CSE 481: Optimization Methods, Spring 2019
Pegasos: Primal Estimated sub-GrAdient SOlver for SVM

Sumukh S
20171404

Abstract

This report contains the details about the Pegasos: Primal Estimated sub-GrAdient SOlver for SVM and about its implementation. As the question statement asks for only details about the implemenation, like the gradient descent procedures, kernelization, loss functions used, etc., these have been explained, along with basic performance metrics of the optimization procedure.

1 Introduction

This method alternates between stochastic gradient descent steps and projection steps. The number of iterations required to obtain a solution of accuracy ϵ is $O(1/\epsilon)$.

The task of learning a support vector machine is cast as a constrained quadratic programming problem.

2 Pegasos algorithm

The algorithm receives 2 input parameters: T - number of iterations to perform; k - the number of examples to use for calculating sub-gradients. . Initially, we set w_1 to any vector whose norm is at most $1/squareroot(\lambda)$. On iteration t of the algorithm, we first choose a set $A_t \subset S$ of size k. Then, we use an approximate objective function,

$$f(\mathbf{w}; A_t) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{k}\sum_{(\mathbf{x},y)\in A_t} \ell(\mathbf{w}; (\mathbf{x}, y))$$

Note that we overloaded our original definition of f as the original objective can be denoted either as f(w) or as f(w; S). We interchangeably use both notations depending on the context . Next, we set the learning rate $\eta_t = 1/(\lambda t)$ and define A_t^+ to be the set of examples for which w suffers a non-zero loss. We now perform a two-step update as follows. We scale w_t by $(1 - \eta_t\lambda)$ and for all examples $(\mathbf{x}, y) \in A_t^+$ we add to w the vector $\frac{y\eta_t}{k}$x. We denote the resulting vector by $\mathbf{w}_{t+\frac{1}{2}}$. This step can be also written as $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t\nabla_t$, where

$$\nabla_t = \lambda\mathbf{w}_t - \frac{1}{|A_t|}\sum_{(\mathbf{x},y)\in A_t^+} y\mathbf{x}$$

INPUT: S, λ, T, k
INITIALIZE: Choose $\mathbf{w}_1$ s.t. $\|\mathbf{w}_1\| \leq 1/\sqrt{\lambda}$
FOR $t = 1, 2, \ldots, T$
Choose $A_t \subseteq S$, where $|A_t| = k$
Set $A_t^+ = \{(\mathbf{x}, y) \in A_t : y \langle \mathbf{w}_t, \mathbf{x} \rangle < 1\}$
Set $\eta_t = \frac{1}{\lambda t}$
Set $\mathbf{w}_{t+\frac{1}{2}} = (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{(\mathbf{x},y) \in A_t^+} y\,\mathbf{x}$
Set $\mathbf{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+\frac{1}{2}}\|}\right\} \mathbf{w}_{t+\frac{1}{2}}$
OUTPUT: $\mathbf{w}_{T+1}$

Figure 1: Pegasos algorithm

The definition of the hinge-loss implies that ∇_t is a sub- gradient of $f(\mathbf{w}; A_t)$ at $\mathbf{w}_t$. Last, we set $\mathbf{w}_{t+1}$ to be the projection of $\mathbf{w}_{t+\frac{1}{2}}$ onto the set

$B = \{\mathbf{w} : \|\mathbf{w}\| \leq 1/\sqrt{\lambda}\}$

That is, $\mathbf{w}_{t+1}$ is obtained by scaling $\mathbf{w}_{t+\frac{1}{2}}$ by $\min\left\{1, 1/\left(\sqrt{\lambda}\left\|\mathbf{w}_{t+\frac{1}{2}}\right\|\right)\right\}$. As we show in our analysis below, the optimal solution of SVM is in the set B. Informally speaking, we can always project back onto the set B as we only get closer to the optimum.

2.1 Adding a bias term

Incorporating a bias term: In many applications, the weight vector w is augmented with a bias term which is a scalar, typically denoted as b. The bias term often plays a crucial role when the distribution of the labels is uneven as is typically the case in text processing applications where the negative examples vastly outnumber the positive ones.

The approach simply amounts to adding one more feature to each instance x thus increasing the dimension to n + 1. The artificially added feature always take the same value. We assume w.l.o.g that the value of the constant feature is 1. Once the constant feature is added the rest of the algorithm remains intact, thus the bias term is not explicitly introduced. The analysis can be repeated verbatim and we therefore obtain the same convergence rate for this modification. Note however that by equating the n+1 component of w with b, the norm-penalty counterpart of f becomes $\|\mathbf{w}\|^2 + b^2$.

The disadvantage of this approach is thus that we solve a slightly different optimization problem. On the other hand, an obvious advantage of this approach is that it requires no modifications to the algorithm itself rather than a modest increase in the dimension and it thus can be used without any restriction on A_t.

2.2 Using Mercer Kernels

One of the main benefits of support vector machines is their ability to incorporate and construct non-linear predictors using kernels which satisfy Mercers conditions.

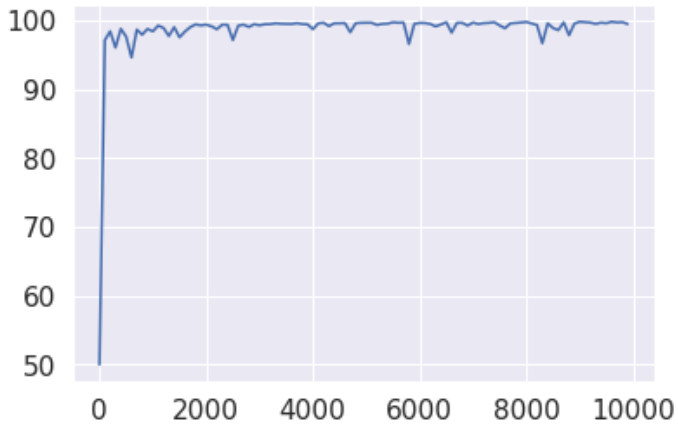

Figure 2: Iterations vs Accuracy for pairwise classifier between classes 0 and 1

The common approach for solving the optimization problem for SVM when kernels are employed is to switch to the dual problem and find the optimal set of dual variables. The approach is to directly minimize the primal problem while still using kernels. The main observation is that if w1 is initialized to be the zero vector, then at each iteration of the algorithm wt can be written as
$\mathbf{w}_t = \sum_{i \in I_t} \alpha_i \mathbf{x}_i$ where I_t is a subset of $\{1, \ldots, m\}$.

Based on the analysis in previous sections, Pegasos finds an ϵ-accurate solution using $\tilde{O}(1/(\delta\lambda\epsilon))$ iterations, while each iteration involves a single inner product between w and x.

Note however that each inner product operation between w and x may require $\min\{m, \tilde{O}(1/(\delta\lambda\epsilon)\})$ evaluations of the kernel function.

3 Experiments

Here, the implementation details are explained.

3.1 Iterations

The maximum iterations is set to be 1000000, for Pegasos, for each of the pairwise classifiers. This gives enough number of iterations for convergence. A plot of number of iterations vs accuracy for classification between the classes 0 and 1 is shown in the figure 2.

3.2 Time taken

The classifier takes 5-7 seconds to train a single pairwise classifier.

For the implemented multi-class classifier, it takes about 3.5 - 5 minutes, as there are $\sum_{i=1}^{9} = 45$ pairwise classifiers.

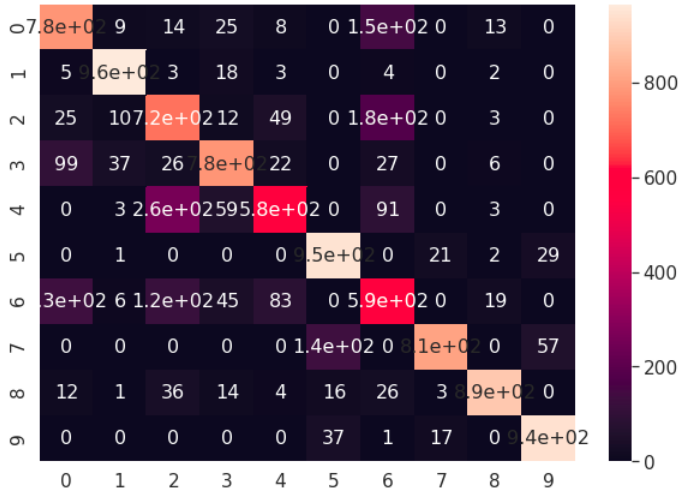

Figure 3: Confusion matrix

```
Training binary classifier between the classes 1 and 0
Classifier accuracy 99.60833333333333
Training binary classifier between the classes 2 and 0
Classifier accuracy 91.225
Training binary classifier between the classes 2 and 1
Classifier accuracy 99.99166666666667
Training binary classifier between the classes 3 and 0
Classifier accuracy 91.46666666666667
Training binary classifier between the classes 3 and 1
Classifier accuracy 98.45833333333334
Training binary classifier between the classes 3 and 2
Classifier accuracy 98.30833333333334
Training binary classifier between the classes 4 and 0
Classifier accuracy 99.02499999999999
Training binary classifier between the classes 4 and 1
```

Figure 4: Accuracies of pair-wise, binary classifiers

4 Results

The results for the multi-classifier turned out to be 80.2% as can be seen when the code is run.

The confusion matrix is shown in the figure 3.

The results of the pairwise classifiers, while training is shown in figure 4.

The screen cast video is here.

4.1 Comparison with the standard libraries:

The Scikit-learn library gave a classification accuracy of 80.37%

5 References

1. Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro - Pegasos: Primal Estimated sub-GrAdient SOlver for SVM