

# Όραση Υπολογιστών

## 2020-2021



## Εργασία 3η

### Καρυπίδης Ευστάθιος 57556

Στην παρακάτω τεχνική αναφορά περιγράφεται η λύση της 3ης εργασίας που έχει ως στόχο την ταξινόμηση πολλών κλάσεων (multi-class classification). Συγκεκριμένα ζητείται να υλοποιηθεί η λειτουργία ταξινόμησης μιας εικόνας κάνοντας χρήση των ταξινομητών k-NN και SVM. Αφού γίνει αναφορά στο απαραίτητο θεωρητικό υπόβαθρο, θα περιγραφούν τα στάδια της λειτουργίας της ταξινόμησης καθώς επίσης θα γίνει και σύγκριση των τελικών αποτελεσμάτων με σκοπό τον έλεγχο της επίδρασης εμπλεκόμενων παραμέτρων του προβλήματος. Τέλος, θα σχολιαστούν πιθανές αποτυχίες των ταξινομητών.

# Περιγραφή του προβλήματος - Θεωρητική ανάλυση

Για την υλοποίηση της εργασίας δόθηκαν δύο βάσεις εικόνων "imagedb" και "imagedb\_test" οι οποίες αποτελούν υποσύνολο της βάσης εικόνων <https://btsd.ethz.ch/shareddata/> (BelgiumTS Dataset), όπου η πρώτη θα χρησιμοποιηθεί για την εκπαίδευση του συστήματος και η δεύτερη για την δοκιμή και αξιολόγηση του.

Γενικά, η διαδικασία που θα ακολουθηθεί είναι η εξής:

1. Ανίχνευση σημείων ενδιαφέροντος και εξαγωγή περιγραφών για κάθε εικόνα του συνόλου δεδομένων εκπαίδευσης
2. Παραγωγή του οπτικού λεξικού(visual vocabulary) βασισμένη στο μοντέλο Bag Of Visual Words και με χρήση του αλγορίθμου K-Means.
3. Κωδικοποίηση εικόνων εκπαίδευσης(αλλά και ελέγχου) χρησιμοποιώντας το λεξικό που παράχθηκε και εξαγωγή ιστογραμμάτων.
4. Ταξινόμηση με τη χρήση των Knn και SVM.

Στην υλοποίηση μου οργάνωσα των κώδικα σε συναρτήσεις ώστε να είναι ακόμη πιο κατανοητή η διαδικασία των βημάτων.

## Ανίχνευση σημείων ενδιαφέροντος - Εξαγωγή περιγραφών

Αρχικά, όπως προαναφέρθηκε για κάθε εικόνα γίνεται εντοπισμός των σημείων ενδιαφέροντος με την χρήση ανιχνευτών και στην συνέχεια μέσω περιγραφών εξάγεται ένα διάνυσμα χαρακτηριστικών για κάθε σημείο ενδιαφέροντος. Αυτά τα διανύσματα είναι πολύ χρήσιμα καθώς μπορούν να χρησιμοποιηθούν ως ταυτότητα του χαρακτηριστικού. Έτσι είναι δυνατό να αποφανθούμε εαν 2 χαρακτηριστικά είναι όμοια. Στην συγκεκριμένη εργασία θα χρησιμοποιηθεί ο ανιχνευτής και περιγραφέας SIFT ο οποίος έχει αναλυθεί σε προηγούμενα εργαστήρια και εργασία. Συνοπτικά, ο Sift είναι αδιάφορος σε πιθανούς μετασχηματισμούς, δηλαδή δεν αλλοιώνεται το αποτέλεσμα του εαν εφαρμοστούν στην εικόνα μετασχηματισμοί όπως translation, rotation, scale η εαν υπάρχουν διαφοροποιήσεις στο φωτισμό η στην οπτική γωνία.

Ολοκληρώνοντας την διαδικασία αυτή αν απλά συγκεντρώσουμε στο σύνολο τα διανύσματα της κάθε εικόνας(από όλες τις κλάσεις) τότε πιθανότατα να εμφανίζονται πολλές φορές διανύσματα "ίδια" (πρακτικά μοιάζουν παρα πολύ) που πιθανόν να περιγράφουν το ίδιο πράγμα. Συνεπώς, απαιτείται κάποιου είδους ομαδοποίηση.

## Παραγωγή οπτικού λεξικού - Bag of Visual Words

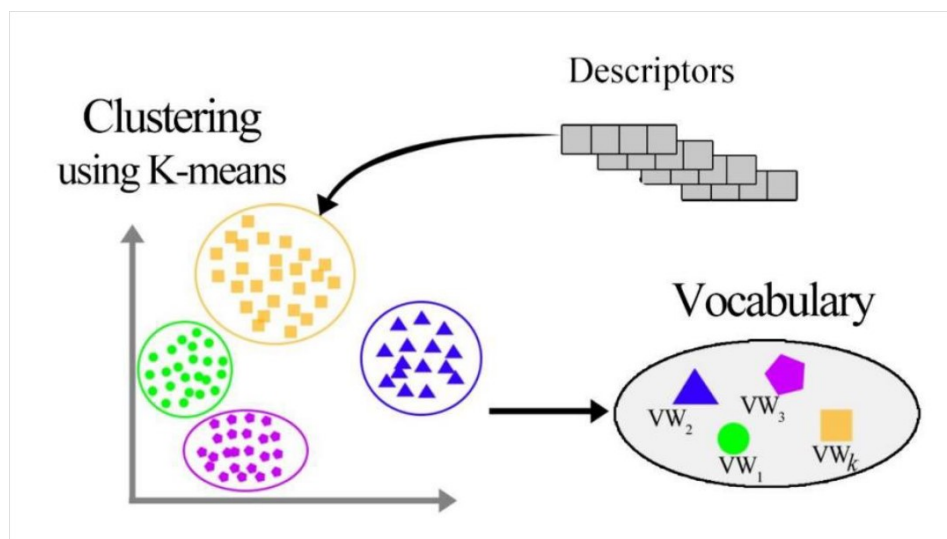
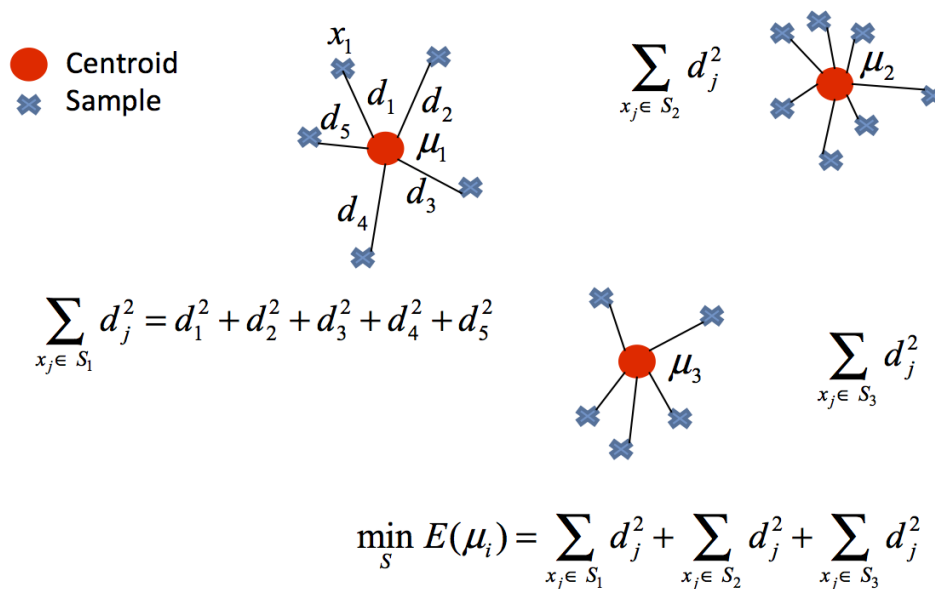
Το bag of visual words είναι ένας κλασσικός και διαδεδομένος αλγόριθμος ο οποίος χρησιμοποιείται για αναγνώριση αντικειμένων και ταξινόμηση εικόνων. Η γενική ιδέα του BOVW είναι να κωδικοποιήσει το σύνολο των τοπικών χαρακτηριστικών που εντοπιστηκαν σε μία εικόνα, σε μία καθολική αναπαράσταση η οποία μπορεί στην συνέχεια να χρησιμοποιηθεί για την σύγκριση δύο εικόνων με βάση το περιεχόμενό τους. Για την υλοποίηση ενός λεξικού, χρησιμοποιείται κάποια μέθοδος ομαδοποίησης/συσταδοποίησης(clustering). Στο πρόβλημά μας λοιπόν, θέλουμε απο το σύνολο των περιγραφών από όλες τις εικόνες να γίνει ομαδοποίηση, δηλαδή να τους διαχωρίσουμε και δημιουργήσουμε διακριτές ομάδες "χαρακτηριστικών"/οπτικών λέξεων με βάση την συχνότητα εμφάνισης και τις αποστάσεις μεταξύ τους(ομοιότητα). Μπορούμε να πούμε πως η ομαδοποίηση μπορεί να θεωρηθεί καλή εαν υπάρχει μεγάλη ομοιότητα εντός ομάδας και μικρή μεταξύ των ομάδων. Στα πλαίσια της εργασίας θα χρησιμοποιηθεί ο αλγόριθμος K-means που είναι απο τους πιο δημοφιλείς αλγορίθμους ομαδοποίησης/συσταδοποίησης.

## Αλγόριθμος K-means

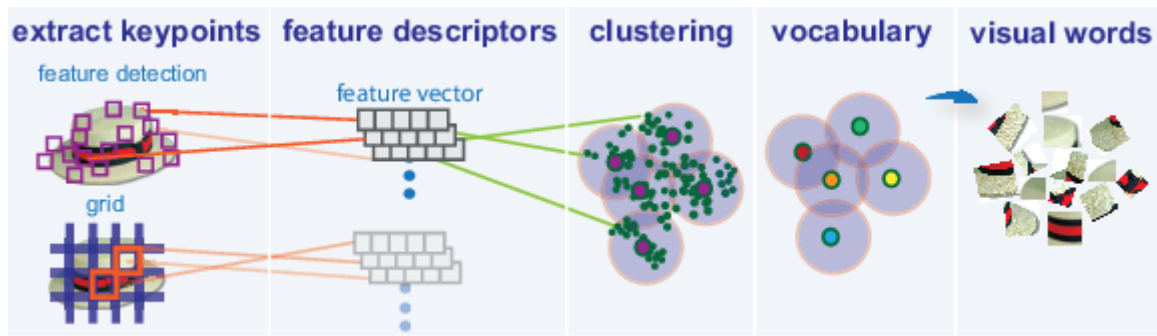
Ο αλγόριθμος K-means δέχεται ως είσοδο τον αριθμό K των ομάδων που θέλουμε να δημιουργήσουμε. Στόχος του είναι για κάθε ομάδα είναι να υπολογίσει τα βέλτιστα κέντρα της. Αυτό το πετυχαίνει προσπαθώντας να ελαχιστοποιήσει μια συνάρτηση κόστους την οποία ορίζουμε ως την ευκλείδια απόσταση μεταξύ κέντρων των ομάδων και των σημείων/χαρακτηριστικών μιας ομάδας.

$$D(X, M) = \sum_{\text{cluster } k} \sum_{\text{point } i \text{ in cluster } k} (\mathbf{x}_i - \mathbf{m}_k)^2$$

Θα μπορούσαμε να πούμε ότι τα βήματα του αλγορίθμου είναι τα εξής. Αρχικά, επιλέγονται τυχαία K κέντρα. Στην συνέχεια υπολογίζεται η απόσταση του κάθε χαρακτηριστικού (σε εμάς του κάθε διανύσματος) από όλα τα κέντρα και γίνεται ανάθεση του κάθε χαρακτηριστικού στο κέντρο με την μικρότερη απόσταση από αυτό. Στην συνέχεια μέσω της συνάρτησης κόστους που ορίστηκε και ανάλογα με τα σημεία που ανατέθηκαν σε κάθε κέντρο επαναυπολογίζεται και ενημερώνεται η θέση του κάθε κέντρου. Η διαδικασία συνεχίζεται με τον υπολογισμό των νέων αποστάσεων και κέντρων και επαναλαμβάνεται έως ο αλγόριθμος τερματιστεί βάσει συγκεκριμένων κριτηρίων.



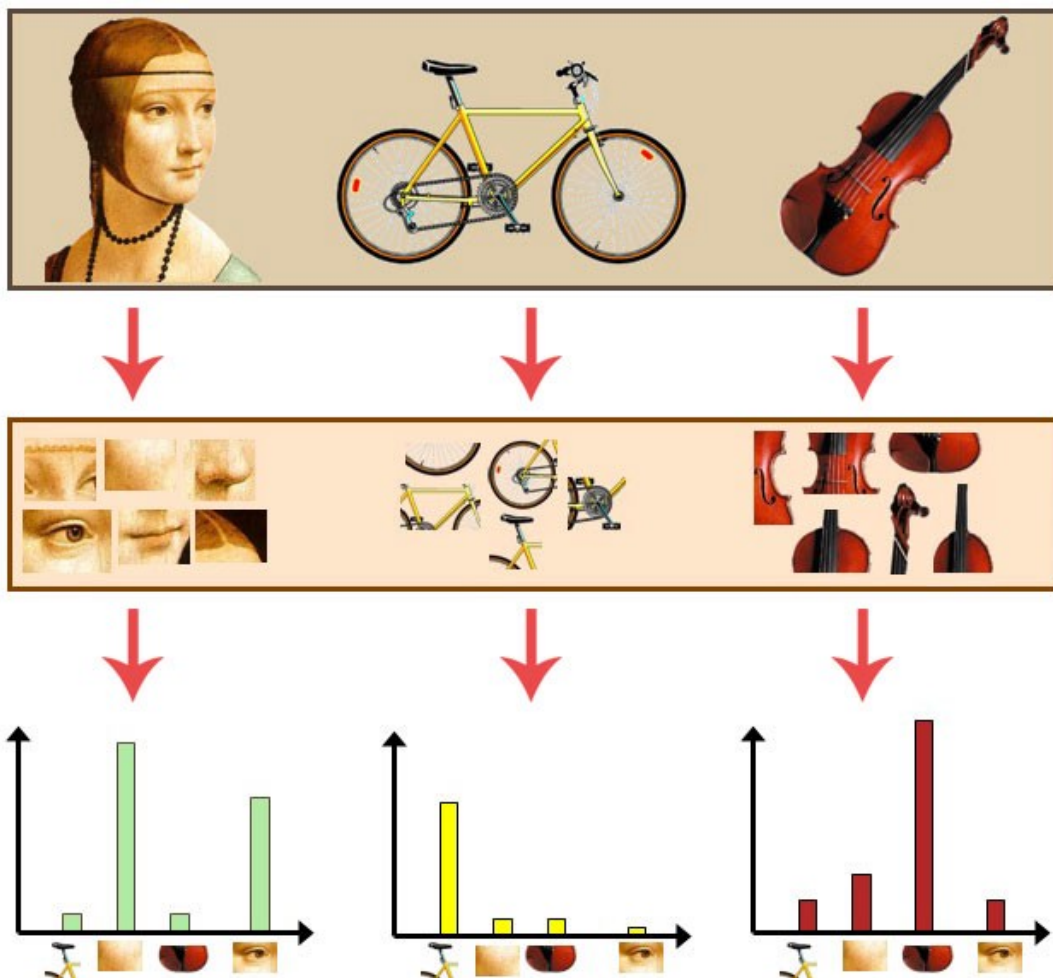
Έως τώρα λοιπόν αυτό που έχει υλοποιηθεί θα μπορούσε να δειχθεί σχηματικά ως εξής:



Επόμενο βήμα είναι η συσχέτιση των περιεχομένων του λεξικού, δηλαδή των οπτικών λέξεων με τους περιγραφείς κάθε εικόνας με σκοπο την παραγωγή ενός ιστογράμματος.

### Κωδικοποίηση (Encoding)

Έχοντας πλέον το λεξικό μπορούμε να κωδικοποιήσουμε το περιεχόμενο των εικόνων με βάση αυτό και τελικά να παράξουμε ένα ιστόγραμμα το οποίο μας δίνει την συχνότητα εμφάνισης των λέξεων του λεξικού σε κάθε εικόνα. Χρησιμοποιώντας αυτήν την πληροφορία καταφέρνουμε σε επίπεδο κώδικα να γνωρίζουμε τι υπάρχει μέσα σε μία εικόνα. Ειδικότερα, το ιστόγραμμα αυτό παράγεται με την εξής διαδικασία. Για κάθε λέξη(διάνυσμα χαρακτηριστικών) μια εικόνας συγκρίνεται με τις λέξεις του λεξικού και αντιστοιχείζεται με την λέξη όπου υπάρχει η μικρότερη απόσταση(δηλαδή μεγαλύτερη ομοιότητα) και μόλις ολοκληρωθεί αυτό γίνεται υπολογισμός της συχνότητας εμφάνισης(ιστόγραμμα). Αξίζει να σημειωθεί τον υπολογισμό της συχνότητα εμφάνισης φροντίζουμε να εφαρμόσουμε κανονικοποίηση ώστε τα ιστογράμματα να μην εξαρτώνται από το πλήθος των λέξεων αλλά να εκφράζουν κατανομή. Τέλος, σώζονται τα ιστογράμματα ώστε να χρησιμοποιηθούν.

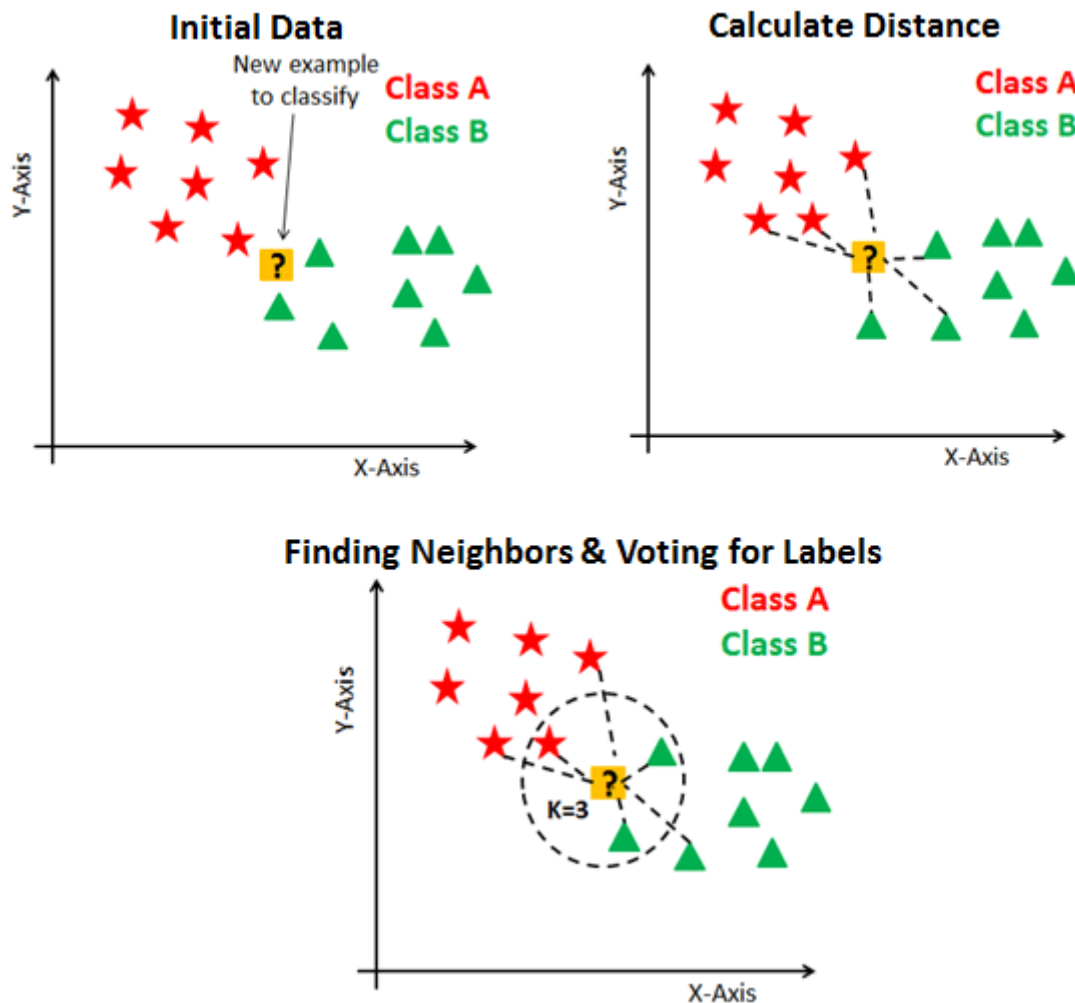


# Ταξινόμηση

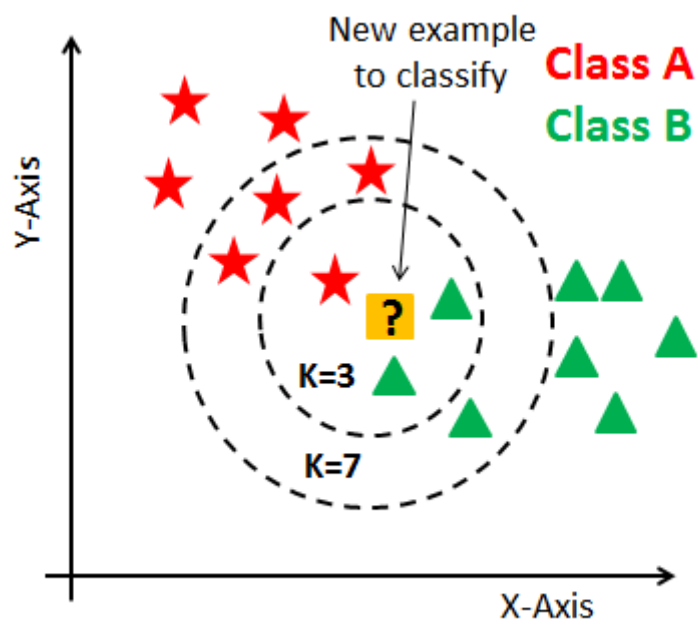
Έχοντας υπολογίσει ιστογράμματα τόσο για τις εικόνες εκπαίδευσης όσο και για τις εικόνες δοκιμής μπορούμε να προχωρήσουμε στο κομμάτι ταξινόμησης. Όπως προαναφέρθηκε θα εξεταστούν οι περιπτώσεις των ταξινομητών K-NN και SVM.

## K-NN Classifier

Ο K-NN είναι το ακρωνύμιο του K-Nearest Neighbors και είναι από τους πιο απλούς, συνηθισμένους αλγόριθμους ταξινόμησης ο οποίος μπορεί να εφαρμοστεί άμεσα για multi-class classification δηλαδή να πάρει επιλέξει μεταξύ πολλών κλάσεων καθώς υλοποιεί μη γραμμικά decision boundaries (όρια αποφάσεων). Ακόμη, μπορεί να θεωρηθεί και γρήγορος αλγόριθμος καθώς δεν χρειάζεται εκπαίδευση αλλά ωστόσο καθυστερεί λίγο περισσότερο κατά την λειτουργία της ταξινόμησης. Ειδικότερα η διαδικασία που ακολουθεί είναι: Αφού υπολογιστεί το ιστογράμμο της άγνωστης εικόνας(εικόνα ερώτημα) με βάση την διαδικασία που αναφέρθηκε παραπάνω, γίνεται σύγκριση με τα ιστογράμματα όλων των εικόνων του συνόλου εκπαίδευσης και υπολογίζονται οι αποστάσεις. Στην συνέχεια "βλέποντας" τους K γείτονες/εικόνες με μικρότερες αποστάσεις ορίζει ως label της άγνωστης εικόνας, την κλάση η οποία εμφανίζεται περισσότερες φορές ανάμεσα στους γείτονες αυτούς. Προφανώς ο αριθμός των γειτώνων μπορεί να επηρεάσει το αποτέλεσμα της πρόβλεψης.



Όπως βλέπουμε στο απλοϊκό παράδειγμα ενός binary classification εάν ορίσουμε  $K=3$  η πρόβλεψη θα είναι class B ενώ αν ορίσουμε  $K=7$  τότε η πρόβλεψη θα είναι Class A.

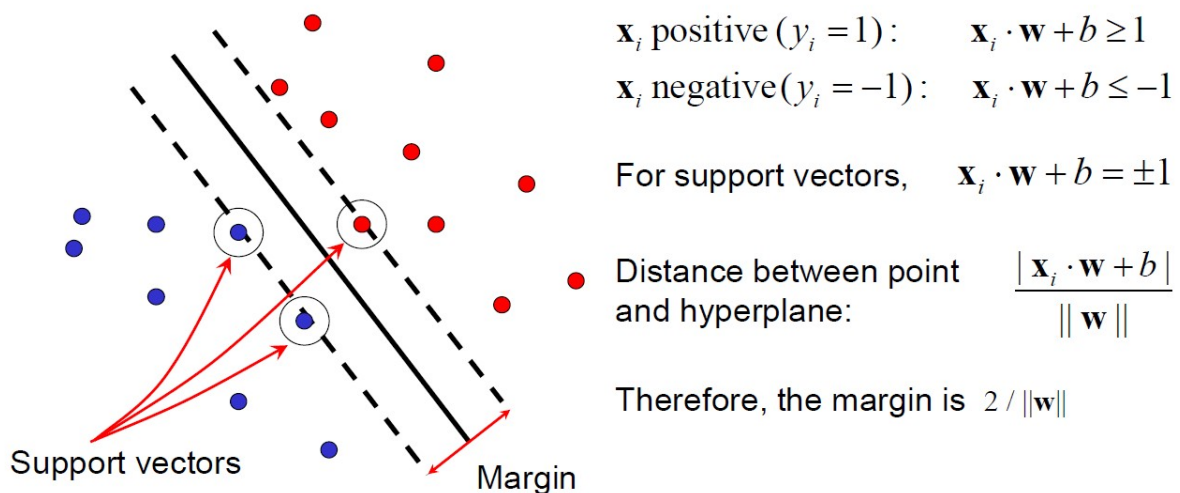




## SVM Classifier

Ο ταξινομητής SVM είναι ακρωνύμιο του Support Vector Machines. Είναι ένας γραμμικός ταξινομητής δηλαδή μπορεί να "τραβήξει" ένα γραμμικό decision boundary πράγμα που εκ πρώτης όψεως καθιστά αδύνατο το multiclass classification. Ωστόσο μπορεί να ακολουθηθεί η μέθοδος one versus all δηλαδή να εκπαιδευτεί ένας ταξινομητής για κάθε κλάση δηλαδή ένα υπερεπίπεδο το οποίο θα διαχωρίζει την εκάστοτε κλάση απ όλες τις υπόλοιπες. Αξίζει να σημειωθεί ότι ο SVM δεν επιλέγει ένα οποιοδήποτε υπερεπίπεδο όπως για παράδειγμα θα έκανε ένα perceptron αλλά προσπαθεί να βρει το βέλτιστο δηλαδή αυτό που να έχει μεγιστή απόσταση από τις 2 κλάσεις των δεδομένων εκπαίδευσης. Ακόμη, μπορεί να υλοποιήσει και ταξινομήσει και μη γραμμικά διαχωρίσιμα δεδομένα, μέσω μιας διαδικασίας όπου τα δεδομένα αυτά προβάλλονται σε ένα χώρο υψηλότερων διαστάσεων χαρακτηριστικών (τουλάχιστον μία φορά μεγαλύτερο) όπου εκεί πλέον τα δεδομένα είναι γραμμικά διαχωρίσιμα. Αυτό γίνεται με μία διαδικασία που ονομάζεται kernel trick.

Εδώ φαίνεται η διαδικασία για την εύρεση του βέλτιστου υπερεπιπέδου.



## Finding the maximum margin hyperplane

1. Maximize margin  $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

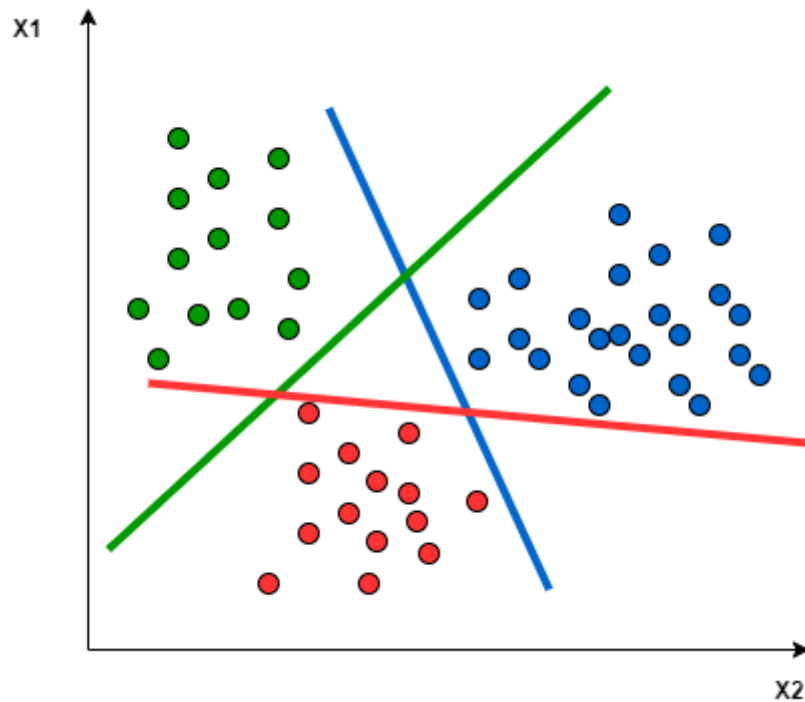
$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

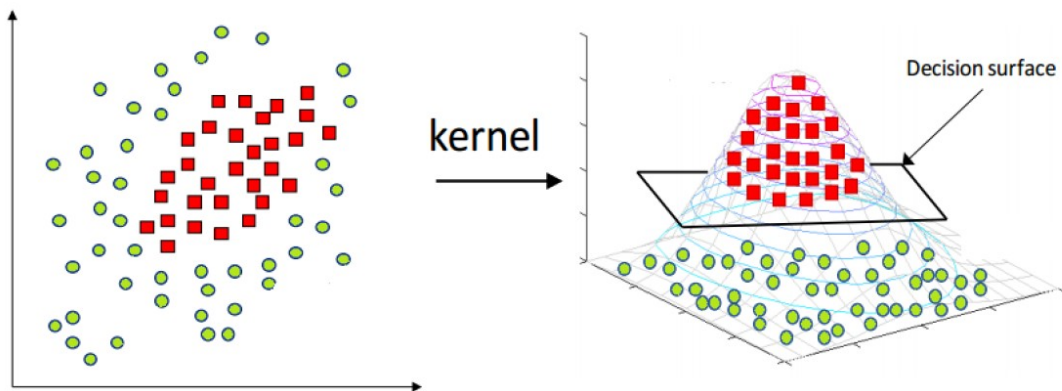
*Quadratic optimization problem:*

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

Εδώ παρουσιάζεται πως επιτυγχάνεται multiclass classification μέσω one versus all. Ουσιαστικά για κάθε μία από τις 3 κλάσεις υλοποιείται ένας SVM δηλαδή ένα υπερεπίπεδο.



Τέλος, παρουσιάζεται πώς δεδομένα που δεν είναι γραμμικά διαχωρίσιμα στις 2 διαστάσεις, μέσω του kernel trick και της προβολής σε 3 διαστάσεις πλέον είναι γραμμικά διαχωρίσιμα και μπορεί ένας SVM να υλοποιήσει ένα υπερεπίπεδο το οποίο όμως στις 2 διαστάσεις δεν θα είναι μια απλή γραμμή.





# Κώδικας - Προγραμματιστική υλοποίηση

Για να είναι πιο κατανοητός ο κώδικας κάθε βήμα έχει υλοποιηθεί με μία ή περισσότερες συναρτήσεις.

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

# Local feature Extraction using SIFT
def extract_local_features(path):
    img = cv.imread(path)
    _, img_desc = sift.detectAndCompute(img, None)
    return img_desc

# Extract descriptors for all images in training set
def get_train_desc(data_folders):
    print('Extracting features...')
    train_descriptors = np.zeros((0, 128))
    for folder in data_folders:
        class_folder = os.path.join(train_data_path, folder)
        print("Accessing class folder ", class_folder)
        files = os.listdir(class_folder)
        for file in files:
            path = os.path.join(class_folder, file)
            desc = extract_local_features(path)
            if desc is None:
                continue
            train_descriptors = np.concatenate((train_descriptors, desc),
axis=0)
    return train_descriptors
```

Αρχικά, δημιούργησα μία συνάρτηση για τον SIFT ώστε να μου επιστρέφει κατευθείαν τους descriptors μια εικόνας της οποίας δίνω το path. Στην συνέχεια μέσω της συνάρτησης get\_train\_desc() η οποία παίρνει ως όρισμα το path του φακέλου που περιέχει τις εικόνες εκπαιδευσείς ομαδοποιημένες σε υποφακέλους ανάλογα με τις κλάσεις τους και επιστρέφει ένα διάνυσμα που περιέχει όλους τους περιγραφείς για όλες τις εικόνες.

Χρησιμοποιώντας το παρακάτω κώδικα-main αποθηκεύουμε στην μεταβλητή train\_desc καθώς και σε ένα αρχείο ώστε να μην τους παράγουμε κάθε φορά, αλλά να τους φορτώνουμε.

```
# -----Main-----
sift = cv.xfeatures2d_SIFT.create()
# Extract Database
train_data_path = "imagedb"
test_data_path = "imagedb_test"
train_data_folders = os.listdir(train_data_path)
test_data_folders = os.listdir(test_data_path)

train_desc = get_train_desc(train_data_folders)
np.save("train_descriptors.npy", train_desc)
train_desc = np.load("train_descriptors.npy")
```

Το αποτέλεσμα που έχουμε είναι το παρακάτω. Όπως ήταν αναμενόμενο η μία διάσταση είναι πολύ μεγάλη καθώς έχει το σύνολο των διανυσμάτων για όλες τις εικόνες εκπαίδευση ενώ η 2η διάσταση είναι 128 πράγμα επίσης αναμενόμενο καθώς μιλάμε για διανύσματα που παράχθηκαν από τον SIFT.

```
> train_desc = (ndarray: (257442, 128)) [[ 0. 0. 0. ... 0. 0. 0.] [ 0. 0. 0. ... 85. 35. 18.] [20. 0. 0. ... 0. 0. 0.] ... [ 5. 1. 0. ... 0. 0. 0.] [16. 8. 7. ... 0. 0. 0.] [63. 15. 0. ... 1. 66. 31.]] ...View as Array
> min = (str) 'ndarray too big, calculating min would slow down debugging'
> max = (str) 'ndarray too big, calculating max would slow down debugging'
> shape = (tuple: 2) (257442, 128)
> dtype = (dtype: 0) float64
> size = (int) 32952576
> array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x00000257EA345630>
```

Στην συνέχεια ακολουθεί η παραγωγή του λεξικού οπτικών λέξεων. Αυτό υλοποιείται μέσω της συνάρτησης `create_vocabulary()` η οποία δέχεται ως ορίσματα τους περιγραφείς που παράχθηκαν παραπάνω καθώς και τον αριθμό λέξεων που επιθυμούμε. Ειδικότερα η παραγωγή του λεξικού γίνεται με τον αλγόριθμο K-means μέσω της συνάρτησης της `opencv cv.BOWKMeansTrain`.

```
def create_vocabulary(train_descriptors, vocabulary_size):
    print('Creating vocabulary using K-Means...')
    K = vocabulary_size
    term_criteria = (cv.TERM_CRITERIA_EPS, 30, 0.1)
    trainer = cv.BOWKMeansTrainer(K, term_criteria, 1, cv.KMEANS_PP_CENTERS)
    vocab = trainer.cluster(train_descriptors.astype(np.float32))
    np.save('Vocabulary_' + str(K) + '.npy', vocab)
    print("Vocabulary_" + str(K) + " created and saved at " + str(os.getcwd()))
    return
```

Την συνάρτηση την καλούμε και στην συνέχεια φορτώνουμε

```
create_vocabulary(train_desc, 50)
vocabulary = np.load('Vocabulary_50.npy')
```

Το αποτέλεσμα είναι το εξής. Ένας πίνακας όπου η μία διάσταση του (η 2η) παραμένει 128 αλλά η άλλη είναι όσο και ο αριθμός των λέξεων που ορίσαμε. Στο παράδειγμα εδώ 50.

```
> vocabulary = (ndarray: (50, 128)) [[33.880108 14.157668 9.267902 ... 12.15931 10.963457 13.459249] [11.307376 10.801314 18.919868 ... 41.32443 9.887199 8.321553] [18.42416 9.337306 9.732936 ... 23.504513 3.527672 3.6197183] ... [20.616955 17. ...View as Array
> min = (float32) 0.64201206
> max = (float32) 174.48343
> shape = (tuple: 2) (50, 128)
> dtype = (dtype: 0) float32
> size = (int) 6400
> array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x0000029C78A82E10>
```

Στην συνέχεια ακολουθεί η κωδικοποίηση των περιγραφέων κάθε εικόνας με βάση το λεγικό οπτικών λέξεων και η δημιουργία ιστογραμμάτων καθώς και η δημιουργία ενός πίνακα που έχει αποθηκευμένες τις κλάσεις των εικόνων. Αυτά γίνονται μέσω των συναρτήσεων `create_boww_descriptor()` και `boww_descriptors()`. Η πρώτη (`create_boww_descriptor()`) υπολογίζει τις αποστάσεις L2 (Ευκλείδειες) για κάθε περιγραφέα μιας εικόνας με τις λέξεις του λεξικού ενώ η δεύτερη (`boww_descriptors()`) κάνει αυτήν την διαδικασία για όλες τις εικόνες καθώς επίσης υλοποιεί και τον πίνακα `labels` που περιέχει το όνομα της κλάσης κάθε εικόνας.

```
# Compute Normalized Histogram
def create_boww_descriptor(desc, voc):
    bow_desc = np.zeros((1, voc.shape[0]))
    for d in range(desc.shape[0]):
        diff = desc[d, :] - voc
        distances = np.sum(np.square(diff), axis=1)
        mini = np.argmin(distances)
        bow_desc[0, mini] += 1
    return bow_desc / np.sum(bow_desc)
```

Την συνάρτηση `bonw_descriptors()` θα την καλέσουμε τόσο για τις εικόνες του συνόλου εκπαίδευσης όσο και του συνόλου ελέγχου καθώς θα χρειαστούν αργότερα.

[illegible]

Πριν προχωρήσω στους ταξινομητές υλοποίησα ένα λεξικό το οποίο συνδέει τα ονόματα των κλάσεων με τα indexes 0-33. Επίσης υλοποιήθηκε μία συνάρτηση η οποία κάνει την ανάποδη αντιστοίχιση δηλαδή επιστρέφει την τιμή του κλειδιού(δηλαδή το ονομα των κλάσεων) βάση την τιμή του value.

```
def get_key(my_dict, val):
    for key, value in my_dict.items():
        if val == value:
            return key

dic = {}
for tdf in range(len(test_data_folders)):
    dic[int(test_data_folders[tdf])] = tdf
```

Το λεξικό που παράγεται είναι το εξής :

```
{4: 0, 5: 1, 7: 2, 8: 3, 10: 4, 12: 5, 13: 6, 17: 7, 18: 8, 19: 9, 21: 10, 27: 11, 28: 12, 29: 13, 30: 14, 31: 15, 32: 16, 34: 17, 35: 18, 37: 19, 38: 20, 39: 21, 41: 22, 42: 23, 43: 24, 45: 25, 47: 26, 51: 27, 53: 28, 54: 29, 56: 30, 57: 31, 58: 32, 59: 33}
```

Στην συνέχεια προχωράμε στην υλοποίηση του K-NN. Όπως τονίζεται στην εκφώνηση πρέπει να γίνει χωρίς την χρήση της σχετική συνάρτησης `cv.ml.KNearest_create()`. Συνεπώς υλοποιήθηκαν τα βήματα του αλγορίθμου που περιγράφηκαν παραπάνω.

```
def knn_classifier(k, train_boww_descriptors, test_boww_descriptors,
                  training_class_labels, testing_class_labels,
                  dictionary):
    knn_predictions = []
    for i in range(testing_class_labels.shape[0]):
        Sum = np.zeros(len(dictionary), dtype=int)
        differences = test_boww_descriptors[i] - train_boww_descriptors
        distances = np.sum(np.square(differences), axis=1)
        sorted_indices = np.argsort(distances)
        for j in range(k):
            pos = dictionary[int(training_class_labels[sorted_indices[j]])]
            Sum[pos] += 1
        result = np.argmax(Sum, axis=0)
        knn_predictions.append(get_key(dictionary, result))
    return knn_predictions
```

Συνοπτικά η παραπάνω συνάρτηση λειτουργεί ως εξής: Για κάθε εικόνα που θέλουμε να ταξινομήσουμε υπολογίζεται η L2 απόσταση του ιστογραμματος της με όλα τα ιστογράμματα των εικόνων εκπαίδευσης και γίνεται sort των θέσεων των αποστάσεων από την μικρότερη προς την μεγαλύτερη. Έπειτα για τον αριθμό γειτόνων k που έχουν οριστεί κοιτάμε σε ποιές κλάσεις ανήκουν και αυξάνουμε κατά ένα αντίστοιχες θέσεις ενός πίνακα `sum`(μεγέθους 34) οπότε τελικά αυτός ο πίνακας μας περιγράφει σε ποιές κλάσεις ανήκουν οι κοντινότερες εικόνες σε σχέση με την εικόνα ερώτημα. Παίρνοντας την θέση που βρίσκεται η μεγαλύτερη τιμή και αποκωδικοποιώντας την θέση αυτή μέσω του λεξικού παίρνουμε την "απόφαση" του ταξινομητή. Τέλος, κρατάμε όλες τις προβλέψεις σε μία λίστα με σκοπό την συγκρισή τους με τα πραγματικά labels και αξιολόγηση του μοντέλου.

```
KNN_predictions = knn_classifier(15, train_boww_desc, test_boww_desc,
                                train_class_labels, test_class_labels, dic)
```

```
> KNN_predictions = ([int: 2140] [28, 19, 4, 47, 13, 35, 13, 56, 32, 7, 19, 19, 32, 18, 18, 7, 7, 32, 32, 7, 32, 7, 7, 51, 7, 7, 7, 7, 32, 7, 57, 38, 53, 7, 47, 7, 47, 32, 7, 7, 7, 7, 7, 13, 53, 7, 51, 19, 10, 32, 18, 7, 53, 53, 53, 47, 57, 7, 7, 53, 7, 7, 53, 7, 32, 32, 19 ... View
```

Στη συνέχεια περνάμε στον ταξινομητή SVM. Η υλοποίηση του χωρίζεται σε 2 στάδια. Αυτό τις εκπαίδευσης των one vs all ταξινομητών και αυτό του classification. Αρχίζοντας από το κομμάτι της εκπαίδευσης, ουσιαστικά έπρεπε να παραμετροποιηθεί ο κώδικας του εργαστηρίου ώστε να δουλεύει αντι για 2 σε 34 κλάσεις που έχουμε στο συγκεκριμένο πρόβλημα. Μια σημαντική παράμετρος που όπως θα παρουσιαστεί αργότερα είναι το `kernel_type`. Η `opencv` δίνει διάφορα πιθανά `kernel_types` που μπορούν να χρησιμοποιηθούν. Χαρακτηριστικά βλέπουμε από τα docs τις `opencv` τις εξής επιλογές:

Enumerator	
CUSTOM	Returned by <code>SVM::getKernelType</code> 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. $K(x_i, x_j) = \min(x_i, x_j)$ .

Παιρνώντας στο κομμάτι της συνάρτησης που υλοποιήθηκε το κλειδί για την σωστή εκπαίδευση είναι η δημιουργία του πίνακα `labels`.

```
def train_svm(path, training_class_labels, training_boww_desc, dictionary,
kernel_type):
    kernel_types = {"LINEAR": cv.ml.SVM_LINEAR, "SIGMOID": cv.ml.SVM_SIGMOID,
"RBF": cv.ml.SVM_RBF,
"CHI2": cv.ml.SVM_CHI2, "INTER": cv.ml.SVM_INTER}
    os.mkdir(path)
    os.chdir(path)
    for s in range(len(dictionary)):
        class_label = get_key(dictionary, s)
        print('Training SVM_' + str(class_label))
        svm = cv.ml.SVM_create()
        svm.setType(cv.ml.SVM_C_SVC)
        svm.setKernel(kernel_types[kernel_type])
        svm.setTermCriteria((cv.TERM_CRITERIA_COUNT, 100, 1.e-06))
        labels = np.array([class_label in a for a in training_class_labels],
np.int32)
        svm.trainAuto(training_boww_desc.astype(np.float32), cv.ml.ROW_SAMPLE,
labels)
        svm.save(str(class_label))
        print("SVMs saved at ", os.getcwd())
    return
```



































Ειδικότερα ο πίνακας `labels` είναι ένας πίνακας ίδιου μεγέθους με τον πίνακα των `training labels` όπου για κάθε κλάση (δηλαδή και επανάληψη) έχει 1 όπου η τρέχουσα κλάση εμφανίζεται στον πίνακα `labels`. Ετσι, σε κάθε επανάληψη έχουμε έναν πίνακα με 1 μονο για μία κλάση και 0 για όλες τις υπόλοιπες. Αυτόν τον πίνακα τον παράγουμε μέσω ενός `list comprehension`. Σε κάθε επανάληψη λοιπόν, εκπαιδεύεται και αποθηκεύεται σε ένα φάκελο ένας SVM ταξινομητής που διαχωρίζει μία κλάση από όλες τις υπόλοιπες και συνολικά στο τέλος έχουμε 34 SVM.

Συγκεκριμένα ορίζουμε αρχικά το path που θέλουμε να αποθηκευτούν τα SVM και στη συνέχεια καλούμε την συνάρτηση ορίζοντας το kernel\_type που επιθυμούμε.

```
svm_path = os.path.join(os.getcwd(), "SVM_" + str(vocabulary.shape[0]) +
                        "_kernel_" + "RBF")
train_svm(svm_path, train_class_labels, train_bovw_desc, dic, "RBF")
```

Όπως προαναφέρθηκε κλειδί για την λειτουργία της εκπαίδευσης είναι ο πίνακας labels ο οποίος όπως βλέπουμε στο παρακάτω παράδειγμα όταν το class\_label είναι ίσο με 4 ο πίνακας labels έχει 1 στις θέσεις όπου και ο πίνακας training\_class\_labels έχει 4. Αντίστοιχα, συνεχίζει η διαδικασία και για τις υπόλοιπες κλάσεις.

[illegible]

charm > Computer Vision > HW3 > SVM_50_kernel_RBF				
Name	^	Date modified	Type	Size
 4		9/1/2021 1:46 πμ	File	32 KB
 5		9/1/2021 1:46 πμ	File	26 KB
 7		9/1/2021 1:46 πμ	File	124 KB
 8		9/1/2021 1:46 πμ	File	53 KB
 10		9/1/2021 1:46 πμ	File	38 KB
 12		9/1/2021 1:46 πμ	File	52 KB
 13		9/1/2021 1:46 πμ	File	110 KB
 17		9/1/2021 1:46 πμ	File	99 KB
 18		9/1/2021 1:46 πμ	File	74 KB
 19		9/1/2021 1:46 πμ	File	87 KB
 21		9/1/2021 1:46 πμ	File	72 KB
 27		9/1/2021 1:46 πμ	File	37 KB
 28		9/1/2021 1:46 πμ	File	92 KB
 29		9/1/2021 1:46 πμ	File	63 KB
 30		9/1/2021 1:46 πμ	File	64 KB
 31		9/1/2021 1:46 πμ	File	60 KB
 32		9/1/2021 1:46 πμ	File	127 KB
 34		9/1/2021 1:46 πμ	File	65 KB
 35		9/1/2021 1:47 πμ	File	79 KB
 37		9/1/2021 1:47 πμ	File	101 KB
 38		9/1/2021 1:47 πμ	File	102 KB
 39		9/1/2021 1:47 πμ	File	116 KB
 41		9/1/2021 1:47 πμ	File	77 KB
 42		9/1/2021 1:47 πμ	File	49 KB
 43		9/1/2021 1:47 πμ	File	46 KB
 45		9/1/2021 1:47 πμ	File	80 KB
 47		9/1/2021 1:47 πμ	File	111 KB
 51		9/1/2021 1:47 πμ	File	53 KB
 53		9/1/2021 1:47 πμ	File	126 KB
 54		9/1/2021 1:47 πμ	File	120 KB
 56		9/1/2021 1:47 πμ	File	122 KB
 57		9/1/2021 1:47 πμ	File	106 KB
 58		9/1/2021 1:47 πμ	File	32 KB
 59		9/1/2021 1:47 πμ	File	68 KB



Αφού λοιπόν παραχθούν τα SVM για κάθε κλάση απομένει το κομμάτι του classification. Αυτό γίνεται μέσω της συνάρτησης `svm_classifier`.

```
def svm_classifier(path, testing_bovw_desc, dictionary):
    if os.getcwd() != path:
        os.chdir(path)
    SVMs = []
    svm_files = os.listdir(path)
    svm_files = sorted(list(map(int, svm_files)))
    svm_files = list(map(str, svm_files))
    for file in svm_files:
        SVMs.append(cv.ml.SVM_load(file))
    svm_predictions = []
    for i in range(testing_bovw_desc.shape[0]):
        result = []
        descriptor = np.expand_dims(testing_bovw_desc[i], axis=1)
        descriptor = np.transpose(descriptor)
        for svm in SVMs:
            res = svm.predict(descriptor.astype(np.float32),
                                flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            result.append(res)
        pos = int(np.argmax(result))
        svm_predictions.append(get_key(dictionary, pos))
    return svm_predictions
```

Οι πρώτες γραμμές αφορούν το path και ειδικότητα την φόρτωση στην λίστα svm\_files τα ονόματα των svm με την σειρά καθώς διαφορετικά δημιουργούνταν πρόβλημα καθώς η os.listdir δεν τα τοποθετούσε όπως επρεπε και συνεπώς χρειάστηκε να γίνει ταξινόμηση των ονομάτων αφού μετατραπούν σε ακεραίους και στην συνέχεια να μετατραπούν ξανα σε str. Στην συνέχεια αυτό που γίνεται ουσιαστικά είναι, το κάθε SVM κάνει ένα prediction και κρατιέται ο δείκτης του SVM που έκανε την καλύτερη πρόβλεψη, ο οποίος δείκτης στην συνέχεια αποκωδικοποιείται μέσω του λεξικού και μας δείνει το όνομα της κλάσης που ανήκει η εικόνα ερώτημα.

```
SVM_predictions = svm_classifier(svm_path, test_bovw_desc, dic)
```

```
SVM predictions = (list: 2149) [18, 7, 7, 47, 7, 7, 38, 7, 7, 38, 7, 7, 7, 7, 7, 7, 7, 32, 7, 38, 7, 7, 38, 7, 7, 7, 7, 39, 7, 7, 7, 38, 7, 7, 38, 7, 38, 7, 7, 7, 7, 7, 7, 38, 38, 38, 7, 38, 7, 7, 7, 38, 38, 7, 53, 53, 53, 38, 7, 7, 7, 7, 38, 7, 7, 7, 7, 7, 7, 7, 7, 7, 38, ... View
```

Τέλος απομένει το κομμάτι της αξιολόγησης δηλαδή να υπολογίζονται οι σωστές προβλέψεις και τα αντίστοιχα ποσοστά. Για το σκοπό αυτό υλοποίησα την συνάρτηση evaluate()

```
def evaluate(predictions, class_labels, dictionary):
    class_precision = {}
    count = 0
    for key in dictionary.keys():
        class_precision[key] = [0, 0, 0]
    for index in range(len(class_labels)):
        class_precision[int(class_labels[index])][1] += 1
        if predictions[index] == class_labels[index]:
            class_precision[int(class_labels[index])][0] += 1
            count += 1
    for value in class_precision.values():
        value[2] = str(float(value[0]) / float(value[1]) * 100)[:6] + "%"
    total_precision = [count, len(class_labels), str(float(count) /
float(len(class_labels)) * 100)[:8] + "%"]
    return class_precision, total_precision
```

Η συνάρτηση αυτή παίρνει ως είσοδο έναν πίνακα με τα predictions και επιστρέφει το λεξικό class\_precision και την λίστα total\_precision. Στο λεξικό ως κλειδιά υπάρχουν τα ονόματα των κλασεων και ως values είναι μια λίστα όπου έχει στην θέση 0 τις σωστές προβλέψεις για την συγκεκριμένη κλάση, στη θέση 1 έχει το σύνολο των εικόνων ελέγχου για την συγκεκριμένη κλάση και στην θέση 2 έχει το ποσοστό επιτυχίας ανα κλάση. Η λίστα total\_precision αντίστοιχα έχει στη θέση 0 το σύνολο των σωστών προβλέψεων, στη θέση 1 το σύνολο των δεδομένων ελέγχου και στην θέση 2 το συνολικό ποσοστό. Για το σκοπό αυτό υλοποίησα την συνάρτηση save\_results για αποθήκευση τους σε json αρχείο. Τέλος, συνολικά για του classifiers στην main έχουμε:

```
def save_results(class_precision, total_precision, filename):
    with open(filename + ".json", 'w') as json_file:
        Results = {
            "Total Results":
                {"Correct Predictions: ": total_precision[0],
                 "Total Images: ": total_precision[1],
                 "Accuracy: ": total_precision[2]}}
        json.dump(Results, json_file, indent=4)
        for key in class_precision.keys():
            Results = {
                "Class " + str(key) + " Results":{
                    "Correct Class "+str(key)+" Predictions: ":
                        class_precision[key][0],
                    "Total Class Images ": class_precision[key][1],
                    "Class Accuracy: ": class_precision[key][2]}}
            json.dump(Results, json_file, indent=4)
    return
```

```
# K Nearest Neighbors Classifier
K_nn = 15
KNN_predictions = knn_classifier(k_nn, train_bovw_desc, test_bovw_desc,
train_class_labels, test_class_labels, dic)
class_precision_knn, total_knn = evaluate(KNN_predictions, test_class_labels,
dic)
knn_precision = float(total_knn[0]) / float(test_class_labels.shape[0])
print("For K=" + str(k_nn) + " Correct Matches K-nn = ", total_knn[0], "Total
pictures", test_class_labels.shape[0])
print(" K-nn Accuracy {a:.5f}%".format(a=knn_precision * 100))
save_results(KNN_predictions, total_knn,
"KNN_"+str(vocabulary.shape[0])+"_K="+str(k_nn))
# Train SVM
kernel = 1
svm_path = os.path.join(os.getcwd(), "SVM_" + str(vocabulary.shape[0]) +
"_kernel_" + "RBF")
train_svm(svm_path, train_class_labels, train_bovw_desc, dic, "RBF")
# SVM Classifier
SVM_predictions = svm_classifier(svm_path, test_bovw_desc, dic)
class_precision_svm, total_svm = evaluate(SVM_predictions, test_class_labels,
dic)
svm_precision = float(total_svm[0]) / float(test_class_labels.shape[0])
print(" Correct Matches SVM = ", total_svm[0], "Total pictures",
test_class_labels.shape[0])
print(" SVM Accuracy {a:.5f}%".format(a=svm_precision * 100))
os.chdir('.')
save_results(class_precision_svm, total_svm,
"SVM_"+str(vocabulary.shape[0])+"_kernel_RBF")
```

# Σύγκριση αποτελεσμάτων και αξιολόγηση πιθανών αποτυχιών

Στο σημείο αυτό θα παρουσιαστούν αποτελέσματα μεταβάλλοντας τιμές εμπλεκόμενων παραμέτρων όπως ο αριθμός των οπτικών λέξεων, ο αριθμός των πλησιέστερων γειτόνων, και ο τύπος του πυρήνα.

K-(Nearest Neighbors)-L2 Distance	Vocabulary Size = 50	Vocabulary Size = 75	Vocabulary Size = 100
1	34.29502%	37.22661%	37.92462%
3	33.73662%	37.97115%	38.20382%
5	37.27315%	38.34342%	38.90181%
10	38.34342%	40.01861%	40.29781%
15	40.01861%	41.13541%	41.18195%
20	39.13448%	40.67008%	41.55421%
50	39.69288%	42.57794%	41.27501%
75	38.94835%	42.34528%	41.55421%
100	39.46021%	41.74034%	41.78688%

Kernel Type	Vocabulary Size = 50	Vocabulary Size = 75	Vocabulary Size = 100
RBF	30.71196%	36.57515%	43.78781%
LINEAR	28.05956%	42.29874%	44.95114%
SIGMOID	4.18799%	4.18799%	4.18799%
CHI2	48.90647%	57.18939%	54.25779%
INTER	47.83620%	52.81526%	56.11913%

Όπως παρατηρούμε αρχικά όλες οι παράμετροι μεταβάλουν την απόδοση των μοντέλων καθώς ακόμη και με τον SVM παρατηρούμε ταξινόμηση με μεγαλύτερο ποσοστό επιτυχίας με συγκεκριμένα kernel. Στον ταξινομητή K-NN παρατηρούμε βέλτιστα αποτελέσματα για λεξικό οπτικών λέξεων ίσο με 75 και για αριθμό γειτόνων ίσο με 50. Γενικά παρατηρώ ότι αρχικά με αύξηση του αριθμού των γειτόνων αυξάνεται και η ακρίβεια ενώ μετά από ένα σημείο παραμένει σταθερή ή εμφανίζονται μικρομεταβολές. Ακόμη, για σταθερό αριθμό γειτόνων παρατηρείται μία αύξηση στην απόδοση καθώς αυξάνεται ο αριθμός των οπτικών λέξεων. Ωστόσο, αυτό θέλει ιδιαίτερη προσοχή καθώς εάν αυξηθούν κατα πολύ ο αριθμός οπτικών λέξεων τα αποτελέσματα θα χειροτερεύσουν. Όσον αφορά τον ταξινομητή SVM παρατηρούμε ότι τα καλύτερα αποτελέσματα μας τα δίνουν τα kernel CHI2(Exponential Chi2 kernel) και INTER(Histogram intersection kernel) ενώ τα χειρότερα το SIGMOID ενώ για σταθερό kernel παρατηρούμε και πάλι σχετική αύξηση της ακρίβειας. Τα συγκεντρωτικά αποτελέσματα ανα κλάση έχουν αποθηκευτεί σε .json αρχεία για ευκολότερη χρήση. Ένα παράδειγμα είναι το παρακάτω.

```

{
  "Total Results": {
    "Correct Predictions": ": 915,
    "Total Images": ": 2149,
    "Accuracy": ": "42.57794%"
  }
}
  "Class 4 Results": {
    "Correct Class 4 Predictions": ": 0,
    "Total Class Images :": 12,
    "Class_Accuracy": ": "0.0%"
  }
}
  "Class 5 Results": {
    "Correct Class 5 Predictions": ": 0,
    "Total Class Images :": 3,
    "Class_Accuracy": ": "0.0%"
  }
}
  "Class 7 Results": {
    "Correct Class 7 Predictions": ": 48,
    "Total Class Images :": 90,
    "Class_Accuracy": ": "53.333%"
  }
}
  "Class 8 Results": {
    "Correct Class 8 Predictions": ": 0,
    "Total Class Images :": 12,
    "Class_Accuracy": ": "0.0%"
  }
}
  "Class 10 Results": {
    "Correct Class 10 Predictions": ": 0,
    "Total Class Images :": 28,
    "Class_Accuracy": ": "0.0%"
  }
}
  "Class 12 Results": {
    "Correct Class 12 Predictions": ": 0,
    "Total Class Images :": 3,
    "Class_Accuracy": ": "0.0%"
  }
}
  "Class 13 Results": {
    "Correct Class 13 Predictions": ": 3,
    "Total Class Images :": 39,
    "Class_Accuracy": ": "7.6923%"
  }
}
  "Class 17 Results": {
    "Correct Class 17 Predictions": ": 16,
    "Total Class Images :": 183,
    "Class_Accuracy": ": "8.7431%"
  }
}
  "Class 18 Results": {
    "Correct Class 18 Predictions": ": 0,
    "Total Class Images :": 1,
    "Class_Accuracy": ": "0.0%"
  }
}
}

```

Τα αποτελέσματα των K-NN τα συγκέντρωσα στους φακέλους K-NN results και SVM results.

Βλέποντας τα αποτελέσματα ανα κλάση για τους διάφορους συνδιασμούς παραμέτρων παρατηρούμε σε πολλές κλάσεις πολύ χαμηλή ακρίβεια ακόμη και 0%. Αυτό το αποτέλεσμα, πέραν την πιθανή αδυναμία των classifier να μην μπορούν να τραβήξουν τα κατάλληλα υπερεπίπεδα ή την μη σωστή επιλογή των παραμέτρων τους(καθώς αυτό είναι μία πολύ δύσκολη διαδικασία για την οποία δεν υπάρχουν "αυστηροί κανόνες ") πολλές φορές οφείλεται στα ίδια τα δεδομένα εκπαίδευσης. Ειδικότερα, υπάρχουν πολλοί παράμετροι και χαρακτηριστικά όσον αφορά τα δεδομένα εκπαίδευσης τα οποίοι μπορούν να "βοηθήσουν" η όχι την εκπαίδευση. Με μία σύντομη περιήγηση στα δεδομένα εκπαίδευσης και ελέγχου παρατηρούμε πως το dataset είναι unbalanced καθώς υπάρχει πολύ μεγάλος αριθμός εικόνων σε κάποιες κλάσεις και σε κάποιες όχι. Ακόμη, παρατηρείται αρκετή διαφορά στα μεγέθη των φωτογραφιών(ανάλυση), στην οπτική γωνία στην οποία έχουν τραβηχτεί, στην φωτεινότητα τους αλλά και στα background. Τέλος, οι αλγόριθμοι που χρησιμοποιήθηκαν δεν "μαθαίνουν" χαρακτηριστικά όπως πιο εξελιγμένες αρχιτεκτονικές νευρωνικών δικτύων(πχ συνελκτικά νευρωνικά δίκτυα).

Ενδεικτικά παρουσιάζω μερικές εικόνες από κάποιες κλάσεις που δείχνουν ότι αναφέρθηκε.



## Βελτίωση αποτελεσμάτων του K-NN

Παρατηρώντας τις μετρικές που προσφέρει η βιβλιοθήκη sklearn(οπώς φαίνεται στο παρακάτω σχήμα), έγινε αναζήτηση για το πως δουλεύουν αυτές σε highdimensional space. Στα πλαίσια αυτής της αναζήτησης βρέθηκαν στοιχεία καθώς και ένα paper, όπου οπώς και φάνηκε τελικά η απόσταση L1 (Manhattan) είναι πολύ πιο αποδοτική από την L2 (Ευκλειδία). Αυτό φαίνεται να αποδίδεται στο φαινόμενο curse of high dimensionality.

identifier	class name	args	distance function
"euclidean"	EuclideanDistance	•	$\sqrt{\sum (x - y)^2}$
"manhattan"	ManhattanDistance	•	$\sum  x - y $
"chebyshev"	ChebyshevDistance	•	$\max( x - y )$
"minkowski"	MinkowskiDistance	p	$\sum  x - y ^p^{1/p}$
"wminkowski"	WMinkowskiDistance	p, w	$\sum  w * (x - y) ^p^{1/p}$
"seuclidean"	SEuclideanDistance	V	$\sqrt{\sum (x - y)^2 / V}$
"mahalanobis"	MahalanobisDistance	V or VI	$\sqrt{(x - y)' V^{-1} (x - y)}$

Έτσι, μιας και ο K-NN δεν απαιτεί εκπαίδευση αποφάσισα να κάνω δοκιμές (στα πλαίσια και της χρονικής παράτασης). Ειδικότερα χρησιμοποίησα την L1 norm αντί της L2.

```
def knn_classifier(k, train_bow_descriptors, test_bow_descriptors,
                  training_class_labels, testing_class_labels,
                  dictionary):
    knn_predictions = []
    for i in range(testing_class_labels.shape[0]):
        Sum = np.zeros(len(dictionary), dtype=int)
        differences = test_bow_descriptors[i] - train_bow_descriptors
        # distances = np.sum(np.square(differences), axis=1) # L2 Distance
        distances = np.sum(np.abs(differences), axis=1) # L1 Distance
        sorted_indices = np.argsort(distances)
        for j in range(k):
            pos = dictionary[int(training_class_labels[sorted_indices[j]])]
            Sum[pos] += 1
        result = np.argmax(Sum, axis=0)
        knn_predictions.append(get_key(dictionary, result))
    return knn_predictions
```

K-(Nearest Neighbors) - L1 Distance	Vocabulary Size = 50	Vocabulary Size = 75	Vocabulary Size = 100
1	41.32154%	49.55793%	52.44300%
3	42.67101%	50.58167%	53.93206%
5	46.25407%	52.62913%	54.11819%
10	48.39460%	55.00233%	58.53886%
15	47.41740%	54.76966%	58.49232%
20	48.44114%	55.56073%	58.49232%
50	48.58074%	53.88553%	57.09632%
75	47.27780%	52.81526%	55.93299%
100	46.39367%	51.60540%	54.49046%



Όπως βλέπουμε η απόδοση του K-NN βελτιώθηκε σε ένα σημαντικό ποσοστό περίπου της τάξης του 10-12%.

Αντίστοιχα, θα μπορούσε να αντικατασταθεί και η απόσταση L2 στην συνάρτηση `create_boww_descriptor()` κατά την δημιουργία του λεξικού. Ωστόσο, η διαδικασία ήταν χρονοβόρα καθώς επίσης αναμένεται ίσως μόνο μια πολύ μικρή αποδοση καθώς εκεί μιλάμε καθαρά για αποστάσεις μεταξύ διανυσμάτων.

Τέλος πιθανή αύξηση της ακρίβειας θα μπορούσε να επιτευχθεί με εφαρμογή της προσέγγισης Spatial Pyramids για την παραγωγή λεξικού και στη συνέχεια ιστογραμμάτων.

## Πηγές - Αναφορές

---

1. Διαφάνιες 9ου Μαθήματος στο E-class: Introduction to visual recognition
2. Διαφάνιες 8ου και 9ου εργαστηρίου: Bag of Visual Words
3. [https://docs.opencv.org/3.4.2/d1/d2d/classcv\\_1\\_1ml\\_1\\_1SVM.html#aad7f1aaccce3c33bb256640910a0e56a66a909b8add6114fde309d24483bcf82](https://docs.opencv.org/3.4.2/d1/d2d/classcv_1_1ml_1_1SVM.html#aad7f1aaccce3c33bb256640910a0e56a66a909b8add6114fde309d24483bcf82)
4. [https://docs.opencv.org/3.4.2/d4/d72/classcv\\_1\\_1BOWKMeansTrainer.html](https://docs.opencv.org/3.4.2/d4/d72/classcv_1_1BOWKMeansTrainer.html)
5. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html#sklearn.neighbors.DistanceMetric>
6. <https://bib.dbvis.de/uploadedFiles/155.pdf> On the Surprising Behavior of Distance Metrics in High Dimensional Space
7. <https://www.di.ens.fr/willow/pdfs/cvpr06b.pdf> Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories
8. <https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf> A Few Useful Things to Know About Machine Learning