας:

Αρχικά ορίζουμε τον κατασκευαστεί της κλάσης με δεδομένα εισόδου το ύψος και το πλάτος του χάρτη (πρόκειται για ένα χάρτη δύο διαστάσεων) καθώς και την διάσταση των δεδομένων που θα συσταδοποιηθούν. Στην μέθοδο αυτή χρησιμοποιούμε τις ακόλουθες δύο μεθόδους **get_bmu_loc** και **get_propagation**, οι οποίες η πρώτη βρίσκει το κόμβο με το καλύτερο ταίριασμα για ένα δεδομένο και η δεύτερη ενημερώνει τα βάρη των γειτονικών κόμβων.

```
# SOM class
class SOM:
    def __init__(self, width, height, dim):
        self.num_iters = 100
```

```
        self.width = width
        self.height = height
        self.dim = dim
        self.node_locs = self.get_locs()

        # Each node is a vector of dimension `dim`
        # For a 2D grid, there are `width * height` nodes
        nodes = tf.Variable(tf.random_normal([width*height, dim]))
        self.nodes = nodes

        # These two ops are inputs at each iteration
        x = tf.placeholder(tf.float32, [dim])
        iter = tf.placeholder(tf.float32)

        self.x = x
        self.iter = iter

        # Find the node that matches closest to the input
        bmu_loc = self.get_bmu_loc(x)

        self.propagate_nodes = self.get_propagation(bmu_loc, x, iter)
```

Ο ορισμός των δύο αυτών μεθόδων δίνεται παρακάτω. Η απόσταση της γειτονίας (**neigh_factor**) δίδεται από την παρακάτω σχέση:

$$f_{ij}(g,h,\sigma_t) = e^{\frac{-d\left((i,j),(g,h)\right)^2}{2\sigma_t^2}}$$

όπου $d\big((i,j),(g,h)\big)$ η απόσταση μεταξύ των συντεταγμένων (i,j) του δεδομένου και των συντεταγμένων (g,h) του BMU και σ_t (**sigma**) είναι η ακτίνα για την εποχή t.

```
    def get_propagation(self, bmu_loc, x, iter):
        num_nodes = self.width * self.height
        rate = 1.0 - tf.div(iter, self.num_iters)
        alpha = rate * 0.5
        sigma = rate * tf.to_float(tf.maximum(self.width, self.height)) / 2.
        expanded_bmu_loc = tf.expand_dims(tf.to_float(bmu_loc), 0)
        sqr_dists_from_bmu = tf.reduce_sum(tf.square(tf.sub(expanded_bmu_loc,
self.node_locs)), 1)
        neigh_factor = tf.exp(-tf.div(sqr_dists_from_bmu, 2 *
tf.square(sigma)))
        rate = tf.mul(alpha, neigh_factor)
```

```
        rate_factor = tf.pack([tf.tile(tf.slice(rate, [i], [1]), [self.dim])
for i in range(num_nodes)])
        nodes_diff = tf.mul(rate_factor, tf.sub(tf.pack([x for i in
range(num_nodes)]), self.nodes))
        update_nodes = tf.add(self.nodes, nodes_diff)
        return tf.assign(self.nodes, update_nodes)

    def get_bmu_loc(self, x):
        expanded_x = tf.expand_dims(x, 0)
        sqr_diff = tf.square(tf.sub(expanded_x, self.nodes))
        dists = tf.reduce_sum(sqr_diff, 1)
        bmu_idx = tf.argmin(dists, 0)
        bmu_loc = tf.pack([tf.mod(bmu_idx, self.width), tf.div(bmu_idx,
self.width)])
        return bmu_loc
```

Επίσης χρησιμοποιούμε και την βοηθητική μέθοδο get_locs για τη δημιουργία μια λίστας με τις θέσεις $(x. y)$ σε όλους τους κόμβους του πλέγματος.

```
    def get_locs(self):
        locs = [[x, y]
                for y in range(self.height)
                for x in range(self.width)]
        return tf.to_float(locs)
```

Τέλος ορίζουμε την μέθοδο **train** όπου ξεκινάμε την έναρξη του Tensorflow Session για το τρέξιμο του αλγορίθμου SOM με τα δεδομένα που έχει δώσει ο χρήστης στη μεταβλητή data.

```
    def train(self, data):
        with tf.Session() as sess:
            sess.run(tf.initialize_all_variables())
            for i in range(self.num_iters):
                for data_x in data:
                    sess.run(self.propagate_nodes, feed_dict={self.x: data_x,
self.iter: i})
            centroid_grid = [[] for i in range(self.width)]
            self.nodes_val = list(sess.run(self.nodes))
            self.locs_val = list(sess.run(self.node_locs))
            for i, l in enumerate(self.locs_val):
                centroid_grid[int(l[0])].append(self.nodes_val[i])
            self.centroid_grid = centroid_grid
```

Για την εφαρμογή των παραπάνω συναρτήσεων εκτελούμε το Notebook «2.3.3 Self-organizing map.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα.

2.3.3 Self-organizing map.ipynb

File Edit View Insert Runtime Tools Help All changes saved

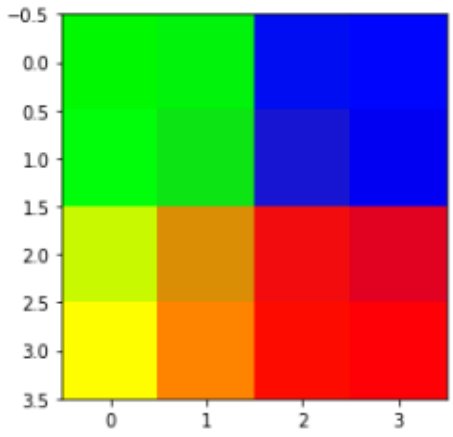
+ Code + Text

Import Data Mining Toolbox

```
[2]  import dmtoolbox_v1 as dm
```

Self-organizing map

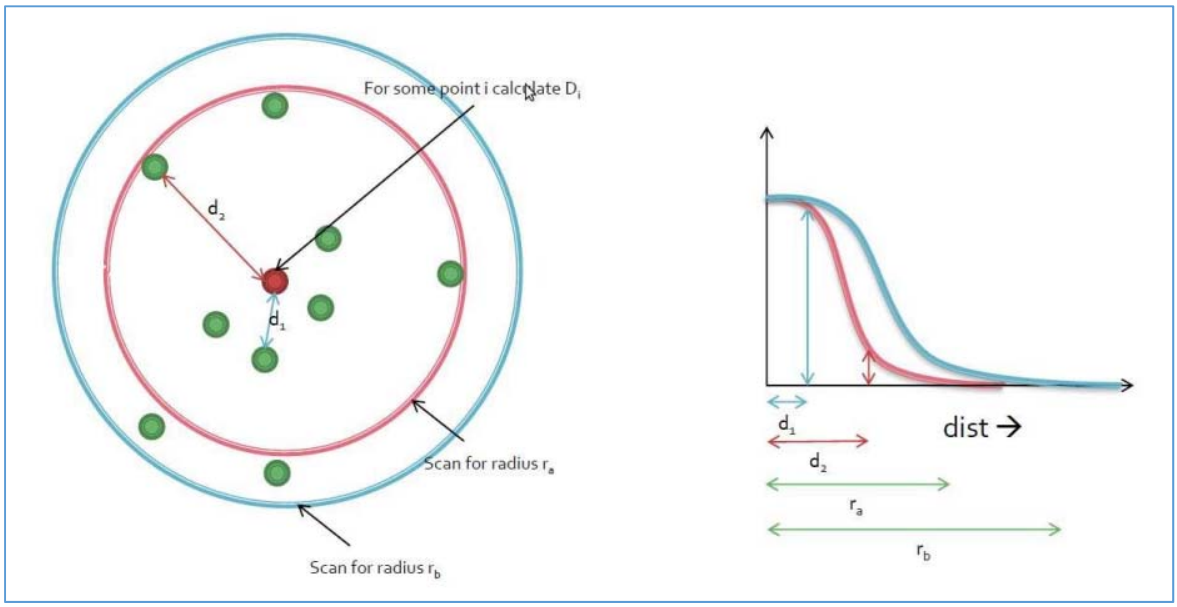
```
from dmtoolbox_v1 import SOM
import numpy as np
import matplotlib.pyplot as plt
colors = np.array(
     [[0., 0., 1.],
      [0., 0., 0.95],
      [0., 0.05, 1.],
      [0., 1., 0.],
      [0., 0.95, 0.],
      [0., 1, 0.05],
      [1., 0., 0.],
      [1., 0.05, 0.],
      [1., 0., 0.05],
      [1., 1., 0.]])

som = SOM(4, 4, 3)
som.train(colors)
plt.imshow(som.centroid_grid)
plt.show()
```



Συγκεκριμένα χρησιμοποιήσαμε ως δεδομένα εισόδου μια λίστα διανυσμάτων τριών διαστάσεων, η οποία αποθηκεύεται στο πίνακα **colors**. Στη συνέχεια δημιουργήσαμε ένα αντικείμενο SOM με πλέγμα 4 επί 4 και με δεδομένα τριών διαστάσεων. Με την εκπαίδευση ο αλγόριθμος SOM μαθαίνει τις συστάδες των δεδομένων, οπότε αποθηκεύσαμε το SOM στη λίστα **centroid_grid** και το εμφανίσαμε ως γραφική απεικόνιση με χρήση της function imshow της βιβλιοθήκης matplotlib. Από τη γραφική απεικόνιση παρατηρούμε ότι για το συγκεκριμένο παράδειγμα έχουν δημιουργηθεί τέσσερις συστάδες.

2.3.4 Subtractive clustering (Αφαιρετική συσταδοποίηση)

Η αφαιρετική συσταδοποίηση (subtractive clustering) (Liang, Yupu, & Yong, 2008) αποτελεί μια επέκταση της μεθόδους συσταδοποίησης με βάση τη μέθοδο συνάρτησης πυκνότητας Mountain Clustering (Yager & Filev, 1994), όπου το υπολογιστικό πρόβλημα της μεθόδου αυτής λύνεται χρησιμοποιώντας ως υποψήφια κέντρα των συστάδων μόνο τα σημεία των δεδομένων αντί να εξεταστούν όλα τα σημεία του πλέγματος. Με αυτή την τροποποίηση, το υπολογιστικό κόστος είναι ανάλογο με το μέγεθος του προβλήματος και δεν αυξάνεται εκθετικά με τη διάσταση το προβλήματος όπως συμβαίνει με την μέθοδο Mountain Clustering. Παρόλο που τα πιθανά κέντρα των συστάδων περιορίζονται στα σημεία των δεδομένων, η μέθοδος μπορεί να θεωρηθεί αρκετά καλή λαμβάνοντας υπόψη το μειωμένο υπολογιστικό κόστος της.



Εικόνα 2-2 Αφαιρετική συσταδοποίηση (Κουτσούκος, 2016)

Ο αλγόριθμος έχει τα ακόλουθα βήματα:

Αρχικά υπολογίζεται η πυκνότητα για κάθε σημείο των δεδομένων, αφού όπως είπαμε παραπάνω, αποτελεί πιθανό κέντρο των συστάδων. Η συνάρτηση πυκνότητας που χρησιμοποιείται είναι η ακόλουθη:

$$D_i = \sum_{j=1}^{n} exp\left(-\frac{\|x_i - x_j\|^2}{\left(\frac{r_a}{2}\right)^2}\right)$$ Εξίσωση 10

όπου r_a είναι μια ακτίνα γειτονίας.

Αφού υπολογιστεί η πυκνότητα για όλα τα σημεία επιλέγεται το σημείο με την μεγαλύτερη πυκνότητα ως το κέντρο της πρώτης συστάδας, αφού η υψηλή πυκνότητα σημαίνει ότι ένα σημείο που έχει πολλούς γείτονες. Το πρώτο αυτό κέντρο των συστάδων x_{c_i} χρησιμοποιείται στη συνέχεια για τον επανυπολογισμό της πυκνότητας για τα άλλα σημεία x_i με βάση τον ακόλουθο τύπο:

$$D_i = D_i - D_{c_i} \cdot exp\left(-\frac{\|x_i - x_{c_i}\|^2}{\left(\frac{r_b}{2}\right)^2}\right)$$ Εξίσωση 11

όπου r_b είμαι μια θετική σταθερά, που ορίζει μια γειτονιά στην οποία θα μειωθεί η συνάρτηση πυκνότητας. Σαν αποτέλεσμα, κάθε κέντρο μιας συστάδας θα έχει σημαντικά μειωμένη τιμή της συνάρτησης πυκνότητας. Μετά από αυτό το βήμα, επιλέγεται ως επόμενο κέντρο αυτό που έχει τη μεγαλύτερη τιμή της ανανεωμένης συνάρτησης πυκνότητας και αυτή η διαδικασία συνεχίζεται μέχρι να φτάσουμε τον επιθυμητό αριθμών συστάδων (Chiu, 1994) (Κουτσούκος, 2016).

Η συνάρτηση που θα υλοποιεί συσταδοποίηση με τον αφαιρετικό αλγόριθμο σε ένα dataset είναι η ακόλουθη:

subtractive_clustering(data, ra, rb, dim, points_n, clusters_n, iteration_n)

και η οποία έχει τις ακόλουθες παραμέτρους εισόδου:

- **data**: χαρακτηριστικά προτύπων.
- **ra**: ακτίνα γειτονίας, της εξίσωσης 10
- **rb**: σταθερά της εξίσωσης 11 για το υπολογισμό της πυκνότητας.
- **dim**: διάσταση των χαρακτηριστικών των προτύπων.
- **points_n**: αριθμός προτύπων.

- **clusters_n**: αριθμός συστάδων.
- **iteration_n**: αριθμός επαναλήψεων.

ενώ έχουμε την ακόλουθη παράμετρο εξόδου:

- **centro_new:** με τις συντεταγμένες των κέντρων των συστάδων που υπολογίστηκαν από την συνάρτηση

Ακολουθεί αναλυτικά ο κώδικας της συνάρτησης:

Αρχικά ορίζουμε τις μεταβλητές που θα χρησιμοποιήσουμε.

```
# Subtractive clustering
def subtractive_clustering(data, ra, rb, dim, points_n, clusters_n,
iteration_n):
    points = tf.constant(data)
    i = tf.constant(data)
    j = tf.constant(data)
    di = tf.Variable(tf.zeros([points_n,1], dtype=np.float64))
    centroid = []
    centroid_density = []
```

Στη συνέχεια ορίζουμε την συνάρτηση **find_density** για το υπολογισμό της πυκνότητας ενός σημείου.

```
    def find_density(xi, xj, step, di):
      i_expanded = tf.expand_dims(xi, 0)
      j_expanded = tf.expand_dims(xj, 1)
      if step == 0:
        density =tf.exp(tf.scalar_mul(-1/((ra/2)**2) ,
tf.reduce_sum(tf.pow(tf.subtract(i_expanded, j_expanded),2),2, keepdims=True)))
        density = tf.reduce_sum(density,1)
        position_density = tf.reduce_max(density)
        position = tf.argmax(density, 0)
        di = tf.assign(di, density)
      else:
        xc_expanded = tf.repeat(centroid[step-1],repeats=[points_n], axis=0)
        density =tf.exp(tf.scalar_mul(-1/((rb/2)**2) ,
tf.reduce_sum(tf.pow(tf.subtract(xi, xc_expanded),2),1, keepdims=True)))
        density = tf.scalar_mul(centroid_density[step-1], density)
        density = tf.subtract(di,density)
        position_density = tf.reduce_max(density)
        position = tf.argmax(density, 0)
        di = tf.assign(di, density)
      position = tf.squeeze(position)
      return position, position_density, di
```

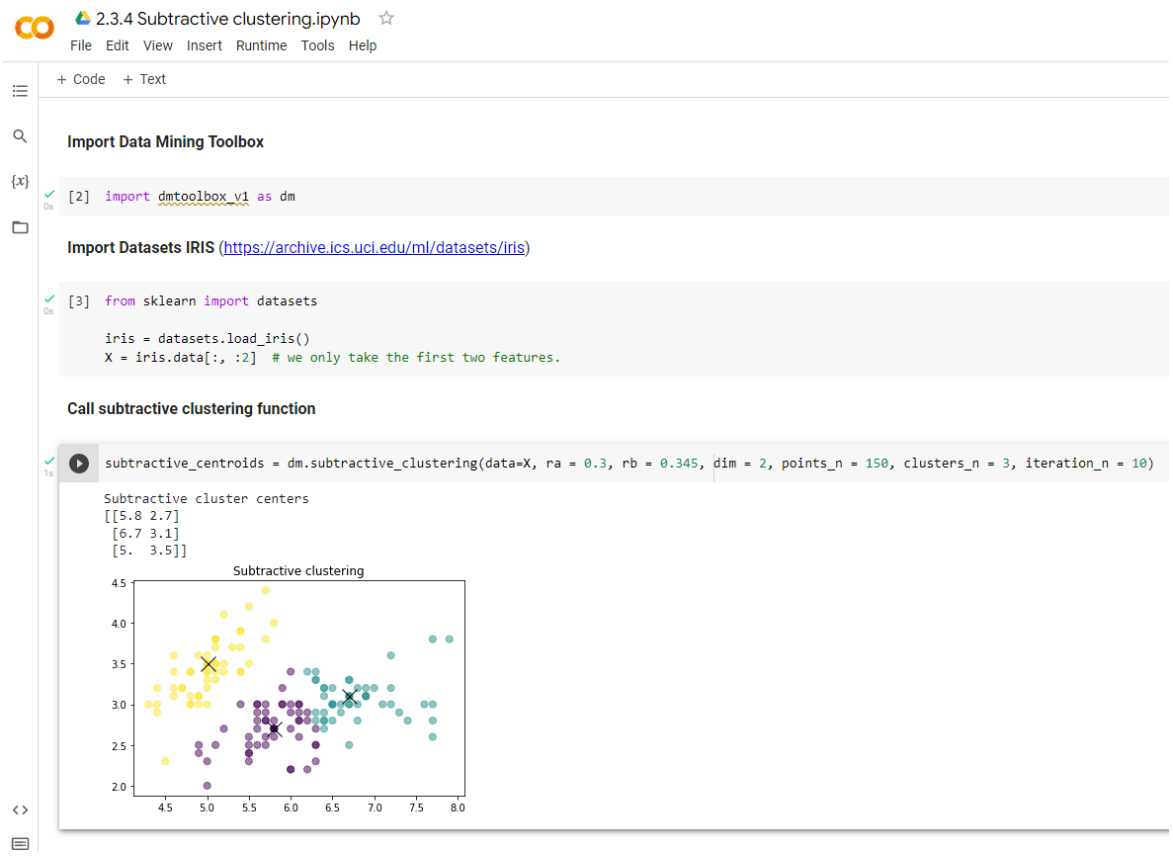
Ακολούθως ξεκινάμε με την έναρξη του Tensorflow Session για το τρέξιμο του αλγορίθμου.

```
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
          init = tf.global_variables_initializer()
          sess.run(init)
          for step in range(clusters_n):
            [position, position_density, di] = find_density(i, j, step, di)
            centroid.append(i[position:position+1, :])
            centroid_density.append(position_density)
          centro = tf.concat(centroid,0)
          centro = tf.reshape(centro, shape=(clusters_n,dim))
          centro_expanded = tf.expand_dims(centro, 1)
          points_expanded = tf.expand_dims(points, 0)
          distances = tf.reduce_sum(tf.square(tf.subtract(points_expanded,
centro_expanded)), 2)
          assignments = tf.argmin(distances, 0)
          assignments_values = assignments.eval() #tensor to numpy.ndarray
          centro_new = centro.eval() #tensor to numpy.ndarray
```

Αφού τελειώσει το session εκτυπώνουμε τις συντεταγμένες των Κέντρων των συστάδων που βρήκαμε, κάνουμε την γραφική παράσταση των δεδομένων μας και των Κέντρων αυτών και επιστρέφουμε τις συντεταγμένες αυτών.

```
    print('Subtractive cluster centers')
    print(centro_new)
    plt.scatter(data[:, 0], data[:, 1], c=assignments_values, s=50, alpha=0.5)
    plt.plot(centro_new[:, 0], centro_new[:, 1], 'kx', markersize=15)
    plt.title('Subtractive clustering')
    plt.show()

    return centro_new
```

Για την εφαρμογή της παραπάνω συνάρτησης εκτελούμε το Notebook «2.3.4 Subtractive clustering.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset Iris (Παράρτημα Γ).



Από το Dataset Iris χρησιμοποιήσαμε και τις τρεις κατηγορίες των λουλουδιών Iris και τα δύο πρώτα χαρακτηριστικά τους. Όσο αφορά τον αριθμό των δειγμάτων χρησιμοποιήσαμε και τα 50 δείγματα της κάθε κατηγορίας. Οι θέσεις των τριών κέντρων των συστάδων που υπολογίστηκαν είναι τα ακόλουθα και τα οποία αποτελούν σημεία από τα δείγματα:

[[5,8 2,7] [6,7 3,1][5,0 3,5]]

Από την γραφική παράσταση βλέπουμε ότι τα κέντρα των τριών συστάδων είναι σωστά τοποθετημένα και αρκετά κοντά με αυτά που υπολογίσθηκαν με την μέθοδο K-means στην ενότητα 2.3.2 ([[5,8 2,7] [6,82391304 3,07826087] [5,00392157 3,40980392])

2.4 Εφαρμογή Tensorflow σε Autoencoders

2.4.1 Γενικά

Στη ενότητα αυτή θα ασχοληθούμε με την Μέθοδο Εξόρυξης Δεδομένων με χρήση των Τεχνητών Νευρωνικών Δικτύων (ΤΝΔ) και συγκεκριμένα με τους Αυτοκωδικοποιητές (Autoencoders) που είδαμε στην ενότητα 1.1.2.4. . Για την συγκεκριμένη μέθοδο εξόρυξης δεδομένων κρίθηκε καλύτερη η χρήση του αντικειμενοστραφούς προγραμματισμού (object-oriented programming) από τον διαδικαστικό προγραμματισμό (procedural programming) που χρησιμοποιήσαμε στις προηγούμενες μεθόδους.

2.4.2 Κλάση Autoencoders

Για την χρήση των Autoencoders δημιουργήσαμε την κλάση **Autoencoder** με τις ακόλουθες μεθόδους:

- Ο κατασκευαστής της κλάσης **Autoencoder(input_dim, hidden_dim, epoch, batch_size, learning_rate)** με τις ακόλουθες παραμέτρους εισόδου:
  - **input_dim**: διάσταση εισόδου του ΤΝΔ
  - **hidden_dim**: διάσταση κρυφού επιπέδου ΤΝΔ
  - **epoch**: σύνολο εποχών εκμάθησης του μοντέλου
  - **batch_size**: μέγεθος παρτίδας εκμάθησης του μοντέλου
  - **learning_rate**: ρυθμός εκμάθησης του μοντέλου
- Μέθοδος **train(data)** η οποία θα δέχεται δεδομένα και θα εκπαιδεύει το δίκτυο του ΤΝΔ.
- Μέθοδος **test(data)** η οποία θα δέχεται ένα διάνυσμα και το οποίο θα το κάνει αρχικά compressed και στην συνέχεια reconstructed. Η μέθοδος θα επιστρέφει το ανακατασκευασμένο διάνυσμα.

Ακολουθεί αναλυτικά ο κώδικας:

Αρχικά έχουμε τον ορισμό του κατασκευαστή της κλάσης ***_init_***, στο οποίο ορίζουμε τις variables, placeholders, optimizers και τους operators, καθώς ότι άλλο που δεν χρειάζεται session.

```
class Autoencoder:
    def __init__(self, input_dim, hidden_dim, epoch, batch_size,
learning_rate):
```

```
        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate

        # Define input placeholder
        x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim])
```

Επειδή θα έχουμε δύο σετ weights και bias (ένα για το βήμα της κωδικοποίησης και ένα για το βήμα της αποκωδικοποίησης) κάνουμε χρήση της λειτουργικότητας **tf.name_scope** για να μπορούμε να τα ξεχωρίζουμε.

```
        # Define variables and models
        with tf.name_scope('encode'):
            weights = tf.Variable(tf.random_normal([input_dim, hidden_dim],
dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            encoded = tf.nn.sigmoid(tf.matmul(x, weights) + biases)
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim],
dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            decoded = tf.matmul(encoded, weights) + biases

        self.x = x
        self.encoded = encoded
        self.decoded = decoded
```

Στην συνέχεια ορίζουμε την συνάρτηση κόστους και επιλέγουμε ως αλγόριθμο βελτιστοποίησης για την εκπαίδευση αυτό του Adam.

```
        # Define cost function and training operation
        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
self.decoded))))
        self.all_loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
self.decoded)), 1))
        self.train_op =
tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)
```

Τέλος κάνουμε χρήση της λειτουργικότητας **tf.train.Saver** για την αποθήκευση των παραμέτρων του μοντέλου που δημιουργήσαμε.

```
        # Define a saver operation
        self.saver = tf.train.Saver()
```

Στην συνέχεια ορίζουμε την μέθοδο κλάσης ***train***, η οποία θα δέχεται ένα dataset και τις παραμέτρου εκπαίδευσης για την μείωση της απώλειας. Η εκπαίδευση θα γίνεται με παρτίδες και για αυτό γίνεται χρήση της βοηθητικής συνάρτησης **get_batch** (Παράρτημα Α). Τέλος ανά 50 κύκλους θα γίνεται η εκτύπωση της απώλειας της ανακατασκευής των δεδομένων (μετά από την κωδικοποίηση και αποκωδικοποίηση τους) ενώ τέλος οι παράμετροι του μοντέλου θα αποθηκεύονται στο αρχείο **model.ckpt**.

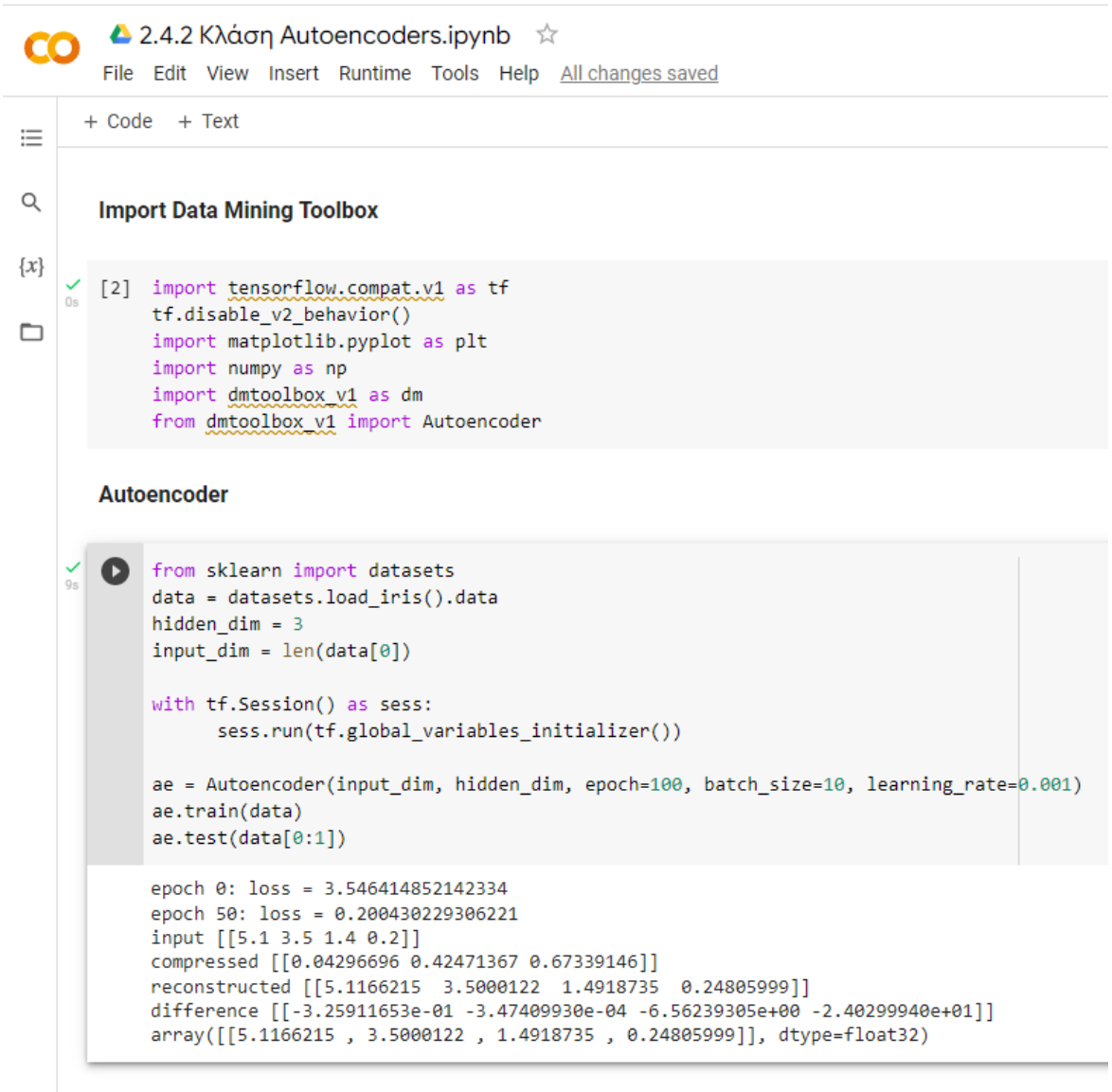
```
    def train(self, data):
        num_samples = len(data)
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            for i in range(self.epoch):
                for j in range(num_samples):
                    batch_data = get_batch(data, self.batch_size)
                    l, _ = sess.run([self.loss, self.train_op],
feed_dict={self.x: batch_data})
                if i % 50 == 0:
                    print('epoch {0}: loss = {1}'.format(i, l))
                    self.saver.save(sess, './model.ckpt')
            self.saver.save(sess, './model.ckpt')
```

Τέλος ορίζουμε την μέθοδο κλάσης ***test***, η οποία θα δέχεται ένα dataset, στην συνέχεια θα το κωδικοποιεί με βάση το μοντέλο της παραμέτρους του οποίου θα διαβάζει από το αρχείο model.ckpt., στο οποίο αποθηκεύτηκαν κατά την εκπαίδευση του με την κλήση της μεθόδου ***train***. Τέλος θα γίνεται η εκτύπωση του αρχικού dataset, του συμπιεσμένου, του ανακατασκευασμένου και τις διαφοράς από το αρχικό. Η μέθοδος θα επιστρέφει το ανακατασκευασμένο dataset.

```
    def test(self, data):
        with tf.Session() as sess:
            self.saver.restore(sess, './model.ckpt')
            hidden, reconstructed = sess.run([self.encoded, self.decoded],
feed_dict={self.x: data})
            difference =
sess.run(tf.multiply(tf.divide(tf.subtract(data,reconstructed),data),100))
        print('input', data)
        print('compressed', hidden)
        print('reconstructed', reconstructed)
        print('difference', difference)
        return reconstructed
```

Για την εφαρμογή της λειτουργικότητας της παραπάνω κλάσης εκτελούμε το Notebook «2.4.2 Κλάση Autoencoders.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα.

Για το συγκεκριμένο παράδειγμα κάναμε χρήση του Iris Dataset (Παράρτημα Γ), όπου όπως γνωρίζουμε έχουμε διανύσματα τεσσάρων διαστάσεων. Τα δεδομένα του dataset αποθηκεύονται στη μεταβλητή data. Οπότε εμείς θα κάνουμε συμπίεση σε τρεις διαστάσεις (hidden_dim = 3). Αρχικά κατασκευάσουμε το αντικείμενο ae της κλάσης Autoencoder με διάσταση εισόδου 4 και διάσταση συμπίεσης 3. Στην συνέχεια κάνουμε εκπαίδευση του δικτύου μας καλώντας τη μέθοδο train πάνω στα δεδομένα data. Τέλος δοκιμάζουμε το μοντέλο μας δοκιμάζοντας να συμπιέσουμε και να ανακατασκευάσουμε το πρώτο διάνυσμα του dataset (data[0:1]).



Παρατηρούμε ότι το διάνυσμα εισόδου [[5,1 3,5 1,4 0,2]] με την συμπίεση και αποσυμπίεση έδωσε ως διάνυσμα εξόδου το [[5,1166215 3,5000122 1,4918735 0,24805999]] μια πολύ καλή προσέγγισης του διανύσματος εισόδου.

2.4.3 Χρήση κλάσης Autoencoders σε συμπίεση εικόνας

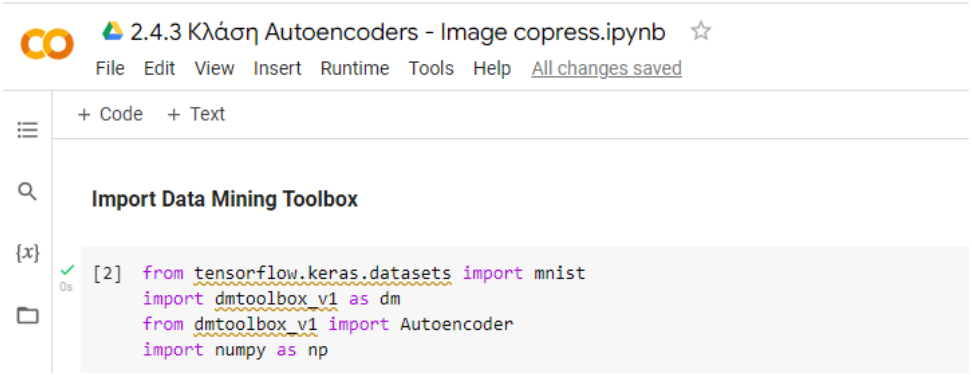
Στην ενότητα αυτή θα κάνουμε χρήση της κλάσης Autoencoders στην συμπίεσης εικόνας και στην συνέχεια στην ανακατασκευή της. Οι μέθοδοι που θα χρησιμοποιήσουμε είναι ακριβώς οι ίδιοι με την προηγούμενη ενότητα, εκτός από την ***test*** την οποία θα αντικαταστήσουμε με την ***compress***. Η μόνη διαφορά των δύο είναι ότι απλά δεν κάνουμε την εκτύπωση των δεδομένων, επιστρέφει μόνο το ανακατασκευασμένο dataset.

```
    def compress(self, data):
          with tf.Session() as sess:
              self.saver.restore(sess, './model.ckpt')
              hidden, reconstructed = sess.run([self.encoded, self.decoded],
feed_dict={self.x: data})
              difference =
sess.run(tf.multiply(tf.divide(tf.subtract(data,reconstructed),data),100))
          return reconstructed
```

Για την εφαρμογή της λειτουργικότητας της παραπάνω κλάσης εκτελούμε το Notebook «2.4.3 Κλάση Autoencoders - Image copress.ipynb» (Παράρτημα Β) οπότε έχουμε τα ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset MNIST (Παράρτημα Γ).

Αρχικά δημιουργούμε ένα αντικείμενο Autoencoder με διαστάσεις εισόδου την διάσταση που έχει μια εικόνα (αφού μια εικόνα είναι 28x28, η διάσταση της είναι 784) και με 600 κρυφές διαστάσεις. Στην συνέχεια εκπαιδεύουμε το αντικείμενο μας με τις 100 πρώτες εικόνες εκπαίδευσης που μας δίδει το συγκεκριμένο dataset.

Από τις εικόνες δοκιμής του Dataset επιλέγουμε 10, τις οποίες αρχικά τις εκτυπώνουμε, στην συνέχεια με χρήση της μεθόδου compress του αντικειμένου Autoencoder της συμπιέζουμε από διάσταση 784 σε 600 και στην συνέχεια την αναδομούμαι και την τελικά την εκτυπώνουμε.

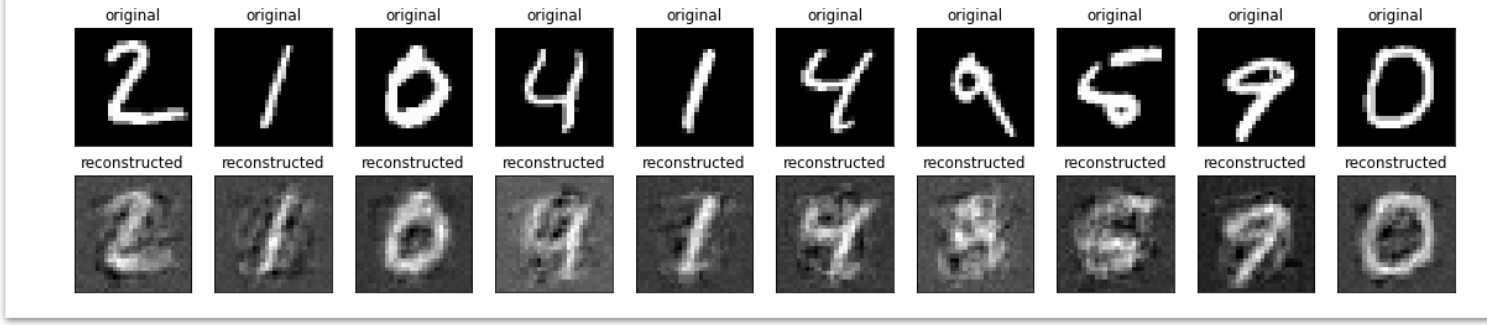



```
(x_train, y_train_old), (x_test, y_test_old) = mnist.load_data()
```

Autoencoders - Image copress

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
hidden_dim = 600
input_dim = len(x_train[0])

ae = Autoencoder(input_dim, hidden_dim, epoch=400, batch_size=10, learning_rate=0.001)
ae.train(x_train[0:100])

n = 10
dm.plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = dm.plt.subplot(2, n, i + 1)
  dm.plt.imshow(x_test[i+1:i+2].reshape(28, 28))
  dm.plt.title("original")
  dm.plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstructed
  ax = dm.plt.subplot(2, n, i + 1 + n)
  img_c = ae.compress(x_test[i+1:i+2])
  dm.plt.imshow(img_c.reshape(28, 28))
  dm.plt.title("reconstructed")
  dm.plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
```

```
epoch 0: loss = 79.02778625488281
epoch 50: loss = 34.01882553100586
epoch 100: loss = 18.21710777282715
epoch 150: loss = 10.095126152038574
epoch 200: loss = 6.325508117675781
epoch 250: loss = 1.5988959074020386
epoch 300: loss = 0.10968347638845444
epoch 350: loss = 0.019155802205204964
```



Από την παραπάνω εμφάνιση των αρχικών εικόνων και την ανακατασκευή, μετά από την συμπίεση και αποσύμπιεση τους, βλέπουμε ότι είναι αρκετά ευκρινείς.

2.4.4 Χρήση κλάσης Autoencoders σε διαγραφή θορύβου σε εικόνα (denoising)

Στην ενότητα αυτή θα ασχοληθούμε με τους Αυτοκωδικοποιητές απαλοιφής θορύβου (Denoising Autoencoder - Αποθορυβοποιητικός Αυτοκωδικοποιητής), και συγκεκριμένα θα κάνουμε χρήση τους, ώστε να κάνουμε διαγραφή θορύβου από μια εικόνα. Ο Denoising Autoencoder (DAE) είναι μια παραλλαγή του Αυτοκωδικοποιητή όπου η είσοδος έχει θόρυβο και η συνάρτηση κόστους προσπαθεί να την αποθορυβοποιήσει. Ο DAE εκπαιδεύεται κάνοντας χρήση του δείγματος εκπαίδευσης και της αλλοιωμένης του μορφής από κάποια συνάρτηση θορύβου.

Για τον DAE δημιουργήσαμε την κλάση **Denoiser** σε αντιστοιχία με την **Autoencoder**, και η οποία έχει τις ακόλουθες μεθόδους:

- Ο κατασκευαστής της κλάσης **Denoiser(input_dim, hidden_dim, epoch, batch_size, learning_rate)** με τις ακόλουθες παραμέτρους εισόδου:
  - **input_dim**: διάσταση εισόδου του ΤΝΔ
  - **hidden_dim**: διάσταση κρυφού επιπέδου ΤΝΔ
  - **epoch**: σύνολο εποχών εκμάθησης του μοντέλου
  - **batch_size**: μέγεθος παρτίδας εκμάθησης του μοντέλου
  - **learning_rate**: ρυθμός εκμάθησης του μοντέλου
- Μέθοδος **add_noise(data)** για την προσθήκη θορύβου στις εικόνες.

- Μέθοδος **train(data)** η οποία θα δέχεται δεδομένα και θα εκπαιδεύει το δίκτυο του ΤΝΔ για την αποθορυβοποίηση ενός διανύσματος (στην περίπτωση μας, μιας εικόνας).
- Μέθοδος **test(data)** η οποία θα δέχεται ένα διάνυσμα (στην περίπτωση μας, μιας εικόνας) με θόρυβο και θα τον αποθορυβοποιεί.

Ακολουθεί αναλυτικά ο κώδικας:

Αρχικά ορίζουμε την μέθοδο του κατασκευαστεί της κλάσης Denoiser.

```
# Autoencoder - Denoiser
class Denoiser:

    def __init__(self, input_dim, hidden_dim, epoch=10000, batch_size=50,
learning_rate=0.001):
        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate

        self.x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim],
name='x')
        self.x_noised = tf.placeholder(dtype=tf.float32, shape=[None,
input_dim], name='x_noised')
        with tf.name_scope('encode'):
            self.weights1 = tf.Variable(tf.random_normal([input_dim,
hidden_dim], dtype=tf.float32), name='weights')
            self.biases1 = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            self.encoded = tf.nn.sigmoid(tf.matmul(self.x_noised,
self.weights1) + self.biases1, name='encoded')
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim],
dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            self.decoded = tf.matmul(self.encoded, weights) + biases
        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
self.decoded))))
        self.train_op =
tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)
        self.saver = tf.train.Saver()
```

Στην συνέχεια ορίζουμε την μέθοδο add_noise για την προσθήκη θορύβου σε μια εικόνα.

```
    def add_noise(self, data):
        noise_type = 'mask-0.2'
        if noise_type == 'gaussian':
            n = np.random.normal(0, 0.1, np.shape(data))
            return data + n
```

```
        if 'mask' in noise_type:
            frac = float(noise_type.split('-')[1])
            temp = np.copy(data)
            for i in temp:
                n = np.random.choice(len(i), round(frac * len(i)),
replace=False)
                i[n] = 0
            return temp
```

Ακολούθως ορίζουμε τις μεθόδου train και test για την εκπαίδευση του ΤΝΔ και την δοκιμή του αντίστοιχα.

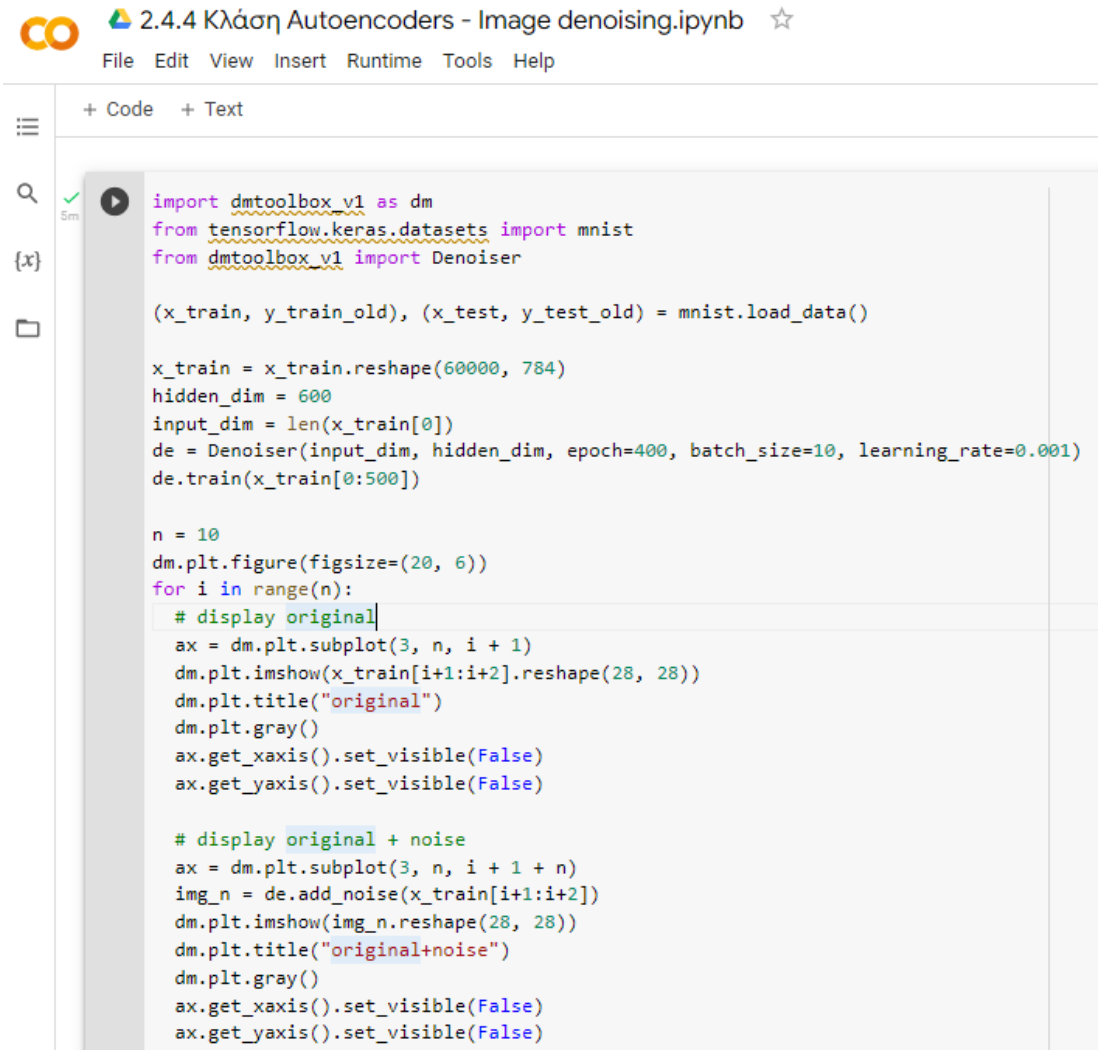
```
    def train(self, data):
        num_samples = len(data)
        data_noised = self.add_noise(data)
        with open('log.csv', 'w') as writer:
            with tf.Session() as sess:
                sess.run(tf.global_variables_initializer())
                for i in range(self.epoch):
                    for j in range(num_samples):
                        batch_data, batch_data_noised = get_batch_dnsr(data,
data_noised, self.batch_size)
                        l, _ = sess.run([self.loss, self.train_op],
feed_dict={self.x: batch_data, self.x_noised: batch_data_noised})
                    if i % 50 == 0:
                        print('epoch {0}: loss = {1}'.format(i, l))
                        self.saver.save(sess, './model.ckpt')
                        epoch_time = int(time.time())
                        row_str = str(epoch_time) + ',' + str(i) + ',' + str(l)
+ '\n'
                        writer.write(row_str)
                        writer.flush()
                self.saver.save(sess, './model.ckpt')

    def test(self, data):
        with tf.Session() as sess:
            self.saver.restore(sess, './model.ckpt')
            hidden, reconstructed = sess.run([self.encoded, self.decoded],
feed_dict={self.x: data, self.x_noised: data})
        return reconstructed
```

Για την εφαρμογή της λειτουργικότητας της παραπάνω κλάσης εκτελούμε το Notebook «2.4.4 Κλάση Autoencoders - Image denoising.ipynb» (Παράρτημα Β) οπότε έχουμε τα

ακόλουθα αποτελέσματα. Για το συγκεκριμένο παράδειγμα χρησιμοποιήσαμε το Dataset MNIST (Παράρτημα Γ).

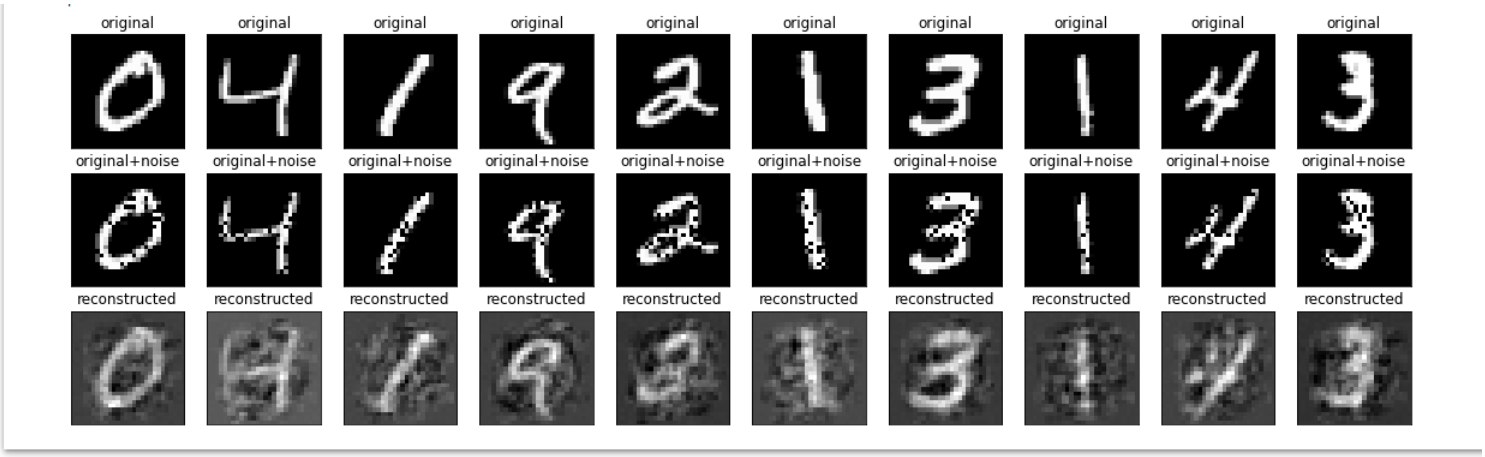
Αρχικά δημιουργούμε ένα αντικείμενο **Denoiser** με διαστάσεις εισόδου την διάσταση που έχει μια εικόνα (αφού μια εικόνα είναι 28x28, η διάσταση της είναι 784) και με 600 κρυφές διαστάσεις. Στην συνέχεια εκπαιδεύουμε το αντικείμενο μας με τις 100 πρώτες εικόνες εκπαίδευσης που μας δίδει το συγκεκριμένο dataset.

Από τις εικόνες δοκιμής του Dataset επιλέγουμε 10, τις οποίες αρχικά τις εκτυπώνουμε, στην συνέχεια τους προσθέτουμε θόρυβο με χρήση της μεθόδου **add_noise** και τις εκτυπώνουμε. Στην συνέχεια με χρήση της μεθόδου **test** του αντικειμένου **Denoiser** την αναδομούμαι χωρίς θόρυβο και την εκτυπώνουμε.



```
# display reconstructed
ax = dm.plt.subplot(3, n, i + 1 + 2*n)
img_new = de.test(img_n)
dm.plt.imshow(img_new.reshape(28, 28))
dm.plt.title("reconstructed")
dm.plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/compat/v2_compat.
Instructions for updating:
non-resource variables are not supported in the long term
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
epoch 0: loss = 64.0228042602539
epoch 50: loss = 28.089649200439453
epoch 100: loss = 20.39529037475586
epoch 150: loss = 19.027568817138672
epoch 200: loss = 16.488405227661133
epoch 250: loss = 14.476308822631836
epoch 300: loss = 13.159516334533691
epoch 350: loss = 12.414178848266602
```



Από την παραπάνω εμφάνιση των αρχικών εικόνων, των εικόνων με θόρυβο και την τελική τους ανακατασκευή, μετά από την αποθορυβοποίηση τους, βλέπουμε ότι είναι αρκετά ευκρινείς.

3 Συμπεράσματα

3.1 Συζήτηση - Συμπεράσματα

Σκοπός της πτυχιακής αυτή εργασίας είναι η χρήση της Μηχανική Μάθησης με χρήση της βιβλιοθήκης Tensorflow σε μια σειρά Μεθόδων Εξόρυξης Δεδομένων.

Ως αποτέλεσμα της πτυχιακής αυτής εργασίας είναι η δημιουργία του Toolbox (dmtoolbox_v1.py) με τις ακόλουθες μεθόδους Εξόρυξης Δεδομένων με χρήση του Tensorflow 1:

Συνάρτηση/Μέθοδος Κλάσης	Δεδομένα Εισόδου	Δεδομένα Εξόδου	Παρατηρήσεις
linear_regression	split ratio x y x_test y_test learning_rate training_epochs	weight bias cost_test	Η συνάρτηση αυτή δέχεται τα δεδομένα X και Y και υπολογίζει το γραμμικό μοντέλο. Τα δεδομένα αυτά μπορούν να γίνονται split σε training και testing data ή να δίδονται ξεχωριστά ως δεδομένα εισόδου. Η συνάρτηση επιστρέφει τις παραμέτρους του μοντέλου και πόσο καλή προσαρμογή έχει γίνει (MSE).
polynomial_regression	split ratio x y x_test y_test learning_rate training_epochs num_coeffs	weight cost_test	Η συνάρτηση αυτή δέχεται τα δεδομένα X και Y και υπολογίζει το πολυωνυμικό μοντέλο. Τα δεδομένα αυτά μπορούν να γίνονται split σε training και testing data ή να δίδονται ξεχωριστά ως δεδομένα εισόδου. Ο χρήστης επίσης δίνει το βαθμό του πολυωνύμου που θέλει να υπολογίσει με την μεταβλητή εισόδου num_coeffs. Η συνάρτηση επιστρέφει τις παραμέτρους του μοντέλου και πόσο καλή προσαρμογή έχει γίνει (MSE).
regularized	ratio x y learning_rate training_epochs num_coeffs	l fc	Η συνάρτηση αυτή δέχεται τα δεδομένα X και Y και υπολογίζει για 10 διαφορετικές τιμές του λ (από 0.1 έως 1.0) το συνολικό κόστος για το πολυωνυμικό μοντέλο το βαθμό του οποίου δίδεται από τον χρήστη με την μεταβλητή εισόδου num_coeffs.

Συνάρτηση/Μέθοδος Κλάσης	Δεδομένα Εισόδου	Δεδομένα Εξόδου	Παρατηρήσεις
logistic2d	x1_label1 x2_label1 x1_label2 x2_label2 learning_rate training_epochs	w_val	Η συνάρτηση υλοποιεί Binary Classification με χρήση της γραμμικής παλινδρόμησης για δύο ανεξάρτητες μεταβλητές X1 και X2 που θέλουμε να κατηγοριοποιήσουμε τα δεδομένα μας σε δύο κατηγορίες Label1 και Label2.
softmax	ratio x1_label0 x2_label0 x1_label1 x2_label1 x1_label2 x2_label2 learning_rate training_epochs	W_val b_val accuracy	Η συνάρτηση κατηγοριοποιεί με την λογική της softmax regression δεδομένα με δύο χαρακτηριστικά σε τρεις κατηγορίες.
image_classification	x_train y_train x_valid y_valid x_test y_test n_classes img_h img_w epochs batch_size display_freq learning_rate n_test color		Η συνάρτηση κατηγοριοποιεί εικόνες με την λογική της softmax regression.
k_means	clusters_n iteration_n X	centroid_values	Η συνάρτηση χρησιμοποιεί την μέθοδο των K-means για την συσταδοποίηση των δεδομένων που δίδονται από τον χρήστη.
Class: SOM	width height dim		Κλάση για την δημιουργία ενός αντικειμένου SOM.
Class: SOM **Method**: get_bmu_loc	x		Μέθοδος της κλάσης SOM για την εύρεση του κόμβου με το καλύτερο ταίριασμα για το δεδομένο (BMU).
Class: SOM **Method**: get_propagation	bmu_loc x iter		Μέθοδος της κλάσης SOM για την ενημέρωση των βαρών των γειτονικών κόμβων του BMU.

Συνάρτηση/Μέθοδος Κλάσης	Δεδομένα Εισόδου	Δεδομένα Εξόδου	Παρατηρήσεις
Class: SOM **Method**: train	data		Μέθοδος της κλάσης SOM για την εκπαίδευση και τη δημιουργία του αυτοοργανωμένου χάρτη
subtractive_clustering	data ra rb dim points_n clusters_n iteration_n	centro_new	Η συνάρτηση χρησιμοποιεί την μέθοδο των subtractive clustering για την συσταδοποίηση των δεδομένων που δίδονται από τον χρήστη.
Class: Autoencoder	input_dim hidden_dim epoch batch_size learning_rate		Κλάση για την δημιουργία ενός αντικειμένου Autoencoder.
Class: Autoencoder **Method**: train	data		Μέθοδος της κλάσης Autoencoder για την εκπαίδευση του ΤΝΔ.
Class: Autoencoder **Method**: test	data		Μέθοδος της κλάσης Autoencoder η οποία δέχεται ένα διάνυσμα και το οποίο το κάνει αρχικά compressed και στην συνέχεια reconstructed.
Class: Denoiser	input_dim hidden_dim epoch batch_size learning_rate		Κλάση για την δημιουργία ενός αντικειμένου Denoiser.
Class: Denoiser **Method**: add_noise	data		Μέθοδος της κλάσης Denoiser για την προσθήκη θορύβου σε μια εικόνα.
Class: Denoiser **Method**: train	data		Μέθοδος της κλάσης Denoiser για την εκπαίδευση του ΤΝΔ, για την αποθορυβοποίηση ενός διανύσματος.
Class: Denoiser **Method**: test	data		Μέθοδος της κλάσης Denoiser η οποία δέχεται ένα διάνυσμα με θόρυβο από το οποίο στην συνέχεια το αφαιρεί.

Πίνακας 3-1 Data Mining Toolbox

Βιβλιογραφία

Έντυπες Πηγές

Bishop, C. (1995). *Neural Networks for Pattern Recognition.* Oxford: CLARENDON PRESS.

Chiu, S. (1994). Fuzzy Model Identification Based on Cluster Estimation. *Journal of Intelligent & Fuzzy Systems*.

Haykin, S. (1999). *Neural Networks a Comprehesive Foundation.* Ontario: McMaster University, Hamilton,.

Liang, Z., Yupu, Y., & Yong, Z. (2008). Eliciting compactT–S fuzzy models using subtractive clustering and coevolutionary particle swarm optimization. *Neurocomputing*, 2569–2575.

Shukla, N. (2017). *Machine Learning with TensorFlow.* New YOr: Manning Publications.

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2020). *Εισαγωγή στην ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ.* ΑΘΗΝΑ: ΕΚΔΟΣΕΙΣ ΤΖΙΟΛΑ.

Yager, R., & Filev, D. (1994). Generation of fuzzy rules by mountain clustering. *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology*, 209–219.

Αλτιντζής, Π. (2018). *ΠΤΥΧΙΑΚΗ: Εξόρυξη δεδομένων στην αξιολόγηση της εκπαίδευσης - Μελέτη περίπτωσης η αξιολόγηση του τμήματος Διοίκησης Επιχειρήσεων του Α.Τ.Ε.Ι.Θ.* Θεσσαλονίκη: Α.Τ.Ε.Ι.Θ.

Βαϊόπουλος, Κ. (2021). *ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ: ΑΥΤΟΚΩΔΙΚΟΠΟΙΗΤΗΣ ΜΕΤΑΒΟΛΩΝ ΓΙΑ ΜΗ-ΠΑΡΕΜΒΑΤΙΚΗ ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΦΟΡΤΙΟΥ.* Θεσσαλονίκη: Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης.

Βερύκιος, Β., Καγκλής, Β., & Σταυρόπουλος, Η. (2015). *Η επιστήμη των δεδομένων μέσα από τη γλώσσα R.* Αθήνα: ΣΕΑΒ, ΚΑΛΛΙΠΟΣ.

Βλαχάβας, Ι., Κεφαλάς, Π., Βασιλειάδης, Ν., Κόκκορας, Φ., & Σακελλαρίου, Η. (2000). *Τεχνητή Νοημοσύνη.* Θεσσαλονίκη: ΕΚΔΟΣΕΙΣ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΜΑΚΕΔΟΝΙΑΣ.

Γεωργούλη, Κ. (2015). *ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ Μια Εισαγωγική Προσέγγιση.* Ζωγράφου: ΣΕΑΒ.

Κουτσούκος, Δ. (2016). *ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ: Ευφυείς τεχνικές εξόρυξης δεδομένων για χρήσεις του διαδικτύου.* ΑΘΗΝΑ: ΕΜΠ - ΣΗΜΜΥ.

Μητσοπούλου, Ε. (2017). *ΔΙΠΛΩΜΑΤΙΚΗ: ΑΝΑΛΥΣΗ ΙΑΤΡΙΚΩΝ ΔΕΔΟΜΕΝΩΝ.* Θεσσαλονίκη: Αριστοτέλειο Πανεπιστήμιο.

Μουμουλίδη, Α., & Τζούμα, Χ. (2018). *ΠΤΥΧΙΑΚΗ: Κατανεμημένη εκπαίδευση μοντέλων Μηχανικής Μάθησης.* Θεσσαλονίκη: ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ.

Ηλεκτρονικές Πηγές

[1] California Housing. Τελευταία πρόσβαση σε ισότοπο 02.10.2002: https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

[2] CIFAR-10. Τελευταία πρόσβαση σε ισότοπο 02.10.2002: https://www.cs.toronto.edu/~kriz/cifar.html

[3] Iris. Τελευταία πρόσβαση σε ισότοπο 02.10.2002: https://archive.ics.uci.edu/ml/datasets/iris

[4] MNIST. Τελευταία πρόσβαση σε ισότοπο 02.10.2002:: http://yann.lecun.com/exdb/mnist/

[5] Zygoulis, P. N. Μεθοδολογίες εξόρυξης γνώσης από βάσεις δεδομένων. Μελέτη περίπτωσης – Ανάλυση δεδομένων επιστημομετρίας ακαδημαϊκού προσωπικού τμημάτων πληροφορικής στην Ελλάδα. Τελευταία πρόσβαση σε ισότοπο 02.10.2002:: https://www.researchgate.net/publication/355062206_Methodologies_exoryxes_gnoses_apo_baseis_dedomenon_Melete_periptoses_-_Analyse_dedomenon_epistemometrias_akademaikou_prosopikou_tmematon_plerophorikes_sten_Ellada

[6] Εξόρυξη_δεδομένων#Ιστορία_και_Εξέλιξη. Τελευταία πρόσβαση σε ισότοπο 02.10.2002:

https://el.wikipedia.org/wiki/Εξόρυξη_δεδομένων#Ιστορία_και_Εξέλιξη

Παράρτημα

Α. Βοηθητικές Συναρτήσεις και βιβλιοθήκες

Βοηθητικές Συναρτήσεις

- split_dataset(x_dataset, y_dataset, ratio)

Συνάρτηση για το τυχαίο διαχωρισμό των δεδομένων x_dataset και y_dataset σε training και test data με ποσοστό διαχωρισμού (ratio) που δίδεται από το χρήστη.

```
# Splitting data to training and test
def split_dataset(x_dataset, y_dataset, ratio):
    arr = np.arange(x_dataset.size)
    np.random.shuffle(arr)
    num_train = int(ratio * x_dataset.size)
    x_train = x_dataset[arr[0:num_train]]
    y_train = y_dataset[arr[0:num_train]]
    x_test = x_dataset[arr[num_train:x_dataset.size]]
    y_test = y_dataset[arr[num_train:x_dataset.size]]
    return x_train, x_test, y_train, y_test
```

- load_image(dataset)

Συνάρτηση για το φόρτωμα των εικόνων από τα dataset CIFAR10 και MNIST.

```
# Load Image dataset
def load_image(dataset):
    from tensorflow.keras.datasets import mnist
    from tensorflow.keras.datasets import cifar10

    if dataset == 'cifar10':
        (x_train, y_train_old), (x_test, y_test_old) = cifar10.load_data()
        plt.matshow(x_train[1])
        x_train = x_train.reshape(50000, 3072)
        x_test = x_test.reshape(10000, 3072)

        y_train = np.zeros(shape=(50000,10))
        for i in range(50000):
          y_train[i,y_train_old[i]] = 1.0

        y_valid = np.zeros(shape=(10000,10))
        for i in range(10000):
          y_valid[i,y_test_old[i]] =1.0
```

```
        return x_train, y_train, x_test, y_valid, x_test, y_valid

    elif dataset == 'mnist':
        (x_train, y_train_old), (x_test, y_test_old) = mnist.load_data()
        x_train = x_train.reshape(60000, 784)
        x_test = x_test.reshape(10000, 784)
        y_train = np.zeros(shape=(60000,10))
        for i in range(60000):
          y_train[i,y_train_old[i]] =1.0

        y_valid = np.zeros(shape=(10000,10))
        for i in range(10000):
          y_valid[i,y_test_old[i]] =1.0
        return x_train, y_train, x_test, y_valid, x_test, y_valid
```

- get_batch(X, size)

Συνάρτηση για την επιλογή μιας παρτίδας από μια λίστα δεδομένων.

```
def get_batch(X, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a]
```

- get_batch_dnsr(X, Xn, size)

Συνάρτηση για την επιλογή μιας παρτίδας από δύο λίστες δεδομένων.

```
def get_batch_dnsr(X, Xn, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a], Xn[a]
```

Βιβλιοθήκες

- **numpy** (ιστότοπος βιβλιοθήκης: https://numpy.org/)

Το Numpy είναι μια βιβλιοθήκη για την γλώσσα Python η οποία έχει δημιουργηθεί με σκοπό να διευκολύνει αρκετές μαθηματικές διαδικασίες. Συγκεκριμένα προσφέρει πληθώρα μαθηματικών και μη συναρτήσεων για τον ευκολότερο χειρισμό διανυσμάτων και πινάκων.

- **matplotlib** (ιστότοπος βιβλιοθήκης: https://matplotlib.org/)

Πρόκειται για μια βιβλιοθήκη της Python δημιουργημένη για την δημιουργία διαγραμμάτων. Έχει αναπτυχθεί με στόχο να μοιάζει με τα διαγράμματα του Matlab. Η Matplotlib προσφέρει αντικειμενοστραφή τρόπο για την δημιουργία πολλών ειδών διαγραμμάτων καθώς και κινούμενων διαγραμμάτων.

- **math** (ιστότοπος βιβλιοθήκης https://docs.python.org/3/library/math.html)

Πρόκειται για μια βιβλιοθήκη της Python που δίνει πρόσβαση στις μαθηματικές συναρτήσεις και σταθερές που είναι ορισμένες στην γλώσσα C, όπως για παράδειγμα sqrt, pow, pi, e, etc.

- **time** (ιστότοπος βιβλιοθήκης: https://docs.python.org/3/library/time.html)

Πρόκειται για μια βιβλιοθήκη της Python σχετικά με συναρτήσεις που έχουν σχέση με το χρόνο.

- **tensorflow.compat.v1** (ιστότοπος βιβλιοθήκης: https://www.tensorflow.org/api_docs/python/tf)

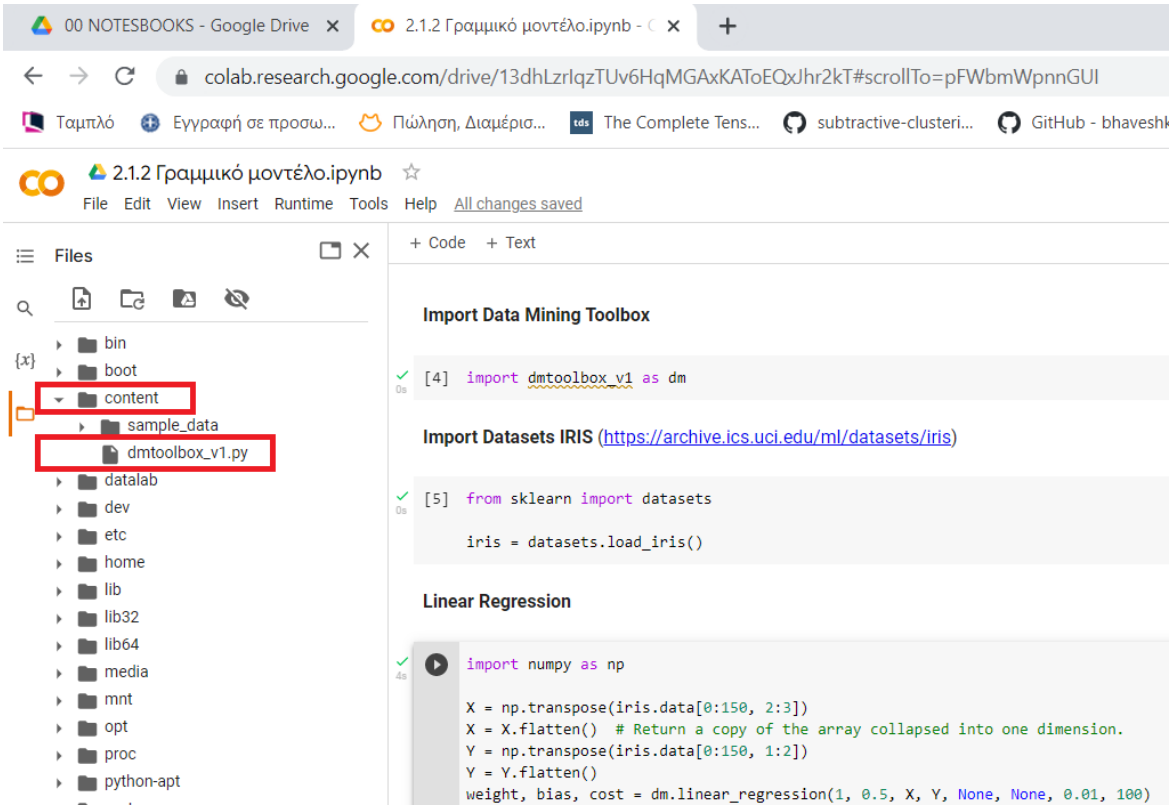
Πρόκειται για την βιβλιοθήκη του Tensorflow της έκδοσης 1.0.. Όπως έχουμε αναφέρει η πτυχιακή αυτή εργασία έχει γίνει στην έκδοση 1.0 και για αυτό στο toolbox που δημιουργήσαμε έχουμε απενεργοποιήσει τις λειτουργικότητες της έκδοσης 2.0 με την εντολή tf.disable_v2_behavior().

- **sklearn** (ιστότοπος βιβλιοθήκης: https://scikit-learn.org/)

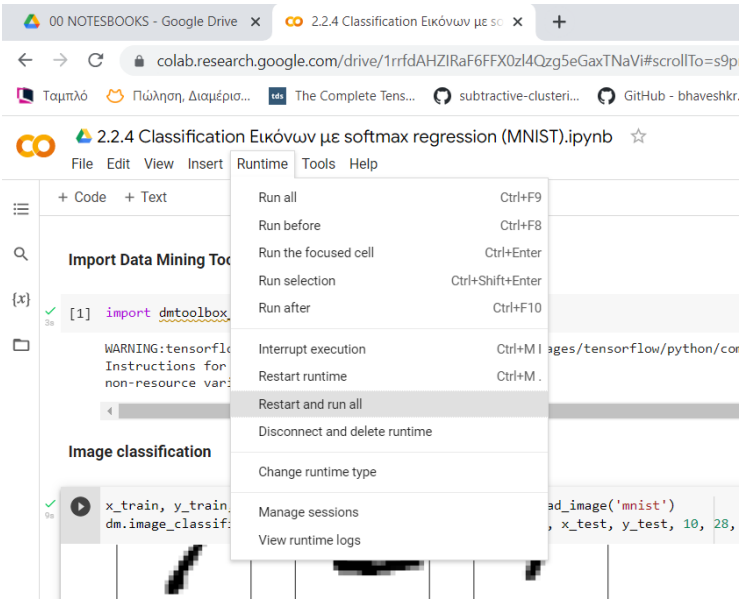
Το Scikit-learn (γνωστό και ως sklearn) προσφέρει ένα ποικίλο σύνολο στατιστικών μοντέλων και μηχανικής μάθησης. Σε αντίθεση με τα περισσότερα modules, το sklearn αναπτύσσεται σε Python αντί για C. Παρά το γεγονός ότι αναπτύχθηκε σε Python, η αποτελεσματικότητα του sklearn αποδίδεται στη χρήση του NumPy για λειτουργίες γραμμικής άλγεβρας και πίνακα υψηλής απόδοσης. Στην συγκεκριμένη εργασία θα την χρησιμοποιήσουμε για τα dataset της.

Β. Εκτέλεση Notebooks

Για την εκτέλεση των Notebooks που αναφέρονται στην εργασία αυτή στο Google Collab θα πρέπει στο φάκελο content να αποθηκευτεί το αρχείο dmtoolbox_v1.py όπου περιέχει το toolbox.



Στην περίπτωση που θέλουμε να εκτελέσουμε ξανά το κώδικα επιλέγουμε από μενού Runtime -> Restart and run all



Γ. Πειραματικά Δεδομένα

Στην πτυχιακή αυτή εργασία χρησιμοποιήθηκαν οι παρακάτω βάσεις δεδομένων:

- **Iris Dataset**

Πρόκειται ίσως για την πιο γνωστή βάση δεδομένων που μπορεί να βρεθεί στην βιβλιογραφία για την αναγνώριση προτύπων. Πρόκειται για ένα σύνολο δεδομένων που εισήγαγε ο βρετανός στατιστικός και βιολόγος Ronald Fisher στην εργασία του το 1936. Πρόκειται για μια κλασική εργασία στο τομέα του και η οποία αναφέρεται ως πηγής πολύ συχνά μέχρι και σήμερα. Το σύνολο των δεδομένων περιέχει 3 κατηγορίες (Iris Setosa, Iris Versicolour και Iris Virginica) από 50 περιπτώσεις η καθεμία, όπου η κάθε κατηγορία αναφέρεται ένα τύπο των λουλουδιών Iris. Αποτελείται από πέντε χαρακτηριστικά: το μήκος του πετάλου, το πλάτος του πετάλου, το μήκος του σε σέπαλου και το πλάτους του σέπαλου σε εκατοστά [3].

Εικόνα 0-1 Άνθος Iris [3]

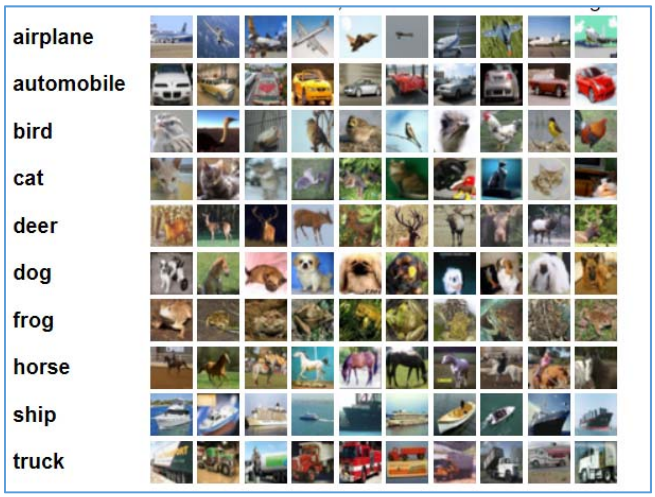
- **CIFAR-10 Dataset**

Πρόκειται για μια βάση δεδομένων η οποία αποτελείται από 60.000 έγχρωμες εικόνες διάστασης 32x32. Είναι χωρισμένο σε 10 κατηγορίες με 6.000 εικόνες ανά κατηγορία. Οι 10 κατηγορίες είναι οι:

1. Αεροπλάνα
2. Αυτοκίνητα
3. Πουλιά
4. Γάτες
5. Ελάφια
6. Σκύλοι
7. Βάτραχοι

8. Άλογα
9. Πλοία
10. Φορτηγά

Κάθε κατηγορία έχει 6.000 εικόνες ανά κλάση. Τέλος υπάρχουν συνολικά 50.000 εικόνες εκπαίδευσης (training images) και 10.000 εικόνες δοκιμής (test images) [2].



Εικόνα 0-2 Δείγμα εικόνων από το dataset CIFAR-10 [2]

- **MNIST Dataset**

Η βάση δεδομένων MNIST είναι μια μεγάλη βάση δεδομένων χειρόγραφων ψηφίων (από το 0 έως το 9) που χρησιμοποιείται συνήθως για την εκπαίδευση διαφόρων συστημάτων επεξεργασίας εικόνας. Η βάση περιέχει 60.000 εικόνες εκπαίδευσης και 10.000 εικόνες δοκιμής. Η κάθε μια εικόνα έχει διάσταση 28x28 [4].

Εικόνα 0-3 Δείγμα εικόνων από το dataset MNIST [4]

- **The California housing dataset**

Πρόκειται για μια βάση δεδομένων που αφορά την μέση αξία κατοικιών στην Καλιφόρνια και η οποία προήλθε από την απογραφή που έγινε στην Η.Π.Α. το 1990. Ο αριθμός των δειγμάτων είναι 20.640 ενώ τα χαρακτηριστικών που μετρήθηκαν είναι τα ακόλουθα [1]:

- MedInc median income in block group
- HouseAge median house age in block group
- AveRooms average number of rooms per household
- AveBedrms average number of bedrooms per household
- Population block group population
- AveOccup average number of household members
- Latitude block group latitude
- Longitude block group longitude

Δ. Κώδικας toolbox

Παρακάτω παρατίθεται ο συνολικός κώδικας του toolbox που δημιουργήθηκε στα πλαίσια αυτής της πτυχιακής εργασίας:

```
# Data Mining Toolbox with TensorFlow 1
# Author: Apostolos Panagiotopoulos-Thomas
# Date: 01/09/2022
# Version: 1.0

import numpy as np
import matplotlib.pyplot as plt
import math
import time
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

# Splitting data to training and test
def split_dataset(x_dataset, y_dataset, ratio):
    arr = np.arange(x_dataset.size)
    np.random.shuffle(arr)
    num_train = int(ratio * x_dataset.size)
    x_train = x_dataset[arr[0:num_train]]
    y_train = y_dataset[arr[0:num_train]]
    x_test = x_dataset[arr[num_train:x_dataset.size]]
    y_test = y_dataset[arr[num_train:x_dataset.size]]
    return x_train, x_test, y_train, y_test

# Linear regression
def linear_regression(split, ratio, x, y, x_test, y_test, learning_rate,
training_epochs):

    if split == 1:
        x_train, x_test, y_train, y_test = split_dataset(x, y, ratio)
        n = x_train.size
        n_test = x_test.size
    else:
        x_train = x
        y_train = y
        n = x.size
        n_test = x_test.size

    X = tf.placeholder(tf.float32)
    Y = tf.placeholder(tf.float32)

    w = tf.Variable(np.random.randn(), name = "weights")
    b = tf.Variable(np.random.randn(), name = "bias")
```

```
    def model(X, w, b):
        return tf.add(tf.multiply(X, w), b)

    y_model = model(X, w, b)

    # Mean Squared Error Cost Function
    cost = tf.reduce_sum(tf.square(Y-y_model)) / (n)
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

    # Starting the Tensorflow Session
    with tf.Session() as sess:
        init = tf.global_variables_initializer()
        sess.run(init)

    # Iterating through all the epochs
        for epoch in range(training_epochs):
            # Feeding each data point into the optimizer using Feed Dictionary
            for (x, y) in zip(x_train, y_train):
                sess.run(train_op, feed_dict={X: x, Y: y})
            # Displaying the result after every 50 epochs
            if (epoch + 1) % 50 == 0:
                # Calculating the cost a every epoch
                c = sess.run(cost, feed_dict = {X : x, Y : y})
                print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(w), "b =",
sess.run(b))
        # Storing necessary values to be used outside the Session
        training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
        weight = sess.run(w)
        bias = sess.run(b)
    sess.close()

    # Print Weight and bias of the calculated model and cost from training data
    print("-------------------------------------------------------", '\n')
    print("Weight =", weight, "bias =", bias, '\n')
    print("Training cost =", training_cost, '\n')

    # Calculating the predictions
    predictions = weight * x_train + bias
    # Calculating the predictions for test data
    predictions_test = weight * x_test + bias
    diff = y_test - predictions_test
    cost_test = 0.0
    for (z) in range(n_test):
      cost_test += math.pow(diff[z], 2)
    cost_test = math.sqrt(cost_test)  / (n_test)
    # Print cost from test data
```

```
    print("Test cost =", cost_test, '\n')

    # Plotting the Results
    plt.plot(x_train, y_train, 'ro', label ='Train data')
    plt.plot(x_test, y_test, 'gx', label ='Test data')
    plt.plot(x_train, predictions, label ='Fitted line')
    plt.title('Linear Regression Result')
    plt.legend()
    plt.show()
    return weight, bias, cost_test

# Polynomial regression
def polynomial_regression(split, ratio, x, y,  x_test, y_test, learning_rate,
training_epochs, num_coeffs):

    if split == 1:
        x_train, x_test, y_train, y_test = split_dataset(x, y, ratio)
        n = x_train.size
        n_test = x_test.size
    else:
        x_train = x
        y_train = y
        n = x.size
        n_test = x_test.size

    # Calculating x values the predictions
    x_min = min(x)
    x_max = max(x)
    x_values = np.ndarray(100, dtype=float)
    for i in range(100):
        x_values[i] = x_min + i * (x_max - x_min)/100

    X = tf.placeholder(tf.float32)
    Y = tf.placeholder(tf.float32)

    def model(X, w):
        terms = []
        for i in range(num_coeffs):
            term = tf.multiply(w[i], tf.pow(X, i))
            terms.append(term)
        return tf.add_n(terms)

    w = tf.Variable([0.] * num_coeffs, name="parameters")
    y_model = model(X, w)

    # Mean Squared Error Cost Function
    cost = tf.reduce_sum(tf.square(Y-y_model)) / (n)
```

```
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Starting the Tensorflow Session
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
# Iterating through all the epochs
    for epoch in range(training_epochs):

        # Feeding each data point into the optimizer using Feed Dictionary
        for (x, y) in zip(x_train, y_train):
            sess.run(train_op, feed_dict={X: x, Y: y})

        # Displaying the result after every 50 epochs
        if (epoch + 1) % 50 == 0:
            # Calculating the cost a every epoch
            c = sess.run(cost, feed_dict = {X : x, Y : y})
            print("Epoch", (epoch + 1), ": cost =", c, "W =", sess.run(w))

    # Storing necessary values to be used outside the Session
    training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
    weight = sess.run(w)
sess.close()

# Print Weight and bias of the calculated model and cost from training data
print("-------------------------------------------------------", '\n')
print("Weight =", weight, '\n')
print("Training cost =", training_cost, '\n')

# Calculating the predictions
predictions = 0
for i in range(num_coeffs):
    predictions += weight[i] * np.power(x_values, i)
# Calculating the predictions for test data
predictions_test = 0
for i in range(num_coeffs):
    predictions_test += weight[i] * np.power(x_test, i)
diff = y_test - predictions_test
cost_test = 0.0
for (z) in range(n_test):
  cost_test += math.pow(diff[z], 2)
cost_test = math.sqrt(cost_test)  / (n_test)
# Print cost from test data
print("Test cost =", cost_test, '\n')

# Plotting the Results
plt.plot(x_train, y_train, 'ro', label ='Training data')
```

```
    plt.plot(x_test, y_test, 'gx', label ='Test data')
    plt.plot(x_values, predictions, label ='Fitted line')
    plt.title('Polynomial Regression Result')
    plt.legend()
    plt.show()
    return weight, cost_test

# Regularized cost function
def regularized(ratio, x, y, learning_rate, training_epochs, num_coeffs):
    reg_lambda = 0.
    n = x.size
    x_data = x
    y_data = y
    (x_train, x_test, y_train, y_test) = split_dataset(x_data, y_data, ratio)
    X = tf.placeholder(tf.float32)
    Y = tf.placeholder(tf.float32)
    l = np.array([])
    fc = np.array([])
    def model(X, w):
        terms = []
        for i in range(num_coeffs):
            term = tf.multiply(w[i], tf.pow(X, i))
            terms.append(term)
        return tf.add_n(terms)

    w = tf.Variable([0.] * num_coeffs, name="parameters")
    y_model = model(X, w)
    cost = tf.div(tf.add(tf.reduce_sum(tf.square(Y-y_model)),
           tf.multiply(reg_lambda, tf.reduce_sum(tf.square(w)))),
           n)
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
    sess = tf.Session()
    init = tf.global_variables_initializer()

    sess.run(init)
    for reg_lambda in np.linspace(0,1,10):
        for epoch in range(training_epochs):
            sess.run(train_op, feed_dict={X: x_train, Y: y_train})
        final_cost = sess.run(cost, feed_dict={X: x_test, Y:y_test})
        print('reg lambda', reg_lambda)
        print('final cost', final_cost)
        l = np.append(l, reg_lambda)
        fc = np.append(fc, final_cost)
    sess.close()

    return l, fc
```

```
# Classification Logistic2d
def logistic2d(x1_label1, x2_label1, x1_label2, x2_label2, learning_rate,
training_epochs):

    x1s = np.append(x1_label1, x1_label2)
    x2s = np.append(x2_label1, x2_label2)
    ys = np.asarray([0.] * len(x1_label1) + [1.] * len(x1_label2))
    X1 = tf.placeholder(tf.float32, shape=(None,), name="x1")
    X2 = tf.placeholder(tf.float32, shape=(None,), name="x2")
    Y = tf.placeholder(tf.float32, shape=(None,), name="y")
    w = tf.Variable([0., 0., 0.], name="w", trainable=True)

    y_model = tf.sigmoid(-(w[2] * X2 + w[1] * X1 + w[0]))
    cost = tf.reduce_mean(-tf.log(y_model * Y + (1 - y_model) * (1 - Y)))
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        prev_err = 0
        for epoch in range(training_epochs):
            err, _ = sess.run([cost, train_op], {X1: x1s, X2: x2s, Y: ys})
            if epoch % 100 == 0:
                print("Epoch" , epoch, ": cost:", err)
            if abs(prev_err - err) < 0.0001:
                break
            prev_err = err

        w_val = sess.run(w, {X1: x1s, X2: x2s, Y: ys})

    x1_boundary, x2_boundary = [], []
    with tf.Session() as sess:
        for x1_test in np.linspace(0, 10, 20):
            for x2_test in np.linspace(0, 10, 20):
                z = sess.run(tf.sigmoid(-x2_test*w_val[2] - x1_test*w_val[1] -
w_val[0]))
                if abs(z - 0.5) < 0.05:
                    x1_boundary.append(x1_test)
                    x2_boundary.append(x2_test)
    print('\n')
    plt.scatter(x1_boundary, x2_boundary, c='b', marker='o', s=20, label='Boundary')
    plt.scatter(x1_label1, x2_label1, c='r', marker='x', s=20, label='Label1')
    plt.scatter(x1_label2, x2_label2, c='g', marker='1', s=20, label='Label2')
    plt.title('Binary Classification')
    plt.legend()
    plt.show()

    return w_val
```

```
# Classification Softmax
def softmax(ratio, x1_label0, x2_label0, x1_label1, x2_label1, x1_label2, x2_label2,
learning_rate, training_epochs):

    num_labels = 3
    batch_size = 50

    plt.scatter(x1_label0, x2_label0, c='r', marker='o', s=60, label='Label0')
    plt.scatter(x1_label1, x2_label1, c='g', marker='x', s=60, label='Label1')
    plt.scatter(x1_label2, x2_label2, c='b', marker='_', s=60, label='Label2')
    plt.title('Data')
    plt.legend()
    plt.show()
    print('\n')


    x1_label0, test_x1_label0, x2_label0, test_x2_label0 = split_dataset(x1_label0,
x2_label0, ratio)
    x1_label1, test_x1_label1, x2_label1, test_x2_label1 = split_dataset(x1_label1,
x2_label1, ratio)
    x1_label2, test_x1_label2, x2_label2, test_x2_label2 = split_dataset(x1_label2,
x2_label2, ratio)

    xs_label0 = np.hstack((x1_label0, x2_label0))
    xs_label1 = np.hstack((x1_label1, x2_label1))
    xs_label2 = np.hstack((x1_label2, x2_label2))

    xs = np.vstack((xs_label0, xs_label1, xs_label2))

    labels = np.matrix([[1., 0., 0.]] * len(x1_label0) + [[0., 1., 0.]] * len(x1_label1)
+ [[0., 0., 1.]] * len(x1_label2))
    arr = np.arange(xs.shape[0])
    np.random.shuffle(arr)
    xs = xs[arr, :]
    labels = labels[arr, :]

    test_xs_label0 = np.hstack((test_x1_label0, test_x2_label0))
    test_xs_label1 = np.hstack((test_x1_label1, test_x2_label1))
    test_xs_label2 = np.hstack((test_x1_label2, test_x2_label2))

    test_xs = np.vstack((test_xs_label0, test_xs_label1, test_xs_label2))
    test_labels = np.matrix([[1., 0., 0.]] * len(test_x1_label0) + [[0., 1., 0.]] *
len(test_x1_label1) + [[0., 0., 1.]] * len(test_x1_label2))

    train_size, num_features = xs.shape
```

```
    X = tf.placeholder("float", shape=[None, num_features])
    Y = tf.placeholder("float", shape=[None, num_labels])

    W = tf.Variable(tf.zeros([num_features, num_labels]))
    b = tf.Variable(tf.zeros([num_labels]))
    y_model = tf.nn.softmax(tf.matmul(X, W) + b)

    cost = -tf.reduce_sum(Y * tf.log(y_model))
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

    correct_prediction = tf.equal(tf.argmax(y_model, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

    with tf.Session() as sess:
      tf.global_variables_initializer().run()

      for step in range(training_epochs * train_size // batch_size):
        offset = (step * batch_size) % train_size
        batch_xs = xs[offset:(offset + batch_size), :]
        batch_labels = labels[offset:(offset + batch_size)]
        err, _ = sess.run([cost, train_op], feed_dict={X: batch_xs, Y: batch_labels})
        if step % 100 == 0:
          print ("Step: ", step, "Error: ", err)

      W_val = sess.run(W)
      print('w', W_val)
      b_val = sess.run(b)
      print('b', b_val)
      print("accuracy", accuracy.eval(feed_dict={X: test_xs, Y: test_labels}))

    return W_val, b_val, accuracy

# Load Image dataset
def load_image(dataset):
    from tensorflow.keras.datasets import mnist
    from tensorflow.keras.datasets import cifar10

    if dataset == 'cifar10':
        (x_train, y_train_old), (x_test, y_test_old) = cifar10.load_data()
        plt.matshow(x_train[1])
        x_train = x_train.reshape(50000, 3072)
        x_test = x_test.reshape(10000, 3072)

        y_train = np.zeros(shape=(50000,10))
        for i in range(50000):
          y_train[i,y_train_old[i]] = 1.0
```

```
        y_valid = np.zeros(shape=(10000,10))
        for i in range(10000):
          y_valid[i,y_test_old[i]] =1.0

        return x_train, y_train, x_test, y_valid, x_test, y_valid

    elif dataset == 'mnist':
        (x_train, y_train_old), (x_test, y_test_old) = mnist.load_data()
        x_train = x_train.reshape(60000, 784)
        x_test = x_test.reshape(10000, 784)
        y_train = np.zeros(shape=(60000,10))
        for i in range(60000):
          y_train[i,y_train_old[i]] =1.0

        y_valid = np.zeros(shape=(10000,10))
        for i in range(10000):
          y_valid[i,y_test_old[i]] =1.0
        return x_train, y_train, x_test, y_valid, x_test, y_valid


# Image Classification
def image_classification(x_train, y_train, x_valid, y_valid, x_test, y_test, n_classes,
img_h, img_w, epochs, batch_size, display_freq, learning_rate, n_test, color):

    img_size_flat = img_h * img_w * color  # hxwXc, the total number of pixels

    def randomize(x, y):
        """ Randomizes the order of data samples and their corresponding labels"""
        permutation = np.random.permutation(y.shape[0])
        shuffled_x = x[permutation, :]
        shuffled_y = y[permutation]
        return shuffled_x, shuffled_y

    def get_next_batch(x, y, start, end):
        x_batch = x[start:end]
        y_batch = y[start:end]
        return x_batch, y_batch

    # Print image data
    print("Size of:")
    print("- Training-set:\t\t{}".format(len(y_train)))
    print("- Validation-set:\t{}".format(len(y_valid)))

    print(y_train[1])
    print('x_train:\t{}'.format(x_train.shape))
    print('y_train:\t{}'.format(y_train.shape))
    print('x_valid:\t{}'.format(x_valid.shape))
```

```
    print('y_valid:\t{}'.format(y_valid.shape))

    y_valid[:5, :]

    def weight_variable(shape):
        """
        Create a weight variable with appropriate initialization
        :param name: weight name
        :param shape: weight shape
        :return: initialized weight variable
        """
        initer = tf.truncated_normal_initializer(stddev=0.01)
        return tf.get_variable('W',
                               dtype=tf.float32,
                               shape=shape,
                               initializer=initer)

    def bias_variable(shape):
        """
        Create a bias variable with appropriate initialization
        :param name: bias variable name
        :param shape: bias variable shape
        :return: initialized bias variable
        """
        initial = tf.constant(0., shape=shape, dtype=tf.float32)
        return tf.get_variable('b',
                               dtype=tf.float32,
                               initializer=initial)

    # Create the graph for the linear model
    # Placeholders for inputs (x) and outputs(y)
    x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='X')
    y = tf.placeholder(tf.float32, shape=[None, n_classes], name='Y')

    # Create weight matrix initialized randomely from N~(0, 0.01)
    W = weight_variable(shape=[img_size_flat, n_classes])

    # Create bias vector initialized as zero
    b = bias_variable(shape=[n_classes])

    output_logits = tf.matmul(x, W) + b

    # Define the loss function, optimizer, and accuracy
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,
logits=output_logits), name='loss')
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate, name='Adam-
op').minimize(loss)
```

```
    correct_prediction = tf.equal(tf.argmax(output_logits, 1), tf.argmax(y, 1),
name='correct_pred')
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='accuracy')

    # Model predictions
    cls_prediction = tf.argmax(output_logits, axis=1, name='predictions')

    # Creating the op for initializing all variables
    init = tf.global_variables_initializer()

    # Create an interactive session (to keep the session in the other cells)
    sess = tf.InteractiveSession()
    # Initialize all variables
    sess.run(init)
    # Number of training iterations in each epoch
    num_tr_iter = int(len(y_train) / batch_size)
    for epoch in range(epochs):
        print('Training epoch: {}'.format(epoch + 1))
        # Randomly shuffle the training data at the beginning of each epoch
        x_train, y_train = randomize(x_train, y_train)
        for iteration in range(num_tr_iter):
            start = iteration * batch_size
            end = (iteration + 1) * batch_size
            x_batch, y_batch = get_next_batch(x_train, y_train, start, end)

            # Run optimization op (backprop)
            feed_dict_batch = {x: x_batch, y: y_batch}
            sess.run(optimizer, feed_dict=feed_dict_batch)

            if iteration % display_freq == 0:
                # Calculate and display the batch loss and accuracy
                loss_batch, acc_batch = sess.run([loss, accuracy],
                                                 feed_dict=feed_dict_batch)

                print("iter {0:3d}:\t Loss={1:.2f},\tTraining Accuracy={2:.01%}".
                      format(iteration, loss_batch, acc_batch))

        # Run validation after every epoch
        feed_dict_valid = {x: x_valid[:n_test], y: y_valid[:n_test]}
        loss_valid, acc_valid = sess.run([loss, accuracy], feed_dict=feed_dict_valid)
        print('---------------------------------------------------------')
        print("Epoch: {0}, validation loss: {1:.2f}, validation accuracy: {2:.01%}".
              format(epoch + 1, loss_valid, acc_valid))
        print('---------------------------------------------------------')

    # Test the network after training
    # Accuracy
```

```
feed_dict_test = {x: x_valid[:n_test], y: y_valid[:n_test]}
loss_test, acc_test = sess.run([loss, accuracy], feed_dict=feed_dict_test)
print('---------------------------------------------------------')
print("Test loss: {0:.2f}, test accuracy: {1:.01%}".format(loss_test, acc_test))
print('---------------------------------------------------------')

def plot_images(images, cls_true, cls_pred=None, title=None):
    """
    Create figure with 3x3 sub-plots.
    :param images: array of images to be plotted, (9, img_h*img_w)
    :param cls_true: corresponding true labels (9,)
    :param cls_pred: corresponding true labels (9,)
    """
    fig, axes = plt.subplots(3, 3, figsize=(9, 9))
    fig.subplots_adjust(hspace=0.3, wspace=0.3)
    for i, ax in enumerate(axes.flat):
        # Plot image.
        if color == 1:
            ax.imshow(images[i].reshape(img_h, img_w), cmap='binary')
        elif color == 3:
            ax.imshow(images[i].reshape(img_h, img_w, color), cmap='binary')

        # Show true and predicted classes.
        if cls_pred is None:
            ax_title = "True: {0}".format(cls_true[i])
        else:
            ax_title = "True: {0}, Pred: {1}".format(cls_true[i], cls_pred[i])

        ax.set_title(ax_title)

        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])

    if title:
        plt.suptitle(title, size=20)
    plt.show(block=False)

def plot_example_errors(images, cls_true, cls_pred, title=None):
    """
    Function for plotting examples of images that have been mis-classified
    :param images: array of all images, (#imgs, img_h*img_w)
    :param cls_true: corresponding true labels, (#imgs,)
    :param cls_pred: corresponding predicted labels, (#imgs,)
    """
    # Negate the boolean array.
    incorrect = np.logical_not(np.equal(cls_pred, cls_true))
```

```
        # Get the images from the test-set that have been
        # incorrectly classified.
        incorrect_images = images[incorrect]

        # Get the true and predicted classes for those images.
        cls_pred = cls_pred[incorrect]
        cls_true = cls_true[incorrect]

        # Plot the first 9 images.
        plot_images(images=incorrect_images[0:9],
                    cls_true=cls_true[0:9],
                    cls_pred=cls_pred[0:9],
                    title=title)

    def plot_example_correct(images, cls_true, cls_pred, title=None):
        """
        Function for plotting examples of images that have been mis-classified
        :param images: array of all images, (#imgs, img_h*img_w)
        :param cls_true: corresponding true labels, (#imgs,)
        :param cls_pred: corresponding predicted labels, (#imgs,)
        """
        # Negate the boolean array.
        correct = np.equal(cls_pred, cls_true)
        # Get the images from the test-set that have been
        # incorrectly classified.
        correct_images = images[correct]

        # Get the true and predicted classes for those images.
        cls_pred = cls_pred[correct]
        cls_true = cls_true[correct]

        # Plot the first 9 images.
        plot_images(images=correct_images[0:9],
                    cls_true=cls_true[0:9],
                    cls_pred=cls_pred[0:9],
                    title=title)

    # Plot some of the correct and misclassified examples
    cls_pred = sess.run(cls_prediction, feed_dict=feed_dict_test)
    cls_true = np.argmax(y_valid[:n_test], axis=1)
    plot_example_correct(x_valid[:n_test], cls_true, cls_pred, title='Correct Examples')
    plot_example_errors(x_valid[:n_test], cls_true, cls_pred, title='Misclassified
Examples')
    plt.show()

# Clustering with K-means
```

```
def k_means(clusters_n, iteration_n, X):

    points = tf.constant(X)
    centroids = tf.Variable(tf.slice(tf.random_shuffle(points), [0, 0], [clusters_n, -
1]))

    points_expanded = tf.expand_dims(points, 0)
    centroids_expanded = tf.expand_dims(centroids, 1)

    distances = tf.reduce_sum(tf.square(tf.subtract(points_expanded,
centroids_expanded)), 2)
    assignments = tf.argmin(distances, 0)

    means = []
    for c in range(clusters_n):
        means.append(tf.reduce_mean(
          tf.gather(points,
                    tf.reshape(
                      tf.where(
                        tf.equal(assignments, c)
                      ),[1,-1])
                   ),reduction_indices=[1]))

    new_centroids = tf.concat(means, 0)

    update_centroids = tf.assign(centroids, new_centroids)
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
      sess.run(init)
      for step in range(iteration_n):
        [_, centroid_values, points_values, assignment_values] =
sess.run([update_centroids, centroids, points, assignments])

      print("centroids", centroid_values)

    plt.scatter(points_values[:, 0], points_values[:, 1], c=assignment_values, s=50,
alpha=0.5)
    plt.plot(centroid_values[:, 0], centroid_values[:, 1], 'kx', markersize=15)
    plt.title('K-means')
    plt.show()
    return centroid_values

# SOM class
class SOM:
    def __init__(self, width, height, dim):
        self.num_iters = 100
```

```
        self.width = width
        self.height = height
        self.dim = dim
        self.node_locs = self.get_locs()

        # Each node is a vector of dimension `dim`
        # For a 2D grid, there are `width * height` nodes
        nodes = tf.Variable(tf.random_normal([width*height, dim]))
        self.nodes = nodes

        # These two ops are inputs at each iteration
        x = tf.placeholder(tf.float32, [dim])
        iter = tf.placeholder(tf.float32)

        self.x = x
        self.iter = iter

        # Find the node that matches closest to the input
        bmu_loc = self.get_bmu_loc(x)

        self.propagate_nodes = self.get_propagation(bmu_loc, x, iter)

    def get_propagation(self, bmu_loc, x, iter):
        num_nodes = self.width * self.height
        rate = 1.0 - tf.div(iter, self.num_iters)
        alpha = rate * 0.5
        sigma = rate * tf.to_float(tf.maximum(self.width, self.height)) / 2.
        expanded_bmu_loc = tf.expand_dims(tf.to_float(bmu_loc), 0)
        sqr_dists_from_bmu = tf.reduce_sum(tf.square(tf.subtract(expanded_bmu_loc,
self.node_locs)), 1)
        neigh_factor = tf.exp(-tf.div(sqr_dists_from_bmu, 2 * tf.square(sigma)))
        rate = tf.multiply(alpha, neigh_factor)
        rate_factor = tf.stack([tf.tile(tf.slice(rate, [i], [1]), [self.dim]) for i in
range(num_nodes)])
        nodes_diff = tf.multiply(rate_factor, tf.subtract(tf.stack([x for i in
range(num_nodes)]), self.nodes))
        update_nodes = tf.add(self.nodes, nodes_diff)
        return tf.assign(self.nodes, update_nodes)

    def get_bmu_loc(self, x):
        expanded_x = tf.expand_dims(x, 0)
        sqr_diff = tf.square(tf.subtract(expanded_x, self.nodes))
        dists = tf.reduce_sum(sqr_diff, 1)
        bmu_idx = tf.argmin(dists, 0)
        bmu_loc = tf.stack([tf.mod(bmu_idx, self.width), tf.div(bmu_idx, self.width)])
        return bmu_loc
```

```
    def get_locs(self):
        locs = [[x, y]
                for y in range(self.height)
                for x in range(self.width)]
        return tf.to_float(locs)

    def train(self, data):
        with tf.Session() as sess:
            sess.run(tf.initialize_all_variables())
            for i in range(self.num_iters):
                for data_x in data:
                    sess.run(self.propagate_nodes, feed_dict={self.x: data_x, self.iter:
i})
            centroid_grid = [[] for i in range(self.width)]
            self.nodes_val = list(sess.run(self.nodes))
            self.locs_val = list(sess.run(self.node_locs))
            for i, l in enumerate(self.locs_val):
                centroid_grid[int(l[0])].append(self.nodes_val[i])
            self.centroid_grid = centroid_grid

# Subtractive clustering
def subtractive_clustering(data, ra, rb, dim, points_n, clusters_n, iteration_n):
    points = tf.constant(data)
    i = tf.constant(data)
    j = tf.constant(data)
    di = tf.Variable(tf.zeros([points_n,1], dtype=np.float64))
    centroid = []
    centroid_density = []

    def find_density(xi, xj, step, di):
      i_expanded = tf.expand_dims(xi, 0)
      j_expanded = tf.expand_dims(xj, 1)
      if step == 0:
        density =tf.exp(tf.scalar_mul(-1/((ra/2)**2) ,
tf.reduce_sum(tf.pow(tf.subtract(i_expanded, j_expanded),2),2, keepdims=True)))
        density = tf.reduce_sum(density,1)
        position_density = tf.reduce_max(density)
        position = tf.argmax(density, 0)
        di = tf.assign(di, density)
      else:
        xc_expanded = tf.repeat(centroid[step-1],repeats=[points_n], axis=0)
        density =tf.exp(tf.scalar_mul(-1/((rb/2)**2) ,
tf.reduce_sum(tf.pow(tf.subtract(xi, xc_expanded),2),1, keepdims=True)))
        density = tf.scalar_mul(centroid_density[step-1], density)
        density = tf.subtract(di,density)
        position_density = tf.reduce_max(density)
        position = tf.argmax(density, 0)
```

```
        di = tf.assign(di, density)
      position = tf.squeeze(position)
      return position, position_density, di

    init = tf.global_variables_initializer()

    with tf.Session() as sess:
          init = tf.global_variables_initializer()
          sess.run(init)
          for step in range(clusters_n):
            [position, position_density, di] = find_density(i, j, step, di)
            centroid.append(i[position:position+1, :])
            centroid_density.append(position_density)
          centro = tf.concat(centroid,0)
          centro = tf.reshape(centro, shape=(clusters_n,dim))
          centro_expanded = tf.expand_dims(centro, 1)
          points_expanded = tf.expand_dims(points, 0)
          distances = tf.reduce_sum(tf.square(tf.subtract(points_expanded,
centro_expanded)), 2)
          assignments = tf.argmin(distances, 0)
          assignments_values = assignments.eval() #tensor to numpy.ndarray
          centro_new = centro.eval() #tensor to numpy.ndarray

    print('Subtractive cluster centers')
    print(centro_new)
    plt.scatter(data[:, 0], data[:, 1], c=assignments_values, s=50, alpha=0.5)
    plt.plot(centro_new[:, 0], centro_new[:, 1], 'kx', markersize=15)
    plt.title('Subtractive clustering')
    plt.show()

    return centro_new

# Autoencoder
def get_batch(X, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a]

class Autoencoder:
    def __init__(self, input_dim, hidden_dim, epoch, batch_size, learning_rate):
        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate

        # Define input placeholder
        x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim])

        # Define variables
```

```
        with tf.name_scope('encode'):
            weights = tf.Variable(tf.random_normal([input_dim, hidden_dim],
dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            encoded = tf.nn.sigmoid(tf.matmul(x, weights) + biases)
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim],
dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            decoded = tf.matmul(encoded, weights) + biases

        self.x = x
        self.encoded = encoded
        self.decoded = decoded

        # Define cost function and training operation
        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
self.decoded))))
        self.all_loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
self.decoded)), 1))
        self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)

        # Define a saver operation
        self.saver = tf.train.Saver()

    def train(self, data):
        num_samples = len(data)
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            for i in range(self.epoch):
                for j in range(num_samples):
                    batch_data = get_batch(data, self.batch_size)
                    l, _ = sess.run([self.loss, self.train_op], feed_dict={self.x:
batch_data})
                if i % 50 == 0:
                    print('epoch {0}: loss = {1}'.format(i, l))
                    self.saver.save(sess, './model.ckpt')
            self.saver.save(sess, './model.ckpt')

    def test(self, data):
        with tf.Session() as sess:
            self.saver.restore(sess, './model.ckpt')
            hidden, reconstructed = sess.run([self.encoded, self.decoded],
feed_dict={self.x: data})
            difference =
sess.run(tf.multiply(tf.divide(tf.subtract(data,reconstructed),data),100))
        print('input', data)
```

```
        print('compressed', hidden)
        print('reconstructed', reconstructed)
        print('difference', difference)
        return reconstructed

    def compress(self, data):
        with tf.Session() as sess:
            self.saver.restore(sess, './model.ckpt')
            hidden, reconstructed = sess.run([self.encoded, self.decoded],
feed_dict={self.x: data})
            difference =
sess.run(tf.multiply(tf.divide(tf.subtract(data,reconstructed),data),100))
        return reconstructed


# Autoencoder - Denoiser
def get_batch_dnsr(X, Xn, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a], Xn[a]

class Denoiser:

    def __init__(self, input_dim, hidden_dim, epoch=10000, batch_size=50,
learning_rate=0.001):
        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate

        self.x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim], name='x')
        self.x_noised = tf.placeholder(dtype=tf.float32, shape=[None, input_dim],
name='x_noised')
        with tf.name_scope('encode'):
            self.weights1 = tf.Variable(tf.random_normal([input_dim, hidden_dim],
dtype=tf.float32), name='weights')
            self.biases1 = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            self.encoded = tf.nn.sigmoid(tf.matmul(self.x_noised, self.weights1) +
self.biases1, name='encoded')
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim],
dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            self.decoded = tf.matmul(self.encoded, weights) + biases
        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
self.decoded))))
        self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)
        self.saver = tf.train.Saver()
```

```
    def add_noise(self, data):
        noise_type = 'mask-0.2'
        if noise_type == 'gaussian':
            n = np.random.normal(0, 0.1, np.shape(data))
            return data + n
        if 'mask' in noise_type:
            frac = float(noise_type.split('-')[1])
            temp = np.copy(data)
            for i in temp:
                n = np.random.choice(len(i), round(frac * len(i)), replace=False)
                i[n] = 0
            return temp

    def train(self, data):
        num_samples = len(data)
        data_noised = self.add_noise(data)
        with open('log.csv', 'w') as writer:
            with tf.Session() as sess:
                sess.run(tf.global_variables_initializer())
                for i in range(self.epoch):
                    for j in range(num_samples):
                        batch_data, batch_data_noised = get_batch_dnsr(data,
data_noised, self.batch_size)
                        l, _ = sess.run([self.loss, self.train_op], feed_dict={self.x:
batch_data, self.x_noised: batch_data_noised})
                    if i % 50 == 0:
                        print('epoch {0}: loss = {1}'.format(i, l))
                        self.saver.save(sess, './model.ckpt')
                        epoch_time = int(time.time())
                        row_str = str(epoch_time) + ',' + str(i) + ',' + str(l) + '\n'
                        writer.write(row_str)
                        writer.flush()
                self.saver.save(sess, './model.ckpt')

    def test(self, data):
        with tf.Session() as sess:
            self.saver.restore(sess, './model.ckpt')
            hidden, reconstructed = sess.run([self.encoded, self.decoded],
feed_dict={self.x: data, self.x_noised: data})
        return reconstructed
```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.