Monochrome Dreams Classification
Documentație Proiect

Coiculescu Veronica-Ionela
Grupa 241

Proiectul urmăreşte utilizarea unor modele de machine learning pentru a clasifica corect un set de imagini alb-negru în clasele corespunzătoare. Dându-se o serie de imagini şi etichetele asociate fiecăreia, am antrenat modelele, am testat acurateţea folosind datele de validare şi am calculat predicţiile pentru datele de test.

Pentru fiecare program am importat toate pachetele de care am nevoie, apoi am citit datele de antrenare, cele de validare și de test. Din documentul "train.txt" am citit numele imaginilor aflate pe prima coloană și clasele de pe a doua coloană, folosind *loadtxt()* din pachetul numpy. Am parcurs lista cu numele, am citit imaginile aflate în fișierul "train" prin intermediul funcției *imread*("train/" + nume_imagine), iar array-ul obținut, care avea inițial dimensiunea 32x32, a fost transformat într-unul cu dimensiunea de 1x1024 pentru a putea procesa mai ușor datele și a fost adăugat în array-ul în care am stocat toate imaginile.În același mod am procedat și cu datele de validare și de test.

```
train_name_images = np.loadtxt('train.txt', 'str',
delimiter=',', usecols=0)
train_classes = np.loadtxt('train.txt', 'int', delimiter=',',
usecols=1)
train_images = []
for name in train_name_images:
    img = image.imread('train/' + name)
    arr = np.asarray(img)
    arr = np.reshape(arr, (1, 1024))
train_images.append(arr[0])
train_images = np.array(train_images)
```

Pentru a clasifica imaginile am folosit mai multe tipuri de clasificatori, dar cea mai bună acurateţe am obţinut-o cu ultimul din lista de mai jos.

I. KNearestNeighbors Classifier - Metoda celor mai apropiați vecini

Am creat clasa KnnClassifier a cărei constructor primește imaginile de antrenare și etichetele acestora, apoi am definit metoda *classify* care primește datele de test, *număr_vecini* și tipul distanței folosite în algoritm(l1 sau l2).

$$\sum_{i=1}^{n} abs(x - y) \; - \; distanța\; Manhattan(l1)$$

$$sqrt(\sum_{i=1}^{n} (x - y)^2) \; - \; distanța\; Euclidiană(l2)$$

În funcție de tipul distanței trimis ca parametru, am reținut într-un array *values* distanța de la datele de test până la fiecare dată de antrenare, am sortat crescător aceste distanțe, am luat indicii corespunzători pentru primele *număr_vecini* cele mai mici distanțe și i-am stocat în *classes_index*. Am determinat clasa care avea frecvența cea mai mare, iar apoi am returnat-o.

```
class KnnClassifier:
    def __init__(self, train_images, train_classes):
        self.train_images = train_images
        self.train_classes = train_classes
    def classify(self, test_images, num_neigh=3,
dist_type='l2'):
        n = len(self.train_images)
        values = np.zeros(n)
        if dist_type == 'l1':
            values = np.sum(abs(self.train_images -
test_images), axis=1)
        elif dist_type == 'l2':
            values = np.sqrt(np.sum((self.train_images -
test_images) ** 2, axis=1))
        classes_index = values.argsort()[:num_neigh]
        classes = self.train_classes[classes_index]
        prediction = np.bincount(classes).argmax()
        return prediction
```

Am făcut mai multe teste pentru a vedea pentru care dintre parametrii obțin o acuratețe mai bună. Am creat o listă `neighs = [3,5,7,9]` și am calculat pentru fiecare număr de vecini și cele două distanțe, obținând : *accuracy_l1= [0.4748, 0.4872, 0.4994, 0.503]* și *accuracy_l2=[0.4760 , 0.4772, 0.483, 0.4916]* . Astfel, am concluzionat că pentru datele din test, voi folosi *număr_vecini* egal cu 9 și distanța l1. Am apelat metoda din clasificator,

am reținut într-o listă fiecare precizie, iar la final am scris în fișierul *csv,* în forma cerută, pe fiecare rând numele imaginii din test și clasa obținută în urma clasificării.
Pe Kaggle am obținut acuratețea de: 0.50426.

II. SVM - Mașini cu vector suport

Pentru început am creat o funcție de normalizare a datelor de antrenare, de validare și de testare. Pe lângă aceste date, am transmis și un parametru *type* care primește tipul de normalizare pe care vrem sa o efectuăm. Pentru a standardiza datele am folosit din pachetul *sklearn, preprocessing* funcții precum *StandardScaler()*, *Normalizer()*, iar apoi *transform()*.

```
def normalize(train_data, test_data, type = None):
    norm = None
    if type == 'standard':
        norm = preprocessing.StandardScaler()
    elif type == 'l1':
        norm = preprocessing.Normalizer('l1')    # Norma L1
    elif type == 'l2':
        norm = preprocessing.Normalizer('l2') # Norma L2
    if norm is not None:
        norm.fit(train_data)
        train_data = norm.transform(train_data)
        test_data = norm.transform(test_data)
    return train_data, test_data
```

Am definit modelul *svm_model* utilizând din pachetul *sklearn, svm* și am testat atât pentru *kernel= 'linear'*, cât și pentru *kernel = 'rbf'*, cu diverse variații ale *parametrului de penalitate pentru eroare C_param* sau ale funcției *gamma* (în cazul rbf). De asemenea, am normalizat datele cu diferite tipuri de normalizări. Am antrenat modelul cu date din *train_images* și *train_classes* folosind metoda *fit()* și apoi am calculat predicțiile prin intermediul lui *predict()*.

Cu normalizare *l1*, *C_param=0.01* și *kernel= 'linear',* am obținut acuratețea *0.4204,* iar cu un *kernel = 'rbf'* și *gamma = 0.5,* acuratețea a scăzut foarte mult până la valoarea 0.1108, timpul de rulare fiind foarte mare.
În final, am ajuns la concluzia că trebuie să folosesc normalizarea standard, *kernel = 'linear'* și C = 0.01, obținând acuratețea de **0.63386**.

```
norm_train, norm_test = normalize(train_images, test_images,
"standard")

svm_model = svm.SVC(kernel = 'linear', C= 0.01)#acc 0.624
svm_model.fit(norm_train, train_classes)
predictions = svm_model.predict(norm_test)
```

III. **CNN- Convolutional neural networks(Rețele neuronale)**

Pentru început am normalizat datele; am calculat media cu ajutorul lui *mean()* și deviația standard prin intermediul lui *std()*, apoi am aplicat formula $x_{normalizat} = \frac{x - mean(x)}{deviatiaStandard}$.

Am creat un model secvențial de mai multe layere pentru care am testat diferiți parametrii, am observat rezultatele și am modificat pentru a obține o acuratețe cât mai bună. Din pachetul *keras, layers* am folosit *Conv2D*(*nr_filtre, dim_nucleu, functie_de_activare) ,* iar pentru primul layer am specificat și `input_shape=[32, 32, 1]` , imaginile având dimensiune de 32x32 și fiind alb-negru. Am folosit funcția de activare *ReLu* care transformă pixelii negativi în 0, iar pe cei pozitivi îi lasă nemodificați, precum și `padding="same"` pentru a nu fi diferențe între dimensiunea de intrare și cea de ieșire(tensorflow va adăuga zerouri pentru a completa locurile libere). Am folosit *MaxPool2D(dim)* pentru a reduce dimensiunea, evitând supra-învățarea și îmbunătății procesarea tuturor datelor, *BatchNormalization()* pentru a menține media spre 0 și dispersia spre 1, precum și *Flatten(), Dropout(rate)* - *rate* face referire la probabilitatea ca elementele să nu fie luate în considerare în timpul antrenării, *Dense(units, functie_de_activare)*.

```
model = keras.models.Sequential([
                                    keras.layers.Conv2D(32, 5,
activation="relu", padding= "same", input_shape=[32, 32, 1]),
                                    keras.layers.MaxPool2D(2),
                                    keras.layers.Conv2D(32, 5,
activation="relu", padding="same"),
                                    keras.layers.MaxPool2D(2),
                                    BatchNormalization(),
                                    keras.layers.Conv2D(128, 5,
activation="relu", padding="same"),
                                    keras.layers.MaxPool2D(2),
                                    BatchNormalization(),
                                    keras.layers.Flatten(),
```

```
                                    keras.layers.Dense(100,
activation="relu"),
                                    keras.layers.Dropout(0.25),
                                    keras.layers.Dense(10,
activation="softmax")  ])
```

Am compilat modelul cu ajutorul lui *compile()*, funcția de pierdere `loss="sparse_categorical_crossentropy"` ,și două tipuri de optimizări, *adam,* dar cu cea de-a doua am obținut cea mai bună acuratețe.

```
model.compile(
 optimizer=tf.keras.optimizers.SGD(learning_rate=0.005,
momentum=0.9),
   loss="sparse_categorical_crossentropy",
        metrics=['accuracy'])
```

Am antrenat modelul cu ajutorul lui *fit(imagini, labels, nr_epoci),* am calculat pentru diferite valori *nr_epoci,* pentru 30 nu am observat nicio îmbunătățire, iar cu 20 am obținut cea mai bună acuratețe pe setul de date primit. Pentru a calcula acuratețea și pierderea, am folosit *evaluate()* pe datele de validare, iar apoi am calculat predicțiile cu ajutorul funcției *predict()*.

```
history = model.fit(norm_train,
                    train_classes,
                    epochs= 20)
test_loss, test_acc = model.evaluate(norm_valid,
validation_classes, verbose=2)
predictions = model.predict(norm_test)
```

Am obținut cea mai bună acuratețe pe Kaggle cu acest clasificator, **0.87360.**

```
C:\Users\Veronica\Documents\cursuriFMI\IA\LABORATOARE\REZOLVAR
Accuracy score: 0.624
Confusion matrix:
[[285  49  32  21  57   6  24  48  48]
 [ 39 330  21  14  23  10  27  42  21]
 [ 19  45 341  11  51  24  18  14  10]
 [ 34  21  22 361  29  38  19  30  24]
 [ 65  32  54  31 299  10  12  31  20]
 [ 11   7  31  39  23 401   6  30  13]
 [ 38  35  19  24  11   8 401   7  37]
 [ 50  27  18  35  28  22  20 308  12]
 [ 48  19   8  23  14  15  42  14 394]]
```

Pentru a calcula acuratețea și matricea de confuzie, am folosit din pachetul *sklearn, metrics* funcțiile accuracy_score() și *confusion_matrix()*.

La final, am scris rezultatele obținute într-un fișier csv așa cum se cerea.

```
92
93  with open('sample_submission2.txt', mode='w', newline='') as file: #punem newline= ' ' pentru a nu mai afisa un rand nou gol dupa fiecare linie
94      file_wr = csv.writer(file, delimiter=',')  # datele vor fi delimitate de virgula in fisier, aflandu-se pe doua coloan
95      file_wr.writerow(['id', 'label'])
96      for i in range(nr_image): #parcurgem imaginile si le clasificam
97          prediction = classifier.classify(test_images[i], neigh, 'l1')
98          file_wr.writerow([test_name_images[i], prediction]) # apoi scriem in fisier rezultatul obtinut
99
```