# Support Vector Machines: A simulation study

Harry Booth
harry.booth.19@ucl.ac.uk

December 18, 2021

**Abstract**

This paper outlines a simple simulation study, comparing the performance of two optimization algorithms used for solving the SVM optimization problem. The two optimization algorithms considered are Sequential Minimisation Optimisation (SMO) and Stochastic Gradient Descent (SGD). The study considers both linear and non-linear kernels, applied to a Gaussian 'blob' data set.

## 1 Introduction

Support Vector Machines (SVMs) can count themselves alongside the most popular modern learning algorithms. As well as being a popular tool for statistical learning, they have caught the attention of those working in large-scale numerical optimization, due to the non-trivial optimization problem that arises from the mathematical formulation of the method. Since their creation, many algorithms have been proposed to solve the optimization problem, each one often growing out of the specific requirements of a particular application domain. For a review see [1] [2].

I start this study by deriving the primal and dual problem for an SVM within the context of the binary classification problem. Here I state both the Hard and Soft Margin problems, describe the connections between the Karush-Kuhn-Tucker (KKT) conditions and the 'support vectors', and preemptively describe potential challenges associated with the optimisation problems. I then outline the two algorithms used in this study - Sequential Minimal Optimization, invented by John Platt in [3], and Stochastic Gradient Descent, which is now commonplace in many Machine Learning applications. I describe theoretical results and consider practical issues regarding their implementation. Finally I present the results of the numerical experiments. This includes a comparison of algorithm performance for varying dataset properties.

## 2 Mathematical Formulation

Suppose that we have a binary classification problem - we have a set of $m$ training examples $(y_i, x_i)_{i=1\cdots m}$ with labels $y_i \in \{-1, 1\}$ and features $x_i \in \mathbb{R}^n$. We wish to formulate a rule $f : \mathbb{R}^n \to \{-1, 1\}$, learnt from the data, which will allow us to assign with a good level of accuracy new labels to new features, i.e. $y_{\text{pred}} = f(x_{new})$.

### 2.1 Primal Problem

Assume that the datapoints are linearly separable, i.e using a hyperplane we can completely separate the examples belonging to each class. Following the intuition from above, we will define our decision boundary as the set of points $D = \{x \in \mathbb{R}^n : w^T x + b = 0\}$. This defines a hyperplane in the space $\mathbb{R}^n$. Our classification rule for an example $x_i$ is then given by

$$h_{w,b}(x_i) = g(w^T x_i + b) \tag{1}$$

where $g(z) = 1$ if $z \geq 0$ and $g(z) = -1$ otherwise. Given this rule, what conditions must $w \in \mathbb{R}^n$, $b \in \mathbb{R}$ satisfy in order to ensure all examples in the training set are given a correct label? We define the functional margin of $(w, b)$ with respect to the training example $(y_i, x_i)$ to be the following

$$\hat{\gamma_i} = y_i(w^T x_i + b) \tag{2}$$

Note that

$$\hat{\gamma_i} > 0 \iff y_i = h_{(w,b)}(x_i) \tag{3}$$

In other words, a positive functional margin indicates correct classification. Furthermore, a large functional margin represents a confident prediction - it is far away from the decision boundary. Intuitively, we want to determine a choice $(w, b)$ so that all training examples are categorized as confidently as possible. Unfortunately, maximising the functional margin does not achieve this in a meaningful way, since given any choice $(w, b)$ and $\alpha > 0$, $(\alpha w, \alpha b)$ produces a larger margin by a factor of $\alpha$ whilst maintaining the same label predictions. This motivates another measure of distance from the decision boundary - the geometric margin. The geometric margin of $(y_i, x_i)$ with respect to $(w, b)$ is defined as follows

$$\gamma_i = y_i \left( \left( \frac{w}{\|w\|} \right)^T x_i + \frac{b}{w} \right) \tag{4}$$

This has the following two properties. If $\|w\| = 1$ then it is equal to the functional margin, and its value is invariant to the scaling of $(w, b)$, which we noted as a shortcoming of the functional margin. One can also define the size of the geometric margin with respect to the entire trainset as $\gamma = \min_{i=1,\cdots,m} \gamma_i$.

This discussion motivates the following optimization problem:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \gamma \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq \gamma, \ i = 1, \ldots, m. \\ & \|w\| = 1 \end{aligned}$$

Solving this problem will result in $(w, b)$ with the largest geometric margin with respect to the training set, whilst ensuring some minimum functional margin. In fact, due to the $\|w\| = 1$ constraint the functional and geometric margins will be equal.

Unfortunately this constraint produces a difficult optimization problem, since it is non-convex. For this reason, it is necessary to transform this formulation into an easier but equivalent one. Using scaling arguments and the invariance of the geometric margin to scaling of $(w, b)$, it can be shown [4] that the problem is equivalent to the following:

$$\begin{aligned} & \min_{w,b} \frac{1}{2}\|w\|^2 \\ & \text{subject to} \quad y_i(w^T x_i + b) \geq 1, \ i = 1, \ldots, m. \end{aligned}$$

This is the primal problem for the Hard Margin SVM, which is relevant in the cases where one has a linearly seperable dataset. In the case where the dataset is not separable, a simple extension allows us to reformulate our optimisation problem. One permits certain examples to have a functional margin less than 1, however we add an L1 penalty term to the main objective to minimise the extent to which this occurs. Hence, the problem becomes

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i$$
$$\text{subject to } y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, \ldots, m.$$
$$\xi_i \geq 0\ i = 1, \ldots, m.$$

The parameter $C \in \mathbb{R}^+$ controls to what extent we penalise examples violating the functional margin constraint. In this sense, it plays the role of a regularization parameter. This is useful when one believes there may be outliers or noisy labelling present in the dataset, and can be tuned through processes such as cross-validation.

The above formulation is known as the primal problem for the soft margin SVM. Aside from extending the scope of the method to non-seperable datasets, it provides us with yet another formulation which is commonly used in Machine Learning applications. This formulation is known as the hinge-loss formulation, and is stated below:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \max(0, 1 - y_i(w^T x_i + b)) \tag{5}$$

This is now an unconstrained, convex problem in $(w, b)$. The term within the summation is known as the 'hinge-loss', which is a non-differentiable function. Note also that the number of terms within the summation grows linearly in the number of training examples. Whilst this unconstrained problem is at first sight easier than the constrained problems, the previous two points indicate that some of the usual unconstrained optimization methods may not be viable. This will be explored later in the study.

Now we have the primal problem, it makes sense to next investigate the corresponding dual problem, using lagrangian duality theory.

## 2.2 Dual Problem

Given the Hard Margin primal problem, we introduce positive Lagrange multipliers $\alpha_i\ i = 1, \cdots m$ for each of the $m$ inequality constraints. Rewriting the inequality constraints as $y_i(w^T x_i + b) - 1 \geq 0$, we form the following Lagrangian function for the primal problem:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i y_i (w^T x_i + b) + \sum_{i=1}^{m} \alpha_i \tag{6}$$

Given the Lagrangian, we can now state the value of the primal problem:

$$p^* = \min_{w,b} \max_{\alpha, \alpha \geq 0} \quad L(w, b, \alpha) \tag{7}$$

Similarly, we can state the value of the dual problem:

$$d^* = \max_{\alpha, \alpha \geq 0} \min_{w,b} \quad L(w, b, \alpha) \tag{8}$$

Looking again at our primal problem, we note that our unconstrained objective function $\frac{1}{2}\|w\|^2$ is convex. Furthermore, the constraints are all affine. Therefore, as a consequence of Slater's Theorem (Strong Duality Theorem) the duality gap is zero, and hence $p^* = d*$. Therefore,we can solve the dual problem in place of the primal problem, obtaining the same optimum parameters $(w, b, \alpha)$ through each method. To find the dual form of the problem explicitly, differentiate $L(w, b, \alpha)$ with respect to $w$ and $b$ for fixed alpha, and solve for zero. Simplifying, we obtain a function of $\alpha$,

$$L(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \tag{9}$$

Hence to solve the dual problem, we must maximise the above, subject to constraints on $\alpha$. These constraints come from the KKT conditions associated with the primal problem. The KKT conditions associated with the primal problem may be stated as follows

$$w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0 \tag{10}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{11}$$

$$y_i(w^T x_i + b) - 1 \geq 0 \quad i = 1, \cdots, m \tag{12}$$

$$\alpha_i \geq 0 \quad i = 1, \cdots, m \tag{13}$$

$$\alpha_i(y_i(w^T x_i + b) - 1) = 0 \quad i = 1, \cdots, m \tag{14}$$

It can be shown that the KKT conditions are a set of necessary and sufficient conditions for $(w, b, \alpha)$ to be a solution, since the SVM is a convex problem (convex objective function, with constraints giving a convex feasible region).

Adding the necessary constraints on $\alpha$ results in the dual problem, which I will now state. To make the form of the problem clearer, I define $H \in \mathbb{R}^{m \times m}$ where $H_{i,j} = y_i y_j \langle x_i, x_j \rangle$. The dual problem is then

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^m} \quad & \frac{1}{2}\alpha^T H \alpha - \sum_{i=1}^{m} \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha \\ & y^T \alpha = 0 \end{aligned}$$

$H$ is symmetric and positive definite, and hence this is a constrained quadratic optimization problem which is well defined.

We see that there is a one to one relationship between the Lagrange multipliers and training examples. Equation 14 is known as the KKT complementary condition. For our problem, it tells us two things; if $\alpha_i > 0$ then we must have $y_i(w^T x_i + b) - 1 = 0$, and hence the training example $(y_i, x_i)$ has a functional margin equal to 1. These training examples are known as support vectors. These are the training examples which lie closest to the decision boundary, and are the critical examples determining the hyper-plane.

The KKT conditions also show us how to determine $(w, b)$ after determining $\alpha$. Using 10 we can determine $w = \sum_{i=1}^{m} \alpha_i y_i x_i$. Furthermore, we can use 14 to determine $b$. This can be done by taking a single $\alpha_i > 0$ and solving for $b$, or alternatively, using all the equations associated with the support vectors to produce a more stable estimate. This can be done in the following way. Letting $X^1_{\text{support}} = \{x_i : \alpha_i > 0 \text{ and } y_i = 1\}$ and defining $X^{-1}_{\text{support}}$ similarly, it can be shown [4] that the following expression yields $b$ satisfying the KKT conditions:

$$b = -\frac{\left(\max_{x \in X^{-1}_{\text{support}}} w^T x + \min_{x \in X^{1}_{\text{support}}} w^T x\right)}{2} \tag{15}$$

From these expressions, one can show that the output of our svm for a new example is given by

$$f(x_{\text{new}}, \alpha, b) = \sum_{i=1}^{m} \alpha_i y_i \langle x_i, x_{\text{new}} \rangle + b \tag{16}$$

Note that $\alpha_i = 0$ for all non-support vectors. Given a small number of support vectors, the number of terms in the summation is therefore significantly reduced, offering a computationally efficient form of prediction for new examples.

For the soft margin problem, we derive the dual problem in the same way. The resulting soft margin dual problem is stated below:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^m} \quad & \frac{1}{2}\alpha^T H \alpha - \sum_{i=1}^{m} \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha \leq C \\ & y^T \alpha = 0 \end{aligned}$$

The support vectors now correspond to the examples with lagrange multipliers satisfying $0 < \alpha_i < C$.

## 2.3 Kernels

We have looked at two different formulations of the optimization problem associated with an SVM, the primal problem and the dual problem. From now on we will assume that we have a non-separable dataset, and hence will use the soft margin formulation. This is more representative of real world datasets, and allows one to regularize the solution through the parameter $C$. Below we recall the dual problem:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^m} \quad & \frac{1}{2}\alpha^T H \alpha - \sum_{i=1}^{m} \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha \leq C \\ & y^T \alpha = 0 \end{aligned}$$

Where $H \in \mathbb{R}^{m \times m}$, $H_{i,j} = y_i y_j \langle x_i, x_j \rangle$.

The output of the SVM is then given by $f(x_{\text{new}}, \alpha, b) = \sum_{i=1}^{m} \alpha_i y_i \langle x_i, x_{\text{new}} \rangle + b$ where

$$b = -\frac{\left( \max_{x \in X^{-1}_{\text{support}}} f(x, \alpha, 0) + \min_{x \in X^{1}_{\text{support}}} f(x, \alpha, 0) \right)}{2} \tag{17}$$

In this formulation, we note that all computations involve the inner product $\langle x_i, x_j \rangle$. We might want to transform features $x \in \mathbb{R}^n$ into some higher dimensional space so that non linear decision boundaries can be learnt. Hence, one might wish to replace $\langle x_i, x_j \rangle$ with $\langle \phi(x_i), \phi(x_j) \rangle$ where $\phi(x) : \mathbb{R}^n \longrightarrow \mathbb{R}^d$ where $d >> n$. Calculating $\phi(x)$ and then the inner product can naively be computed in $O(d)$ time. However, supposing there exists a function $K$ such that $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$, this inner product can be computed in $O(n)$ time. In this way one obtains the benefits of working in a higher dimensional space for no extra computational cost. Furthermore, it is also not necessary to explicitly construct the feature map $\phi$. This is a consequence of Mercer's Theorem. Suppose we have a function $K : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$, then $\exists \phi$ s.t $K(x, z) = \langle \phi(x), \phi(z) \rangle$ $\forall x, z$ if the Kernel matrix associated with $X_m = \{x_w, x_2, \cdots x_m\}$ is symmetric positive definite, for any $X_m$ where $m < \infty$.

Given this theorem, it is easy to incorporate Kernels into the dual problem - We simply replace $\langle x_i, x_j \rangle$ with $K(x_i, x_j)$ for some suitable Kernel function. This is known as the kernel trick. Our matrix $H$ is then symmetric positive definite since it is a valid Mercer Kernel, and we have a well defined quadratic optimisation problem.

It is less clear how one can incorporate kernels into the primal problem. Chapelle in [5] showed that one can include Kernels into the primal formulation. I will use this approach for training kernel classifiers using SGD.

### 2.4 Challenges

The QP problem that arises from the dual formulation cannot be easily solved using standard QP techiniques due to the scale of the problem. The matrix $H$ scales quadratically with the number of training examples. Any method which deals with an explicit matrix construction will quickly become infeasible as the number of training points scales. The linear constraint $y^T\alpha$ is also tricky.

The QP problem arising from the primal problem looks simpler than the dual problem. It also might be easier to circumvent the dimension of the problem scaling with the number of examples. However, it doesn't appear to offer a simple way to introduce the kernel trick.

Chapelle [5] argues that one should choose either the dual problem or primal problem dependent on which is larger - the number of examples $m$ or the dimension of the feature vectors $n$. It can be shown that the number of operations to compute and invert the primal problem is $O(mn^2 + n^3)$, and the for the dual it is $O(nm^2 + m^3)$. Hence, if the number of examples is very large one should choose the primal, and if the dimension of the features is very large one should choose the dual problem.

I will investigate algorithms which solve both the primal and dual problems,to enable comparison through experiment.

## 3 Algorithms: SMO and SGD

### 3.1 SMO: Sequential Minimal Optimisation

The SMO algorithm was invented specifically for solution of the SVM dual problem by John Platt in 1998 [3]. The algorithm follows a decomposition strategy. Heuristically, decomposition methods choose a subset of the components of $\alpha$, and approximately solve a sub problem in just these components whilst keeping the value of the other components fixed. There are now numerous decomposition methods which solve the SVM dual problem - see [6],[7]. The SMO algorithm was the first of such methods, and specifically considers the case where the size of the sub-problem is 2. The derivation and resultant method can be found in its entirety in [3]. Simplified pseudo-code can be found in [8].

Convergence of the SMO algorithm rests on a result shown by Osuna et al [9]. The theorem proves that a large QP problem can be broken into a series of smaller QP problems. As long as at least one example that violates the KKT conditions is added to the examples for the previous sub-problem, each step will reduce the overall objective function and maintain a feasible point that obeys all of the KKT constraints. Therefore, a sequence of QP sub-problems that always add at least one violator will be guaranteed to converge.

The SMO has the following advantages:

- Due to its decomposition approach, it is memory efficient and unaffected (from a memory perspective) by the size of the trainset. For example, it avoids the construction of any gradients or hessians.
- SMO solves each sub problem analytically. Hence, it avoids any issues associated with numerical techniques such as ill-conditioning, numerical rounding errors and associated instabilities. Each sub problem is also solved extremely quickly.
- SMO is simple to implement and therefore to troubleshoot. It is simple to amend the algorithm for the inclusion of kernels.

The SMO has the following disadvantages:

- The number of variables when solving in the dual scales linearly with the number of train examples. The matrix $H$ scales quadratically. If the solutions $\alpha$ are not particularly sparse (many support vectors),many outer (subset selection) iterations are required.

## 3.2 SGD: Stochastic Gradient Descent

Stochastic gradient descent is an adaption of the extremely common optimisation approach known as gradient descent. In the context of support vector machines, it is a method applied to the hinge-loss formulation of the primal, in order to directly determine $(w, b)$. I state the hinge formulation again for reference:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \max(0, 1 - y_i(w^T x_i + b)) \tag{18}$$

Stochastic gradient descent proposes to minimise the above approximation through stochastic approximation to the actual gradient. Whilst the hinge loss is not differentiable, one can bypass this difficulty by computing sub gradients instead. Sub gradients are generalizations of gradients to non-differentiable functions. Formally, a vector $g$ is subgradient to $f$ at point $x$ if

$$f(y) \geq f(x) + G^T(y - x) \tag{19}$$

The way in which stochastic gradient descent approximates the problem is as follows; select a random batch $I_k \subset \{1, 2, \cdots, m\}$, and then compute the (sub) gradient of

$$\frac{1}{2}\|w\|_2^2 + C\frac{m}{|I_k|}\sum_{i \in I_k} \max(0, 1 - y_i(w^T x_i + b)) \tag{20}$$

Using this (sub) gradient approximation, gradient descent is performed where the step size is simply decreasing in $k$ according to a fixed schedule. Often $n_k = k^{-1}$ or $n_k = k^{-\frac{1}{2}}$.

In order to be computationally cheap, let $|I_k| = 1$. At iteration $k$ we denote the index of this random example by $j_k$. Denote the subgradient of (20) by $g_k$. Then starting from $w = 0$ and $b = 0$, stochastic gradient descent makes the following updates:

$$(w_{k+1}, b_{k+1}) \leftarrow (w_k, b_k) - \eta_k g_k \tag{21}$$

where it can be shown that

$$g_k = \begin{cases} (w, 0) & \text{if } 1 - y_{j(k)}(w^T x_{j(k)} + b) \leq 0 \\ (w, 0) - CNy_{j(k)}(x_{j(k)}, 1) & \text{otherwise.} \end{cases} \tag{22}$$

When one talks about convergence properties of stochastic algorithms,convergence statements usually relate to the expected value of the objective function. For the SVM problem and step lengths $n_k = \frac{CN}{k}$, it can be shown that the expected value of the objective function s within $O(k^{-1}\log k)$ of the optimal within $k$ iterations.

Advantages of SGD:

- Cheap to compute. Using batch size greater than one means one can exploit code parallelization. Method is therefore suitable for extremely large train sets.
- Easy to implement
- Recent research [10] has suggested that SGD has a regularizing properties. This can be useful if one wants to obtain a regularized solution without changing the problem formulation, although its properties are not fully understood.

Disadvantages of SGD:

- Stochastic algorithm, hence one might have to run multiple tests to ensure solution stability

- Not clear how to define a good stopping rule - the solution path will fluctuate naturally due to its stochastic nature. The size of these fluctuations are not known a priori, and hence determining when the algorithm has finished 'improving' is difficult.
- Solution will always be more approximate than other numeric deterministic methods

## 4 Numerical Experiments

The purpose of the numerical experiments is to investigate the performance of SMO and SGD under a number of changing data set properties. The properties examined will be; number of samples, dimension of feature vectors and the degree of separation between the classes.

### 4.1 Datasets

For the numerical experiments, I have opted to use an artificial dataset which can be generated using the Sk-Learn python library. The function used is

```
sklearn.datasets.make_blobs(n_samples=100, n_features=2, centers=None, cluster_std=1.0)
```

This samples multivariate isotropic Gaussians with specified mean and standard deviation. The primary reason for using this data set is flexibility; it will allow me to investigate the performance of each algorithm under different types of data set.

An example data set with a fitted Gaussian kernel decision boundary determined via SMO is shown below. The data set is 2-dimensional and consists of 50 examples.

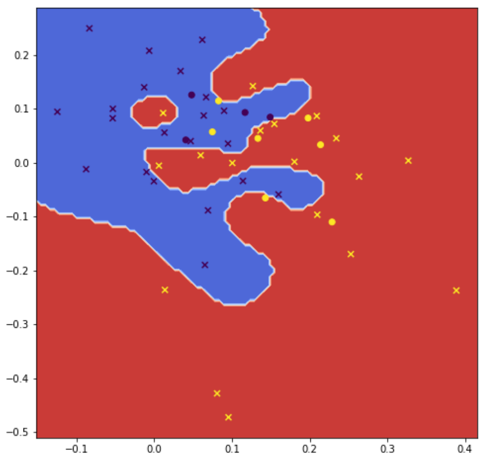


Figure 1: SMO - Gaussian Kernel Example

### 4.2 Convergence and Performance characteristics

#### 4.2.1 Varying sample size

First I vary the sample size of the data set. In other words, I vary $m$ where $x_i \in \mathbb{R}^m$. For the linear and kernel dual problem, this has the effect of increasing the dimensionality of $\alpha$, since for the dual problem $\alpha \in \mathbb{R}^m$. Furthermore, the matrix $H \in \mathbb{R}^{m \times m}$ scales quadratically with the number of training examples.

For the linear primal problem in its hinge formulation, it increases the number of terms within the summation, however the variable $w \in \mathbb{R}^n$ is unaffected. For kernel learning in the primal, the kernel matrix $K$ increases quadratically with the number of training examples. The dimensionality of $w$ also increases, since now $w \in \mathbb{R}^m$ - see [5].

From a computation perspective, one would expect SGD to demonstrate faster CPU performance particularly in the linear case. This is because it is unaffected by the number of terms within the summation, since it only sub-samples a (user specified) fixed number of terms to approximate the gradient. Furthermore, the implementation of the SMO algorithm contains a slow outer loop which iterates over all of the $\alpha$ components. To my knowledge it is not possible to parallelize this part of the computation, since each pass is dependent on the complete history of passes.

Below I show a comparison of the CPU computation time for increasing sample size. The dotted lines indicate the CPU times and the solid the number of iterations. For this demonstration, the number of iterations for SMO corresponds to the total number of outer and inner loop passes, and in SGD the total number of approximate gradients computed, and the number of subsequent steps taken.

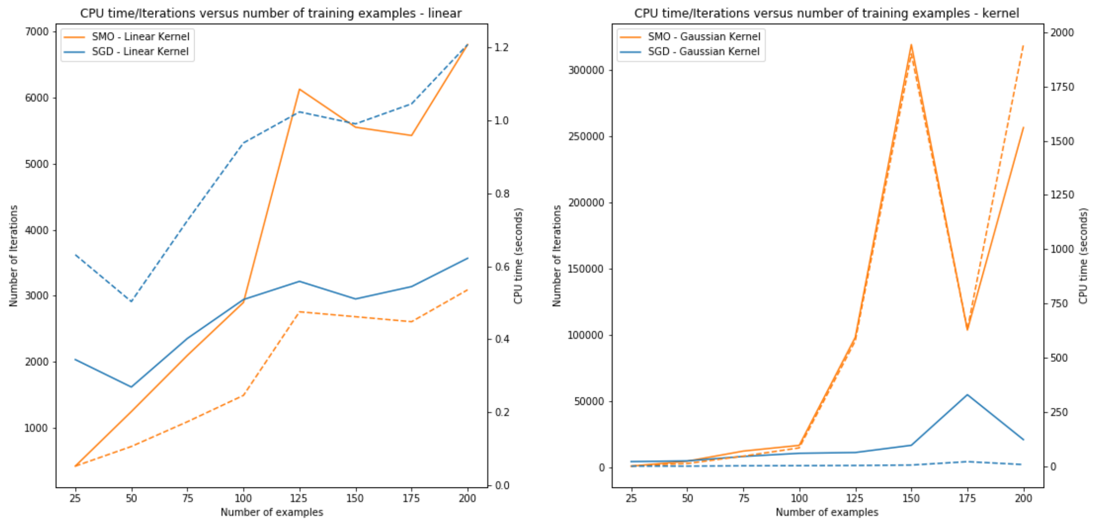


Figure 2: CPU time and Iterations - Changing Sample Size

The performance in the linear case is broadly comparable. However, the CPU time of SMO is consistently lower for all sample sizes. In the kernel case, the number of iterations necessary for the SMO algorithm to achieve the specified tolerance is considerably higher, particularly as the number of examples exceeds 100. The fewer iterations means that the CPU time of SGD is significantly lower.

Now we look at the convergence properties of the SMO algorithm. Below is the 2-norm residual of each iterate compared against the optimum (dual) objective function value. The optimum solution to the dual in each experiment was computed using CVXPY. In this demonstration, iterations in SGD correspond to the number of epochs (i.e. the number of steps taken in an approximate gradient direction). In SMO each iteration corresponds to a complete outer loop pass.

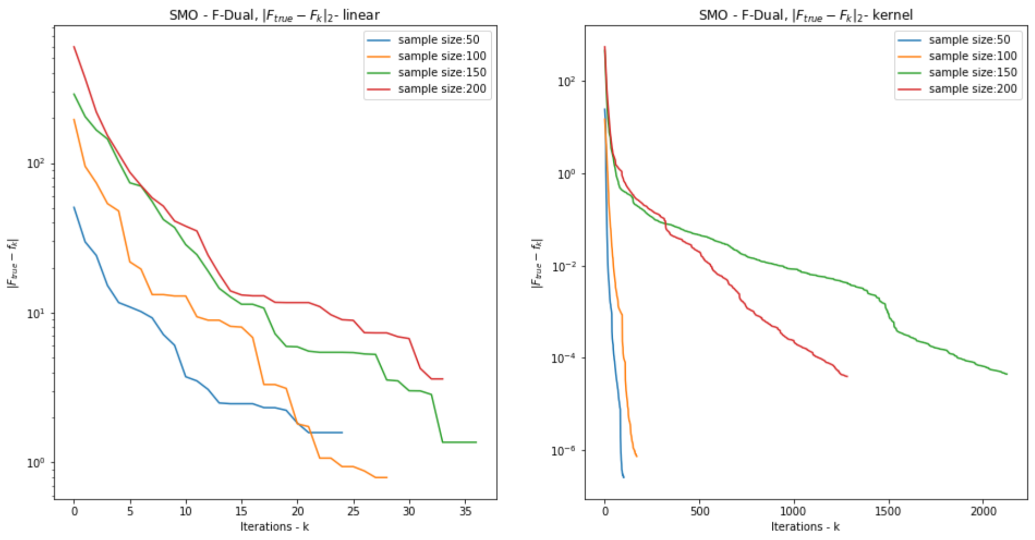
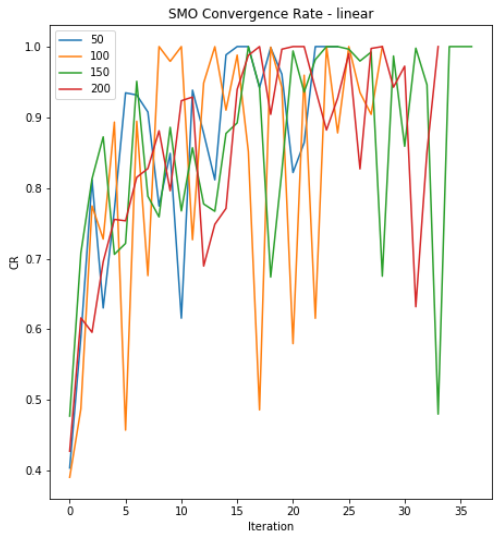


Figure 3: SMO - Dual Objective Residual - Changing Sample Size

In the linear case, the rates at which the residual decreases is comparable for all sample sizes. For kernel learning, we can see a significant departure for $m = 150, 200$. This could be due to the large number of support vectors in these examples - kernel learning significantly increases the number of support vectors, and the SMO algorithm is known to perform poorly as this number increases.

Below I plot the convergence rate of the SMO algorithm in all cases. I define the convergence at iterate $k$ for the trajectory path $(\alpha_k)_{k=0}^{L}$ to be $c_k = \frac{\|\alpha_{k+1} - \alpha_L\|}{\|\alpha_k - \alpha_L\|}$.



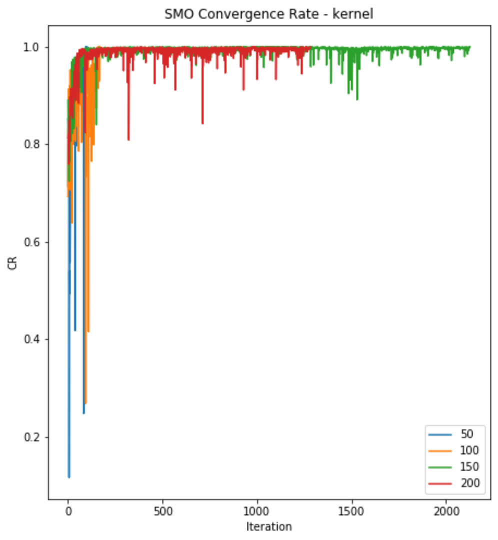


Figure 4: SMO - Convergence Rate - Changing Sample Size

Although it is noisy, the empirical convergence rates computed appear to suggest sub-linear convergence. This is in agreement with the theory of SMO - see [11].

Now we look at the convergence properties of the SGD algorithm. Below is the 2-norm residual of each iterate compared against the optimum (primal-hinge) objective function value. It is has been shown that the expected value of the objective function f is within $O(k^{-1}\log(k))$ after $k$ iterations. I plot this bound in purple for a suitable constant.

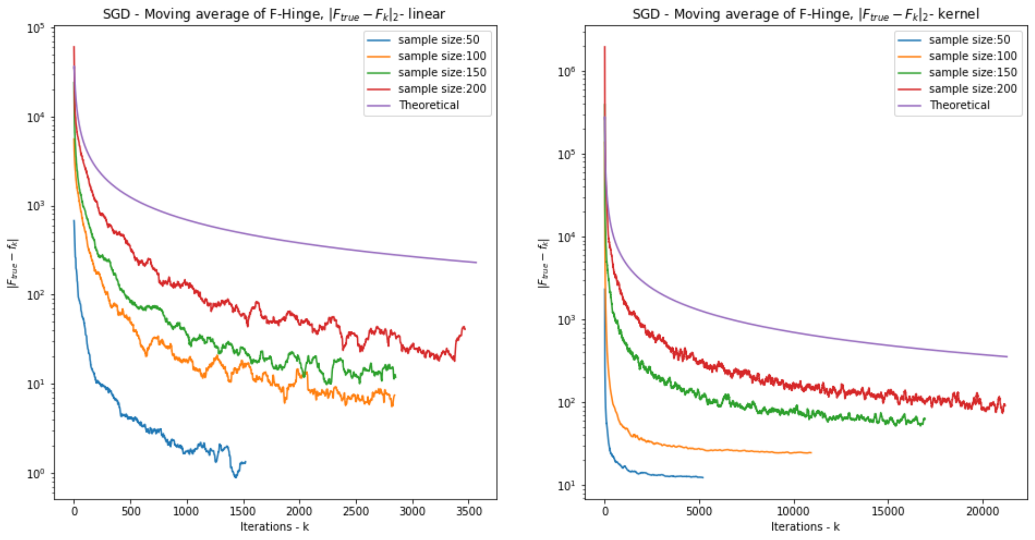


Figure 5: SGD - Hinge Objective Residual - Changing Sample Size

We see that the residuals appear to satisfy the theoretical bound. Furthermore, there is a clear progression in the number of iterations required and the subsequent optimality of the solution as the number of training examples increases. To be explicit, larger number of examples requires more iterations to reach a specified optimality in the objective function. Below I plot the convergence rate. The top row shows the rate over the full number iteration, and the second zooms in by starting from iterate 100 (hence removing the large drop at the beginning).

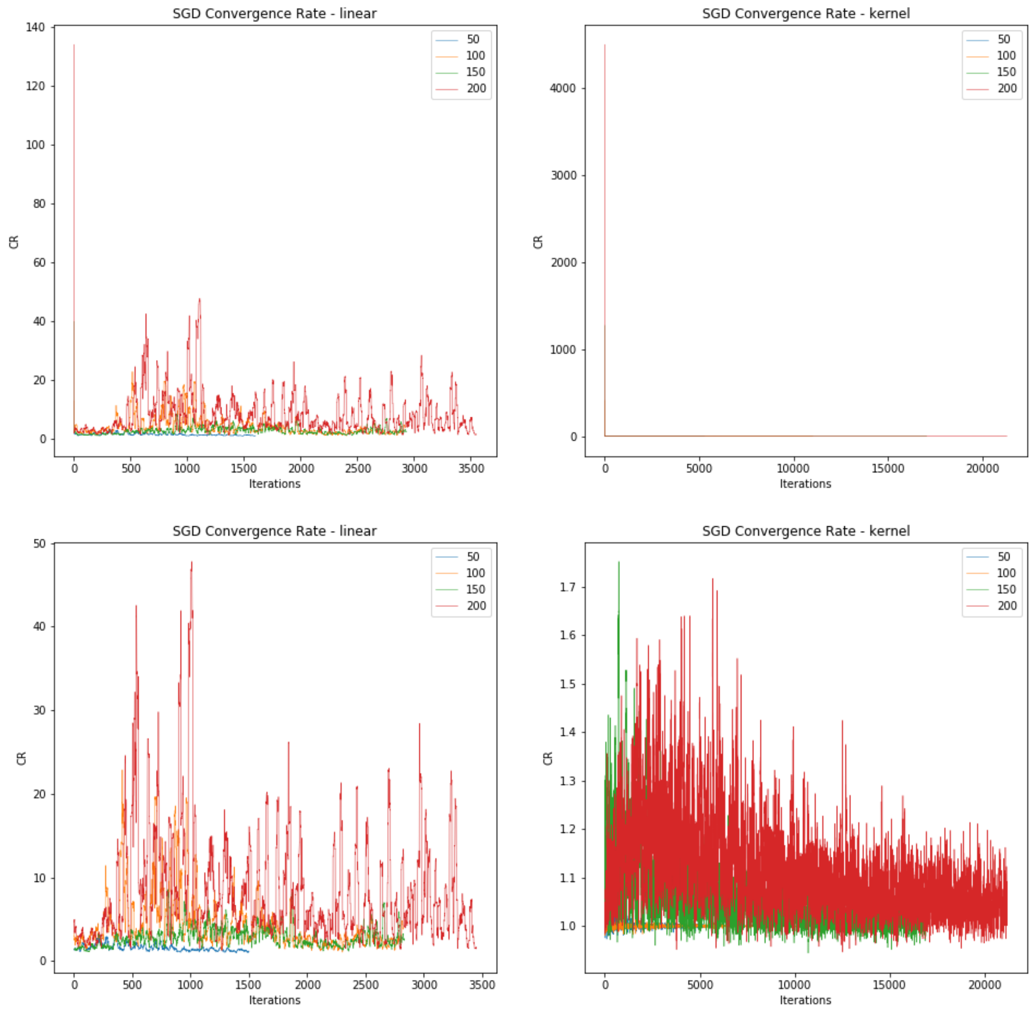


Figure 6: SGD - Convergence Rate - Changing Sample Size

The first thing to note is the noise present in the ratio, particularly when the number of examples is higher, and when considering the kernel problem. It is difficult to draw any conclusions about the convergence rate from the charts. It does suggest however that improvements to the algorithm might consider changing the step length regime $\eta_k$. For my experiments, $\eta_k = \frac{1}{k}$. However, some authors suggest a more aggressive reduction in step size once the estimate gets close to the region of solution. This might suppress the noise seen in the estimate and hence the rate of convergence.

### 4.2.2 Varying dimension

Now I vary the dimension $n$ of the feature vectors $x_i \in \mathbb{R}^n$. For the linear primal problem, this has the effect of increasing the dimensionality of $w$, since for the linear primal problem $w \in \mathbb{R}^n$. The dimension has less of an effect on the dual formulation from the perspective of problem size, since the features only appear in the problem when evaluated within the dot products. This is also true with both the linear and kernel problem.

Below I show a comparison of the CPU computation time for increasing number of features. The dotted lines indicate the CPU times and the solid the number of iterations.

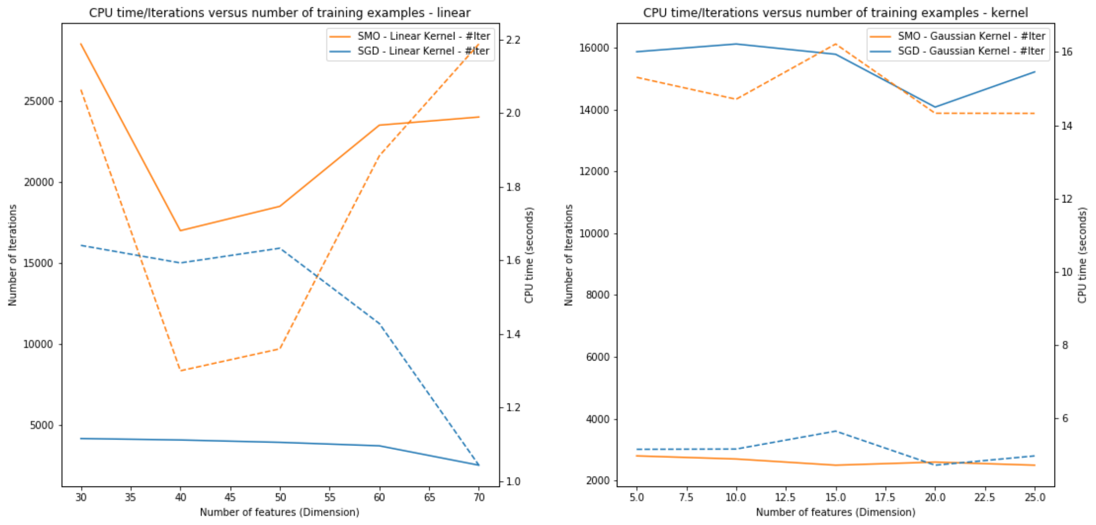


Figure 7: CPU time and Iterations - Changing Dimension

The results are quite mixed. For the linear problem, the SGD algorithm appears to outperform the SMO algorithm, both in terms of the number of iterations and the resulting computation time. Interestingly, the CPU time for SGD decreases for the higher dimensional problem.

The hypothesis that the dimension has little affect on the kernel formulation appears to be supported by the right hand side figure. Again, we see that the SGD outperforms the SMO algorithm on CPU computation time, despite requiring far more iterations.

Now I look at the convergence properties of the SMO algorithm. Below is the 2-norm residual of each iterate compared against the optimum (dual) objective function value as the dimension is varied.

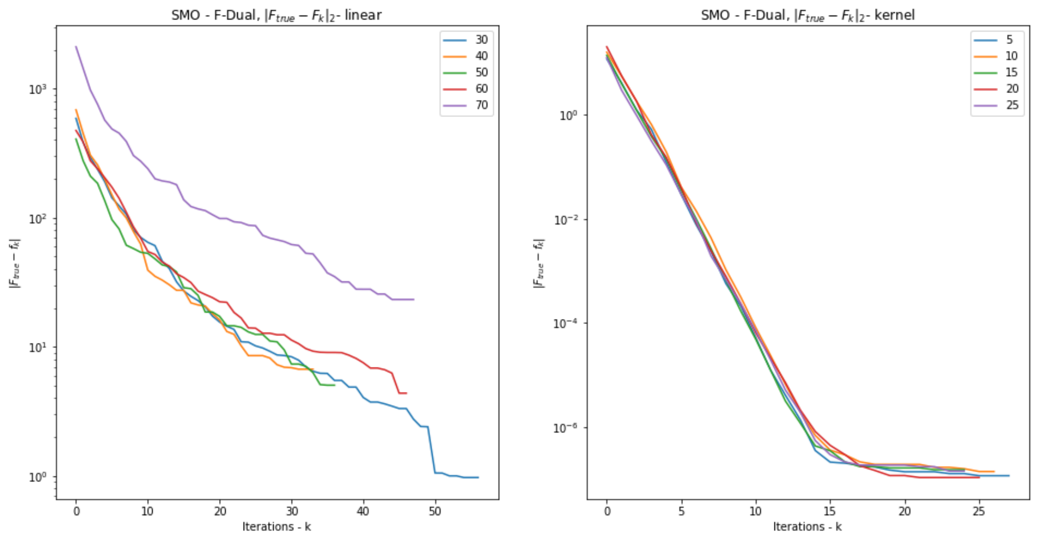


Figure 8: SMO - Dual Objective Residual - Changing Dimension

We see that the kernel problem is seemingly unaffected by the dimension of the feature vectors - all show a similar reduction across iterations. Next I plot the convergence rate.

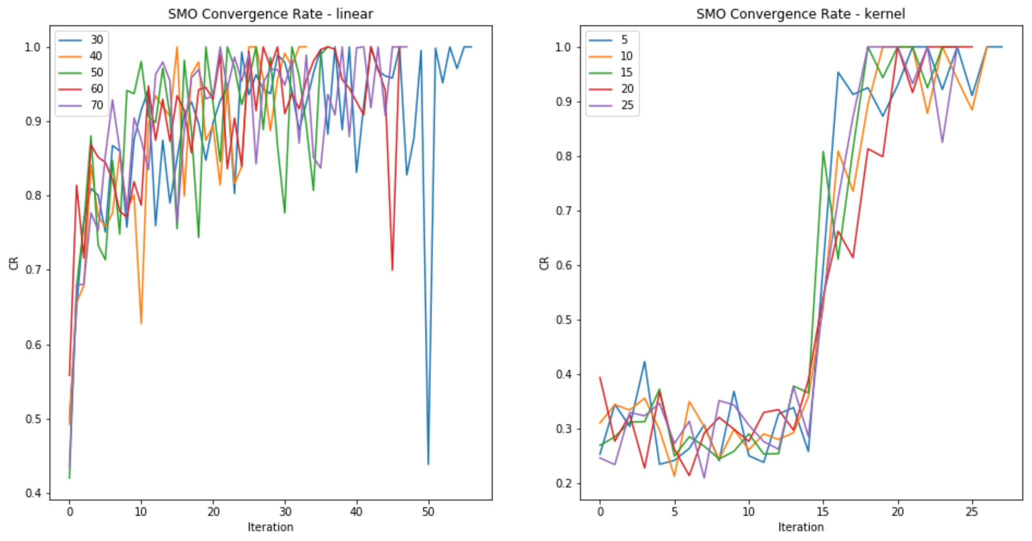


Figure 9: SMO - Convergence Rate - Changing Dimension

The charts suggest sub-linear convergence. The slower convergence during the initial iterations for the kernel method is interesting.

Now I investigate the convergence of the SGD algorithm. The 2-norm residual of each iterate compared against the optimum (primal-hinge) objective function value is displayed below, along with the theoretical bound.

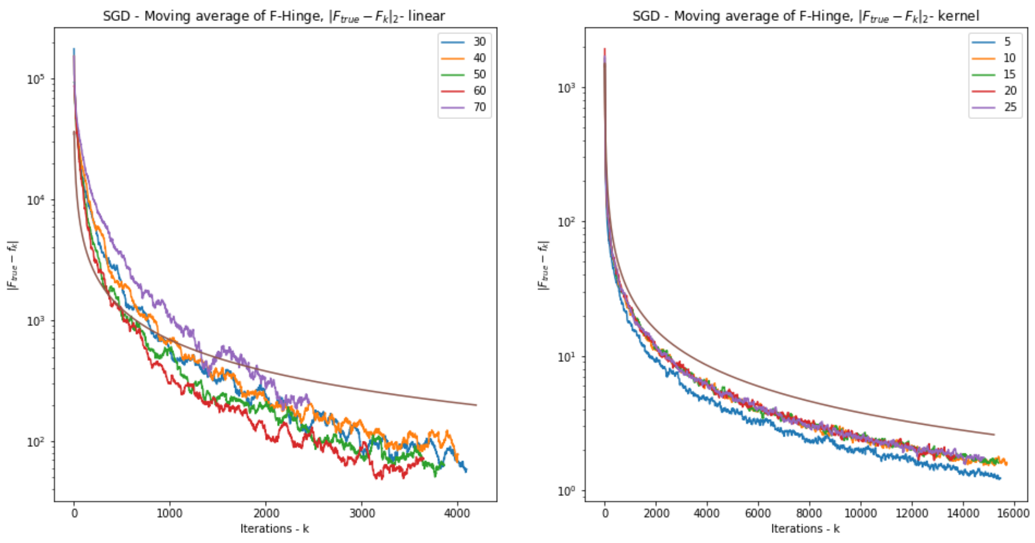


Figure 10: SGD - Hinge Objective Residual - Changing Dimension

The residuals appear to satisfy the bound. Similarly to the SMO algorithm, increasing the number of features does not appear to significantly affect the rate at which the optimum solution is approached, or the number of iterations required.

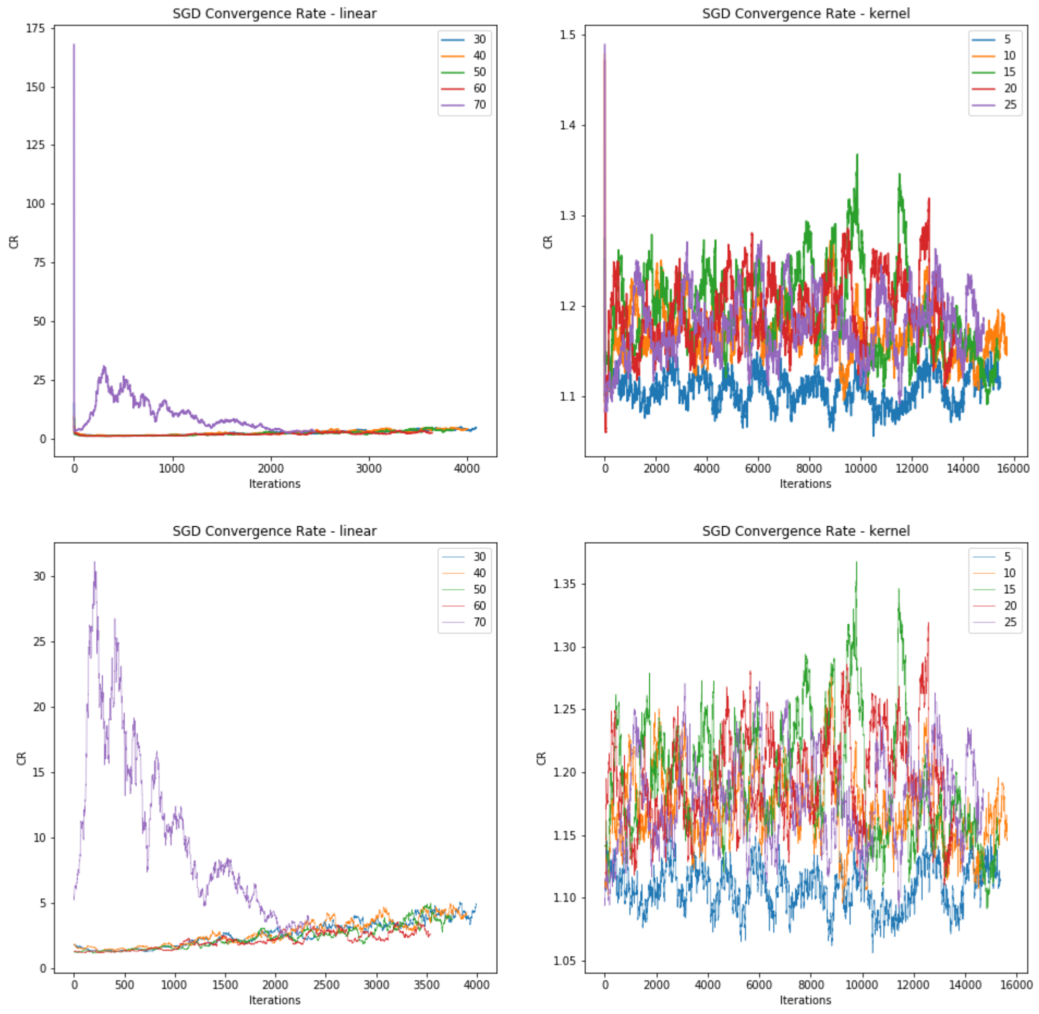


Figure 11: SGD - Convergence Rate - Changing Dimension

### 4.2.3 Varying separability

Now I decreasing the 'separability' of the feature vectors $x_i \in \mathbb{R}^n$. This involves increasing the variance of each Gaussian within the 'blob' dataset, whilst keeping the first moment of each the same. Intuitively, this should make fitting a decision boundary harder, and hence have an observable affect on the performance of SMO and SGD.

First I compare CPU computation time for increasing variance (decreasing separability). The dotted lines indicate the CPU times,the solid line the number of iterations.

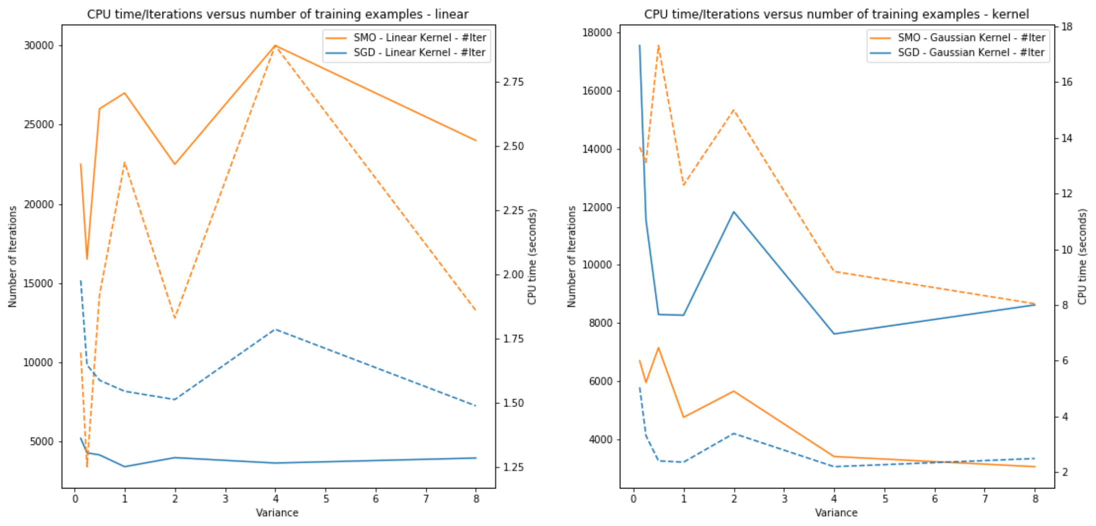


Figure 12: CPU time and Iterations - Changing Separability

Controlling for each method, there is no clear relationship between CPU time/Iteration count as the variance within the dataset is increased. Interestingly, the SMO algorithm outperforms SGD in the Kernel problem, in terms of the CPU computation time.

Now I plot the dual objective function 2-norm residual at each iterate, as well as the computed convergence rate.

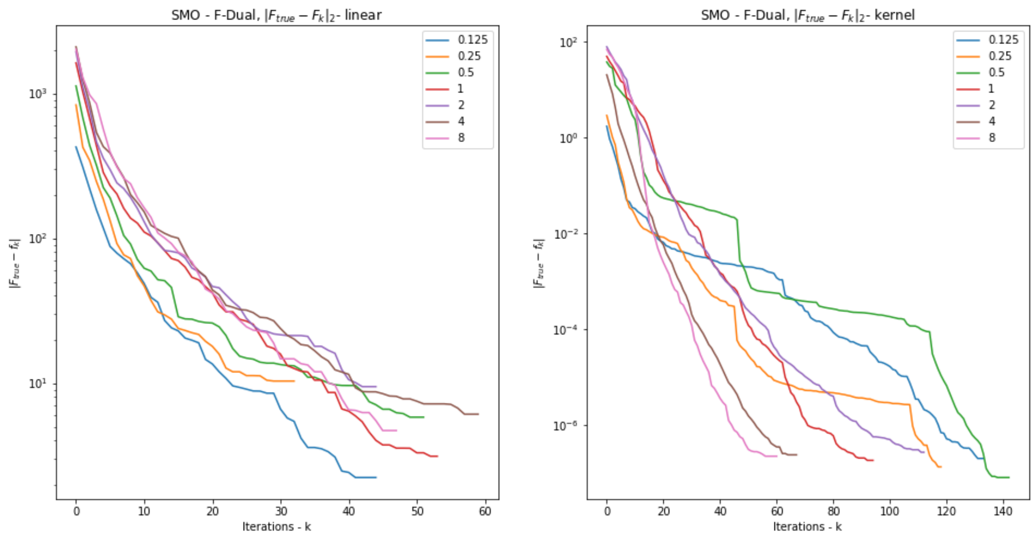


Figure 13: SMO - Dual Objective Residual - Changing Separability

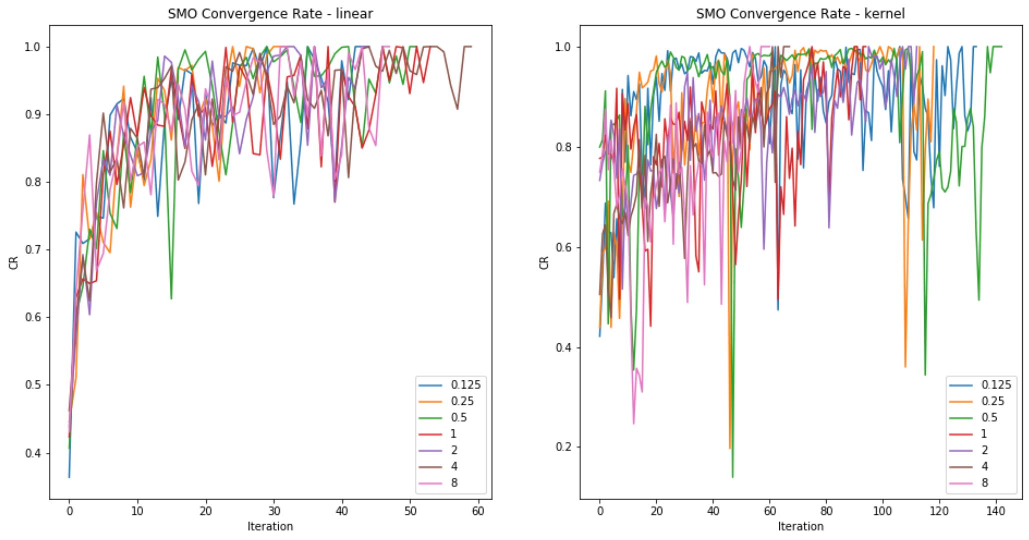


Figure 14: SMO - Convergence Rate - Changing Separability

In the linear case, it appears as though higher variance results in a slightly harder optimisation problem for the SMO algorithm to solve. However, the affect appears to be marginal, and the result is not clear from examination of the kernel problem. We note however, that in the kernel problem the convergence rate is extremely noisy. This could be due to the large number of support vectors resulting from the highly overlapping train sets and the kernel decision boundary. I now produce the same analysis for the SGD algorithm.

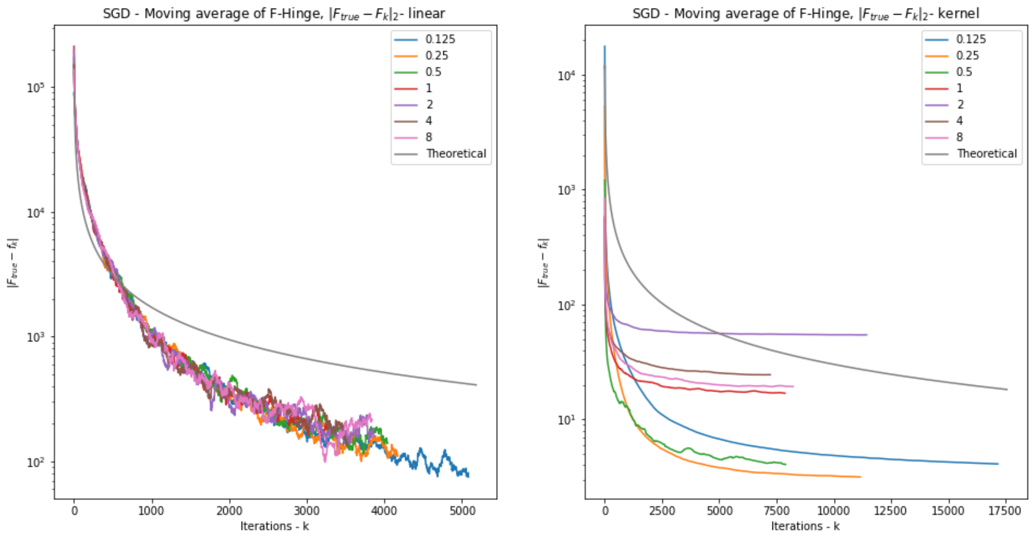


Figure 15: SGD - Hinge Objective Residual - Changing Separability

The linear problem appears unaffected by the increasing variance. The residuals decrease in line with the theoretical bound, at roughly the same rate. However, the results are interesting for the kernel setting. For variance levels higher and equal to 1, the algorithm appears to have converged to a different solution to the one found by CVXPY. Increasing the number of iterations does not decrease the objective function further, and hence the algorithm has missed a more 'optimum' solution found by the alternative solver.

The optimality of a solution is defined by the value of the objective function evaluated at the solution. It is known however that SGD can have a regularizing affect when applied to learning problems [10]. A regularized solution by definition does not minimise the objective function - its intention is to avoid overfitting.

As the variance of ones dataset increases and the separability of the datasets decreases, there is a higher potential to over fit a kernel classifier. It is possible that the results observed above is evidence of SGD regularizing effects.

## References

[1] S. Cumani and P. Laface. Analysis of large-scale svm training algorithms for language and speaker recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(5):1585–1596, 2012.

[2] John Shawe-Taylor and Shiliang Sun. A review of optimization methodologies in support vector machines. *Neurocomputing*, 74(17):3609 – 3618, 2011.

[3] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, April 1998.

[4] Andrew Ng. Cs229 lecture notes , support vector machines, part v.

[5] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[6] Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. A parallel software for training large-scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 07 2006.

[7] RE Fan, PH Chen, and Chih-Jen Lin. Working set selection using second order information for training svm. *Journal of Machine Learning Research*, 6:1889–1918, 01 2005.

[8] Andrew Ng. Cs229 lecture notes , the simplified smo algorithm.

[9] Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. pages 276–285. IEEE, 1997.

[10] Bangti Jin and Xiliang Lu. On the regularizing property of stochastic gradient descent. *Inverse Problems*, 35(1):015004, nov 2018.

[11] J. López and J. R. Dorronsoro. The convergence rate of linearly separable smo. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2013.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273297, September 1995.