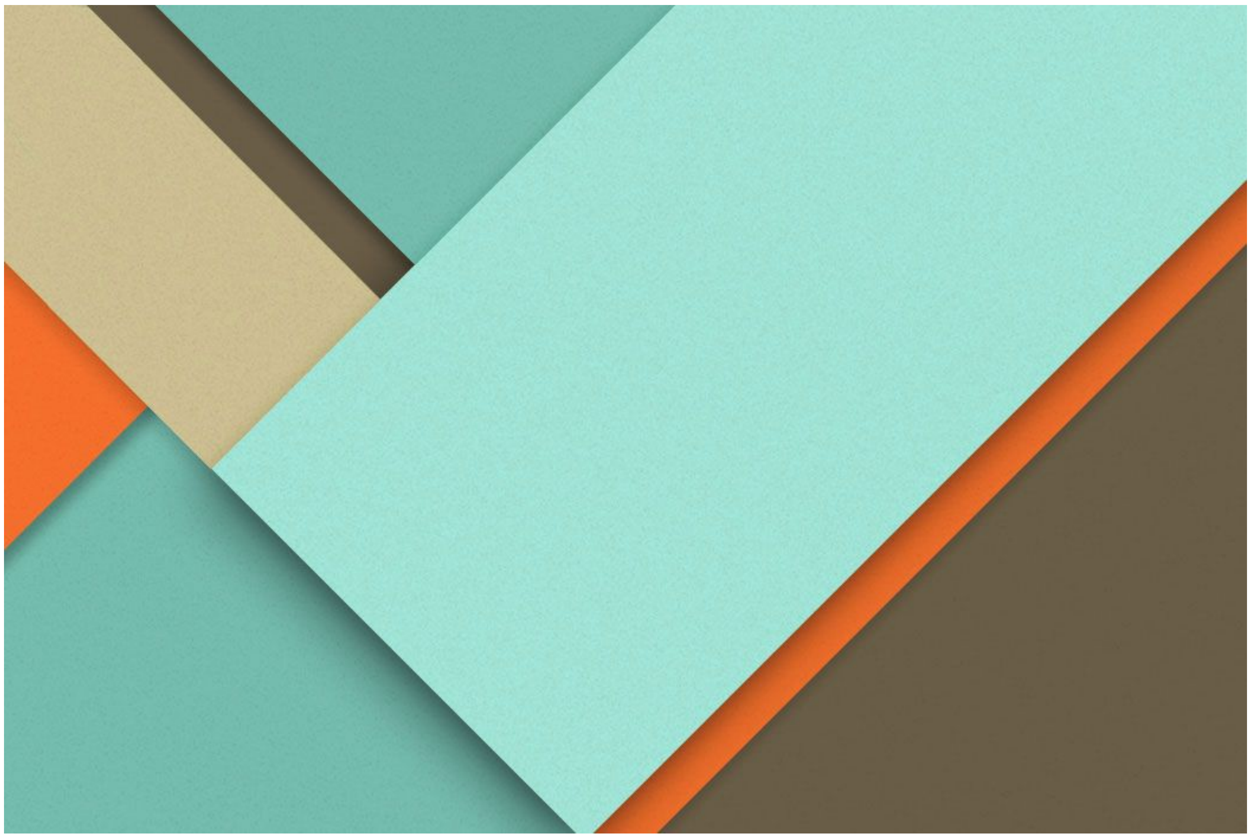


Pegasos

Pradeep.p (20161145)

Non kernelized pegasos

Support Vector Machines are one of the most effective and popular classification learning tool .A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In 2-D space this hyperplane is a line dividing a plane in two parts where in each class lay in either side .It is in fact an unconstrained empirical loss minimization with a penalty term for the norm of the classifier that is being learned . The svm problem is stated as follows , Given a training set (xi, yi) where i = 1 to m . xi ∈ R and yi ∈ {-1, 1} we have to find the minimum value of ,

$$\text{Min}_w\ \lambda/2\ ||w||^2 + 1/m \sum_{(x,y)\in S} L(w; (x,y))$$

Where L is loss function and is defined as follows ,

$$L = \max\{0, 1 - y \langle w,x\rangle \}$$

Usually the above problem is solved by converting it into its dual form and then by solving the dual problem we can get the optimal solution . But in this project we are using pegasos method to solve the above problem in which we don’t convert it into its dual form rather we optimize the given primal objective function using algorithms called pegasos . Pegasos is quite similar to gradient descent except the fact that we use sub gradients instead of the gradient while running the algorithm , this is because to cope with the non differentiability of the hinge loss function . In other words pegasos does stochastic gradient descent on the primal objective function of the svm .

In pegasos algorithm initially the w is set to zero . At every iteration the w will get updated by the randomly chosen training sample . With the help of the training example we can approximate the objective function as follows ,

$$f(\mathbf{w}; i_t) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \ell(\mathbf{w}; (\mathbf{x}_{i_t}, y_{i_t})) \ .$$

The subgradient of this above approximate objective is ,

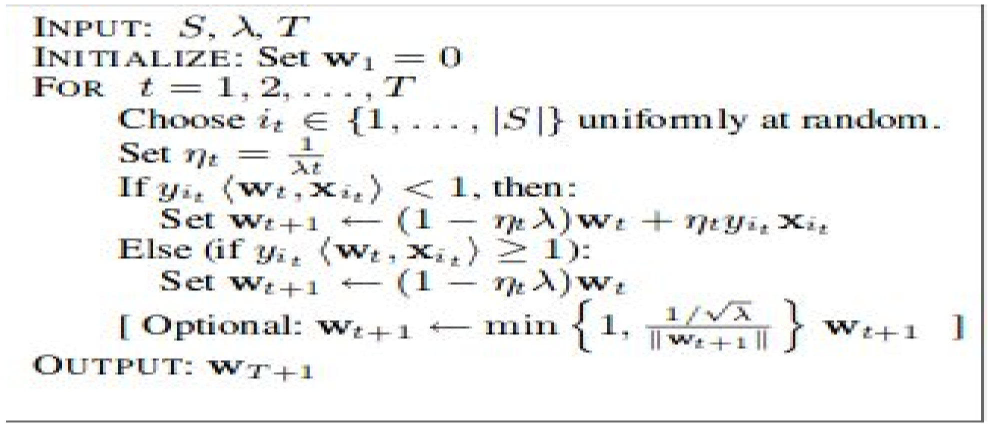
$$\nabla_t = \lambda\,\mathbf{w}_t - \mathbb{1}[y_{i_t}\,\langle \mathbf{w}_t, \mathbf{x}_{i_t}\rangle < 1]\, y_{i_t}\mathbf{x}_{i_t} \ ,$$

where1[y 〈w,x〉 <1]is the indicator function which takes a value of one if its argument is true (w yields non-zero loss on the example(x,y)), and zero otherwise . After finding out the sub gradient , similar to gradient descent method we update w as follows ,

$$w(t+1) = w(t) - eta * \nabla t .$$

Where eta = 1/(λt) .

Pseudo Code of the non kernelized pegasos algorithm:



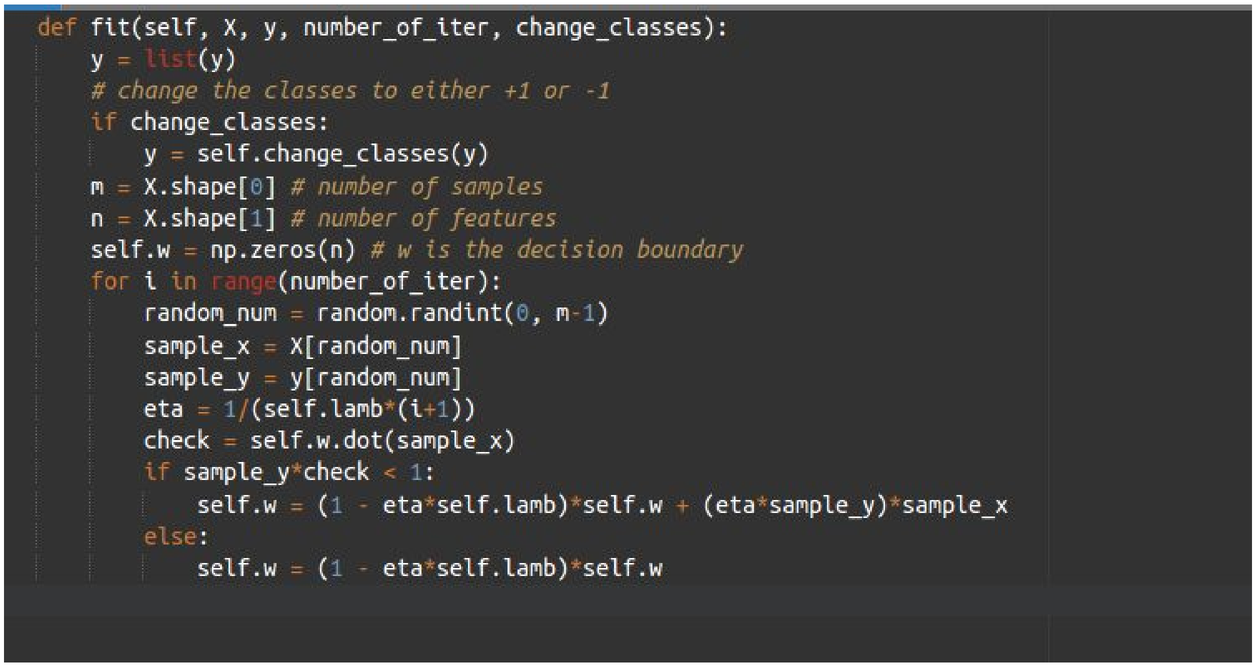
```
INPUT: $S, \lambda, T$
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
    Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.
    Set $\eta_t = \frac{1}{\lambda t}$
    If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:
        Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$
    Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):
        Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$
    [ Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$ ]
OUTPUT: $\mathbf{w}_{T+1}$
```

The above given pseudo code we only considered one random sample at every step , instead if we took n samples < |S| then it is called mini batch pegasos algorithm . In the project I have implemented the pegasos based on the above pseudo code instead of mini batch .

The parameter update for this algorithm is similar to the perceptron learning algorithm and can be extended by average pooling gradient with respect to multiple gradients .

The parameters or the initial constants in the algorithm are λ and T (number of iterations) . Depending upon these two values the output value of w will be changed . In the following sections we will briefly describe how the w changes depending upon these parameters with visualization .

Below given , is the code snippet from the code I have implemented for non kernelized pegasos algorithm .



```python
def fit(self, X, y, number_of_iter, change_classes):
    y = list(y)
    # change the classes to either +1 or -1
    if change_classes:
        y = self.change_classes(y)
    m = X.shape[0] # number of samples
    n = X.shape[1] # number of features
    self.w = np.zeros(n) # w is the decision boundary
    for i in range(number_of_iter):
        random_num = random.randint(0, m-1)
        sample_x = X[random_num]
        sample_y = y[random_num]
        eta = 1/(self.lamb*(i+1))
        check = self.w.dot(sample_x)
        if sample_y*check < 1:
            self.w = (1 - eta*self.lamb)*self.w + (eta*sample_y)*sample_x
        else:
            self.w = (1 - eta*self.lamb)*self.w
```

In the following sections I will show the working of the implemented algorithm on the banknote authentication dataset . It is a multivariate dataset having 4 attributes and have only two classes . You can find more about the dataset at this link https://archive.ics.uci.edu/ml/datasets/banknote+authentication .

Following the results obtained when our implemented pegasos algorithm was used for the classification of the above dataset .

```
 precision   recall  f1-score   support


        0       1.00   0.96   0.98          147
        1       0.96   1.00   0.98          128


   micro avg         0.98   0.98   0.98          275
   macro avg         0.98   0.98   0.98          275
weighted avg         0.98   0.98   0.98          275
```

We got a an accuracy of nearly 98 percentage when lambda = 0.1 and the number of iterations in algorithm is 1000 . For the inbuilt linear svm we got the following results ,

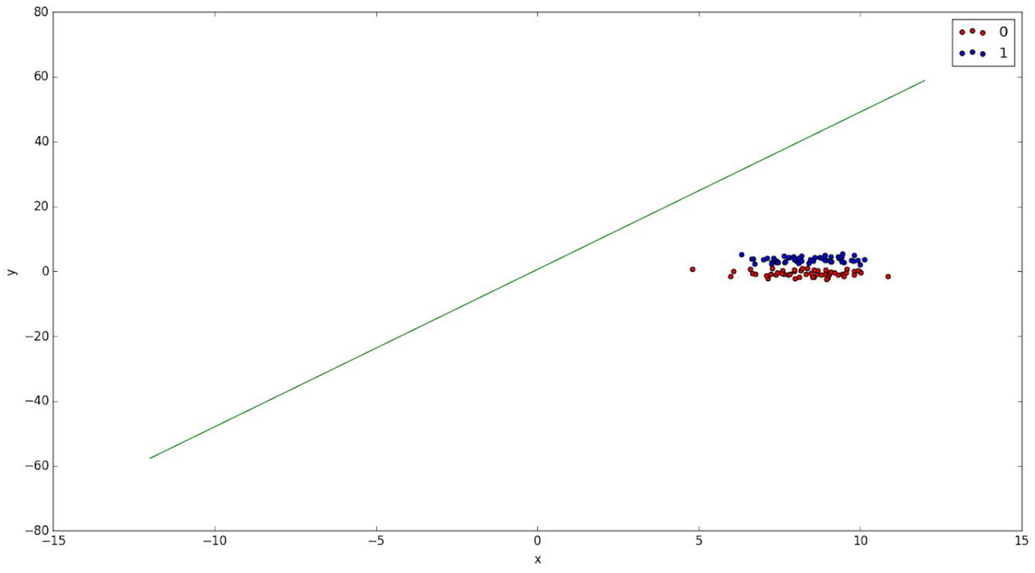
```
          precision   recall  f1-score   support


        0       0.99   0.99   0.99          147
        1       0.98   0.98   0.98          128


   micro avg         0.99   0.99   0.99          275
   macro avg         0.99   0.99   0.99          275
weighted avg         0.99   0.99   0.99          275
```

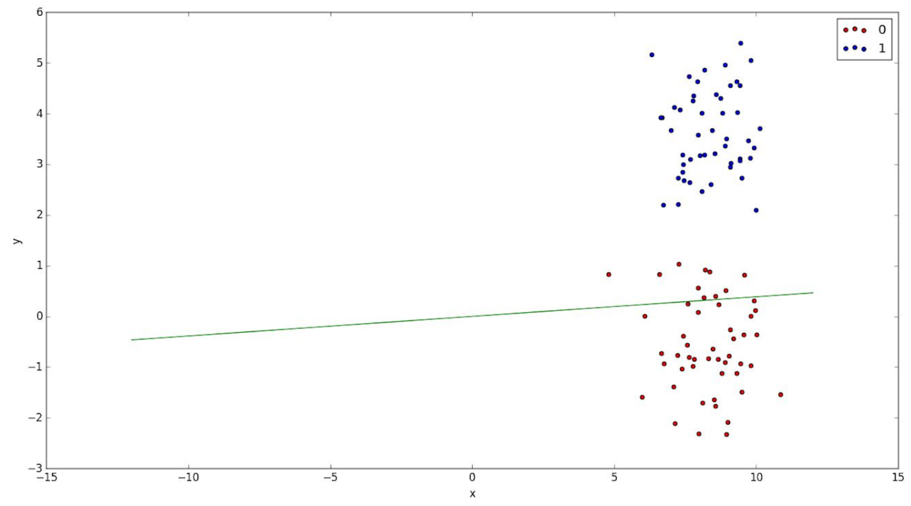
Following is the visualization of the how the decision boundary changes with number of iterations . For visualisation I have generated datasets having only 2 features .

Number of iterations = 10

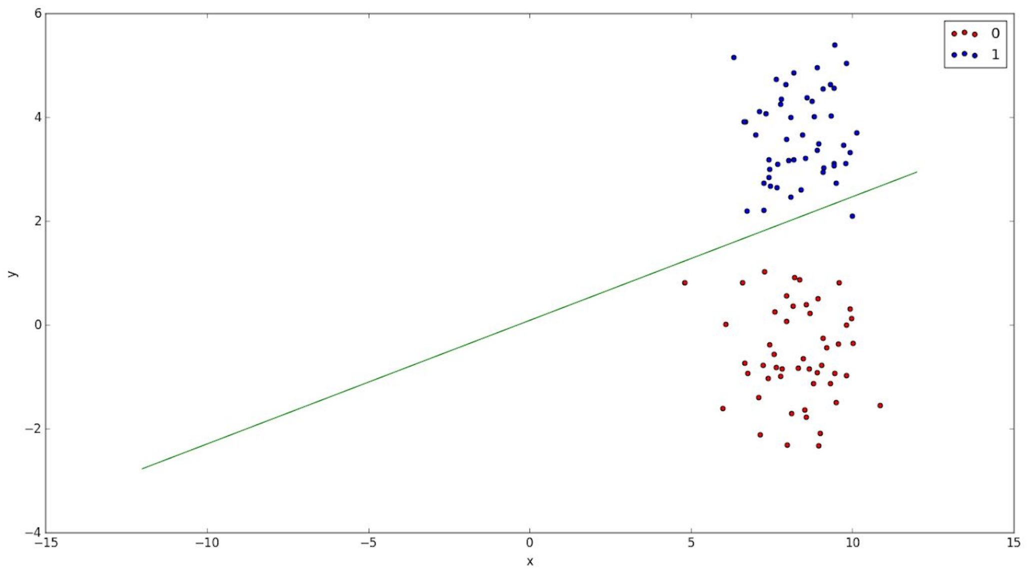


* Scale is different in this and below figures that is why they are looking differently .

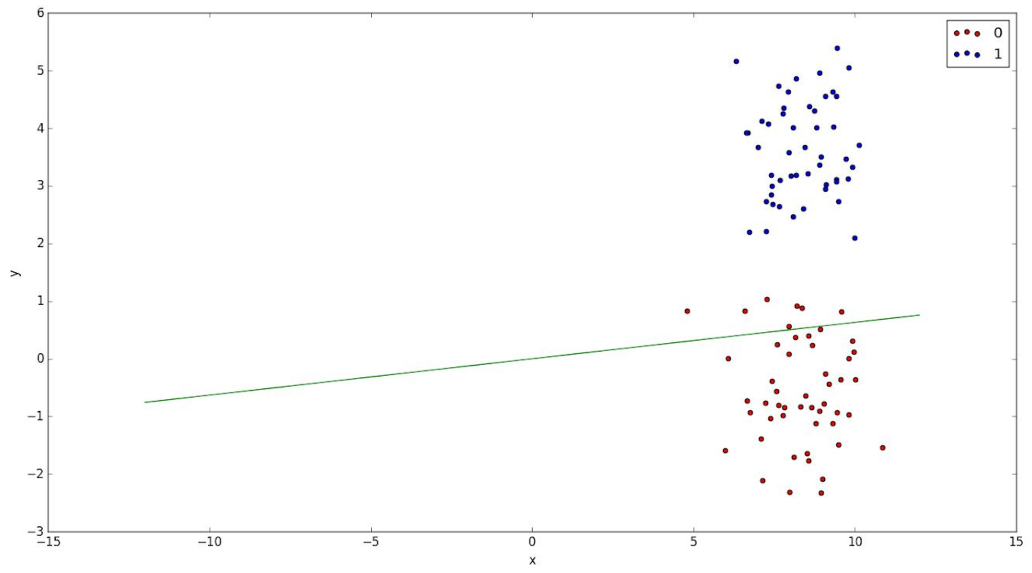
Number of iterations = 30



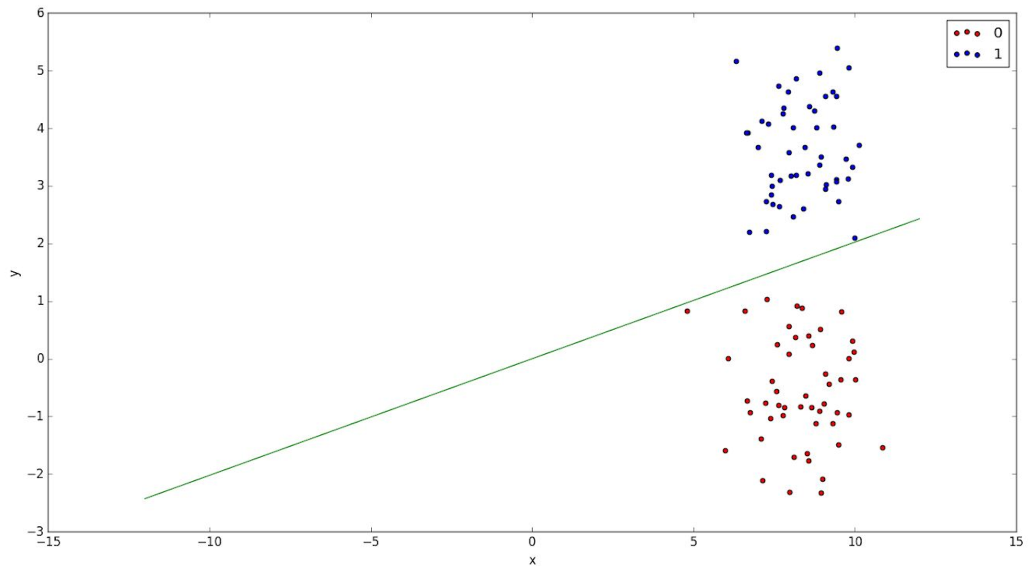
Number of iterations = 50



Number of iterations = 70



Number of iterations = 100



kernelized pegasos

Similar to svm dual problem we can also use the kernalization here too in the pegasos algorithm . This is one of the main benefits of SVMs that they can be used with kernels rather then with direct access to the feature vectors x . This property came from the representer theorem which states that the optimal solution can be expressed as a linear combination of the training instance . It is therefore possible to train and use a SVM without direct access to the training instances,and instead only access their inner products as specified through a kernel operator .

In kernel pegasos the problem becomes ,

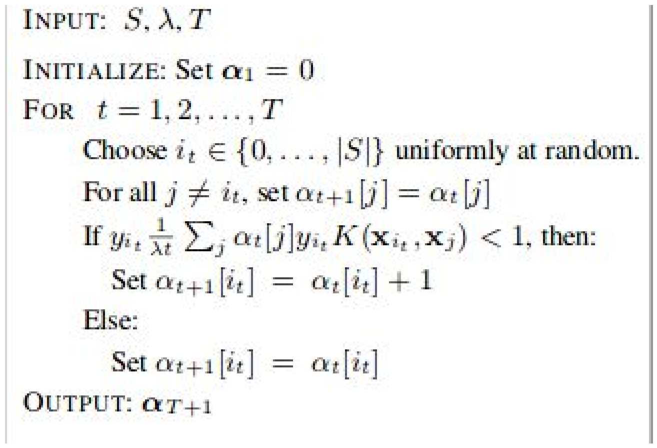
$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\phi(\mathbf{x}),y)) \ ,$$

Where ,

$$\ell(\mathbf{w};(\phi(\mathbf{x}),y)) = \max\{0, 1 - y\langle\mathbf{w},\phi(\mathbf{x})\rangle\} \ .$$

However, the mapping φ(·) is never specified explicitly but rather through a kernel operator K(x,x′) = 〈φ(x),φ(x′)〉 yielding the inner products after the mapping φ(·) .

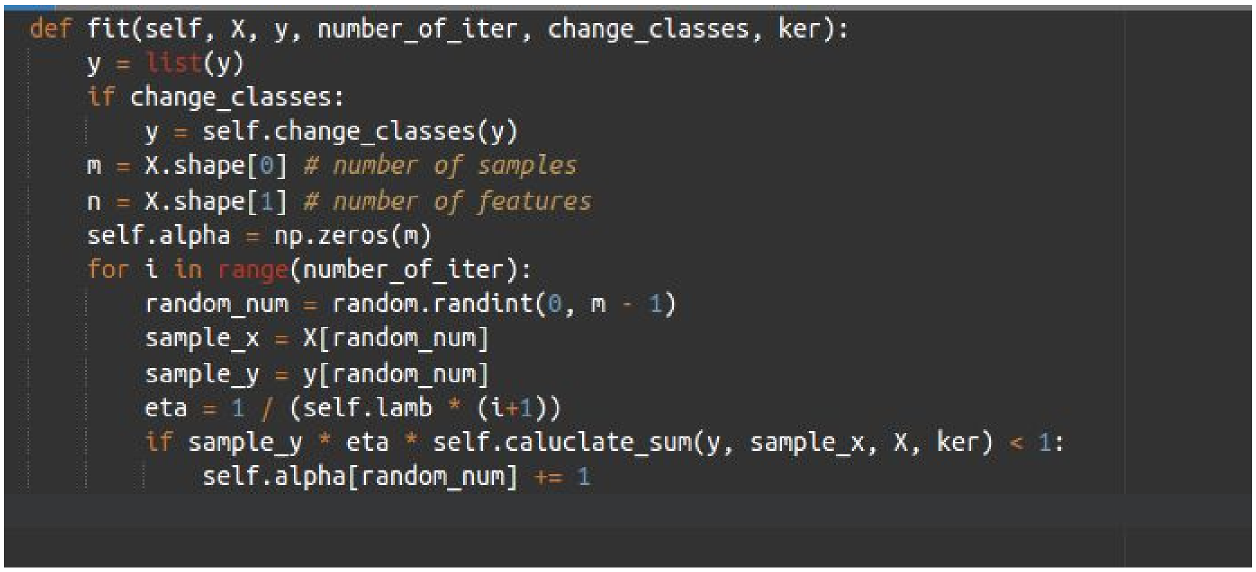
The pseudo code for the kernel pegasos is ,



```
INPUT: $S, \lambda, T$
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: $\boldsymbol{\alpha}_{T+1}$
```

Note that in the third step it is yj inside summation not yit .

The code snippet from my code for the above pseudo is ,



```python
def fit(self, X, y, number_of_iter, change_classes, ker):
    y = list(y)
    if change_classes:
        y = self.change_classes(y)
    m = X.shape[0] # number of samples
    n = X.shape[1] # number of features
    self.alpha = np.zeros(m)
    for i in range(number_of_iter):
        random_num = random.randint(0, m - 1)
        sample_x = X[random_num]
        sample_y = y[random_num]
        eta = 1 / (self.lamb * (i+1))
        if sample_y * eta * self.caluclate_sum(y, sample_x, X, ker) < 1:
            self.alpha[random_num] += 1
```

I have implemented the following kernels in the code :

1) Gamma radial basis function kernel

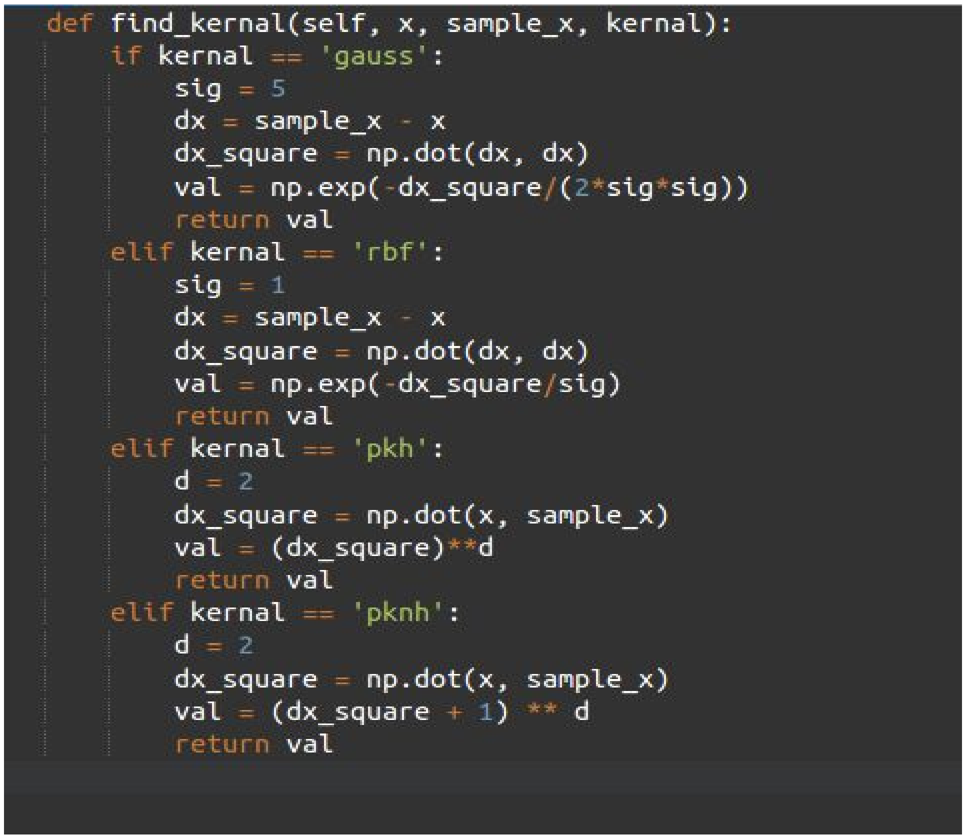
$$K(\mathbf{x}, \mathbf{x'}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x'}\|^2}{2\sigma^2}\right)$$

2) polynomial kernel

$$K(x, y) = (x^\mathsf{T} y + c)^d$$

Both homogeneous and non homogeneous .

The code snippet for the above functions are :



```
def find_kernal(self, x, sample_x, kernal):
    if kernal == 'gauss':
        sig = 5
        dx = sample_x - x
        dx_square = np.dot(dx, dx)
        val = np.exp(-dx_square/(2*sig*sig))
        return val
    elif kernal == 'rbf':
        sig = 1
        dx = sample_x - x
        dx_square = np.dot(dx, dx)
        val = np.exp(-dx_square/sig)
        return val
    elif kernal == 'pkh':
        d = 2
        dx_square = np.dot(x, sample_x)
        val = (dx_square)**d
        return val
    elif kernal == 'pknh':
        d = 2
        dx_square = np.dot(x, sample_x)
        val = (dx_square + 1) ** d
        return val
```

Following the results obtained when our implemented pegasos algorithm was used for the classification of the above banknote dataset .

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       144
           1       1.00      1.00      1.00       131

   micro avg       1.00      1.00      1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

Link to screen cast :

https://drive.google.com/open?id=1itAFN5RK9uCyI31J0kf3jgkoYHwmVk6D