ΕΞΟΡΥΞΗ ΓΝΩΗΣΗΣ ΑΠΟ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΤΟΝ ΠΑΓΚΟΣΜΙΟ ΙΣΤΟ

ΕΡΓΑΣΙΑ 4
ΔΙΑΓΩΝΙΣΜΟΣ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗΣ

Διαμάντη Κατερίνα – 3090053
Ντινιάκου Θάλεια – 3100132
Χατζηηλία Σοφία – 3100202

Α.Πρόλογος

Σκοπός της εργασίας είναι να προβλέψουμε την επιβίωση των επιβατών του Τιτανικού, βάσει του δείγματος που διαθέτουμε(train.csv). Μια γενική περιγραφή είναι η εξής: Αρχικά νορμαλοποιούμε τα δεδομένα για να μπορέσουμε να τα επεξεργαστούμε και για να κρατήσουμε εκείνα τα δεδομένα, τα οποία μας είναι χρήσιμα για την κατηγοριοποίηση των επιβατών, δηλαδή βγάζουμε τα δεδομένα που προκαλούν "θορυβο". Στη συνέχεια, για την κατηγοριοποίηση των επιβατών χρησιμοποιούμε τον αλγόριθμο k κοντινότερων γειτόνων (knn) με μέτρο την ευκλείδια απόσταση.

Ακολουθούν αναλυτικα οι κλάσεις και οι μέθοδοι που χρησιμοποιήθηκαν, αλλά και πληροφορίες για την εκτέλεση και τα τελικά αποτελέσματα.

Β.Περιγραφή Κλάσεων και Μεθόδων

Κλάση Normalization:

(ο κώδικας περιέχεται στο αρχείο normalization.py)

Δημιουργήσαμε την κλάση normalization, έτσι ώστε να κανονικοποιήσουμε τα δεδομένα. Η κλάση αυτή λέγεται normalization και έχει ως όρισμα έναν πίνακα τον οποίο και νορμαλοποιεί. Σκοπός της κλάσης αυτής είναι:

•Να φέρουμε τα δεδομένα σε τέτοια μορφή, ώστε να απεικονίζονται σωστά στον v-διαστατό χώρο και να μην υπάρχουν ορίσματα με αλφαριθμητικά.
•να αφαιρέσουμε δεδομένα, τα οποία δεν βοηθούν στην κατηγοριοποίηση των επιβατών.
•να καλύψουμε χαρακτηριστικά που λείπουν σε επιβάτες.

1) ΑΦΑΙΡΕΣΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ:

Όπως φαίνεται και στον κώδικα διαγράφουμε με χρήση της εντολή delete τα εξής :

PassengerId , Embarked, Cabin , Fare, Ticket. Τα παραπάνω χαρακτηριστικά αφαιρέθηκαν, αφού δεν βοηθούν στην κατηγοριοποίηση των δεδομένων.

2) ΝΟΡΜΑΛΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ:

Φύλο: Για να απεικονισθεί το φύλλο στο χ και να είναι σε αριθμητική τιμή ορίσαμε το θηλυκό με -1 και το αρσενικό με 1.

Όνομα : Το όνομα του επιβάτη δεν προσφέρει κάποια πληροφορία στην ταξινόμηση. Παρατηρήσαμε όμως ότι ο τίτλος θα μας βοηθήσει στην συμπλήρωση ηλικιών που δεν διαθέτουμε. Έτσι χωρίσαμε όλους τους τίτλους που υπάρχουν στο σύνολο των δεδομένων μας σε 4 κλάσεις ανάλογα με το range της ηλικίας. Βάλαμε δηλαδή στην ίδια κλάση τίτλους που έχουν πάνω κάτω την ίδια ηλικία. Διατρέξαμε τον πίνακα στην στήλη που περιέχεται το όνομα και κάναμε αντικατάσταση ως εξής:
•Miss. Mlle. Ms. : Ανήκουν στην κατηγορία 1 με ηλικία
•Mrs. Mme. Lady. Countess. : Ανήκουν στην κατηγορία 2
•Master. : Ανήκει στην κατηγορία 3.
•Mr. Dr. Rev. Major. Sir. Col. Capt. Jonkheer. Don. : Ανήκουν στην κατηγορία 4.

Ηλικία: Όπως αναφέραμε παραπάνω υπάρχουν ηλικίες οι οποίες λείπουν για αρκετούς επιβάτες και επειδή η ηλικία αποτελεί κρίσιμο χαρακτηριστικό για την κατηγοριοποίηση του κάθε επιβάτη αν σώθηκε ή όχι προβήκαμε στα εξής:
Αρχικά διατρέχουμε την στήλη με τις ηλικίες και όπου δεν υπάρχει βάζουμε 0 για να μπορούμε να το επεξεργαστούμε. Μετά, υπολογίζουμε τον μέσο όρο ανά κλάση τίτλου. Δηλαδή, όπου η ηλικία δεν είναι μηδέν και είναι κλάση 1,2,3 ή 4 υπολογίζουμε το avg1 avg2 avg3 ή avg4 αντίστοιχα. Οι τιμές c1, c2, c3, c4 μετράνε τα ορίσματα που έχουν που έχουν ηλικία, ώστε να κάνουμε την πράξη avg = avg/c για να βγάλουμε το μέσο όρο. Βάζουμε στους επιβάτες που δεν έχουν ηλικία το μέσο όρο της κλάσης στην οποία ανήκουν. Τέλος, διαγράφουμε την στήλη που περιέχει τις κλάσεις των τίτλων αφού δεν θα τη χρειαστούμε πια στην κατηγοριοποίηση.

Κλάση kNN:

(ο κώδικας περιέχεται στο αρχείο kNN.py)

Στην κλάση kNN υλοποιείται ο αλγόριθμος kNN(k nearest neighbors). Συγκεκριμένα, η ιδέα του αλγορίθμου είναι η εξής: με συγκεκριμένο k (αριθμός γειτόνων), εξετάζει κάποιο συγκεκριμένο χαρακτηριστικό κάθε γείτονα από κάποιο dataSet. Στην συνέχεια, με όρισμα κάποιο TestSet, αναθέτει σε κάθε στοιχείο του TestSet το χαρακτηριστικό που έχει η πλειοψηφία των k κοντινότερων γειτόνων του από το dataSet. Οι k κοντινότεροι γείτονες υπολογίζονται βάσει της ευκλείδιας απόστασης(βλέπε παρακάτω).

Στην δική μας περίπτωση, ο kNN έχει τα εξής ορίσματα:
- k: ο αριθμός των κοντινότερων γειτόνων
- X: το dataSet πάνω στο οποίο θα εκπαιδευτεί ο αλγόριθμος.
- label: το χαρακτηριστικό υπό εξέταση
- y: το testSet στα στοιχεία του οποίου θέλουμε να αναθέσουμε τα labels.

Αρχικά, καλώντας την EuclideanDistance, υπολογίζουμε τις αποστάσεις μεταξύ του y και κάθε γραμμή του X. Αυτό υλοποιείται από τις ακ'όλουθες γραμμές κώδικα:

```
for i in range(m):
distance = euclideanDistance(y, X[i,:])
distances.append(distance)
```

Στην συνέχεια, βάσει του δοθέντος k, βρίσκουμε τους k κοντινότερους γείτονες.

```
neighbors = []
for i in range(k):
neighbors.append(sortedDistances[i])
```

Τέλος, πρέπει να αναθέσουμε στο y το label των γειτόνων που καθορίστηκαν παραπάνω. Για να καθοριστεί η κλάση του y, βρήκαμε την κλάση που ανήκει η πλεοψηφία των καθορισμένων κοντινότερων γειτόνων του. Έπειτα, αναθέσαμε στο y αυτή την κλάση, όπως φαίνεται παρακάτω:

```
classVotes = {}
for j in range(k):
response = sortedLabels[j]
if response in classVotes:
classVotes[response] += 1
else:
classVotes[response] = 1
sortedVotes = collections.OrderedDict(sorted(classVotes.items()))
keys = sortedVotes.keys()
values = sortedVotes.values()
label = keys[values.index(max(values))]

return label
```

Κλάση EuclideanDistance:

(ο κώδικας περιέχεται στο αρχείο euclideanDistance.py)

```
def euclideanDistance(vectorA, vectorB):
```

```
# Computes the euclidean distance between two vectors. The
# two vectors must have the same size.

mA = len(vectorA) # length of vectorA
mB = len(vectorB) # length of vectorB
assert mA == mB, 'The two vectors must have the same size'
distance = 0
for i in range(mA):
distance = distance + pow((vectorA[i]-vectorB[i]),2)
distance = sqrt(distance)
return distance
```

Στην κλάση EuclideanDistance υλοποιείται η συνάρτηση euclidianDistance(Vector A, Vector B), η οποία υπολογίζει την ευκλείδια απόσταση μεταξύ 2 διανυσμάτων A και B. Συγκεκριμένα, χρειαζόμαστε την συγκεκριμένη συνάρτηση για να αποθηκεύσουμε τις αποστάσεις όλων των διανυσμάτων μεταξύ τους, για την υλοποίηση του αλγόριθμου του kNN. Τα 2 διανύσματα οφείλουν να έχουν το ίδιο μέγεθος. Η συνάρτηση επιστρέφει την απόσταση μεταξύ των 2 διανυσμάτων.

Κλαση Classify:

```
from numpy import *
from normalization import normalization
from kNN import kNN

def classify(trainSet, trainLabels, testSet):
    trainSet = normalization(trainSet)
    testSet = normalization(testSet)
```

Στην κλάση Classify υλοποιείται η συνάρτηση classify, η οποία έχει τα εξής ορίσματα:
-trainSet,
-trainLabels,
-testSet.

Τα ορίσματα αυτά περιγράφονται πιο κάτω στο σημείο που αναλύεται η Κλαση Main στο κομμάτι Γ. του report. Η classify καλείται από την main.

Αρχικά, κάνουμε normalization στα trainSet και testSet, καλώντας την συνάρτηση normalization της ομώνυμης Κλάσης. Η λειτουργία της Κλάσης normalization, όπως και η συνάρτηση normalization περιγράφεται πιο πάνω. Τα καινούρια δεδομένα που προκύπτουν μετά την επεξεργασία που τελείται από την normalization εκχωρούνται στα trainSet και testSet αντίστοιχα.

```
    #k = sqrt(testSet.shape[0]).astype(int)
    k=4
```

Στην συνέχεια, πρέπει να επιλέξουμε τον αριθμό που θα δώσουμε στο k, το οποίο αντιπροσωπεύει το πλήθος των k κοντινότερων γειτόνων, για τους οποίους θα εκτελεστεί ο αλγόριθμος kNN πιο μετά. Επιλέξαμε να βάλουμε k= 4, καθώς για αυτό το k πήραμε το καλύτερο accuracy, δηλαδη το Total Accuracy: 0.820426487093. Το δοκιμάσαμε επίσης για:
k=2 => Total Accuracy: 0.79012345679
k=5 => Total Accuracy: 0.803591470258
k=10 => Total Accuracy: 0.802469135802
k=20 => Total Accuracy: 0.783389450056
k=30 => Total Accuracy: 0.762065095398

Ένας άλλος τρόπος για την επιλογή του k είναι ένας εμπειρικός κανόνας τον οποίο βρίκαμε στο Internet μετά από αναζήτηση για την επιλογή του καλύτερου k. Συμφωνα με αυτόν, θέτοντας k ίσο με την τετραγωνική ρίζα του πλήθους των δεδομένων εκπαίδευσης οδηγούμαστε σε καλύτερα αποτελέσματα. Με τα συγκεκριμένα δεδομένα έχουμε, κάθε φορά που καλείται η classify και συνεπώς και ο kNN, k=9. Για αυτό το k, έχουμε Total Accuracy: 0.800224466891.

```
predictedLabels = zeros(testSet.shape[0])

    for i in range(testSet.shape[0]):
        predictedLabels[i] = kNN(k, trainSet, trainLabels, testSet[i,:])

    return predictedLabels
```

Στην συνέχεια, αρχικοποιούμε την δομή στην οποία μετά θα αποθηκευσουμε τα αποτελέσματα του kNN. Την έχουμε ονομάσει predictedLabels, βάζουμε σε κάθε θέση 0 και έχει το μέγεθος του testSet, δηλαδή των δεδομένων ελέγχου, καθώς πιο κάτω θα δούμε ότι θα της εκχωριθεί μία πρόβλεψη τιμής για κάθε διάνυσμα (vector) του συνόλου αυτού.

Ακολουθεί μια δομή επανάληψης for, στην οποία για κάθε i από 0 μέχρι το πλήθος των δεδομένων του testSet εκχωρείται το αποτέλεσμα της συνάρτησης kNN της κλάσης kNN, η οποία περιγράφεται πιο πάνω, στον predictedLabels, έτσι ώστε σε κάθε θέση i του predictedLabels να υπάρχει η πρόβλεψη (αν επιβίωσε ή όχι) για το διάνυσμα του testSet στην θέση i.

Τέλος, επιστρέφεται στην main ο predictedLabels.

Γ. Εκτέλεση

Εκτελέσαμε τον κώδικά μας με την δοθείσα main.py, η οποία δεν τροποποιήθηκε, όπως

ζητείται στην εκφώνηση του project. Συγκεκριμένα, στη main αρχικά ανακτούμε το dataSet που μας δίνεται με τις πληροφορίες των επιβατών και το αποθηκεύουμε στην μεταβλητή X. Στην συνέχεια, η στήλη "Survived" αποθηκεύεται στην μεταβλητή y, και διαγράφεται από τον πίνακα X. Έπειτα, εφαρμόζεται η διασταυρωμένη επικύρωση. Στις μεταβλητές trainSet και trainLabels αποθηκεύουμε, για καθεμία από τις 10 περιπτώσεις της διασταυρωμένης επικύρωσης, τα δεδομένα του συνόλου εκπαίδευσης και τις κατηγορίες στις οποίες ανήκουν. Στις μεταβλητές testSet και testLabels αποθηκεύουμε τα δεδομένα του συνόλου αξιολόγησης και τις κατηγορίες στις οποίες αυτά ανήκουν.

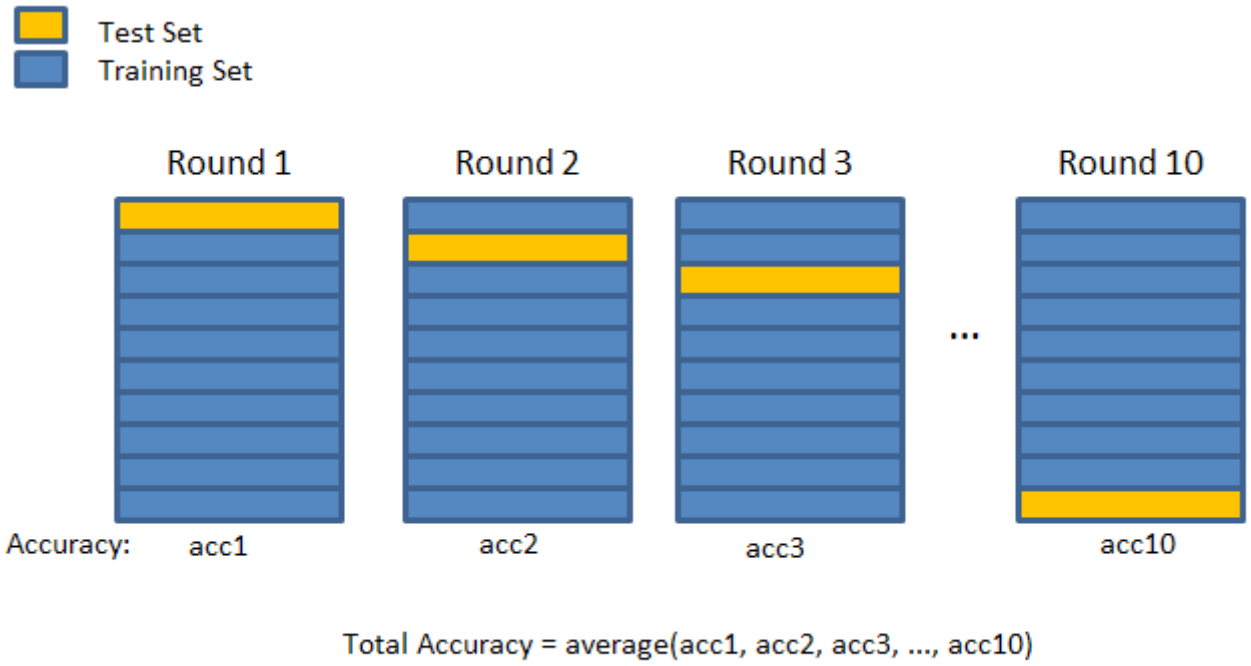
Μετά, η συνάρτηση classify() προβλέπει τις κατηγορίες των στιγμιοτύπων του συνόλου αξιολόγησης και τα αποτελέσματα συγκρίνονται με τις πραγματικές κατηγορίες των στιγμιοτύπων, που βρίσκονται στη μεταβλητή testLables, έτσι ώστε τέλος να μπορέσουμε να υπολογίσουμε την ακρίβεια για καθεμία από τις 10 περιπτώσεις, αλλά και τη συνολική ακρίβεια.

Το **cross validation (διασταυρωμένη επικύρωση)** χρησιμοποιείται για την αξιολόγηση μοντέλων και είναι μια μέθοδος για την εκτίμηση της απόδοσης ενός μοντέλου σε ένα ανεξάρτητο σύνολο δεδομένων. Χρησιμοποιείται κυρίως σε προβλήματα κατηγοριοποίησης, σαν το πρόβλημα που μας δίνεται.

Όταν έχουμε ένα μοντέλο με μια ή περισσότερες παραμέτρους και ένα σύνολο δεδομένων, το οποίο μπορούμε να χρησιμοποιήσουμε για να εκπαιδεύσουμε το μοντέλο, η διαδικασία εκπαίδευσης βελτιστοποιεί τις παραμέτρους του μοντέλου, έτσι ώστε αυτό να περιγράφει τα δεδομένα εκπαίδευσης όσο το δυνατόν ακριβέστερα. Αν χρησιμοποιήσουμε το μοντέλο σε ένα ανεξάρτητο σύνολο αξιολόγησης, το μοντέλο δεν θα περιγράφει το σύνολο αξιολόγησης τόσο καλά όσο το σύνολο εκπαίδευσης.
Αυτό είναι η υπερπροσαρμογή (overfitting), η οποία είναι πολύ πιθανό να συμβεί οταν το μοντέλο αποτελείται από πολλές παραμέτρους, όπως αυτό που χρησιμοποιούμε εμείς, ή όταν το σύνολο εκπαίδευσης είναι μικρό. Το cross validation παρέχει τρόπους εκτίμησης της απόδοσης που θα έχει ένα μοντέλο σε ένα υποθετικό σύνολο αξιολόγησης, όταν αυτό δεν είναι διαθέσιμο.

Εμείς χρησιμοποιούμε έναν non-exhaustive τρόπο για να αξιολογήσουμε τον ταξινομητή kNN που υλοποιήσαμε. Χρησιμοποιούμε **k-fold cross validation (διασταυρωμένη επικύρωση k δειγμάτων)** με k = 10.
Σύμφωνα με αυτή την μέθοδο, το αρχικό σύνολο δεδομένων (test.csv) διαιρείται σε k τυχαία υποσύνολα δεδομένων ίσου μεγέθους. Θεωρουμε ένα από αυτά σύνολο αξιολόγησης και τα υπόλοιπα k −1 υποσύνολα σύνολο εκπαίδευσης. Η διαδικασία αυτή επαναλαμβάνεται k φορές, έτσι ώστε κάθε k υποσύνολο να χρησιμοποιείται ακριβώς μια φορά ως σύνολο αξιολόγησης και τις υπόλοιπες k − 1 φορές ως κομμάτι του συνόλου εκπαίδευσης. Στην συνέχεια, τα k αποτελέσματα συνδυάζονται για να παράγουν μια κοινή αξιολόγηση του μοντέλου.
Η αξιολόγηση γίνεται με βάση την ακρίβεια (accuracy), δηλαδή τα στιγμιότυπα των

συνόλων αξιολόγησης που κατηγοριοποιήθηκαν σωστά προς το συνολικό πλήθος των στιγμιοτύπων.



Ένα ιδανικό (τέλειο) μοντέλο θα επέστρεφε ακρίβεια ίση με 1. Στην πράξη όμως, είναι πολύ δύσκολο ένα μοντέλο να προσεγγίσει το ιδανικό, οπότε τα μοντέλα συνήθως επιτυγχάνουν χαμηλότερες ακρίβειες.

Το cross vlidation μας δίνεται ήδη υλοποιημένο και περιέχεται στην συνάρτηση main της Κλάσης main.

Τέλος, υπολογίζεται και τυπώνεται το Accuracy της κατηγοριοποίησης που εκτελέσθηκε με βάση το πρόγραμμά μας. Υπολογίζεται με τις ακόλουθες εντολές:

```
print 'Accuracy: ' + str(float(correct)/(testLabels.size))
totalCorrect += correct
totalInstances += testLabels.size
print 'Total Accuracy: ' + str(totalCorrect/float(totalInstances))
```

Συγκεκριμένα, μετά από εκτέλεση του προγράμματος πήραμε τα εξής αποτελέσματα που φαίνονται παρακάτω:

Accuracy: 0.744444444444
Accuracy: 0.887640449438
Accuracy: 0.786516853933
Accuracy: 0.820224719101

Accuracy: 0.842696629213
Accuracy: 0.797752808989
Accuracy: 0.775280898876
Accuracy: 0.808988764045
Accuracy: 0.898876404494
Accuracy: 0.842696629213
Total Accuracy: 0.820426487093

Το συγκεκριμένο total Accuracy προέκυψε από διάφορες δοκιμές που σχετίζονται με τον αριθμό γειτόνων που επιλέγουμε κατα την εκτέλεση του αλγορίθμου kNN, δηλαδή με διάφορες δοκιμές της τιμής του k. Όπως προαναφέρθηκε, καταλήξαμε στην τιμή του k για την οποία έχουμε το μεγαλύτερο total accuracy, δηλαδή k=4.