Dimitriou Ioannis, 3214
Antoniou Serafeim-Ilias, 2640

Machine Learning Report

1st assignment

Εισαγωγή

Στο πλαίσιο αυτής της εργασίας ασχοληθήκαμε με την πειραματική βάση δεδομένων **fashion MNIST**, η οποία περιέχει ασπρόμαυρες εικόνες ενδυματολογικού περιεχομένου χαμηλής ευκρίνειας με διάσταση 28x28. Για την υλοποίηση της άσκησης χρησιμοποιήθηκε το Google Colab (περισσότερες πληροφορίες εδώ) προκειμένου να αποφευχθούν διάφορα θέματα συμβατότητας κατά την εκτέλεση του κώδικα. Ο κώδικας είναι χωρισμένος σε 4 αρχεία Python ονομασμένα την αντίστοιχα για την εκάστοτε μέθοδο που υλοποιούν. Στη συνέχεια, αναλύονται οι αλγόριθμοι ταξινόμησης που χρησιμοποιήθηκαν, αλλά και η επίδοση τους στο δεδομένο σύνολο εκπαίδευσης με κριτήρια την ακρίβεια (accuracy) και το F1 score.

Method #1 - Nearest Neighbor k-NN with Euclidean & Cosine Distance

- Αρχεία **method1a.py** και **method1b.py**

Αρχικά, φορτώνουμε το dataset που θα βοηθήσει στην εκπαίδευση:

```
1   import tensorflow as tf
2   import numpy as np
3   from sklearn import metrics
4   from sklearn.neighbors import KNeighborsClassifier
5
6   # import dataset
7   fashion_mnist = tf.keras.datasets.fashion_mnist
8
9   # train/test images/labels are used in training and testing respectively
10  # they are in numpy array form, to see their dimension do train_images.shape e.c.
11  (train_images, train_labels), (test_images,
12                                 test_labels) = fashion_mnist.load_data()
13
14  # there are 10 labels, with these names
15  class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
16                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Τα δεδομένα χρειάζονται προεπεξεργασία πριν εκπαιδεύσουν το δίκτυο. Παρατηρούμε ότι οι τιμές των pixels κυμαίνονται από 0 έως 255, πρέπει να τις βάλουμε στο διάστημα [0, 1], οπότε τις διαιρούμε με το 255:

```
18  # process the values of dataset
19  train_images = train_images / 255.0
20  test_images = test_images / 255.0
```

Για την ευκολότερη επεξεργασία, πρέπει να μετατρέψουμε τις εικόνες των training set και testing set, από την αρχική τους μορφή που είναι τρισδιάστατοι πίνακες, σε διδιάστατους:

```
23   nsamples, nx, ny = train_images.shape
24   d2_train_dataset = train_images.reshape((nsamples, nx*ny))
25   samples, x, y = test_images.shape
26   d2_test_dataset = test_images.reshape((samples, x*y))
```

Κρατάμε ένα μικρό, αντιπροσωπευτικό τμήμα του dataset επειδή η επεξεργασία ολόκληρου του dataset έπαιρνε ώρες για την ολοκλήρωση της:

```
29   x_test = d2_test_dataset[0:1000]
30   y_test = test_labels[0:1000]
```

Έπειτα, εφαρμόζουμε τη μέθοδο Εύρεσης Κοντινότερου Γείτονα (**Nearest Neighbor k-NN**) και στα 2 Python αρχεία με τη διαφορά ότι στο method1a.py αρχείο εφαρμόζουμε τη μέθοδο 1 με Ευκλίδεια Απόσταση, ενώ στο method1b.py αρχείο εφαρμόζουμε τη μέθοδο 1 με Συνημιτονοειδή Απόσταση:

Ευκλίδεια Απόσταση:

```
34   model = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
35
36   # feed data to train the model
37   model.fit(d2_train_dataset, train_labels)
38
39   # test the trained model and get the predictions
40   prediction = model.predict(x_test)
41
42   # print metrics needed
43   accuracy = metrics.accuracy_score(y_test, prediction)
44   precision = metrics.precision_score(y_test, prediction, average='macro')
45   recall = metrics.recall_score(y_test, prediction, average='macro')
46   f1 = 2*(precision*recall)/(precision + recall)
```

Συνημιτονοειδής Απόσταση:

```
35   model = KNeighborsClassifier(n_neighbors=1, metric=distance.cosine)
36
37   # feed data to train the model
38   model.fit(d2_train_dataset, train_labels)
39
40   # test the trained model and get the predictions
41   prediction = model.predict(x_test)
42
43   # print metrics needed
44   accuracy = metrics.accuracy_score(y_test, prediction)
45   precision = metrics.precision_score(y_test, prediction, average='macro')
46   recall = metrics.recall_score(y_test, prediction, average='macro')
47   f1 = 2*(precision*recall)/(precision + recall)
```

Στο τέλος και των 2 αρχείων, αφού εφαρμοστεί η μέθοδος το μοντέλο κάνει τις απαραίτητες προβλέψεις με βάση τα δεδομένα από τα οποία “τράφηκε” και εκτυπώνονται οι μετρικές που χρειαζόμαστε για τη μελέτη της αποδοτικότητας των αλγορίθμων που είναι η ακρίβεια και το F1 score.

Για τη σωστή μελέτη του αλγορίθμου, αλλά και τη σωστή αξιολόγηση της επίδοσης του, μεταβάλλαμε τις τιμές της μεταβλητής k από 1 σε 5 και έπειτα σε 10 για να δούμε τη διακύμανση και την επιρροή που ασκεί ο αριθμός των k κοντινότερων γειτόνων στις μετρικές των αποτελεσμάτων.

Τα αποτελέσματα είναι τα ακόλουθα:

Euclidean Distance

Nearest Neighbor k-NN	Accuracy	F1 Score	Runtime
method1a.py (k = 1)	0,844	0,851117916562853	1min 48sec
method1a.py (k = 5)	0,860	0,864013592592479	2min 17sec
method1a.py (k = 10)	0,856	0,860942228659324	1min 59sec

Cosine distance

Nearest Neighbor k-NN	Accuracy	F1 Score	Runtime
method1b.py (k = 1)	0,851	0,8563754541006	36min 54sec
method1b.py (k = 5)	0,862	0,864848552368497	38min 7sec
method1b.py (k = 10)	0,858	0,860376194390656	37min 58sec

Method #2 - Neural Networks with Sigmoid Activation Function

- Αρχεία **method2a.py** και **method2b.py**

Τα αρχικά βήματα στη μέθοδο 2 μοιάζουν με αυτά της μεθόδου 1. Ξεκινάμε φορτώνοντας το dataset και τον πίνακα με τα 10 labels που αντιστοιχούν σε 10 είδη ρουχισμού:

```
7   fashion_mnist = tf.keras.datasets.fashion_mnist
8
9   # train/test images/labels are used in training and testing respectively
10  # they are in numpy array form, to see their dimension do train_images.shape e.c.
11  (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
12
13  # there are 10 labels, with these names
14  class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
15                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Διαιρούμε τις τιμές της ανάλυσης των εικόνων του dataset για να χωρέσουν στο διάστημα [0, 1]:

```
18  train_images = train_images / 255.0
19  test_images = test_images / 255.0
```

Το πρώτο σκέλος της μεθόδου 2 με βάση την εκφώνηση ζητάει να κατασκευάσουμε ένα νευρωνικό δίκτυο με 1 κρυμμένο επίπεδο και 500 κρυμμένους νευρώνες με σιγμοειδή συνάρτηση ενεργοποίησης χρησιμοποιώντας ως μέθοδο βελτιστοποίησης την Stochastic Gradient Descent. Σημαντικό κομμάτι είναι οι παράμετροι του tf.keras.layers.Dense οι οποίες μαθαίνονται κατά την εκπαίδευση. Αναλυτικότερα, η εντολή tf.keras.layers.Flatten αποτελεί το πρώτο επίπεδο του νευρωνικού και μετατρέπει τις εικόνες από έναν 28x28 2D πίνακα, σε ένα μονοδιάστατο πίνακα με 28*28=784 pixels. Αυτό το layer απλώς παραμορφώνει τα δεδομένα. Στη συνέχεια, αφού γίνουν flat τα pixels το νευρωνικό αποτελείται από 2 dense layers όπου το πρώτο περιέχει 500 νευρώνες. Κάθε node περιέχει ένα score που δείχνει σε ποια από τις 10 κατηγορίες ρούχων ανήκει η τρέχουσα εικόνα:

```
24  model = tf.keras.Sequential([
25      tf.keras.layers.Flatten(input_shape=(28, 28)),
26      tf.keras.layers.Dense(500, activation='sigmoid'),
27      tf.keras.layers.Dense(10)
28  ])
29
30  # using stohastic gradient descent (sgd) and taking accuracy into consideration
31  model.compile(optimizer='sgd',
32                loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
33                metrics=['accuracy'])
```

Στη συνέχεια, ταϊζουμε το μοντέλο με δεδομένα, εφαρμόζεται η συνάρτηση softmax και το μοντέλο κάνει τις προβλέψεις του πάνω στο dataset:

```
36  model.fit(train_images, train_labels, epochs=10)
37
38  # evaluate model and get metrics needed
39  loss, accuracy = model.evaluate(test_images,  test_labels, verbose=2)
40
41  # apply softmax function on the output layer
42  probability_model = tf.keras.Sequential([model,tf.keras.layers.Softmax()])
43
44  # test model and get predictions
45  predictions = probability_model.predict(test_images)
46  labels_predicted = np.argmax(predictions,axis = 1)
47  f1 = metrics.f1_score(test_labels,labels_predicted,average='macro')
```

Εδώ κάνουμε χρήση κατάλληλων συναρτήσεων για την απεικόνιση των αποτελεσμάτων κάτω από κάθε εικόνα, όπου κόκκινο είναι το ποσοστό λάθους στις προβλέψεις, μπλε το σωστό:

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')

num_rows = 4
num_cols = 5
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()

print('\nTest accuracy:', accuracy)
print('\nF1 score: ', f1)
```

Αντίστοιχα, ακολουθήθηκε η ίδια τακτική και στο method2b.py αρχείο, με τη διαφορά ότι εκεί προστέθηκε ένα ακόμη κρυφό dense layer (3 στο σύνολο, ενώ πριν είχαμε 2) το οποίο περιέχει 200 νευρώνες.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(500, activation='sigmoid'),
    tf.keras.layers.Dense(200, activation='sigmoid'),
    tf.keras.layers.Dense(10)
])
```

Τα αποτελέσματα των 2 παραλλαγών της μεθόδου 2 είναι τα εξής:

1 Hidden Layer with 500 Neurons

Neural Network (sigmoid activation function)	Accuracy	F1 Score	Runtime
method2a.py	0,814800024032593	0,816735123330619	1min 13sec

2 Hidden Layers with 500 Neurons & 200 Neurons respectively

Neural Network (sigmoid activation function)	Accuracy	F1 Score	Runtime
method2b.py	0,803600013256073	0,800866505719493	1min 34sec

Method #3 - Support Vector Machines (SVM)

- Αρχεία **method3a.py, method3b.py** και **method3c.py**

Τα αρχικά βήματα που ακολουθήσαμε στην υλοποίηση της τρίτης μεθόδου, είναι ίδια με προηγουμένως. Φορτώσαμε το dataset, τον πίνακα με τα labels για τα είδη ρουχισμού, μεταφέραμε τις τιμές των εικόνων στο διάστημα [0, 1], έγινε η μετατροπή του τρισδιάστατου πίνακα σε διδιάστατο. Στη συνέχεια, κρατήσαμε ένα μικρό τμήμα του dataset και χρησιμοποιήσαμε μηχανές διανυσματικής στήριξης εφαρμόζοντας γραμμική συνάρτηση πυρήνα (linear kernel) όπως μας ζητήθηκε στο πρώτο σκέλος της μεθόδου 3:

```
x_train = d2_train_dataset[0:10000]
y_train = train_labels[0:10000]
x_test = d2_test_dataset[0:10000]
y_test = test_labels[0:10000]

# Support Vector Machine model with linear kernel and one-vs-rest stategic (ovr)
# it is the same with the one-vs-all
model = svm.SVC(kernel = 'linear', decision_function_shape = 'ovr')
```

Έπειτα τροφοδοτήσαμε το μοντέλο με τα δεδομένα εκπαίδευσης, έγινε το απαραίτητο testing της απόδοσης του και τυπώθηκαν οι αντίστοιχες μετρήσεις:

method3a.py

```
model.fit(x_train,y_train)

# test the trained model and get the predictions
prediction = model.predict(x_test)

# print metrics needed
accuracy = metrics.accuracy_score(y_test, prediction)
precision = metrics.precision_score(y_test, prediction, average = 'macro')
recall = metrics.recall_score(y_test, prediction, average = 'macro')
f1 = 2*(precision*recall)/(precision + recall)

print('\nAccuracy :', accuracy)
print('\nF1 score :', f1)
```

Στο επόμενο σκέλος της μεθόδου 3 εφαρμόστηκε Gaussian συνάρτηση πυρήνα, ενώ η υπόλοιπη διαδικασία είναι ακριβώς ίδια με προηγουμένως:

method3b.py

```
# Support Vector Machine model with rbf (Gaussian) kernel and one-vs-rest stategic (ovr)
# it is the same with the one-vs-all
model = svm.SVC(kernel = 'rbf', decision_function_shape = 'ovr')
```

Σε αυτό το τμήμα της εργασίας, εφαρμόσαμε τη συνημιτονοειδή συνάρτηση πυρήνα (cosine kernel). Κρατήσαμε ένα διαφορετικό, μικρότερο τμήμα του αρχικού dataset γιατί δεν επαρκούσε η μνήμη:

method3c.py

```
x_train = d2_train_dataset[0:20000]
y_train = train_labels[0:20000]
x_test = d2_test_dataset[0:10000]
y_test = test_labels[0:10000]
```

Έπειτα εφαρμόστηκε μία δική μας συνημιτονοειδής συνάρτηση πυρήνα σε συνδυασμό με το SVM μοντέλο και την “ένας-εναντίον υπολοίπων” στρατηγική:

```
def cosine_kernel(X, Y):
    norm_1 = np.sqrt((X ** 2).sum(axis=1)).reshape(X.shape[0], 1)
    norm_2 = np.sqrt((Y ** 2).sum(axis=1)).reshape(Y.shape[0], 1)
    return X.dot(Y.T) / (norm_1 * norm_2.T)

# Support Vector Machine model with custom cosine kernel and one-vs-rest stategic (ovr)
# it is the same with the one-vs-all
model = svm.SVC(kernel = cosine_kernel, decision_function_shape = 'ovr')
```

Μετά, “ταΐσαμε” τα δεδομένα στο μοντέλο μας, δοκιμάσαμε το πλέον εκπαιδευμένο μοντέλο και τις προβλέψεις του και τέλος τυπώθηκαν οι μετρήσεις:

```
model.fit(x_train,y_train)

# test the trained model and get the predictions
prediction = model.predict(x_test)

# print metrics needed
accuracy = metrics.accuracy_score(y_test, prediction)
precision = metrics.precision_score(y_test, prediction, average = 'macro')
recall = metrics.recall_score(y_test, prediction, average = 'macro')
f1 = 2*(precision*recall)/(precision + recall)

print('\nAccuracy :', accuracy)
print('\nF1 score :', f1)
```

Αναλυτικά τα αποτελέσματα της μεθόδου 3 και των 3 παραλλαγών της είναι τα ακόλουθα:

Support Vector Machines (SVM)	Accuracy	F1 Score	Runtime
method3a.py	0,8222	0,822336125781375	1min 16sec
method3b.py	0,8531	0,852898448533705	1min 39sec
method3c.py	0,8398	0,839045774170128	29sec

Method #4 - Naive Bayes Classifier with Normal Distribution

- Αρχείο **method4.py**

Τα πρώτα βήματα υλοποίησης της μεθόδου 4 είναι όμοια με πριν, φορτώσαμε το dataset της **fashion MNIST**, έγινε εισαγωγή των labels, μετατροπή των τιμών στο πεδίο [0, 1] και η μετατροπή των πινάκων:

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

# train/test images/labels are used in training and testing respectively
# they are in numpy array form, to see their dimension do train_images.shape e.c.
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# there are 10 labels, with these names
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# process the values of dataset
train_images = train_images / 255.0
test_images = test_images / 255.0

# reshape training/testing images from 3d to 2d arrays
nsamples, nx, ny = train_images.shape
x_train = train_images.reshape((nsamples,nx*ny))
samples, x, y = test_images.shape
x_test = test_images.reshape((samples,x*y))
```

Μετέπειτα, χρησιμοποιήσαμε το Gaussian Naive Bayes μοντέλο με τη συνάρτηση GaussianNB, δώσαμε τα δεδομένα στο μοντέλο, είδαμε το testing του και τις προβλέψεις του και τυπώθηκαν τα απαραίτητα αποτελέσματα:

```
model = GaussianNB()

# feed data to train the model
model.fit(x_train,train_labels)

# test model and get predictions
predictions = model.predict(x_test)

# print metrics needed
accuracy = metrics.accuracy_score(test_labels, predictions)
precision = metrics.precision_score(test_labels, predictions, average = 'macro')
recall = metrics.recall_score(test_labels, predictions, average = 'macro')
f1 = 2*(precision*recall)/(precision + recall)

print('\nAccuracy :', accuracy)
print('\nF1 score :', f1)
```

Τα αποτελέσματα της 4ης μεθόδου είναι τα εξής:

Normal Distribution

Naive Bayes Classifier	Accuracy	F1 Score	Runtime
method4.py	0,5856	0,609820691741005	2sec

Αναλυτικά τα αποτελέσματα όλων των μεθόδων και των παραλλαγών τους είναι τα εξής:

Classification Algorithms Results

Classification Algorithm	File	Accuracy	F1 Score	Runtime
Nearest Neighbor k-NN / Euclidean	**method1a.py (k=1)**	0,844	0,851117916562853	1min 48sec
Nearest Neighbor k-NN / Euclidean	**method1a.py (k=5)**	0,860	0,864013592592479	2min 17sec
Nearest Neighbor k-NN / Euclidean	**method1a.py (k=10)**	0,856	0,860942228659324	1min 59sec
Nearest Neighbor k-NN / Cosine	**method1b.py (k=1)**	0,851	0,8563754541006	36min 54sec
Nearest Neighbor k-NN / Cosine	**method1b.py (k=5)**	0,862	0,864848552368497	38min 7sec
Nearest Neighbor k-NN / Cosine	**method1b.py (k=10)**	0,858	0,860376194390656	37min 58sec
Neural Network	**method2a.py**	0,814800024032593	0,816735123330619	1min 13sec
Neural Network	**method2b.py (K1=500, K2=200)**	0,803600013256073	0,800866505719493	1min 34sec
Support Vector Machines	**method3a.py (Linear kernel)**	0,8222	0,822336125781375	1min 16sec
Support Vector Machines	**method3b.py (Gaussian kernel)**	0,8531	0,852898448533705	1min 39sec
Support Vector Machines	**method3c.py (Cosine kernel)**	0,8398	0,839045774170128	29sec
Naive Bayes	**method4.py**	0,5856	0,609820691741005	2sec

Συνεπώς, από τη σύγκριση των αποτελεσμάτων συμπεραίνουμε ότι η πιο αποδοτική μέθοδος είναι η Nearest Neighbor k-NN με συνημιτονοειδή απόσταση και 5 κοντινότερους γείτονες.