Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:
«Машинное обучение»
По лабораторной работе №1
«Логистическая регрессия в качестве нейронной сети»

Выполнил: Сенькович Дмитрий Сергеевич
магистрант кафедры информатики
группы №858642

Проверил: Стержанов Максим Валерьевич
доцент, кандидат технических наук

Минск 2019

Оглавление

1 Постановка задачи

Имеется большой и маленький наборы изображений цифр 28x28. Требуется обучить логистическую регрессию на различных объемах данных, предварительно исследовав и очистив данные.

2 Решение задачи

2.1 Отображение случайно выбранных изображений

Изображения будем отображать в оттенках серого, выбрав по одному изображению каждой буквы:

```
large_dataset_filenames = {}

      for letter in CLASSES:

            for r, d, f in os.walk(LARGE_DATASET_NAME + letter):

                  large_dataset_filenames[letter] = f

      print('Large dataset images filenames has been read')

      small_dataset_filenames = {}

      for letter in CLASSES:

            for r, d, f in os.walk(SMALL_DATASET_NAME + letter):

                  small_dataset_filenames[letter] = f

      print('Small dataset images filenames has been read')


samples = []

for letter in CLASSES:

      index = random.randint(0, len(large_dataset_filenames[letter]) - 1)

      letter_image = io.imread(LARGE_DATASET_NAME + letter + "/" +
large_dataset_filenames[letter][index], as_gray=True)

      samples.append(letter_image)

plot_images(samples)

plt.show()
```

Полученные изображения представлены ниже:

Рисунок 2.1.1. Изображения случайно выбранных букв

2.2 Очистка данных

Изначально для каждой буквы в большом датасете порядка 52 тысяч изображений, в маленьком чуть больше 1900.

Перед очисткой загрузим изображения следующим образом:

```
t = TicToc('reading datasets')

t.tic()

large_dataset_images = {}

corrupted_filenames = {}

corrupted_files_count = 0

for letter in CLASSES:

     large_dataset_images[letter] = []

     corrupted_filenames = []

     for filename in large_dataset_filenames[letter]:

          try:


large_dataset_images[letter].append(io.imread(LARGE_DATASET_NAME
+ letter + "/" + filename, as_gray=True).ravel())

          except:

               print(letter)
```

```
                print(filename)
                corrupted_filenames.append(filename)
                corrupted_files_count += 1
     for corrupted_filename in corrupted_filenames:

large_dataset_filenames[letter].remove(corrupted_filename)
print('Large dataset images has been read. Corrupted files
count: %s' % corrupted_files_count)
small_dataset_images = {}
corrupted_files_count = 0
for letter in CLASSES:
     small_dataset_images[letter] = []
     corrupted_filenames = []
     for filename in small_dataset_filenames[letter]:
          try:

small_dataset_images[letter].append(io.imread(SMALL_DATASET_NAME
+ letter + "/" + filename, as_gray=True).ravel())
          except:
                print(letter)
                print(filename)
                corrupted_filenames.append(filename)
                corrupted_files_count += 1
     for corrupted_filename in corrupted_filenames:

small_dataset_filenames[letter].remove(corrupted_filename)
```

```
print('Small dataset images has been read. Corrupted files
count: %s' % corrupted_files_count)

t.toc()

print(t.elapsed)
```

Некоторые файлы прочитать не удается, поэтому мы просто-напросто пропускаем их.

В датасетах довольно много дубликатов, которые, для начала, мы очистим следующим образом: удалим все повторяющиеся изображения из маленького датасета в большом (таким образом, мы удаляем изображения из большого датасета):

```
t = TicToc('detecting duplicates')

t.tic()

total_duplicates_count = 0

for letter in CLASSES:

     print(letter)

     duplicates_count = 0

     duplicate_indices = set()

     for i in range(len(small_dataset_images[letter])):

          small_dataset_image = small_dataset_images[letter][i]

          for j in range(len(large_dataset_images[letter])):

               large_dataset_image =
large_dataset_images[letter][j]

               if np.array_equal(small_dataset_image,
large_dataset_image):

                    duplicate_indices.add(j)

                    duplicates_count += 1
```

```
    total_duplicates_count += duplicates_count

    print('Duplicated count %s for letter %s' %
(duplicates_count, letter))

    for duplicate_index in duplicate_indices:

        duplicate_filename =
large_dataset_filenames[letter][duplicate_index]

        os.remove(LARGE_DATASET_NAME + letter + "/" +
duplicate_filename)

print('Total duplicates count %s' % total_duplicates_count)

t.toc()

print(t.elapsed)
```

Теперь очистим дубликаты среди изображений для каждой буквы в пределах одного датасета. Попробуем очевидный способ:

```
t = TicToc('detecting duplicates in the same dataset')

t.tic()

for letter in CLASSES:

    print(letter)

    duplicate_indices = {}

    for i in range(len(small_dataset_images[letter])):

        small_dataset_image_i =
small_dataset_images[letter][i]

        for j in range(len(small_dataset_images[letter])):

            if i == j:

                continue
```

```
                small_dataset_image_j =
small_dataset_images[letter][j]

                if np.array_equal(small_dataset_image_i,
small_dataset_image_j):

                    if i not in duplicate_indices and j not in
duplicate_indices:

                        duplicate_indices[i] = set()

                    if i in duplicate_indices:

                        duplicate_indices[i].add(j)

                    elif j in duplicate_indices:

                        duplicate_indices[j].add(i)

    for key, duplicate_indices_per_image in
duplicate_indices.items():

         for duplicate_index_per_image in
duplicate_indices_per_image:

                duplicate_filename =
small_dataset_filenames[letter][duplicate_index_per_image]

                try:

                    os.remove(SMALL_DATASET_NAME + letter + "/"
+ duplicate_filename)

                except OSError:

                    pass

t.toc()

print(t.elapsed)
```

Датасет немного уменьшился, но заняло это очень много времени. При попытке очистить данные большого датасета таким образом процесс для двух букв занял более часа, после чего было решение ускорить очистку,

сравнивая изображения по значения md5 хеша, что сработало за несколько секунд:

```
import os
import hashlib


LARGE_DATASET_NAME = 'notMNIST_large/'
SMALL_DATASET_NAME = 'notMNIST_small/'
CLASSES = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']


def prefix(letter):
    return LARGE_DATASET_NAME + letter + "/"


def remove_duplicates(dir):
    unique = []
    for filename in os.listdir(dir):
        if os.path.isfile(dir + filename):
            filehash = hashlib.md5(open(dir + filename,
"rb").read()).hexdigest()
        if filehash not in unique:
            unique.append(filehash)
        else:
            os.remove(dir + filename)


for letter in CLASSES:
    print(letter)
```

```
    remove_duplicates(prefix(letter))
```

2.3 Обучение

Обучим логистическую регрессию на выборке из 200.000 букв по 20.000 каждой, выбранных случайным образом. Использовать будем LogisticRegression из библиотеки sklearn. Для ускорения обучения будем использовать значение параметра n_jobs = 4. Соответственно, и алгоритм оптимизации будем подбирать, которые способен работать параллельно, выберем алгоритм оптимизации saga. Для обучения будет использовать следующую функцию:

```
def train(total_sample_size, large_dataset_images,
small_dataset_images):

     x_train = []

     y_train = []

     sample_size = total_sample_size / len(CLASSES)

     large_dataset_indices = {}

     for letter in CLASSES:

          large_dataset_indices[letter] = set()

          while len(large_dataset_indices[letter]) <
sample_size:


large_dataset_indices[letter].add(random.randint(0,
len(large_dataset_images[letter]) - 1))

     for i in range(len(CLASSES)):

          letter = CLASSES[i]

          indices = large_dataset_indices[letter]

          for index in indices:

               image = large_dataset_images[letter][index]
```

```
                x_train.append(image)
                # y_train.append([1 if x == i else 0 for x in
range(len(CLASSES))])
                y_train.append(i)
    x_train = np.array(x_train)
    y_train = np.array(y_train)
    print(x_train.shape)
    print(y_train.shape)
    x_test = []
    y_test = []
    for i in range(len(CLASSES)):
        letter = CLASSES[i]
        for image in small_dataset_images[letter]:
            x_test.append(image)
            # y_test.append([1 if x == i else 0 for x in
range(len(CLASSES))])
            y_test.append(i)
    x_test = np.array(x_test)
    y_test = np.array(y_test)
    print(x_test.shape)
    print(y_test.shape)

    t = TicToc('learning')
    t.tic()
```

```
    # 50 iter elapsed: 11.2min finished # Accuracy ::
0.8896808860620682

    # 200 iter elapsed: 38.6min finished # Accuracy ::
0.8888035968856234

    mul_lr = linear_model.LogisticRegression(n_jobs=4,
verbose=10, max_iter=200, solver='saga').fit(x_train, y_train)

    # mul_lr = linear_model.LogisticRegression(n_jobs=4,
verbose=10, max_iter=100, solver='newton-cg').fit(x_train,
y_train)

    print("Multinomial Logistic regression Test Accuracy : ",
metrics.accuracy_score(y_test, mul_lr.predict(x_test)))

    t.toc()

    print(t.elapsed)
```

Для интереса, попробуем обучить модель сначала с максимально допустимым количеством итераций алгоритма оптимизатора 50, а затем 100. В первом случае получим accuracy 0.8896808860620682, а во втором 0.8888035968856234 со временем выполнения 11.2 и 38.6 минут. Accuracy в данном случае является подходящей оценкой качества алгоритма, т.к. данные для каждого класса сбалансированы (по 20.000 на каждую букву).

Попробовав другой алгоритм оптимизации с 100 итерациями, получили более двух часов работы без сходимости и какого-либо результата, вследствие чего было решено не продолжать эксперимент.

2.4 Проверка качества обучения в зависимости от объема обучающей выборки

Построим график обучения логистической регрессии в зависимости от объема обучающей выборки для значения размера выборки 50, 100, 1000 и 50.000:

```
sample_sizes = [50, 100, 1000, 50000]

accuracies = []
```

```
for total_sample_size in sample_sizes:

     accuracies.append(train(total_sample_size,
large_dataset_images, small_dataset_images))

plt.plot(sample_sizes, accuracies)

plt.show()
```

Ниже представлены график качества обучения:

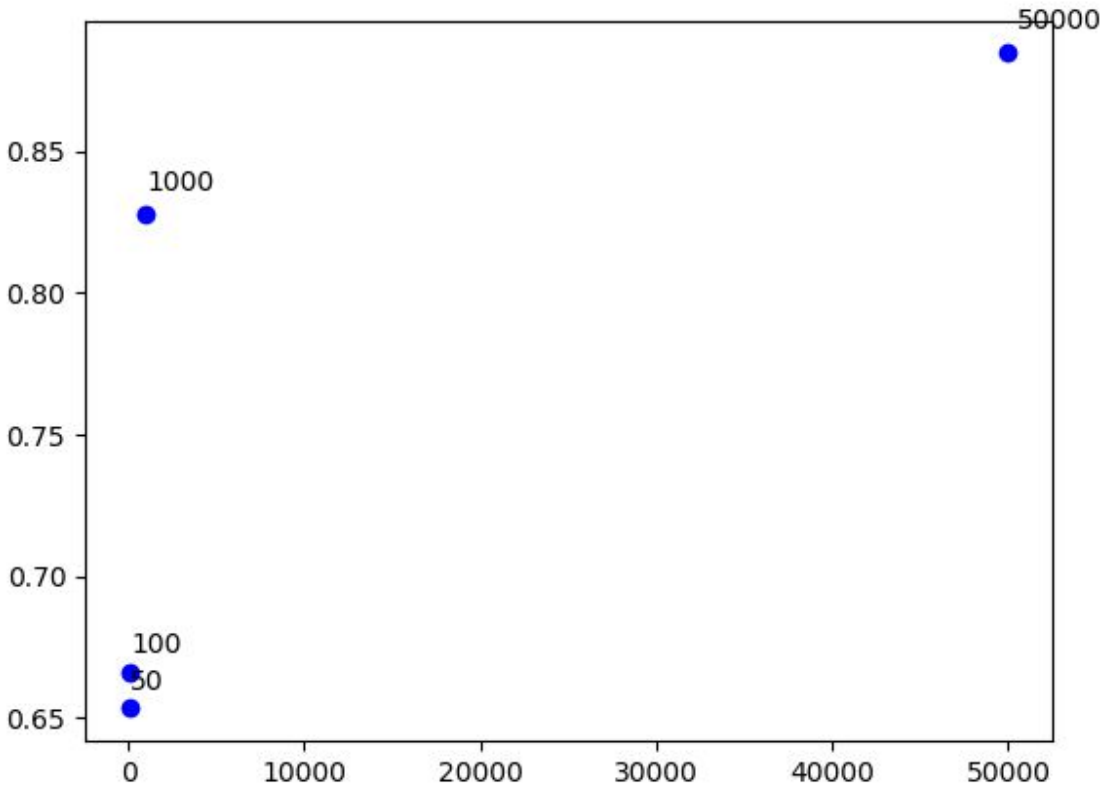

Рисунок 2.4.1. Качество обучения логистической регрессии в зависимости от объема выборки