3η Εργασία
Όραση Υπολογιστών
Μπούρχα Ιωάννα 58019

Επιβλέπων καθηγητής: Ιωάννης Πρατικάκης
Επιβλέπων εργαστηρίου: Λάζαρος Τσοχατζίδης

Ακαδημαϊκό έτος: 2022 - 2023

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Η παρούσα εργασία αναφέρεται στο πρόβλημα της ταξινόμησης δεδομένων σε κλάσεις. Ζητείται η υλοποίηση κώδικα σε python ο οποίος θα υλοποιεί ταξινόμηση πολλαπλών κλάσεων (multi-class classification).

Ο κώδικας θα βασιστεί στο μοντέλο Bag of Visual Words (BOVW) και για την παραγωγή του λεξικού χρησιμοποιήθηκε ο αλγόριθμος K-Means. Η ταξινόμηση των εικόνων έγινε με τους αλγορίθμους KNN και SVM των οποίων τα αποτελέσματα θα αναλυθούν ανάλογα των παραμέτρων τους.

Εισαγωγή δεδομένων τον κώδικα

Αρχικά, ξεκινάμε φορτώνοντας τις βιβλιοθήκες που θα χρησιμοποιήσουμε και τους φακέλους που περιέχουν τις εικόνες για την εκπαίδευση και αξιολόγηση του κώδικα.

```
import os
import cv2 as cv
import numpy as np

" INPUTS "
train_directory = "imagedb"
test_directory  = "imagedb_test"
```

Η βιβλιοθήκη os αξιοποιείται για τον προσδιορισμό της τοποθεσίας των αρχείων.

Η βιβλιοθήκη cv2 της OpenCV παρέχει ένα σύνολο εργαλείων για την επεξεργασία εικόνων υπό μορφή πινάκων.

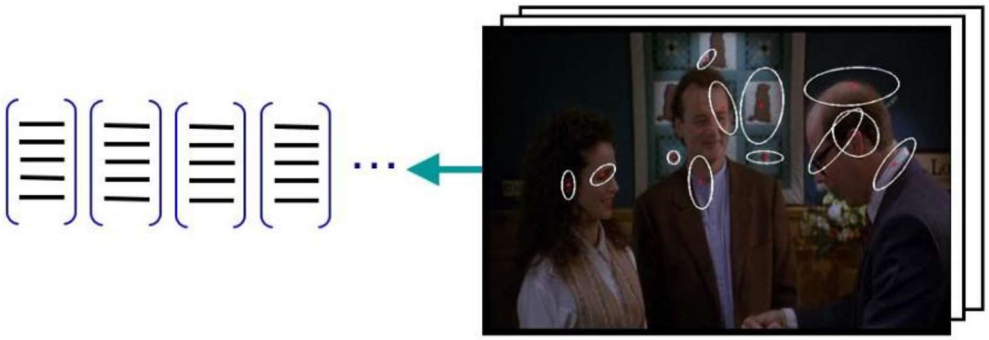
Η βιβλιοθήκη numpy παρέχει σύνολο εργαλείων για επεξεργασία διανυσμάτων και πινάκων.

Ο προσδιορισμός του συνόλου των φακέλων όπου περιέχονται ταξινομημένες οι εικόνες για την εκπαίδευση και αξιολόγηση του κώδικα γίνεται με τις εντολές:

```
train_folders_outer = os.listdir(train_directory)
test_folders_outer = os.listdir(test_directory)
```

Μοντέλο BOVW

Το μοντέλο αυτό βασίζεται στην ιδέα ότι κάθε εικόνα μπορεί να προσδιοριστεί από ένα σύνολο χαρακτηριστικών. Όπως είδαμε εκτενώς στην προηγούμενη εργασία, τα χαρακτηριστικά αυτά πρέπει να είναι αναλλοίωτα ως προς του διάφορους γεωμετρικούς μετασχηματισμούς που μπορεί να υποστεί εικόνα. Πρόκειται, δηλαδή, για τα σημεία ενδιαφέροντος (key-points) και τους περιγραφείς (descriptors) τους, η εξαγωγή των οποίων γίνεται με τον αλγόριθμο SIFT.



Εικόνα 1: Εξαγωγή περιγραφέων από την εικόνα

Οι περιγραφείς ομαδοποιούνται σε δοθέν πλήθος K ομάδες (clusters) με την βοήθεια του αλγορίθμου K-Means, δηλαδή με βάση το πόσο όμοιοι είναι μεταξύ τους. Η ομοιότητα κρίνεται με βάση την γεωμετρική απόσταση. Έτσι, κάθε ομάδα αποτελείται τελικά από ένα σύνολο περιγραφέων (descriptors) οι οποίοι παρουσίασαν την ελάχιστη απόσταση, δηλαδή την μέγιστη ομοιότητα, μεταξύ τους. Οι ομάδες (clusters) αυτές αποτελούν και τις λέξεις (words) του λεξικού. Το σύνολο των λέξεων ονομάζεται λεξικό (vocabulary).

Αλγόριθμος K-Means:

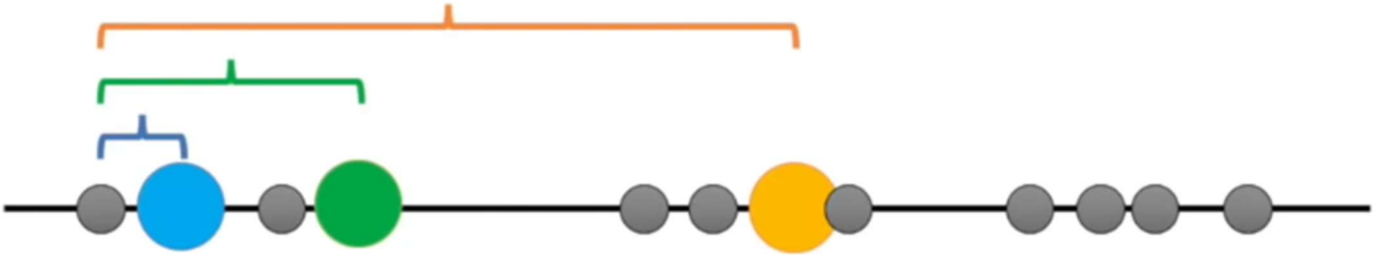
- Επιλογή K τυχαίων περιγραφέων που προσδιορίζουν το κέντρο των αντίστοιχων ομάδων.
- Υπολογισμός της ευκλείδειας απόστασης κάθε περιγραφέα με τα κέντρα των ομάδων.
- Ο κάθε περιγραφέας ανήκει στην ομάδα από της οποίας το κέντρο είχε την μικρότερη απόσταση.
- Προσδιορίζω το κέντρο κάθε ομάδας με βάση την ευκλείδεια απόσταση.
- Επαναλαμβάνω τα τρία τελευταία βήματα έως ότου ικανοποιηθεί το κέντρο των ομάδων μετατοπιστεί κατά μία συγκεκριμένη τιμή.

Όπως είναι κατανοητό, το αποτέλεσμα του αλγορίθμου εξαρτάται σε μεγάλο βαθμό από τα αρχικά κέντρα των ομάδων τα οποία προσδιορίζονται με τυχαίο τρόπο. Η ποιότητα του αποτελέσματος εξαρτάται από διακύμανση κάθε ομάδας. Δεδομένου ότι κάθε ομάδα πρέπει να έχει παρόμοιο πλήθος στοιχείων, το επιθυμητό αποτέλεσμα είναι αυτό με την μικρότερη διακύμανση. Συνεπώς, επαναλαμβάνουμε την προαναφερόμενη διαδικασία αρκετές φορές.

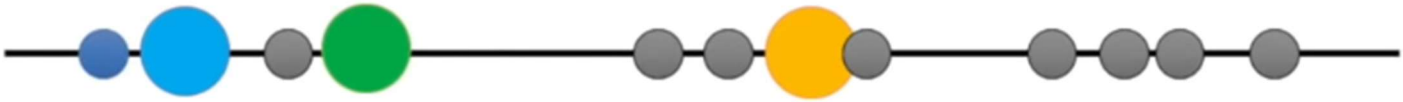
Ο αλγόριθμος τερματίζει και επιστρέφει το αποτέλεσμα έπειτα από συγκεκριμένο πλήθος επαναλήψεων και όταν τα κέντρα των ομάδων κάθε επανάληψης μετατοπιστούν κατά μία συγκεκριμένη τιμή. Για τον κώδικά μου οι τιμές που επιλέχτηκαν είναι αντίστοιχα 30 και 0.1, ίδιες με εκείνες του κώδικα του αντίστοιχου εργαστηρίου.

K-Means 1: Αρχικά μη ταξινομημένα δεδομένα εκπαίδευσης

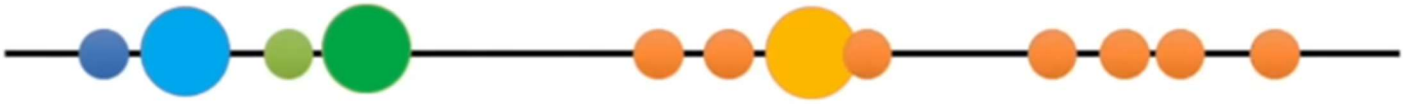
K-Means 2: Τυχαία επιλογή K = 3 κέντρων ομάδων



K-Means 3: Προσδιρισμός της απόστασης του πρώτου σημείου από κάθε κέντρο ομάδας



K-Means 4: Ανάθεση του πρώτου σημείου στην ομάδα από της οποίας το κέντρο είχε την μικρότερη απόσταση



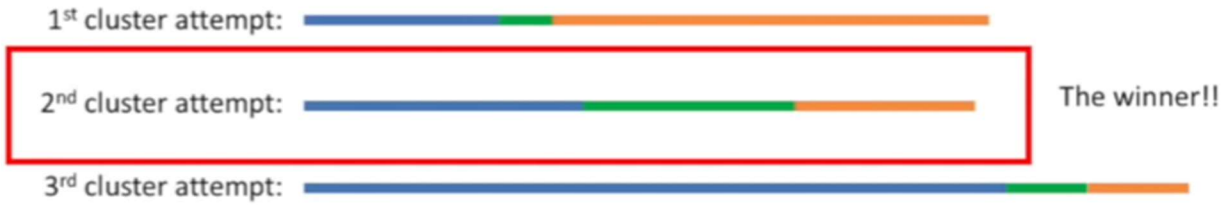
K-Means 5: Αποτέλεσμα μετά την επανάληψη για όλα τα σημεία

K-Means 6: Προσδιορισμός των νέων κέντρων κάθε ομάδας

K-Means 7: Διακύμανση πρώτης επανάληψης

K-Means 8: Επανάληψη με διαφορετικά αρχικά κέντρα

K-Means 9: Διακύμανση δεύτερης επανάληψης



K-Means 10: Διακύμανση για τρεις επαναλήψεις

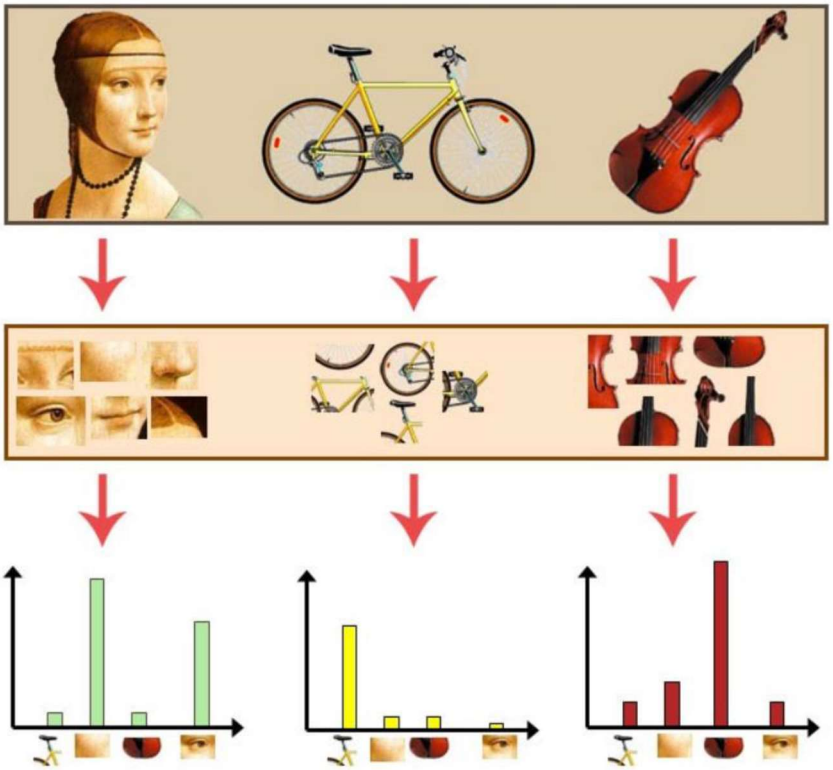
Ο κώδικας για την εξαγωγή των τοπικών χαρακτηριστικών και την δημιουργία του λεξικού είναι:

```
sift = cv.xfeatures2d_SIFT.create()

# ------------------- Create Vocabulary -------------------------
def create_vocabulary(train_directory, number_of_clusters, number_of_neighbors):
    """
    I used k-means cluster in order to create the vocabulary.
    The vocabulary may contain different number of clusters according to number_of_clus-
ters parameter.
    """
    print('---- Creating vocabulary ----')

    filename = str(number_of_clusters) + '_vocabulary.npy'

    if os.path.exists(filename):
        print("The vocabulary already exists.\n")
    else:
        # running for the first time
        train_descs = extract_features(train_directory)

        print('Creating clusters with K-Means alorithm ...')
        term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)  # stops when the centers move 0.1
        trainer = cv.BOWKMeansTrainer(number_of_clusters, term_crit, 1, cv.KMEANS_PP_CEN-
TERS)
        vocabulary = trainer.cluster(train_descs.astype(np.float32))

        filename = str(number_of_clusters) + '_vocabulary.npy'
        np.save(filename, vocabulary)
        print("The vocabulary has been created.\n")

def extract_features(train_directory):
    train_folders = os.listdir(train_directory)

    print('Extracting features...')
    train_descs = np.zeros((0, 128))  # 128 because we use SIFT algorithm
    for folder in train_folders:
        folder_path = os.path.join(train_directory, folder)
        files = os.listdir(folder_path)
        for file in files:
            # print(file)
            path = os.path.join(folder_path, file)  # path for every image
            desc = extract_local_features(path)
            if desc is None:  # prevent bugs
                print("\t\tNONE")
                continue
            train_descs = np.concatenate((train_descs, desc), axis=0)
        print('\t' + folder + ': DONE')
    return train_descs

def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img)
    desc = sift.compute(img, kp)
    return desc[1]
```

```
create_vocabulary(train_directory, number_of_clusters, number_of_neighbors)
```

Έχοντας δημιουργήσει το λεξικό, επιδιώκουμε να ταξινομήσουμε τις εικόνες εκπαίδευσης. Αρχικά, για κάθε εικόνα εξάγουμε τους περιγραφείς. Για κάθε περιγραφέα προσδιορίζεται η λέξη για την οποία βρίσκεται πιο κοντά. Έτσι, δημιουργείται ένας πίνακας όπου παρουσιάζεται η συχνότητα εμφάνισης κάθε λέξης στην εικόνα, δηλαδή το κανονικοποιημένο πλήθος των περιγραφέων της εικόνας που αντιστοιχεί σε κάθε λέξη του λεξικού. Προφανώς, το μέγεθος του

πίνακα ισούται με το μέγεθος του λεξικού. Ο πίνακας αυτός ονομάζεται ιστόγραμμα και αποτελεί τον περιγραφέα της εικόνας σύμφωνα με το μοντέλο BOVW (global descriptor).



Εικόνα 2: Ταξινόμηση εικόνων εκπαίδευσης (πρώτη γραμμή) με δοθέν λεξικό (δεύτερη γραμμή) και δημιουργία ιστογράμματος (τρίτη γραμμή)

Όπως φαίνεται και στην Εικόνα 2, για κάθε εικόνα εκπαίδευσης εξάγονται οι τοπικοί περιγραφείς έκαστος εκ των οποίων συγκρίνεται με το λεξικό προς την δημιουργία ιστογράμματος. Τελικώς, η εικόνα ανήκει στην λέξη (cluster) η οποία εμφανίζεται πιο συχνά, δηλαδή σε αυτήν με την μεγαλύτερη τιμή στο ιστόγραμμα.

Εφόσον οι εικόνες εκπαίδευσης είναι δοσμένες σε ξεχωριστούς φακέλους ανάλογα με το περιεχόμενό τους, μπορώ να δημιουργήσω ετικέτες, οι οποίες θα χρησιμοποιηθούν στην συνέχεια. Επαναλαμβάνω την ίδια διαδικασία και για τις εικόνες αξιολόγησης.

Ο αντίστοιχος κώδικας είναι:

```
def extract_global_descriptors(type, number_of_clusters, train_directory, test_directory,
number_of_neighbors):  # LAB def index

    # BOVW coding
    vocabulary = np.load(str(number_of_clusters) + '_vocabulary.npy')
    descriptor_extractor = cv.BOWImgDescriptorExtractor(sift,
cv.BFMatcher(cv.NORM_L2SQR))
    descriptor_extractor.setVocabulary(vocabulary)

    img_paths = []
    bow_descs = np.zeros((0, vocabulary.shape[0]))

    if type == 'training':
        print('---- Extracting Global Descriptors in Training Set----')
        filename_database = str(number_of_clusters) + '_bow_descs.npy'
        if os.path.exists(filename_database):
            print("bow_descs, labels and path have already been created")
        else:
            """
Creates the BOW, or in other words an array of every global_descriptor of each image in
training set, a database according to the features (global_descriptors) of each image in
training set
            """
            train_folders = os.listdir(train_directory)

            train_labels = np.zeros((0, 1))
            temp_label = np.zeros((1, 1))

            for folder in train_folders:
                folder_path = os.path.join(train_directory, folder)
                files = os.listdir(folder_path)

                for file in files:
                    path = os.path.join(folder_path, file)
                    img_paths.append(path)

                    # getting the global descriptor for each image in training set
                    img = cv.imread(path)
                    kp = sift.detect(img)
                    bow_desc = descriptor_extractor.compute(img, kp)

                    # collecting every global descriptor creating the Bag Of Words
                    bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)

                    train_labels = np.concatenate((train_labels, temp_label), axis=0)
                temp_label[0] = temp_label[0] + 1

                print("\t" + str(folder) + ": DONE")

            print('Train Database (global descriptors) created.')
            filename_database = str(number_of_clusters) + '_bow_descs.npy'
            np.save(filename_database, bow_descs)

            print('Train labels created.')
            filename_labels = 'labels.npy'
            np.save(filename_labels, train_labels)

            print('Created paths for images in training set.\n')
np.save('paths', img_paths)
    elif type == 'testing':
        test_folders = os.listdir(test_directory)
        print('---- Extracting Global Descriptors in Testing Set----')

        filename_database = str(number_of_clusters) + '_bow_descs_test.npy'
        if os.path.exists(filename_database):
            print("bow_descs, labels and path have already been created")
        else:
```

```
            print('Creating global descriptors and labels for testing set...')
            test_labels = np.zeros((0, 1))
            temp_label = np.zeros((1, 1))

            for folder in test_folders:
                folder_path = os.path.join(test_directory, folder)
                files = os.listdir(folder_path)
                for file in files:
                    path = os.path.join(folder_path, file)
                    img_paths.append(path)

                    # getting the global descriptor for each image in training set
                    img = cv.imread(path)
                    kp = sift.detect(img)
                    bow_desc = descriptor_extractor.compute(img, kp)

                    bow_descs = np.concatenate((bow_descs, bow_desc),axis=0)
                    test_labels = np.concatenate((test_labels, temp_label), axis=0)
                temp_label[0] = temp_label[0] + 1
                print("\t" + str(folder) + ": DONE")

            print('Test Database (global descriptors) created.')
            filename_database = str(number_of_clusters) + '_bow_descs_test.npy'
            np.save(filename_database, bow_descs)

            print('Test labels created.')
            filename_labels = 'labels_test.npy'
            np.save(filename_labels, test_labels)

            print('Created paths for images in testing set.\n')
            np.save('paths_test', img_paths)
    else:
        print("Typo in the final parameter in extract_global_descriptors.")
        print("\tIt can be 'training' or 'testing' ")
        exit(-1)
```

```
extract_global_descriptors('training', number_of_clusters, train_directory, test_direc-
tory, number_of_neighbors)
extract_global_descriptors('testing', number_of_clusters, train_directory, test_direc-
tory, number_of_neighbors)
```

Τονίζεται ότι η κλάση της εικόνας, τόσο του συνόλου εκπαίδευσης όσο και του συνόλου αξιολόγησης, προκύπτει μόνο από τον φάκελο στον οποίον βρίσκεται.

Το πρόβλημα της ταξινόμησης

Δοθέντων δεδομένων σε γνωστές κλάσεις (σύνολο εκπαίδευσης, training dataset), ζητείται η ταξινόμηση άγνωστης κλάσης δεδομένων.

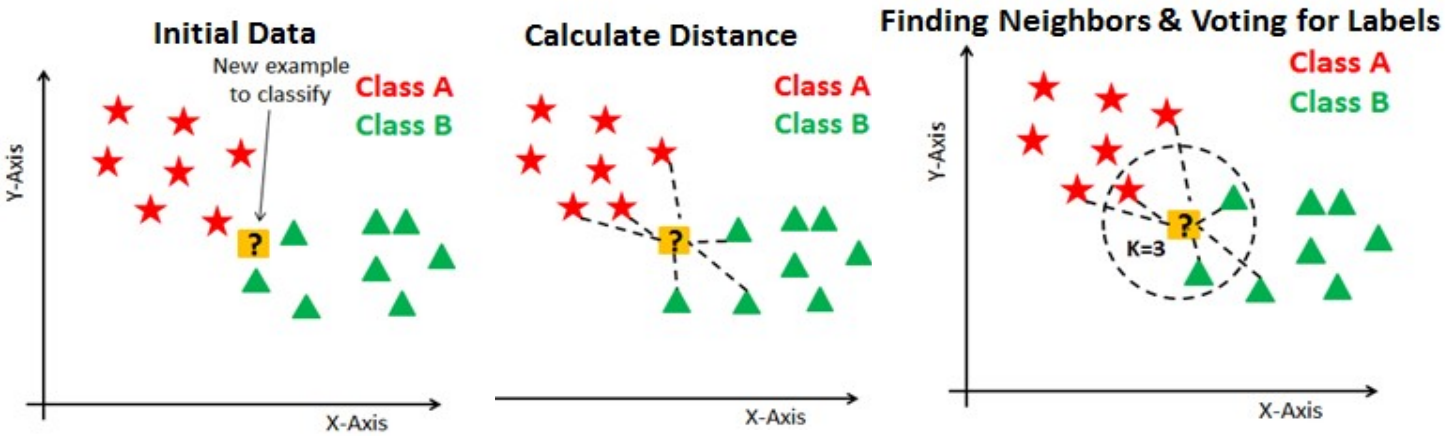
Για την περίπτωση παραπάνω των δύο κλάσεων ακολουθείται είτε η στρατηγική one-versus-all είτε η στρατηγική one-versus-one. Στην πρώτη περίπτωση υπάρχουν τόσοι ταξινομητές όσοι και οι κλάσεις και ο κάθε ταξινομητής προβλέπει αν το δείγμα δεδομένων ανήκει στην κλάση που αναφέρεται ή όχι. Στη δεύτερη, ο κάθε ταξινομητής χρησιμοποιεί δείγματα δεδομένων από δύο κλάσεις και προβλέπει σε ποια από τις δύο κλάσεις ανήκει το άγνωστο δείγμα που τίθεται προς ταξινόμηση.

Παρότι υπάρχουν αρκετά μοντέλα ταξινόμησης στην παρούσα εργασία θα εξετάσουμε τα μοντέλα KNN και SVM.

K Nearest Neighbors – one versus one

Ο αλγόριθμος αυτός είναι ένας από τους απλούστερους αλγόριθμους ταξινόμησης. Πρόκειται για μία μη παραμετρική μέθοδος που χρησιμοποιεί τα k κοντινότερα σημεία του δειγματοχώρου στο δείγμα προς ταξινόμηση για την εξαγωγή της κλάσης στην οποία προβλέπεται να ανήκει το δείγμα προς ταξινόμηση. Η επιλογή της παραμέτρου k εξαρτάται από τα δεδομένα και μπορεί να επιλεχθεί χρησιμοποιώντας διάφορες τεχνικές. Η κλάση στην οποία ανήκει τελικά το δείγμα προκύπτει από την πλειοψηφία της κλάσης των γειτόνων.

Ουσιαστικά, η κλάση κάθε εικόνας αξιολόγησης προκύπτει από την πλειοψηφία της κλάση στην οποία ανήκουν οι K κοντινότεροι, σύμφωνα με την ευκλείδεια απόσταση, περιγραφείς εικόνων σύμφωνα με το μοντέλο BOVW (global descriptor).



Εικόνα 3: Διάγραμμα περιγραφής KNN ταξινομητή

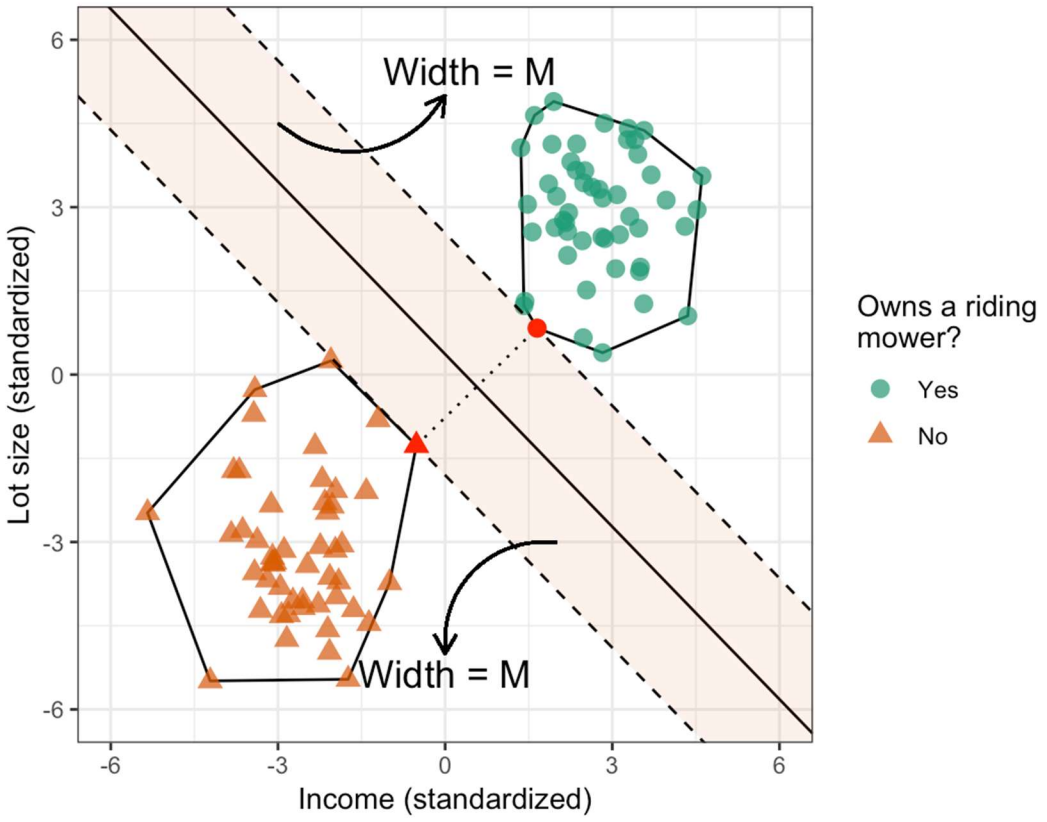
Ο αντίστοιχος κώδικας είναι:

```
" KNN CLASSIFIER"
def euclidean_distance(d1, d2):  # Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
    distances = []
    for i in range(d2.shape[0]):
        distance = np.sqrt(np.sum((d1 - d2[i]) ** 2))
        distances.append(distance)
    return np.asarray(distances)  # turning list into numpy array

def knn_classifier(number_of_neighbors, number_of_clusters, number_of_classes):
    bow_descs_training = np.load(str(number_of_clusters) + '_bow_descs.npy')
    bow_descs_testing = np.load(str(number_of_clusters) + '_bow_descs_test.npy')

    labels_training = np.load('labels.npy')

    predictions_final = []

    for i in range(bow_descs_testing.shape[0]):
        distances = euclidean_distance(bow_descs_testing[i], bow_descs_training)

        idx = np.argpartition(distances, number_of_neighbors)
        idx = idx[:number_of_neighbors]
        """ np.argpartition returns the indexes of number_of_neighbors smallest values in
the given array. It's not guaranteed to be in order from smallest to largest."""

        predictions = np.zeros(number_of_classes)
        for m in range(len(idx)):
            p = idx[m]
            if labels_training[p] >= 0:
                c = int(labels_training[p])
                predictions[c] += 1
        predictions_final.append(np.argmax(predictions))

    return predictions_final  # returns the class of every global_descriptor, aka of
every image in testing set
```

Support Vector Machine – one versus all

Ο αλγόριθμος αυτός είναι ανήκει στην κατηγορία των δυαδικών ταξινομητών (binary classifications), δηλαδή ταξινομεί το νέο δεδομένο σε δύο κλάσεις. Δοθέντος το σύνολο των εικόνων εκπαίδευσης και των αντίστοιχων κλάσεων στις οποίες ανήκουν, τα δείγματα αναπαρίστανται ως σημεία στον δειγματοχώρο. Με τη χρήση υπερεπιπέδων διαχωρίζονται τα δείγματα στις δύο κλάσειw, των οποίων η απόσταση είναι η μέγιστη δυνατή.

Όταν δοθεί μία νέα εικόνα προς ταξινόμηση, τοποθετείται στον δειγματοχώρο ο global descriptor και εντοπίζεται η κλάση της με βάση το υπερεπίπεδο στο οποίο βρέθηκε.



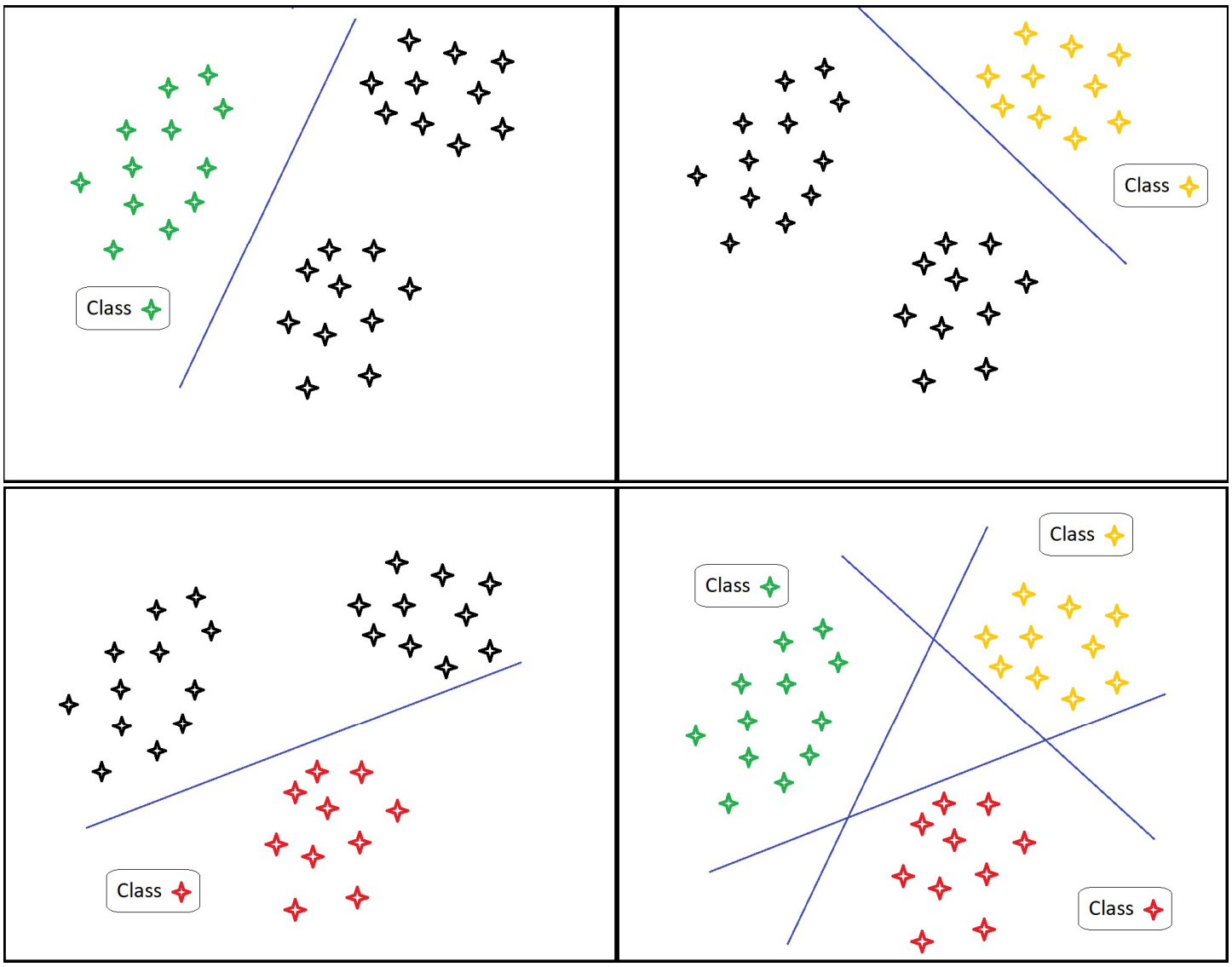
Εικόνα 4: Ταξινόμηση SVM

Το υπερεπίπεδο αυτό, πυρήνας του αλγορίθμου, μπορεί να περιγραφθεί με διάφορες εξισώσεις. Μερικά παραδείγματα των πυρήνων που προσφέρει η OpenCV είναι:

Enumerator	
CUSTOM	Returned by SVM::getKernelType in case when custom kernel has been set
LINEAR	Linear kernel. No mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option. $K(x_i, x_j) = x_i^T x_j$.
POLY	Polynomial kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + coef0)^{degree}, \gamma > 0$.
RBF	Radial basis function (RBF), a good choice in most cases. $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0$.
SIGMOID	Sigmoid kernel: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + coef0)$.
CHI2	Exponential Chi2 kernel, similar to the RBF kernel: $K(x_i, x_j) = e^{-\gamma \chi^2(x_i, x_j)}, \chi^2(x_i, x_j) = (x_i - x_j)^2/(x_i + x_j), \gamma > 0$.
INTER	Histogram intersection kernel. A fast kernel. $K(x_i, x_j) = min(x_i, x_j)$.

Στον κώδικα χρησιμοποιήθηκαν οι: RBF, CHI2, INTER και LINEAR.

Εφόσον στο δικό μας πρόβλημα έχουμε περισσότερες των δύο κλάσεις, η κατηγοριοποίηση θα γίνει ανάλογα με το εάν ανήκει η εικόνα σε μία κλάση κάθε φορά ή όχι. Επομένως, θα χρειαστεί να καλέσουμε τον ταξινομητή SVM για όσες κλάσεις διαθέτουμε. Η τελική κλάση προκύπτει από το υπερεπίπεδο της κλάσης με το οποίο «ταιριάζει» περισσότερο. Το ταίριασμα υλοποιείται κατευθείαν από έτοιμη συνάρτηση της βιβλιοθήκης OpenCV.



Εικόνα 5: Ταξινόμηση με SVM σε πρόβλημα τριών κλάσεων

Ο αντίστοιχος κώδικας είναι:

```
" SVM "
def svm_create(img_ind, bow_descs, kernel, name_of_class, number_of_clusters, num-
ber_of_neighbors):
    filename = str(number_of_clusters) + '_svm_' + str(kernel) + '_' + str(name_of_class)
    if os.path.exists(filename):
        print(str(name_of_class) + " has already been trained with SVM.")
    else:
        svm = cv.ml.SVM_create()
        svm.setType(cv.ml.SVM_C_SVC)
        svm.setTermCriteria((cv.TERM_CRITERIA_COUNT, 100, 1.e-06))

        if   kernel == 'RBF':        svm.setKernel(cv.ml.SVM_RBF)
        elif kernel == 'CHI2':       svm.setKernel(cv.ml.SVM_CHI2)
        elif kernel == 'LINEAR':     svm.setKernel(cv.ml.SVM_LINEAR)
        elif kernel == 'INTER':      svm.setKernel(cv.ml.SVM_INTER)
        else:
            print("Typo in the kernel parameter in svm_classifier.")
```

```
            print("\tIt can be 'RBF' or 'CHI2' or 'LINEAR' or 'POLY' or 'INTER' or 'SIG-
MOID' ")
            exit(-1)

        svm.trainAuto(bow_descs.astype(np.float32), cv.ml.ROW_SAMPLE, img_ind)
        svm.save(str(number_of_clusters) + '_svm_' + str(kernel) + '_' +
str(name_of_class))
        print("\t" + str(name_of_class) + ": Done")


def svm_1vsAll(kernel, train_folders, number_of_clusters, number_of_neighbors):
    svm = cv.ml.SVM_create()
    bow_descs = np.load(str(number_of_clusters) + '_bow_descs.npy')
    paths = np.load('paths.npy')

    " use as many svm classifiers as the classes in which the training set is divided "
    for i in range(len(train_folders)):
        name_of_class = train_folders[i]

        img_ind = np.array([name_of_class in a for a in paths], np.int32)

        svm_create(img_ind, bow_descs, kernel, name_of_class, number_of_clusters, num-
ber_of_neighbors)
        svm.load(str(number_of_clusters) + '_svm_' + str(kernel) + '_' +
str(name_of_class))


def svm_1vsAll_classifier(bow_desc, test_folders, train_folders, number_of_clusters, num-
ber_of_neighbors, kernel):
    " has to be called for each image in testing set"
    svm = cv.ml.SVM_create()

    prediction = []
    for i in range(len(test_folders)):
        svm = svm.load(str(number_of_clusters) + '_svm_' + str(kernel) + '_' +
str(train_folders[i]))
        prediction.append(svm.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1][0][0])

    return prediction
```

Αξιολόγηση Συστήματος

Για την αξιολόγηση του συστήματος καλώ τους ακόλουθες συναρτήσεις χρησιμοποιώντας ως είσοδο τις εικόνες αξιολόγησης.

```
def knn_classifier_test(number_of_neighbors, number_of_clusters, number_of_classes,
train_folders, test_folders):
    labels_test = np.load('labels_test.npy')  # used to test the accuracy
    predictions = knn_classifier(number_of_neighbors, number_of_clusters, number_of_clas-
ses)

    # get the size of every folder in test directory
    test_folder_size = []
    for folder in test_folders:
        folder_path = os.path.join(test_directory, folder)
        files = os.listdir(folder_path)
        test_folder_size.append(len(files))

    accuracy = []

    failure_names = []
    failure_names_index = []
    paths = np.load('paths_test.npy')  # used to find the failed images
    for i in range(labels_test.shape[0]):
        _, a = os.path.split(paths[i])
        if predictions[i] == labels_test[i]:
            accuracy.append(1)
        else:
            accuracy.append(0)
            _, failed_file_name = os.path.split(paths[i])
            failure_names.append(failed_file_name)
            failure_names_index.append(i)

    accuracy_tot = sum(accuracy) / labels_test.shape[0] * 100
    print("Accuracy: " + str("%.2f" % accuracy_tot) + "%")
    first = 0
    for x in range(len(test_folder_size)):
        last = first + test_folder_size[x]
        temp = accuracy[first: last]
        accuracy_in_folder = sum(temp) / labels_test.shape[0] * 100
        first = last

        print("\t" + str(test_folders[x]) + "\t" + str("%.2f" % accuracy_in_folder) +
"%")

    print("Failed attempts:" + str(labels_test.shape[0] - sum(accuracy)))
    for x in range(len(failure_names)):
        index = failure_names_index[x]
        training_class = train_folders[predictions[index]]
        test_class = test_folders[int(labels_test[index])]
        print("\t" + str(failure_names[x]) + "\tAlgorithm: " + str(training_class) +
"\t\tTrue: " + str(test_class))

    return accuracy
```

```
def svm_1vsAll_classifier_test(test_directory, train_folders, number_of_clusters, num-
ber_of_neighbors):
    print('---- Testing SVM 1 VS ALL Classification: ----')

    vocabulary = np.load(str(number_of_clusters) + '_vocabulary.npy')
    descriptor_extractor = cv.BOWImgDescriptorExtractor(sift,
cv.BFMatcher(cv.NORM_L2SQR))
    descriptor_extractor.setVocabulary(vocabulary)

    test_folders = os.listdir(test_directory)
    class_folder = 0

    accuracy = []
    failure = 0
    failure_names = []
    failure_predicted = []
    failure_test = []
    thesi = 0
    for folder in test_folders:
        folder_path = os.path.join(test_directory, folder)
        files = os.listdir(folder_path)

        success = 0
        for file in files:
            path = os.path.join(folder_path, file)

            # getting the global descriptor for each image in testing set
            img = cv.imread(path)
            kp = sift.detect(img)
            bow_desc = descriptor_extractor.compute(img, kp)

            # classifying with svm 1 vs all
            result = np.asarray(
                svm_1vsAll_classifier(bow_desc, test_folders, train_folders,
                                      number_of_clusters, number_of_neighbors))

            class_predicted = np.argmin(result)
            # testing the outcome of my algorithm
            if class_folder == class_predicted:
                success += 1
            else:
                failure += 1
                failure_names.append(file)
                failure_predicted.append(class_predicted)
                failure_test.append(class_folder)
            thesi += 1

        class_folder += 1
        accuracy.append(success)

    accuracy_tot = sum(accuracy) / (sum(accuracy) + failure) * 100
    print("Accuracy: " + str("%.2f" % accuracy_tot) + "%")
    for i in range(len(accuracy)):
        accurasy_class = accuracy[i] / (sum(accuracy) + failure) * 100
        print("\t" + str(train_folders[i]) + ": \t" + str("%.2f" % accurasy_class) + "%")

    print("Failed attempts:" + str(failure))
    for x in range(len(failure_names)):
        training_class = train_folders[failure_predicted[x]]
        test_class = test_folders[int(failure_test[x])]
        print("\t" + str(failure_names[x]) + "\tAlgorithm: " + str(training_class) +
"\t\t True: " + str(test_class))

    return accuracy_tot
```

Συνεπώς, το σύστημά μου υλοποιείται με την κλήση της ακόλουθης συνάρτησης.

```
def my_code(number_of_clusters, number_of_neighbors, kernel, train_directory, test_direc-
tory):
    train_folders_outer = os.listdir(train_directory)
    test_folders_outer = os.listdir(test_directory)

    " BOVW Creation "
    create_vocabulary(train_directory, number_of_clusters)
    extract_global_descriptors('training', number_of_clusters, train_directory, test_di-
rectory, number_of_neighbors)
    extract_global_descriptors('testing', number_of_clusters, train_directory, test_di-
rectory, number_of_neighbors)

    " KNN Classifier "
    print('Vocabulary size: ' + str(number_of_clusters))
    print ('\n---- Testing KNN Classification: ----')
    for n in range(len(number_of_neighbors)):
        knn_classifier(number_of_neighbors[n], number_of_clusters, len(train_fold-
ers_outer))
        knn_classifier_test(number_of_neighbors[n], number_of_clusters, len(train_fold-
ers_outer), train_folders_outer, test_folders_outer)
        custom_print('\n')

    print ("\n")

    " SVM "
    custom_print('---- Testing SVM 1 VS ALL Classification: ----')
    for k in range(len(kernels)):
        print ("KERNEL: " + str(kernel[k]))
        svm_1vsAll(kernel[k], train_folders_outer, number_of_clusters, number_of_neigh-
bors)
        svm_1vsAll_classifier_test(test_directory, train_folders_outer, number_of_clus-
ters, number_of_neighbors, kernel[k])
        custom_print("\n")
```

```
for c in range(len(number_of_clusters)):

    file_name = str('console_' + str(number_of_clusters[c]))
    f = open((file_name + '.txt'), 'w')

    my_code(number_of_clusters[c], number_of_neighbors, kernels, train_directory,
test_directory)

    f.close()

print("**************************************************************************
************************")
    print("\t\t\t\t\t\t\tSaved outputs in txt file: " + file_name)

print("**************************************************************************
************************\n\n")
```

Σε αυτό το σημείο να τονιστεί ότι ορισμένα αποτελέσματα που εκτυπώνονται στην οθόνη καταγράφονται παράλληλα και σε ένα αρχείο κειμένου. Για την εκτύπωση αυτών των δεδομένων τόσο στην κονσόλα όσο και στο αρχείο χρησιμοποιώ την συνάρτηση:

```
def custom_print(message_to_print):
    print(message_to_print)
    log_file = (file_name + '.txt')
    with open(log_file, 'a') as of:
        of.write(str(message_to_print) + '\n')
```

Η αξιολόγηση θα γίνει για:

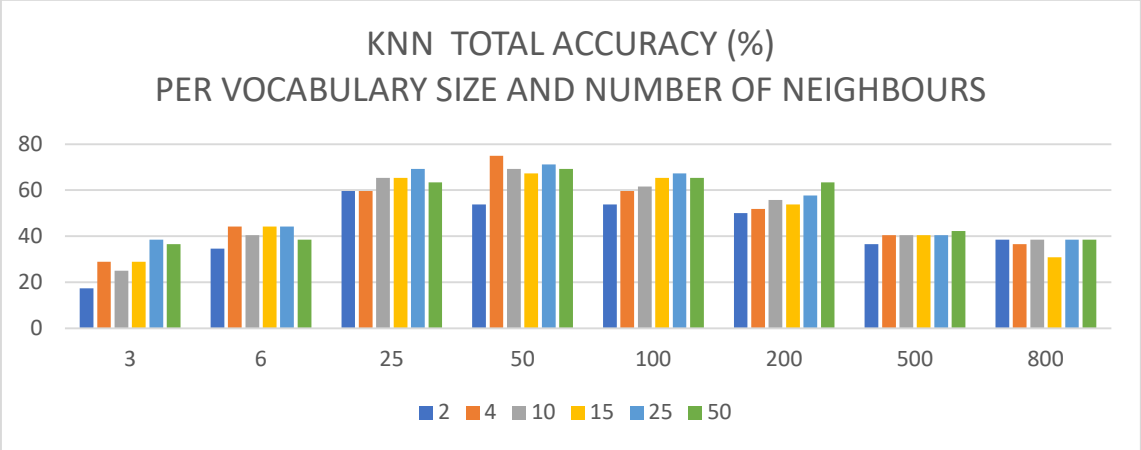
- ⇨ διαφορετικά ως προς το μέγεθος λεξικά (K- Means), δηλαδή για διαφορετικές τιμές της μεταβλητής number_of_clusters,
- ⇨ διαφορετικό πλήθος γειτόνων (KNN), δηλαδή για διαφορετικές τιμές της μεταβλητής number_of_neighbors
- ⇨ διαφορετικό είδος πυρήνα (SVM).

Συγκεκριμένα, οι τιμές που επιλέχτηκαν είναι:

```
number_of_clusters  = [3, 6, 25, 50, 100, 200, 500, 800]
number_of_neighbors = [2, 4, 10, 15,  25,  50]

kernels = ['RBF', 'CHI2', 'LINEAR', 'INTER']
```

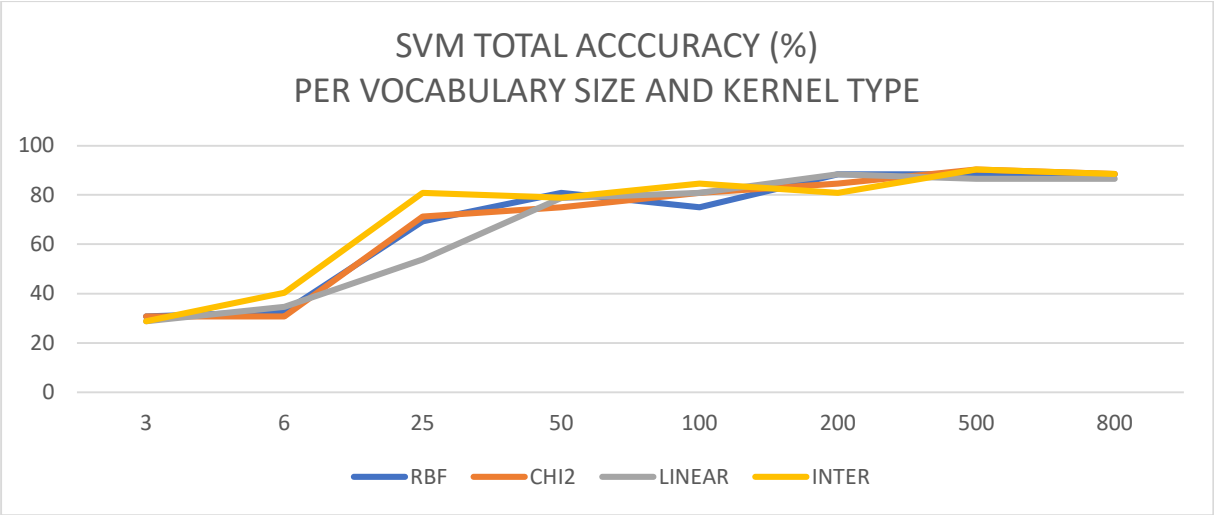
Οι πίνακες και τα αντίστοιχα διαγράμματα που δείχνουν την απόδοση των δύο ταξινομητών φαίνονται παρακάτω:

ACCURACCY	KNN Number of Neighbors					
Words	**2**	**4**	**10**	**15**	**25**	**50**
3	17,31	28,85	25,01	28,84	38,48	36,55
6	34,62	44,24	40,39	44,23	44,24	38,47
25	59,62	59,61	65,38	65,39	69,22	63,45
50	53,84	75	69,23	67,31	71,15	69,22
100	53,84	59,61	61,53	65,38	67,31	65,38
200	50,01	51,91	55,77	53,84	57,68	63,46
500	36,54	40,38	40,38	40,38	40,38	42,31
800	38,46	36,53	38,46	30,77	38,46	38,45



FAILURES	KNN Number of Neighbors					
Words	**2**	**4**	**10**	**15**	**25**	**50**
3	43	37	39	37	32	33
6	34	29	31	29	29	32
25	21	21	18	18	16	19
50	24	13	16	17	15	16
100	24	21	20	18	17	18
200	26	25	23	24	22	19
500	33	31	31	31	31	30
800	32	33	32	36	38,46	32

ACCURACCY	SVM KERNEL			
Words	**RBF**	**CHI2**	**LINEAR**	**INTER**
3	30,77	30,76	28,85	28,85
6	32,69	30,77	34,62	40,39
25	69,23	71,16	53,85	80,77
50	80,77	75	78,76	78,84
100	75	80,76	80,76	84,61
200	88,45	84,61	88,45	80,76
500	88,45	90,37	86,53	90,37
800	88,45	88,45	86,53	88,45



FAILURES	SVM KERNEL			
Words	**RBF**	**CHI2**	**LINEAR**	**INTER**
3	36	36	37	37
6	35	36	34	31
25	16	15	24	10
50	10	13	11	11
100	13	10	10	8
200	6	8	6	10
500	6	5	7	5
800	6	6	7	6

Από τα παραπάνω συμπεραίνουμε ότι:

- ο ταξινομητής SVM στην πλειοψηφία των περιπτώσεων δίνει πιο ακριβή αποτελέσματα
- η αύξηση του μεγέθους του λεξικού οδηγεί σε
    - μείωση της ακρίβειας για τον KNN
    - αύξηση της ακρίβειας για τον SVM

Τα καλύτερα αποτελέσματα για τον KNN δίνονται για μέγεθος λεξικού 50 και πλήθος γειτόνων 10, ενώ για τον SVM για μέγεθος λεξικού 500 και φίλτρο CHI2 ή LINEAR. Για μικρότερα λεξικά, το φίλτρο INTER δίνει καλύτερα αποτελέσματα.

Ακόμη, παρατηρώντας τις αστοχίες ανά κατηγορία, επιδιώκεται η εύρεση της αιτίας των σφαλμάτων αυτών.

Για την κατηγορία των μηχανών (motorbike), ο αλγόριθμος κατέταξε τις εικόνες ως ποδήλατο (touring-bike). Δεδομένου των γεωμετρικών χαρακτηριστικών των δύο αντικειμένων είναι λογική η ύπαρξη αυτού του σφάλματος. Αξιοσημείωτο είναι ότι δεν παρατηρείται αντίστοιχο σφάλμα και στην κλάση των ποδηλάτων, όπως ίσως αναμενόταν. Αυτό οφείλεται στο γεγονός ότι οι εικόνες των μηχανών είναι λιγότερες από εκείνες των ποδηλάτων και ότι οι εικόνες των μηχανών είναι μοναδικής οπτικής γωνίας (πλάγια), σε αντίθεση με εκείνη των ποδηλάτων που υπάρχουν αρκετές εικόνες με διαφορετικές οπτικές γωνίες.

Για την κατηγορία των αεροπλάνων, ο αλγόριθμος κατέταξε τις εικόνες ως σχολικό λεωφορείο (school-bus), όπως ακριβώς και για την κατηγορία των αυτοκινήτων (car-side).