Όραση Υπολογιστών

Εργασία 3η

Αλβανάκη Παρασκευή
ΑΜ 57286

Εκφώνηση 3ης Εργασίας

Ζητείται να υλοποιηθεί πρόγραμμα σε Python με τη χρήση της βιβλιοθήκης OpenCV το οποίο θα αφορά στο πρόβλημα της ταξινόμησης πολλαπλών κλάσεων.

Τα ζητούμενα είναι τα εξής :

- Ζητούμενο 1ο:

  Παραγωγή οπτικού λεξικού (visual vocabulary) βασισμένη στο μοντέλο Bag of Visual Words (BOVW). Η δημιουργία του λεξικού να γίνει με τη χρήση του αλγορίθμου K-Means χρησιμοποιώντας όλες τις εικόνες του συνόλου εκπαίδευσης .

- Ζητούμενο 2ο:

  Εξαγωγή περιγραφέα σε κάθε εικόνα εκπαίδευσης (imagedb_train) με βάση το μοντέλο BOVW χρησιμοποιώντας το λεξικό που προέκυψε κατά το βήμα 1.Για το βήμα αυτό δεν μπορείτε να χρησιμοποιήσετε τη σχετική κλάση της OpenCV cv.BOWImgDescriptorExtractor

- Ζητούμενο 3ο:

  Με βάση το αποτέλεσμα του βήματος 2, να υλοποιηθεί η λειτουργία ταξινόμησης μιας εικόνας κάνοντας χρήση των δυο παρακάτω ταξινομητών : α. Του αλγορίθμου k-NN χωρίς τη χρήση της σχετικής OpenCV συνάρτησης (cv.ml.KNearest_create()). β. Του σχήματος one-versus-all όπου για κάθε κλάση εκπαιδεύεται ένας SVM ταξινομητής.

- Ζητούμενο 4ο:

  Αξιολόγηση του συστήματος: Χρησιμοποιώντας το σύνολο δοκιμής (imagedb_test), να μετρηθεί η ακρίβεια του συστήματος (και στις δύο περιπτώσεις ταξινομητών) που εκφράζεται ως το ποσοστό των επιτυχών ταξινομήσεων. Κατά την αξιολόγηση να ελέγξετε την επίδραση των εμπλεκομένων παραμέτρων, όπως ο αριθμός των οπτικών λέξεων (Βήμα 1), ο αριθμός των πλησιέστερων γειτόνων (Βήμα 3α) και ο τύπος του πυρήνα (kernel) του SVM (Βήμα 3β).

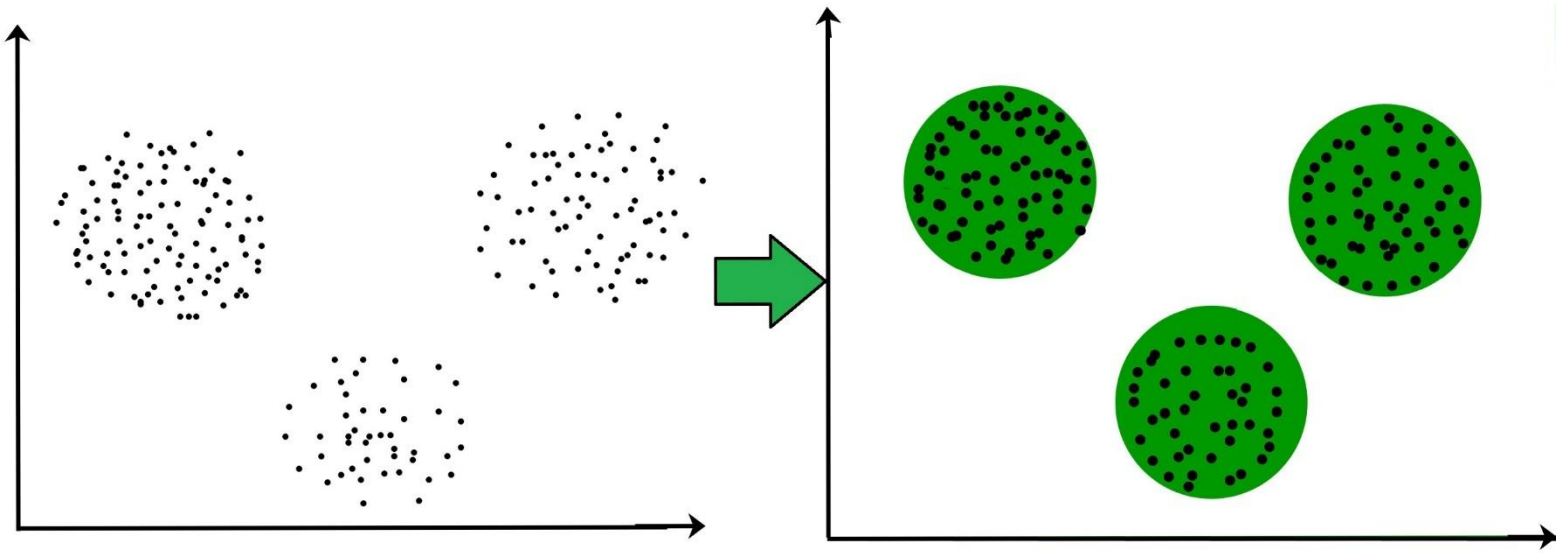
Θεωρητικό υπόβαθρο

➢ Unsupervised learning method

Μία μέθοδος μάθησης χωρίς επίβλεψη είναι μια μέθοδος στην οποία αντλούμε αναφορές από σύνολα δεδομένων που αποτελούνται από δεδομένα εισόδου χωρίς επισημασμένες απαντήσεις(labels). Είναι μία από τις τρεις βασικές κατηγορίες της μάθησης μηχανών, μαζί με την supervised και την reinforcement learning.

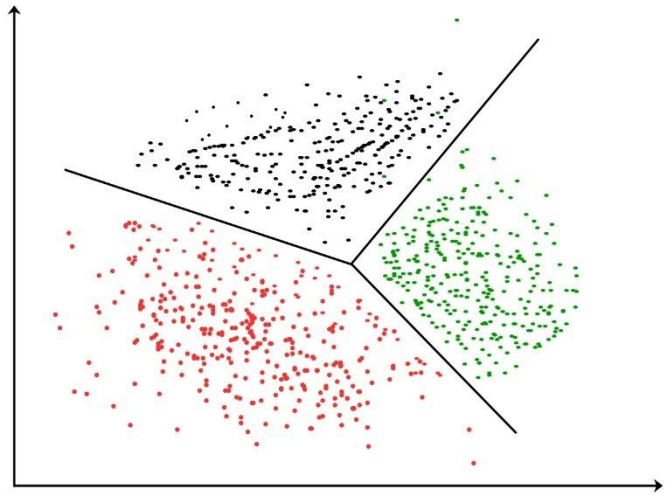
➢ Clustering

Η ομαδοποίηση είναι η διαδικασία της διαίρεσης των σημείων δεδομένων σε διάφορες ομάδες έτσι ώστε τα σημεία δεδομένων στις ίδιες ομάδες να είναι περισσότερο παρόμοια με άλλα σημεία δεδομένων στην ίδια ομάδα και διαφορετικά από τα σημεία δεδομένων σε άλλες ομάδες. Είναι βασικά μια συλλογή αντικειμένων βάσει της ομοιότητας και της ανομοιότητας μεταξύ τους.



➢ K-means Algorithm

Αλγόριθμος ομαδοποίησης K-means είναι ο απλούστερος αλγόριθμος μάθησης χωρίς επίβλεψη ο οποίος επιλύει το clustering probem. Ανήκει στην κατηγορία της επίπεδης συσταδοποίησης διότι παράγει ένα σύνολο συσταδοποιήσεων οι οποίες δεν έχουν κάποια ιδιαίτερη δομή-σχέση μεταξύ τους. Ο αλγόριθμος έχει ως στόχο τη βελτιστοποιήση μίας συνάρτησης – της συνάρτησης κόστους. ο κάθε σημείο ανατίθεται στο cluster με το κοντινότερο κεντρικό σημείο σε σχέση με το σημείο. Ο αριθμός των ομάδων, K, είναι είσοδος στον αλγόριθμο. Το κεντρικό σημείο είναι (συνήθως) το μέσο (mean) των σημείων της συστάδας (το οποίο μπορεί να μην είναι ένα από τα δεδομένα εισόδου).



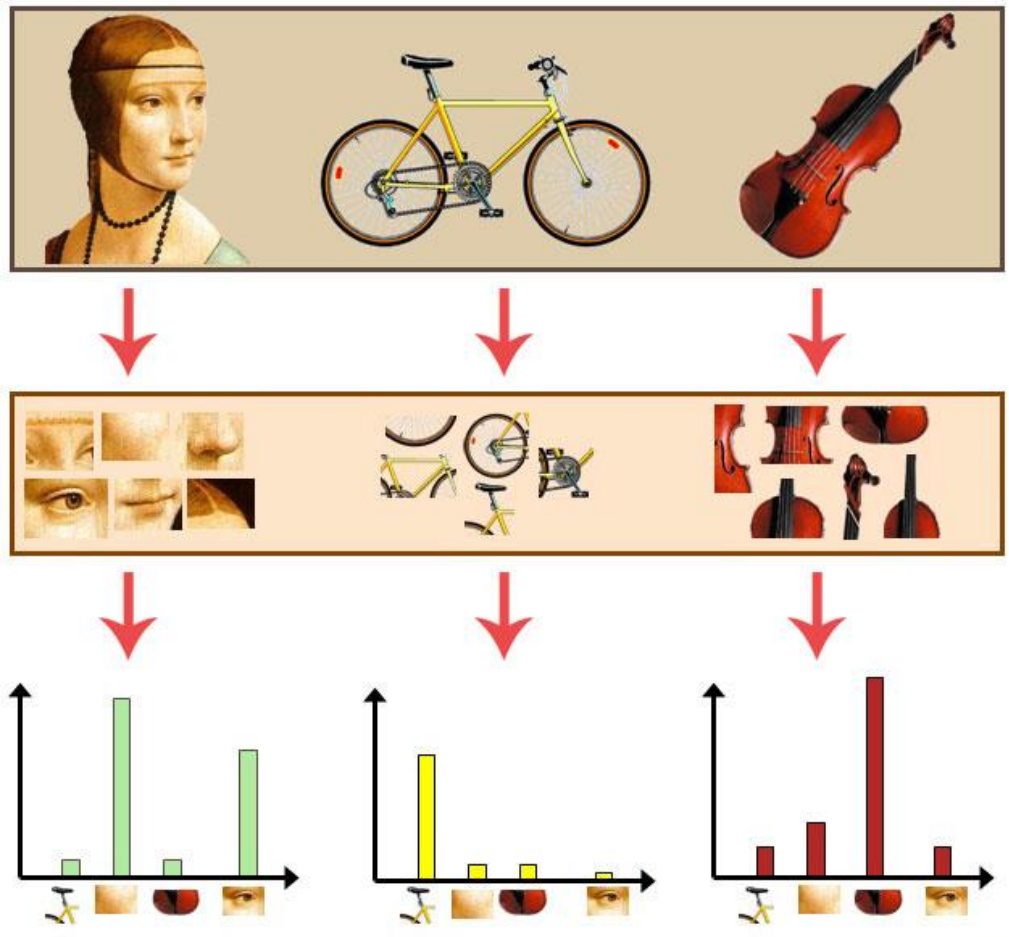
- **Ο αλγόριθμος λειτουργεί ως εξής:**

1. Αρχικά αρχικοποιούμε τα σημεία k, που ονομάζονται μέσα(means) δηλαδή τα κέντρα .
2. Κατηγοριοποιούμε κάθε στοιχείο στο πλησιέστερο μέσο με βάση την απόσταση του από αυτό και ενημερώνουμε τις συντεταγμένες του μέσου, οι οποίες είναι οι μέσοι όροι των αποστάσεων αντικειμένων που ταξινομούνται στο μέσο μέχρι στιγμής.
3. Επαναλαμβάνουμε τη διαδικασία για ένα δεδομένο αριθμό επαναλήψεων και στο τέλος έχουμε τις ομάδες μας(clusters).

Ο αλγόριθμος ολοκληρώνεται όταν οι ενημερώσεις που γίνονται σε κάθε mi είναι αμελητέες. Σημαντικό σημείο του αλγορίθμου είναι η αρχικοποίηση των k-διανυσμάτων.

➢ Bag of Visual Words (BOVW)

Το Bag of Visual Words (BOVW) χρησιμοποιείται συνήθως στην ταξινόμηση εικόνων. Το μοντέλο αυτό έχει ως στόχο να κωδικοποιήσει το σύνολο των τοπικών χαρακτηριστικών που εντοπίστηκαν σε μια εικόνα, σε μια καθολική αναπαράσταση της εικόνας. Οι κωδικοποιημένες αναπαραστάσεις μπορούν να χρησιμοποιηθούν για να υπολογίσουμε πόσο μοιάζουν δύο εικόνες, με βάση το περιεχόμενό τους.



○ Building a bag of visual words:

1. Εξαγωγή των τοπικών χαρακτηριστικών σε κάθε εικόνα του training set
2. Δημιουργία λέξεων (συνήθως με k-means)
3. Αντιστοίχιση των keypoints κάθε εικόνας σε μια λεξη
4. Δημιουργία ιστογράματος σε κάθε εικόνα σύμφωνα με τη συχνότητα εμφάνισης των λέξεων μεσα στην εικόνα

- **Κωδικοποίηση μίας εικόνας:**

1. Εξαγωγή των τοπικών χαρακτηριστικών
2. Αντιστοίχιση κάθε τοπικού χαρακτηριστικού με την κοντινότερη λέξη του λεξικο.
3. Δημιουργία ιστογράμματος με τη συχνότητα εμφάνισης κάθε λέξης μέσα στην εικόνα.

Στόχος του BOVW είναι να αντιπροσωπεύει μια εικόνα ως ένα σύνολο χαρακτηριστικών (image features). Τα χαρακτηριστικά γνωρίσματα αποτελούνται από σημεία-κλειδιά και περιγραφείς (descriptors). Χρησιμοποιούμε τα σημεία κλειδιά και τα περιγραφικά στοιχεία για να κατασκευάσουμε λεξιλόγια/λεξικά (vocabularies) και να αναπαριστούμε κάθε εικόνα ως ιστογράμματα συχνοτήτων των χαρακτηριστικών που βρίσκονται στην εικόνα.

➢ Multiclass Classification

Στη μηχανική μάθηση, η ταξινόμηση σε πολυάριθμες κατηγορίες είναι το πρόβλημα της ταξινόμησης των περιπτώσεων σε μία από τις τρεις ή περισσότερες κατηγορίες. (Η ταξινόμηση των περιπτώσεων σε μία από τις δύο κατηγορίες ονομάζεται δυαδική ταξινόμηση.)
Ενώ μερικοί αλγόριθμοι ταξινόμησης επιτρέπουν φυσικά τη χρήση περισσότερων από δύο τάξεις, άλλοι είναι από τη φύση τους δυαδικοί αλγόριθμοι. αυτά μπορούν, ωστόσο, να μετατραπούν σε multiclass ταξινομητές με ποικίλες στρατηγικές. Δύο multiclass classifications που θα αναλυθούν παρακάτω είναι οι k-nearest neighbours (K-nn) και η Support Vector Machines (SVM).

➢ K-nn

Ο αλγόριθμος k-nearest neighbors (k-NN) είναι μια μη παραμετρική μέθοδος που χρησιμοποιείται για ταξινόμηση. Η είσοδος αποτελείται από τα k πλησιέστερα παραδείγματα εκπαίδευσης στο χώρο των χαρακτηριστικών. Ένα αντικείμενο ταξινομείται από μια πλειονότητα των γειτόνων του, με το αντικείμενο να ανατίθεται στην τάξη που είναι περισσότερο κοινή μεταξύ των πλησιέστερων γειτόνων του (k είναι ένας θετικός ακέραιος, συνήθως μικρός). Αν k = 1, τότε το αντικείμενο απλώς αποδίδεται στην κλάση εκείνου του μοναδικού πλησιέστερου γείτονα. Μια χρήσιμη τεχνική μπορεί να χρησιμοποιηθεί για να αποδώσει βάρος στις συνεισφορές των γειτόνων, έτσι ώστε οι πλησιέστεροι γείτονες να συνεισφέρουν περισσότερο στο μέσο όρο από ό, τι οι πιο μακρινοί. Για παράδειγμα, ένα κοινό σχήμα βαρύτητας συνίσταται στο να δώσουμε σε κάθε γείτονα ένα βάρος 1 / d, όπου d είναι η απόσταση από τον γείτονα.

➢ SVM

Στη μηχανική μάθηση, τα SVM είναι υπό εποπτεία μαθησιακά μοντέλα με συναφείς αλγόριθμους εκμάθησης που αναλύουν δεδομένα που χρησιμοποιούνται για ανάλυση ταξινόμησης και παλινδρόμησης. Δεδομένου ότι ένα σύνολο εκπαιδευτικών παραδειγμάτων, κάθε ένα από τα οποία χαρακτηρίζεται ότι ανήκουν σε μία ή την άλλη από τις δύο κατηγορίες, ένας αλγόριθμος κατάρτισης SVM δημιουργεί ένα μοντέλο που εκχωρεί νέα παραδείγματα σε μία ή την άλλη κατηγορία, καθιστώντας τον έναν μη πιθανοτικό δυαδικό γραμμικό ταξινομητή . Ένα μοντέλο SVM είναι μια αναπαράσταση των παραδειγμάτων ως σημεία στο διάστημα, χαρτογραφημένα έτσι ώστε τα παραδείγματα των ξεχωριστών κατηγοριών χωρίζονται από ένα σαφές κενό που είναι όσο το δυνατόν ευρύτερο. Στη συνέχεια, νέα παραδείγματα χαρτογραφούνται στον ίδιο χώρο και προβλέπεται να ανήκουν σε μια κατηγορία που βασίζεται στην πλευρά του χάσματος επί του οποίου πέφτουν.

➢ One vs All

Η στρατηγική One-vs-all περιλαμβάνει την εκπαίδευση ενός μόνο ταξινομητή ανά τάξη, με τα δείγματα αυτής της κλάσης ως θετικά δείγματα και όλα τα άλλα δείγματα ως αρνητικά. Αυτή η στρατηγική απαιτεί από τους βασικούς ταξινομητές να παράγουν μια πραγματική βαθμολογία εμπιστοσύνης για την απόφασή τους, και όχι μόνο μια ετικέτα κλάσης. Οι διακριτές ετικέτες κλάσης από μόνες τους μπορούν να οδηγήσουν σε ασάφειες, όπου προβλέπονται πολλαπλές τάξεις για ένα μόνο δείγμα.

Υλοποίηση Εργασίας

1. Παραγωγή οπτικού λεξικού (visual vocabulary) βασισμένη στο μοντέλο Bag of Visual Words (BOVW). Η δημιουργία του λεξικού να γίνει με τη χρήση του αλγορίθμου K-Means χρησιμοποιώντας όλες τις εικόνες του συνόλου εκπαίδευσης (imagedb_train).

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

train_folders = ['imagedb_train']
sift = cv.xfeatures2d_SIFT.create()
def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img) #find the keypoints
    desc = sift.compute(img, kp)#calculate descriptors for every keypoint
    desc = desc[1]
    return desc
# Extract Database
print('Extracting features...')
train_descs = np.zeros((0, 128))
for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            path = os.path.join(folder, file,image)
            desc = extract_local_features(path)
            if desc is None:
                continue
            train_descs = np.concatenate((train_descs, desc), axis=0) #all the
keypoints of all the images in the train folder
# Create vocabulary
print('Creating vocabulary...')
term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
trainer = cv.BOWKMeansTrainer(50, term_crit, 1, cv.KMEANS_PP_CENTERS) # 50
clusters,termination criteria,Use kmeans++ center initialization by Arthur and
Vassilvitskii
vocabulary = trainer.cluster(train_descs.astype(np.float32))#make the numbers floats
and train the clusters
np.save('vocabulary.npy', vocabulary)

```

2. Εξαγωγή περιγραφέα σε κάθε εικόνα εκπαίδευσης (imagedb_train) με βάση το μοντέλο BOVW χρησιμοποιώντας το λεξικό που προέκυψε κατά το βήμα 1. Για το βήμα αυτό δεν μπορείτε να χρησιμοποιήσετε τη σχετική κλάση της OpenCV (cv.BOWImgDescriptorExtractor)

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

train_folders = ['imagedb_train']
sift = cv.xfeatures2d_SIFT.create()

def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img) #find the keypoints
    desc = sift.compute(img, kp)#calculate descriptors for every keypoint
    desc = desc[1]
    return desc
def match(image, myvocabulary):
    n1 = image.shape[0]
    matches = np.zeros((1, myvocabulary.shape[0]))
    for m in range(n1):
        fv = image[m, :]  # for every keypoint with 127 vectors)
        diff = (fv - myvocabulary) ** 2  # find the difference in distance between clusters
and keypoints squared
        distances = np.sum(diff, axis=1)  # add the differences for every vector of each
keypoint with each cluster
        i2 = np.argmin(distances)  # find which cluster is the closest one to the keypoint
        matches[0, i2] += 1 # our keypoint belongs to the closest cluster so increase the
amount of times this cluster is seen in the image

    return matches #array with the frequencies of the appearences of each cluster in an
image

# Extract Database
print('Extracting features...')
train_descs = np.zeros((0, 128))
for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            path = os.path.join(folder, file,image)
            desc = extract_local_features(path)
            if desc is None:
                continue
            train_descs = np.concatenate((train_descs, desc), axis=0) #all the keypoints of
all the images in the train folder


# Create vocabulary
print('Creating vocabulary...')
term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
trainer = cv.BOWKMeansTrainer(50, term_crit, 1, cv.KMEANS_PP_CENTERS) # 50
clusters,termination criteria,Use kmeans++ center initialization by Arthur and
Vassilvitskii
vocabulary = trainer.cluster(train_descs.astype(np.float32))#make the numbers floats anf
```

```
train the clusters
np.save('vocabulary.npy', vocabulary)

# Load vocabulary
vocabulary = np.load('vocabulary.npy')
# Create Index
print('Creating index...')
img_paths = []
train_descs = np.zeros((0, 128))
bow_descs = np.zeros((0, vocabulary.shape[0])) #number of clusters

for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            path = os.path.join(folder, file,image)
            desc = extract_local_features(path)

            if desc is None:
                continue
            bow_desc=match(desc,vocabulary)
            img_paths.append(path)
            bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)#array with all the
frequencies of each cluster in all the photos
np.save('index.npy', bow_descs)
with open('index_paths.txt', mode='w+') as file:
    json.dump(img_paths, file)

# Load Index
bow_descs = np.load('index.npy')
with open('index_paths.txt', mode='r') as file:
    img_paths = json.load(file)
```

3. Με βάση το αποτέλεσμα του βήματος 2, να υλοποιηθεί η λειτουργία ταξινόμησης μιας εικόνας κάνοντας χρήση των δυο παρακάτω ταξινομητών : α. Του αλγορίθμου k-NN χωρίς τη χρήση της σχετικής OpenCV συνάρτησης (cv.ml.KNearest_create()). β. Του σχήματος one-versus-all όπου για κάθε κλάση εκπαιδεύεται ένας SVM ταξινομητής.

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

```
train_folders = ['imagedb_train']
sift = cv.xfeatures2d_SIFT.create()


def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img) #find the keypoints
    desc = sift.compute(img, kp)#calculate descriptors for every keypoint
    desc = desc[1]
    return desc


def create_svm(labels, bow_descs):
    svm = cv.ml.SVM_create()
    svm.setType(cv.ml.SVM_C_SVC) #uses C as the tradeoff parameter between the size of
margin and the number of training points which are misclassified
    # svm.setKernel(cv.ml.SVM_LINEAR)
    # svm.setKernel(cv.ml.SVM_CHI2)
    # svm.setKernel(cv.ml.SVM_INTER)
    # svm.setKernel(cv.ml.SVM_POLY)
    # svm.setKernel(cv.ml.SVM_SIGMOID)

    svm.setKernel(cv.ml.SVM_RBF)
    svm.setTermCriteria((cv.TERM_CRITERIA_COUNT, 40, 1.e-06))#termination criteria
    svm.trainAuto(bow_descs.astype(np.float32), cv.ml.ROW_SAMPLE, labels)
    svm.save('svm')
    return svm


def match(image, myvocabulary):
    n1 = image.shape[0]
    matches = np.zeros((1, myvocabulary.shape[0]))
    for m in range(n1):
        fv = image[m, :]  # for every keypoint with 127 vectors)
        diff = (fv - myvocabulary) ** 2  # find the difference in distance between
clusters and keypoints squared
        distances = np.sum(diff, axis=1)  # add the differences for every vector of
each keypoint with each cluster
        i2 = np.argmin(distances)  # find which cluster is the closest one to the
keypoint
        matches[0, i2] += 1 # our keypoint belongs to the closest cluster so increase
the amount of times this cluster is seen in the image

    return matches #array with the frequencies of the appearences of each cluster in
an image


def fncorrect(max):

    correct = 0  # initialize a valiable to help us with the success rate
    if max == 0:
        print('It is a fighter-jet')
        name = "fighter-jet"
        if name in path:
            correct = correct + 1
    elif max == 1:
        print('It is a motorbike')
        name = "fire-truck"
        if name in path:
            correct = correct + 1
    elif max == 2:
```

```
            print('It is a school-bus')
            name = "school-bus"
            if name in path:
                correct = correct + 1
        elif max == 3:
            print('It is a touring-bike')
            name = "touring-bike"
            if name in path:
                correct = correct + 1
        elif max == 4:
            print('It is an airplane')
            name = "airplane"
            if name in path:
                correct = correct + 1
        elif max == 5:
            print('It is a car-side')
            name = "car-side"
            if name in path:
                correct = correct + 1

        return correct


    # # Extract Database
    # print('Extracting features...')
    # train_descs = np.zeros((0, 128))
    # for folder in train_folders:
    #     files = os.listdir(folder)
    #     for file in files:
    #         path1 = os.path.join(folder, file)
    #         images = os.listdir(path1)
    #         for image in images:
    #             path = os.path.join(folder, file,image)
    #             desc = extract_local_features(path)
    #             if desc is None:
    #                 continue
    #             train_descs = np.concatenate((train_descs, desc), axis=0) #all the
    keypoints of all the images in the train folder


    # # Create vocabulary
    # print('Creating vocabulary...')
    # term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
    # trainer = cv.BOWKMeansTrainer(50, term_crit, 1, cv.KMEANS_PP_CENTERS) # 50
    clusters,termination criteria,Use kmeans++ center initialization by Arthur and
    Vassilvitskii
    # vocabulary = trainer.cluster(train_descs.astype(np.float32))#make the numbers floats
    anf train the clusters
    # np.save('vocabulary.npy', vocabulary)

    # Load vocabulary
    vocabulary = np.load('vocabulary.npy')
    # # Create Index
    # print('Creating index...')
    # img_paths = []
    # train_descs = np.zeros((0, 128))
    # bow_descs = np.zeros((0, vocabulary.shape[0])) #number of clusters
    #
    # for folder in train_folders:
    #     files = os.listdir(folder)
    #     for file in files:
```

```
#         path1 = os.path.join(folder, file)
#         images = os.listdir(path1)
#         for image in images:
#             path = os.path.join(folder, file,image)
#             desc = extract_local_features(path)
#
#             if desc is None:
#                 continue
#             bow_desc=match(desc,vocabulary)#array that has for every photo which
keypoint goes to each cluster
#             img_paths.append(path)
#             bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)
# np.save('index.npy', bow_descs)
# with open('index_paths.txt', mode='w+') as file:
#     json.dump(img_paths, file)

# Load Index
bow_descs = np.load('index.npy')
with open('index_paths.txt', mode='r') as file:
    img_paths = json.load(file)

item1 = 0
test_folders = ['imagedb_test']
correct=0
# ========= CLASSIFICATION k_NN ==========
# Διατρέχω την κάθε εικόνα
for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            item1 = item1 + 1
            path = os.path.join(folder, file, image)
            desc = extract_local_features(path)  # Δημιουργία descriptors
            bow_desc = match(desc, vocabulary)
            K = 6
            distances = np.sum((bow_desc - bow_descs) ** 2, axis=1)
            sorted_ids = np.argsort(distances)
            sum_fighter_jet = 0
            sum_motorbike = 0
            sum_school_bus = 0
            sum_touring_bike = 0
            sum_airplanes = 0
            sum_car_side = 0
            for i in range(K):
                if 'fighter-jet' in img_paths[sorted_ids[i]]:
                    sum_fighter_jet += 1 / (distances[sorted_ids[i]] + 1)
                elif 'motorbikes' in img_paths[sorted_ids[i]]:
                    sum_motorbike += 1 / (distances[sorted_ids[i]] + 1)
                elif 'school-bus' in img_paths[sorted_ids[i]]:
                    sum_school_bus += 1 / (distances[sorted_ids[i]] + 1)
                elif 'touring-bike' in img_paths[sorted_ids[i]]:
                    sum_touring_bike += 1 / (distances[sorted_ids[i]] + 1)
                elif 'airplanes' in img_paths[sorted_ids[i]]:
                    sum_airplanes += 1 / (distances[sorted_ids[i]] + 1)
                elif 'car-side' in img_paths[sorted_ids[i]]:
                    sum_car_side += 1 / (distances[sorted_ids[i]] + 1)
            sums = [sum_fighter_jet, sum_motorbike, sum_school_bus, sum_touring_bike,
sum_airplanes, sum_car_side]
            # print (sums)
```

```
            max1 = np.argmax(sums)  # Επιστρέφει τους δείκτες των μέγιστων τιμών κατά
#μήκος ενός άξονα
            correct = fncorrect(max1) + correct
print("")
print((correct / item1) * 100, "%\n")
pass
# ======================================================= 1 vs all  svm
=================================================


correct_1_vs_all = 0


labels0 = np.array(['fighter-jet' in a for a in img_paths], np.int32)
labels1 = np.array(['motorbikes' in a for a in img_paths], np.int32)
labels2 = np.array(['school-bus' in a for a in img_paths], np.int32)
labels3 = np.array(['touring-bike' in a for a in img_paths], np.int32)
labels4 = np.array(['airplanes' in a for a in img_paths], np.int32)
labels5 = np.array(['car-side' in a for a in img_paths], np.int32)

svm0 = create_svm(labels0, bow_descs)
svm1 = create_svm(labels1, bow_descs)
svm2 = create_svm(labels2, bow_descs)
svm3 = create_svm(labels3, bow_descs)
svm4 = create_svm(labels4, bow_descs)
svm5 = create_svm(labels5, bow_descs)
item2 = 0
for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            item2 = item2 + 1
            path = os.path.join(folder, file, image)
            desc = extract_local_features(path)
            bow_desc = match(desc, vocabulary)
            responses_1_vs_all = np.zeros(6)
            responses_1_vs_all[0] = svm0.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            responses_1_vs_all[1] = svm1.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            responses_1_vs_all[2] = svm2.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            responses_1_vs_all[3] = svm3.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            responses_1_vs_all[4] = svm4.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            responses_1_vs_all[5] = svm5.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
            final_response_1_vs_all = np.argmin(responses_1_vs_all)
            correct_1_vs_all += fncorrect(final_response_1_vs_all)

        print('Success rate of 1 vs all: ' + str((correct_1_vs_all / item2) * 100) +
'%')

        pass
```

Με K-nn απόδοση 84.41330998248687 % και SVM απόδοση 58.66900175131349%

4.Αξιολόγηση του συστήματος: Χρησιμοποιώντας το σύνολο δοκιμής (imagedb_test), να μετρηθεί η ακρίβεια του συστήματος (και στις δύο περιπτώσεις ταξινομητών) που εκφράζεται ως το ποσοστό των επιτυχών ταξινομήσεων. Κατά την αξιολόγηση να ελέγξετε την επίδραση των εμπλεκομένων παραμέτρων, όπως ο αριθμός των οπτικών λέξεων (Βήμα 1), ο αριθμός των πλησιέστερων γειτόνων (Βήμα 3α) και ο τύπος του πυρήνα (kernel) του SVM (Βήμα 3β).

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

train_folders = ['imagedb_train']
sift = cv.xfeatures2d_SIFT.create()


def extract_local_features(path):
    img = cv.imread(path)
    kp = sift.detect(img) #find the keypoints
    desc = sift.compute(img, kp)#calculate descriptors for every keypoint
    desc = desc[1]
    return desc


def create_svm(labels, bow_descs,kernel):
    svm = cv.ml.SVM_create()
    svm.setType(cv.ml.SVM_C_SVC) #uses C as the tradeoff parameter between the size of
margin and the number of training points which are misclassified
    # svm.setKernel(cv.ml.SVM_LINEAR)
    # svm.setKernel(cv.ml.SVM_CHI2)
    # svm.setKernel(cv.ml.SVM_INTER)
    # svm.setKernel(cv.ml.SVM_POLY)
    # svm.setKernel(cv.ml.SVM_SIGMOID)

    svm.setKernel(kernel)
    svm.setTermCriteria((cv.TERM_CRITERIA_COUNT, 40, 1.e-06))#termination criteria
    svm.trainAuto(bow_descs.astype(np.float32), cv.ml.ROW_SAMPLE, labels)
    svm.save('svm')
    return svm


def match(image, myvocabulary):
    n1 = image.shape[0]
    matches = np.zeros((1, myvocabulary.shape[0]))
    for m in range(n1):
        fv = image[m, :]  # for every keypoint with 127 vectors)
        diff = (fv - myvocabulary) ** 2  # find the difference in distance between clusters
and keypoints squared
        distances = np.sum(diff, axis=1)  # add the differences for every vector of each
keypoint with each cluster
        i2 = np.argmin(distances)  # find which cluster is the closest one to the keypoint
        matches[0, i2] += 1 # our keypoint belongs to the closest cluster so increase the
amount of times this cluster is seen in the image

    return matches #array with the frequencies of the appearences of each cluster in an
image
```

```
def fncorrect(max):

    correct = 0  # initialize a valiable to help us with the success rate
    if max == 0:
        # print('It is a fighter-jet')
        name = "fighter-jet"
        if name in path:
            correct = correct + 1
    elif max == 1:
        # print('It is a motorbike')
        name = "fire-truck"
        if name in path:
            correct = correct + 1
    elif max == 2:
        # print('It is a school-bus')
        name = "school-bus"
        if name in path:
            correct = correct + 1
    elif max == 3:
        # print('It is a touring-bike')
        name = "touring-bike"
        if name in path:
            correct = correct + 1
    elif max == 4:
        # print('It is an airplane')
        name = "airplane"
        if name in path:
            correct = correct + 1
    elif max == 5:
        # print('It is a car-side')
        name = "car-side"
        if name in path:
            correct = correct + 1

    return correct


# Extract Database
print('Extracting features...')
train_descs = np.zeros((0, 128))
for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            path = os.path.join(folder, file, image)
            desc = extract_local_features(path)
            if desc is None:
                continue
            train_descs = np.concatenate((train_descs, desc),
                                         axis=0)  # all the keypoints of all the images in
the train folder

# Create vocabulary
print('Creating vocabulary...')
term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
trainer = cv.BOWKMeansTrainer(50, term_crit, 1,
                              cv.KMEANS_PP_CENTERS)  # 50 clusters,termination
criteria,Use kmeans++ center initialization by Arthur and Vassilvitskii
vocabulary = trainer.cluster(train_descs.astype(np.float32))  # make the numbers floats anf
```

```
train the clusters
np.save('vocabulary.npy', vocabulary)
# Load vocabulary
vocabulary = np.load('vocabulary.npy')
# Create Index
print('Creating index...')
img_paths = []
train_descs = np.zeros((0, 128))
bow_descs = np.zeros((0, vocabulary.shape[0]))  # number of clusters

for folder in train_folders:
    files = os.listdir(folder)
    for file in files:
        path1 = os.path.join(folder, file)
        images = os.listdir(path1)
        for image in images:
            path = os.path.join(folder, file, image)
            desc = extract_local_features(path)

            if desc is None:
                continue
            bow_desc = match(desc, vocabulary)  # array that has for every photo which
keypoint goes to each cluster
            img_paths.append(path)
            bow_descs = np.concatenate((bow_descs, bow_desc), axis=0)
np.save('index.npy', bow_descs)
with open('index_paths.txt', mode='w+') as file:
    json.dump(img_paths, file)

# Load Index
bow_descs = np.load('index.npy')
with open('index_paths.txt', mode='r') as file:
    img_paths = json.load(file)



test_folders = ['imagedb_test']

numberofx=[]
differentK=[]
#========= CLASSIFICATION k_NN ==========
    # Διατρέχω την κάθε εικόνα
for x in range(2, 30, 2):
    item1 = 0
    correct = 0
    for folder in test_folders:
        files = os.listdir(folder)
        for file in files:
            path1 = os.path.join(folder, file)
            images = os.listdir(path1)
            for image in images:
                item1 = item1 + 1
                path = os.path.join(folder, file, image)
                desc = extract_local_features(path)
                bow_desc = match(desc, vocabulary)
                K = x  # how many neighbours to take into consideration
                distances = np.sum((bow_desc - bow_descs) ** 2, axis=1)
                sorted_ids = np.argsort(distances)  # clusters sorted from closest to
farthest
                sum_fighter_jet = 0
                sum_motorbike = 0
```

```
                sum_school_bus = 0
                sum_touring_bike = 0
                sum_airplanes = 0
                sum_car_side = 0
                for i in range(K):
                    if 'fighter-jet' in img_paths[sorted_ids[i]]:
                        sum_fighter_jet += 1 / (distances[sorted_ids[i]] + 1)
                    elif 'motorbikes' in img_paths[sorted_ids[i]]:
                        sum_motorbike += 1 / (distances[sorted_ids[i]] + 1)
                    elif 'school-bus' in img_paths[sorted_ids[i]]:
                        sum_school_bus += 1 / (distances[sorted_ids[i]] + 1)
                    elif 'touring-bike' in img_paths[sorted_ids[i]]:
                        sum_touring_bike += 1 / (distances[sorted_ids[i]] + 1)
                    elif 'airplanes' in img_paths[sorted_ids[i]]:
                        sum_airplanes += 1 / (distances[sorted_ids[i]] + 1)
                    elif 'car-side' in img_paths[sorted_ids[i]]:
                        sum_car_side += 1 / (distances[sorted_ids[i]] + 1)
                sums = [sum_fighter_jet, sum_motorbike, sum_school_bus, sum_touring_bike,
sum_airplanes,
                        sum_car_side]

                max1 = np.argmax(sums)  # maximum
                correct = fncorrect(max1) + correct
        # print("")
        # print((correct / item1) * 100, "%\n")
    numberofx.append(x)
    differentK.append((correct / item1) * 100)
print(numberofx)
print(differentK)
pass
# ======================================================= 1 vs all  svm
=====================================================

correct_1_vs_all = 0
labels0 = np.array(['fighter-jet' in a for a in img_paths], np.int32)
labels1 = np.array(['motorbikes' in a for a in img_paths], np.int32)
labels2 = np.array(['school-bus' in a for a in img_paths], np.int32)
labels3 = np.array(['touring-bike' in a for a in img_paths], np.int32)
labels4 = np.array(['airplanes' in a for a in img_paths], np.int32)
labels5 = np.array(['car-side' in a for a in img_paths], np.int32)
i=0
svm_kernel = [cv.ml.SVM_LINEAR, cv.ml.SVM_CHI2, cv.ml.SVM_INTER, cv.ml.SVM_SIGMOID,
cv.ml.SVM_RBF]
svm_kernel_name = ['SVM_LINEAR', 'SVM_CHI2', 'SVM_INTER', 'SVM_SIGMOID', 'SVM_RBF']
for kernel_type in svm_kernel:

    item2 = 0
    correct_1_vs_all = 0
    svm0 = create_svm(labels0, bow_descs, kernel_type)
    svm1 = create_svm(labels1, bow_descs, kernel_type)
    svm2 = create_svm(labels2, bow_descs, kernel_type)
    svm3 = create_svm(labels3, bow_descs, kernel_type)
    svm4 = create_svm(labels4, bow_descs, kernel_type)
    svm5 = create_svm(labels5, bow_descs, kernel_type)
    for folder in test_folders:
        files = os.listdir(folder)
        for file in files:
            path1 = os.path.join(folder, file)
            images = os.listdir(path1)
            for image in images:
                item2 = item2 + 1  # number of images
```

```
                path = os.path.join(folder, file, image)
                desc = extract_local_features(path)
                bow_desc = match(desc, vocabulary)
                responses_1_vs_all = np.zeros(6)
                responses_1_vs_all[0] = \
                    svm0.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
                responses_1_vs_all[1] = \
                    svm1.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
                responses_1_vs_all[2] = \
                    svm2.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
                responses_1_vs_all[3] = \
                    svm3.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
                responses_1_vs_all[4] = \
                    svm4.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
                responses_1_vs_all[5] = \
                    svm5.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
                final_response_1_vs_all = np.argmin(responses_1_vs_all)
                correct_1_vs_all += fncorrect(final_response_1_vs_all)

    print('Success rate of 1 vs all: ' + str((correct_1_vs_all / item2) * 100) + '%' +
'with kernel type ' +
          svm_kernel_name[i])
    i = i + 1

pass
```

Για 20 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

[51.61290322580645, 53.2258064516129, 53.2258064516129, 54.83870967741935, 51.61290322580645, 51.61290322580645, 53.2258064516129, 53.2258064516129, 50.0, 51.61290322580645, 53.2258064516129, 53.2258064516129, 51.61290322580645, 50.0]

Success rate of 1 vs all: 20.967741935483872%with kernel type SVM_LINEAR

Success rate of 1 vs all: 41.935483870967744%with kernel type SVM_CHI2

Success rate of 1 vs all: 19.35483870967742%with kernel type SVM_INTER

Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID

Success rate of 1 vs all: 50.0%with kernel type SVM_RBF

Για 30 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

[54.83870967741935, 56.451612903225815, 54.83870967741935, 56.451612903225815, 58.06451612903226, 58.06451612903226, 54.83870967741935, 54.83870967741935, 54.83870967741935, 54.83870967741935, 54.83870967741935, 56.451612903225815, 56.451612903225815, 54.83870967741935]

Success rate of 1 vs all: 1.6129032258064515%with kernel type SVM_LINEAR

Success rate of 1 vs all: 51.61290322580645%with kernel type SVM_CHI2

Success rate of 1 vs all: 27.419354838709676%with kernel type SVM_INTER

Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID

Success rate of 1 vs all: 50.0%with kernel type SVM_RBF

Για 40 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

[54.83870967741935, 58.06451612903226, 56.451612903225815, 58.06451612903226, 56.451612903225815, 56.451612903225815, 56.451612903225815, 54.83870967741935, 53.2258064516129, 53.2258064516129, 50.0, 51.61290322580645, 53.2258064516129, 51.61290322580645]

Success rate of 1 vs all: 3.225806451612903%with kernel type SVM_LINEAR

Success rate of 1 vs all: 43.54838709677419%with kernel type SVM_CHI2

Success rate of 1 vs all: 32.25806451612903%with kernel type SVM_INTER

Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID

Success rate of 1 vs all: 48.38709677419355%with kernel type SVM_RBF

Για 50 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

[50.0, 56.4516, 53.2258, 54.8387, 53.2258, 54.8387, 54.8387, 51.61290, 54.8387, 53.2258, 54.8387, 51.6129, 51.6129, 50.0]

Success rate of 1 vs all: 19.35483870967742% with kernel type SVM_LINEAR

Success rate of 1 vs all: 53.2258064516129% with kernel type SVM_CHI2

Success rate of 1 vs all: 24.193548387096776% with kernel type SVM_INTER

Success rate of 1 vs all: 16.129032258064516% with kernel type SVM_SIGMOID

Success rate of 1 vs all: 33.87096774193548%with kernel type SVM_RBF

Για 60 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

[53.2258064516129, 56.451612903225815, 54.83870967741935, 58.06451612903226, 54.83870967741935, 54.83870967741935, 54.83870967741935, 54.83870967741935, 53.2258064516129, 56.451612903225815, 54.83870967741935, 54.83870967741935, 53.2258064516129, 53.2258064516129]

Success rate of 1 vs all: 3.225806451612903%with kernel type SVM_LINEAR

Success rate of 1 vs all: 45.16129032258064%with kernel type SVM_CHI2

Success rate of 1 vs all: 45.16129032258064%with kernel type SVM_INTER

Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID

Success rate of 1 vs all: 45.16129032258064%with kernel type SVM_RBF

Για 70 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]

[50.0, 59.67741935483871, 62.903225806451616, 56.451612903225815, 56.451612903225815, 56.451612903225815, 54.83870967741935, 53.2258064516129, 53.2258064516129, 53.2258064516129, 53.2258064516129, 53.2258064516129, 51.61290322580645, 50.0]

Success rate of 1 vs all: 19.35483870967742%with kernel type SVM_LINEAR

Success rate of 1 vs all: 54.83870967741935%with kernel type SVM_CHI2

Success rate of 1 vs all: 29.03225806451613%with kernel type SVM_INTER
Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID
Success rate of 1 vs all: 59.67741935483871%with kernel type SVM_RBF

Για 80 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
[53.2258064516129, 56.451612903225815, 59.67741935483871, 56.451612903225815, 56.451612903225815, 54.83870967741935, 56.451612903225815, 56.451612903225815, 58.06451612903226, 56.451612903225815, 53.2258064516129, 51.61290322580645, 51.61290322580645, 50.0]
Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_LINEAR
Success rate of 1 vs all: 48.38709677419355%with kernel type SVM_CHI2
Success rate of 1 vs all: 29.03225806451613%with kernel type SVM_INTER
Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID
Success rate of 1 vs all: 51.61290322580645%with kernel type SVM_RBF

Για 90 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
[51.61290322580645, 58.06451612903226, 58.06451612903226, 54.83870967741935, 56.451612903225815, 56.451612903225815, 56.451612903225815, 53.2258064516129, 54.83870967741935, 54.83870967741935, 58.06451612903226, 56.451612903225815, 56.451612903225815, 56.451612903225815]
Success rate of 1 vs all: 17.741935483870968%with kernel type SVM_LINEAR
Success rate of 1 vs all: 46.774193548387096%with kernel type SVM_CHI2
Success rate of 1 vs all: 27.419354838709676%with kernel type SVM_INTER
Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID
Success rate of 1 vs all: 61.29032258064516%with kernel type SVM_RBF

Για 100 cluster:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
[56.451612903225815, 54.83870967741935, 54.83870967741935, 59.67741935483871, 58.06451612903226, 58.06451612903226, 58.06451612903226, 58.06451612903226, 56.451612903225815, 56.451612903225815, 56.451612903225815, 56.451612903225815, 54.83870967741935, 53.2258064516129]
Success rate of 1 vs all: 22.58064516129032%with kernel type SVM_LINEAR
Success rate of 1 vs all: 58.06451612903226%with kernel type SVM_CHI2
Success rate of 1 vs all: 37.096774193548384%with kernel type SVM_INTER
Success rate of 1 vs all: 16.129032258064516%with kernel type SVM_SIGMOID
Success rate of 1 vs all: 50.0%with kernel type SVM_RBF

Παρατηρήσεις

- Παρατηρούμε πως στο K-nn τη μεγαλύτερη αποδοση (62.903225806451616) την είχαν τα 70 cluster me K=6 δηλαδή λαμβάνοντας υπ’ όψην 6 σημεια

- Στο SVM την μεγαλύτερη απόδοση (61.29032258064516%) την είχαν τα 90 cluster με kernel type SVM_RBF

Μια προσέγγυση για να επιλέξουμε τον αριθμό των σημείων K στον K-nn είναι sqrt(n) οπου n ο αριθμός των δειγματων που χρησιμοποιουνται για το training.

Για να επιλέξουμε ποιον αλγόριθμο θα χρησιμοποιήσουμε πρέπει να λάβουμε υπ' όψην τον αριθμό παραδειγμάτων στο σύνολο εκπαίδευσης, τις διαστάσεις του χαρακτηρισμένου χώρου,το αν εχουμε συσχετισμένα χαρακτηριστικά και το αν είναι overfitting ένα πρόβλημα

K-nn

Το K-NN είναι ανθεκτικό σε δεδομένα εκπαίδευσης με πολύ θόρυβο και είναι αποτελεσματικό σε περίπτωση μεγάλου αριθμού εκπαιδευτικών παραδειγμάτων. Ωστόσο, σε αυτόν τον αλγόριθμο πρέπει να καθορίσουμε την τιμή της παραμέτρου K (αριθμός πλησιέστερων γειτόνων) και τον τύπο της απόστασης που θα χρησιμοποιηθεί. Ο χρόνος υπολογισμού είναι επίσης πολύς, καθώς πρέπει να υπολογίσουμε την απόσταση κάθε στιγμιότυπου για όλα τα δείγματα εκπαίδευσης

SVM

Χρησιμοποιούμε SVM όταν το πρόβλημά δεν είναι γραμμικά διαχωρίσιμο(χρησιμοποιώντας kernel όπως RBF). Ένας άλλος λόγος που θα χρησιμοποιούσαμε το SVM είναι αν έχουμε ένα χώρο με πολύ μεγάλες διαστάσεις (π.χ. ταξινόμηση κειμένου). Το σημαντικότερο μειονέκτημα των SVM είναι ότι μπορεί να είναι ιδιαίτερα ανεπαρκή στο training. Συνήθως, απαιτεί αρκετό χρόνο και γι' αυτόν το λόγο δε συνίσταται σε προβλήματα με μεγάλο αριθμό παραδειγμάτων εκπαίδευσης (training examples).

Το KNN εκτελείται καλύτερα σε σύνολο δεδομένων με χώρο χαμηλών διαστάσεων ενώ το SVM είναι καλύτερο σε σύνολο δεδομένων με μεγάλες διαστάσεις.Η απόδοση εξαρτάται από την επιλογή του k στο KNN και την επιλογή hyper-plane και kernel στο SVM. Παρόλο που το KNN δίνει καλύτερα αποτελέσματα, το SVM είναι πιο αξιόπιστο και θεωρείται ως ένας real time classifier.