Μαρία Αρετή

Γερμανού 57807

7ο Εξάμηνο 2022

Όραση Υπολογιστών

Τεχνική Αναφορά - Εργασία 3η

Η τεχνική αναφορά περιλαμβάνει:

I. Την θεωρητική ανάλυση της εργασίας.

II. Την περιγραφή του κώδικα που παραδόθηκε.

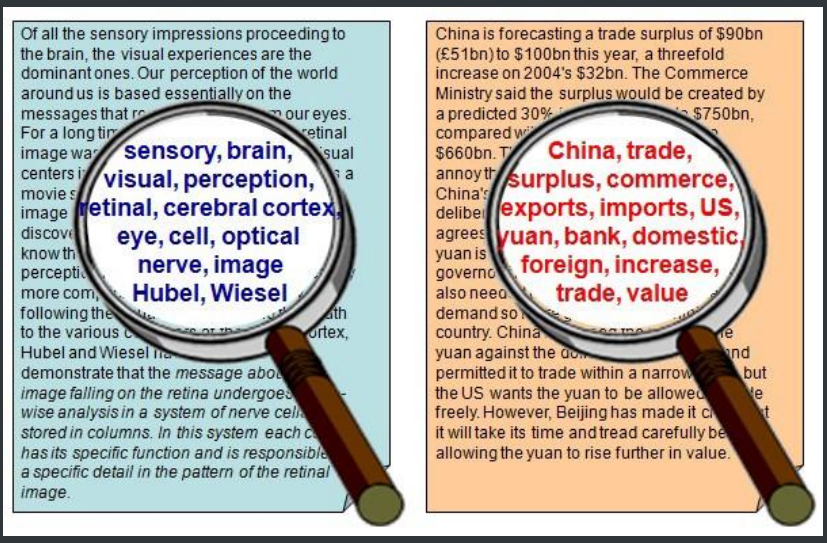
III. Την ανάλυση των αποτελεσμάτων.

Ι. Θεωρητική ανάλυση της εργασίας.

Σύμφωνα με την εκφώνηση της τρίτης εργασίας, θα πρέπει να επιλυθεί ένα πρόβλημα ταξινόμησης πολλαπλών κλάσεων. Κατεβάζοντας υποσύνολα βάσης εικόνων απο το "kaggle" γίνεται η εκπαίδευση και η δοκιμή του συστήματος και έπειτα η αξιολόγησή του. Αναλυτικά θα παρουσιαστεί κάθε βήμα στην περιγραφή του κώδικα που θα γίνει παρακάτω και μαζί θα αναλυθούν και τα αποτελέσματα. Προς το παρόν ας παρουσιαστεί βηματικά η θεωρητική ανάλυση και μεθοδολογία που θα ακολουθηθεί παρακάτω.

// Δημιουργία οπτικών λεξικών:

Για την αυτόματη ταξινόμηση άγνωστων εικόνων σε κλάσεις που έχουν εκπαιδευτεί με βάση το περιεχόμενο των εικόνων, χρησιμοποιείται για τον σκοπό αυτό ονομάζεται "Bag Of Visual Words". Γενικά αυτό το μοντέλο χρησιμοποιείται συχνά σε προβλήματα ταξινόμησης εικόνων και όπως φαίνεται και από το όνομά του αποτελεί "σάκο" λέξεων. Στην πραγματικότητα αυτό το σύνολο δεν αποτελείται από λέξεις, αλλά από χαρακτηριστικά εικόνων που υπολογίζονται και εξάγονται από κάθε εικόνα που χρησιμοποιείται. Ανάλογα με την συχνότητα εμφάνισης των "λέξεων" σε κάθε εικόνα εξάγουμε τα τοπικά χαρακτηριστικά και φτιάχνουμε ένα ιστόγραμμα. Να αναφερθεί πως ακόμα και με την αλλαγή του προσανατολισμού των εικόνων τα ίδια χαρακτηριστικά και πάλι εντοπίζονται.

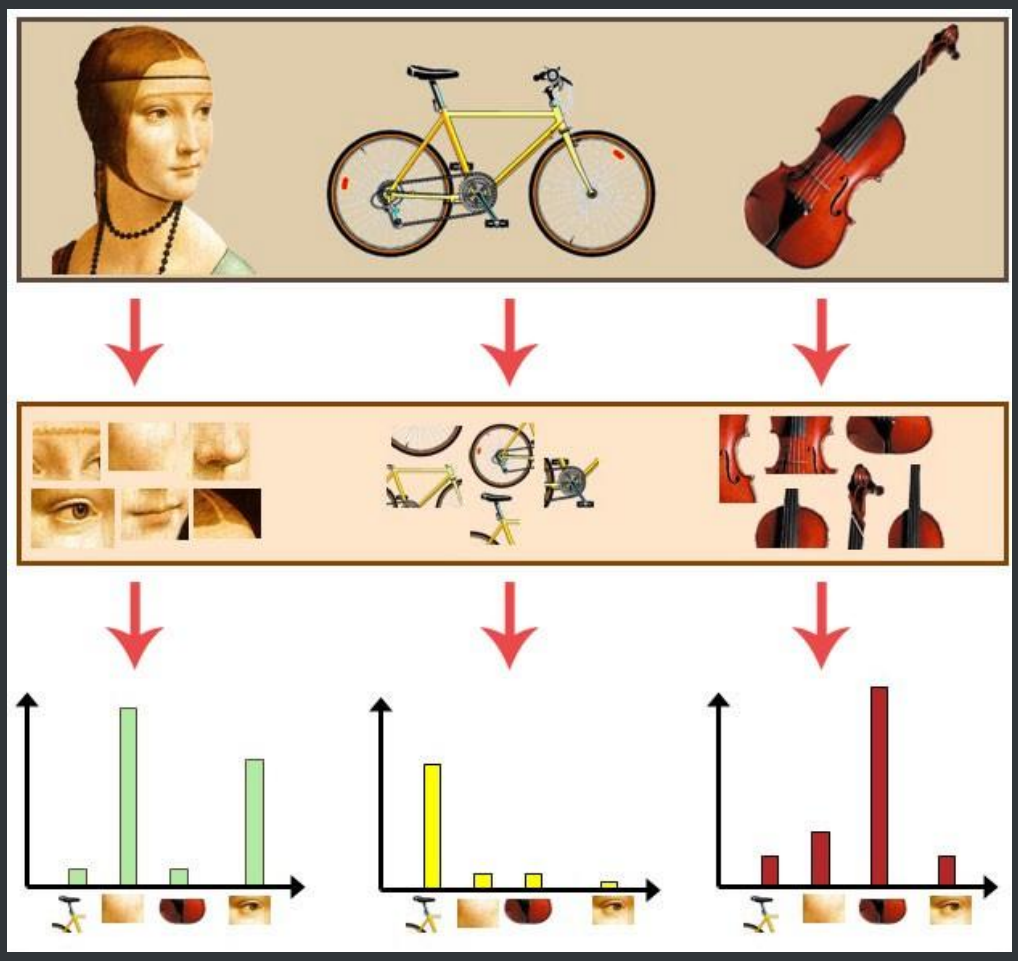


Keywords σε έγγραφο

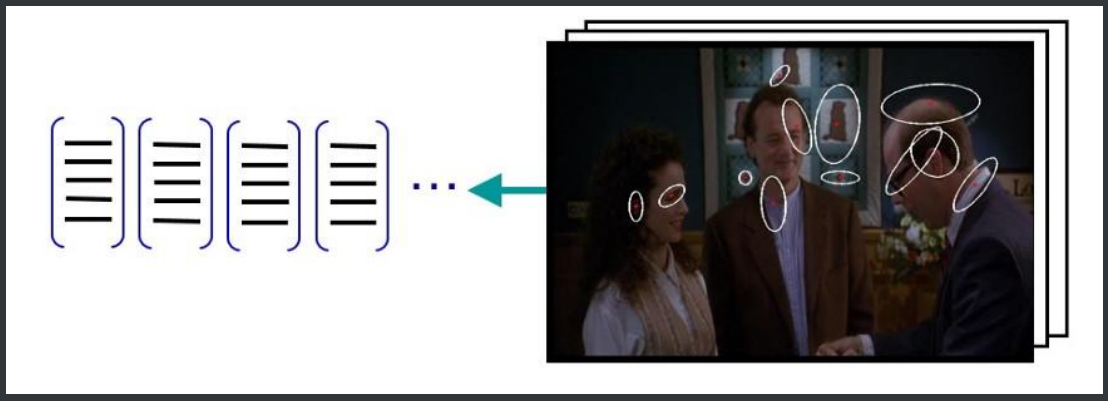
Ι. Θεωρητική ανάλυση της εργασίας.

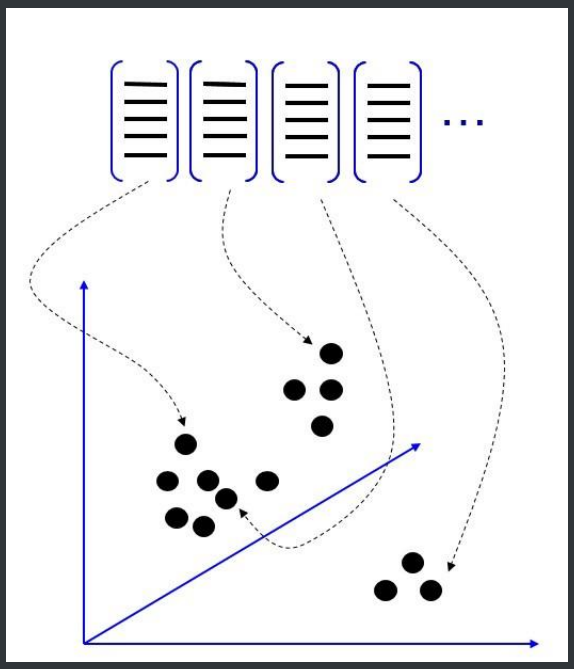
// Διαδικασία Δημιουργίας Λεξιλογίων:

Πριν αναλυθεί ο αλγόριθμος που θα χρησιμοποιηθεί για την διαδικασία δημιουργίας λεξιλογίων, θα περιγραφεί η ίδια η διαδικασία. Στο μοντέλο αυτό ταξινομούμε τα χαρακτηριστικά των εικόνων σαν λέξεις που περιγράφουν την εικόνα. Αρχικά υπολογίζονται ανιχνευτές και περιγραφείς της κάθε εικόνας. Χρησιμοποιούνται για να κατασκευαστούν λεξιλόγια και να εκπροσωπηθεί κάθε εικόνα ως ένα ιστόγραμμα συχνότητας των χαρακτηριστικών που υπάρχουν σε εκείνη. Από αυτό το ιστόγραμμα, αργότερα, μπορούμε να βρούμε άλλες παρόμοιες εικόνες ή να προβλέψουμε την κατηγορία της κάθε εικόνας. Συγκεκριμένα:



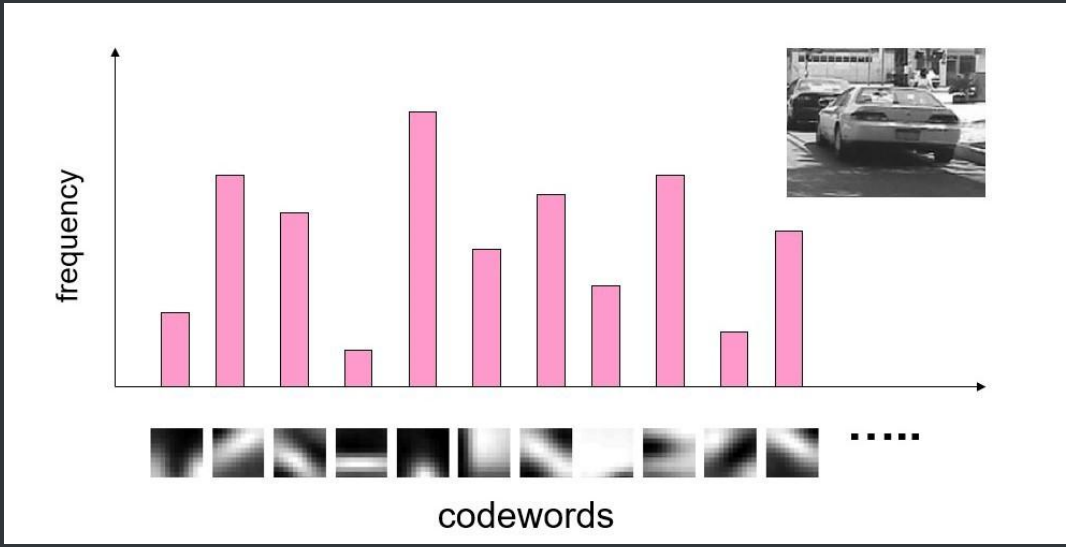
Ιστόγραμμα οπτικών "λέξεων"





Εξαγωγή χαρακτηριστικών και "Clustering" περιγραφέων

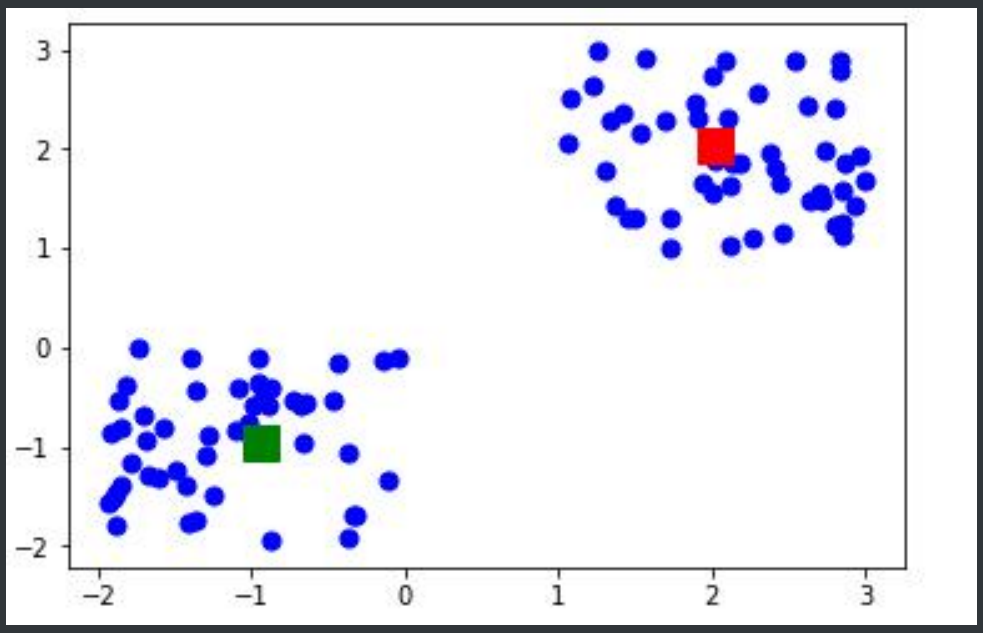
Όπως έχουμε δει και σε προηγούμενη εργασία, θα χρησιμοποιηθεί ένας αλγόριθμος για την εξαγωγή των χαρακτηριστικών με την μέθοδο που επιλέγουμε για παράδειγμα "SIFT". Στην συνέχεια φτιάχνουμε τους "clusters" ή αλλιώς ταξινομητές από τους περιγραφείς χρησιμοποιώντας την μέθοδο "K-means" που θα αναλύσουμε παρακάτω. Με την συνάρτηση που χρησιμοποιείται επιστρέφονται τα κέντρα των "clusters" που απαρτίζουν το λεξικό. Τέλος φτιάχνουμε τα ιστογράμματα συχνότητας από την συχνότητα εμφάνισης των λέξεων στις εικόνες. Αυτά τα ιστογράμματα αποτελούν το μοντέλο μας, "Bag Of Visual Words".



Ιστόγραμμα εικόνας

// Ανάλυση της μεθόδου "K-means":

Χρησιμοποιώντας το μοντέλο που αναφέραμε σε ένα συγκεκριμένο σύνολο εικόνων, μπορούμε να υπολογίσουμε τους πιο κοντινούς γείτονες μιας εικόνας. Χρησιμοποιείται ο αλγόριθμος "K-means" για την ομαδοποίηση χαρακτηριστικών ως εξής. Εφόσον υπολογιστούν τα χαρακτηριστικά και τα κέντρα των διαφορετικών ομάδων, τότε ο αλγόριθμος διαλέγει τυχαία έναν αριθμό κέντρων που ορίζεται από τον χρήστη. Έπειτα υπολογίζεται για όλα τα σημεία η απόστασή τους από όλα τα κέντρα. Στην συνέχεια αναθέτει το κάθε χαρακτηριστικό στην ομάδα με κέντρο που έχει την μικρότερη απόσταση και υπολογίζει το νέο κέντρο για κάθε ομάδα ως τον μέσο όρο των χαρακτηριστικών που έχει η ομάδα αυτή. Μετά υπολογίζεται η απόσταση μεταξύ κάθε χαρακτηριστικού και του νέου κέντρου. Αν κανένα σημείο δεν έχει κέντρο διαφορετικής ομάδας πιο κοντινό κατά τον υπολογισμό τότε τερματίζεται ο αλγόριθμος. Στην συγκεκριμένη εργασία έπειτα από την ομαδοποίηση όπως προαναφέρθηκε προχωράμε στην δημιουργία λεξικών με το μοντέλο που αναλύθηκε παραπάνω. Πλέον εφόσον έχουν δημιουργηθεί τα λεξικά και έχουν γίνει οι απαραίτητες κωδικοποιήσεις, το επόμενο βήμα είναι η ταξινόμηση με την χρήση αλγορίθμων ταξινομητών που θα αναλύσουμε παρακάτω.



Αναπαράσταση ομαδοποίησης με την χρήση του “K-means”

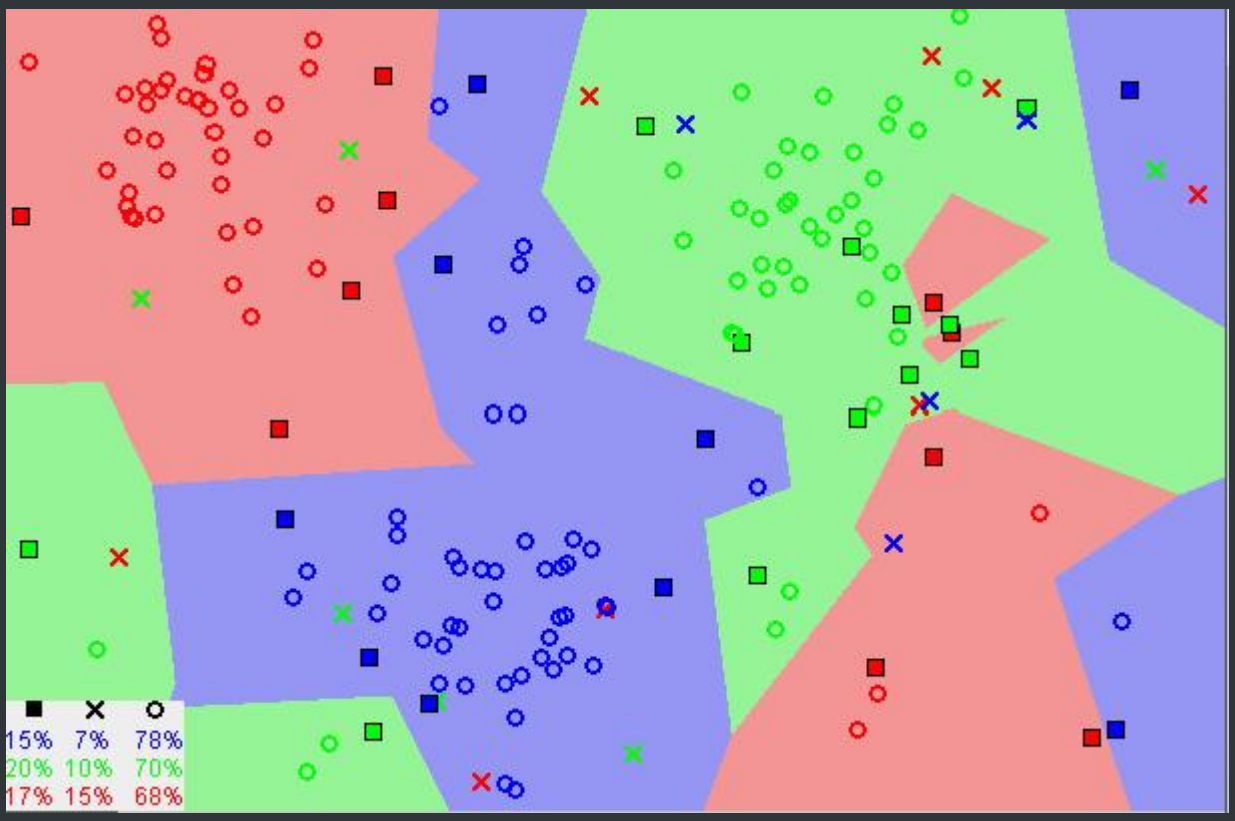
// Ταξινόμηση

Γενικότερα η διαδικασία της ταξινόμησης και η χρήση ταξινομητών χρησιμεύει στην ταξινόμηση άγνωστων ως προς την κλάση δεδομένων. Για αυτόν τον λόγο ακολουθούνται δύο τύπου στρατηγικές, “one versus all” ή “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. Στην δεύτερη περίπτωση ο κάθε ταξινομητής χρησιμοποιεί δείγματα δεδομένων από δύο κλάσεις και προβλέπει σε ποια από τις δύο κλάσεις ανήκει το άγνωστο δείγμα όπου πρέπει να ταξινομηθεί.

// Χρήση ταξινομητή “K-nearest neighbors”

Ο συγκεκριμένος ταξινομητής υποθέτει πως έχουν ήδη ομαδοποιηθεί τα δεδομένα ανάλογα με την απόσταση των κεντρώων τους. Αυτό μπορεί να γίνει με τον αλγόριθμο υπολογισμού αποστάσεων που αναφέρθηκε παραπάνω. Ο συγκεκριμένος ταξινομητής χρησιμοποιεί τα κοντινότερα σημεία του δειγματοχώρου για την εξαγωγή της

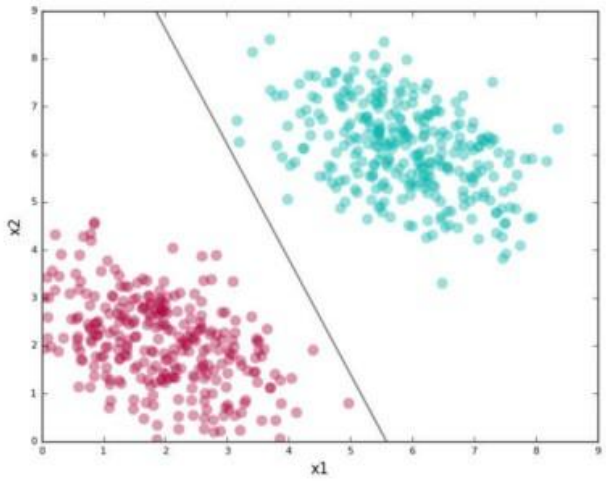
κλάσης στην οποία προβλέπεται να ανήκει το δείγμα προς ταξινόμηση. Η κλάση που ανατίθεται στο σημείο, εξάγεται από την πλειοψηφία των γειτόνων. Οι συνηθέστερες μετρικές που χρησιμοποιούνται είναι η Ευκλείδεια απόσταση και η απόσταση Hamming.

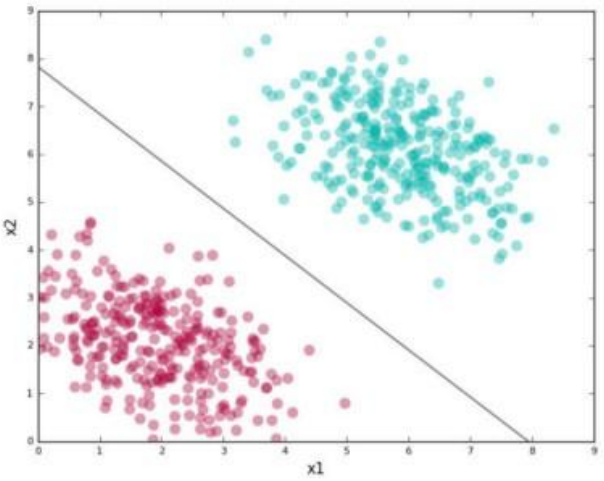


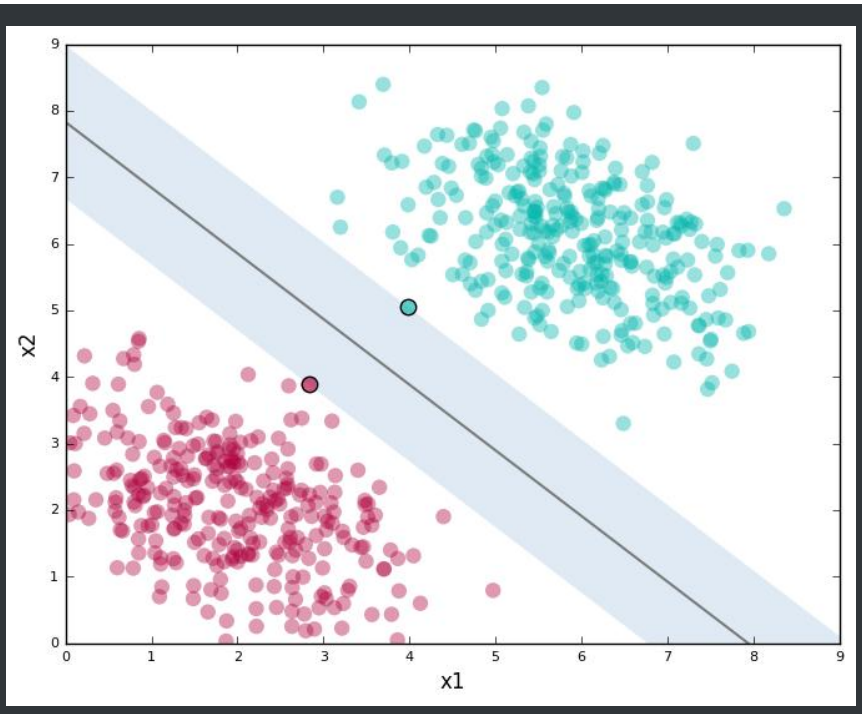
Αναπαράσταση δεδομένων και της μικρής απόστασής τους το ένα από το άλλο.

// Χρήση ταξινομητή “Support Vector Machines”

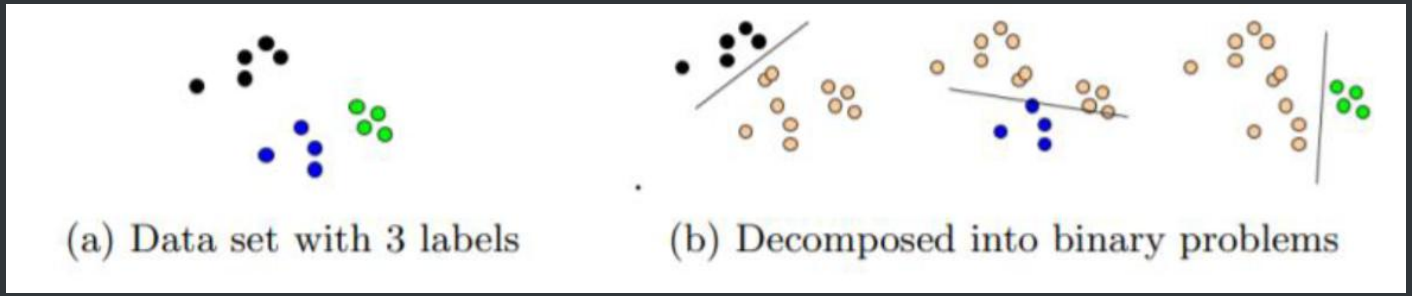
Στον συγκεκριμένο ταξινομητή υπολογίζεται ένα υπερεπίπεδο που διαχωρίζει βέλτιστα δύο κλάσεις. Το διαχωριστικό υπερεπίπεδο θα πρέπει να μεγιστοποιεί την απόσταση μεταξύ των κλάσεων. Σε αντίθεση με τον προηγούμενο ταξινομητή, ο συγκεκριμένος υποστηρίζει προβλήματα των δύο κλάσεων. Σε περίπτωση που έχουμε να λύσουμε προβλήματα που απαιτούν περισσότερες κλάσεις, χρησιμοποιούμε περισσότερα υπερεπίπεδα και έναν ταξινομητή για κάθε κλάση. Έτσι ο διαχωρισμός γίνεται για κάθε ταξινομητή – κλάση σε δεδομένα που ανήκουν στην κλάση αυτή και σε όλα τα υπόλοιπα.







Ταξινομητής δύο κλάσεων.



Μετατροπή προβλήματος πολλαπλών κλάσεων σε δυαδικό "one vs all".

ΙΙ. Περιγραφή Κώδικα.

// Περιγραφή Κώδικα: Αρχικά γίνεται διαχωρισμός των βάσεων εικόνων που δίνονται. Η πρώτη "imagedb" θα χρησιμοποιηθεί για την εκπαίδευση του συστήματος και η δεύτερη "imagedb_test" για την δοκιμή και την αξιολόγησή του. Ξεκινάμε με την δημιουργία directories των δοθείσων εικόνων. Δημιουργία αντικείμενου υπεύθυνο για την εξαγωγή χαρακτηριστικών με "SIFT".

```
#_____________________________________________________________________

                Step one: Call of functions that creating the vocabularies.
#_____________________________________________________________________

print("Creating vocabularies...")
# Create SIFT object
sift_voc = cv.xfeatures2d_SIFT.create()
# Train directory
train_directory_voc = "imagedb/"
# Train folders
train_folders_voc = [dI1 for dI1 in os.listdir(train_directory_voc) if
                     os.path.isdir(os.path.join(train_directory_voc, dI1))]
```

// Δημιουργία ιστογραμμάτων σύμφωνα με τα λεξικά που δημιουργούνται στο υπάρχον directory, διαφορετικού μεγέθους και αριθμού λέξεων. Έπειτα οι κωδικοποιημένες εικόνες – λέξεις χρησιμοποιούνται για την δημιουργία περιγραφέων χαρακτηριζόμενοι ως εκπαιδευμένοι και αποθηκεύονται.

```
#_____________________________________________________________________

    Step two: Call of functions that create histograms according to vocabularies found before.
#_____________________________________________________________________

# Create vocabularies with different number of words
for i in range(50, 201, 100):
    # Call create_vocabulary
    create_vocabulary(i, train_directory_voc, train_folders_voc, sift_voc)

# Vocabulary path
vocabulary_path = "vocabularies/"
# Use all of the vocabularies to create train features
for vocabulary in os.listdir(vocabulary_path):
    # Call train_features
    train_features(train_directory_voc, train_folders_voc, vocabulary_path + vocabulary, sift_voc)

#
_____________________________________________________________________
```

II. Περιγραφή Κώδικα.

// Η ανάλυση των συναρτήσεων που καλούνται κατά την εκτέλεση των βημάτων θα γίνει παρακάτω όπως και η ανάλυση των δύο μοντέλων. Προς το παρόν θεωρούμε πως δημιουργούνται λεξικά διαφορετικού αριθμού λέξεων σε ένα συγκεκριμένο φάκελο που έχουμε ορίσει εμείς με την μέθοδο "SIFT". Έπειτα καλείται συνάρτηση που είναι υπεύθυνη για δημιουργία κωδικοποιημένων περιγραφέων για εκπαίδευση.

```
#_____________________________________________________________________________

    Step two: Call of functions that create histograms according to vocabularies found before.
#_____________________________________________________________________________

# Create vocabularies with different number of words
for i in range(50, 201, 100):
    # Call create_vocabulary
    create_vocabulary(i, train_directory_voc, train_folders_voc, sift_voc)

# Vocabulary path
vocabulary_path = "vocabularies/"
# Use all of the vocabularies to create train features
for vocabulary in os.listdir(vocabulary_path):
    # Call train_features
    train_features(train_directory_voc, train_folders_voc, vocabulary_path + vocabulary, sift_voc)

#
_____________________________________________________________________________
```

// Ανάλυση της συνάρτησης δημιουργίας λεξικού: Αρχικά ελέγχουμε αν υπάρχει το directory που θα αποθηκεύσουμε τα δημιουργημένα λεξικά, αν όχι το δημιουργούμε. Στην συνέχεια καλείται η συνάρτηση που θα εξάγει τα χαρακτηριστικά από κάθε εικόνα σε κάθε φάκελο καλώντας μια συνάρτηση εξαγωγής χαρακτηριστικών και δημιουργίας περιγραφών με την μέθοδο "SIFT". Επομένως σε κάθε φάκελο κάθε εικόνας συλλέγονται οι υπάρχοντες περιγραφείς και αυξάνεται ο μετρητής μετά το πέρας της επανάληψης. Έτσι η συνάρτηση επιστρέφει μια λίστα με τους περιγραφείς που προορίζονται για την εκπαίδευση του συστήματος και τον αριθμό των εικόνων από τις οποίες έγινε η εξαγωγή των χαρακτηριστικών και περιγραφέων.

```
def extract_local_features(image_path, sift):
    # print("Extracting local features...")
    image = cv.imread(image_path)
    keypoints = sift.detect(image)
    descriptors = sift.compute(image, keypoints)
    descriptors = descriptors[1]
    return descriptors
```

II. Περιγραφή Κώδικα.

```
def create_feature_database(train_directory, train_folders, sift):
    print("Extracting features...")
    # The number of images used for the training
    number_of_images = 0
    # Initialize an array 128-dimensional, because we use SIFT Descriptor
    train_descriptors = np.zeros((0, 128))
    for folder in train_folders:
        # Create the path of every folder
        folder_path = os.path.join(train_directory, folder)
        # Create a list containing the names of the entries in the directory given by path
        files = os.listdir(folder_path)
        for file in files:
            # Create the path of every image
            path = os.path.join(folder_path, file)
            # Call extract_local_features and take Descriptors
            descriptors = extract_local_features(path, sift)
            # Prevent some bugs, because sometimes it takes some noise like "..."
            if descriptors is None:
                continue
            # Do concatenate the train_descriptors (all ready exists) with new descriptors
            train_descriptors = np.concatenate((train_descriptors, descriptors), axis=0)
            number_of_images += 1
    return train_descriptors, number_of_images
```

// Ανάλυση της συνάρτησης συλλογής των χαρακτηριστικών για εκπαίδευση, με την χρήση των υπαρχόντων χαρακτηριστικών: Από το "path" που ορίστηκε σαν παράμετρος της συνάρτησης, μπορώ να πάρω τον αριθμό των κωδικοποιημένων εικόνων και τα αρχεία των κωδικοποιημένων περιγραφέων. Επιπλέον με τον ίδιο τρόπο αν δεν υπάρχει κάποιο "directory" το δημιουργούμε. Αρχικοποιούνται οι μεταβλητές των λεξικών, των αριθμών των εικόνων και των περιγραφέων από το μοντέλο "BOVW". Με ανάλογη προσπέλαση στους φακέλους για εκπαίδευση, γίνεται εξαγωγή χαρακτηριστικών με την μέθοδο "SIFT" και εισάγεται στην λίστα που δημιουργήθηκε για τους περιγραφείς η επιστροφή της συνάρτησης. Έπειτα καλείται μια συνάρτηση κωδικοποίησης των περιγραφέων που εξάχθηκαν μέσα στην επανάληψη που αναφέραμε, ώστε μετά το πέρας της, να έχει αυξηθεί ο μετρητής των εικόνων και να έχουν συλλεχθεί οι κωδικοποιημένοι περιγραφείς.

// Ανάλυση της συνάρτησης κωδικοποίησης και δημιουργίας ιστογραμμάτων: Παρακάτω θα παρουσιαστεί πρώτα η συνάρτηση κωδικοποίησης και μετά η συνάρτηση συλλογής των χαρακτηριστικών για εκπαίδευση. Κατά την κωδικοποίηση αρχικοποιείται ένας πίνακας περιγραφέων και από το όρισμα της συνάρτησης, γίνεται προσπέλαση στους περιγραφείς υπολογίζοντας την απόσταση μεταξύ των περιγραφέων και των στοιχείων του διανύσματος των λεξικών. Μετά υπολογίζεται

η ευκλείδεια απόσταση και το σύνολο των γραμμών. Στην συνέχεια παίρνουμε την ελάχιστη απόσταση των τοπικών χαρακτηριστικών από τις "λέξεις" και αυξάνουμε κατά μια την μεταβλητή που δείχνει την συχνότητα εμφάνισης του χαρακτηριστικού που υπάρχει σε κάθε εικόνα. Τέλος κωδικοποιείται κάθε εικόνα με την χρήση των λεξικών και τα ιστογράμματα αποθηκεύονται στον αρχικοποιημένο πίνακα.

```
def encode_bovw_descriptor(descriptors, voc):
    bovw_descriptor = np.zeros((1, voc.shape[0]))
    for descriptor in range(descriptors.shape[0]):
        distances = np.sum((descriptors[descriptor, :] - voc) ** 2, axis=1)
        index_min = np.argmin(distances)
        bovw_descriptor[0, index_min] += 1
    bovw_descriptor = bovw_descriptor / np.sum(bovw_descriptor)
    return bovw_descriptor
```

```
def train_features(train_directory, train_folders, vocabulary_path_tf, sift):
    # Extract the number of visual words from files of vocabulary_path
    number_of_visual_words = [s for s in vocabulary_path if s.isdigit()]
    number_of_visual_words = ''.join(number_of_visual_words)
    # Filename of BoVW encoded descriptors
    filename = "bovw_encoded_descriptors_" + str(number_of_visual_words)
    # Check if a directory exists
    if not os.path.isdir("./bovw_descs"):
        # Make directory
        os.makedirs("bovw_descs")

        vocabulary_tf = np.load(vocabulary_path_tf)
        number_of_images = 0
        bovw_descriptors = np.zeros((0, vocabulary_tf.shape[0] + 1))
        for folder, class_i in zip(train_folders, range(len(train_folders))):
            # Join the train_directory and folder (names)
            folder_path = os.path.join(train_directory, folder)
            # A list containing the names of the entries in the directory given by path
            files = os.listdir(folder_path)
            for file in files:
                # Join the folder_path and the file (names)
                path = os.path.join(folder_path, file)
                # Call extract_local_features
                descriptor = extract_local_features(path, sift)
                # Call encode_bovw_descriptor
                bovw_descriptor = encode_bovw_descriptor(descriptor, vocabulary_tf)
                # DUMB THING
                bovw_descriptor = np.append(bovw_descriptor, [class_i])
                # Increase the dimension of the existing array by one more dimension
                bovw_descriptor = bovw_descriptor[:, np.newaxis]
                # Reshape the array so as to it can be concatenate
                bovw_descriptor = bovw_descriptor.reshape((bovw_descriptor.shape[1],
bovw_descriptor.shape[0]))
                # Do concatenate
                bovw_descriptors = np.concatenate((bovw_descriptors, bovw_descriptor), axis=0)
                number_of_images += 1
```

II. Περιγραφή Κώδικα.

```
    # Save the BoVW encoded descriptors ("Trained")
        np.save("bovw_descs/" + filename, bovw_descriptors)
        print("\nNumber of images : ", number_of_images)
        print("Vocabulary : ", vocabulary_path_tf)
        print("Filename : ", filename, "\n")
```

Σε αυτό το σημείο να αναφερθεί πως η διαδικασία που αναλύθηκε στο στάδιο που βρίσκεται ο κώδικας φαίνεται να ολοκληρώνεται μόνο για τις εικόνες που προορίζονται για εκπαίδευση. Κάτι το οποίο δεν είναι αρκετό διότι θεωρητικά θα πρέπει να ακολουθηθεί η ίδια διαδικασία και για τις εικόνες που προορίζονται για την αξιολόγηση του συστήματος. Παρακάτω που θα αναλυθεί ο κώδικας των ταξινομητών, θα φανεί πως μέσα σε αυτά τα αρχεία καλούνται οι ίδιες συναρτήσεις που αναλύθηκαν μέχρι τώρα αλλά για τις εικόνες που προορίζονται για αξιολόγηση αυτήν την φορά.

// Περιγραφή του K-Nearest Neighbors ταξινομητή: Όπως αναφέρθηκε το πρόγραμμα χωρίζεται σε δύο ενότητες. Στην εκπαίδευση και την αξιολόγηση. Με την κατάλληλη εντολή αρχίζει ο αλγόριθμος καλώντας διάφορες συναρτήσεις που θα αναφέρουμε παρακάτω.

```
def run_knn(option):
    if option == "run":
        # Run K-Nearest Neighbor for each class, using all of the train features
        # Create SIFT object
        sift = cv2.xfeatures2d_SIFT.create()
        # Train directory
        test_directory = "imagedb_test"
        # Training dataset
        test_folders = [dI for dI in os.listdir(test_directory) if
os.path.isdir(os.path.join(test_directory, dI))]
        # Train set path
        train_set_path = "bovw_descs/"
        knn_data_frame = pd.DataFrame(
            columns=['image_path', 'class', 'predicted_class', 'knn_neighbors', 'vocabulary_words'])
        # Set the k
        number_of_neighbors = [2, 5, 9, 17, 25, 37, 40, 55]
        for train in os.listdir(train_set_path):
            for k in number_of_neighbors:
                knn_data_frame = test_knn(train_set_path + train, test_directory, test_folders, k,
sift, knn_data_frame)

        knn_data_frame.to_csv('knn.csv')
    else:
        exit(0)

run_knn("run")
```

II. Περιγραφή Κώδικα.

Δεδομένου ότι κάποιες συναρτήσεις έχουν αναλυθεί θα παρακαμφθούν σε αυτό το σημείο. Αρχικά φορτώνονται οι συναρτήσεις από το πρόγραμμα που έχουν αναλυθεί μέχρι στιγμής. Μέσα στην τρεχούμενη συνάρτηση δημιουργείται το "directory" για τις εικόνες που προορίζονται για την αξιολόγηση και την εκπαίδευση αντίστοιχα. Ακόμα δημιουργείται το "data-frame" που περιέχει στήλες με τους αναγραφόμενους τίτλους και η λίστα με "K" αριθμούς γειτόνων ως κάθε στοιχείο της λίστας. Έτσι εκτελείται επανάληψη που κάνει προσπέλαση σε κάθε φάκελο του "directory" που ορίστηκε για τους περιγραφείς των εκπαιδευμένων εικόνων, και ανάλογα με τον αριθμό "K" που έχουμε ορίσει, καλείται η συνάρτηση αξιολόγησης που εκτελεί λειτουργίες που θα αναφέρουμε παρακάτω.

// Ανάλυση συναρτήσεων υπολογισμών απόστασης: Όπως προαναφέρθηκε υπάρχουν δύο τρόποι για τον υπολογισμό της ευκλείδειας απόστασης.

```
# Calculate the Euclidean Distance between two vectors
def euclidean_distance0(row1, row2):
    # Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
    distance = np.sqrt(np.sum((row1 - row2) ** 2))
    return distance


# Second way with loop
def euclidean_distance(row1, row2):
    # Initialize distance
    distance = []
    # Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
    for x in range(len(row1)):
        distance = np.sqrt(np.sum((row1 - row2) ** 2))
    return distance
```

// Ανάλυση συνάρτησης εντοπισμού γειτόνων: Αρχικοποιείται μια άδεια λίστα που θα περιέχει ευκλείδειες αποστάσεις εκπαιδευμένων και μη εικόνων, καλώντας μια από τις παραπάνω συναρτήσεις, όσο γίνεται η προσπέλαση στις εικόνες για εκπαίδευση. Έπειτα ταξινομούνται οι αποστάσεις κατά φθίνουσα σειρά με την αντίστοιχη εικόνα προς εκπαίδευση και ανάλογα με το όρισμα που καθορίζει το πλήθος των γειτόνων, συλλέγεται η κάθε εικόνα προς εκπαίδευση με την μικρότερη απόσταση, ως γείτονας. Η επιστροφή της συνάρτησης είναι η λίστα των γειτόνων.

II. Περιγραφή Κώδικα.

```
def get_k_neighbors(train_dataset, test_row, number_of_neighbors):
    # Initialize a list named distances
    distances = []
    for train_row in train_dataset:
        # Call euclidean_distance
        distance = euclidean_distance(test_row, train_row[:-1])
        # Appends an element to the end of the list (list => distances)
        distances.append((train_row, distance))
    # Sort the list of tuples by the distance (in descending order)
    distances.sort(key=lambda tup: tup[1])
    # Initialize a list named neighbors
    neighbors = []
    for i in range(number_of_neighbors):
        # Appends an element to the end of the list (list => neighbors)
        neighbors.append(distances[i][0])
    return neighbors
```

// Ανάλυση συνάρτησης δημιουργίας προβλέψεων και ταξινόμησης με τον ταξινομητή "K-neighbors": Αρχικά καλείται η συνάρτηση εντοπισμού γειτόνων και αυτό που επιστρέφεται αποθηκεύεται σε μια λίστα. Έτσι αποθηκεύεται στην επιστρεφόμενη μεταβλητή το στοιχείο που εμφανίζεται πιο συχνά.

```
# Make a classification prediction with k-neighbors
def prediction_and_classification(train_dataset, test_row, number_of_neighbors):
    # Call get_k_neighbors
    neighbors = get_k_neighbors(train_dataset, test_row, number_of_neighbors)

    neighbor_classes = [row[-1] for row in neighbors]

    prediction = max(set(neighbor_classes), key=neighbor_classes.count)
    # prediction = The predicted class identifier
    return prediction
```

// Με διαφορετικό τρόπο θα μπορούσαμε να δημιουργήσουμε την λίστα με τις προβλέψεις όπως στην παρακάτω συνάρτηση. Όμως μετέπειτα στην συνάρτηση που γίνεται η αξιολόγηση η μεταβλητής, ως λίστα, δημιουργεί πρόβλημα.

```
# Don't use it on test_knn because can not compare with class, because predictions is list
# KNN Algorithm
def knn_classification(train_dataset, test_row, number_of_neighbors):
    # Initialize a list named predictions
    predictions = []
    for row in test_row:
        # Call prediction_and_classification
        prediction = prediction_and_classification(train_dataset, row, number_of_neighbors)
        # Appends an element to the end of the list (list => predictions)
        predictions.append(prediction)
    return predictions
```

II. Περιγραφή Κώδικα.

// Ανάλυση της συνάρτησης αξιολόγησης: Αρχικά για να γίνει η αξιολόγηση πρέπει να επιβεβαιωθεί η ύπαρξη του "dataset". Αν υπάρχει, θα βρεθεί από την ονομασία ο αριθμός των λέξεων - εικόνων. Δημιουργούμε "directory" που περιέχει πίνακα με τα λεξικά και αρχικοποιούνται οι μετρητές των εικόνων που ελέγχθηκαν και οι σωστά αντιστοιχισμένες. Έπειτα γίνεται προσπέλαση στους φακέλους που υπάρχουν για αξιολόγηση και δεύτερη προσπέλαση στις εικόνες κάθε φακέλου. Σε κάθε επανάληψη αυξάνεται ο μετρητής των εικόνων, δημιουργούνται οι περιγραφείς γίνεται εξαγωγή χαρακτηριστικών, γίνεται η κωδικοποίηση και καλείται η συνάρτηση πρόβλεψης και ταξινόμησης. Κατά συνέπεια μετά το κάλεσμα αυτής της συνάρτησης γίνεται έλεγχος της πρόβλεψης που επιστρέφεται με τον αριθμό που αντιστοιχεί στον εκάστοτε φάκελο. Τέλος δημιουργούμε ένα "dataframe" με τις ακόλουθες πληροφορίες και μετράμε το ποσοστό των ορθών ταξινομήσεων.

```
# Test KNN
def test_knn(train_set_path, test_directory, test_folders, number_of_neighbors, sift, result_df):
    #  Split the path name into a pair head and tail.
    head, tail = os.path.split(train_set_path)
    if tail not in os.listdir(head):
        print("This dataset does not exist.")

    # Load the encoded training set
    train_set = np.load(train_set_path)
    number_of_visual_words = int(''.join([s for s in train_set_path if s.isdigit()]))

    # Set the path for vocabularies
    vocabulary_path = "vocabularies/vocabulary_" + str(number_of_visual_words) + ".npy"

    # Load vocabularies
    vocabulary = np.load(vocabulary_path)
    # Initialize
    number_of_images = 0
    number_of_correct = 0
    for folder, class_i in zip(test_folders, range(len(test_folders))):
        folder_path = os.path.join(test_directory, folder)
        # Create a list containing the names of the entries in the directory given by path
        files = os.listdir(folder_path)
        for file in files:
            number_of_images += 1
            # Create the path of every image
            path = os.path.join(folder_path, file)
            # Call extract_local_features
            desc = extract_local_features(path, sift)
            # Call encode_bovw_descriptor
            bovw_desc = encode_bovw_descriptor(desc, vocabulary)
            # Call prediction and classification
            prediction = prediction_and_classification(train_set, bovw_desc, number_of_neighbors)
            if prediction == class_i:
                number_of_correct += 1
```

II. Περιγραφή Κώδικα.

```
        # DataFrame of the results for classification
        result_df = result_df.append(
            pd.Series([path, class_i, prediction, number_of_neighbors, number_of_visual_words],
                      index=result_df.columns), ignore_index=True)
    # Calculate accuracy
    # The percentage of successful classifications
    accuracy = round(number_of_correct * 100 / number_of_images, 4)

    # Prints
    print("K-Nearest-Neighbors prediction completed.\n")
    print("Number of visual words :", number_of_visual_words)
    print("Number of neighbors (K) :", number_of_neighbors)
    print("Number of trained pictures : ", train_set.shape[0])
    print("Number of tested pictures : ", number_of_images)
    print("Number of pictures correctly classified: ", number_of_correct)
    print("The success rate is: ", accuracy, "%")
    print("\n\n")

    return result_df
```

// Περιγραφή του SVM ταξινομητή: Στο συγκεκριμένο πρόγραμμα η βασική συνάρτηση που είναι υπεύθυνη για την λειτουργία του προγράμματος, είναι αυτή που καθορίζει την εκπαίδευση ή την αξιολόγηση ανάλογα με την είσοδο. Κατά την εκπαίδευση ακολουθείται η ίδια διαδικασία όσον αφορά τα "directories" και την δημιουργία των "dataset" για εκπαίδευση. Έπειτα ορίζεται ο τύπος των kernel και το επιθυμητό σχετικό σφάλμα έψιλον. Στην συνέχεια γίνεται η προσπέλαση στους φακέλους και τις εικόνες για εκπαίδευση. Δημιουργείται "directory" για τα αρχεία SVM ανάλογα με τους φακέλους για εκπαίδευση, τον αριθμό των λέξεων,τον τύπο kernel και το σχετικό σφάλμα. Τέλος καλείται η συνάρτηση του SVM ταξινομητή που θα αναλυθεί παρακάτω.

II. Περιγραφή Κώδικα.

```
def training_testing(option):
    if option == "Training":
        # Train an SVM for each class, using all of the train features

        # Create SIFT object
        sift = cv2.xfeatures2d_SIFT.create()
        # Train directory
        train_directory = "imagedb"
        # Train dataset
        training_folders = [dI for dI in os.listdir(train_directory) if
                            os.path.isdir(os.path.join(train_directory, dI))]
        # Train features set path
        train_set_path = "train_dbs/"
        # The types of kernel
        # type_of_kernels = ["RBF", "CHI2", "INTER", "SIGMOID"]
        # type_of_kernels = ["RBF", "CHI2"]
        type_of_kernels = ["INTER", "SIGMOID", "LINEAR"]

        # The desired accuracy (epsilon)
        # epsilons = [1.e-08, 1.e-06, 1.e-04]
        # epsilons = [1.e-06, 1.e-04]
        epsilons = [1.e-06]

        for folder, class_i in zip(training_folders, range(len(training_folders))):
            for train_features in os.listdir(train_set_path):
                # The number of visual words that have been used
                number_of_words = int(''.join([s for s in train_features if s.isdigit()]))
                # Create the path of every file (train_features)
                path = os.path.join(train_set_path, train_features)
                training_set_features_path = np.load(path)
                for kernel in type_of_kernels:
                    for epsilon in epsilons:
                        # Set the path and name of the SVM file
                        svms_path = "SVMs/"
                        svm_name = "svm_" + str(class_i) + '_' + str(
                            number_of_words) + "words_" + '_' + kernel + '_' + str(epsilon)
                        svm_path_name = svms_path + svm_name
                        # Call train_svm_classifier
                        train_svm_classifier(training_set_features_path, class_i, kernel, epsilon,
svm_path_name)
                        print("SVM with name :", svm_name, "trained.")
```

Όσον αφορά την δεύτερη επιλογή, την αξιολόγηση, γίνεται όμοια η διαδικασία δημιουργίας φακέλων και των στηλών του “dataframe” των αποτελεσμάτων. Όμοια ορίζω μεταβλητές αριθμού λέξεων, kernel, σχετικού σφάλματος, καλείται η συνάρτηση αξιολόγησης με τον ταξινομητή SVM για κάθε λέξη – εικόνα και κάθε τύπου kernel, που θα αναλύσουμε παρακάτω και το “dataframe” γεμάτο πλέον μετατρέπεται σε “csv”.

II. Περιγραφή Κώδικα.

```
    elif option == "Testing":

        # Create SIFT object
        sift = cv2.xfeatures2d_SIFT.create()
        # Test directory
        test_directory = "imagedb_test"
        # Test dataset
        test_folders = [dI for dI in os.listdir(test_directory) if
os.path.isdir(os.path.join(test_directory, dI))]

        # DataFrame for SVMs
        svms_df = pd.DataFrame(columns=['image_path', 'class', 'predicted_class', 'vocabulary_words',
'kernel', 'epsilon'])
        # SVMs path
        svms_path = "SVMs/"

        # The number of visual word TODO: SET THE NUMBER OF WORDS
        # number_of_words = [50, 100, 150, 200, 300, 400, 500, 600, 700]
        number_of_words = [50, 100, 300, 500]

        # The types of kernel
        # type_of_kernels = ["RBF", "POLY", "CHI2", "INTER", "SIGMOID"]
        # type_of_kernels = ["RBF",  "CHI2"]
        type_of_kernels = ["INTER", "SIGMOID", "LINEAR"]
        # The desired accuracy (epsilon)
        # epsilons = [1.e-08, 1.e-06, 1.e-04]
        # epsilons = [1.e-06, 1.e-04]
        epsilons = [1.e-06]

        for word in number_of_words:
            for kernel in type_of_kernels:
                for epsilon in epsilons:
                    svms_df = test_svms(svms_path, test_folders, test_directory, (word, kernel,
epsilon), sift, svms_df)

        # Save results to CSV
        svms_df.to_csv("svm.csv")


training_testing("Training")
training_testing("Testing")
```

// Περιγραφή της συνάρτησης εκπαίδευσης SVM: Αρχικά δημιουργείται ένα SVM μοντέλο και ορίζεται ο τύπος του ταξινομητή που θα καθορίσει τον τρόπο κατάτμησης και ορίζεται και ο τύπος του kernel που θα χρησιμοποιηθεί. Έπειτα καθορίζεται το κριτήριο τερματισμού και δημιουργείται ένας πίνακας με τις ετικέτες ανάλογα με τον αριθμό των εικόνων. Έτσι προχωράμε στην εκπαίδευση του μοντέλου και στην αποθήκευσή του.

ΙΙ. Περιγραφή Κώδικα.

```
# Train an SVM to classify an image based on the type of kernel and epsilon
def train_svm_classifier(training_set, train_class, type_of_kernel, epsilon, svm_path_name):
    """
    training_set: The training set (2D array)
    train_class: The class identifier
    type_of_kernel: The type of kernel that will be used for the SVM
    epsilon: The desired accuracy
    svm_path_name: The path and name of the SVM file that it will be saved

    """

    # Create an empty SVM model
    svm = cv2.ml.SVM_create()
    # Set the type of a SVM formulation
    # The SVM type C_SVC stands for C-Support Vector Classification that means
    # n-class classification (n ≥ 2), allows imperfect separation of classes with penalty multiplier
C for outliers
    svm.setType(cv2.ml.SVM_C_SVC)

    # If-statement for setting SVM kernel type
    if type_of_kernel == "RBF":
        # Radial basis function (RBF) kernel
        svm.setKernel(cv2.ml.SVM_RBF)
    elif type_of_kernel == "POLY":
        # Polynomial kernel:
        svm.setKernel(cv2.ml.SVM_POLY)
    elif type_of_kernel == "CHI2":
        # Exponential Chi2 kernel, it is similar to the RBF kernel
        svm.setKernel(cv2.ml.SVM_CHI2)
    elif type_of_kernel == "INTER":
        # Histogram intersection kernel. It is a fast kernel.   [ K(xi,xj)=min(xi,xj) ]
        svm.setKernel(cv2.ml.SVM_INTER)
    elif type_of_kernel == "SIGMOID":
        # Sigmoid kernel
        svm.setKernel(cv2.ml.SVM_SIGMOID)
    elif type_of_kernel == "LINEAR":
        # Linear kernel
        svm.setKernel(cv2.ml.SVM_LINEAR)

    # Termination Criteria
    svm.setTermCriteria((cv2.TERM_CRITERIA_EPS, 100, epsilon))
    # Create an array with labels as much as the images are
    labels = np.array([int((train_class == i)) for i in training_set[:, -1]])
    # Train SVM
    svm.trainAuto(training_set[:, :-1].astype(np.float32), cv2.ml.ROW_SAMPLE, labels)

    # Check if a directory exists (SVMs)
    if not os.path.isdir("./SVMs"):
        # Make directory / mkdir-p
        os.makedirs("SVMs")

    # Save the trained SVM
    svm.save(svm_path_name)

    return 0
```

// Περιγραφή της συνάρτησης μοντέλου SVM: Όπως αναλύθηκε και παραπάνω η διαδικασία προσπέλασης των εικόνων προς αξιολόγηση αρχίζει από την δημιουργία μοντέλου SVM, την εξαγωγή χαρακτηριστικών και περιγραφέων, την κωδικοποίηση και την πρόβλεψη του μοντέλου.

II. Περιγραφή Κώδικα.

```
# one-vs-All classification
def svm_one_vs_all(svms_path, test_dataset, test_directory, train_parameters, sift):
    """
    svms_path: The path to the directory for SVMs
    test_dataset: A list of the names of classes of the test dataset (is a list of  folders names)
    test_directory: The path of the directory that contains the test dataset
    train_parameters: A tuple with the parameters that will be used to train the SVMs
    sift: SIFT object that will be used by extract_local_features

    """

    # Load vocabulary
    vocabulary = np.load('vocabularies/vocabulary_' + str(train_parameters[0]) + '.npy')
    # Initialize a list named predictions
    predictions = []
    for class_i in range(len(test_dataset)):
        # Define SVM's filename
        svm_filename = 'svm_' + str(class_i) + '_' + str(train_parameters[0]) + 'words_' + '_' \
                     + train_parameters[1] + '_' + str(train_parameters[2])
        if svm_filename in os.listdir(svms_path):
            # Create an empty SVM model
            svm = cv2.ml.SVM_create()
            # Load SVMs
            svm = svm.load(svms_path + svm_filename)
            # Call extract_local_features
            descriptors = extract_local_features(test_directory, sift)
            # Call encode_bovw_descriptor
            bovw_descriptor = encode_bovw_descriptor(descriptors, vocabulary)
            # Prediction
            # cv.ml.STAT_MODEL_RAW_OUTPUT is used for more than one SVMs
            prediction = svm.predict(bovw_descriptor.astype(np.float32),
flags=cv2.ml.STAT_MODEL_RAW_OUTPUT)
            # Appends an element to the end of the list (list => predictions)
            predictions.append(prediction[1])
        else:
            print("There isn't any file in SVMs with these parameters")

    predicted_class = predictions.index(min(predictions))

    return predicted_class
```

// Περιγραφή της συνάρτησης αξιολόγησης SVM: Αρχικοποιούνται μεταβλητές μέτρησης εικόνων και σωστών αντιστοιχίσεων των ταξινομημένων εικόνων. Έπειτα όπως έχουμε ξαναδεί γίνεται προσπέλαση των εικόνων ως προς αξιολόγηση και ως πρόβλεψη αποθηκεύεται η επιστροφή της συνάρτησης μοντέλου που είδαμε παραπάνω. Όμοια γίνεται έλεγχος για ορθό αποτέλεσμα όπως νωρίτερα, αποθηκεύονται τα αποτελέσματα σε ένα "dataframe", υπολογίζεται το ποσοστό των σωστών αξιολογήσεων και επιστρέφεται το "dataframe" των αποτελεσμάτων.

```python
# Test SVMs using the One vs. All scheme
def test_svms(svms_path, test_dataset, test_directory, train_parameters, sift, result_df):
    """

    :param svms_path: The path to the directory for SVMs
    :param test_dataset: A list of the names of classes of the test dataset (is a list of  folders names)
    :param test_directory: The path of the directory that contains the test dataset
    :param train_parameters: A tuple with the parameters that will be used to train the SVMs
    :param sift:SIFT object that will be used by extract_local_features
    :param result_df: DataFrame of the results for SVM classification

    """

    # Initialize variable for counting the number of images
    number_of_images = 0
    # Initialize variable for counting the number of correctly classified images
    number_of_correct = 0

    for folder, class_i in zip(test_dataset, (range(len(test_dataset)))):
        # Create the path of every class
        folder_path = test_directory + "/" + folder
        # folder_path = os.path.join(test_directory, folder)
        # Create a list containing the names of the entries in the directory given by path
        files = os.listdir(folder_path)
        for file in files:
            # Create the path of every image
            path = folder_path + "/" + file
            # path = os.path.join(folder_path, file)
            # Call svm_one_vs_all
            prediction = svm_one_vs_all(svms_path, test_dataset, path, train_parameters, sift)
            number_of_images += 1
            # Prediction
            if prediction == class_i:
                number_of_correct += 1

            # DataFrame of the results for classification
            result_df = result_df.append(
                pd.Series([path, class_i, prediction, train_parameters[0], train_parameters[1],
                           train_parameters[2]],
                          index=result_df.columns), ignore_index=True)

    # Calculate accuracy
    # The percentage of successful classifications
    accuracy = round(number_of_correct * 100 / number_of_images, 4)

    # Prints
    print("One-Versus-All SVM prediction completed.\n")
    print("Number of vocabulary words =", train_parameters[0])
    print("Type of kernel : ", train_parameters[1])
    print("Epsilon = ", train_parameters[2])
    print("Number of tested pictures: ", number_of_images)
    print("Number of pictures correctly classified: ", number_of_correct)
    print("The success rate is: ", accuracy, "%")
    print("\n\n")

    return result_df
```

III. Ανάλυση Αποτελεσμάτων.

// Ανάλυση Αποτελεσμάτων με την χρήση "K-NN":

```
Creating vocabularies...
vocabulary_50.npy
Extracting features...
The vocabulary vocabulary_50.npy has been created and saved successfully
vocabulary_150.npy
Extracting features...
The vocabulary vocabulary_150.npy has been created and saved successfully

Number of images :  482
Vocabulary :  vocabularies/vocabulary_150.npy
Filename :  bovw_encoded_descriptors_150


Number of images :  482
Vocabulary :  vocabularies/vocabulary_50.npy
Filename :  bovw_encoded_descriptors_50

K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 2
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  24
The success rate is:  46.1538 %


K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 5
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  25
The success rate is:  48.0769 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 9
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  32
The success rate is:  61.5385 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 17
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  31
The success rate is:  59.6154 %
```

III. Ανάλυση Αποτελεσμάτων.

```
K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 25
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  31
The success rate is:  59.6154 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 37
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  33
The success rate is:  63.4615 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 40
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  32
The success rate is:  61.5385 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 55
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  33
The success rate is:  63.4615 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 150
Number of neighbors (K) : 60
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  34
The success rate is:  65.3846 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 2
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  28
The success rate is:  53.8462 %
```

III. Ανάλυση Αποτελεσμάτων.

```
K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 5
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  36
The success rate is:  69.2308 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 9
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  37
The success rate is:  71.1538 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 17
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  37
The success rate is:  71.1538 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 25
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  37
The success rate is:  71.1538 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 37
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  35
The success rate is:  67.3077 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 40
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  38
The success rate is:  73.0769 %
```

III. Ανάλυση Αποτελεσμάτων.

```
K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 55
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  34
The success rate is:  65.3846 %



K-Nearest-Neighbors prediction completed.

Number of visual words : 50
Number of neighbors (K) : 60
Number of trained pictures :  482
Number of tested pictures :  52
Number of pictures correctly classified:  33
The success rate is:  63.4615 %


Process finished with exit code 0
```

// Ανάλυση Αποτελεσμάτων με την χρήση "K-NN": Όπως είναι ορατό από τα παραπάνω αποτελέσματα, όσο μικρότερη είναι η τιμή της παραμέτρου "K" τόσο καλύτερα αποτελέσματα παίρνουμε για τις 50 λέξεις, ενώ το αντίθετο συμβαίνει για τις 150 λέξεις. Παρόλα αυτά και στις δύο περιπτώσεις η πορεία δεν είναι γραμμική. Παρατηρούνται καλύτερα αποτελέσματα για K: 40 και για K: 60 στις 50 και 150 λέξεις αντίστοιχα. Για υψηλότερα ποσοστά ακρίβειας κατά την ταξινόμηση συνιστάται η δημιουργία λεξικών με περισσότερες λέξεις.

// Ανάλυση Αποτελεσμάτων με την χρήση "SVM": Με την χρήση των kernels τύπου "["INTER", "SIGMOID", "LINEAR"]": Παρατηρώ ότι επίσης η τιμή της παραμέτρου K καθορίζει όμοια το ποσοστό ακρίβειας των αποτελεσμάτων, όμως ανάλογα τον τύπο "Kernel" που χρησιμοποιείται, έχω και διαφορετικά αποτελέσματα. Τα μεγαλύτερα ποσοστά υπάρχουν με τον "SIGMOID" τύπο "Kernel".

Πηγές:

[

- support-vector-machines-tutorial
- machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- bag-of-visual-words-in-a-nutshell-9ceea97ce0fb
- understanding-k-means-clustering-in-machine-learning-6a6e67336aa1]

]