Report for OM Project

Pegasos SVM

Haard Panchal
201501153

Youtube link of screencast: https://youtu.be/nC5oE6XJF-M

Importing and hyperparameters:

```
import sys
import os
import gzip
import numpy as np

selected_classes = [3,8] # Select any two classes from the FashionMNIST
dataset
lb = 1 # Lambda value
T = 1000 # Number of iterations

#Train and Tested on FashionMNIST
```

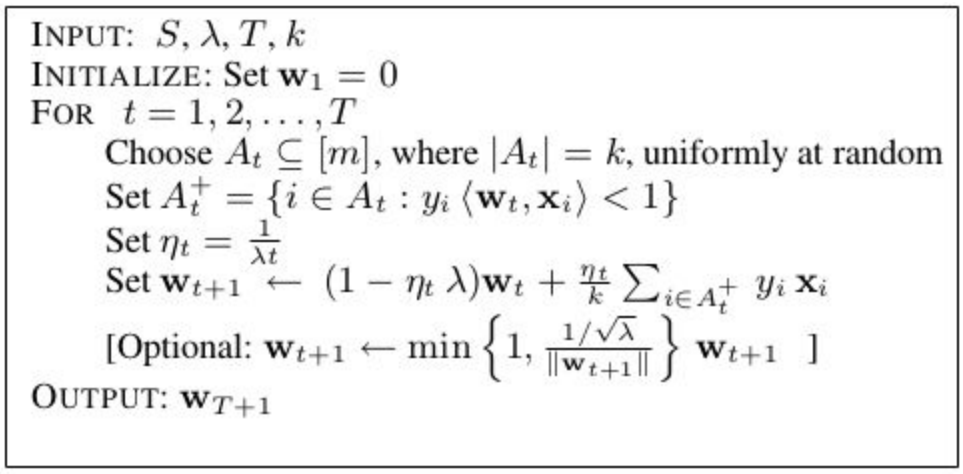
Loading and formatting the dataset:

```
def load_mnist(path='.', kind='train'):
      """Load MNIST data from `path`"""
      labels_path = os.path.join(path,
                                 '%s-labels-idx1-ubyte.gz'
                                 % kind)
      images_path = os.path.join(path,
                                 '%s-images-idx3-ubyte.gz'
                                 % kind)
      with gzip.open(labels_path, 'rb') as lbpath:
      labels = np.frombuffer(lbpath.read(), dtype=np.uint8,
                                 offset=8)
      with gzip.open(images_path, 'rb') as imgpath:
      images = np.frombuffer(imgpath.read(), dtype=np.uint8,
                                 offset=16).reshape(len(labels), 784)
      return images, labels
```

```
(train_images, train_labels) = load_mnist(".","train")
(test_images, test_labels) = load_mnist(".","t10k")

print(train_labels)

i = 0
X_train = []
y_train = []
for label in train_labels:
      if label == selected_classes[0]:
      X_train.append(np.array(train_images[i]))
      y_train.append(1)
      if label == selected_classes[1]:
      X_train.append(train_images[i])
      y_train.append(-1)
      i += 1

i = 0
X_test = []
y_test = []
for label in test_labels:
      if label == selected_classes[0]:
      X_test.append(np.array(test_images[i]))
      y_test.append(1)
      if label == selected_classes[1]:
      X_test.append(test_images[i])
      y_test.append(-1)
      i += 1


print("Train: ", len(X_train))
print("Test: ", len(X_test))

print(X_train[0].shape)
print("Train labels ",len(y_train))
```

Training Non Kernel Version:

Algorithm used:




```
def train_nonkernel(X_train, y_train, T, lb):
      t = 1
      w = np.zeros(X_test[0].shape)
      # for (x,yi) in zip(X_train, y_train):
      idx = np.random.permutation(len(X_train)) #Randomize the order
      print(idx)
      for i in idx:
      x, yi = X_train[i], y_train[i]
      nt  = 1.0/(lb*t)
      if yi * np.dot(w, x) < 1:
            w = (1 - nt * lb) * w + nt * yi * x
            # print(w)
      elif yi * np.dot(w, x) >= 1:
            w = (1 - nt * lb) * w
            # print(w)
      w = min(1, (1/(lb)**(1/2))/np.linalg.norm(w)) * w
      print(np.linalg.norm(w))
      t += 1
      return w
```

Testing the non-kernel version:

```
def test_nonkernel(w, X_test, y_test):
```

```
    total = 0
    correct = 0
    for (x,yi) in zip(X_test, y_test):
    pred = np.dot(w, x)
    if yi * pred > 0:
          correct += 1
    total += 1
    return correct, total
```

Training kernel version:

Algorithm used:

INPUT: S, λ, T

INITIALIZE: Set $\boldsymbol{\alpha}_1 = 0$

FOR $t = 1, 2, \ldots, T$

Choose $i_t \in \{0, \ldots, |S|\}$ uniformly at random.

For all $j \neq i_t$, set $\alpha_{t+1}[j] = \alpha_t[j]$

If $y_{i_t} \frac{1}{\lambda t} \sum_j \alpha_t[j] y_{i_t} K(\mathbf{x}_{i_t}, \mathbf{x}_j) < 1$, then:

Set $\alpha_{t+1}[i_t] = \alpha_t[i_t] + 1$

Else:

Set $\alpha_{t+1}[i_t] = \alpha_t[i_t]$

OUTPUT: $\boldsymbol{\alpha}_{T+1}$

```
def train_kernel(X_train, y_train, T):
```

```
    al = np.zeros(len(X_train))
    idx = np.random.permutation(len(X_train))
    print(idx)
    t = 0
    for i in idx:
    x, yi = X_train[i], y_train[i]
    s = 0
    for j in range(len(X_train)):
        s += al[j]*y_train[j]*K(x,X_train[j])
    if yi*(1/lb)*s < 1:
        al[i] = al[i] + 1
    if t >= T:
        break
    else:
        t += 1
    print("Iteration of Kernel Training: ", t)
    return al
```

Testing kernel version:

```
def test_kernel(al, X_test, y_test, X_train, y_train, T):
    total = 0
    correct = 0
    t = 0
    for (x,yi) in zip(X_test, y_test):
    s = 0
    for j in range(len(X_train)):
        s += al[j]*y_train[j]*K(x,X_train[j])
    if yi*(1/lb)*s < 1:
        correct += 1
    total += 1
    if t >= T:
        break
    else:
        t += 1
    print("Testing iteration: ", t)
    return correct, total
```

The kernel used was the RBF kernel or the Gaussian kernel:

```
def K(x1, x2):
```

```
        # return np.dot(phi(x1), phi(x2))
        return np.exp(-1*np.linalg.norm(x1-x2)**2) #The RBF Kernel
```

Main function:

```
w = train_nonkernel(X_train, y_train, 10000, lb)
print("Without kernel: ",test_nonkernel(w, X_test, y_test)) # Not using no.
of iteration as it is fast
al = train_kernel(X_train, y_train, T)
print("With Kernel: ", test_kernel(al, X_test, y_test, X_train, y_train,
50))
```

Statistics:

Kernel version:

Accuracy: 1951/2000 or 97.55 %

Non-kernel version

Accuracy: 51/51 Could not run it for the whole test data as it was taking way too much time.