

Monochrome Dreams Classification

Participanții trebuie să antreneze un model de învățare automată care să clasifice imagini alb-negru (monochrome) într-una dintre cele nouă categorii, numerotate cu etichete de la 0 la 8.

Pentru antrenare, au fost puse la dispoziție 30000 imagini cu etichete corespunzătoare. Pentru validare sunt 5000 imagini cu etichete corespunzătoare. Setul de testare este alcătuit din 5000 de imagini.

Astfel, fișierele de input sunt următoarele:

- `train.txt` – fișier metadata care conține, pe fiecare rând, denumirea fișierului de tip imagine și label-ul corespunzător, pentru fiecare dintre imaginile de antrenare.
- `validation.txt` - fișier metadata care conține, pe fiecare rând, denumirea fișierului de tip imagine și label-ul corespunzător, pentru fiecare dintre imaginile de validare.
- `test.txt` - fișier metadata care conține denumirile fișierelor de testare de tip imagine, câte unul pe fiecare rând.
- `sample_submission.txt` – exemplu de submitie în format corect.

Încărcarea datelor

Pentru a încărca imaginile în așa fel încât clasificatorii să poată prelucra datele, abordarea a fost următoarea:

```
import numpy as np
from PIL import Image
import copy
```

Condiții prealabile (librării necesare):

- **numpy** pentru colecția array
- **PIL** pentru funcția de încărcare a imaginilor .png
- **copy** pentru a face deep-copy array-urilor

```
def load_data(data_type):
    data_type = data_type.lower()
    ids = []
    samples = []
    labels = []

    path = 'data/' + data_type + '.txt'
    fin = open(path, mode='r', encoding='utf-8')
```

Îmi creez o funcție `load_data` care să îmi încarce oricare dintre seturile de date (train / validation / test), în funcție de parametrul `data_type` pe care i-l dau. Pentru setul de date respectiv, deschid fișierul metadata.

```

for line in fin.readlines():
    if data_type != 'test':
        id, label = line.split(',')
        labels.append(int(label[0]))
    else:
        id = line.replace("\n", "")
        ids.append(id)

```

Parcurg fișierul metadata linie cu linie. În cazul datelor de “train” și de “validation”, pe fiecare linie avem denumirea imaginii .png (pe care o voi numi *id*) și *label*-ul ei, separate prin virgulă. În cazul datelor de “test”, nu am decât id-urile. Populez lista de id-uri și (dacă e cazul) lista de label-uri.

```

img_path = 'data/' + data_type + '/' + id
img_obj = Image.open(img_path)

img = copy.deepcopy(np.asarray(img_obj).flatten())

img_obj.close()

samples.append(img)

```

Din folderul aferent (train / validation / test) deschid imaginea cu id-ul curent folosind `Image.open()` din librăria PIL, și o încarc într-un numpy array, iar apoi o aplatizez (flatten), pentru a obține un array unidimensional. Folosesc `deepcopy()` pentru a nu lucra mai departe pe o referință către array, ci pe o copie a sa. Populez lista de sample-uri (imagini).

Preprocesarea datelor

```

sc = StandardScaler()
samples = sc.fit_transform(samples)

```

Mă ocup de preprocesare în aceeași funcție *load_data*, înainte de a finaliza încărcarea datelor. Valorile pixelilor imaginilor variază între 0 și 255. Pentru ca modelele de învățare automată să obțină rezultate mai bune, am ales să standardizez datele folosind `StandardScaler` din `scikit learn`. `StandardScaler` centrează datele încât distribuția datelor va avea media 0 și deviația standard 1. Scorul standard al unei probe *x* se calculează prin următoarea formulă:

$$z = (x - u) / s$$

unde:

u – media valorilor probelor de antrenare

s – deviația standard a probelor de antrenare

(Precizare: pentru modelul Naive Bayes, nu am putut folosi direct `StandardScaler` deoarece are nevoie de valori pozitive)

```
print(sc.mean_)  
print(sc.scale_)
```

```
[134.4790507 130.58284724 129.13342889 ... 127.69067698 132.18556048  
134.03549882]  
[91.54921551 91.39876305 91.28154734 ... 88.91867801 89.29171088  
89.50172785]
```

Afișez în consolă o previzualizare pentru medie și deviația standard (pentru datele de antrenare).

```
samples = np.array(samples)  
labels = np.array(labels)  
  
return ids, samples, labels
```

Până acum am stocat datele în liste clasice din Python. Înainte să le returnez, le transform în numpy array-uri, întrucât acestea sunt optimizate pentru învățare automată: sunt mai rapide, consumă mai puțină memorie și au diferite funcții care facilitează lucrul cu datele.

```
train_ids, train_samples, train_labels = load_data('train')  
validation_ids, validation_samples, validation_labels = load_data('validation')  
test_ids, test_samples, test_labels = load_data('test')
```

Folosindu-mă de funcția mea, încarc toate datele de intrare ale problemei.

Configurarea unui model

Pentru obținerea unui scor cât mai bun, am testat mai mulți clasificatori, majoritatea fiind cei predați în cadrul laboratoarelor.

Am obținut următoarele rezultate, atunci când am testat pe datele de validare:

- Naive-Bayes – 39.1%
- Metoda celor mai apropiați vecini (KNN) – 57.7%
- Mașini cu vector suport – **73.7%**
- Random Forest – 61.1%
- Rețea de perceptroni – **78.4%**

În cele ce urmează, voi prezenta cele două metode care au conferit cel mai bun scor.

Rețea de perceptroni

```
from sklearn.neural_network import MLPClassifier
```

Din scikit learn am importat clasificatorul *MLPClassifier*, a cărui denumire vine de la Multi-Layer Perceptron Classifier. Acesta se bazează pe o rețea neuronală, în sarcina căreia revine procesul de clasificare.

Perceptronul este modelarea matematică a unui neuron din corpul uman, adică, în termeni de învățare automată, un clasificator liniar. Caracteristicile sale sunt:

- valorile sau stratul de input
- ponderile și bias-ul
- suma
- funcția de activare

O rețea neuronală este formată din mai mulți perceptroni pe mai multe straturi. Numărul de straturi și numărul de perceptroni este setat în funcție de problema care trebuie tratată, și pot fi ajustate modificând hiperparametrii.

```
model = MLPClassifier(hidden_layer_sizes=668, max_iter=500)
```

Instantiez un clasificator, căruia îi configurez următorii parametri:

- *hidden_layer_sizes=668* denotă faptul că am un singur strat ascuns, acela având 668 neuroni. Valoarea 668 a fost obținută cu ajutorul următoarei formule:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

N_i = number of input neurons.

N_o = number of output neurons.

N_s = number of samples in training data set.

α = an arbitrary scaling factor usually 2-10.

- *max_iter=500* setează un număr maxim de 500 epoci

Restul parametrilor primesc valoarea implicită. Printre aceștia se numără:

- *activation='relu'* adică funcția $f(x) = \max(0, x)$
- *solver='adam'* algoritm pentru optimizare eficient, care consumă puțină memorie
- *batch_size=auto* la fiecare pas se încarcă 200 de imagini
- *learning_rate='constant'* rata de învățare este constantă
- *learning_rate_init=0.001* parametrul ratei de învățare
- *early_stopping=False* nu oprim învățarea chiar dacă progresul între epoci scade semnificativ

Antrenarea și testarea performanței

```
model.fit(train_samples, train_labels)
predicted = model.predict(validation_samples)
```

Antrenez modelul pe datele de antrenare *train_samples*, cu etichetele asociate *train_labels*. Modelul antrenat va prezice etichetele pentru imaginile de validare *validation_samples*.

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

Tot din scikit learn, din modulul *metrics*, am importat funcțiile *accuracy_score* și *confusion_matrix* pentru testarea acurateței.

```
accuracy = accuracy_score(validation_labels, predicted)
print(accuracy)
```

Funcția *accuracy_score* compară label-urile corecte cu cele prezise și întoarce o rată de similaritate cuprinsă între 0 și 1. Astfel, am obținut 0.74.

```
cm = confusion_matrix(validation_labels, predicted)
print(cm)
```

Funcția *confusion_matrix* returnează un numpy array bidimensional în care *cm[i][j]* reprezintă numărul de imagini care au de fapt label-ul “i”, dar au fost prezise ca fiind “j”.

```
0.74
[[354  30  17  15  45  14  45  18  32]
 [ 14 450  10  12  10   6   5  15   5]
 [ 10  26 373  13  43  36   6  23   3]
 [ 11  20  15 369  31  32  46  30  24]
 [ 37  32  23  16 392  14   7  24   9]
 [   3   5  18  11  17 468  11  20   8]
 [ 21  23  14   3   2   7 478   6  26]
 [ 26  34  12  25  24  28  16 348   7]
 [ 24   5   0   3   4  13  57   3 468]]

Process finished with exit code 0
```

Întrucât scorul obținut este de 74%, putem observa că cele mai multe rezultate sunt pe diagonala principală a matricei, de unde reiese că majoritatea predicțiilor au avut rezultat corect. Făcând suma pe linii și pe coloane, putem vizualiza tendința modelului de a favoriza o clasă în momentul deciziei:

clasa	imagini	predicții	diferență
0	570	500	-70
1	527	625	+98
2	533	482	-51
3	578	467	-111
4	554	568	+14
5	561	618	+57
6	580	671	+91
7	520	487	-33
8	577	582	+5

Mașini cu vector suport

```
from sklearn import svm
```

Din scikit learn am importat *svm*, modul care conține mai multe metode de învățare supervizată, precum clasificatorii SVC, NuSVC și LinearSVC. De menționat este că SVM-urile au o performanță optimă atunci când primesc liste dense de sample-uri (utilizez *numpy ndarray*).

```
model = svm.SVC()
```

M-am folosit de SVC (Support Vector Classifier). Acesta este destul de bun la a efectua clasificări multi-clasă, deci este potrivit pentru situația noastră, în care avem 8 clase. Printre hiperparametrii SVC-ului se numără (cu valoarea lor implicită):

- $C=1.0$ parametru de regularizare care controlează predispunerea la clasificare greșită
- $kernel='rbf'$ specifică ce funcție kernel ('linear' / 'rbf' / 'poly' / 'sigmoid' / 'precomputed') să fie folosită atunci când datele nu sunt liniar separabile

Voi lucra cu kernelul RBF, al cărei formulă este

$$K(u, v) = \exp(-gamma * ||u - v||^2)$$

- $gamma='auto'$ coeficient kernel pentru 'rbf', 'poly' și 'sigmoid'; valoarea pentru auto este $1/num_features$
- $max_iter=-1$ se poate seta un număr maxim de iterații, by default nu există număr maxim

Antrenarea și testarea performanței

```
model.fit(train_samples, train_labels)
predicted = model.predict(validation_samples)
```

Antrenez modelul pe datele de antrenare *train_samples*, cu etichetele asociate *train_labels*. Modelul antrenat va prezice etichetele pentru imaginile de validare *validation_samples*.

```
accuracy = accuracy_score(validation_labels, predicted)
print(accuracy)
cm = confusion_matrix(validation_labels, predicted)
print(cm)
```

La fel ca la MLP, folosesc *accuracy_score* din *sklearn.metrics* pentru a măsura performanța. Astfel, cu ajutorul SVC am obținut un scor de 73.7%.

```
0.7378
[[339  26  19  20  42   4  33  46  41]
 [ 25 408  11  14  12   5   8  33  11]
 [ 14  22 386  16  36  26  10  19   4]
 [ 28  14  13 425  20  23  16  18  21]
 [ 36  18  26  28 387  14   3  25  17]
 [  7  10  18  26  16 447   6  28   3]
 [ 30  13  13   9   6   5 464   6  34]
 [ 37  14  19  30  29  17  12 350  12]
 [ 23   4   5   7   3   8  39   5 483]]

Process finished with exit code 0
```

Matricea de confuzie pentru SVC ne arată numărul de imagini clasificate corect (pe diagonala principală) sau greșit (pe celealte poziții din tablou), în funcție de clasa din care fac parte (indexul liniei) clasa prezisă (indexul coloanei).

Scrierea submișilor

```
test_labels = model.predict(test_samples)
```

La fel ca mai devreme la validare, folosesc metoda *predict* pentru a genera predicțiile modelului, de data aceasta pentru etichetele imaginilor de test.

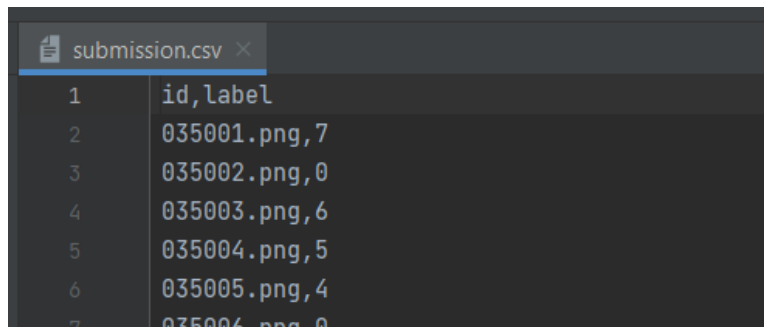
Fișierul .csv de output trebuie să conțină pe primul rând antetul *id*, *label* , iar apoi, pentru fiecare imagine de test, pe câte un rând, denumirea imaginii și label-ul prezis de către model, separate prin virgulă.

```
predicted_csv = model.predict(test_samples)

wr = open("submission.csv", 'w')

wr.write("id,label" + "\n")
pairs = zip(test_ids, predicted_csv)
for id, prediction in pairs:
    wr.write(id + ',' + str(int(prediction)) + '\n')
```

Obțin predicțiile pentru datele de testare. Scriu în fișierul submission.csv fiecare id al imaginilor de test, alături de predicția pentru label-ul acesteia.



1	id,label
2	035001.png,7
3	035002.png,0
4	035003.png,6
5	035004.png,5
6	035005.png,4
7	035006.png,0