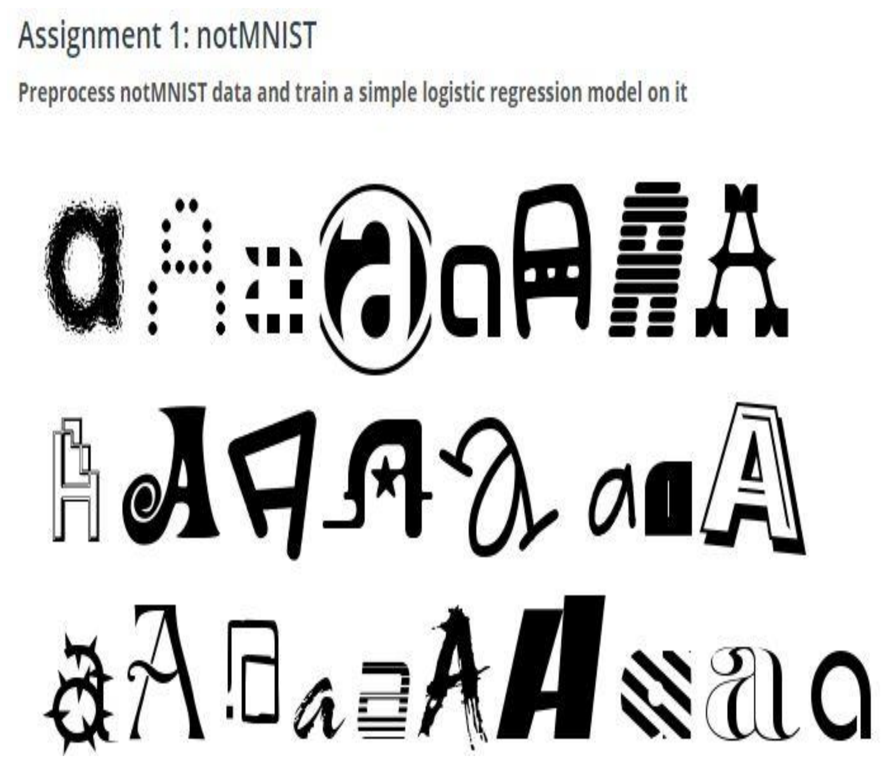
UDACITY Deep Learning
2강

"Assignment : notMNIST"

발표자 : 홍정우

– Getting Started with notMNIST

Bio Keras

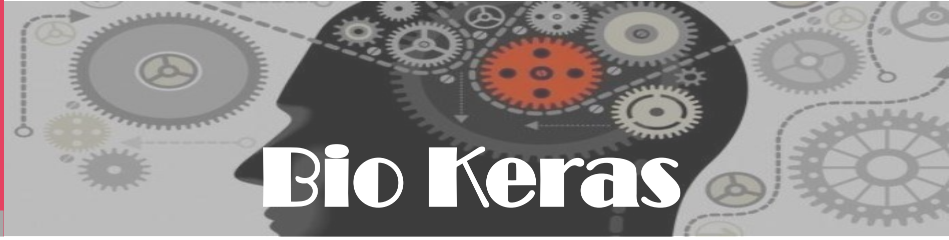

Assignment 1: notMNIST

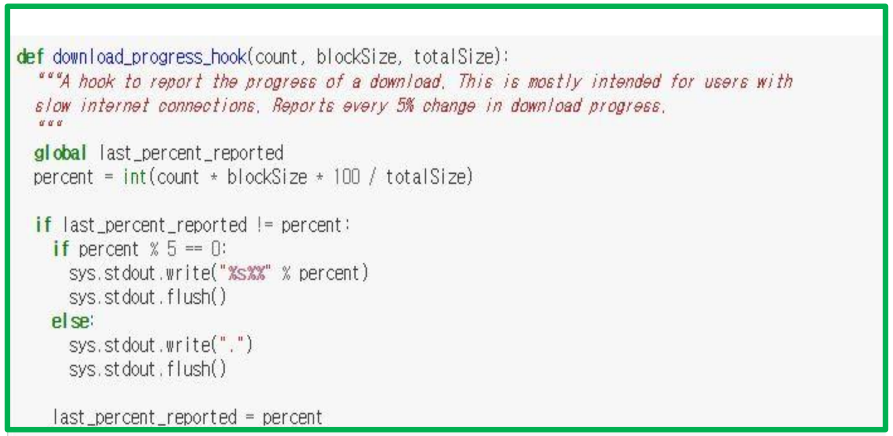
Preprocess notMNIST data and train a simple logistic regression model on it

"그림에 원본 코드가 링크 되어 있어요!"

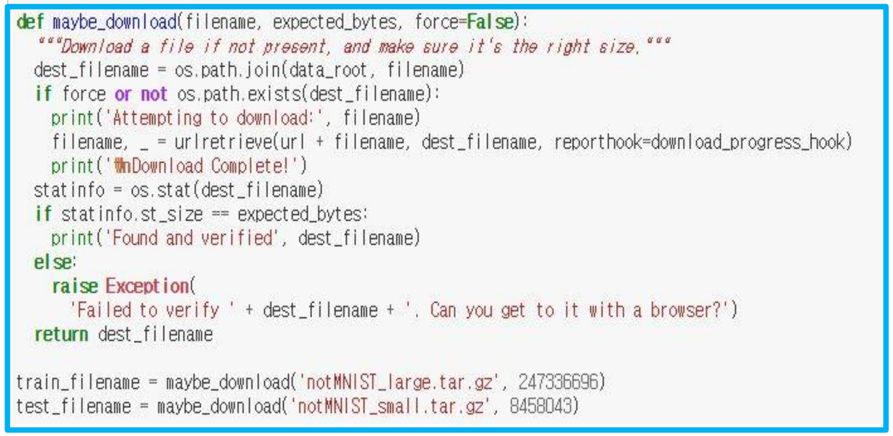
- ✓ A~J의 10개의 Class를 가진다.
- ✓ 캐릭터 랜더링 되어있다.
- ✓ MNIST보다 큰 데이터를 가지며, 노이즈가 많다.

– Getting Started with notMNIST





```
def download_progress_hook(count, blockSize, totalSize):
  """A hook to report the progress of a download. This is mostly intended for users with
  slow internet connections. Reports every 5% change in download progress.
  """
  global last_percent_reported
  percent = int(count * blockSize * 100 / totalSize)

  if last_percent_reported != percent:
    if percent % 5 == 0:
      sys.stdout.write("%s%%" % percent)
      sys.stdout.flush()
    else:
      sys.stdout.write(".")
      sys.stdout.flush()

    last_percent_reported = percent
```

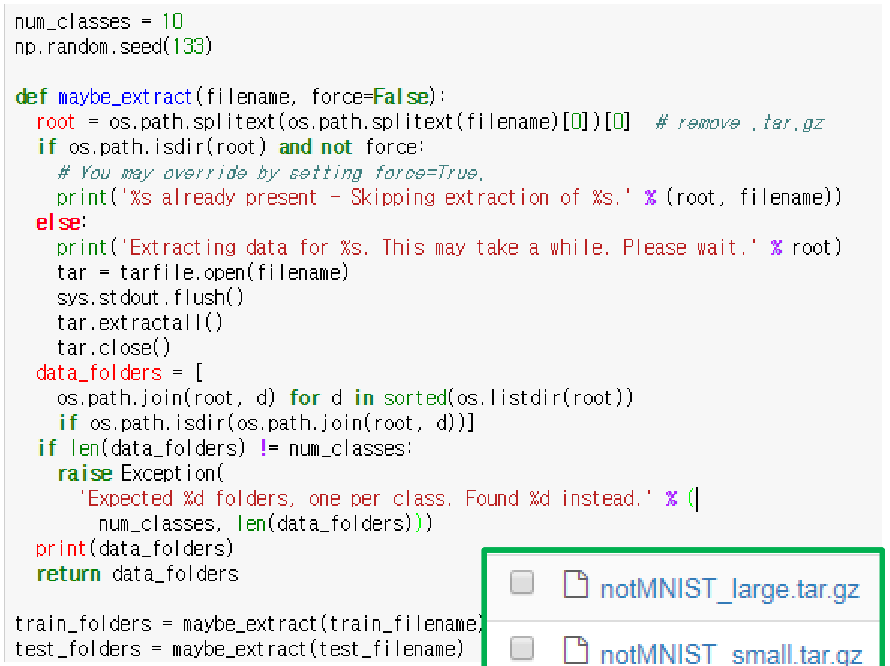


```
def maybe_download(filename, expected_bytes, force=False):
  """Download a file if not present, and make sure it's the right size."""
  dest_filename = os.path.join(data_root, filename)
  if force or not os.path.exists(dest_filename):
    print('Attempting to download:', filename)
    filename, _ = urlretrieve(url + filename, dest_filename, reporthook=download_progress_hook)
    print('₩nDownload Complete!')
  statinfo = os.stat(dest_filename)
  if statinfo.st_size == expected_bytes:
    print('Found and verified', dest_filename)
  else:
    raise Exception(
      'Failed to verify ' + dest_filename + '. Can you get to it with a browser?')
  return dest_filename

train_filename = maybe_download('notMNIST_large.tar.gz', 247336696)
test_filename = maybe_download('notMNIST_small.tar.gz', 8458043)
```

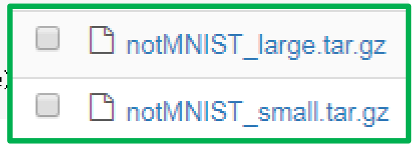
- ✓ 코드를 실행하면 다운로드가 시작합니다.
- ✓ 코드 하단에 나오는 진행률을 확인하여, 다운로드가 끝날 때까지 기다리세요.

– Getting Started with notMNIST

Bio Keras



```python
num_classes = 10
np.random.seed(133)

def maybe_extract(filename, force=False):
  root = os.path.splitext(os.path.splitext(filename)[0])[0]  # remove .tar.gz
  if os.path.isdir(root) and not force:
    # You may override by setting force=True.
    print('%s already present - Skipping extraction of %s.' % (root, filename))
  else:
    print('Extracting data for %s. This may take a while. Please wait.' % root)
    tar = tarfile.open(filename)
    sys.stdout.flush()
    tar.extractall()
    tar.close()
  data_folders = [
    os.path.join(root, d) for d in sorted(os.listdir(root))
    if os.path.isdir(os.path.join(root, d))]
  if len(data_folders) != num_classes:
    raise Exception(
      'Expected %d folders, one per class. Found %d instead.' % (
        num_classes, len(data_folders)))
  print(data_folders)
  return data_folders

train_folders = maybe_extract(train_filename)
test_folders = maybe_extract(test_filename)
```

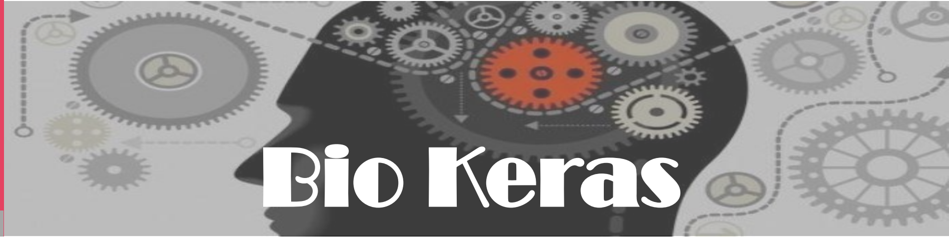


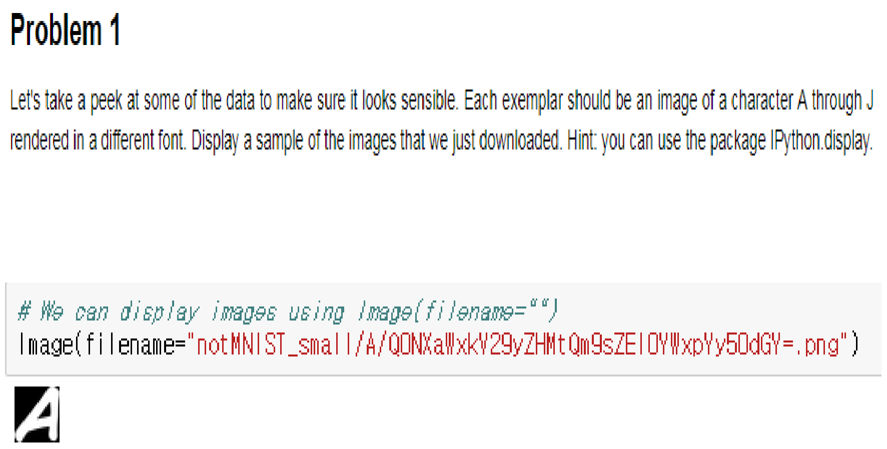
notMNIST_large.tar.gz

notMNIST_small.tar.gz

- ✓ 다운로드가 끝나면 tar.gz라는 확장자를 가진 두 개의 파일을 확인할 수 있어요.
- ✓ 좌측 코드를 실행 시키면 두 개의 폴더에 압축을 풀 수 있습니다.

Problem 1 : Display some images





Problem 1

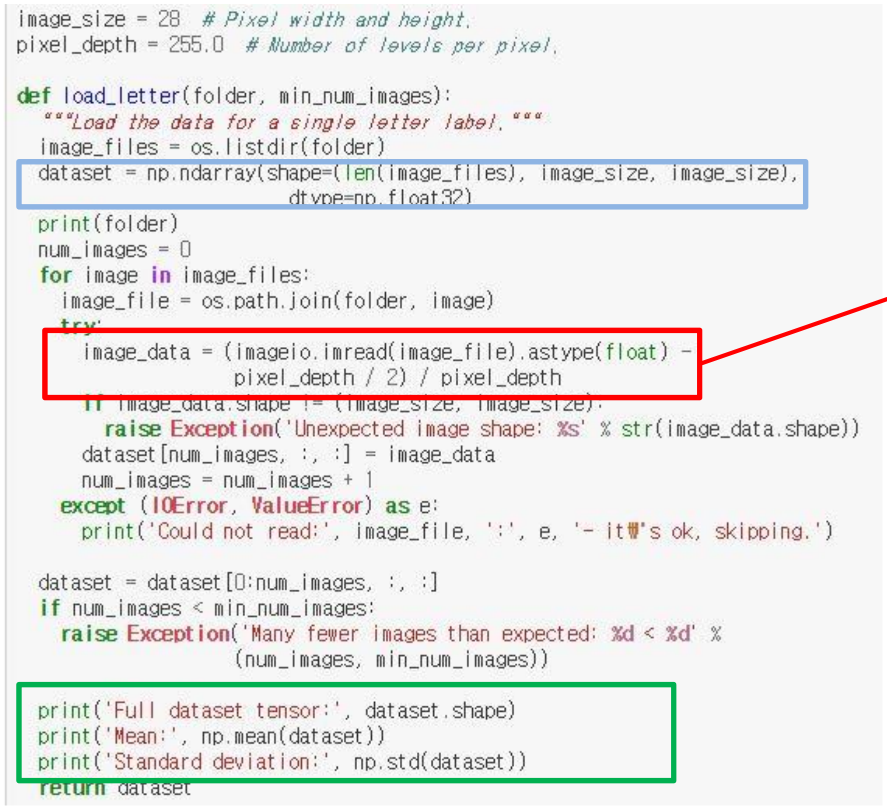
Let's take a peek at some of the data to make sure it looks sensible. Each exemplar should be an image of a character A through J rendered in a different font. Display a sample of the images that we just downloaded. Hint: you can use the package IPython.display.

```
# We can display images using Image(filename="")
Image(filename="notMNIST_small/A/Q0NXaWxkV29yZHMtQm9sZEl0YWxpYy50dGY=.png")
```

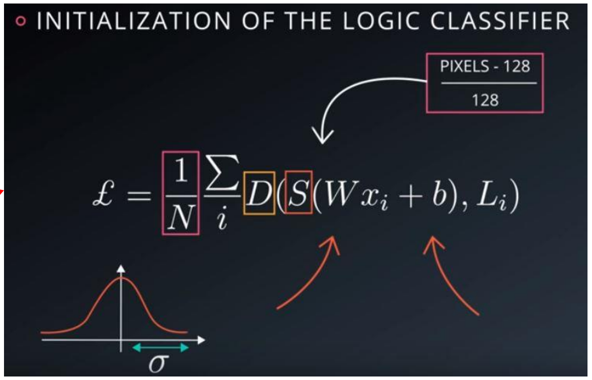
- ✓ Problem 1
  각 서브 폴더마다 하나의 문자가 있고, 올바른 데이터가 있고, 실제로 그것을 참조 할 수 있는지 확인해 보자.

- ✓ Image(filename=“파일 위치”)를 사용하면 다음과 같이 확인 할 수 있어요.

– Converting & nomalizing

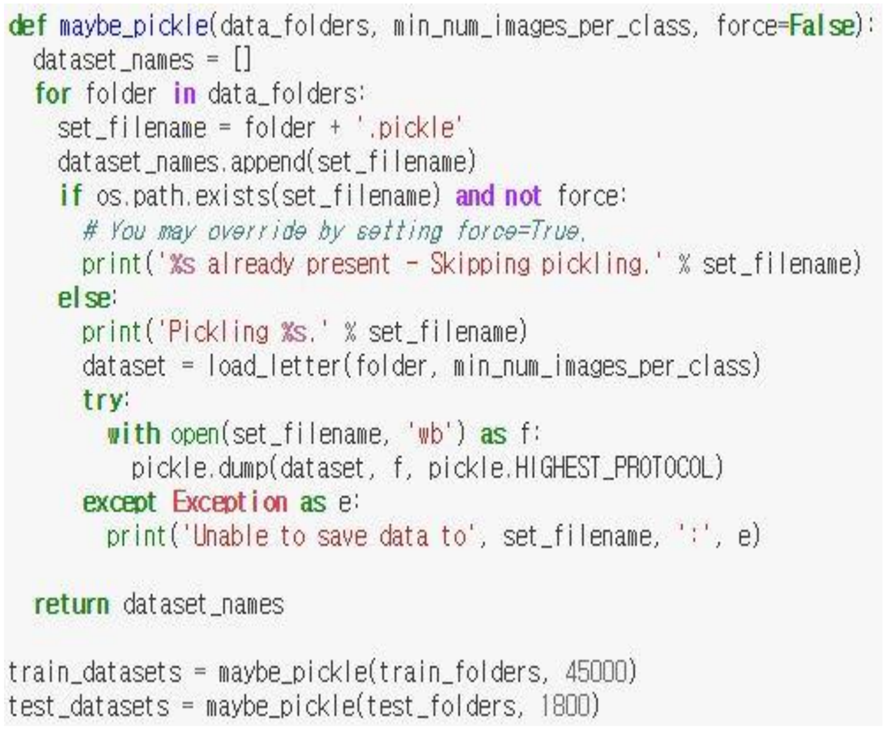


```
image_size = 28  # Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

def load_letter(folder, min_num_images):
  """Load the data for a single letter label."""
  image_files = os.listdir(folder)
  dataset = np.ndarray(shape=(len(image_files), image_size, image_size),
                         dtype=np.float32)
  print(folder)
  num_images = 0
  for image in image_files:
    image_file = os.path.join(folder, image)
    try:
      image_data = (imageio.imread(image_file).astype(float) -
                    pixel_depth / 2) / pixel_depth
      if image_data.shape != (image_size, image_size):
        raise Exception('Unexpected image shape: %s' % str(image_data.shape))
      dataset[num_images, :, :] = image_data
      num_images = num_images + 1
    except (IOError, ValueError) as e:
      print('Could not read:', image_file, ':', e, '- it\'s ok, skipping.')

  dataset = dataset[0:num_images, :, :]
  if num_images < min_num_images:
    raise Exception('Many fewer images than expected: %d < %d' %
                    (num_images, min_num_images))

  print('Full dataset tensor:', dataset.shape)
  print('Mean:', np.mean(dataset))
  print('Standard deviation:', np.std(dataset))
  return dataset
```


INITIALIZATION OF THE LOGIC CLASSIFIER
PIXELS - 128
128
$£ = \frac{1}{N}\sum_i D(S(Wx_i + b), L_i)$
σ

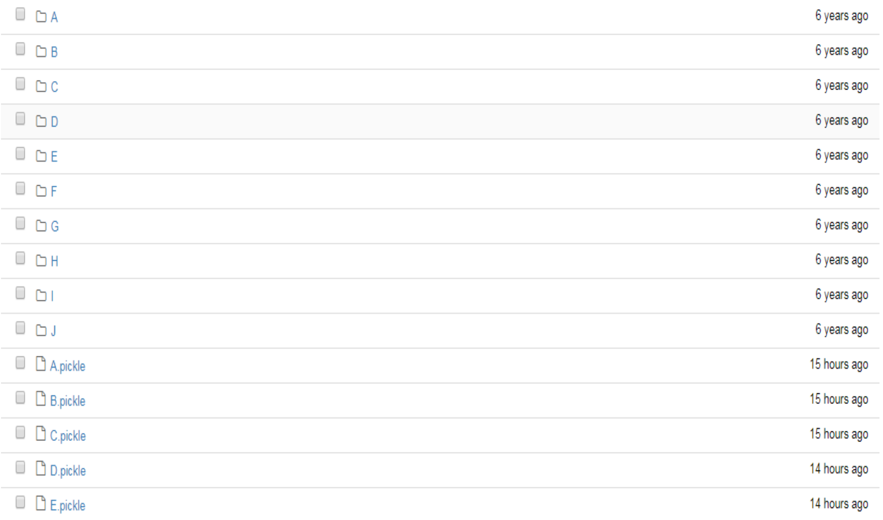
- ✓ 데이터를 디스크에 보관하지 않고, 빠르게 접근할 수 있고, 조작하기 쉬운 형태로 메모리에 로드 합니다.
- ✓ 1차원의 데이터를 3차원으로 변환, 표준화

– Converting & nomalizing



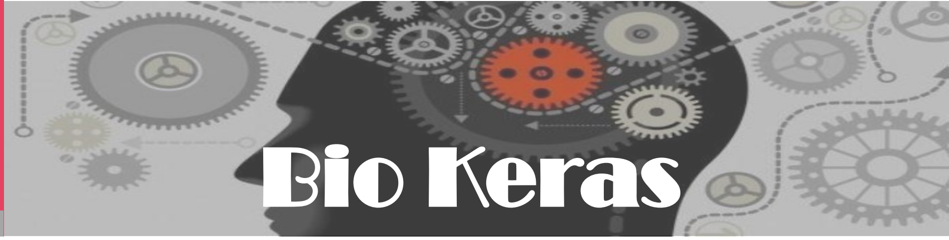
```python
def maybe_pickle(data_folders, min_num_images_per_class, force=False):
  dataset_names = []
  for folder in data_folders:
    set_filename = folder + '.pickle'
    dataset_names.append(set_filename)
    if os.path.exists(set_filename) and not force:
      # You may override by setting force=True.
      print('%s already present - Skipping pickling.' % set_filename)
    else:
      print('Pickling %s.' % set_filename)
      dataset = load_letter(folder, min_num_images_per_class)
      try:
        with open(set_filename, 'wb') as f:
          pickle.dump(dataset, f, pickle.HIGHEST_PROTOCOL)
      except Exception as e:
        print('Unable to save data to', set_filename, ':', e)

  return dataset_names

train_datasets = maybe_pickle(train_folders, 45000)
test_datasets = maybe_pickle(test_folders, 1800)
```

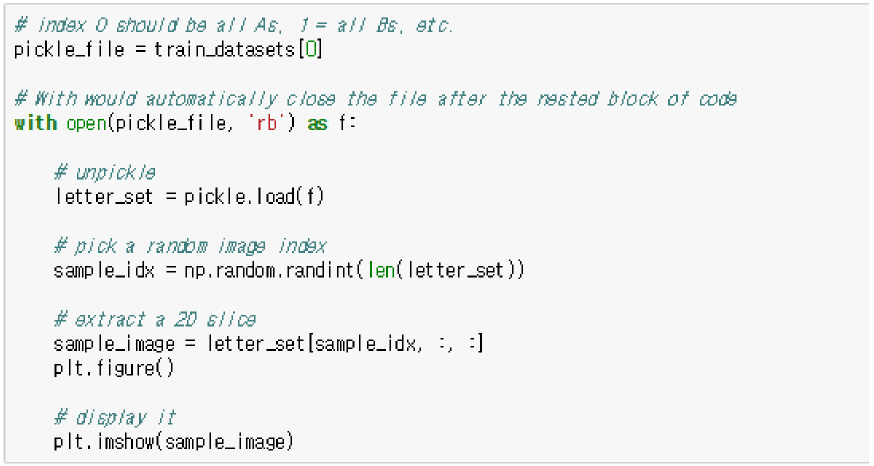
✓ 데이터 처리 중 메모리의 문제와 같이 중단할 상황이 발생할 것을 염려하여 각 데이터를 폴더에 저장합니다.



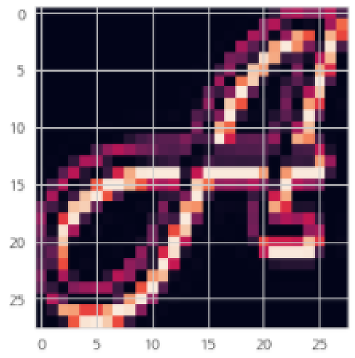
A	6 years ago
B	6 years ago
C	6 years ago
D	6 years ago
E	6 years ago
F	6 years ago
G	6 years ago
H	6 years ago
I	6 years ago
J	6 years ago
A.pickle	15 hours ago
B.pickle	15 hours ago
C.pickle	15 hours ago
D.pickle	14 hours ago
E.pickle	14 hours ago

Problem 2 : Verify normalized





```
# index 0 should be all As, 1 = all Bs, etc.
pickle_file = train_datasets[0]

# With would automatically close the file after the nested block of code
with open(pickle_file, 'rb') as f:

    # unpickle
    letter_set = pickle.load(f)

    # pick a random image index
    sample_idx = np.random.randint(len(letter_set))

    # extract a 2D slice
    sample_image = letter_set[sample_idx, :, :]
    plt.figure()

    # display it
    plt.imshow(sample_image)
```

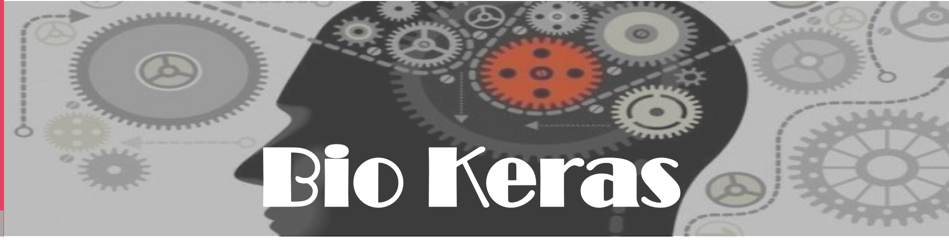


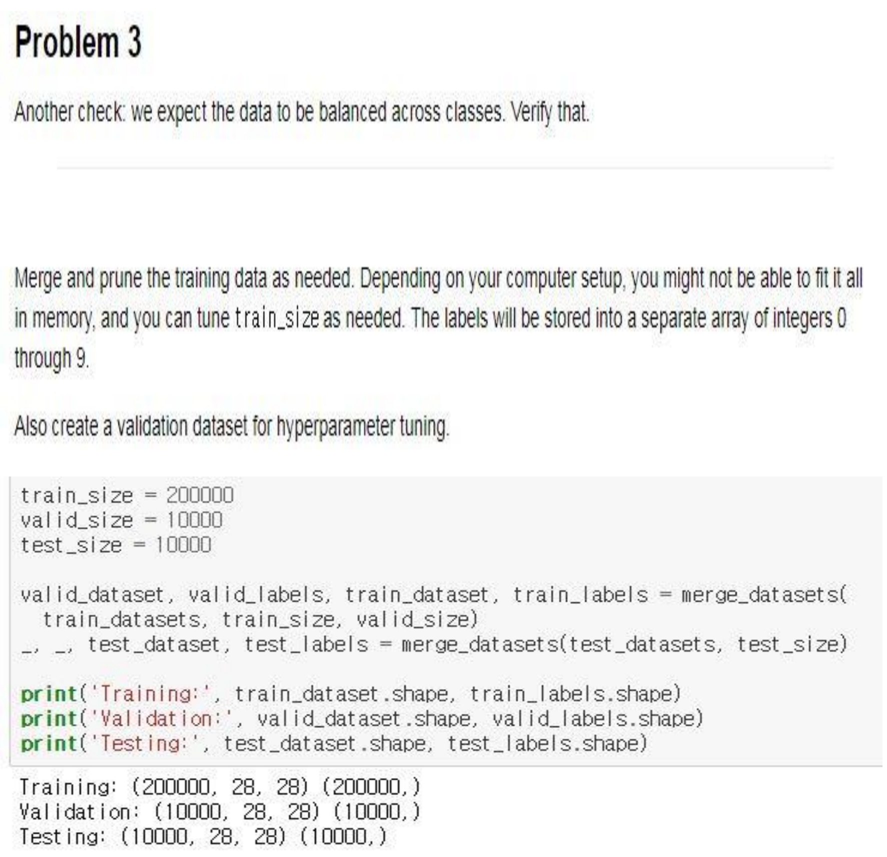
Problem 2

Let's verify that the data still looks good. Displaying a sample of the labels and images from the ndarray. Hint: you can use matplotlib.pyplot.

- ✓ 우리는 일련의 과정을 거친 데이터가 여전히 사용 가능한지 검증해야 합니다. 시각화를 이용해 확인합니다.

Problem 3 : Verify data is balanced





Problem 3

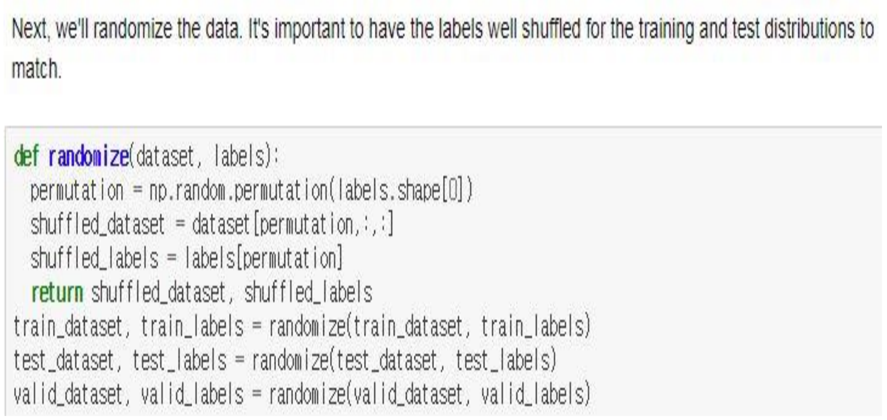
Another check: we expect the data to be balanced across classes. Verify that.

Merge and prune the training data as needed. Depending on your computer setup, you might not be able to fit it all in memory, and you can tune train_size as needed. The labels will be stored into a separate array of integers 0 through 9.

Also create a validation dataset for hyperparameter tuning.

```
train_size = 200000
valid_size = 10000
test_size = 10000

valid_dataset, valid_labels, train_dataset, train_labels = merge_datasets(
  train_datasets, train_size, valid_size)
_, _, test_dataset, test_labels = merge_datasets(test_datasets, test_size)

print('Training:', train_dataset.shape, train_labels.shape)
print('Validation:', valid_dataset.shape, valid_labels.shape)
print('Testing:', test_dataset.shape, test_labels.shape)
```

```
Training: (200000, 28, 28) (200000,)
Validation: (10000, 28, 28) (10000,)
Testing: (10000, 28, 28) (10000,)
```



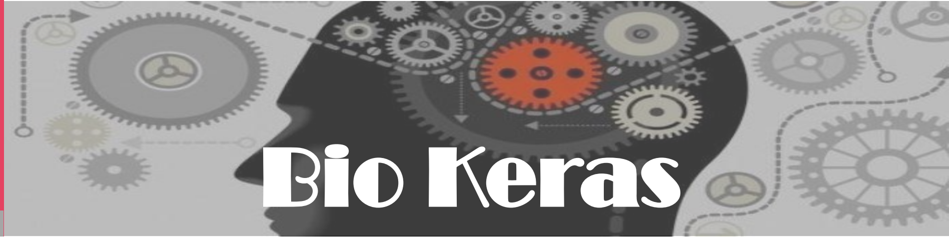
Next, we'll randomize the data. It's important to have the labels well shuffled for the training and test distributions to match.

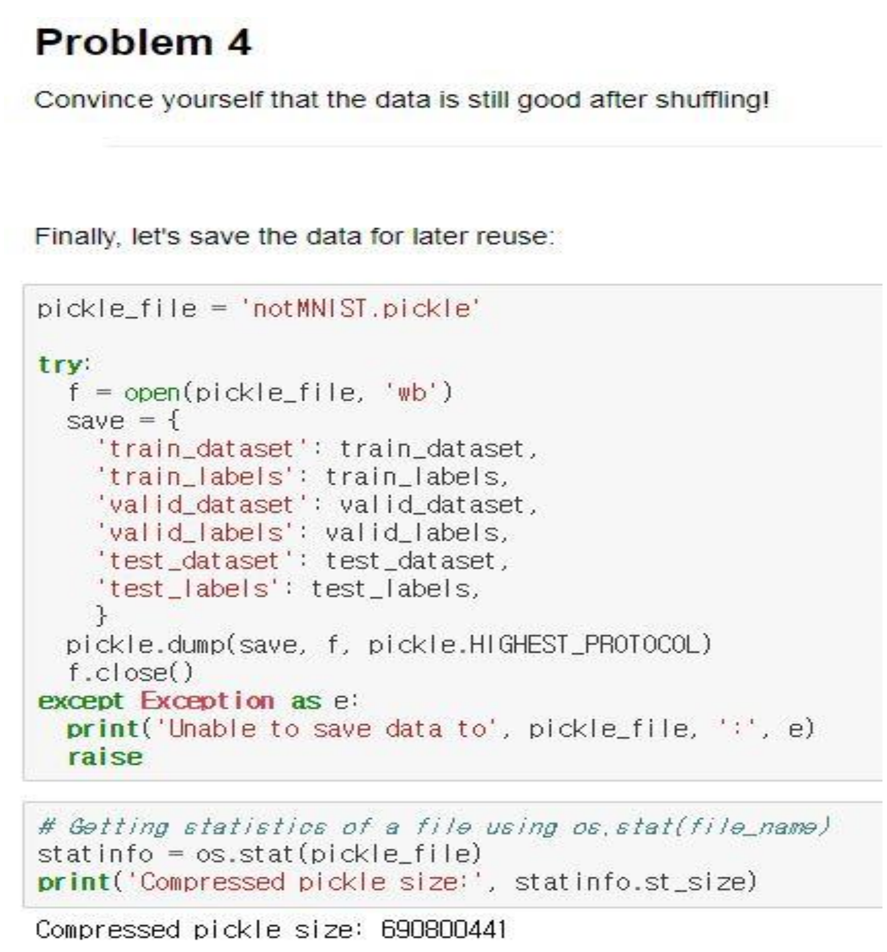
```
def randomize(dataset, labels):
  permutation = np.random.permutation(labels.shape[0])
  shuffled_dataset = dataset[permutation,:,:]
  shuffled_labels = labels[permutation]
  return shuffled_dataset, shuffled_labels
train_dataset, train_labels = randomize(train_dataset, train_labels)
test_dataset, test_labels = randomize(test_dataset, test_labels)
valid_dataset, valid_labels = randomize(valid_dataset, valid_labels)
```

- ✓ Training, Validation, Testing 데이터셋을 만들고, 모델이 학습하는 순서가 특정 Lable에 편향되지 않게 shuffle 합니다.

Problem 4 :

Shuffle samples and





Problem 4

Convince yourself that the data is still good after shuffling!

Finally, let's save the data for later reuse:

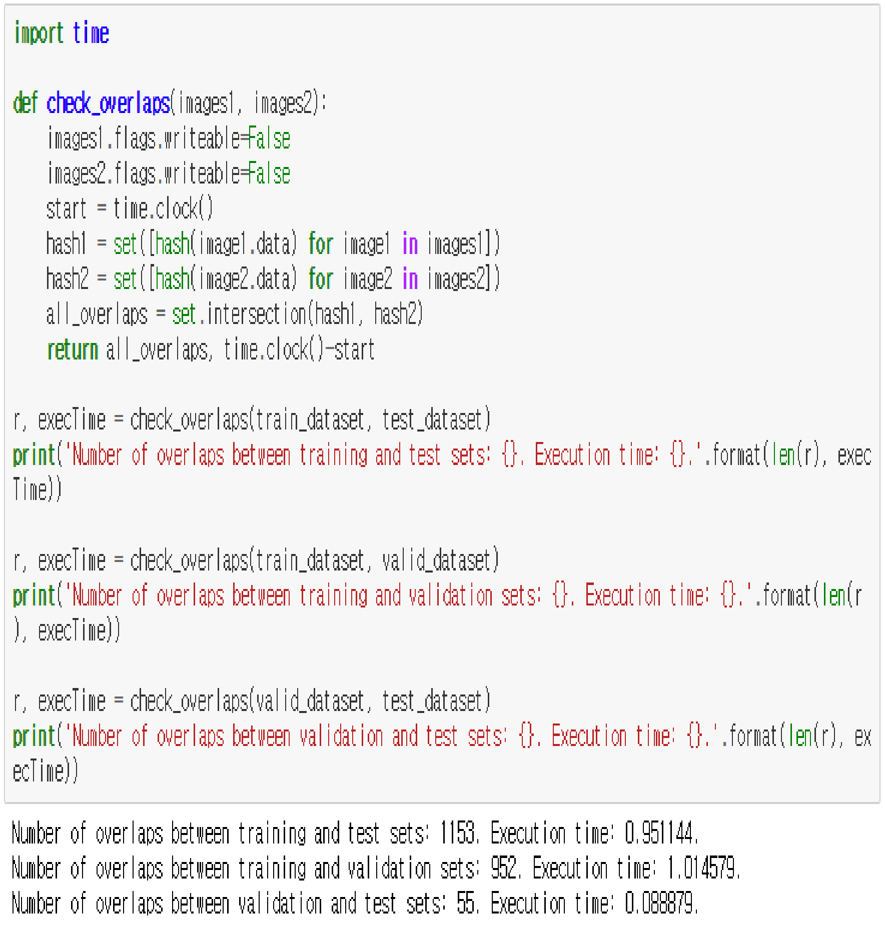
```
pickle_file = 'notMNIST.pickle'

try:
  f = open(pickle_file, 'wb')
  save = {
    'train_dataset': train_dataset,
    'train_labels': train_labels,
    'valid_dataset': valid_dataset,
    'valid_labels': valid_labels,
    'test_dataset': test_dataset,
    'test_labels': test_labels,
    }
  pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
  f.close()
except Exception as e:
  print('Unable to save data to', pickle_file, ':', e)
  raise
```

```
# Getting statistics of a file using os.stat(file_name)
statinfo = os.stat(pickle_file)
print('Compressed pickle size:', statinfo.st_size)
```

```
Compressed pickle size: 690800441
```

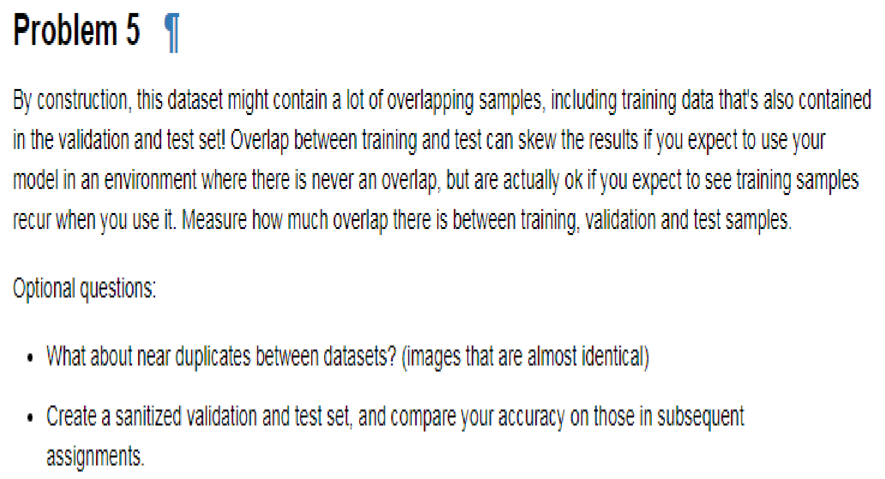
- ✓ 데이터를 shuffling한 후 관리하기 쉽게 하나의 파일로 저장합니다.
- ✓ 저장한 파일의 크기를 확인합니다.

Problem 5 : Find overlapping samples



```
import time

def check_overlaps(images1, images2):
    images1.flags.writeable=False
    images2.flags.writeable=False
    start = time.clock()
    hash1 = set([hash(image1.data) for image1 in images1])
    hash2 = set([hash(image2.data) for image2 in images2])
    all_overlaps = set.intersection(hash1, hash2)
    return all_overlaps, time.clock()-start

r, execTime = check_overlaps(train_dataset, test_dataset)
print('Number of overlaps between training and test sets: {}. Execution time: {}.'.format(len(r), execTime))

r, execTime = check_overlaps(train_dataset, valid_dataset)
print('Number of overlaps between training and validation sets: {}. Execution time: {}.'.format(len(r), execTime))

r, execTime = check_overlaps(valid_dataset, test_dataset)
print('Number of overlaps between validation and test sets: {}. Execution time: {}.'.format(len(r), execTime))
```

```
Number of overlaps between training and test sets: 1153. Execution time: 0.951144.
Number of overlaps between training and validation sets: 952. Execution time: 1.014579.
Number of overlaps between validation and test sets: 55. Execution time: 0.088879.
```



Problem 5 ¶

By construction, this dataset might contain a lot of overlapping samples, including training data that's also contained in the validation and test set! Overlap between training and test can skew the results if you expect to use your model in an environment where there is never an overlap, but are actually ok if you expect to see training samples recur when you use it. Measure how much overlap there is between training, validation and test samples.

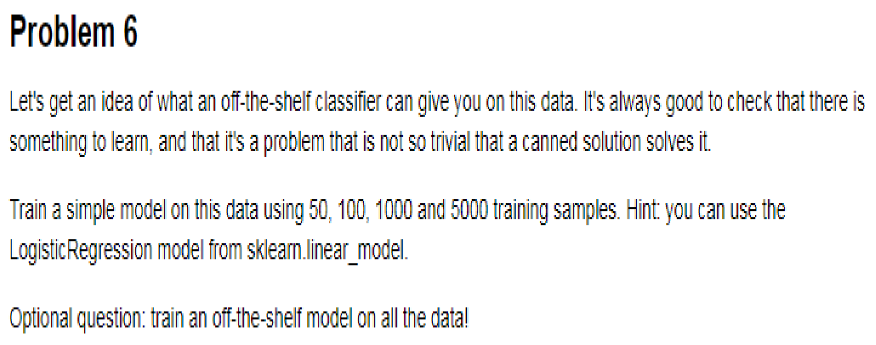
Optional questions:

- What about near duplicates between datasets? (images that are almost identical)
- Create a sanitized validation and test set, and compare your accuracy on those in subsequent assignments.

✓ 훈련, 검증, 테스트 데이터 셋의 중복을 확인합니다.

✓ 훈련 데이터와 테스트 데이터의 중복은 과학습(overfitting)할 초래할 가능성이 있습니다.

Problem 6 : Train a simple ML



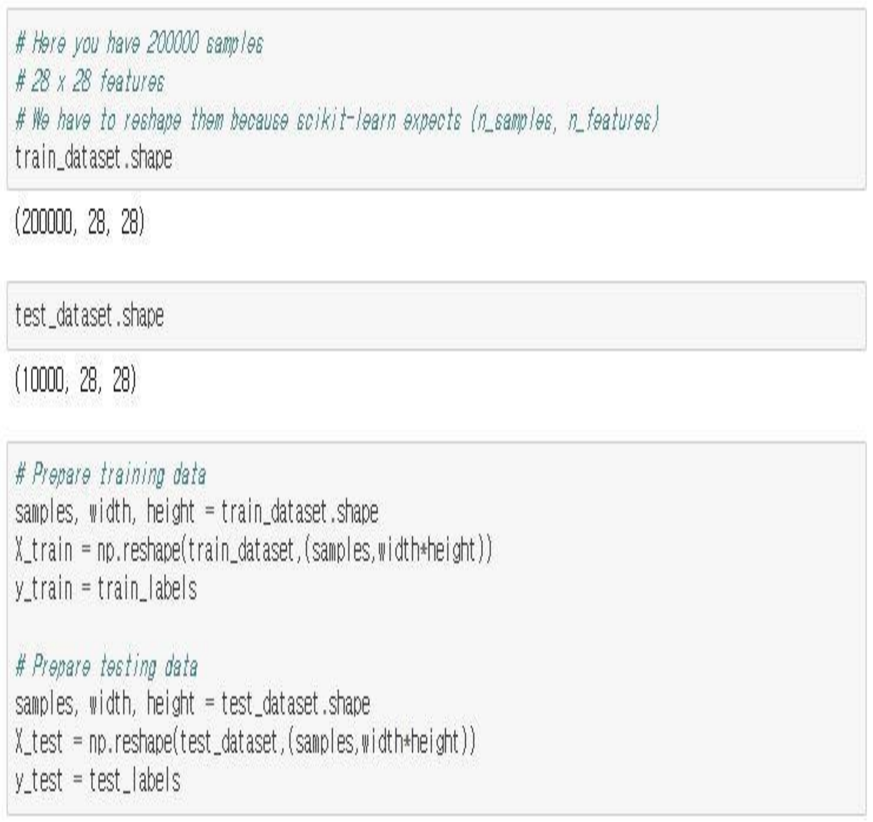
Problem 6

Let's get an idea of what an off-the-shelf classifier can give you on this data. It's always good to check that there is something to learn, and that it's a problem that is not so trivial that a canned solution solves it.

Train a simple model on this data using 50, 100, 1000 and 5000 training samples. Hint: you can use the LogisticRegression model from sklearn.linear_model.

Optional question: train an off-the-shelf model on all the data!

- ✓ 간단한 모델을 이용해 feature의 성능을 확인합니다.
- ✓ 우선 학습 데이터를 준비합니다.



```
# Here you have 200000 samples
# 28 x 28 features
# We have to reshape them because scikit-learn expects (n_samples, n_features)
train_dataset.shape
```

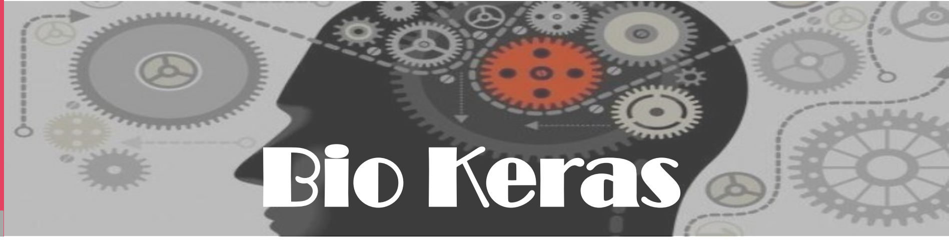
```
(200000, 28, 28)
```

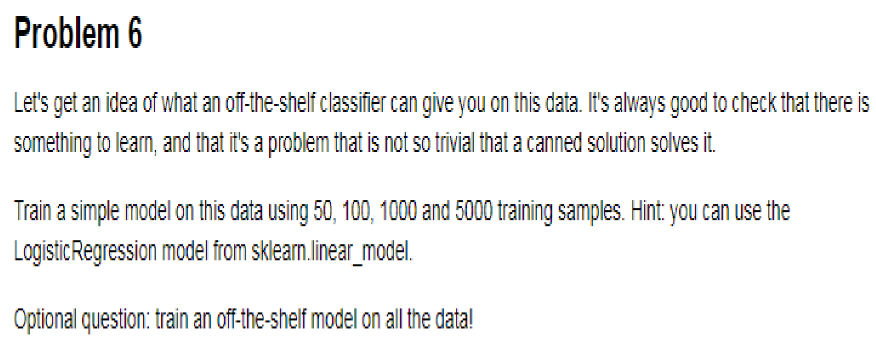
```
test_dataset.shape
```

```
(10000, 28, 28)
```

```
# Prepare training data
samples, width, height = train_dataset.shape
X_train = np.reshape(train_dataset,(samples,width*height))
y_train = train_labels

# Prepare testing data
samples, width, height = test_dataset.shape
X_test = np.reshape(test_dataset,(samples,width*height))
y_test = test_labels
```

Problem 6 : Train a simple ML





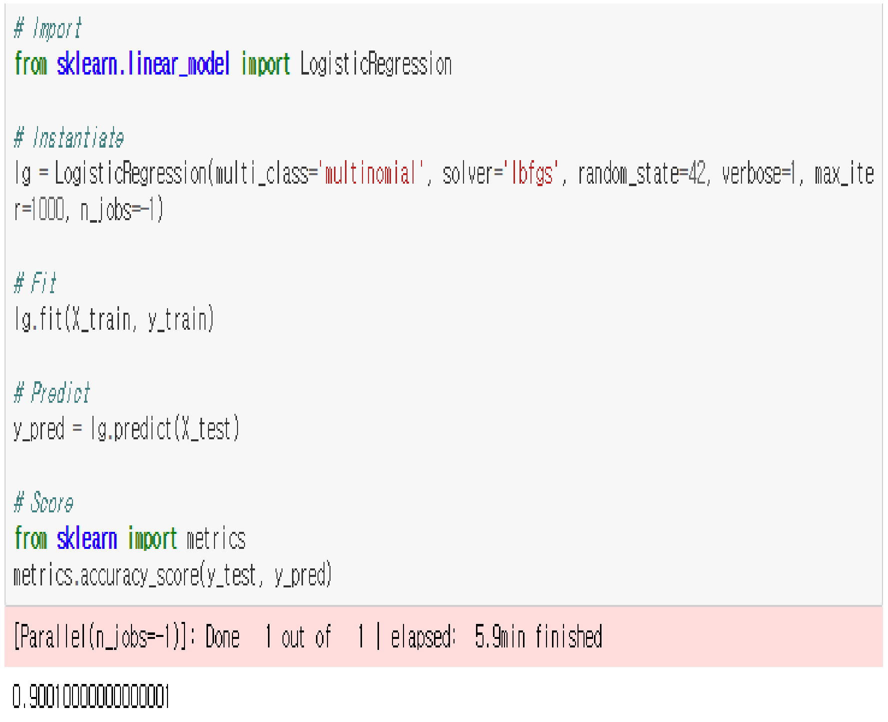
Problem 6

Let's get an idea of what an off-the-shelf classifier can give you on this data. It's always good to check that there is something to learn, and that it's a problem that is not so trivial that a canned solution solves it.

Train a simple model on this data using 50, 100, 1000 and 5000 training samples. Hint: you can use the LogisticRegression model from sklearn.linear_model.

Optional question: train an off-the-shelf model on all the data!

✓ Sklearn에 내장된 로지스틱 회귀 함수를 이용하여 학습 후 score함수를 이용해 평균 accuracy를 확인합니다.



```
# Import
from sklearn.linear_model import LogisticRegression

# Instantiate
lg = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=42, verbose=1, max_ite
r=1000, n_jobs=-1)

# Fit
lg.fit(X_train, y_train)

# Predict
y_pred = lg.predict(X_test)

# Score
from sklearn import metrics
metrics.accuracy_score(y_test, y_pred)
```

```
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:  5.9min finished
```

```
0.9001000000000001
```