Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №1**

«Логистическая регрессия в качестве нейронной сети»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2020

ХОД РАБОТЫ

**Задание.**

В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A … J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке:

* https://commondatastorage.googleapis.com/books1000/notMNIST\_large.tar.gz (большой набор данных);
* https://commondatastorage.googleapis.com/books1000/notMNIST\_small.tar.gz (маленький набор данных).

Описание данных на английском языке доступно по ссылке:

http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html

1. Загрузите данные и отобразите на экране несколько из изображений с помощью языка Python;
2. Проверьте, что классы являются сбалансированными, т.е. количество изображений, принадлежащих каждому из классов, примерно одинаково (В данной задаче 10 классов).
3. Разделите данные на три подвыборки: обучающую (200 тыс. изображений), валидационную (10 тыс. изображений) и контрольную (тестовую) (19 тыс. изображений)
4. Проверьте, что данные из обучающей выборки не пересекаются с данными из валидационной и контрольной выборок. Другими словами, избавьтесь от дубликатов в обучающей выборке.
5. Постройте простейший классификатор (например, с помощью логистической регрессии). Постройте график зависимости точности классификатора от размера обучающей выборки (50, 100, 1000, 50000). Для построения классификатора можете использовать библиотеку SkLearn ([http://scikit-learn.org](http://scikit-learn.org/)).

**Результат выполнения:**

1. Код отображения символов от А до J с одинаковым шрифтом:

fig=plt.figure(figsize=(8, 8))

columns = 5

rows = 2

for i in range(1, columns\*rows + 1):

path = os.path.join(TRAIN\_DATA\_PATH, letters[i - 1], 'aG9tZXdvcmsgbm9ybWFsLnR0Zg==.png')

img = mpimg.imread(path)

fig.add\_subplot(rows, columns, i)

plt.imshow(img)

plt.show()

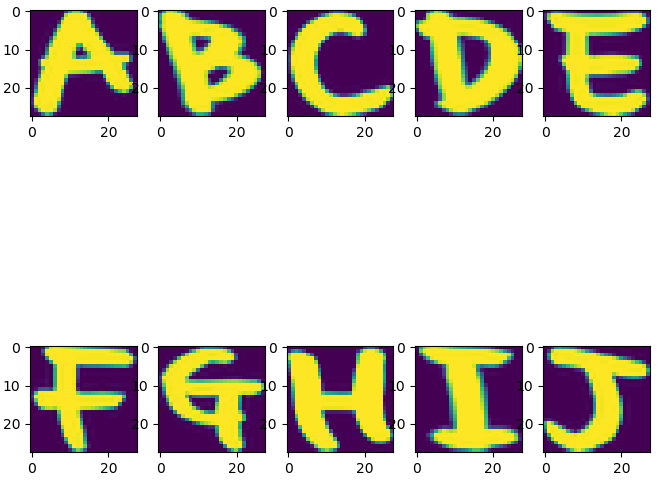
****

Рисунок 1 – результат выполнения

2. Код реализации:

fig, ax = plt.subplots()

density = []

for i in letters:

path = os.path.join(TRAIN\_DATA\_PATH, i)

density.append(len(os.listdir(path)))

print(density)

ax.bar(letters, density, width=0.5)

ax.set\_ylabel('Amount of files')

ax.set\_title('Files in each group')

plt.show()

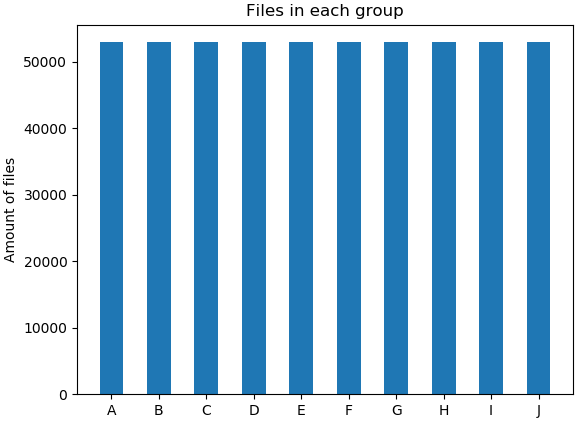


Рисунок 2 – результат выполнения

Согласно данному графику, выборка является сбалансированной.

3. Выгрузка и разделение данных:

def get\_dataset(dataset\_path):

dataset = {}

for i in letters:

dataset[i] = []

path = os.path.join(dataset\_path, i)

for file in os.listdir(path):

dataset[i].append(file)

return dataset

def cut\_dataset(dataset, from\_range, to\_range):

dataset\_copy = {}

for letter in letters:

dataset\_copy[letter] = dataset[letter][from\_range:to\_range]

return dataset\_copy

def transform\_to\_array(dataset):

X, y = [], []

for letter in letters:

for file in dataset[letter]:

X.append(file)

y.append(letter)

return X, y

X, y = transform\_to\_array(cut\_dataset(get\_dataset(TRAIN\_DATA\_PATH), 0, 21000))

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.047619, random\_state=1)

X\_test, y\_test = transform\_to\_array(get\_dataset(TEST\_DATA\_PATH))

4. Код реализации:

def get\_overlaps(X1, X2, y1, y2):

x\_1 = to\_tuple(X1)

x\_2 = to\_tuple(X2)

dict1 = {}

dict2 = {}

set1 = set()

set2 = set()

for i in range(len(x\_1)):

dict1[x\_1[i]] = y1[i]

for i in range(len(x\_2)):

dict2[x\_2[i]] = y2[i]

del x\_1

del x\_2

set1.update(dict1)

set2.update(dict2)

diff = set.difference(set1, set2)

del set1

del set2

del dict2

x = []

y = []

for i in diff:

x.append(i)

y.append(dict1[i])

return x, y

def load\_images(X, y, source):

x\_res, y\_res = [], []

for i in range(len(X)):

path = os.path.join(source, y[i], X[i])

try:

image\_data = imageio.imread(path)

x\_res.append(image\_data)

y\_res.append(y[i])

except:

print(f'Bad file: {path}')

return x\_res, y\_res

print('Files info fetched...')

X\_train, y\_train = load\_images(X\_train, y\_train, TRAIN\_DATA\_PATH)

X\_val, y\_val = load\_images(X\_val, y\_val, TRAIN\_DATA\_PATH)

X\_test, y\_test = load\_images(X\_test, y\_test, TEST\_DATA\_PATH)

print('Images loaded into memory...')

print(len(X\_train)) # 200.000

X\_without\_val, y\_without\_val = get\_overlaps(X\_train, X\_val, y\_train, y\_val)

print(len(X\_without\_val)) # 174.808

X\_without\_val\_test, y\_without\_val\_test = get\_overlaps(X\_without\_val, X\_test, y\_without\_val, y\_test)

print(len(X\_without\_val\_test)) # 171.395

np.save('X\_train', np.array(X\_without\_val\_test))

np.save('y\_train', np.array(y\_without\_val\_test))

np.save('X\_val', np.array(X\_val))

np.save('y\_val', np.array(y\_val))

np.save('X\_test', np.array(X\_test))

np.save('y\_test', np.array(y\_test))

В данном примере сначала выгружается бинарное представление картинок в память для каждого датасета. После этого ищется разница между train и val датасетами. После этого ищется разница между выборкой, где удалены дубликаты с val выборки, с тестовой выборкой. Алгоритм удаления дубликатов довольно прост: создаётся два сета, где ключом является бинарное представление картинки, а значение – буква на этой картинке. После этого ищется разница между этими сетами, которая трансформируется опять в обычный массив.

После того, как дуликаты из тренировочной выборки были удалены – все три датасета записываются как numpy массив на жесткий диск, для того, чтобы в последующих лабораторных сразу загружать готовые датасеты.

5. Код реализации:

import numpy as np

import collections

def load(conv=False):

X\_train = convert\_3d\_to\_2d(np.load('../processed/notMNIST/X\_train.npy'), conv) / 255 # with normalization

y\_train = np.load('../processed/notMNIST/y\_train.npy')

X\_test = convert\_3d\_to\_2d(np.load('../processed/notMNIST/X\_test.npy'), conv) / 255 # with normalization

y\_test = np.load('../processed/notMNIST/y\_test.npy')

# {'G': 17495, 'E': 17446, 'B': 17445, 'A': 17441, 'F': 17382, 'C': 17315, 'D': 17315, 'J': 17256, 'H': 17104, 'I': 15196}

print(collections.Counter(y\_train))

return X\_train, y\_train, X\_test, y\_test

def convert\_3d\_to\_2d(arr, conv):

return np.reshape(arr, [arr.shape[0], arr.shape[1] \* arr.shape[2]]) if not conv else np.reshape(arr, [arr.shape[0], arr.shape[1], arr.shape[2], 1])

from sklearn.linear\_model import LogisticRegression

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

import sys, os

sys.path.append(os.path.abspath(os.path.join('..', 'loaders')))

from load\_not\_mnist import load

X\_train, y\_train, X\_test, y\_test = load()

def get\_precision(k\_samples=50):

# all parameters not specified are set to their defaults

logisticRegr = LogisticRegression()

logisticRegr.fit(X\_train[0:k\_samples], y\_train[0:k\_samples])

score = logisticRegr.score(X\_test[0:k\_samples], y\_test[0:k\_samples])

return score

print(50, get\_precision(50))

print(100, get\_precision(100))

print(1000, get\_precision(1000))

print(50000, get\_precision(50000))

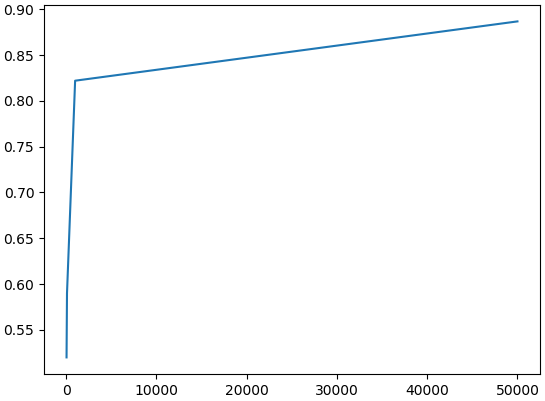


Рисунок 3 – зависимость score от размеров выборки

**Программный код:**

import os

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from sklearn.model\_selection import train\_test\_split

from goto import with\_goto

import imageio

import numpy as np

letters = ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J')

TRAIN\_DATA\_PATH = os.path.join('..', 'data', 'notMNIST\_large')

TEST\_DATA\_PATH = os.path.join('..', 'data', 'notMNIST\_small')

if not os.path.isdir(TRAIN\_DATA\_PATH) or not os.path.isdir(TEST\_DATA\_PATH):

raise Exception(f'No data found in path: {TRAIN\_DATA\_PATH}')

def to\_tuple(x):

return tuple(tuple(tuple(l2) for l2 in l1) for l1 in x)

def get\_overlaps(X1, X2, y1, y2):

x\_1 = to\_tuple(X1)

x\_2 = to\_tuple(X2)

dict1 = {}

dict2 = {}

set1 = set()

set2 = set()

for i in range(len(x\_1)):

dict1[x\_1[i]] = y1[i]

for i in range(len(x\_2)):

dict2[x\_2[i]] = y2[i]

del x\_1

del x\_2

set1.update(dict1)

set2.update(dict2)

diff = set.difference(set1, set2)

del set1

del set2

del dict2

x = []

y = []

for i in diff:

x.append(i)

y.append(dict1[i])

return x, y

def get\_dataset(dataset\_path):

dataset = {}

for i in letters:

dataset[i] = []

path = os.path.join(dataset\_path, i)

for file in os.listdir(path):

dataset[i].append(file)

return dataset

def cut\_dataset(dataset, from\_range, to\_range):

dataset\_copy = {}

for letter in letters:

dataset\_copy[letter] = dataset[letter][from\_range:to\_range]

return dataset\_copy

def transform\_to\_array(dataset):

X, y = [], []

for letter in letters:

for file in dataset[letter]:

X.append(file)

y.append(letter)

return X, y

def load\_images(X, y, source):

x\_res, y\_res = [], []

for i in range(len(X)):

path = os.path.join(source, y[i], X[i])

try:

image\_data = imageio.imread(path)

x\_res.append(image\_data)

y\_res.append(y[i])

except:

print(f'Bad file: {path}')

return x\_res, y\_res

# 1

fig=plt.figure(figsize=(8, 8))

columns = 5

rows = 2

for i in range(1, columns\*rows + 1):

path = os.path.join(TRAIN\_DATA\_PATH, letters[i - 1], 'aG9tZXdvcmsgbm9ybWFsLnR0Zg==.png')

img = mpimg.imread(path)

fig.add\_subplot(rows, columns, i)

plt.imshow(img)

plt.show()

# 2

fig, ax = plt.subplots()

density = []

for i in letters:

path = os.path.join(TRAIN\_DATA\_PATH, i)

density.append(len(os.listdir(path)))

print(density)

ax.bar(letters, density, width=0.5)

ax.set\_ylabel('Amount of files')

ax.set\_title('Files in each group')

plt.show()

# 3

X, y = transform\_to\_array(cut\_dataset(get\_dataset(TRAIN\_DATA\_PATH), 0, 21000))

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.047619, random\_state=1)

X\_test, y\_test = transform\_to\_array(get\_dataset(TEST\_DATA\_PATH))

# 4

print('Files info fetched...')

X\_train, y\_train = load\_images(X\_train, y\_train, TRAIN\_DATA\_PATH)

X\_val, y\_val = load\_images(X\_val, y\_val, TRAIN\_DATA\_PATH)

X\_test, y\_test = load\_images(X\_test, y\_test, TEST\_DATA\_PATH)

print('Images loaded into memory...')

print(len(X\_train)) # 200.000

X\_without\_val, y\_without\_val = get\_overlaps(X\_train, X\_val, y\_train, y\_val)

print(len(X\_without\_val)) # 174.808

X\_without\_val\_test, y\_without\_val\_test = get\_overlaps(X\_without\_val, X\_test, y\_without\_val, y\_test)

print(len(X\_without\_val\_test)) # 171.395

np.save('X\_train', np.array(X\_without\_val\_test))

np.save('y\_train', np.array(y\_without\_val\_test))

np.save('X\_val', np.array(X\_val))

np.save('y\_val', np.array(y\_val))

np.save('X\_test', np.array(X\_test))

np.save('y\_test', np.array(y\_test))

# 5

from sklearn.linear\_model import LogisticRegression

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

import sys, os

sys.path.append(os.path.abspath(os.path.join('..', 'loaders')))

from load\_not\_mnist import load

X\_train, y\_train, X\_test, y\_test = load()

def get\_precision(k\_samples=50):

# all parameters not specified are set to their defaults

logisticRegr = LogisticRegression()

logisticRegr.fit(X\_train[0:k\_samples], y\_train[0:k\_samples])

score = logisticRegr.score(X\_test[0:k\_samples], y\_test[0:k\_samples])

return score

"""

If we run without normalization, code will execute much longer and results

will be following: 0.44, 0.59, 0.772, 0.8857615894039735

"""

print(50, get\_precision(50)) # 0.52

print(100, get\_precision(100)) # 0.59

print(1000, get\_precision(1000)) # 0.822

print(50000, get\_precision(50000)) # 0.8867229224524674