ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ
1η Σειρά Ασκήσεων

ΟΜΑΔΑ:
Tucaliuc Agnes Monalisa, 3346
Joanis Prifti, 3321

Θέμα: Μέθοδοι ταξινόμησης δεδομένων

Παρακάτω φαίνονται τα αποτελέσματα των αλγορίθμων που είχαμε να υλοποιήσουμε:
Τις μεθόδους τις υλοποιήσαμε στο PyCharm Professional με έκδοση Python 3.7 (την κάθε μέθοδο σε ξεχωριστό αρχείο .py

	Accuracy	F1_score
Nearest Neighbor k-NN με Ευκλείδεια απόσταση (Euclidean distance)	k=1 k=5 k=10 84.97% 85.54% 85.15%	k=1 k=5 k=10 85.03% 85.46% 85.06%
Nearest Neighbor k-NN με συνημιτονοειδή απόσταση (cosine distance)	k=1 k=5 k=10 85.78% 85.78% 85.78%	k=1 k=5 k=10 85.60% 85.60% 85.60%
Neural Networks με 1 κρυμμένο επίπεδο και K=500 κρυμμένους νευρώνες	86.15%	86.12%
Neural Networks με 2 κρυμμένα επίπεδα αποτελούμενο από K1=500 και K2=200 νευρώνες	86.89%	86.95%
Support Vector Machines (SVM): γραμμική συνάρτηση πυρήνα (linear kernel)	84.63%	84.56%
Support Vector Machines (SVM): Gaussian συνάρτηση πυρήνα (kernel)	88.29%	88.24%
Support Vector Machines (SVM): συνημιτονοειδή συνάρτηση πυρήνα (cosine kernel)	84.78%	84.46%
Naïve Bayes classifier	67.29%	66.00%

Βάση αποτελεσμάτων της εκπαίδευσης παρακάτω φαίνονται ταξινομημένοι οι αλγόριθμοι βάση accuracy και f1_score (αύξουσα σειρά):

1. Naïve Bayes classifier
2. Support Vector Machines (SVM): γραμμική συνάρτηση πυρήνα (linear kernel)
3. Support Vector Machines (SVM): συνημιτονοειδή συνάρτηση πυρήνα (cosine kernel)
4. Nearest Neighbor k-NN με Ευκλείδεια απόσταση (Euclidean distance) (k=5)
5. Nearest Neighbor k-NN με συνημιτονοειδή απόσταση (cosine distance) (k=1)
6. Neural Networks με 1 κρυμμένο επίπεδο και K=500 κρυμμένους νευρώνες
7. Neural Networks με 2 κρυμμένα επίπεδα αποτελούμενο από K1=500 και K2=200 νευρώνες
8. Support Vector Machines (SVM): Gaussian συνάρτηση πυρήνα (kernel)

Παρατηρήσεις:

- Στον αλγόριθμο Nearest Neighbor k-NN με Ευκλείδεια απόσταση (Euclidean distance) μετά το training we achieved highest accuracy of 85.54% on validation data for k=5.

Ο κώδικας:

```
#https://customers.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/
#https://towardsdatascience.com/understanding-and-using-k-nearest-neighbours-aka-knn-for-classification-of-digits-a55e00cc746f
#https://www.tutorialspoint.com/scikit_learn/scikit_learn_kneighbors_classifier.htm

from tensorflow import keras
import numpy as np
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

#Downloading mnist dataset
fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) = fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255
```

```
trImages = trImages.reshape(trImages.shape[0], trImages.shape[1] *
trImages.shape[2]) #monodiastata
tImages = tImages.reshape(tImages.shape[0], tImages.shape[1] *
tImages.shape[2])

#k values
kVals = [1, 5, 10]
accuracies = []

# loop over various values of `k` for the k-Nearest Neighbor classifier
for k in kVals:
   # train the k-Nearest Neighbor classifier with the current value of `k`
   #p=2 that means eucliadian distance
   model = KNeighborsClassifier(n_neighbors=k, p=2)
   model.fit(trImages, trLabels) #fortwnoume to montelo

   predictions = model.predict(tImages) #ksekinaei thb ekpaideush

   # evaluate the model and update the accuracies list
   score=model.score(tImages, tLabels)
   print("k=%d, accuracy=%.2f%%" % (k, metrics.accuracy_score(tLabels,
predictions) * 100))
   print("k=%d, f1_score=%.2f%%" % (k, metrics.f1_score(tLabels, predictions,
average= "weighted") * 100))
   accuracies.append(score)

   # find the value of k that has the largest accuracy
   i = int(np.argmax(accuracies))
   print("k=%d achieved highest accuracy of %.2f%% on validation data" %
(kVals[i],

accuracies[i] * 100))
   # show a final classification report demonstrating the accuracy of the
classifier
   # for each of the label
   print("EVALUATION ON TESTING DATA")
   print(classification_report(tLabels, predictions)) # matrix with our all-
statistics
```

Terminal results:

k=1, accuracy=84.97%
k=1, f1_score=85.03%
k=1 achieved highest accuracy of 84.97% on validation data
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support

0	0.78	0.80	0.79	1000
1	0.98	0.97	0.98	1000
2	0.73	0.78	0.76	1000
3	0.89	0.85	0.87	1000
4	0.77	0.73	0.75	1000
5	0.99	0.86	0.92	1000
6	0.61	0.62	0.61	1000
7	0.90	0.95	0.92	1000
8	0.98	0.96	0.97	1000
9	0.90	0.97	0.93	1000
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000

k=5, accuracy=85.54%
k=5, f1_score=85.46%
k=5 achieved highest accuracy of 85.54% on validation data
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.77	0.85	0.81	1000
1	0.99	0.97	0.98	1000
2	0.73	0.82	0.77	1000
3	0.90	0.86	0.88	1000
4	0.79	0.77	0.78	1000
5	0.99	0.82	0.90	1000
6	0.66	0.57	0.61	1000
7	0.88	0.96	0.92	1000
8	0.97	0.95	0.96	1000
9	0.90	0.97	0.93	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.85	10000
weighted avg	0.86	0.86	0.85	10000

k=10, accuracy=85.15%
k=10, f1_score=85.06%
k=5 achieved highest accuracy of 85.54% on validation data
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.77	0.86	0.81	1000
1	0.99	0.96	0.98	1000

2	0.72	0.81	0.76	1000
3	0.91	0.87	0.89	1000
4	0.78	0.76	0.77	1000
5	1.00	0.81	0.89	1000
6	0.64	0.56	0.60	1000
7	0.87	0.97	0.92	1000
8	0.97	0.95	0.96	1000
9	0.89	0.96	0.93	1000
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000

Process finished with exit code 0

- Στον αλγόριθμο Nearest Neighbor k-NN με συνημιτονοειδή απόσταση (cosine distance) μετά το training we achieved highest accuracy of 85.78% on validation data for k=1.

Ο κώδικας:

```
#https://notebook.community/YorkUIRLab/eosdb/word-movers-distance-in-python
from tensorflow import keras
import numpy as np
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) = fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255

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

kVals = [1, 5, 10]
accuracies = []

# loop over various values of `k` for the k-Nearest Neighbor classifier
for k in kVals:
   # train the k-Nearest Neighbor classifier with the current value of `k`
   model = KNeighborsClassifier(metric='cosine', algorithm='brute')
   model.fit(trImages, trLabels)
```

```
    predictions = model.predict(tImages)
    # evaluate the model and update the accuracies list
    score=model.score(tImages, tLabels)
    print("k=%d, accuracy=%.2f%%" % (k, metrics.accuracy_score(tLabels,
predictions) * 100))
    print("k=%d, f1_score=%.2f%%" % (k, metrics.f1_score(tLabels, predictions,
average= "weighted") * 100))
    accuracies.append(score)

    # find the value of k that has the largest accuracy
    i = int(np.argmax(accuracies))
    print("k=%d achieved highest accuracy of %.2f%% on validation data" %
(kVals[i],

accuracies[i] * 100))
    # show a final classification report demonstrating the accuracy of the
classifier
    # for each of the label
    print("EVALUATION ON TESTING DATA")
    print(classification_report(tLabels, predictions))
```

Terminal results:

k=1, accuracy=85.78%
k=1, f1_score=85.60%
k=1 achieved highest accuracy of 85.78% on validation data
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.78	0.87	0.82	1000
1	0.99	0.97	0.98	1000
2	0.74	0.80	0.77	1000
3	0.92	0.87	0.90	1000
4	0.75	0.84	0.79	1000
5	1.00	0.76	0.87	1000
6	0.75	0.57	0.65	1000
7	0.88	0.95	0.91	1000
8	0.97	0.97	0.97	1000
9	0.84	0.98	0.90	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

k=5, accuracy=85.78%
k=5, f1_score=85.60%
k=1 achieved highest accuracy of 85.78% on validation data
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.78	0.87	0.82	1000

1	0.99	0.97	0.98	1000
2	0.74	0.80	0.77	1000
3	0.92	0.87	0.90	1000
4	0.75	0.84	0.79	1000
5	1.00	0.76	0.87	1000
6	0.75	0.57	0.65	1000
7	0.88	0.95	0.91	1000
8	0.97	0.97	0.97	1000
9	0.84	0.98	0.90	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

k=10, accuracy=85.78%
k=10, f1_score=85.60%
k=1 achieved highest accuracy of 85.78% on validation data
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.78	0.87	0.82	1000
1	0.99	0.97	0.98	1000
2	0.74	0.80	0.77	1000
3	0.92	0.87	0.90	1000
4	0.75	0.84	0.79	1000
5	1.00	0.76	0.87	1000
6	0.75	0.57	0.65	1000
7	0.88	0.95	0.91	1000
8	0.97	0.97	0.97	1000
9	0.84	0.98	0.90	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Process finished with exit code 0

- Στον αλγόριθμο Neural Networks με 1 κρυμμένο επίπεδο και K=500 κρυμμένους νευρώνες

Ο κώδικας:

```
#https://towardsdatascience.com/building-your-first-neural-network-in-tensorflow-2-tensorflow-for-hackers-part-i-e1e2f1dfe7a0
#https://becominghuman.ai/simple-neural-network-on-mnist-handwritten-digit-dataset-61e47702ed25
#https://keras.io/guides/sequential_model/
#https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/
#https://stackoverflow.com/questions/48987959/classification-metrics-cant-
```

```
handle-a-mix-of-continuous-multioutput-and-multi-la
#https://stackoverflow.com/questions/52269187/facing-valueerror-target-is-
multiclass-but-average-binary

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from tensorflow import keras
import numpy as np
from keras.layers import Dense, Flatten
from keras.utils import to_categorical

#neurons
K=500

#Downloading mnist dataset
(trImages, trLabels), (tImages, tLabels) =
keras.datasets.fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255

# Convert tLabels into one-hot format
temp = []
for i in range(len(tLabels)):
   temp.append(to_categorical(tLabels[i], num_classes=10))
tLabels = np.array(temp)

# Convert trLabels into one-hot format
temp = []
for i in range(len(trLabels)):
   temp.append(to_categorical(trLabels[i], num_classes=10))
trLabels = np.array(temp)

model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(units=K, activation='relu')) #or activation='sigmoid'
model.add(Dense(units=10, activation='softmax'))

model.summary() #prints our informantion

model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['acc'])

model.fit(trImages, trLabels, epochs=10,
          validation_data=(tImages,tLabels))

predictions = model.predict(tImages)
predictions=np.argmax(predictions, axis=1)
tLabels=np.argmax(tLabels, axis=1)

# accuracy: (tp + tn) / (p + n)
```

```
accuracy = accuracy_score(tLabels, predictions)
print('Accuracy: %f' % (accuracy*100))

# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(tLabels, predictions, average='weighted')
print('F1 score: %f' % (f1*100))
```

Terminal results:

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 500)	392500
dense_1 (Dense)	(None, 10)	5010

Total params: 397,510
Trainable params: 397,510
Non-trainable params: 0

2021-04-16 13:37:46.731612: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
2021-04-16 13:37:46.756591: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 3593085000 Hz
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.9518 - acc: 0.7050 - val_loss: 0.5587 - val_acc: 0.8109
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.5102 - acc: 0.8261 - val_loss: 0.5010 - val_acc: 0.8273
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4631 - acc: 0.8405 - val_loss: 0.4778 - val_acc: 0.8323
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4364 - acc: 0.8507 - val_loss: 0.4499 - val_acc: 0.8441
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4144 - acc: 0.8579 - val_loss: 0.4362 - val_acc: 0.8456
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4049 - acc: 0.8613 - val_loss: 0.4388 - val_acc: 0.8480
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3900 - acc: 0.8658 - val_loss: 0.4209 - val_acc: 0.8492
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3746 - acc: 0.8677 - val_loss: 0.4119 - val_acc: 0.8524

```
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3679 - acc: 0.8742 -
val_loss: 0.3981 - val_acc: 0.8606
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3580 - acc: 0.8746 -
val_loss: 0.3930 - val_acc: 0.8615
Accuracy: 86.150000
F1 score: 86.129184

Process finished with exit code 0
```

- Στον αλγόριθμο Neural Networks με 2 κρυμμένα επίπεδα αποτελούμενο από K1=500 και K2=200 νευρώνες

Ο κώδικας:

```
import numpy as np
from keras.layers import Dense, Flatten
from keras.utils import to_categorical

#neurons
K1=500
K2=200

#Downloading mnist dataset
(trImages, trLabels), (tImages, tLabels) =
keras.datasets.fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255

# Convert tLabels into one-hot format
temp = []
for i in range(len(tLabels)):
   temp.append(to_categorical(tLabels[i], num_classes=10))
tLabels = np.array(temp)

# Convert trLabels into one-hot format
temp = []
for i in range(len(trLabels)):
   temp.append(to_categorical(trLabels[i], num_classes=10))
trLabels = np.array(temp)

model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(units=K1, activation='relu'))#or activation='sigmoid'
model.add(Dense(units=K2, activation='relu'))#or activation='sigmoid'
model.add(Dense(units=10, activation='softmax'))
```

```
model.summary()

model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['acc'])

model.fit(trImages, trLabels, epochs=10,
          validation_data=(tImages,tLabels))

predictions = model.predict(tImages)
predictions=np.argmax(predictions, axis=1)
tLabels=np.argmax(tLabels, axis=1)

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(tLabels, predictions)
print('Accuracy: %f' % (accuracy*100))

# precision tp / (tp + fp)
precision = precision_score(tLabels, predictions, average='weighted')
print('Precision: %f' % (precision*100))

# recall: tp / (tp + fn)
recall = recall_score(tLabels, predictions, average='weighted')
print('Recall: %f' % (recall*100))

# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(tLabels, predictions, average='weighted')
print('F1 score: %f' % (f1*100))
```

Terminal results:

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 500)	392500
dense_1 (Dense)	(None, 200)	100200
dense_2 (Dense)	(None, 10)	2010

Total params: 494,710
Trainable params: 494,710
Non-trainable params: 0

2021-04-16 13:41:24.691102: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
2021-04-16 13:41:24.712610: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 3593085000 Hz
Epoch 1/10

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.9088 - acc: 0.7192 - val_loss: 0.5204 - val_acc: 0.8224
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4827 - acc: 0.8335 - val_loss: 0.4724 - val_acc: 0.8341
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4336 - acc: 0.8503 - val_loss: 0.4636 - val_acc: 0.8355
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4056 - acc: 0.8601 - val_loss: 0.4119 - val_acc: 0.8543
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3890 - acc: 0.8665 - val_loss: 0.4019 - val_acc: 0.8577
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3677 - acc: 0.8726 - val_loss: 0.3969 - val_acc: 0.8587
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3551 - acc: 0.8756 - val_loss: 0.3839 - val_acc: 0.8646
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3398 - acc: 0.8804 - val_loss: 0.3720 - val_acc: 0.8669
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3280 - acc: 0.8829 - val_loss: 0.3644 - val_acc: 0.8701
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3181 - acc: 0.8872 - val_loss: 0.3617 - val_acc: 0.8689
Accuracy: 86.890000
Precision: 87.091073
Recall: 86.890000
F1 score: 86.958978

Process finished with exit code 0
```

- Στον αλγόριθμο Support Vector Machines (SVM): γραμμική συνάρτηση πυρήνα (linear kernel) (slower)

Ο κώδικας:

```
#https://customers.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/
#https://towardsdatascience.com/understanding-and-using-k-nearest-neighbours-aka-knn-for-classification-of-digits-a55e00cc746f
#https://www.tutorialspoint.com/scikit_learn/scikit_learn_kneighbors_classifier.htm
#https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization
#SVM_LINEAR
from tensorflow import keras
import numpy as np
```

```
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.svm import SVC
#Downloading mnist dataset

fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) = fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255

trImages = trImages.reshape(trImages.shape[0], trImages.shape[1] *
trImages.shape[2]) #monodiastata
tImages = tImages.reshape(tImages.shape[0], tImages.shape[1] *
tImages.shape[2])

accuracies = []

model = SVC(kernel='linear', decision_function_shape='ovr') # Linear Kernel
model.fit(trImages, trLabels) #fortwnoume to montelo

predictions = model.predict(tImages) #ksekinaei thb ekpaideush

# evaluate the model and update the accuracies list
score=model.score(tImages, tLabels)
print("accuracy=%.2f%%" % (metrics.accuracy_score(tLabels, predictions) *
100))
print("f1_score=%.2f%%" % (metrics.f1_score(tLabels, predictions, average=
"weighted") * 100))
accuracies.append(score)

# show a final classification report demonstrating the accuracy of the
classifier
# for each of the label
print("EVALUATION ON TESTING DATA")
print(classification_report(tLabels, predictions)) # matrix with our all-
statistics
```

Terminal results:

accuracy=84.63%
f1_score=84.56%
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.75	0.81	0.78	1000
1	0.97	0.96	0.97	1000
2	0.73	0.77	0.75	1000
3	0.85	0.84	0.85	1000
4	0.76	0.77	0.77	1000
5	0.93	0.94	0.93	1000
6	0.63	0.56	0.59	1000

```
           7       0.92      0.93      0.93      1000
           8       0.95      0.93      0.94      1000
           9       0.95      0.94      0.95      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000


Process finished with exit code 0
```

- Στον αλγόριθμο Support Vector Machines (SVM): Gaussian συνάρτηση πυρήνα (kernel) (most slower)

Ο κώδικας:

```
#https://customers.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/
#https://towardsdatascience.com/understanding-and-using-k-nearest-neighbours-aka-knn-for-classification-of-digits-a55e00cc746f
#https://www.tutorialspoint.com/scikit_learn/scikit_learn_kneighbors_classifier.htm
#https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization
#SVM_gaussian
from tensorflow import keras
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.svm import SVC

#Downloading mnist dataset
fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) = fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255

trImages = trImages.reshape(trImages.shape[0], trImages.shape[1] *
trImages.shape[2]) #monodiastata
tImages = tImages.reshape(tImages.shape[0], tImages.shape[1] *
tImages.shape[2])

accuracies = []

model = SVC(kernel='rbf', decision_function_shape='ovr') # Gaussian Kernel
model.fit(trImages, trLabels) #fortwnoume to montelo

predictions = model.predict(tImages) #ksekinaei thb ekpaideush
```

```
# evaluate the model and update the accuracies list
score=model.score(tImages, tLabels)
print("accuracy=%.2f%%" % (metrics.accuracy_score(tLabels, predictions) *
100))
print("f1_score=%.2f%%" % (metrics.f1_score(tLabels, predictions, average=
"weighted") * 100))
accuracies.append(score)

# show a final classification report demonstrating the accuracy of the
classifier
# for each of the label
print("EVALUATION ON TESTING DATA")
print(classification_report(tLabels, predictions)) # matrix with our all-
statistics
```

Terminal results:

accuracy=88.29%
f1_score=88.24%
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.83	0.86	0.84	1000
1	0.99	0.96	0.98	1000
2	0.79	0.82	0.80	1000
3	0.87	0.89	0.88	1000
4	0.81	0.81	0.81	1000
5	0.96	0.95	0.96	1000
6	0.72	0.66	0.69	1000
7	0.93	0.95	0.94	1000
8	0.97	0.98	0.97	1000
9	0.96	0.95	0.96	1000
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

Process finished with exit code 0

- Στον αλγόριθμο Support Vector Machines (SVM): συνημιτονοειδή συνάρτηση πυρήνα (cosine kernel) (least slower)

Ο κώδικας:

```
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.metrics.pairwise import cosine_similarity

#Downloading mnist dataset
```

```
fashion_mnist = keras.datasets.fashion_mnist
(trImages, trLabels), (tImages, tLabels) = fashion_mnist.load_data()

#normalize
trImages = trImages.astype('float32')
tImages = tImages.astype('float32')
trImages=trImages/255
tImages=tImages/255

trImages = trImages.reshape(trImages.shape[0], trImages.shape[1] *
trImages.shape[2]) #monodiastata
tImages = tImages.reshape(tImages.shape[0], tImages.shape[1] *
tImages.shape[2])

accuracies = []

model = SVC(kernel = metrics.pairwise.cosine_similarity,
decision_function_shape='ovr') # Cosine Kernel
model.fit(tImages, tLabels) #fortwnoume to montelo BALAME TA TEST IMAGES ,
BECAUSE OF MEMORY ISSUES

predictions = model.predict(tImages) #ksekinaei thb ekpaideush

# evaluate the model and update the accuracies list
score=model.score(tImages, tLabels)
print("accuracy=%.2f%%" % (metrics.accuracy_score(tLabels, predictions) *
100))
print("f1_score=%.2f%%" % (metrics.f1_score(tLabels, predictions, average=
"weighted") * 100))
accuracies.append(score)

# show a final classification report demonstrating the accuracy of the
classifier
# for each of the label
print("EVALUATION ON TESTING DATA")
print(classification_report(tLabels, predictions)) # matrix with our all-
statistics
```

Terminal results:

accuracy=84.78%
f1_score=84.46%
EVALUATION ON TESTING DATA

	precision	recall	f1-score	support
0	0.78	0.84	0.81	1000
1	0.98	0.95	0.96	1000
2	0.75	0.73	0.74	1000
3	0.80	0.90	0.85	1000
4	0.71	0.80	0.75	1000
5	0.97	0.93	0.95	1000
6	0.70	0.49	0.58	1000
7	0.90	0.94	0.92	1000

```
           8       0.94      0.96      0.95      1000
           9       0.93      0.95      0.94      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.84     10000
weighted avg       0.85      0.85      0.84     10000


Process finished with exit code 0
```

- Στον αλγόριθμο Naïve Bayes classifier βάλαμε τα test images (tImages) και τα test labels (tLabels) διότι δεν μας έφτασαν οι υπολογιστικοί πόροι που διαθέταμε

Ο κώδικας:

```
#https://deeplearningcourses.com/c/data-science-supervised-machine-learning-in-python
# https://www.udemy.com/data-science-supervised-machine-learning-in-python
# This is an example of a Naive Bayes classifier on MNIST data.
#https://github.com/lazyprogrammer/machine_learning_examples/blob/master/supervised_class/nb.py
from __future__ import print_function, division
from future.utils import iteritems
from tensorflow import keras
from sklearn import metrics
import numpy as np
from scipy.stats import multivariate_normal as mvn

class NaiveBayes(object):
    def fit(self, X, Y, smoothing=1e-2):
        self.gaussians = dict()
        self.priors = dict()
        labels = set(Y)
        for l in labels:
            current_x = X[Y == l]
            self.gaussians[l] = {
                'mean': current_x.mean(axis=0),
                'var': current_x.var(axis=0) + smoothing,
            }
            self.priors[l] = float(len(Y[Y == l])) / len(Y)

    def score(self, X, Y):
        P = self.predict(X)
        return np.mean(P == Y)

    def predict(self, X):
        N, D = X.shape
        K = len(self.gaussians)
        P = np.zeros((N, K))
        for c, g in iteritems(self.gaussians):
            mean, variance = g['mean'], g['var']
            P[:, c] = mvn.logpdf(X, mean=mean, cov=variance) +
np.log(self.priors[c])
        return np.argmax(P, axis=1)
```

```
if __name__ == '__main__':
    # Downloading mnist dataset
    fashion_mnist = keras.datasets.fashion_mnist
    (trImages, trLabels), (tImages, tLabels) = fashion_mnist.load_data()

    # normalize
    trImages = trImages.astype('float32')
    tImages = tImages.astype('float32')
    trImages = trImages / 255
    tImages = tImages / 255

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

    model = NaiveBayes()
    model.fit(trImages, trLabels)
    predictions = model.predict(tImages)
    print("accuracy = %.2f%% " % (metrics.accuracy_score(tLabels, predictions)
* 100))
    print("f1_score = %.2f%%" % (metrics.f1_score(tLabels, predictions,
average="weighted") * 100))
```

Terminal results:

accuracy = 67.29%
f1_score = 66.00%

Process finished with exit code 0

Link για τα αρχεία στο google drive:
https://drive.google.com/drive/folders/1vMpcYObwkTbgErAJ9Aiyo_oxACqPmJGS?usp=sharing