Όραση Υπολογιστών

Εργασία 3

Ακαδημαϊκό Έτος: 2019-2020

Καρασακαλίδης Αλέξανδρος Ηλίας

ΑΜ: 57448

- Ζητούμενα

Στην εκφώνηση της εργασίας μας ζητείται να ταξινομήσουμε ένα σύνολο εικόνων σε πολλαπλές κλάσεις. Συγκεκριμένα, οι απαιτήσεις της εργασίας είναι οι εξής:

1. Παραγωγή οπτικού λεξικού βασισμένη στο μοντέλο Bag of Visual Words. Η δημιουργία του λεξικού να γίνει με την χρήση του αλγορίθμου K-Means χρησιμοποιώντας του σύνολο των εικόνων εκπαίδευσης.

2. Εξαγωγή περιγραφέα σε κάθε εικόνα εκπαίδευσης με βάση το μοντέλο Bag of Visual Words χρησιμοποιώντας το προηγούμενο λεξικό. (Δεν επιτρέπεται η χρήση της κλάσης *cv.BOWImgDescriptorExtractor*).

3. Με βάση το αποτέλεσμα του προηγούμενου βήματος, να υλοποιηθεί η λειτουργία ταξινόμησης μιας εικόνας με την χρήση των παρακάτω ταξινομητών:

    a. Του αλγορίθμου k-NN (χωρίς την χρήση της συνάρτησης cv.ml.KNearest_create()).
    b. Του σχήματος one-versus-all όπου για κάθε κλάση εκπαιδεύεται ένας SVM ταξινομητής.

4. Αξιολόγηση του συστήματος χρησιμοποιώντας το σύνολο των εικόνων δοκιμής για την μέτρηση της ακρίβειας του συστήματος (ποσοστό επιτυχών ταξινομήσεων) του κάθε ταξινομητή. Να ελεγχθεί επίσης η επίδραση του αριθμού των οπτικών λέξεων, ο αριθμός των πλησιέστερων γειτόνων και ο τύπος του πυρήνα του SVM στην ακρίβεια των προβλέψεων.

Τα είδη των κλάσεων των εικόνων για τις οποίες μας ορίστηκαν τα παραπάνω ζητήματα είναι:

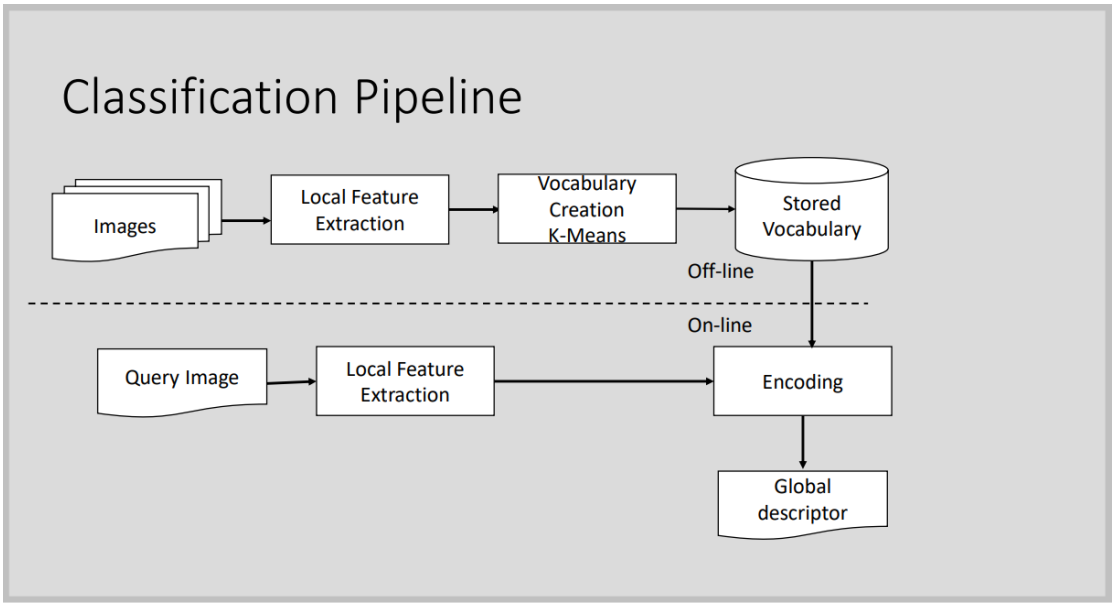
- Fighter Jet
- Motorbike
- School Bus
- Bicycle
- Airplane
- Car (Side view)

- Ανάλυση του Κώδικα

Ο κώδικας αποτελείται από ένα σύνολο συναρτήσεων. Η πιο βασική συνάρτηση, είναι αυτή η οποία διαβάζει μια εικόνα με βάση το path της, υπολογίζει τους descriptors της με την μέθοδο SIFT και μας τους επιστρέφει:

```
def extract_local_features(path):
    img = cv.imread(path)

    kp = sift.detect(img)
    desc = sift.compute(img, kp)
    return desc[1]
```

Για την δημιουργία του οπτικού λεξικού, πρέπει να πάρουμε τους descriptors των εικόνων που έχουμε ως training set και να τα κατηγοριοποιήσουμε με την K-Means:



Τις παραπάνω διεργασίες τις εκτελούμε με τις παρακάτω συναρτήσεις:

```
def import_train_imgs_desc(dir):
    train_descs = np.zeros((0, 128),dtype=np.float32)
    lsf = os.listdir(dir)
    print("Geting Descriptors: ")
    total = len(lsf)
    cntr=0
    for folders in lsf:
        files = os.listdir(dir+"/"+folders)
        for imgs in files:
            path = dir+"/"+folders+"/"+imgs
            desc = extract_local_desc(path)
            train_descs = np.concatenate((train_descs, desc), axis=0)
        cntr+=1
        print((cntr/total)*100,' % Done')
    return train_descs

def vocabulary(train_descs,ftr):
    print('Creating Vocabulary...')
    term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
    trainer = cv.BOWKMeansTrainer(ftr, term_crit, 2, cv.KMEANS_PP_CENTERS)
    vocabulary = trainer.cluster(train_descs.astype(np.float32))
    return vocabulary

def create_voc(words,dir):
    train_descs = import_train_imgs_desc(dir)
    voc = vocabulary(train_descs, words)
    np.save('vocabulary.npy', voc)
    return voc

def load_voc():
    return np.load('vocabulary.npy')
```

Με την συνάρτηση ‘import_train_imgs_desc’ συγκεντρώνουμε το σύνολο των descriptor των εικόνων που μας δίνονται στο training set. Στην συνάρτηση ‘vocabulary’ και με την αντίστοιχη συνάρτηση της OpenCV, ομαδοποιούμε τους descriptors για να παραχθεί το οπτικό λεξικό.

Η συνάρτηση create_voc καλεί τις παραπάνω συναρτήσεις ώστε να δημιουργηθεί το λεξικό το οποίο και αποθηκεύει στον υπολογιστή, σε περίπτωση που κληθεί η load_voc, η οποία θα ‘φορτώσει’ το οπτικό λεξικό που δημιουργήσαμε σε τυχών προηγούμενες εκτελέσεις του κώδικα, γλυτώνοντάς μας χρόνο.

Έχοντας φτιάξει λοιπόν το οπτικό λεξικό, συνεχίζουμε με την δημιουργία του ιστογράμματος της κάθε εικόνας που έχουμε στο training set, ώστε να δημιουργήσουμε τον επικαλούμενο Global Descriptor τους:

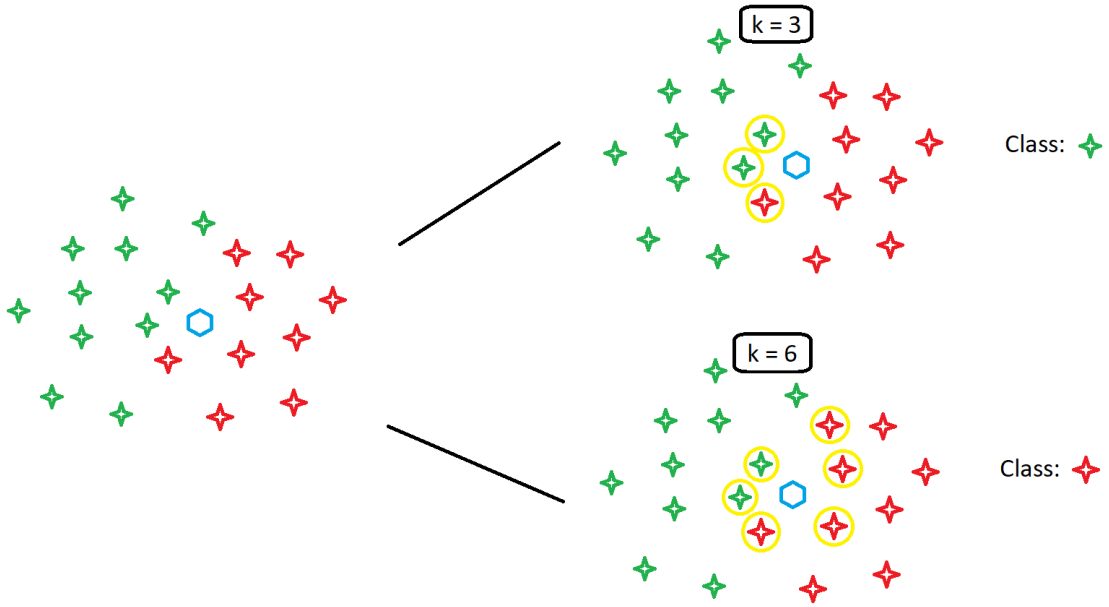
```
def Desc_Extractor(path,voc):

    desc = extract_local_desc(path)

    bf = cv.BFMatcher(cv.NORM_L2SQR)

    matches = bf.match(desc, voc)

    hist = np.zeros(len(voc), dtype=np.float32)

    for i in matches:
        hist[i.trainIdx]+=1

    sm = np.sum(hist)
    for i in range(len(hist)):
        hist[i]=hist[i]/sm

    hist = np.array([hist])
    return hist

def encoder(dir,voc):
    b_desc = np.zeros((0, len(voc)))
    lsf = os.listdir(dir)
    print("Creating Bag of Words: ")
    total = len(lsf)
    cntr=0
    for folders in lsf:
        files = os.listdir(dir+"/"+folders)
        for imgs in files:
            path = dir+"/"+folders+"/"+imgs

            hist = Desc_Extractor(path,voc)

            b_desc = np.concatenate((b_desc, hist), axis=0)

        cntr+=1
        print((cntr/total)*100,' % Done')
    return b_desc

def create_bow(voc,dir):
    bow_d = encoder(dir, voc)
    np.save('bow_descs.npy', bow_d)
    return bow_d

def load_bow():
    return np.load('bow_descs.npy')
```

Η συνάρτηση Desc_Extractor μας επιστρέφει για κάθε εικόνα το ιστόγραμμα της σύγκρισης των descriptor της με το οπτικό λεξικό που φτιάξαμε, συλλέγοντας τελικά το σύνολο των ιστογραμμάτων όλων των εικόνων του training set, με την συνάρτηση encoder. Καταλήγουμε έτσι να έχουμε σχηματίσει το μοντέλο Bag Of Visual Words (BOVW) για τις εικόνες εκπαίδευσης.

Τέλος, με τις συναρτήσεις create_bow, load_bow δημιουργούμε και αποθηκεύουμε ένα μοντέλο ή φορτώνουμε ένα ήδη δημιουργημένο αντίστοιχα.

Με το BOVW έτοιμο, προχωράμε στην δημιουργία των μεθόδων ταξινόμησης των test images που θα κατηγοριοποιήσουμε, ξεκινώντας με την k-Nearest Neighbor (k-NN).

Ο αλγόριθμος k-NN είναι ένας από τους απλούστερους αλγορίθμους κατηγοριοποίησης.



Η κατηγοριοποίηση γίνεται με βάση τα k πιο κοντινά σημεία στο εξεταζόμενο (με βάση την Ευκλείδεια απόστασή τους). Στην δικιά μας περίπτωση όμως το κάθε ‘σημείο’ αποτελείται από τουλάχιστον 25 συνιστώσες (ίσο με το μέγεθος του λεξικού μας). Οι συναρτήσεις που εφαρμόζουν τον k-NN είναι:

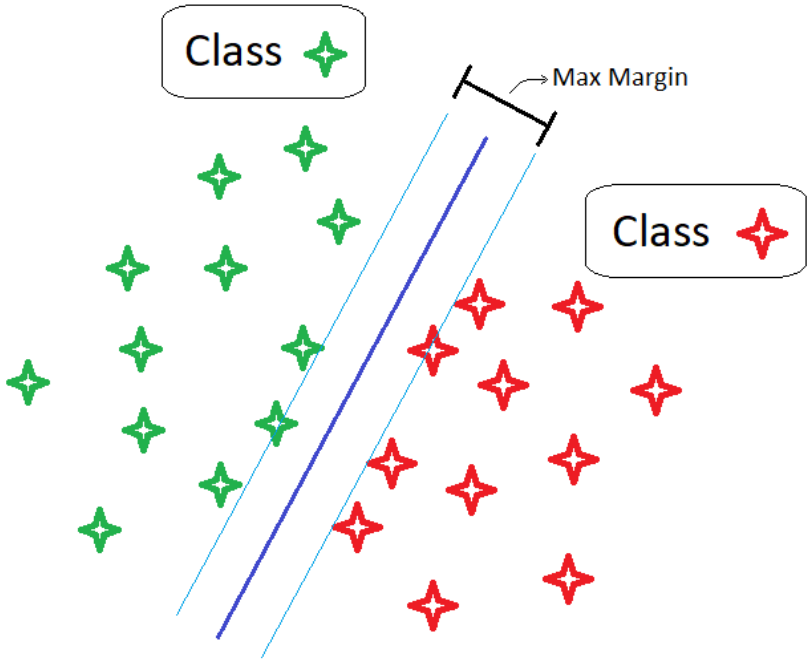
```
def indexes(dir):
    folders = os.listdir(dir)
    cntr=0
    lst = []
    for i in folders:
        for j in os.listdir(dir+'/'+i):
            lst.append(cntr)
        cntr = cntr + 1
    return np.array(lst)

def kNN(img_data,bow_d,NN):
    train_indexes = indexes('imagedb_train')
    distances = np.sqrt(np.sum(np.square(img_data-bow_d),axis=1))
    tst = img_data-bow_d
    sorted_distances = np.argsort(distances)
    lst = []
    for i in range(NN):
        lst.append(train_indexes[sorted_distances[i]])
    a = np.argmax(np.bincount(np.array(lst)))
    return np.argmax(np.bincount(np.array(lst)))
```

Αξίζει να σημειωθεί ότι στην παραπάνω συνάρτηση, αλλά και σε κάθε άλλη συνάρτηση, η κλάση της κάθε εικόνας (η πραγματική – αλλά και η προβλεπόμενη) ορίζονται με βάση τον φάκελο από τον οποίο προέρχεται η κάθε εικόνα. Η συνάρτηση indexes υλοποιεί αυτή ακριβώς την διατύπωση.

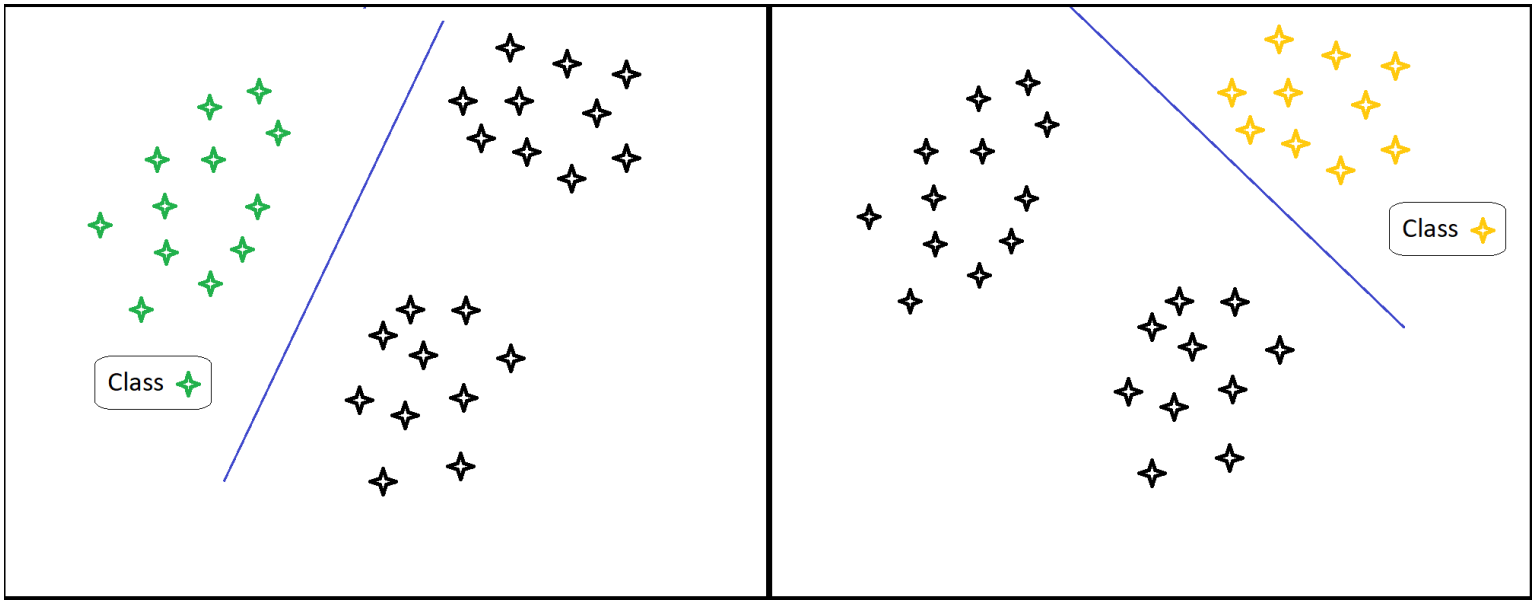
Η άλλη μέθοδος κατηγοριοποίησης γίνεται με την χρήση του σχήματος one-versus-all όπου για την κάθε κλάση εκπαιδεύεται ένας SVM ταξινομητής.

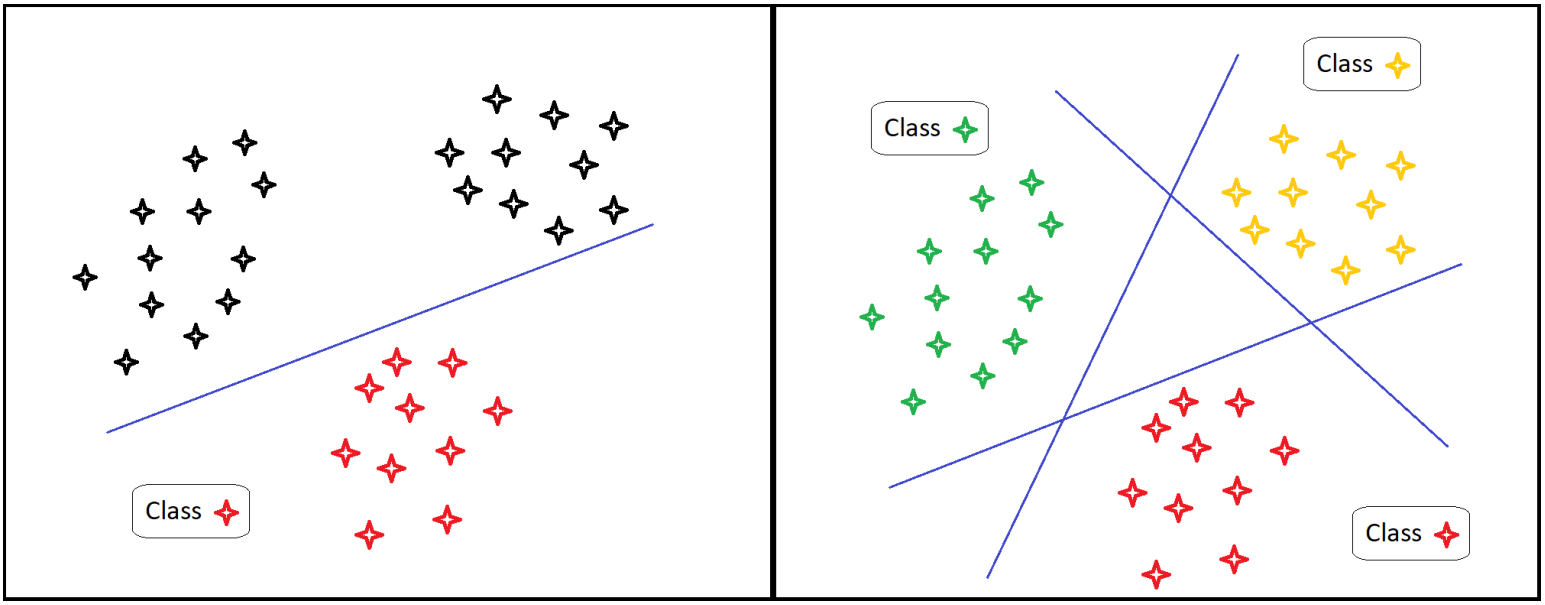
Η SVM ταξινόμηση ανήκει στην κατηγορία των binary classifications, ταξινομεί δηλαδή τα ‘σημεία’ μας σε δύο κλάσεις.

Ο SVM με γραμμικό kernel για παράδειγμα, κατηγοριοποιεί σύμφωνα με το διπλανό σχήμα.



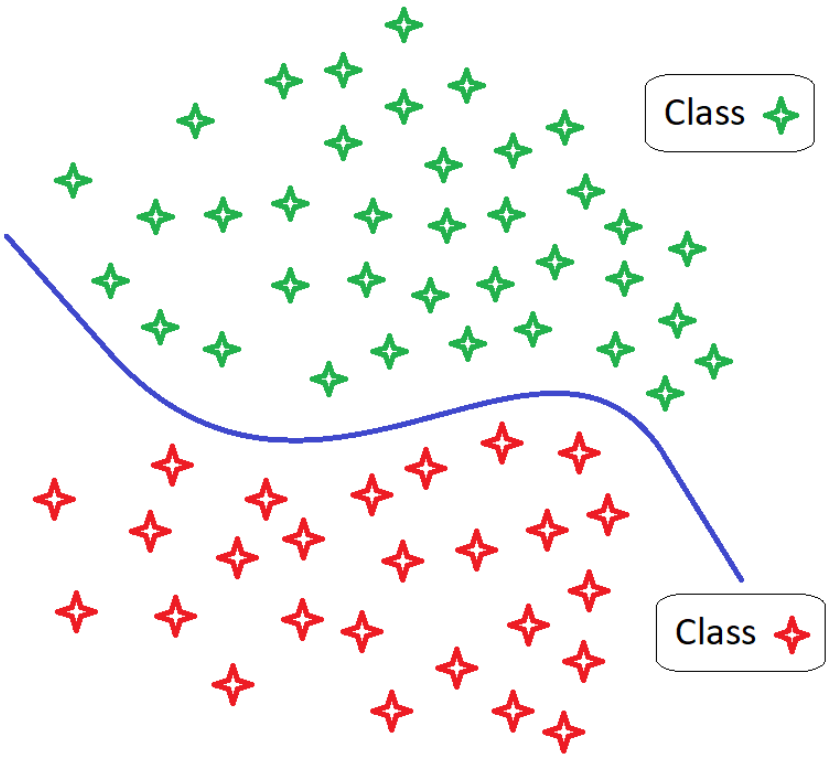
Για να εφαρμοστεί στο δικό μας πρόβλημα, όπου πρέπει να κατηγοριοποιήσουμε τα ‘σημεία’ (εικόνες) σε 6 κλάσεις, χρησιμοποιείται το σχήμα one-versus-all (όπως φαίνεται παρακάτω για τον SVM με linear kernel).





Χρησιμοποιούμε τόσους SVM, όσους οι συνολικές κλάσεις που θέλουμε να μπορούμε να κατηγοριοποιήσουμε. Κάθε αντικείμενο λοιπόν, θα εξετάζεται από κάθε SVM. Σε αυτόν στον οποίο 'ταιριάζει' περισσότερο, αντιστοιχεί και η κλάση του.

Αντίστοιχα με τον γραμμικό kernel, υπάρχουν και άλλοι, όπως για παράδειγμα ο πολυωνυμικός.



Εμείς χρησιμοποιούμε τους εξής 3:

- Linear SVM
- Histogram Intersection SVM
- RBF SVM

Οι συναρτήσεις με τις οποίες φτιάχνεται τόσο ο κάθε SVM όσο και το συνολικό σχήμα one-versus-all είναι:

```
def create_SVM(img_ind,bow_d,name,kernel):

    svm = cv.ml.SVM_create()
    svm.setType(cv.ml.SVM_C_SVC)
    svm.setKernel(kernel)
    svm.setTermCriteria((cv.TERM_CRITERIA_COUNT, 100, 1.e-06))
    svm.trainAuto(bow_d.astype(np.float32), cv.ml.ROW_SAMPLE, img_ind)
    svm.save(name)

def create_SVM_one_vs_all(bow_d,kernel):
    paths = image_paths('imagedb_train')

    Fighters = np.array(['jet' in a for a in paths], np.int32)
    Motorbikes = np.array(['motorbikes' in a for a in paths], np.int32)
    Busses = np.array(['bus' in a for a in paths], np.int32)
    Bikes = np.array(['touring' in a for a in paths], np.int32)
    Planes = np.array(['airplanes' in a for a in paths], np.int32)
    Cars = np.array(['car' in a for a in paths], np.int32)

    create_SVM(Fighters, bow_d, 'svm_fighters',kernel)
    create_SVM(Motorbikes, bow_d, 'svm_motorbikes',kernel)
    create_SVM(Busses, bow_d, 'svm_busses',kernel)
    create_SVM(Bikes, bow_d, 'svm_bikes',kernel)
    create_SVM(Planes, bow_d, 'svm_planes',kernel)
    create_SVM(Cars, bow_d, 'svm_cars',kernel)

def guess_with_SVMs(test_img_path,voc, normalize):
    names=['fighters','motorbikes','busses','bikes','planes','cars']

    bow_desc = Desc_Extractor(test_img_path,voc)

    if normalize:
        bow_desc = norm(bow_desc)

    matching=[0,0,0,0,0,0]

    for i in range(6):
        svm = cv.ml.SVM_create()
        svm = svm.load('svm_'+names[i])
        matching[i] = svm.predict(bow_desc.astype(np.float32),
flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1][0][0]

    return matching
```

Η συνάρτηση create_SVM_one_vs_all φτιάχνει έναν SVM (create_SVM) για κάθε κλάση εικόνων που έχουμε (με βάση στον φάκελο τον οποίο ανήκουν). Η συνάρτηση guess_with_SVMs κατηγοριοποιεί την κάθε εικόνα με κάθε SVM που έχουμε φτιάξει και ‘διαλέγει’ τον SVM στον οποίο ανήκει με μεγαλύτερη βεβαιότητα η εικόνα που κατηγοριοποιείται (άρα και την κλάση της).

Τέλος, καλούμαστε να αξιολογήσουμε τις παραπάνω μεθόδους, κατηγοριοποιώντας το σύνολο των test images με αυτές. Οι αντίστοιχες συναρτήσεις για την αξιολόγηση της kNN και του one-versus-all είναι οι παρακάτω:

```
def check_KNN(dir,NN,voc,bow_d,normalize):

    file = os.listdir(dir)
    results = []
    cntr = 0
    for folders in file:
        imgs = os.listdir(dir + '/' + folders)
        for i in imgs:
            path = dir + '/' + folders + '/' + i
            img_desc = Desc_Extractor(path, voc)
            if normalize:
                img_desc = norm(img_desc)
            result = kNN(img_desc, bow_d, NN)
            if result == cntr:
                results.append(1)
            else:
                results.append(0)
        cntr = cntr + 1

    return sum(results) / len(results)

def check_SVM(dir,voc,normalize):

    lsf = os.listdir(dir)
    success = []
    cntr = 0
    for folders in lsf:
        img_name = os.listdir(dir+ '/' + folders)
        for i in img_name:
            path = dir + '/' + folders + '/' + i
            tmp = np.asarray(guess_with_SVMs(path, voc, normalize))
            if cntr == np.argmin(tmp):
                success.append(1)
            else:
                success.append(0)
        cntr = cntr + 1
    return sum(success) / len(success)
```

Βέβαια, η αξιολόγηση έγινε σε διάφορα λεξικά που δημιουργήσαμε, με διάφορους αριθμούς k για την k-NN αλλά και για τους διαφορετικούς kernel για τους SVM, με την χρήση των παρακάτω συναρτήσεων:

```
def norm(data):
    for i in range(data.shape[0]):
        data[i] = data[i]/np.sqrt(np.sum(np.square(data[i])))
    return data
```

```
def full_all(voc,bow_d,normalize):
    test_dir = 'imagedb_test'
    NN = [1,3,6,12,18,36]
    knnR = []
    for i in range(len(NN)):
        print('Classifying with ' + str(NN[i]) +'-NN...')
        knnR.append('kNN - '+ str(NN[i]) + ' Neighbours Rate: ' +
str(check_KNN(test_dir,NN[i],voc,bow_d,normalize)))

    kernels = [cv.ml.SVM_LINEAR,cv.ml.SVM_INTER,cv.ml.SVM_RBF]
    kernel_name = ['Linear','Histogram Intersection', 'RBF']
    svmR = []
    for i in range(len(kernels)):
        print('Classifying with SVM and ' + kernel_name[i] + ' kernel...')
        create_SVM_one_vs_all(bow_d, kernels[i])
        svmR.append('SVM [' + kernel_name[i] + '] Success Rate: ' +
str(check_SVM(test_dir,voc,normalize)))

    return knnR, svmR

def multi_check(dir,normalize = True):

    vocsize = [25,50,100,250,500,800]
    results = []

    for loop in range(len(vocsize)):
        print('Vocabulary size ', vocsize[loop], ' :')
        voc = create_voc(vocsize[loop], dir)
        bow_d = create_bow(voc, dir)

        if normalize:
            bow_d = norm(bow_d)

        knns, svms = full_all(voc, bow_d, normalize)
        results.append('Voc_Size = ' + str(vocsize[loop]))
        for i in knns:
            results.append(i)
        for i in svms:
            results.append(i)

    return results
```

Συγκεκριμένα δημιουργήθηκαν λεξικά των 25, 50, 100, 250, 500 και 800 λέξεων, με τα οποία έγιναν κατηγοριοποιήσεις με την k-NN για 1, 3, 6, 12, 18 και 36 ‘γείτονες’ αλλά και με τα SVM kernels που προαναφέρθηκαν.

Τέλος, μια σημαντική διαφοροποίηση ανάμεσα στις αξιολογήσεις είναι η κανονικοποίηση (Ευκλείδεια κανονικοποίηση) των ιστογραμμάτων των εικόνων αλλά και του συνόλου του BOVW. Μάλιστα, αξιολογήθηκαν όλα τα προηγούμενα με αλλά και χωρίς κανονικοποίηση.

Το σύνολο των ποσοστών επιτυχίας κάθε περίπτωσης καταγράφεται στους παρακάτω πίνακες:

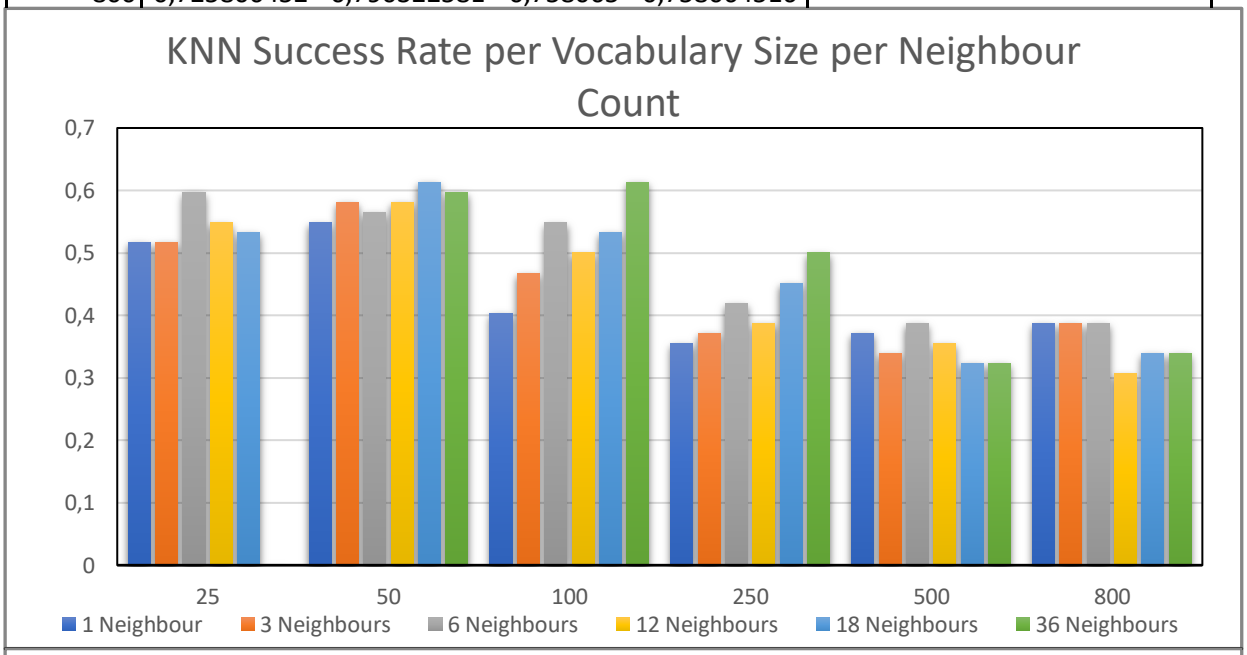
- Αποτελέσματα και ανάλυση

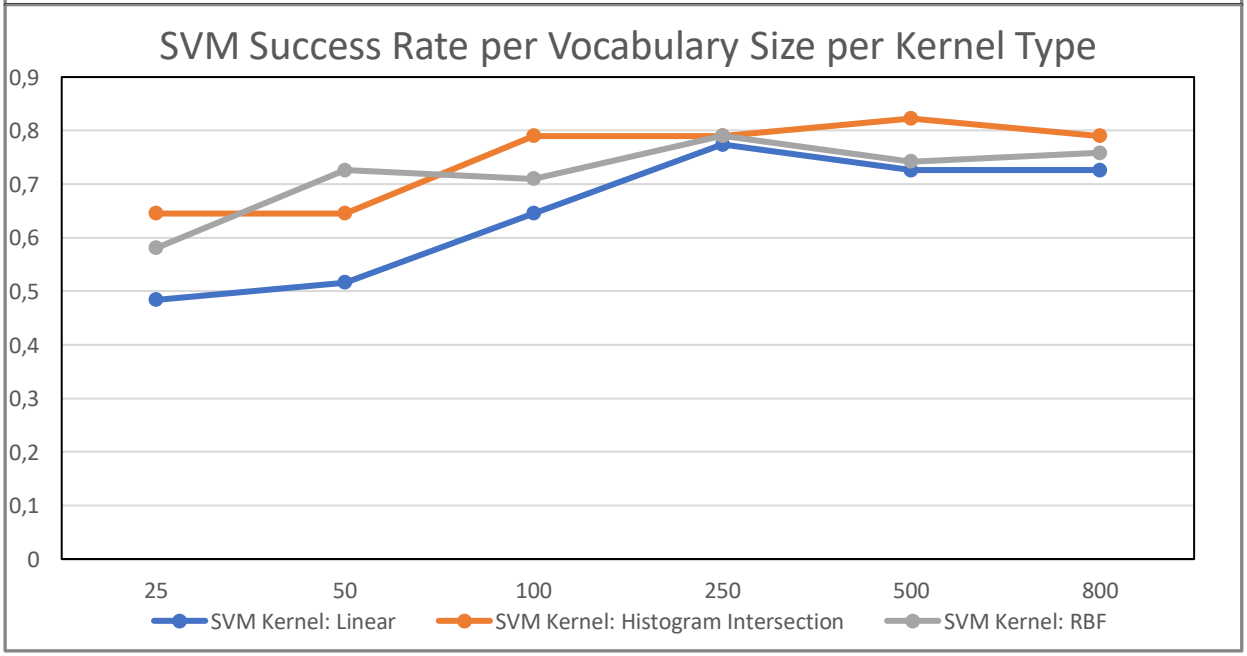
Χωρίς κανονικοποίηση:

words	knn-1	knn-3	knn-6	knn-12	knn-18	knn-36	knn-avg
25	0,516129	0,516129	0,596774	0,548387	0,532258	-	0,541935
50	0,548387	0,580645	0,564516	0,580645	0,612903	0,596774	0,580645
100	0,403226	0,467742	0,548387	0,5	0,532258	0,612903	0,510753
250	0,354839	0,370968	0,419355	0,387097	0,451613	0,5	0,413978
500	0,370968	0,33871	0,387097	0,354839	0,322581	0,322581	0,349462
800	0,387097	0,387097	0,387097	0,306452	0,33871	0,33871	0,357527

words	svm - linear	svm - inter	svm - rbf	svm-avg
25	0,483870968	0,64516129	0,580645	0,569892473
50	0,516129032	0,64516129	0,725806	0,629032258
100	0,64516129	0,790322581	0,709677	0,715053763
250	0,774193548	0,790322581	0,790323	0,784946237
500	0,725806452	0,822580645	0,741935	0,76344086
800	0,725806452	0,790322581	0,758065	0,758064516

Standard Deviation	
knn-avg	0,098668
svm-avg	0,085713



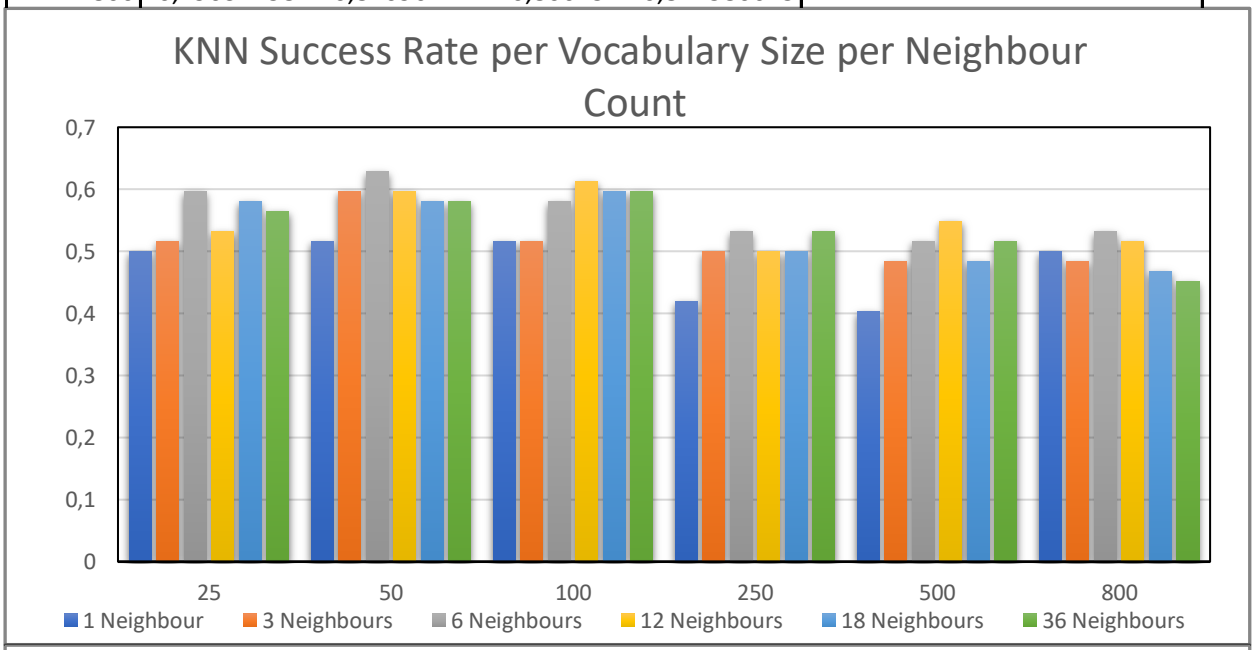


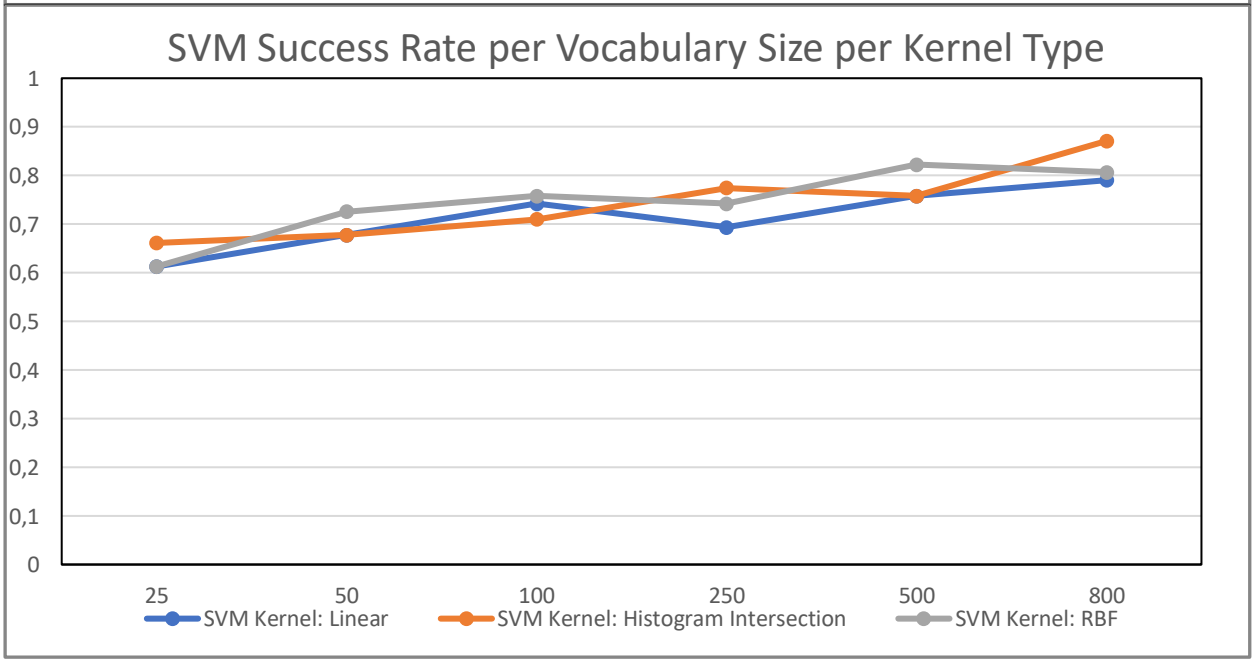
Με κανονικοποίηση:

words	knn-1	knn-3	knn-6	knn-12	knn-18	knn-36	knn-avg
25	0,5	0,516129	0,596774	0,532258	0,580645	-	0,545161
50	0,516129	0,596774	0,629032	0,596774	0,580645	0,580645	0,583333
100	0,516129	0,516129	0,580645	0,612903	0,596774	0,596774	0,569892
250	0,419355	0,5	0,532258	0,5	0,5	0,532258	0,497312
500	0,403226	0,483871	0,516129	0,548387	0,483871	0,516129	0,491935
800	0,5	0,483871	0,532258	0,516129	0,467742	0,451613	0,491935

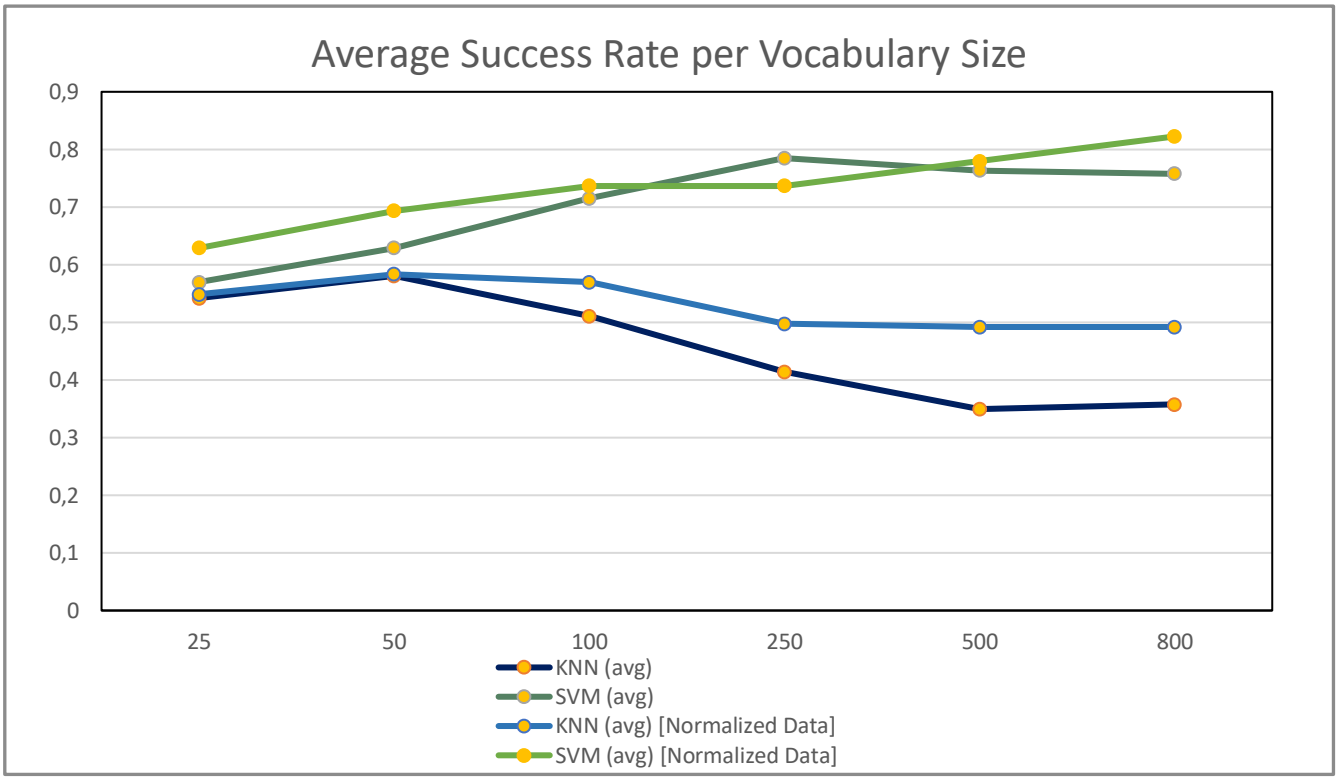
words	svm - linear	svm - inter	svm - rbf	svm-avg
25	0,612903226	0,661290323	0,612903	0,629032258
50	0,677419355	0,677419355	0,725806	0,693548387
100	0,741935484	0,709677419	0,758065	0,73655914
250	0,693548387	0,774193548	0,741935	0,73655914
500	0,758064516	0,758064516	0,822581	0,779569892
800	0,790322581	0,870967742	0,806452	0,822580645

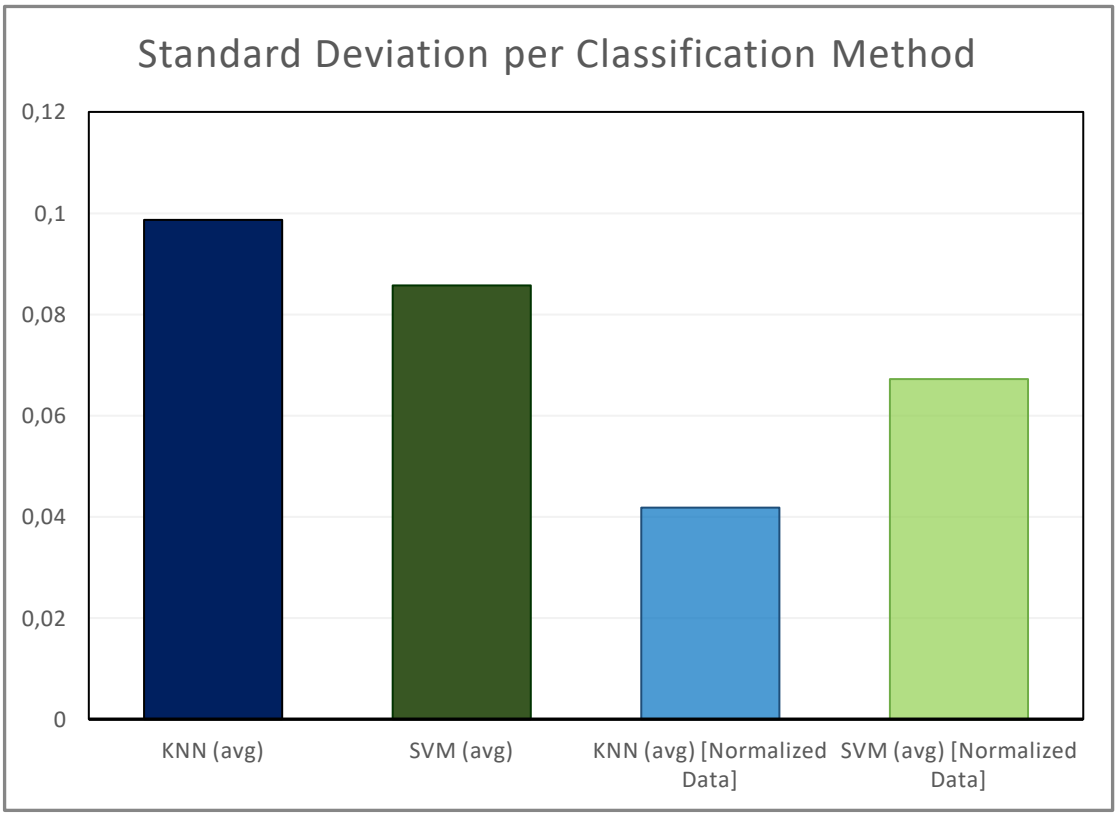
Standard Deviation	
knn-avg	0,041807
svm-avg	0,067208





Συνολικά:


Average Success Rate per Vocabulary Size
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
25
50
100
250
500
800
KNN (avg)
SVM (avg)
KNN (avg) [Normalized Data]
SVM (avg) [Normalized Data]


Standard Deviation per Classification Method
0,12
0,1
0,08
0,06
0,04
0,02
0
KNN (avg)
SVM (avg)
KNN (avg) [Normalized Data]
SVM (avg) [Normalized Data]

Καταλήγουμε λοιπόν στα εξής συμπεράσματα:

- Οι SVM είναι στην πλειοψηφία των περιπτώσεων ακριβέστεροι από τους k-NN.

- Μεγαλύτερο λεξικό (με σταθερό σύνολο train images) συνεπάγεται:
  - Μεγαλύτερη ακρίβεια για τους SVM
  - Μικρότερη ακρίβεια για τους k-NN

- Καλύτερο -γενικά- εύρος γειτόνων για τους k-NN από 6 μέχρι 18 γείτονες.

- Αναλόγως με το αν είναι κανονικοποιημένα τα δεδομένα ή όχι:

  - Μικρότερη απόκλιση μεταξύ της ακρίβειας των SVM μοντέλων τόσο μεταξύ των λεξικών διαφορετικού μεγέθους, όσο και μεταξύ των διαφορετικών kernel στην περίπτωση κανονικοποιημένων δεδομένων.

  - Το ίδιο παρατηρείται - και μάλιστα σε μεγαλύτερο βαθμό - στους k-NN αλγορίθμους.

  - Εμφανώς καλύτερος - κατά μέσο όρο - σε μη κωδικοποιημένα δεδομένα, ο SVM με kernel RBF. Σε κωδικοποιημένα δεδομένα είχε την επιτυχέστερη κατηγοριοποίηση που παρατηρήθηκε σε οποιαδήποτε δοκιμή έγινε με οποιαδήποτε μέθοδο.

  - Σε σχεδόν κάθε περίπτωση, η κανονικοποίηση οδήγησε σε μεγαλύτερα ποσοστά επιτυχίας.

Περαιτέρω βελτιώσεις των αποτελεσμάτων μπορεί να περιλαμβάνουν:

- Αύξηση του συνόλου train images:
    - Πρόσθεση νέων εικόνων.
    - Δημιουργία εικόνων με μετασχηματισμούς των ήδη υπάρχων εικόνων.

- Διαγραφή συγκεκριμένων εικόνων του συνόλου train images. Συγκεκριμένα, των εικόνων που δεν απεικονίζουν ικανοποιητικά τα εν λόγω αντικείμενα, ή που περιέχουν ακραίες περιπτώσεις των εν λόγω αντικειμένων (π.χ. αεροπλάνο που εκρήγνυται).

Καρασακαλίδης Αλέξανδρος Ηλίας

20/12/2019