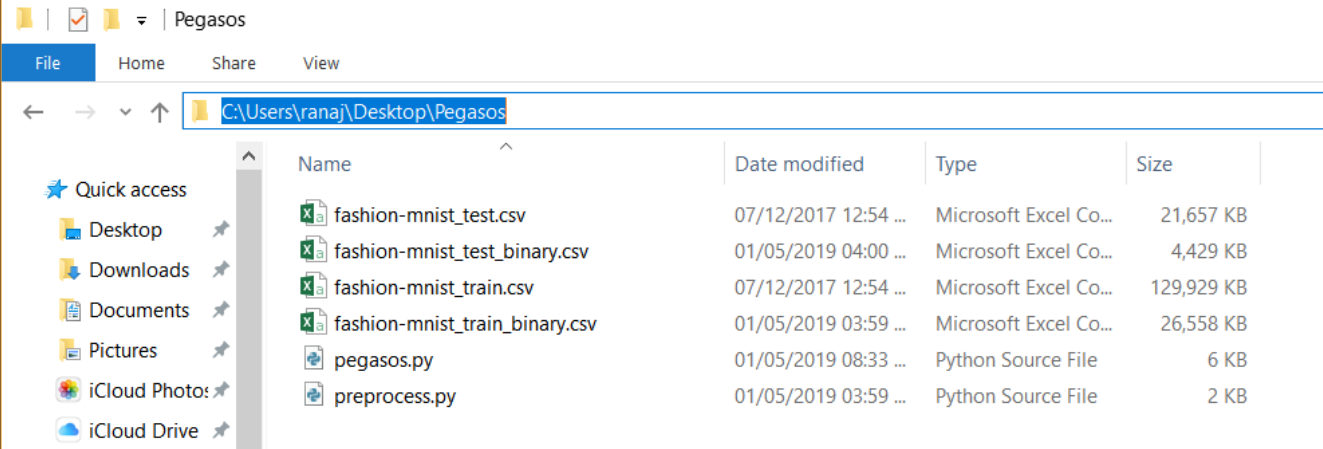


```
C:\> Command Prompt
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ranaaj>cd C:\Users\ranaaj\Desktop\Pegasos

C:\Users\ranaaj\Desktop\Pegasos>python pegasos.py
Loaded Training Samples
Loaded Testing Samples
Computing Gram Matrix
Computed Gram Matrix
Computing 5 Iterations of Pegasos
Completed Pegasos
The training time of Pegasos with 5 Iterations is 0.139138
The time required to predict 1999 of samples is 3.384710
The accuracy of implemented Pegasos is 0.874937
```



Name	Date modified	Type	Size
fashion-mnist_test.csv	07/12/2017 12:54 ...	Microsoft Excel Co...	21,657 KB
fashion-mnist_test_binary.csv	01/05/2019 04:00 ...	Microsoft Excel Co...	4,429 KB
fashion-mnist_train.csv	07/12/2017 12:54 ...	Microsoft Excel Co...	129,929 KB
fashion-mnist_train_binary.csv	01/05/2019 03:59 ...	Microsoft Excel Co...	26,558 KB
pegasos.py	01/05/2019 08:33 ...	Python Source File	6 KB
preprocess.py	01/05/2019 03:59 ...	Python Source File	2 KB

# Primal Estimated sub-GrAdient SOlver for SVM (Pegasos)

SCREENCAST LINK:

[HTTPS://DRIVE.GOOGLE.COM/FILE/D/1ZAQYAKPVCPPHTD9ORBEGIK9RL77BGF4M/VIEW?USP=SHARING](https://drive.google.com/file/d/1ZAQYAKPVCPPHTD9ORBEGIK9RL77BGF4M/view?usp=sharing)

Ranajit Saha | 20172119 | 1<sup>st</sup> May, 2019

## Dataset

FashionMNIST dataset has been used for both binary and multiclass classification.

- For binary classification task, all 6000 examples of class 0 and random 6000 examples of all other classes have been chosen for training and testing samples contains 1000 of each class chosen in the same manner as training. For kernelized version, we used a total of 2000 samples in training taken randomly from the above training samples.
- For multiclass classification task, the all the nine classes are used. The training phase contains 1000 examples from each class.
- Each image is of size  $32 \times 32$  which is flattened to a vector of size 784.

## Implementation details

The implemented hard margin Pegasos has the following options:

- Non-Kernelized and Kernelized SVM implementation
- Kernelized SVM has the option of using the following Kernels.
  - Linear Kernel
  - Homogeneous and in-homogeneous polynomial kernel of any degree passed as a parameter
  - Hyperbolic Tangent Kernel
- The Hyper-parameters of the SVM are no. of iterations, lambda as the regularization parameter.
- Binary classification and multiclass classification

After training the SVM classifier, the support vectors and the weight matrices are generated. Using them we classify the new test samples. The testing accuracies are listed in *Results* section. The multiclass classification is done using one-vs-all classifier.

## Core parts of the SVM

- **Pegasos optimization algorithm:** The following code snippet is responsible for learning the coefficient vector  $\mathbf{w}$  optimizing the primal objective in stochastic gradient descent manner. If the Gram matrix is empty then non-kernelized version

of the algorithm is used. In the end the support vectors **SV** are identified by the non-zero coefficients. The commented step is the optional step where we use gradient-projection approach where we limit the set of admissible solutions to the ball of radius  $1/\sqrt{\lambda}$ . The comparison of test accuracies between using this step and omitting this projection is reported in *Results* section.

```
def Pegasos(X_train, Y_train, GramMatrix, lambdaa = 0.1, Iterations = 5):
    samples = X_train.shape[0]
    if GramMatrix.shape[0] > 0:
        w = np.zeros(samples)
    else:
        w = np.random.rand(X_train.shape[1])
    time = 1
    for i in range(Iterations):
        for tau in range(len(X_train)):
            if GramMatrix.shape[0] == 0:
                etat = 1/(time*lambdaa)
                wx = np.dot(w, X_train[tau])
                w *= (1-1/time)
                if Y_train[tau] * wx < 1:
                    w += etat * Y_train[tau] * X_train[tau]
                    # w *= min(1, 1/(math.sqrt(lambdaa)*LA.norm(w)))
            else:
                wx = np.dot(w, GramMatrix[tau])
                w *= (1-1/time)
                if (Y_train[tau]*wx < 1):
                    w[tau] += Y_train[tau] / (1/lambdaa)
            time += 1
    non_zero_indices = np.where(w != 0)
    SV = X_train[non_zero_indices, :]
```

- **Computing the Gram matrix:** The Gram matrix of size  $N \times N$  has been calculated in the following part,  $N$  being the number of training examples. To

```
def computeGRAM(X_train, Kernel):
    gram = np.zeros((X_train.shape[0], X_train.shape[0]))
    for i in range(X_train.shape[0]):
        for j in range(i, X_train.shape[0]):
            gram[i,j] = Kernel(X_train[i], X_train[j])
    for i in range(X_train.shape[0]):
        for j in range(i):
            gram[i,j] = gram[j,i]
    return gram
```

reduce the complexity of calculating the matrix we use the symmetricity property of the Gram matrix.

- **Sub-Gradient:** For hinge loss the sub-gradient is

$$\mathbf{v}_t = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i z < 1 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

The code snippet which uses this sub-gradient to update the coefficients is

```
w += etat * Y_train[tau] * X_train[tau]
```

- **Loss function:** The optimization is derived based on *hinge loss*. The following snippet is based on hinge loss where the weights are updated only if  $\mathbf{y}_\tau \mathbf{w}^T \mathbf{x}_\tau$  in case of non-kernelized usage or  $\mathbf{y}_\tau \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_\tau)$  in case of kernelized usage is less than 1.

- Non-Kernelized

```
if Y_train[tau] * wx < 1:
    w += etat * Y_train[tau] * X_train[tau]
```

- Kernelized

```
if(Y_train[tau]*wx < 1):
    w[tau] += Y_train[tau] / (1/lambdada)
```

## Results

- **Time Complexity:** The training time of *Pegasos* for binary classification task with a total of 12000 training samples with 5 iterations is 0.522602. The time required to predict 1999 of samples is 0.004988 and to predict 1000 of samples is 0.003961.
- **Accuracies of binary classification task:** We report the accuracies of binary classification task on FashionMNIST dataset with both the implemented Pegasos and sklearn SVM for various settings. Each accuracies are reported after 5 iterations of the training algorithm.

Settings	Pegasos Accuracy	sklearn SVM Accuracy
Non-Kernelized	0.892946	0.894972
Non-Kernelized with Projection	0.500250	
Linear Kernel	0.874937	0.879940
Homogeneous Kernel with degree 3	0.850425	
Non-homogeneous Kernel with degree 3	0.850425	
Hyperbolic Tangent Kernel	0.500250	

## Reference

1. <https://github.com/avaitle/Pegasos>