Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей
Кафедра Информатики

Отчёт
по лабораторной работе №1:

Логистическая регрессия в качестве нейронной сети.

Студент М. С. Петрусевич
Проверил М. В. Стержанов

Минск 2020

Содержание

1 Цель работы

Изучить использование логистической регрессии в качестве нейронной сети.

2 Данные

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.

3 Ход выполнения

Для начала необходимо скачать набор данных notMNIST, который представляет из себя датасет букв латинского алфавита. За это отвечает функция prepare_dataset:

```
def prepare_dataset(url, save_as_name):
    """
    Prepares dataset to work with it
    :param url: url where resources is storing
    :param save_as_name: name of directory where it will be saved
    """
    if not os.path.exists(DATASETS_SOURCES_ROOT):
        logging.info("Dataset's source root doesn't exist,
           creating it...")
        os.mkdir(DATASETS_SOURCES_ROOT)

    if not os.path.exists(DATASETS_UNPACKED_ROOT):
        logging.info("Dataset's unpacking directory doesn't exist
           , creating it...")
        os.mkdir(DATASETS_UNPACKED_ROOT)

    logging.info("Start the dataset downloading...")
    sources_path = get_path_to_sources_dir(save_as_name)
    if os.path.exists(sources_path):
        logging.info("Dataset exists, downloading was ended...")
        return
    else:
        os.mkdir(sources_path)
        filename = fetch_dataset(url, sources_path)
        unpacked_path = get_path_to_unpacked_dir(save_as_name)
        unpack_dataset(filename, unpacked_path)
```

Отобразим данные из датасета:

Далее сбалансируем данные, т.е. убедимся, что количество данных в каждом наборе примерно одинаково. Реализуем это через структуру данных Map:

```
base = images_count[const.LEARNING_LETTERS[0]]
        for letter in const.LEARNING_LETTERS:
            if images_count[letter] - base > const.
               CLASSES_DIFFERENCE_ERROR:
                logging.error("Images have too much error!")
            logging.info("Classes have normal error. [letter s ,
               elements s]", letter, images_count[letter])
```

Получим следующий вывод:

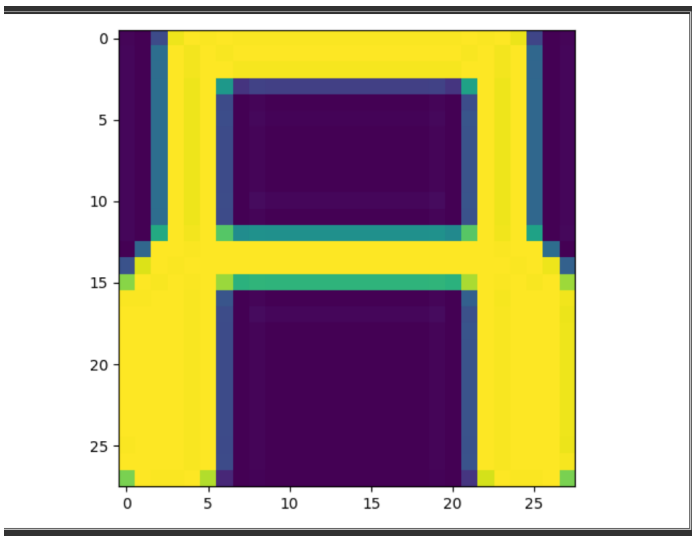

Рисунок 3.1 – Пример данных

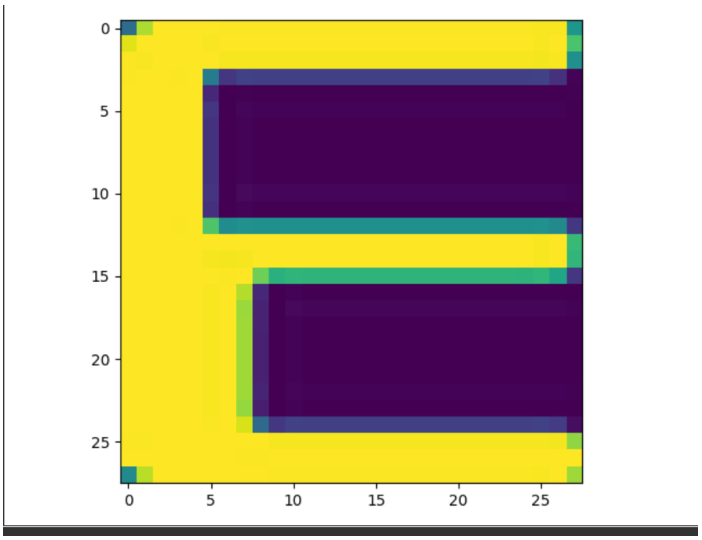

Рисунок 3.2 – Пример данных

```
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter A , elements 1872]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter B , elements 1873]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter C , elements 1873]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter D , elements 1873]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter E , elements 1873]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter F , elements 1872]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter G , elements 1872]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
```

```
   error. [letter H , elements 1872]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter I , elements 1872]
2020-04-19 00:22:55,385: INFO --- root --- Classes have normal
   error. [letter J , elements 1872]
```

Далее, разделим датасет на выборки - обучающую, валидационную и тестовую:

```
training_set = []
validation_set = []
test_set = []

logging.info("Start sorting into sets")
for letter in const.LEARNING_LETTERS:
    path_to_img_dir = get_path_to_unpacked_dir(
       usable_dataset_name) \
                      + "/" \
                      + usable_dataset_name \
                      + "/" \
                      + letter
    files_in_dir = list(map(lambda path: path_to_img_dir
       + "/" + path, os.listdir(path_to_img_dir)))
    files_count = len(files_in_dir)
    # calculate how many pics are going to each set
    to_training = int(files_count * const.
       TRAIN_SET_PERCENTS)
    to_validation = int(files_count * const.
       VALIDATION_SET_PERCENTS)
    to_test = files_count - to_training - to_validation
    # using slice put file paths to sets
    training_set = training_set + files_in_dir[0:
       to_training - 1]
    validation_set = validation_set + files_in_dir[
       to_training: to_training + to_validation - 1]
    test_set = test_set + files_in_dir[to_training +
       to_validation: to_training + to_validation +
       to_test - 1]


logging.info(
    "Total set was separted to 3 sets: training (%d - %d
       %%), validation (%d - %d %%) and test (%d - %d %%)
       ",
    len(training_set), const.TRAIN_SET_PERCENTS * 100,
    len(validation_set), const.VALIDATION_SET_PERCENTS *
       100,
    len(test_set), const.TEST_SET_PERCENTS * 100
```

```
)
```

Избавимся от дубликатов в выборках:

```
uniq_images = {}
duplicate_images = {}
logging.info("Start deleting duplicates...")
for letter in const.LEARNING_LETTERS:
    path_to_img_dir = get_path_to_unpacked_dir(
       usable_dataset_name) \
                      + "/" \
                      + usable_dataset_name \
                      + "/" \
                      + letter
    files_in_dir = os.listdir(path_to_img_dir)
    for file in files_in_dir:
        full_path = path_to_img_dir + "/" + file
        hash = calc_file_hash(full_path)
        if hash not in uniq_images:
            uniq_images[hash] = full_path
        else:
            duplicate_images[hash] = full_path

logging.info("Was found d duplicate images, total count
   of unique images = d", len(duplicate_images),
              len(uniq_images))

logging.info("Delete duplicated images from the sets")
for non_uniq_hash in duplicate_images:
    non_uniq_path = duplicate_images[non_uniq_hash]
    if non_uniq_path in training_set:
        training_set.remove(non_uniq_path)
    if non_uniq_path in validation_set:
        validation_set.remove(non_uniq_path)
    if non_uniq_path in test_set:
        test_set.remove(non_uniq_path)

logging.info(
    "Total count of sets after duplicate deleting:
       training: d ; validation: d ; test: d",
    len(training_set),
    len(validation_set),
    len(test_set)
)
```

Увидим следующий вывод:

```
2020-04-19 00:22:55,433: INFO --- root --- Total set was separted
    to 3 sets: training (11220 - 60 %), validation (3730 - 20 %)
   and test (3744 - 20 %)
```

```
2020-04-19 00:22:55,433: INFO --- root --- Start deleting
   duplicates...
2020-04-19 00:23:05,754: INFO --- root --- Was found 236
   duplicate images, total count of unique images = 18232
2020-04-19 00:23:05,754: INFO --- root --- Delete duplicated
   images from the sets
2020-04-19 00:23:05,819: INFO --- root --- Total count of sets
   after duplicate deleting: training: 11150 ; validation: 3670 ;
    test: 3638
```

Далее натренируем логистическую регресси на выборках, для этого будем использовать библиотеку SkLearn. Обучение модели на n примерах описано в функции train_on_n_examples:

```
    def train_on_n_examples(self, x_train, x_test, y_train,
       y_test):
        train_examples_count = x_train.shape[1]
        test_examples_count = x_test.shape[1]
        logistic_regression = LogisticRegression(max_iter=1000,
           tol=1e-2, C=0.5, solver='liblinear',
                                                 penalty='l1')  #
                                                     1 mln
        logistic_regression.fit(x_train.T, y_train)
        logging.info("Model has been trained successfully!")

        score_result = logistic_regression.score(x_test.T, y_test
           )
        score_result_2 = logistic_regression.score(x_train.T,
           y_train)
        logging.info("Train examples count: d",
           train_examples_count)
        logging.info("Test examples count: d",
           test_examples_count)
        logging.info("Score on test data: f", score_result)
        logging.info("Score on train data: f", score_result_2)
        return (train_examples_count, score_result)
```

Результат выполнения обучения будет выведен в консоль:

```
2020-04-19 00:23:08,747: INFO --- root --- Train examples count:
   50
2020-04-19 00:23:08,747: INFO --- root --- Test examples count:
   3638
2020-04-19 00:23:08,747: INFO --- root --- Score on test data:
   0.619571
2020-04-19 00:23:08,747: INFO --- root --- Score on train data:
   1.000000
2020-04-19 00:23:08,768: INFO --- root --- Model has been trained
    successfully!
```

```
2020-04-19 00:23:08,782: INFO --- root --- Train examples count:
   100
2020-04-19 00:23:08,782: INFO --- root --- Test examples count:
   3638
2020-04-19 00:23:08,782: INFO --- root --- Score on test data:
   0.728422
2020-04-19 00:23:08,782: INFO --- root --- Score on train data:
   1.000000
2020-04-19 00:23:09,086: INFO --- root --- Model has been trained
    successfully!
2020-04-19 00:23:09,105: INFO --- root --- Train examples count:
   1000
2020-04-19 00:23:09,105: INFO --- root --- Test examples count:
   3638
2020-04-19 00:23:09,105: INFO --- root --- Score on test data:
   0.841396
2020-04-19 00:23:09,105: INFO --- root --- Score on train data:
   1.000000
2020-04-19 00:23:11,881: INFO --- root --- Model has been trained
    successfully!
2020-04-19 00:23:11,944: INFO --- root --- Train examples count:
   11150
2020-04-19 00:23:11,944: INFO --- root --- Test examples count:
   3638
2020-04-19 00:23:11,944: INFO --- root --- Score on test data:
   0.884827
2020-04-19 00:23:11,944: INFO --- root --- Score on train data:
   0.924574
```

Продемонстрируем это на графике:

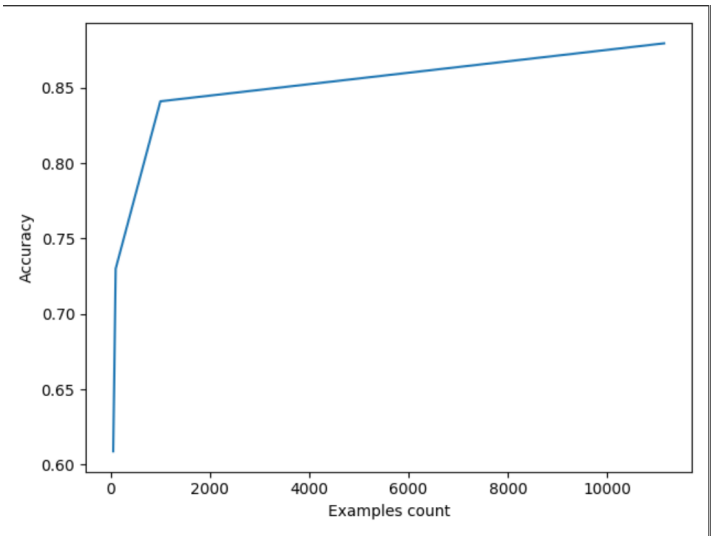

Рисунок 3.3 – График обучения