Όραση Υπολογιστών

3η Εργασία

Εμμανουηλίδης Κωνσταντίνος

A.M. 57315

ΠΕΡΙΕΧΟΜΕΝΑ

ΘΕΩΡΗΤΙΚΗ ΑΝΑΛΥΣΗ

ΤΑΞΙΝΟΜΗΣΗ

Το πρόβλημα της ταξινόμησης αναφέρει ότι δοθέντων κάποιων δεδομένων που είναι γνωστό σε ποιες κλάσεις ανήκουν ζητείται η ταξινόμηση άγνωστων ως προς την κλάση δεδομένων στις αρχικές κλάσεις. Για την περίπτωση παραπάνω των δύο κλάσεων ακολουθείται είτε η στρατηγική one-versus-all είτε η στρατηγική one-versus-one.

Στην πρώτη περίπτωση υπάρχουν τόσοι ταξινομητές όσοι και οι κλάσεις και ο κάθε ταξινομητής προβλέπει αν το δείγμα δεδομένων ανήκει στην κλάση που αναφέρεται ή όχι.

Στη δεύτερη στρατηγική ο κάθε ταξινομητής χρησιμοποιεί δείγματα δεδομένων από δύο κλάσεις και προβλέπει σε ποια από τις δύο κλάσεις ανήκει το άγνωστο δείγμα που τίθεται προς ταξινόμηση.

Παρότι υπάρχουν αρκετά μοντέλα ταξινόμησης στην συγκεκριμένη εργασία θα εξετάσουμε τα μοντέλα kNN και SVM.

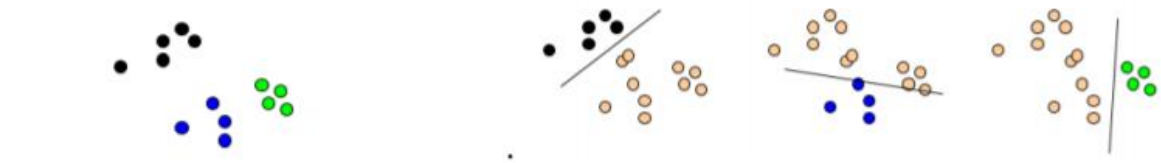
K – NEAREST NEIGHBORS (kNN)

Ο αλγόριθμος k – nearest neighbors είναι μία μη παραμετρική μέθοδος που χρησιμοποιεί τα k κοντινότερα σημεία του δειγματοχώρου στο δείγμα προς ταξινόμηση για την εξαγωγή της κλάσης στην οποία προβλέπεται να ανήκει το δείγμα προς ταξινόμηση. Οι συνηθέστερες μετρικές που χρησιμοποιούνται είναι η Ευκλείδεια απόσταση και η απόσταση Hamming, ενώ η επιλογή της παραμέτρου k εξαρτάται από τα δεδομένα και μπορεί να επιλεχθεί χρησιμοποιώντας διάφορες ευρετικές τεχνικές.

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

SUPPORT VECTOR MACHINES

Τα support vector machines είναι μοντέλα συσχετιζόμενα με αλγορίθμους εκμάθησης που στοχεύουν στην παλινδρόμηση ή ταξινόμηση δεδομένων. Ειδικότερα, δοθέντος ενός συνόλου δειγμάτων εκπαίδευσης και των αντίστοιχων κλάσεων που ανήκουν, τα δείγματα αναπαριστούν σημεία στο δειγματοχώρο και με χρήση υπερεπιπέδων διαχωρίζονται ανάλογα με την κλάση στην οποία ανήκουν. Στη συνέχεια, όταν τίθεται ένα δείγμα προς ταξινόμηση ουσιαστικά τοποθετείται και αυτό στο χώρο και εντοπίζεται η κλάση στην οποία ανήκει με βάση τα υπερεπίπεδα που έχουν προκύψει



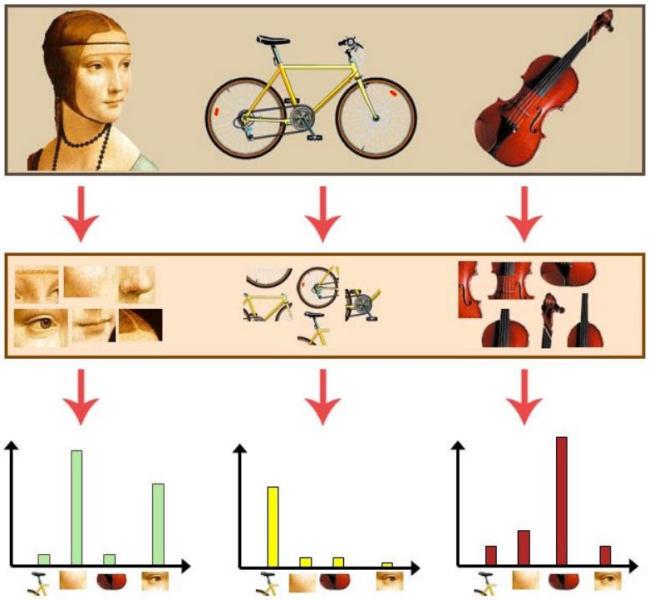
(a) Data set with 3 labels (b) Decomposed into binary problems

ΔΙΑΦΟΡΕΤΙΚΟΙ ΠΥΡΗΝΕΣ ΣΤΟ ΜΟΝΤΕΛΟ SVM

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

BAG OF VISUAL WORDS MODEL

Η γενική ιδέα του μοντέλου bag of visual words (BOVW) είναι να αντιπροσωπεύει μια εικόνα ως σύνολο χαρακτηριστικών(features). Τα χαρακτηριστικά γνωρίσματα αποτελούνται από keypoints και τους περιγραφείς. Τα keypoints είναι τα σημεία "ξεχωρίζουν" σε μια εικόνα, οπότε δεν έχει σημασία η εικόνα να περιστρέφεται, να συρρικνώνεται ή να επεκτείνεται, τα keypoints της θα είναι πάντα τα ίδια. Και ο περιγραφέας είναι η περιγραφή του βασικού σημείου. Χρησιμοποιούμε τα keypoints και τους περιγραφείς για να κατασκευάσουμε λεξιλόγια και να αναπαριστούμε κάθε εικόνα ως ιστογράμματα συχνοτήτων των χαρακτηριστικών που υπάρχουν στην εικόνα. Από το ιστόγραμμα συχνότητας, αργότερα, μπορούμε να βρούμε άλλες παρόμοιες εικόνες ή να προβλέψουμε την κατηγορία της εικόνας.



ΑΛΓΟΡΙΘΜΟΣ PYRAMID

Ο Pyramid είναι ένας γρήγορος αλγόριθμος που λειτουργεί ως kernel και χρησιμοποιεί την κατάσταση του Mercer για να χαρτογραφήσει το σύνολο χαρακτηριστικών σε υψηλή διάσταση σε πολυδιάστατα ιστογράμματα πολλαπλών αναλύσεων. Ένα πλεονέκτημα αυτών των ιστογραμμάτων πολλαπλής ανάλυσης είναι η ικανότητα τους να εντοπίζουν συνυπάρχουσα χαρακτηριστικά. Ο kernel χτίζει ιστογράμματα πολλαπλών αναλύσεων με τη συσσώρευση σημείων δεδομένων σε διακεκριμένες περιοχές αυξανόμενου μεγέθους.

Άσκηση

Η παρούσα εργασία διαπραγματεύεται το πρόβλημα της ταξινόμησης δεδομένων σε κλάσεις. Στόχος του συγκεκριμένης άσκησης η υλοποίηση προγράμματος σε Python, το οποίο θα αφορά στο πρόβλημα της ταξινόμησης πολλαπλών κλάσεων (multi-class classification).

Στο πρώτο ερώτημα παράγεται το οπτικό λεξικό (visual vocabulary) βασισμένο στο μοντέλο Bag of Visual Words. Στη συνέχεια, γίνεται εξαγωγή περιγραφέα σε κάθε εικόνα εκπαίδευση χρησιμοποιώντας το λεξικό που προέκυψε στο προηγούμενο βήμα. Κατόπιν, υλοποιείται η λειτουργία ταξινόμησης μιας εικόνας κάνοντας χρήση των ταξινομητών kNN και SVM. Τέλος, συγκρίνονται τα αποτελέσματα των παραπάνω ταξινομητών ανάλογα με τις εισερχόμενες σε αυτούς παραμέτρους.

Τα βήματα που ακολουθήθηκαν για την υλοποίηση της άσκησης βρίσκονται αναλυτικότερα παρακάτω.

ΒΗΜΑΤΑ

Ερώτημα 1

- ✓ Αρχικά, φορτώνουμε τις εικόνες που θα χρησιμοποιηθούν τόσο για εκπαίδευση όσο και για τον έλεγχο της αποτελεσματικότητας του μοντέλου.
- ✓ Στη συνέχεια, υπολογίζουμε τους ανιχνευτές και τους περιγραφείς τους με χρήση συναρτήσεων της OpenCV
- ✓ Δημιουργούμε το λεξικό(vocabulary) με χρήση της συνάρτησης BOWKmeanstrainer.cluster().Η συνάρτηση επιστρέφει τα κέντρα των clusters που απαρτίζουν το λεξικό.

Ερώτημα 2

- ✓ Υπολογίζουμε τους ανιχνευτές και τους περιγραφείς σε κάθε εικόνα εκπαίδευσης με χρήση συναρτήσεων της OpenCV
- ✓ Υλοποιούμε συνάρτηση που βρίσκει για κάθε περιγραφέα κάθε εικόνας το word του λεξικού που παράχθηκε στο προηγούμενο ερώτημα και έχει ιστόγραμμα που είναι πιο "κοντά" στον περιγραφέα.

Ερώτημα 3

- Ταξινομητής kNN
  - ✓ Υπολογίζουμε τους ανιχνευτές και τους περιγραφείς σε κάθε εικόνα αξιολόγησης με χρήση συναρτήσεων της OpenCV
  - ✓ Βρίσκουμε την λέξη που βρίσκεται πιο « κοντά » σε κάθε περιγραφέα κάθε εικόνας αξιολόγησης
  - ✓ Υλοποιούμε συνάρτηση που επιτυγχάνει τη λειτουργία ταξινόμησης των εικόνων αξιολόγησης με βάση τον αλγόριθμο kNN.Η συνάρτηση υπολογίζει τους k κοντινότερους « γείτονες » του εκάστοτε σημείου και το αναθέτει στην κλάση που εμφανίζεται με μεγαλύτερη συχνότητα στους k κοντινότερους « γείτονες »
  - ✓ Υπολογίζουμε το ποσοστό επιτυχίας της ταξινόμησης με kNN

- Ταξινομητής SVM
  - ✓ Δημιουργούμε labels για τα δείγματα δεδομένων εκπαίδευσης και αξιολόγησης
  - ✓ Εκπαιδεύουμε έναν SVM ταξινομητή για κάθε κλάση ταξινόμησης(στρατηγική one-versus-all)
  - ✓ Ταξινομούμε τα δεδομένα αξιολόγησης με χρήση του εκπαιδευμένου μοντέλου SVM.Κατά την ταξινόμηση των δειγμάτων αξιολόγησης επιλέγουμε την κλάση με ταξινομητή στον οποίο βρίσκεται πιο κοντά το δείγμα αξιολόγησης.

Ερώτημα 4

- ✓ Υπολογίζουμε την ακρίβεια του συστήματος ταξινόμησης με kNN και SVM
- ✓ Εξετάζουμε τη διαφορά των αποτελεσμάτων για επιλογή διαφορετικού αριθμού οπτικών λέξεων, πλησιέστερων γειτόνων και τύπου πυρήνα(kernel) του SVM

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΥΛΟΠΟΙΗΣΗ

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

image_db = "imagedb_train"
image_db_test = "imagedb_test"

sift = cv.xfeatures2d_SIFT.create()

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

def find_bow_desc(desc, vocab):
    bow_desc = np.zeros((1, vocab.shape[0]), dtype = np.float32)
    for i in range(desc.shape[0]):
        diff = desc[i, :] - vocab
        dists = np.sum(np.square(diff), axis=1)
        min_dist_index = np.argmin(dists)
        # Increase frequency of the word in the histogram
        bow_desc[0, min_dist_index] += 1
    # Return histogram
    return bow_desc
```

```python
    data_expanded_transpose = np.transpose(data_expanded)
    min_pred = 10000000000000000
    for elem in list_with_svms:
        prediction = elem.predict(data_expanded_transpose.astype(np.float32), flags=cv.ml.STAT_MODEL_RAW_OUTPUT)[1]
        if prediction[0] <= min_pred:
            min_pred = prediction[0]
            best_svm = list_with_svms.index(elem)
    return best_svm


folders = os.listdir(image_db)
train_descs = np.zeros((0, 128))
print('Finding train_descs...')

for folder in folders:
    folder_path = os.path.join(image_db, folder)
    files = os.listdir(folder_path)
    for file in files:
        file_path = os.path.join(folder_path, file)
        desc = extract_local_features(file_path)
        if desc is None:
            print("None")
            continue
        train_descs = np.concatenate((train_descs, desc), axis=0)

print('Creating vocabulary...')
term_crit = (cv.TERM_CRITERIA_EPS, 30, 0.1)
trainer = cv.BOWKMeansTrainer(50, term_crit, 1, cv.KMEANS_PP_CENTERS)
vocabulary = trainer.cluster(train_descs.astype(np.float32))
np.save('vocabulary.npy', vocabulary)
```

```python
vocabulary = np.load('vocabulary.npy')

#<---------------------------------- Second Task ---------------------------------------->
sift = cv.xfeatures2d_SIFT.create()
train_labels = np.zeros((0,1))
temp_label = np.zeros((1,1))


img_paths =[]

bow_descs = np.zeros((0, vocabulary.shape[0]))
for folder in folders:
    folder_path = os.path.join(image_db, folder)
    files = os.listdir(folder_path)
    for file in files:
        file_path = os.path.join(folder_path, file)
        desc = extract_local_features(file_path)
        bow_desc = find_bow_desc(desc, vocabulary)
        img_paths.append(file_path)
        # computing histogram of image:
        bow_descs = np.concatenate((bow_descs, bow_desc), axis = 0)
        # labeling each image:
        train_labels = np.concatenate((train_labels, temp_label), axis = 0)
    temp_label[0] = temp_label[0] + 1
np.save('bow_descs.npy', bow_descs)
np.save('train_labels.npy', train_labels)

# Load bow_descs and train_labels
bow_descs = np.load('bow_descs.npy')
train_labels = np.load('train_labels.npy')
print('Found bow_descs & train_labels')
```

```python
img_paths =[]

test_bow_descs = np.zeros((0, vocabulary.shape[0]))
test_labels = np.zeros((0,1))
temp_label = np.zeros((1,1))
folders = os.listdir(image_db_test)
for folder in folders:
    folder_path = os.path.join(image_db, folder)
    files = os.listdir(folder_path)
    for file in files:
        file_path = os.path.join(folder_path, file)
        desc = extract_local_features(file_path)
        test_bow_desc = find_bow_desc(desc, vocabulary)
        img_paths.append(file_path)
        # computing histogram of test image:
        test_bow_descs = np.concatenate((test_bow_descs, test_bow_desc), axis = 0)
        # labeling each test image:
        test_labels = np.concatenate((test_labels, temp_label), axis = 0)
    temp_label[0] = temp_label[0] + 1
np.save('test_bow_descs.npy', test_bow_descs)
np.save('test_labels.npy', test_labels)

# Load test_bow_descs and test_labels
test_bow_descs = np.load('test_bow_descs.npy')
test_labels = np.load('test_labels.npy')
print('Found test_bow_descs & test_labels')
```

```
#<---------------------------------------Task 3 & 4---------------------------------------------->

## KNN algorithm ##
print("Starting kNN ...")
k = 2
counter = 0

for i in range(test_labels.shape[0]):
    sum_list = np.zeros(6, dtype=int)
    dists = np.sum((test_bow_descs[i] - bow_descs) ** 2, axis=1)
    sorted_ids = np.argsort(dists)
    for j in range (k):
        sum_list[int(train_labels[sorted_ids[j]])] += 1

    pred_label = np.argmax(sum_list, axis=0)
    #print('Prediction '+str(pred_label)+ 'Label '+str(test_labels[i]))
    if pred_label  == test_labels[i]:
        counter += 1
print("Success ratio "+str(100*counter/len(test_labels))+" %")


## SVM algorithm ##
print("Starting SVM ...")
#
# Create SVM classifiers
for i in range (6):
    temp_labels = np.zeros((train_labels.shape),dtype=int)
    for j in range (train_labels.shape[0]):
        if  i == train_labels[j]:
            temp_labels[j] = 1
    svm = cv.ml.SVM_create()
    svm.setType(cv.ml.SVM_C_SVC)
    svm.setKernel(cv.ml.SVM_RBF)
    svm.setTermCriteria((cv.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
    svm.trainAuto(bow_descs.astype(np.float32), cv.ml.ROW_SAMPLE, temp_labels)
    svm.save("SVM" + str(i))
    print ("Created SVM" + str(i))

# Test SVM Model
counter = 0
SVMs_list = None
for i in range (6):
    loaded_svm = cv.ml.SVM_load("SVM" + str(i))
    if SVMs_list is None:
        SVMs_list = [loaded_svm]
    else:
        SVMs_list.append(loaded_svm)

for i in range (test_bow_descs.shape[0]):
    predicted_label = classify_with_svm(test_bow_descs[i], SVMs_list)
    #print('Predicted label = ', str(predicted_label), 'Correct label = ', str(test_labels[i]))
    if predicted_label == test_labels[i]:
        counter += 1

percentage = 100 *counter/test_labels.shape[0]
print("The success ratio is: "+ str(percentage) +" %")
```

ΑΠΟΤΕΛΕΣΜΑΤΑ

Εκτελώντας το πρόγραμμα που έχει υλοποιηθεί σε Python παρατηρούμε ότι πραγματοποιούμε ταξινόμηση με υψηλό ποσοστό επιτυχίας τόσο με το μοντέλο SVM όσο και με τον αλγόριθμο kNN. Είναι εμφανές μάλιστα ότι το μοντέλο SVM παρέχει υψηλότερη ακρίβεια σε σχέση με τον kNN.Αξίζει να σημειωθεί, επίσης, ότι καίριο ρόλο στην επιτυχία του μοντέλου μας διαδραματίζει η επιλογή παραμέτρων, όπως ο αριθμός των οπτικών λέξεων, ο αριθμός των πλησιέστερων γειτόνων καθώς και ο τύπος του πυρήνα (kernel) που χρησιμοποιείται στον SVM.

Όπως είναι ορατό από τα κάτωθι αποτελέσματα λαμβάνουμε υψηλότερα ποσοστά ακρίβειας στον kNN για αριθμό οπτικών λέξεων ίσο με 100, ενώ στον SVM για αριθμό λέξεων 200.

Η επιλογή της παραμέτρου είναι ορατό ότι επηρεάζει άμεσα το αποτέλεσμα του kNN και όσο μικρότερη τιμή έχει τόσο καλύτερα αποτελέσματα παίρνουμε.Η καλύτερη δυνατή τιμή της παραμέτρου k είναι k=2.

Τέλος, αναφορικά με την επίδραση του kernel παρατηρούμε ότι ο για επιλογή kernel RBF ή CHI2 που έχουν παρεμφερή λειτουργία λαμβάνουμε τη μεγαλύτερη ακρίβεια. Επίσης, ικανοποιητικά αποτελέσματα παρέχει και ο INTER, ενώ ο LINEAR υστερεί σημαντικά σε σχέση με τους υπόλοιπους λόγω της μη γραμμικής διαχωρισημότητας των δεδομένων.

Συνολικά επιλέγοντας αριθμό λέξεων ίσο με 100, παράμετρο k=2 και kernel RBF ή CHI2 λαμβάνουμε γρήγορα αποτελέσματα υψηλής ακρίβειας και με τους δύο αλγορίθμους.

- Για αριθμό οπτικών λέξεων ίσο με 50:

K=2	**83.71278458844134**
K=3	77.75831873905429
K=5	74.78108581436076
K=10	71.97898423817864

RBF	**91.06830122591944**
CHI2	**92.29422066549913**
INTER	75.30647985989492
LINEAR	17.513134851138354
SIGMOID	19.439579684763572

- Για αριθμό οπτικών λέξεων ίσο με 100:

K=2	**85.63922942206655**
K=3	79.50963222416813
K=5	77.75831873905429
K=10	71.62872154115587

RBF	**94.57092819614711**
CHI2	**95.97197898423818**
INTER	87.04028021015762
LINEAR	26.26970227670753
SIGMOID	18.21366024518389

- Για αριθμό οπτικών λέξεων ίσο με 200:

K=2	**82.31173380035027**
K=3	81.43607705779334
K=5	76.70753064798599
K=10	70.75306479859896

RBF	**95.6217162872154**
CHI2	**98.9492119089317**
INTER	98.42381786339755
LINEAR	40.4553415061296
SIGMOID	18.38879159369527

Σχετική Βιβλιογραφία

1. Διαφάνειες Μαθήματος " Όραση Υπολογιστών ", καθηγητή Ιωάννη Πρατικάκη

2. https://en.wikipedia.org/wiki/Multiclass_classification

3. https://en.wikipedia.org/wiki/Support-vector_machine

4. https://en.wikipedia.org/wiki/Bag-of-words_model_in_computer_vision

5. https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb

6.https://docs.opencv.org/3.4/d1/d2d/classcv_1_1ml_1_1SVM.html#aad7f1aaccced3c33bb256640910a0e56