Project: Pegasos SVM

Chris Andrew
2018701019

Linear SVM:

Although a support vector machine model (binary classifier) is more commonly built by solving a quadratic programming problem in the dual space, it can be built fast by solving the primal optimization problem also. The Pegasos algorithm solves the primal optimization problem with sub-gradient solver using stochastic gradient descent.

The Hinge Loss is used for the minimising the misclassification penalty. Since the hinge-loss is not continuous, the sub-gradient of the objective is considered instead for the gradient computation for a single update step with SGD.

Formally, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{+1, -1\}$, find the minimizer of the problem

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

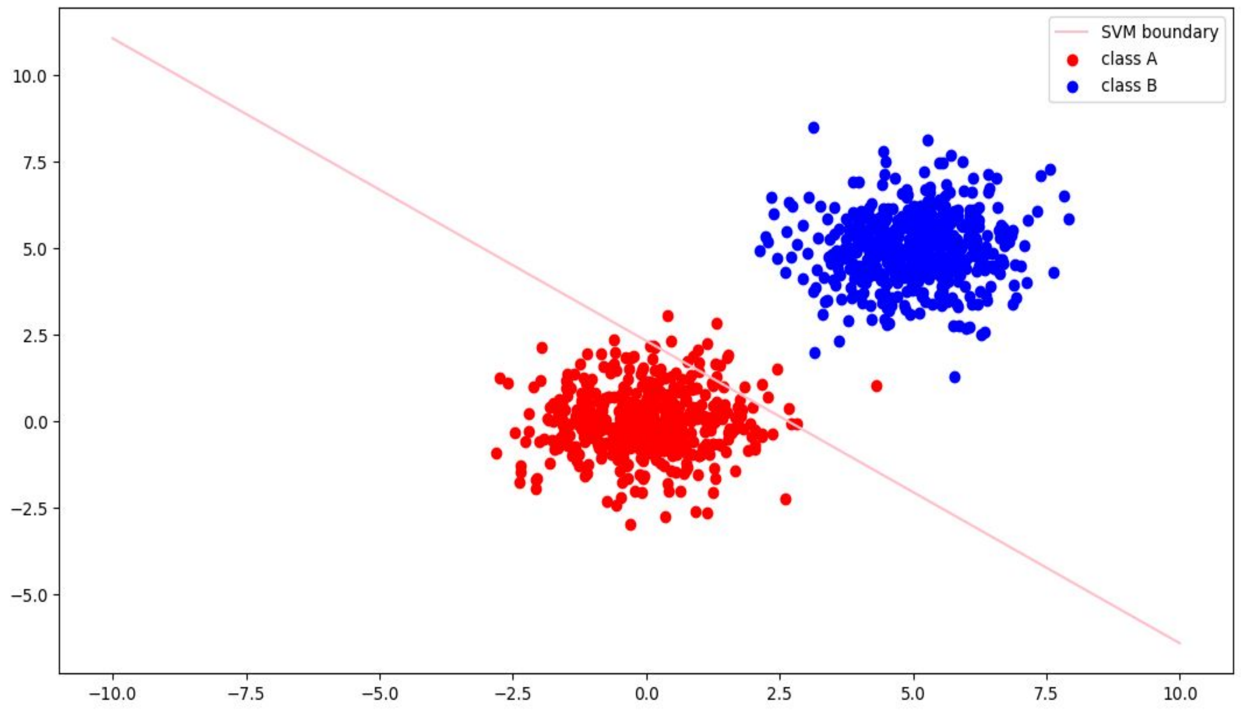
where $$\ell(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y \langle \mathbf{w}, \mathbf{x}\rangle\} \quad \text{(Hinge Loss)}$$

The algorithm for updating the weight vectors is given as:

```
Inputs: a list of example feature vectors $X$
        a list of outputs $Y$
        regularization parameter $\lambda$
        the number of steps $T$
$w = (0,\dots,0)$
for $t$ in $[1,\dots,T]$
  select randomly a training instance $x_i$, with a corresponding output label $y_i$
  $\eta = \frac{1}{\lambda \cdot t}$
  score = $w \cdot x_i$
  if $y \cdot$score $< 1$
    $w = (1 - \eta \cdot \lambda) \cdot w + (\eta \cdot y_i) \cdot x_i$
  else
    $w = (1 - \eta \cdot \lambda) \cdot w$
the end result is $w$
```

Results:

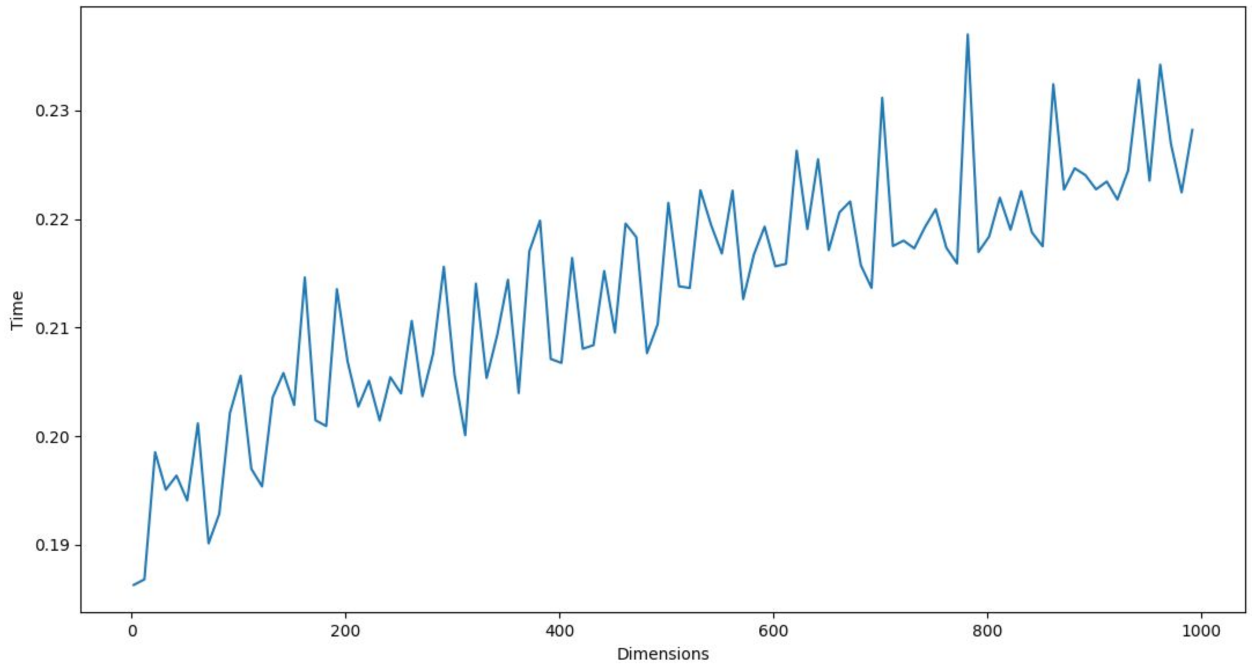
We created a toy dataset and tested the linear SVM on it[C=1]. The data was linearly separable. We were able to achieve 98.2% accuracy on this data. We also plotted the decision boundary to visualise the errors.



Fashion MNIST:
We also selected two classes from the Fashion MNIST dataset at random and ran the linear SVM on it. We were able to achieve on average **97.5%** accuracy on it.[C=1]

Time Complexity:
Time **would not vary** with number of samples since only one random sample per iteration is considered for updation.

We ran the linear SVM on random data and increased the dimensionality of the data while measuring time taken to train. The number of samples was kept constant to measure how the time taken varied with dimensionality.



In general, **time would increase linearly with dimensionality**, since all operations are linear operations. This is clearly seen in the graph.

Kernel SVM:

For kernel SVMs the kernel matrix is used to optimise the primal objective function. Hinge loss is again used, but this time the variables are modified accordingly.

Training then involves solving the minimization problem:

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

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

$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ yielding the inner products after the mapping $\phi(\cdot)$.

The algorithm for updating the weight vectors is given as:

```
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 $y_{i_t} \frac{1}{\lambda t} \sum_j \alpha_t[j] \, y_j \, K(\mathbf{x}_{i_t}, \mathbf{x}_j) < 1$, then:
        Set $\alpha_{t+1}[i_t] = \alpha_t[i_t] + 1$
    Else:
        Set $\alpha_{t+1}[i_t] = \alpha_t[i_t]$
OUTPUT: $\boldsymbol{\alpha}_{T+1}$
```

The Kernelized Pegasos Algorithm.

$$\mathbf{w}_{t+1} = \frac{1}{\lambda t} \sum \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \phi(\mathbf{x}_{i_t}) \rangle < 1] \, y_{i_t} \phi(\mathbf{x}_{i_t}) \, .$$

$$\alpha_{t+1}[j] = |\{t' \leq t : i_{t'} = j \wedge y_j \langle \mathbf{w}_{t'}, \phi(\mathbf{x}_j) \rangle < 1\}| \, .$$

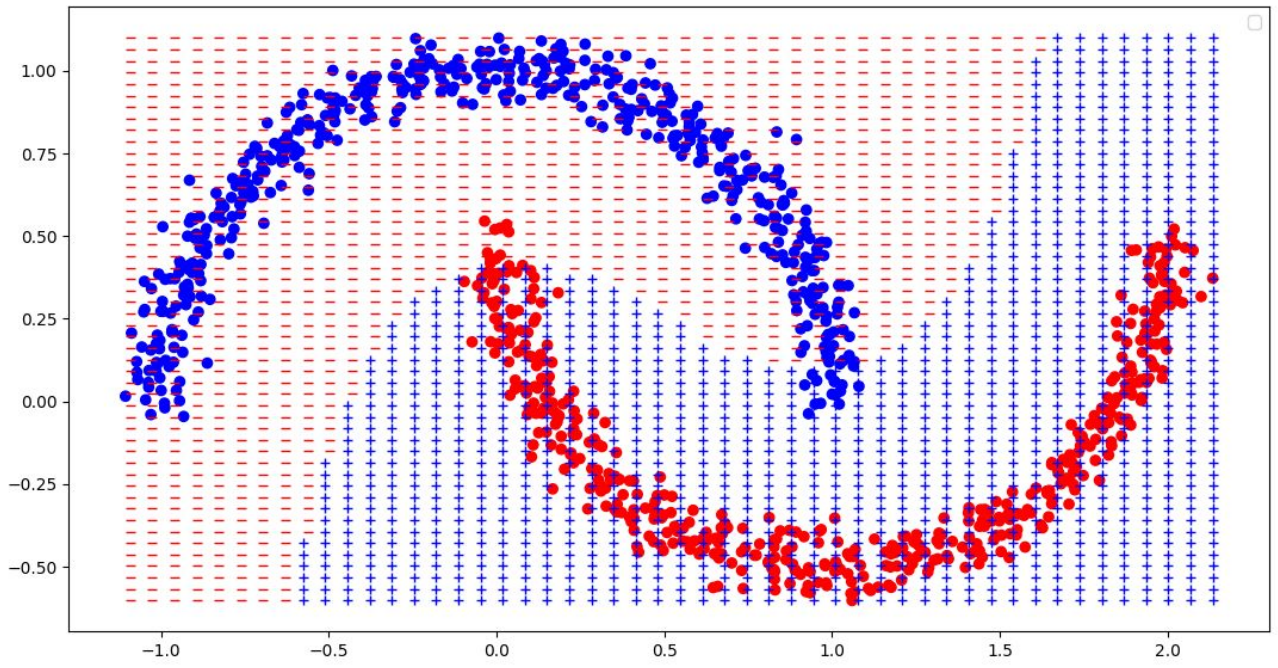
$$\mathbf{w}_{t+1} = \frac{1}{\lambda t} \sum_{j=1}^{m} \alpha_{t+1}[j] \, y_j \phi(\mathbf{x}_j) \, .$$

parametrized through $\boldsymbol{\alpha} \in \mathbb{R}^m$. The training objective can then be written in terms of the α variables and kernel evaluations:

$$\min_{\alpha} \frac{\lambda}{2} \sum_{i,j=1}^{m} \alpha[i]\alpha[j] K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \sum_{j=1}^{m} \alpha[j] K(\mathbf{x}_i, \mathbf{x}_j)\}$$

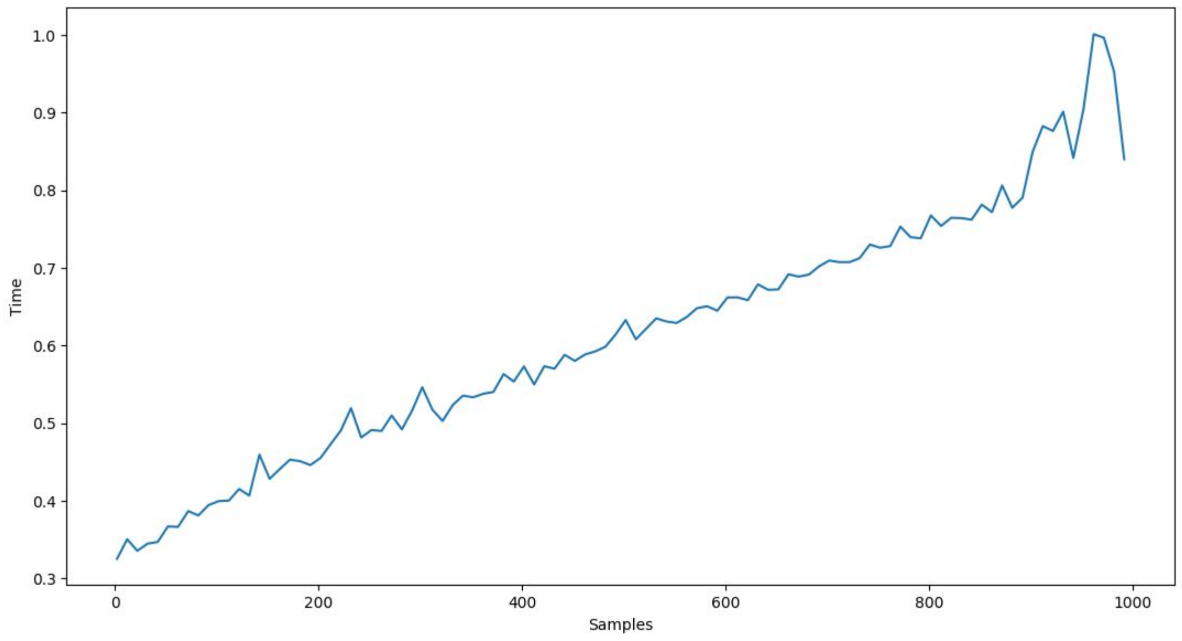
Results:

We created a toy dataset using the moon dataset from scikit-learn and tested the kernel SVM on it using an RBF kernel[C=0.1, sigma=0.5]. The data was not linearly separable. We were able to achieve 96.7% accuracy on this data. We also plotted the decision boundary to visualise the errors.

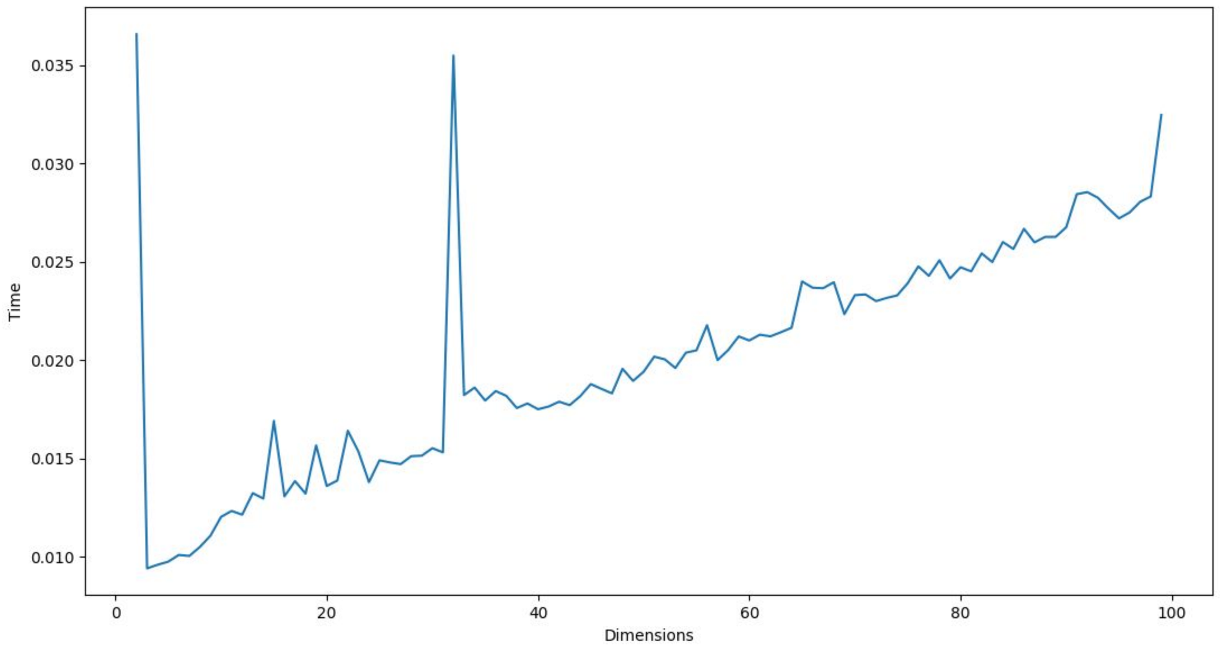


Fashion MNIST:
We also selected two classes from the Fashion MNIST dataset at random and ran the linear SVM on it. We were able to achieve on average **51.2%** accuracy on it.[C=0.005, sigma=0.5]. We did a hyperparameter search for finding the best values of C and sigma. Since the data is highly linearly separable, we did not achieve high results with an RBF kernel on Fashion MNIST.

Time Complexity:
Time **will vary** with the number of samples in this case since the kernel value is calculated using all the samples in the dataset. The number of dual variables also depends on the number of samples in the training data. It will **vary linearly with the number of samples**, since the number of iterations remain same, we just do more operations per iteration. If the model is trained on every sample, then time would vary exponentially. We plotted the graph to see how it varies with the number of samples for Pegasos, which clearly shows a linear relation.



We ran the linear SVM on random data and increased the dimensionality of the data while measuring time taken to train. The number of samples was kept constant to measure how the time taken varied with dimensionality.



In general, time would **increase linearly with dimensionality**, which is clearly seen in the graph[The spike in between may be because data generated is random in each case].

Demo:

A demo video of the code running is available at the given link.
https://drive.google.com/file/d/15K1UF3X5rD6Lj1K2jfWoYT9newE83WCH/view?usp=sharing

---