Σπυρίδων Σιλίρας – 3330

Μηχανική Μάθηση : 1η Εργασία

Γενικές Σημειώσεις:

Καταρχήν, σε όλες τις μεθόδους ταξινόμησης έχω φορτώσει τα αρχεία σύμφωνα με τις οδηγίες της ιστοσελίδας – οδηγού που μας υποδείξατε. Οι εικόνες έχουν την διάσταση 28x28 pixels και για να γινεί η σωστή μελέτη, τα δεδομένα (είτε εκπαίδευσης, είτε ελέγχου), έχουν μετατραπεί με την εντολή reshape(), σε μονοδιάστατα στοιχεία όπου η κάθε εικόνα αποτελείται από 28x28 = 784 τιμές, όπου είναι και η διάσταση των δεδομένων (x = (x1, x2, … , x784)). Τώρα, ανάλογα και με τις απαιτήσεις τις κάθε μεθόδου ταξινόμησης έχει γίνει και μείωση αυτής της διάστασης (για λόγους υπολογιστικού κόστους). Για τον υπολογισμό των στατιστικών έχει χρησιμοποιηθεί η βιβλιοθήκη sklearn και συγκεκριμένα οι μέθοδοι accuracy_score, recall_score, precision_score και f1_score, όπου το accuracy_score αφορά την ακρίβεια όλου του μοντέλου για κάθε μία από τις 10 κατηγορίες, ενώ τα recall_score, precision_score και f1_score, τυπώνουν ένα πίνακα με 10 στοιχεία όπου κάθε στοιχείο αφορά την αντίστοιχη τιμή της κάθε κατηγορίας.

1ο Πείραμα (k-NN Classifier):

Έχω χρησιμοποιήσει τον KNeighborsClassifier() της βιβλιοθήκης **sklearn** για την εκπόνηση αυτού του πειράματος, στην οποία ο αριθμός των γειτόνων που θα ληφθούν υπόψιν κατά την ταξινόμηση καθορίζεται από την παράμετρο n_neighbors και ο τρόπος υπολογισμού της απόστασης δίνεται στην παράμετρο metric (στην περίπτωσή μας by default υπολογισμός της απόστασης γίνεται με την χρήση της Ευκλείδειας απόστασης και αν θέλουμε συνημιτονοειδή απόσταση, θέτουμε metric = ‘cosine’). Τα ποτελέσματα που παρατηρήθηκαν δίνονται παρακάτω:

1) **n_neighbors = 1**

	Euclidean	**Cosine**
Accuracy	84.87 %	85.76 %
F1 Score	[0.78934386 0.97891566 0.75519073 0.86823289 0.75166411 0.92299465 0.61226508 0.92495127 0.97012658 0.92980769]	[0.80705191 0.98495486 0.7573201 0.88854806 0.76467734 0.91096634 0.65703971 0.92074364 0.96878147 0.91560585]

2) n_neighbors = 5

	Euclidean	**Cosine**
Accuracy	85.54 %	85.78 %
F1 Score	[0.81081081 0.9772842 0.77154969 0.88114754 0.78041393 0.89934354 0.61366062 0.91786055 0.9635996 0.93076923]	[0.82369668 0.97831568 0.77270543 0.89676425 0.79074162 0.86523216 0.64735945 0.9146165 0.96942356 0.90078449]

3) n_neighbors = 9

	Euclidean	**Cosine**
Accuracy	85.19 %	85.16 %
F1 Score	[0.81119545 0.9786802 0.76807512 0.88559755 0.77363311 0.88679245 0.60874144 0.91366224 0.95959596 0.92720307]	[0.82031985 0.97679112 0.77049976 0.89346372 0.78807639 0.84610951 0.62283737 0.91039085 0.96503497 0.89594843]

<u>Συμπέρασμα:</u> Η μέθοδος έχει πολύ γρήγορο χρόνο εκτέλεσης είτε με Ευκλείδεια, είτε με συνημιτονοειδή απόσταση και σχετικά καλά αποτελέσματα ταξινόμησης στο σύνολο ελέγχου. Φαίνεται ότι, καθώς ο αριθμός των γειτόνων που λαμβάνονται υπόψιν μεγαλώνει (>5), η ακρίβεια αρχίζει να πέφτει με αργό ρυθμο.

<u>2ο Πείραμα (Neural Networks):</u>

Για την κατασκευή του νευρωνικού δικτύου έχω χρησιμοποιήση τις μεθόδους τις βιβλιοθήκης tensorflow. Συγκεκριμένα:

Επίπεδο εισόδου : Flatten(input_shape=(28,28))

Κρυμμένα επίπεδα : Dense(number_of_neurons, activation)

Επίπεδο εξόδου : Dense(10, activation = softmax)

Τα αποτελέσματα που παρατηρήθηκαν δεδομένου του αριθμό των επιπέδων και των κρυμμένων νευρώνων που δόθηκαν στην εκφώνηση και μέθοδο εκπαίδευσης stochastic gradient descent παρουσιάζονται παρακάτω (αριθμός εποχών = 40):

1) **1 κρυμμένο επίπεδο με 500 κρυμμένους νευρώνες και συνάρτηση ενεργοποίησης τη σιγμοειδή συνάρτηση:**

Accuracy = 84.84 %

F1 Score = [0.8137865 0.96718829 0.74536585 0.861 0.75272008 0.93004529

0.60263447 0.91579472 0.94233613 0.93591654]

2) **2 κρυμμένα επίπεδα με 500 και 200 κρυμμένους νευρώνες στο πρώτο και στο δεύτερο κρυμμένο επίπεδο αντίστοιχα και συνάρτηση ενεργοποίησης τη σιγμοειδή συνάρτηση:**

Accuracy = 84.33 %

F1 Score = [0.81313375 0.9660071 0.73850868 0.85179714 0.74092742 0.925907

0.60386965 0.91392031 0.93983093 0.93366337]

Συμπέρασμα: Και τα δύο νευρωνικά δίκτυα δίνουν καλά αποτελέσματα στο σύνολο ελέγχου. Να σημειωθεί ότι ακόμα και μετά από 40 εποχές περάσματος των δεδομένων εκπαίδευσης, το συνολικό σφάλμα συνεχίζει να μειώνεται αλλά με πάρα πολύ αργό ρυθμό και ουσιαστικά να μην παρατηρείται καμία σημαντική αλλαγή στην ακρίβεια των νευρωνικών δικτύων στο σύνολο ελέγχου. Τέλος, χρησιμοποιώντας μια παραλλαγή της stochastic gradient descent ως μέθοδο εκπαίδευσης, την adam optimizer, η ακρίβεια του νευρωνικού δικτύου αγγίζει το 90%.

3ο Πείραμα (Support Vector Machines):

Για την κατασκευή των Support Vector Machines, έχω χρησιμοποιήσει μεθόδους της βιβλιοθήκης sklearn. Επειδή θέλουμε oneVsAll classifier, έχω χρησιμοποιήσει τη μέθοδο OneVsRestClassifier(). Ανάλογα με την συνάρτηση πυρήνα που θέλουμε να έχουμε, δίνουμε ως παράμετρο στην μέθοδο SVC(), kernel = ‘linear’, kernel = ‘rbf’ και kernel = cosine_similarity για γραμμική, Gaussian και συνημιτονοειδή αντίστοιχα. Στο πείραμα αυτό είναι αναγκαίο να μειώσουμε τις διαστάσεις των εικόνων για να μειωθεί όσο το δυνατόν περισσότερο ο χρόνος εκτέλεσης. Η μείωση της διάστασης επιτυγχάνεται με την χρήση της μεθόδου resize της βιβλιοθήκης skimage. (Νέα διάσταση, 14x14 = 196). Επίσης, χρησιμοποιώ τα 40000 από τα 60000 δεδομένα εκπαίδευσης. Τα αποτελέσματα των μεθόδων:

1) **Kernel = ‘linear’**

Accuracy = 82.85 %

F1 Score = [0.79685194 0.93891974 0.70086338 0.82036503 0.73775216 0.92004049 0.55770293 0.91448344 0.9294809 0.94452774]

2) **Kernel = 'rbf' (Gaussian)**

Accuracy = 86.51 %

F1 Score = [0.81218781 0.96579477 0.78111166 0.86073501 0.78574963 0.95209279 0.62998921 0.93287266 0.96150049 0.95347674]

3) **Kernel = cosine_similarity (Cosine)**

Accuracy = 82.08 %

F1 Score = [0.79108095 0.93265823 0.69769821 0.80374707 0.71317118 0.93106996 0.51992643 0.91151755 0.92891918 0.92993631]

Συμπέρασμα: Μεταξύ των $3^{ων}$ συναρτήσεων πυρήνα, καλύτερη αποδεικνύεται η Gaussian,η οποία όχι μόνο πετυχαίνει μεγαλύτερο ποσοστό ακρίβειας, αλλά εκτελείται και σε συντομότερο χρονικό διάστημα σε σχέση με τις άλλες δύο.

4ο Πείραμα (NaiveBayes Classifier):

Για την δημιουργία του ταξινομητή έχω ακολουθήσει τις οδηγίες για την κατασκευή του. Έχω κατασκευάσει δύο συναρτήσεις που υλοποιούν τον ταξινομητή, compute_mean_varriance() που υπολογίζει την μέση τιμή, και διακύμανση για κάθε μία από τις 10 κατηγορίες των δεδομένων εκπαίδευσης. Για τον υπολογισμό τους επίσης, πρέπει να σημειωθεί η παρατήρηση ότι, εφόσον τα παραδείγματα εκπαίδευσης είναι 60000 και οι κατηγορίες είναι ισοπίθανα κατανεμημένες, συμπεραίνουμε ότι σε κάθε κατηγορία αντιστοιχούν 6000 παραδείγματα. Έπειτα, έχω δημιουργήσει την συνάρτηση fit() η οποία εισάγει στον πίνακα pred τις προβλέψεις του ταξινομητή για κάθε ένα από τα παραδείγματα ελέγχου. Σ' αυτό το σημείο να σημειώσω ότι παρατηρήθηκε κάποιες τιμές στον πίνακα variance βρέθηκαν να είναι 0, με αποτέλεσμα να μην ορίζεται η τετραγωνική ρίζα του variance. Σ αυτήν την περίπτωση απλά προσέθεσα το 0 στο ήδη υπάρχον άθροισμα. Τέλος, μετά από αρκετά πειράματα συνειδητοποίησα ότι όσο ρίχνω τις διαστάσεις των εικόνων τόσο καλύτερα και πιο γρήγορα είναι τα αποτελέσματα του ταξινομητή. Έτσι, φαίνεται η διάσταση 7*7 = 49 να είναι η καλύτερη περίπτωση. Τα αποτελέσματα για 7*7 είναι:

Accuracy = 39.27 %

F1 Score = [0.25876461 0.73514077 0.13262599 0.45905421 0.39536303 0.31711586 0.0757156 0.49896337 0.0099108 0.59047619]

Επίλογος: Το καλύτερο αποτέλεσμα στο σύνολο ελέγχου μας το δίνει η μέθοδος των Support Vector Machines με Gaussian συνάρτηση πυρήνα, η οποία είναι και

πολύ γρήγορη στην εκτέλεση της. Επίσης αρκετά κοντά της βρίσκεται και η μέθοδος k-Nearest-Neighbors με αριθμό γειτόνων ίσο με 5 και συνημιτονοειδή συνάρτηση απόστασης, η οποία είναι και η πιο γρήγορη σε σχέση με όλες τις μεθόδους που παρατηρήθηκαν.

ΠΑΡΑΡΤΗΜΑ (Πηγαίος Κώδικας):

- K-Nearest-Neighbors Classifier:

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from skimage.transform import resize
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names =
['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle
boot']

#normalize the data
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
train_images = train_images/255.0
test_images = test_images/255.0

#reshape train & test data so they can be used in KNeighborsClassifier
```

```
train_images = train_images.reshape(train_images.shape[0],
train_images.shape[1]*train_images.shape[2])
test_images = test_images.reshape(test_images.shape[0],
test_images.shape[1]*test_images.shape[2])

neigh = KNeighborsClassifier(n_neighbors=17, metric='cosine')
neigh.fit(train_images, train_labels)

pred = neigh.predict(test_images)

print('accuracy = ', accuracy_score(test_labels, pred))

print('recall = ', recall_score(test_labels, pred, average=None))
print('precision = ', precision_score(test_labels, pred, average=None))
print('f1_score = ', f1_score(test_labels, pred, average=None))
```

- Multilayer Perceptron:

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation

from sklearn import preprocessing
from skimage.transform import resize
```

```
from sklearn.metrics import accuracy_score

from sklearn.metrics import recall_score

from sklearn.metrics import precision_score

from sklearn.metrics import f1_score


fashion_mnist = tf.keras.datasets.fashion_mnist


(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()


class_names =
['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle
boot']


train_images = train_images/255.0

test_images = test_images/255.0


model = tf.keras.Sequential([

        tf.keras.layers.Flatten(input_shape=(28,28)),

        tf.keras.layers.Dense(100, activation='sigmoid'),

        tf.keras.layers.Dense(50, activation='sigmoid'),

        tf.keras.layers.Dense(10, activation='softmax')

])


model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])


model.fit(train_images, train_labels, epochs=40)


results = model.evaluate(test_images, test_labels, verbose=0)


propability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
```

```
predictions = propability_model.predict(test_images)

pred = []

for i in predictions:
        pred.append(np.argmax(i))

print('accuracy = ', accuracy_score(test_labels, pred))
print('recall = ', recall_score(test_labels, pred, average=None))
print('precision = ', precision_score(test_labels, pred, average=None))
print('f1 score = ', f1_score(test_labels, pred, average=None))
```

- Support Vector Machines:

```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from skimage.transform import resize

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

fashion_mnist = tf.keras.datasets.fashion_mnist
```

```
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names =
['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle
boot']

train_images_resized = []

test_images_resized = []

#normalize the data

train_images = train_images.astype('float32')

test_images = test_images.astype('float32')

train_images = train_images/255.0

test_images = test_images/255.0

#resize to 14x14

for i in range(0, len(train_images)):

        resized = resize(train_images[i], (14,14))

        train_images_resized.append(resized)

train_images = np.array(train_images_resized)

for i in range(0, len(test_images)):

        resized = resize(test_images[i], (14,14))

        test_images_resized.append(resized)

test_images = np.array(test_images_resized)
```

```
#reshape train & test data so they can be used in KNeighborsClassifier

train_images = train_images.reshape(train_images.shape[0],
train_images.shape[1]*train_images.shape[2])

test_images = test_images.reshape(test_images.shape[0],
test_images.shape[1]*test_images.shape[2])


print(train_images.shape)

print(test_images.shape)


clf = OneVsRestClassifier(SVC(kernel=cosine_similarity)).fit(train_images[0:40000],
train_labels[0:40000])


pred = clf.predict(test_images)


print('accuracy = ', accuracy_score(test_labels, pred))


print('recall = ', recall_score(test_labels, pred, average=None))

print('precision = ', precision_score(test_labels, pred, average=None))

print('f1_score = ', f1_score(test_labels, pred, average=None))
```

- Naïve Bayes Classifier:

```
import tensorflow as tf

import math

import numpy as np

import matplotlib.pyplot as plt


from skimage.transform import resize

from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import recall_score

from sklearn.metrics import precision_score

from sklearn.metrics import f1_score


fashion_mnist = tf.keras.datasets.fashion_mnist


(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()


class_names =
['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle
boot']


train_images_resized = []

test_images_resized = []


dimensions = 7*7


#normalize the data

train_images = train_images.astype('float32')

test_images = test_images.astype('float32')

train_images = train_images/255.0

test_images = test_images/255.0


for i in range(0, len(train_images[0:60000])):

        resized = resize(train_images[i], (7,7))


        train_images_resized.append(resized)


train_images = np.array(train_images_resized)
```

```
for i in range(0, len(test_images)):
        resized = resize(test_images[i], (7,7))

        test_images_resized.append(resized)

test_images = np.array(test_images_resized)

#reshape train & test data so they can be used in KNeighborsClassifier
train_images = train_images.reshape(train_images.shape[0],
train_images.shape[1]*train_images.shape[2])
test_images = test_images.reshape(test_images.shape[0],
test_images.shape[1]*test_images.shape[2])

mean = []
variance = []

def compute_mean_variance():
        #countClasses = [0,0,0,0,0,0,0,0,0,0]
        values = []

        for i in range (0, 10):
                l = []
                for j in range (0, dimensions):
                        l.append(0)

                values.append(l)

        for i in range(0, len(train_images)):
                for j in range(0, dimensions):
                        values[train_labels[i]][j] += train_images[i][j]
```

```
                #countClasses[train_labels[i]] += 1

        for i in range(0, 10):
                m = []
                v = []
                for j in range(0, dimensions):
                        m.append(values[i][j] / 6000)

                mean.append(m)

                for j in range(0, dimensions):
                        v.append(((values[i][j] - mean[i][j]) ** 2) / 6000)

                variance.append(v)

compute_mean_variance()

print(test_images.shape)

pred = []

def fit():

        for i in range(0, len(test_images)):
                print(i)
                g_value = []
                for j in range(0, 10):
                        sum1 = 0
                        sum2 = 0
```

```
                    for k in range(0, dimensions):
                        if(variance[j][k] == 0):
                            sum1 += 0
                        else:
                            sum1 += math.log(math.sqrt(variance[j][k]))
                        sum2 += (((test_images[i][k] - mean[j][k]) ** 2) /
2*variance[j][k])

                    g_value.append(-sum1 - sum2)

            max_value = max(g_value)

            pred.append(g_value.index(max_value))


fit()

predictions = np.array(pred)

print('accuracy = ', accuracy_score(test_labels, predictions))

print('recall = ', recall_score(test_labels, predictions, average=None))
print('precision = ', precision_score(test_labels, predictions, average=None))
print('f1_score = ', f1_score(test_labels, predictions, average=None))
```