A Survey of Sparse Linear Models

Abhishek Jaiantilal

Department of Computer Science

University of Colorado - Boulder, CO, USA.

abhishek.jaiantilal@colorado.edu

Abstract

We live in a data-intensive world and every day a lot of new data is being generated in domains like the internet, genetics, and multimedia, among others. This paper explores recent advances in sparse linear models, mainly in terms of algorithmic development and to a lesser extent exploring ways to apply them to large datasets. We discuss mathematical motivation behind these algorithms and also show results on contemporary algorithms and computer architectures.

1 Introduction

Researchers are looking into, both computational and algorithmically, ways to tackle the onslaught of enormous data. A peculiar similarity, within the datasets generated in internet, genetics, and multimedia, among others, is that the examples are high-dimensional in nature. A study, by Kunder [13], showed that the amount of stored data in the World Wide Web (WWW) is increasing at a tremendous rate. And we can expect increasing amount of data within other fields too.

Let us discuss more about the data encountered on the Internet. According to Google [5], their crawler encountered 18 million unique keywords for the English language (including numeric and alphanumeric keywords) while crawling the web. In a document-keyword space, each document example then is a 18 million dimensional vector. Intuitively we expect to encounter more data with passing time and thus storing and evaluating becomes a challenge. On the positive side, increasing computing power and advances in mathematics that can help us now tackle the problem of handling such enormous amount of data. In this paper our primary goal will be to present algorithms that can scale to large datasets. To a lesser extent we will also look into how increasing amount of computational power can help in gaining a time speedup for these algorithms.

As machine learning researchers, we are interested in understanding the data. We want to answers questions about the data like: What is so special for this example? Can a particular dimension help us in distinguishing between two examples? and so on. Consider a web document example: it is hard to guess the document type if the only clues mentioned are keywords like 'a', 'the' and so on. On the other hand, a document having a keyword like 'microsoft' is a dead giveaway clue that it might be about the Microsoft company. In short, certain dimensions (or features) might help us distinguish among classes. The goal of our paper is to discuss algorithms that exploit the importance of features to achieve higher accuracy.

Our discussion will be insufficient without understanding some basic optimization techniques. It is important to discuss these, as the advances in optimization have been the precursor to methods that can help us solve large sized problems, especially those algorithms that exploit dimension (or feature) importance. Next, we discuss a simple mathematical framework that will help us better to understand the concept of ‘understanding the data’.

1.1 Mathematical Interpretation

In this subsection we will discuss basic mathematics that will help us establish a concrete mathematical formulation for the earlier, vaguely, discussed concepts.

1.1.1 Linear Models

In this paper we will limit only to supervised machine learning methods. Additionally we assume that the training and test data are assigned with either class labels or target values (and with no missing data). We then use a learning algorithm to make a model which generates a label or value, given an example. Such supervised learning of labels or values is well known within the statistics community as classification or regression respectively. Various metrics, can be later used, to check the accuracy of the generated model.

We will first start with a discussion about a simple linear model. Over here we assume that the data consist of N examples and each example is a D dimensional vector. A mathematical representation for a linear system of equations is shown in (1), where y is a Nx1 vector, β is a Dx1 vector, X is a NxD matrix and $f()$ is a function. There is a row to row correspondence between y and X for each example, which we represent here as $\{y_i, x_{:,i}\}$. We use a column major indexing for X. Elementary algebra instructs that multiple solutions might exist if X is under determined. Later, we will discuss some recent work by Donoho [14], in which he shows that under certain circumstances we can get an unique solution for certain types of under determined linear system of equations.

$$\text{General Models :} \qquad y = f(X\beta) \tag{1}$$

$$\text{General Linear (Regression) Model :} \qquad y = X\beta \tag{2}$$

$$\text{General Linear (Classification) Model :} \quad y_i = \begin{cases} 1 & \text{if} \quad x_{:,i}\beta > 1 \\ -1 & \text{if} \quad x_{:,i}\beta \le 1 \end{cases} \tag{3}$$

In a machine learning context, y is what we referred earlier as labels or target values, whereas X represents (observed) data for all examples. The goal here is to find out a D-dimensional β vector which fulfills (2) or (3). The formulation in (2) is called regression and the goal is to fit the model to a set of target values. The formulation in (3) is called (binary) classification and the goal is to fit the model to output a set of labels corresponding here to the set {-1,+1}.

Note: We will like to inform the reader that dimensions will be interchageably called features or variables.

1.1.2 Sparse Linear Models

As we explained earlier that we are interested in 'understanding the data'. Intuitively, for the earlier document example about Microsoft, finding out the document type will be much easier if we know the right keywords. Thus for (2) or (3), we are interested in obtaining a sparse β vector. This means that many of the values within β are explicitly set to zero and there are just a few nonzero values. This also has bearings from a Occam's razor perspective as we are choosing the model that explains using the least parameters. But, we just made our original problem harder as now we are concerned with both getting a good accuracy and a sparse β vector.

The relevance of such sparse linear models is evident in most datasets and even more relevant when we have thousands of dimensions. Perhaps, most of these dimensions are noisy or redundant. Assuming that we have algorithms that allow us to get such sparse solutions, we then would have a powerful mechanism to eliminate such redundant, noisy and irrelevant features from our model. Intuitively, this will allow the model to achieve higher accuracy and be less susceptible to noise.

Now that we are on a firm mathematical footing, it is ripe time to motivate the readers via some real world examples that can be helped by sparse linear models.

1.2 Motivation via Applications/Datasets

In this subsection, we will discuss various high dimensional datasets. These datasets come from varied fields including genetics, statistics, information theory, datamining, among others. Most of these datasets have few examples, except for web data, and are high-dimensional in nature.

1.2.1 Gene Data

Gene data is usually obtained via microarray technique and there is a high cost associated to obtain new examples. Typically, gene datasets will have tens of examples and thousands of dimensions. A prominent gene dataset based paper, used in many machine learning papers, is by Golub *et al.* [26]. In it, they discuss methods that distinguish between ALL and AML types of leukemia from Microarray data. Also discussed are methods to find sets of features that might help distinguish between the two types of leukemia. The training set consists of 38 examples and test set of 34 samples, each a 6817 dimensional vector. They use correlation between each feature and the class label to come up with a feature set (by choosing the most correlated features). This process can lead to selecting features that are misleading, as one does not take into account the interactions among features[1].

In many papers, including *'Regularization and variable selection via the elastic net'* by Zou *et al.* [50], microarray data is used as a benchmark to show that choosing relevant features can help boost the accuracy of the classifier. Typically, gene data has thousands of features and around hundred's of samples, making it a good dataset for 'implicit'[2] feature selection based algorithms. Another recent research area is about investigating feature groups (features related to one another).

1.2.2 Miscellaneous Datasets

Real world robotic data, using vision or other sensors, has numerous types of features and whose importance is environmentally, dependent is a good fit for feature selection. Other data include web data like Reuter's RCV1 dataset (in a document-keyword format), which is used extensively in information retrieval, has about 50K features and 800K examples for 100+ classes. Each example in the RCV1 dataset is generated from a newswire article with the feature value being the number of times a word is used in that article. Other examples include video and image data used for compression. It is sufficient to say that, these days, there is no dearth of sources producing such high dimensional datasets.

1.3 Additional Mathematical Concepts

Before delving into machine learning topics, it will be wise to discuss a bit about the history behind sparse linear models.

In statistics, the earliest work on sparse linear models was by Tibshirani [39]. He shows that a lasso penalty (1-norm penalty) in a regression setting[3] can get a sparse β. Though at the time there were no optimization techniques powerful enough to use his theoretical result for a practical use on large datasets.

The earliest work on sparse linear models is from the wavelet community in the early 90's. We will not discuss the internal working of wavelets and fourier transforms in this paper, but restrict ourselves to certain interesting properties. Also the following discussion applies to some types of image and data compression. In brief: wavelets help us to represent signals or images using the magnitudes of a few basis functions and

[1]lack of feature-interaction is a hindrance for applying this method to high-dimensional datasets.
[2]meaning that feature selection is built into the algorithm, and not as a preprocessing step.
[3]we will discuss this in a later section.

the same happens in image and data compression. Consider that in (1) the basis functions are represented as columns of matrix X. If we solve the linear system we might have many of the values in the β vector to be near zero. If we set many of those near-zero values to 0, we will obtain a degraded reconstruction and a lossy-compression[3].

Wavelets and Fourier transforms can be represented in an orthonormal basis space. This special property allows for the application of various computationally easy methods to solve (1). Next, we will discuss in brief about optimization and tie it with the linear systems of equations that we discussed earlier.

1.3.1 Standard formulation in Optimization

The material in this subsection is modeled around the convex optimization[4] book by Boyd *et al.* [4]. The standard formulation to represent an optimization problem is as follows:

$$\min \; f_0(x) \; \text{ subject to } f_i(x) \leq 0, i = 1, \ldots, m \;\text{ and }\; h_i(x) = 0, i = 1, \ldots, l \tag{4}$$

$$\text{Let } p* = \inf\{f_0(x) | f_i(x) \leq 0, i = 1, \ldots, m, \text{ and } \; h_i(x) = 0, i = 1, \ldots, l\} \tag{5}$$

$$\text{Feasible Solution: } x_{feas} = \{x | f_i(x) \leq 0, i = 1, \ldots, m, \text{ and } \; h_i(x) = 0, i = 1, \ldots, l\} \tag{6}$$

$$\text{Optimal Solution: } x_{opt} = \{x | f_i(x) \leq 0, i = 1, \ldots, m, \text{ and } \; h_i(x) = 0, i = 1, \ldots, l, \; f_0(x) = p*\} \tag{7}$$

In (4), $f_0(x)$ is the function that needs to be minimized. Additionally we have $f_i(x)$ which are some inequality constraint functions and $h_i(x)$ which are some equality constraint functions. These constraints define a feasibility set (x_{feas}) inside which the minimizer of $f_0(x)$ needs to be found out. Over here we define $p*$ as the set of minimizer(s) of $f_0(x)$.

Many a times the optimization problem can be transformed into an equivalent alternative form which we describe in the next subsection. Other popular ways, which the reader can refer in [34] and [4], to generate an equivalent alternative form, and not discussed in our paper, is by mapping the variables to a different space or by transforming the objective and constraint function interchangeably.

1.3.2 Transforming Optimization Problems and Conditions for a Minima

Lagrange Multipliers: Generally optimization problems are formulated as minimizing (or maximizing[5]) a function under some constraints as:

$$\text{minimize } f_0(x) \quad \text{subject to } f_i(x) \leq 0, i = 1, \ldots, m \;\text{ and }\; h_i(x) = 0, i = 1, \ldots, l$$

[4]opposite formulation from [34] where inequalities are shown as ≥ 0.
[5]not used in our context, and thus not shown.

where variable $x \in R^n$, f_i are the inequality constraint and h_i are the equality constraints. The Lagrangian function is then defined on $L : R^n \text{x} R^m \text{x} R^p \rightarrow R$ with

$$L(x, \lambda, \upsilon) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{l} \upsilon_i h_i(x) \tag{8}$$

In (8), vectors λ and υ are called the Lagrange Multipliers. We have defined the original function (called the *primal*) into a *dual* function. For many convex functions, the primal and the dual functions are equivalent and optimizing for either of them will lead to the same global minima.

$$\text{Constrained:} \qquad \min f_0(x) \quad \text{subject to } f_i(x) \leq 0, i = 1, \ldots, m \quad \text{and} \quad h_i(x) = 0, i = 1, \ldots, l \tag{9}$$

$$\text{Unconstrained:} \qquad \min f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{l} \upsilon_i h_i(x) \tag{10}$$

In (9) and (10), we have two equivalent formulation, due to the nature of the primal and dual, which we will call as **constrained** and **unconstrained** optimization respectively. Next, we will discuss the conditions that are needed for a point to be considered a minima.

$$\begin{aligned}
\text{KKT Conditions:} \qquad f_i(\tilde{x}) &\leq 0, i = 1, ..., m && (11)\\
h_i(\tilde{x}) &= 0, i = 1, ..., l && (12)\\
\tilde{\lambda}_i &\geq 0, i = 1, ..., m && (13)\\
\tilde{\lambda}_i f_i(\tilde{x}) &= 0, i = 1, ..., m && (14)\\
\nabla f_0(\tilde{x}) + \sum_{i=1}^{m} \tilde{\lambda}_i \nabla f_i(\tilde{x}) + \sum_{i=1}^{l} \tilde{\upsilon}_i \nabla h_i(\tilde{x}) &= 0 && (15)
\end{aligned}$$

Karush-Kuhn-Tucker (KKT) Conditions: KKT Conditions, shown above, are a necessary and sufficient set of conditions for an optimization problem to have x̃ and ($\tilde{\lambda}$,$\tilde{\upsilon}$) as primal and dual optimal points respectively (when f_0 is convex, f_i and h_i are affine and x̃, $\tilde{\lambda}$, $\tilde{\upsilon}$ are any points for the optimization formulation (10) discussed earlier). Feasible conditions, in case of KKT, are required to prove that the point lies in the domain of possible optimal points. The first two conditions imply that x̃ is primal feasible and as $\tilde{\lambda}_i \geq 0$, Langragian $L(x, \lambda, \upsilon)$ is convex in x. The last condition is directly implied from the fact that at the optimal minimum (x̃,$\tilde{\lambda}$,$\tilde{\upsilon}$) of (15), the gradient should disappear ($\nabla L = 0$) [a necessary condition for optimality].

The reader should refer to [4] or [34] for more information about the KKT conditions. The gist that should be taken here is that the KKT conditions are necessary and sufficient to prove the optimality of a

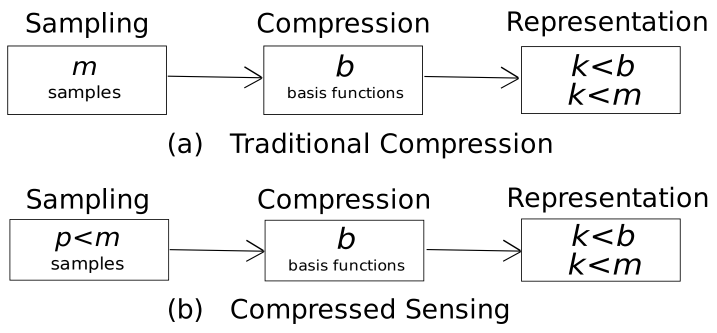


Figure 1: Compression Techniques

solution.

Next, we will discuss a field that is came into existence due to sparse datasets. There are many excellent theoretical implications within the field that justifies briefly discussing about it.

1.4 Additional Datasets in Compressed Sensing

Compressed Sensing (CS) is a relatively new field and is related to data compression. Consider a typical compression mechanism, as shown in Figure 1a, in which we sample some signal data (m samples) and represent it using some few fixed basis functions (k), thus achieving compression. The chosen k basis functions do not require all the signal data but only certain part of the original signal data. Thus to get the required k basis functions we do not need to sample the whole data but only certain part of it, which drives the core idea behind CS[6]. Intuitively one might think that the basis functions are adaptive – meaning that the chosen basis functions are dependent on the original data. A study by Donoho [14] shows that this is not true and the required p samples are non-adaptive in nature (not dependent on the data). We will now discuss mathematically how sparse datasets come into picture and how the ideas driving CS use the same optimization methods that a machine learning researcher is also interested in.

The majority of the work in CS consists of working in orthonormal basis space. This includes wavelets and fouriers, among others, and generally is referred to as dictionaries in literature. Summarizing Donoho from [14]: We first collect some basis functions waveforms as an $\underline{n\text{x}d \text{ matrix '}\Phi\text{'}}$ and decompose a signal m as follows:

$$m = \Phi\alpha \quad \text{(OR)} \tag{16}$$

$$m = \Phi\alpha + e_d, \text{ where } e_d \text{ is residual \& } \alpha \text{ are weights to be found.} \tag{17}$$

[6]'what if instead of sampling everything, we sample only that part which is required.'

where m is a nx1 vector, Φ is a nxd matrix and α is a dx1 vector.

Donoho shows that for certain type of orthonormal dictionaries, if m_K denotes the vector m with everything except the K largest coefficients set to 0, then $||m - m_K||_p$ depends only on the L^p norm[7], with $K \asymp \epsilon^{(p-2)/2p}$, ϵ is error. This means that the error in reconstruction will depend on the norm of the metric used. Further, he shows that the same approximation that can be achieved with K largest coefficients can also be achieved with $p \approx Klog(n)$ pieces of information. For example, classes of images with m pixels will need around $p = O(Klog(m))$ non-adaptive measurements to be faithfully reconstructed, rather than usual m pixel samples. This is the core idea behind Compressed Sensing and what we explain for in Figure 1b. For more information about the theoretical underpinnings behind Compressed Sensing, the reader is encouraged to refer to Donoho *et al.* [18, 15, 17, 16]. We can not do justice of explaining the underlying theory in a paragraph or two but the gist, via the example of image compression, is that – instead of sampling the whole image we need to sample some fixed pixels in the image and still get the same kind of compression.

The representation of the signal in a basis space is through a linear model and the reason why sparsity, in the α vector, is desired can be summed up by the goal required in compression (compressed sensing included): find out the K largest coefficients that can give a good enough representation.

Ideally, this can be formulated as the following L^0 minimization problem:

$$\min \quad ||\alpha||_0 \text{ subject to } m = \Phi\alpha \tag{18}$$

which amounts to finding exactly K non-zero coefficients in the α vector, and is a NP hard (Nondeterministic Polynomial-time hard) problem due to the L^0 penalty[8].

A lot of literature, by Donoho *et al.* [18, 15, 17, 16], is devoted on proving that many compression techniques possess certain properties that can help relax these NP-hard conditions. They show that the orthonormal basis space associated with the various transforms (wavelet, fourier, etc.) allows the NP-hard L^0 minimization problem and the Linear programming (LP) based L^1 minimization problem, shown in (19), to have the same solution. Linear programming is known to be solvable in polynomial time, thus allowing for large scale problems (millions of variables) to be solved in a reasonable time frame. We usually cannot exactly fit $m = \phi\alpha$ so we resort to the least squares measure as in (20) which is quadratic programming.

$$\min \quad ||\alpha||_1 \text{ subject to } m = \Phi\alpha \tag{19}$$

$$\text{Transformed to} \qquad \min ||m - \phi\alpha||_2^2 \text{ subject to } ||\alpha||_1 \leq t \tag{20}$$

[7] L^p norm is mathematically defined as $||a||_p = (a^{1/p})^p$.
[8] L^0: $||\alpha||_0$ = count number of non-zeros in α.

Next, we will be discussing about polytopes in general and convex polytopes in particular. We discuss polytopes because they have direct implications to the optimization problems being solved in (18) and (19). Excellent discussions on polytopes can be found in Zeigler [47], Brondsted *et al.* [6] and Grunbaum *et al.* [27]. Polytopes, in layman's term, are polygons in any dimension. Mathematically a polytope, from Ziegler [47], is defined as:

$$conv(\Phi) = \quad \{\lambda_1\phi_1 + \ldots + \lambda_d\phi_d : \{\phi_1, \ldots, \phi_d\} \subseteq \Phi, \lambda_i \geq 0, \sum_{i=1}^{d} \lambda_i = 1\} \tag{21}$$
$$\text{Where, } \phi_i \text{ is the } i^{th} \text{ column of } \Phi$$

An simple interpretation of the polytope can be thought as the convex hull (defined by the function *conv*()) that is generated by (21). If one considers a rubber string that wraps around some points in a 2-D space, then the convex hull in basically the area (maximal) covered inside the taut rubber string. Now imagine the same for higher dimensions: the convex hull is basically an outer shell without any dents (and thus convex).

Let us now discuss how optimization and polytopes are related. Each constraint in an optimization problem can be thought of as a hyperplane dividing the solution space into two sides, one side following the constraint and other violating it. Intersection of such hyperplanes will generate a polytope and a feasible solution lies within or on the surface this polytope. One can imagine that the intersection of such hyperplanes will create vertices, edges (line between two vertices) and faces (surface created by 3 or more vertices). Now as we saw earlier that depending on whether we are working on a constrained formulation or the unconstrained formulation the constraints are different. That is the primal and dual optimization formulation will generate polytopes of different shapes.

We would like to mention a few additional properties of convex polytopes that will help connect the dots between the LP (Linear Programming) = NP equivalence. A k-neighborliness polytope is a polytope for which every k vertices span a face (all k vertices lie on the same hyperplane surface). Sporyshev *et al.* [42] show that there is a relation between polytope neighborliness and the sparsity of the solution and derived phase transitions, which plot sparsity of solution vs. neighborliness of the polytope generated by the optimization formulation.

Donoho *et al.* [18] gave hard conditions (empirically and theoretically) for many orthonormal transforms that occur in Compressed Sensing (CS). They show that the k-neighborliness is directly related to the L^0 minimization problem, and due to the fact that many of the low k-neighborly polytopes are well studied, the problem be easily solved with LP (thus getting the LP=NP equivalence for certain conditions), and not as an NP hard combinatorial problem. In layman mathematical terms it can be summarized as follows: if α is known to be sparse in nature, than the polytope constructed with $m = \Phi\alpha$ instead of being a polytope in R^d dimension is probably a simpler polytope in a much lesser dimension $R^l, l << d$ and due to the fact that such simpler polytopes are well studied, a LP based approach is equivalent to a combinatorial based approach (NP-hard).

A tangential concept, in Compressed Sensing (CS), not covered in this paper that the reader is directed to explore more is the recent theory by Candes *et al.* [7] about random projection of matrices and the bounds placed on it.

2 Theoretical Concepts in Regularization

The ideas behind regularization have their roots in the Occam's razor principle. An interpretation of the Occam's razor is to prefer simpler models over complex models. Regularization is an effort in that direction. Regularized models have a generic formulation as:

$$\min \quad L(X, y, \beta) + \lambda J(\beta) \tag{22}$$

In (22), $L(X, y, \beta)$ is known as a **loss function** that needs to be minimized over the parameter space of β. X and y are known and are the depicted for the data matrix and labels (or target values) respectively. $J(\beta)$ is known as a **penalty function** over the parameter β. The idea is to penalize the model if the parameter β is unstable. For example, if the parameter β is over-fitting the data or has high magnitude value ,we will like to penalize the model more. λ parameter acts as a tradeoff parameter between the importance of the loss function compared to the penalty function. The reader is encouraged to refer to Hastie *et al.* [29] which has a good discussion on the relative tradeoff in regularization for many machine learning techniques.

Let us continue onwards by motivating the need of regularized models for the linear system of equations in (1). We will assume that y is a Nx1 vector, β is a Dx1 vector, X is a NxD matrix.

$$\text{Equation (1)} \qquad y = X\beta$$

$$\Rightarrow \boxed{\beta = X^{-1}y}, \quad \text{if X is square and non-singular} \tag{23}$$

$$\text{Least Squares} \qquad \min ||y - X\beta||_2^2 \tag{24}$$

$$\frac{d(||y - X\beta||_2^2)}{d\beta} = 0 \Rightarrow -2X^Ty + 2\beta X^TX = 0 \Rightarrow \boxed{\beta = \frac{X^Ty}{X^TX}} \tag{25}$$

Equation (23) is only defined when X is square and non-singular. Thus we need an alternate method for the non-square matrix case. A popular metric is the least squares measure, which due to specific properties[9] is preferred by statisticians, and is shown in (24). If the metric is differentiated with respect to β and equated to 0 (the gradient is 0 at minimum), we obtain a closed form solution as shown in (25). This solution is defined only if X^TX is non-singular and is unstable if X^TX is singular. The latter is known as an ill-posed problem. This happens if there is multicollinearity, meaning that each variable β_i has high correlation with other variables. Alternatively, it can also be interpreted as: the columns of X are linear combinations of other columns in X. Do note that the formulation is termed, in literature, as **ordinary least squares**.

Multiple researchers, Hoerl [30], Moore [32], Tikhonov [41] and Penrose [36], at various times, came up with the idea of what we will refer to as ridge regression in order to tackle ill-posed problems. Ridge regression, shown in (26), is a popular example of regularization for least squares and is formulated as an additional L^2 norm on the β estimates. We mean that we use the penalty function $J(\beta) = ||\beta||_2^2 = \sum_i \beta_i^2$. Now, setting the gradient to 0 as done earlier in (25), we get a closed form solution for β as shown in (27).

$$\text{Ridge Regression:} \min ||Y - X\beta||_2^2 + \lambda||\beta||_2^2 \tag{26}$$

$$\frac{d(||y - X\beta||_2^2 + \lambda||\beta||_2^2)}{d\beta} = 0 \Rightarrow -2X^Ty + 2\beta X^TX + 2\lambda\beta = 0 \Rightarrow \boxed{\beta = \frac{X^Ty}{X^TX + \lambda I}} \tag{27}$$

$$\text{pseudoinverse :} \quad \boxed{\beta = X^+y}, \quad \text{where} \quad X^+ = (X^TX + \lambda I)^{-1}X, \quad \text{compare to (23)} \tag{28}$$

Just a side-note: Moore-Penrose pseudoinverse [1], which is denoted by $X^+ = (X^TX + \delta I)^{-1}X$, was developed to invert a singular matrix as shown in (28). Do compare it to the solution of (23) - as can be seen that the pseudoinverse helps in inverting a non-square matrix which might be singular. Please do note that a closed form solution does not exist when the penalty term uses the L^1 norm on the β estimates (that is $J(\beta) = ||\beta||_1^1 = \sum_i |\beta_i|$).

We will next see the difference brought due to regularization for least squares compared to the ordinary least squares solution. In Figure 2, we plot values of β estimates for $0 \leq \lambda \leq 1$ when using ridge regression on the Pima Indians Diabetes dataset (768 examples, 8 dimensions) from the UCI repository [3]. In the case when there is no regularization (λ=0), the magnitude of $\beta's$ is varied and larger, compared to when there is some regularization involved ($\lambda >$0). As can be seen that on increasing the regularization, through λ, we cause a proportional shrinkage (decrease) of the β estimates.

[9]best linear unbiased estimator, discussed more in Wiesberg [44].

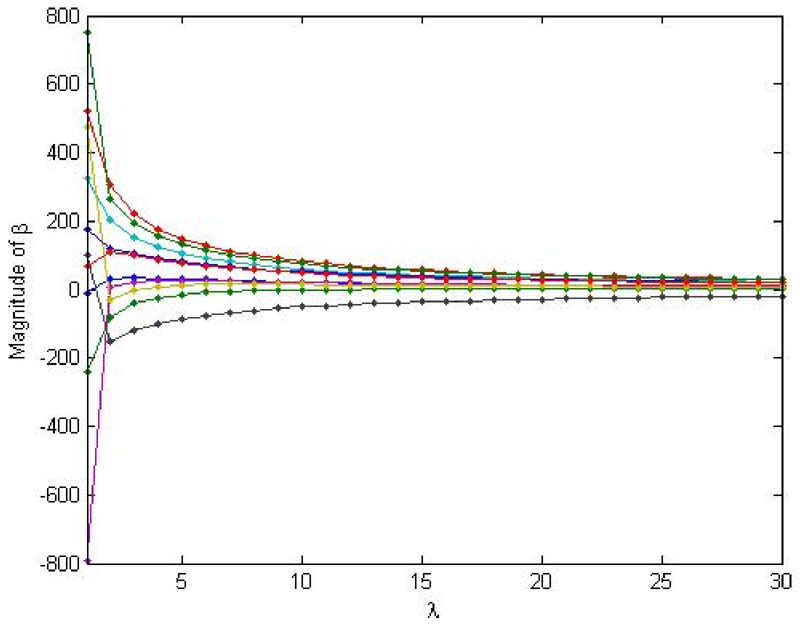


Figure 2: *Ridge Path for* $0 \leq \lambda \leq 30$ *for the Pima Indians Diabetes Dataset*

From now onwards, we will refer to the L^p norm on the β term as the **L^p penalty** and algorithms using such penalty terms as **Penalized algorithms**.

2.1 Feature Selection (FS), Regression and Regularization:

We can now formally define the goal of algorithms producing sparse models in machine learning as Feature Selection(FS): find the model that can explain the problem, by being frugal on the number of variables and at the same time being good at future predictions. Note that this is now a two-pronged solution: choosing a model that is sparse in features (and thereby eliminating potentially noisy features and/or redundant features) and at the same time performing acceptably well on future data. Combinatorially, we have to choose the best feature subset from a possible total of 2^D feature subsets, where the number of variables or features=D. If we are not concerned with the computational complexity involved, FS can help solve the two problems of achieving accuracy and 'understanding the data' simultaneously.

To achieve the goal of finding the best subset, we will additionally need metrics to evaluate and choose the best subset. Please note that we will not discuss the various metrics like f-statistics or t-statistics in this paper and for more information the reader is encouraged to refer to the text by Weisberg [44]. We will though mention the fact that these metrics are used to evaluate if a particular feature, not in the model, would actually help the model or not if included. Do note that we will be talking about features as variables interchangeably.

Subset Selection in Regression : Stepwise Methods are a popular family of methods used for selecting feature subsets and discussed more in Weisberg [44]. They can be classified into the following 3 types based on how they add/delete features from the model.

- **Forward Selection(FwS):** Begin with a model that has a single independent variable having the highest correlation with y. At each step add that 'other' (not-already in the model) variable that helps the model most (has highest f or t-statistics of all the variables not in model). Keep on doing this till the model has a specified number of variables or appropriate f or t-statistic.
- **Backward Elimination (BE):** Begin with the full model and start by eliminating one variable at a time, till the model has a specified number of variables or appropriate f or t-statistic.
- **Stepwise(SW):** SW starts like FwS, but with the difference that variables can be dropped. Also, if a variable outside the model increases the f or t-statistic compared to a variable that is inside the model, then the variables can be swapped.

SW methods work best when independent (uncorrelated or orthogonal) variables are involved **but** the goal of FS in Regression makes most sense when there are many correlated features with the target values/labels y. Also, ordering (how features enter the model) does not necessarily imply the importance of the feature: a pair of features that are added later on might work better than the one entered earlier (and might also result in the features being swapped out of the final model). And thus, the best subset with n features might be totally different compared to the best subset with $n+1$ features for the same dataset, implying that feature interaction is an important factor to look for. Such reasons make ad-hoc methods for FS non-viable for large datasets. Later, we will see that current FS methods behave a lot like the SW methods and thus suffer from related problems.

2.1.1 Orthogonal Design Case ($X^T X = I$)

Orthogonal Design case ($X^T X = I$) is well studied, and discussed in Chen *et al.* [11] and Tibshirani [39], as many datasets including that based on Fourier/Wavelet transforms have orthogonal properties. The goal of studying the orthogonal design case in this subsection is to exhibit what really happens in case of the penalized regression models compared to the ordinary least squares (**OLS**) regression. Do note that when we use OLS, a uniform prior is assumed for the β estimates. Though, for L^2 (**Ridge**) penalty and the L^1

(**Lasso**[10]) penalty we assume a Gaussian and a Laplacian prior respectively for the β estimates. This will result in variations in the generated models for the ridge and the lasso penalty.

In (29), we expanded the formulation for ridge regression and then substituted the formulation from (30) to obtain a closed form solution as shown in (31). Similarly we obtained a close form solution for lasso regression as shown in (32).

$$\begin{aligned}
\text{Ridge:}||y - X\beta||_2^2 + \lambda||\beta||_2^2 = &\quad y^Ty - 2y^TX\beta + X^T\beta^T\beta X + \lambda\beta^T\beta \\
= &\quad y^Ty - 2y^TX\beta + \beta^T\beta + \lambda\beta^T\beta \quad (\because X^TX = I) \qquad (29) \\
\text{If we fit for OLS,} \quad y = &\quad X\beta_{ols} \text{ where } \beta_{ols} = \text{OLS estimate} \\
\Rightarrow &\quad X^Ty = X^TX\beta_{ols} = \beta_{ols} \qquad (30) \\
\text{Ridge: } ||y - X\beta||_2^2 + \lambda||\beta||_2^2 = &\quad y^Ty - 2\beta_{ols}^T\beta + \beta^T\beta + \lambda\beta^T\beta \quad \text{Using (30) in (29)} \\
\text{At minimum, } (\nabla = 0) &\quad -2\beta_{ols} + 2\beta + 2\lambda\beta \Rightarrow \boxed{\text{For Ridge, } \beta = \frac{\beta_{ols}}{(1+\lambda)}} \qquad (31) \\
\text{Lasso:}||y - X\beta||_2^2 + \lambda||\beta||_1 = &\quad y^Ty - 2y^TX\beta + X^T\beta^T\beta X + \lambda|\beta| \\
= &\quad y^Ty - 2\beta_{ols}^T\beta + \beta^T\beta + \lambda|\beta| \quad (\because X^TX = I) \\
\text{At minimum, } (\nabla = 0) &\quad -2\beta_{ols} + 2\beta + \lambda|1| \Rightarrow \boxed{\text{For Lasso, } \beta = sign(\beta_{ols})(\beta_{ols} - \lambda_1)_+}, \quad \lambda_1 = \lambda/2 \qquad (32)
\end{aligned}$$

We plot the above derived closed form solutions in Figure 3. We can also obtain such close form solutions for different penalties to exhibit their relation to the OLS solution and thus there are multiple plots for different penalties in that figure. For all plots, the magnitude of β_{ols} (OLS estimates) are represented on both axes. The blue line at 45° represents the magnitude of β_{ols}, whereas the black line represent the magnitude for the respective penalty.

As shown in (31) and Figure 3a, ridge regression shrinks the OLS estimates by a factor of $(1 + \lambda)$, whereas in lasso regression (from (32) and Figure 3b), the estimates are nonzero only if the variable magnitude $> \lambda$. We refer to the former property as **shrinkage** and the latter property as **thresholding**. Thus, such estimate shrinkage/thresholding will tend to cause completely different profiles for the β estimates when different penalties are involved. Such thresholding is evident in plots 3c, 3b, 3e and 3f which are plotted for L^0, $L^{0.2}$, $L^{0.5}$ and weighted lasso penalties[11] respectively. We will like to point out to the reader that, from the plots, its evident that thresholding will happen if $L^p, p \leq 1$ and shrinkage if $L^p, p > 1$, where L^p represents the

[10]Lasso Penalty was coined by Tibshirani [39] due to it's property of *lassoing* variables into the model.
[11]we will cover this in a later section.

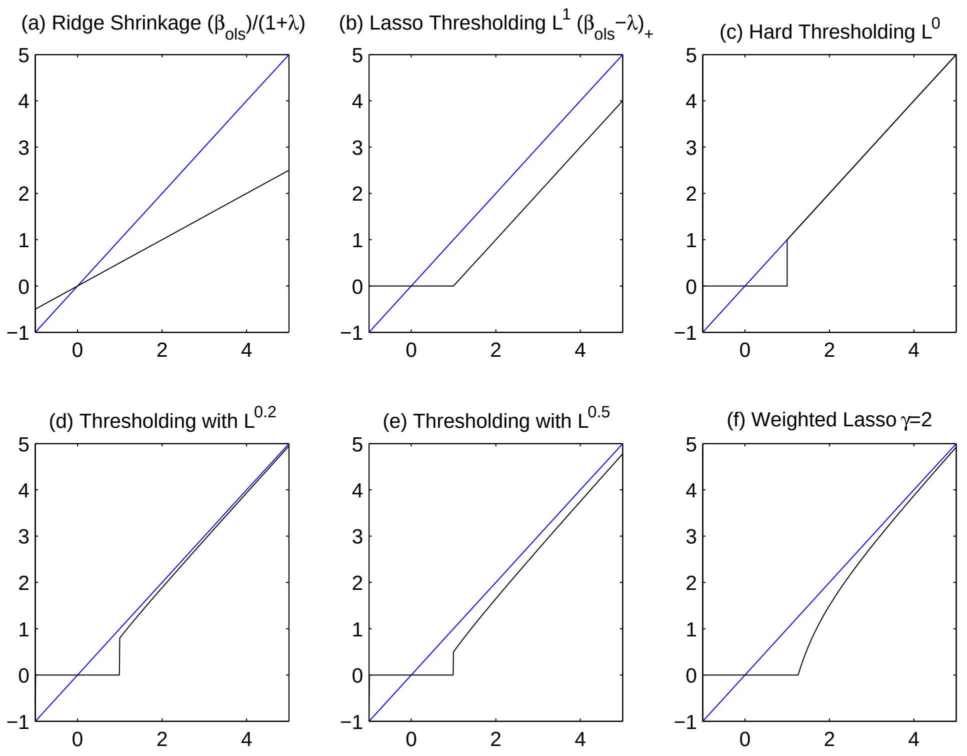


Figure 3: *Shrinkage/Thresholding Plot for various Penalties*

pth-norm penalty. We will later prove that these properties have a solid theoretical underpinning.

We will also like to point out to the reader that L^0, L$^{0.2}$, L$^{0.5}$ and weighted lasso penalties (in Figures 3c, 3b, 3e and 3f respectively) almost trace to the OLS estimates for variables that should have large estimates. This is particularly interesting as both lasso and ridge penalties never quite get to the OLS estimates for variables that should have large estimates. We will later see that this property of tracing to the OLS estimates is an important property if correctness in feature selection is to be achieved. This brings up an important question that will be answered later on: "*is the lasso penalty enough (correct) for sparsity?*"

2.1.2 Vanilla Empirical Proof of Shrinkage and Thresholding

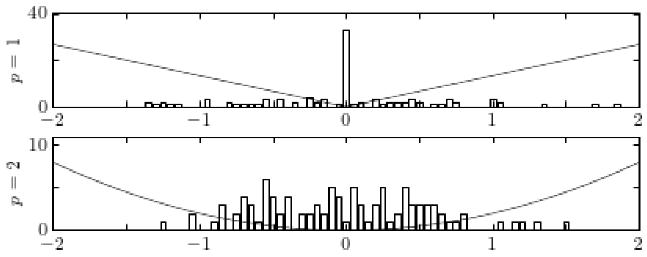
In Figure 4a, we show the histogram of feature magnitudes for the lasso and ridge penalties, denoted by $p = 1$ and $p = 2$ respectively, for a test regression problem[12]. For the lasso penalty, the majority of feature weights are 0. For the ridge penalty, most features are non-zero, but there are more features having higher magnitudes in the lasso penalty compared to the ridge penalty. This is due to what was observed in Figure 3a, where due to shrinkage the estimate magnitudes are shrunk. This results in estimates that should have

[12]copied from Boyd *et al.* [4].

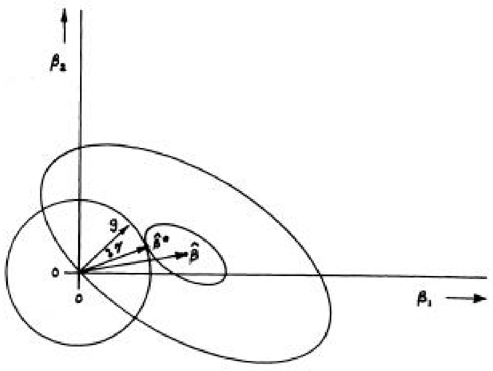
higher magnitudes never quite getting those high magnitudes when ridge penalty is involved. Whereas for the lasso penalty, in Figure 3b, we do have higher magnitudes for estimates which should have higher magnitudes, but do note that there is always some difference between the lasso estimates and the OLS estimates. Intuitively this suggests that lasso penalty will have more estimates having larger magnitudes compared to the ridge penalty. Also there will be very few non-zero estimates when ridge penalty is involved. Properties of shrinkage and thresholding are thus also evident from these empirical results.

2.1.3 L^p ball in Regularization

Continuing onwards on our discussion about why penalties have such varied behavior can also be understood by examining the intersection of the optimization curve (for the loss function) with a L^p ball (for the penalty function). Note that till this point we have emphasized on talking about unconstrained optimization problem for the ridge and lasso regression. In an earlier section, we discussed about the constrained optimization problem and showed that we can transform it, via Lagrange multipliers, to an unconstrained problem, as shown in (9) and (10).



(a) *β Magnitude Histogram*



(b) *L^2 ball in Ridge Regression*

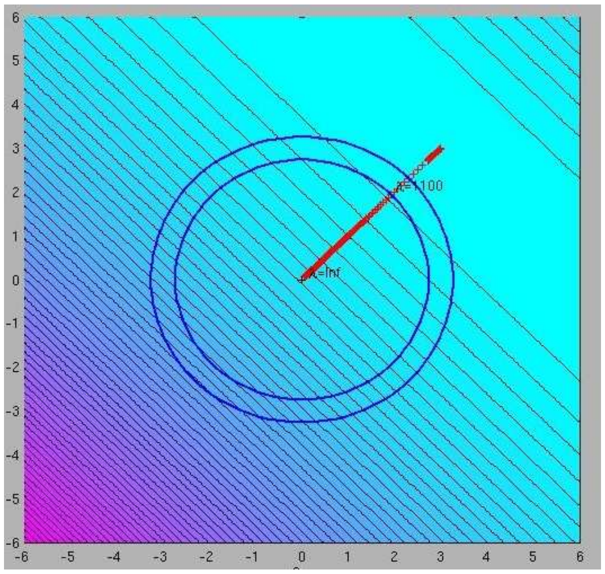
Figure 4: Plots for Magnitude Histogram and L^2 ball

$$\min_{\beta} \sum_{i} L(x_{:,i}, y_i, \beta) + \lambda ||\beta||_p \iff \min_{\beta} \sum_{i} L(x_{:,i}, y_i, \beta), \;\; ||\beta||_p \leq t \tag{33}$$

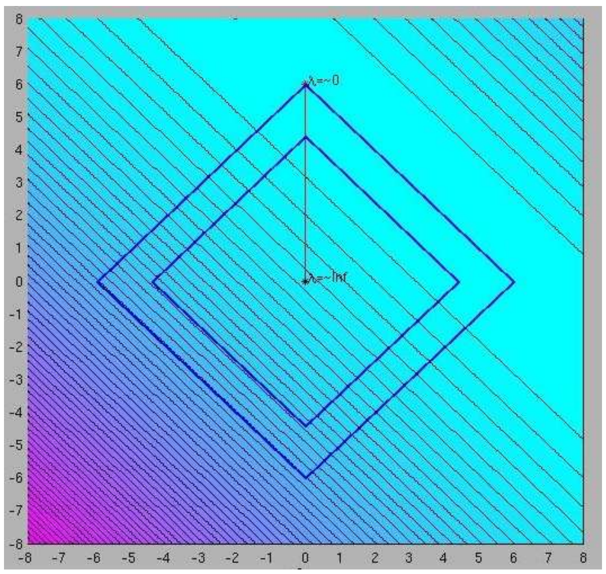
Both formulations in (33), are **equivalent**, as the former is the dual form of the latter[13]. For example, if we let t to be small, the corresponding problem can be formulated with λ having a large magnitude (and then even a small $||\beta||_p$ will accentuate the loss).

[13]primal and dual formulation which we discussed in Section 1.3.2.

In Figure 4b, from Marquardt *et al.* [45], a hypothetical regression problem is shown in which there are two variable β_1 and β_2 (on the two axes), with the point $\hat{\beta}$ at the center of the ellipse being the OLS solution. The smaller ellipse is the locus of points for which the sum of squares is constant. The circle at the origin represents the L^2 ball for the L^2 penalty and is tangent to the small ellipse at $\hat{\beta}^*$. The vector $\hat{\beta}^*$ is the shortest vector (and solution of the ridge regression) that will give the OLS estimates as small as anywhere on the smaller ellipse. As expected, this same scheme can be used for the L^1 ball, which due to the specific geometry of the L^1 ball, will have $\hat{\beta}^*$ be either $(\beta_1, 0)$ or $(0, \beta_2)$ with a higher probability than other values. Another 'intuitive' way to think about this problem is: ***'find the minima of the loss function within/on the L^p ball (of constraint)'***.



(a) *Ridge Regression for 2 predictors (correlation=1)*



(b) *Lasso Regression for 2 predictors (correlation=1)*

Figure 5: Plots for L^1 & L^2 ball with 2 predictors (correlation=1) in a Regression Setting

In Figure 5a and 5b, we plot the least squares regression loss function for two predictors (represented on the axes), with the lighter blue representing lesser function value (with the lightest blue representing the minima). The predictors in this case have a correlation of 1. In Figure 5a, we can see that the L^2 ball touches the optimization curve at only one point, whereas the L^1 ball, in Figure 5b, can touch the optimization curve at multiple points. This means that we can have multiple equivalent solutions for the lasso regression. Also note that there are two L^1 and L^2 balls in both the figures corresponding to different penalties (the smaller ball corresponding to a lesser penalty).

2.1.4 A Bayesian Twist to Penalties

Let us now discuss penalties on the basis of the Bayesian priors that are imparted via them. For the solution of OLS, we assume an unbounded uniform prior distribution on the variables, but for the lasso penalty we imply a Laplacian prior and for the ridge penalty, a Gaussian prior. The Laplacian prior is taller, thinner but wider, thus preferring higher magnitude β_i but with most of them set to 0, whereas the Gaussian prior is thicker but short statured, thus preferring smaller magnitudes but with lesser probability of a zero magnitude β_i. This interpretation is from a maximum likelihood estimation context by Boyd *et al.* [4].

2.1.5 Theoretical underpinnings for Shrinkage and Thresholding

We reasonable showed, earlier in this subsection, through various angles via the orthogonal design case, L^p ball analogy, Bayesian priors and empirical results, that sparsity is existent for the lasso penalty. We will now present the theoretical reason for the existence of sparsity when $\mathrm{L}^p, p \leq 1$ penalties are involved.

Optimization methods for a general class of L^p penalties, in a regression setting, called the bridge regression (term by Frank *et al.* [22]) and shown in (34) are discussed in Fu *et al.* [24].

$$\text{Bridge Regression: For } \gamma \geq 1, t \geq 0 \qquad \min_{\beta} ||y - X\beta||^2 \text{ subject to } \sum |\beta_j|^\gamma \leq t \tag{34}$$

The loss function being minimized in (34) is convex (discussed later), thus allowing for a global minimum. For each of the different values of $\gamma \geq 1$, we will have a different kind of L^p ball but the idea of intersecting the L^p ball with the loss function remains the same. For $\gamma < 1$, the probability of getting a sparser solution is higher than $\gamma = 1$, but the problem is non-convex and thus suffers from multiple local minimas. Let us now discuss analytically why the $\mathrm{L}^p, p \leq 1$ penalized algorithms exhibit sparsity whereas the $\mathrm{L}^p, p > 1$ penalized algorithms do not.

For more information about convex functions, please browse through Section 3.1. A norm p in Banach space is strictly convex if and only if $x \neq y$ and $||x||_p = ||y||_p = 1$, which together imply that $||x+y||_p < 2$ (and just convex if $||x+y||_p = 2$)[14]. This effectively shows that while $\mathrm{L}^p, p > 1$ penalty is strictly convex, L^1 (*lasso*) penalty is convex but not strictly. Let us now present two lemmas (from Zou *et al.* [51]) that show the behavior of the β estimates, for strictly convex and convex norms, in presence of two similar features.

We will use the notation for N examples and D dimensions/features by representing y as a Nx1 vector, β as a Dx1 vector and X as a NxD matrix.

[14]A norm must follow the triangle inequality (distance), if there are 3 points A,B,C, a norm must have distance(A,C)$\leq$ distance(A,B)+distance(B,C).

Lemma 1: *Assume we have data in which $x_i = x_j, \quad i, j \in \{1...D\}$ and x_i, x_j are columns in X. If the penalty is strictly convex (e.g. L^2 (ridge) penalty) then predicted $\beta_i = \beta_j$.*

Proof: If $\beta_i \neq \beta_j$, we build another model with $\beta_i^* = \beta_j^* = \frac{1}{2}(\beta_i + \beta_j)$. Thus, we have two minimizers $(\beta_i, \beta_j) \neq (\beta_i^*, \beta_j^*)$, but with $X\beta^* = X\beta$ which is not possible as the penalty is strictly convex and violates $|\beta^*|_2 \neq |\beta|_2$.

Lemma 2: *As L^1 penalty is not strictly convex we can have more than 1 minimizer like:*

$$\beta_k^* = \begin{cases} \beta_k & \text{if } k \neq i \ \& \ k \neq j \\ (\beta_i + \beta_j)s & \text{if } k = i \\ (\beta_i + \beta_j)(1-s) & \text{if } k = j, \end{cases} \tag{35}$$

Proof: If $\beta_i\beta_j < 0$, we build another model with $\beta_i^* = \beta_j^* = \frac{1}{2}(\beta_i + \beta_j)$. Thus we have two minimizers, $(\beta_i, \beta_j) \neq (\beta_i^*, \beta_j^*)$, with $X\beta^* = X\beta$. But now, we have $|\beta^*| < |\beta|$ which violates the lasso penalty[15]. Thus we require that $\beta_i\beta_j \geq 0$ and from which (35) is derived. It basically means that if there are two similar predictors in a lasso setting, both of their signs will be the same (thus they do not cancel each other out) with the magnitudes at any point being divided in an interpolated fashion.

One can think of the L$^p, p \leq 1$ function as flat at the minimum as shown in Figure 7b (instead of a single unique point as in Figure 7a) and thus one can have different combinations to get different minimizers but with the same minimum function value. Thus, in summary: L$^p, p > 1$ penalty will make similar predictors to have similar values whereas the lasso penalty can (up to the algorithm) try to choose orthogonal predictors and suppress similar predictors. This is also evident from Figure 5b, in which instead of choosing solutions with both predictors, we can choose solutions at the sharp ends of the L^1 ball thus ending up with solution(s) that have just one predictor in the final model. This is also the connection between stepwise (which also chooses orthogonal predictors) regression and lasso regression (and as we will show later are not quite that different).

2.2 Oracle Properties and L^1 Penalty

We summarized earlier that the L^1 penalty encourages sparsity in the β vector, there are still questions to be answered with regards to the L^1 penalty like: *Whether the final model is consistent with the data? What is the relation of λ with the consistent model? What effects are caused by model bias for estimates having large magnitudes?*

[15]We can construct a solution with $\beta_i \to \infty, \beta_j \to -\infty$, with $X\beta^* = X\beta$ but not following the $|\beta| < c$.

In this subsection we will try to answer the above questions. The reader is cautioned not to get sidetracked concerning themselves about the working of the equations, as we have presented them as standard results for the completeness of this paper.

Asymptotic properties (shown below) of estimators, discussed in Davidson *et al.* [12], is the study of whether estimates converge to the true model estimates or not, as the sample size (N, which we represent as n over here) goes to infinity. Intuitively, the estimates of the parameters $\hat{\beta}$ ($\hat{\beta}^n$ for n samples) should approach the true parameters, which we represent as β_o over here, given enough samples (n).

1 Consistency property: $\lim_{n\to\infty} \hat{\beta}^n = \beta_o$

2 Normality property: for $n \to \infty$, sequence $n^{1/2}(\beta^n - \beta_o) \to W,$ where $W = N(0, \sigma^2)$

The consistency property implies that one would be able to estimate the true parameters if $n \to \infty$, whereas the normality property has implications about the rate of estimation, and both are related by a factor of $n^{1/2}$. When asymptotic normality holds, consistency also holds as $n^{1/2}(\beta^n - \beta_o)$ is O(1) $\Rightarrow$ $(\beta^n - \beta_o)$ is O($n^{-1/2}$), which is called the **root-n-consistent** property. This means that the difference between the estimator and true value is proportional to a factor of $\sqrt{n}$.

If the penalty is **root-n-consistent**, it means that asymptotically the penalty estimates are within a factor of $\sqrt{n}$ to the true parameters and if the penalty is not **root-n-consistent** (and assuming to be non-consistent[16]) than it means that asymptotically the estimates may diverge from the true parameters. This means is that in case there is noisy features in the data, a non **root-n-consistent** penalty will asymptotically start including noisy features into the model.

We will now summarize the results in Knight *et al.* [25], who investigated such asymptotic properties for the bridge regression. Instead of showing the general results for bridge regression we limit ourselves to lasso regression as:

$$\text{Lasso Regression:} \qquad \min_{\beta} ||y - X\beta||_2^2 + \lambda_n \sum_j |\beta_j|, \;\; \lambda_n \text{varies with n} \tag{36}$$

$$\text{Assume that} \qquad C_n = \frac{1}{n}\sum X^T X \to C \;\text{ with } C \geq 0$$

$$\hat{\beta}^{(n)} \text{ be the Lasso estimate and } A \text{ is the true feature model.}$$

Theorem 1: *If $\lambda_n/n \to \lambda_0 \geq 0$ then $\hat{\beta}^{(n)} \to_d$ argmin(Z) where $Z(\phi) = (\phi - \beta)^T C(\phi - \beta) + \lambda_0 \sum_{j=1}^{d} |\phi_j|$.*

Which thereby means that if $\lambda_n = O(n)$, argmin(Z) $= \beta$ and so $\hat{\beta}^{(n)}$ is consistent.

[16]for the sake of argument we assume here that other possible but not root-n-consistent estimators are also not consistent.

Theorem 2: *If $\lambda_n/\sqrt{n} \rightarrow \lambda_0 \geq 0$ then $\sqrt{n}(\hat{\beta}^{(n)} - \beta) \rightarrow_d argmin(V)$ where $V(u) = -2u^T W + u^T C u + \lambda_0 \sum_{j=1}^{p}[u_j sign(\beta_j) I(\beta_j \neq 0) + |u_j| I(\beta_j = 0)]$ and W has a $N(0, \sigma^2 C)$ distribution.*

For the above two theorems we describe λ_n **as the optimal** λ chosen for n samples. What we describe above is that, for consistency and normality, lasso regression follows the above two theorems. This is alright if we are not interested in feature selection ability of the lasso penalty as we have no information about the chosen feature set via the above theorems.

For **consistency** in **feature selection (FS)**, we need A_n=$\{\ j \mid \hat{\beta}_j^{(n)} \neq 0\}$ and $\lim_n$ P$(A_n = A)$=1, that is we asymptotically find the correct set of features. **Both theorems imply that the lasso estimates are root-n consistent, but tell nothing about the feature selection ability of the lasso penalty**. Zou [49] showed later that though the lasso estimator is consistent, it is **inconsistent as a feature selecting (FS) estimator** (which requires that we find the true model A_n as $n \rightarrow \infty$). He further proves that convergence rate of $\hat{\beta}^n$ needs to be slower than $\sqrt{n}$ if FS consistency is to be achieved, thus not following the optimal estimation rate.

Fan *et al.* [21], in a similar context, also argue about oracle[17] properties for a FS penalty. They summarize that if a penalty needs to identify the right subset model with an optimal estimation rate, it needs to follow the following 3 properties:

1. *Sparsity:* automatically setting small estimations to 0.
2. *Unbiasedness:* low bias when true coefficients are large.
3. *Continuity:* continuous w.r.t $|\beta| + \nabla J_p(|\beta|)$, where $J_p(|\beta|) = (\sum |\beta|^p)^{1/p}$ is set to 0, when undefined, so as to reduce instability[18].

L^p, p >1 penalties do not support sparsity (from Figure 3a), L^p, $0 \leq p < 1$ penalties do not support continuity (from Figure 3c, 3d, 3e, as there is a sudden jump at around 1) and L^1 penalty does not have unbiasedness (due to the fact that some variables, that should have larger magnitude, trail the OLS estimates by a certain value as seen in Figure 3b and discussed earlier in Section 2.1.1).

The oracle properties are not followed by any norm penalty thus leading to the proposal of the SCAD Penalty, shown below, by Fan *et al.* [21].

[17]like a Greek oracle the penalty should predict correctly in all cases.

[18]For Continuity: differentiation of the penalty should be smooth but for $L^p p$ <1, $\frac{d}{d\beta}||\beta||^p = p\beta^{p-1} \approx \frac{1}{0^{1-p}}$ = undefined if p <1 & $\beta = 0$.

$$\text{Smoothly Clipped Absolute Deviation (SCAD) Penalty} = \begin{cases} a|\beta| & 0 \le |\beta| < a \\ -(|\beta|^2 - 2ka|\beta| + a^2)/(2(k-1)) & a \le |\beta| < ka \\ (k+1)a^2/2 & |\beta| \ge ka \end{cases}$$

The L^1 penalty does not support oracle FS properties as observed in the estimates of the orthogonal design case: we shift by a constant λ due to the thresholding $[sign(\beta_{ols})(\beta_{ols} - \lambda)_+]$, thus biasing large valued parameters. SCAD penalty puts the estimates back on track by adding λ for high valued parameters. For more information refer to Fan *et al.* [21].

$$\text{Weighted Lasso Regression: } \min_{\beta} ||y - X\beta||^2 + \lambda \sum_j W_j |\beta_j| \tag{37}$$

$$\text{Improved 1-norm SVM: } \min_{\beta,\beta_0} \sum_i [1 - y_i(x_{:,i}\beta + \beta_0)]_+ + \lambda \sum_j W_j |\beta_j|, \text{ where } W_j = |\beta(l_2)_j|^{-\gamma}, \gamma = \{0.1, \dots\} \tag{38}$$

$$\beta(l_2) = \arg\min_{\beta,\beta_0} \sum_i [1 - y_i(x_{:,i}\beta + \beta_0)]_+ + \lambda_2 \sum_j ||\beta_j||_2^2 \tag{39}$$

Zou [49] showed that the unbiasedness for large valued parameters can be achieved if a weighted lasso penalty is used as shown in (37), thereby achieving unbiasness, consistency in FS and asymptotic normality. We show the weighted lasso formulation for regression and classification in (37) and (38) respectively. We also show the OLS plot in Figure 3f, and as can be seen that variables that have larger magnitudes have similar OLS and weighted lasso estimates. It means that the weighted lasso correctly estimates the parameters when they are large in magnitude. In short, the weighted lasso now follows the unbiasedness condition that the normal lasso penalty does not follow, thereby correctly following the asymptotic conditions required for FS. The added advantage of using the weighted lasso penalty is that the loss function still remains convex in nature and, as we will see later, will require almost no modification to the usual lasso penalized based algorithms.

So how does one find out these weights? The requirement, as mentioned by Zou [49], is the weights are somehow found from the data. Zou demonstrated, in [49] and [48], that estimates from the ridge penalty based algorithms can be used as weights.

Zou, in [48], showed that the weighted lasso penalty can be applied to 1-norm SVM (which we call as L^1 penalty based SVM) by using the weights (depicted by $\beta(l_2)$) from 2-norm SVM (which we call as L^2 penalty based SVM and shown in (39)) and which we show in (38). Note: We represent an example pair as $\{y_i, x_{:,i}\}$.

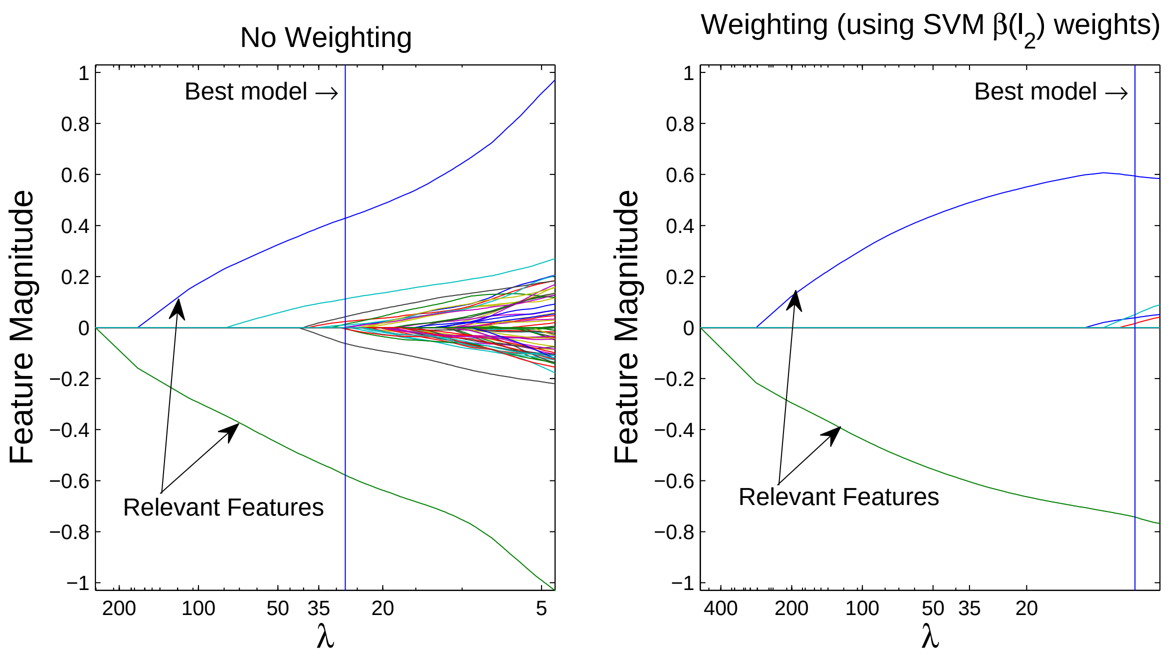


Figure 6: Non-weighted Vs. Weighted β estimates for Hinge^2 loss SVM algorithm

For Figure 6, we generated a classification dataset with 100 features/dimensions, 200 training examples and 200 test examples and use the Hinge^2 loss with L^1 penalty on it (we will discuss the algorithm in detail later on). There were two relevant features and the rest of the features are noisy but have a correlation of 0.2 with each other. The true features have the largest magnitudes in both plots. We set the weights for the weighted case as $W_j = |\beta(l_2)_j|^{-1}$ via the SVM weights $\beta(l_2)_j$. We plot for different values of the regularization parameter λ on the X-axis and the feature magnitude on the Y-axis. As can be seen, in plot for the weighted case, the weights helps keeping the magnitudes of the relevant features somewhat more than the non-relevant features. This shows empirically that knowing the right features can help suppressing the magnitudes of the wrong features, thus proving that the weighted L^1 penalty based algorithm can find out the correct set of features asymptotically.

2.3 Various models, Regularization and Sparsity

Support Vector Machines (SVM) have been very successfully applied to a plethora of machine learning problems. We will not go into the details of their inner working, so the reader is suggested the text by Hastie *et al.* [29]. SVM's in their basic form exhibit sparsity, as the final model depends only on support vectors within the margin. Note that we represent an example pair as $\{y_i, x_{:,i}\}$.

$$\text{Equivalent formulations} \begin{cases} \min\limits_{\beta,\beta_0} \sum\limits_i [1 - y_i(x_{:,i}\beta + \beta_0)]_+ + \lambda||\beta||_2^2 \\ \min\limits_{\beta,\beta_0} C \sum\limits_i \xi_i + \frac{1}{2}||\beta||_2^2, \quad \text{with } \xi_i \geq 0,\ y_i(\beta_0 + x_{:,i}\beta) \geq 1 - \xi_i \end{cases} \tag{40}$$

$$\text{Hinge loss} \quad [1 - y_i(x_{:,i}\beta + \beta_0)]_+ \quad \text{Hinge}^2 \text{ loss} \quad [1 - y_i(x_{:,i}\beta + \beta_0)]_+^2 \tag{41}$$

In (40), 'linear' loss is accumulated by use of ξ_i for the examples that are on the incorrect side of the separating hyperplane. Also, we have $\lambda = 1/2C$. We explicitly try weighting between the penalty and the loss and create various possible weight models, like a typical regularized algorithm. We then choose the model that performs best on the validation data using cross validation or other methods. β estimates, in the case of 2-norm SVM, are mostly non-zeros as we are using the L^2 penalty[19]. Also note the hinge and hinge^2 loss in (41). Next, we will discuss some interesting algorithms.

- **Dantzig Selector(DS):** In a seminal paper in compressed sensing, Candes *et al.* [7] attacked the problem of regression estimation:

$$\text{Dantzig Selector:} \min_\beta ||X^T(y - X\beta)||_\infty \text{ subject to } ||\beta||_1 \leq s \tag{42}$$

  DS, which is shown in (42), is a linear programming (LP) based problem compared to a quadratic programming (QP) problem for lasso regression. Efron *et al.* [20] show that there is much similarity between Least Angle Regression (LARS), a lasso-like regression algorithm, and DS in terms of β estimate paths for various values of s. As we will discuss later, this similarity is intuitive enough. The importance of this paper was that it gave strict bounds for signal recovery in compressed sensing.

- **1-norm SVM :** 1-norm SVM in (43) is the L^1 penalty formulation for SVM. Zhu *et al.* [46] showed that the β estimates change linearly and so one can effectively calculate the whole regularization path using linear programming (LP) at each step. We found this to be a tad-bit slower due to the LP step compared to a simple interpolation formula that comes with a double differentiable loss like that of

[19]In SVMs, sparsity in the # of support vectors is in-turn derived from the singularity (touching 0) occurring in the loss at 1.

Least Squares error (as in LARS). Please note that we will discuss more about this in a later section.

$$\min_{\beta,\beta_0} \sum_i [1 - y_i(x_{:,i}\beta + \beta_0)]_+ + \lambda||\beta||_1 \tag{43}$$

- *'Boosting - Regularized Path to a Maximum Margin Classifier'* by Rosset *et al.* [38] showed that boosting works by approximating the exponential loss with the L^1 penalty. We will not go into the details of boosting or the working of the paper but we briefly restate the authors conclusion that the excellent results obtained for boosting are in part due to the L^1 penalty.

2.3.1 Feature Groups instead of Individual Features

A limitation with lasso regression (via the Least Angle Regression (LARS) algorithm), as we will see later, is that the final model is limited to **minimum(N-1,D)**, for N examples and D features, 'orthogonal'[20] features. Instead of a single orthogonal feature, we might want to include all related features.

- **Elastic Net:** As seen earlier in Lemma-1, the L^2 (*ridge*) penalty tends to give similar predictors similar weights (which we refer as feature grouping), thus the elastic penalty was proposed by Zou *et al.* [50], shown in (44), which uses both the L^1 and L^2 penalty, with the idea that the L^1 penalty will promote sparsity and the L^2 penalty, feature grouping.

$$\min_{\beta} ||y - X\beta||^2 + \delta||\beta||_2^2 + \lambda||\beta||_1^1 \tag{44}$$

  For the elastic net, due to the L^2 penalty, true β estimates of large magnitudes will have increased shrinkage. For the orthogonal design case: elastic net estimates are shrunk by a factor $(1+\delta)$ and are non-zero only if the estimates are $> \lambda$. Thus we have the best of both shrinkage and thresholding.

- **Mixed Regularized SVM:** Wang *et al.* [43] later used the elastic penalty in conjunction with SVM's, which is shown in (45), thus allowing for feature groups (similar features similar magnitudes).

$$\min_{\beta,\beta_0} \sum_i [1 - y_i(x_{:,i}\beta + \beta_0)]_+ + \delta||\beta||_2^2 + \lambda||\beta||_1^1 \tag{45}$$

This concludes the discussion of various formulations and their motivational underpinnings. In the next section, we will discuss optimization methods to solve the aforementioned formulations.

[20]LARS algorithm has the final magnitudes in such a way that they span an orthogonal space.

3 Optimization Methods

In this section we initially talk in terms of basic optimization formulas, deriving them further into optimization methods. We will first talk about Taylor Series, which can approximate a function *f(x)* (infinitely differentiable) in a neighborhood of 'a' as: $f(x) = f(a) + \frac{\nabla f(a)}{1!}(x-a) + \frac{\nabla^2 f(a)}{2!}(x-a)^2 + \frac{\nabla^3 f(a)}{3!}(x-a)^3 + \ldots$

For a point 'y' near the point 'a' we can approximate the value of *f(y)* in 1^{st} order as:

$$f(y) = f(a) + \nabla f(a)(y-a) \qquad \text{Also at a local minima x*, } \nabla f(x*) = 0 \tag{46}$$

$$\Rightarrow y = a + \frac{f(y) - f(a)}{\upsilon}, \text{ where } \upsilon = \nabla f(a) \Rightarrow y \approx a - h\nabla f(a), h > 0 \tag{47}$$

We modify (47), via a small positive value h, to reflect that we can not move usually a full step in the direction of the gradient as that might cause an overshoot (note we assumed that the function is 1^{st}-order differentiable but in the general case it might be an n^{th}-order non-linear function). We also use a $-ve$ sign in (47) to show that we are moving in the direction opposite to the gradient (which points in the direction of maximum increase for the function), thus moving in the direction of maximum decrease.

We will next discuss about two methods that will help lay the foundation of the optimization methods discussed in this paper.

- **Gradient & Newton Method :** The gradient method works by generating successive points for moving toward the minima via iterating as shown below, which is a result from (47). Also $h_k > 0$ is called the ***step size***.

$$\text{Gradient Method: Choose } x_0 \in R^n, \qquad x_{k+1} = x_k - h_k \nabla f(x_k), \;\; k = 0, 1... \tag{48}$$

$$\text{Newton Method: Choose } x_0 \in R^n, \qquad x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k), \;\; k = 0, 1... \tag{49}$$

For the 2^{nd} order approximation, we get the minima at $f'(x) + f''(x)\nabla x = 0$, thus deriving the Newton method. The only difference between the gradient and the newton method is that the step size is now approximated as $h_k \approx [\nabla^2 f(x_k)]^{-1}$. The newton method has a quadratic rate of convergence, whereas the gradient method has a linear rate of convergence[21]. The newton method, though has a higher cost of calculating the inverse and the hessian during each iteration. Excellent treatment of various issues with the gradient and newton method is done in Nesterov [33]. Note: in case of **climbing up** the optimization curve the direction is not a positive semidefinite matrix ($\nabla^2 f(x_k) \leq 0$).

[21]when solution is in a neighborhood.

- **Steepest Descent Method for $\mathbf{L}^1$ norm :** In the gradient method, $-h_k \nabla f(x_k)$ is called the directional derivative of f in the direction of h_k. A normalized steepest descent direction in L^1-norm can be defined as:

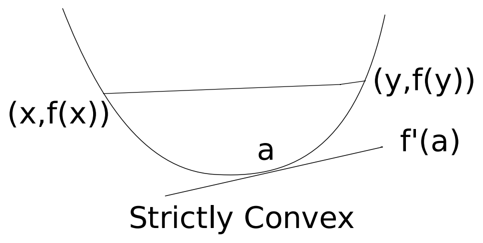
$$\Delta x_{nsd} = \min_{\forall k} \{ -\nabla f(x_k) h_k \mid ||h_k||_1 = 1 \} \tag{50}$$

  In (50), the most normalized steepest direction towards the minima is found. During each iteration, the component of $\nabla f(x)$ with the **maximum** absolute value is chosen, and we decrease or increase the corresponding component of x, according to the sign of $(\nabla f(x))_i$. The advantage, when using the L^1 penalty, is only one component is updated every iteration. Think of this in terms of an optimization setup: At each iteration, one moves along the path (increase/decrease the variable β_i) for which the loss is maximum, continuing till the point where some other variable starts dominating and then move along the new variable's path.[22]

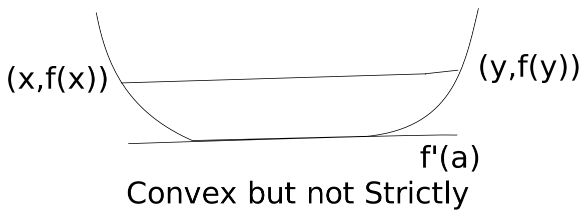
3.1 Convex Functions and Optimality

A function $f : R^n \rightarrow R$ is convex if the **dom**[23] of function 'f' is a convex set and if for all x, $y \in$ **dom** f, and there exists θ with $0 \leq \theta \leq 1$ such that,

$$\begin{aligned} &\text{Convex: } f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y) \\ &\text{Strictly Convex: } f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta) f(y) \end{aligned} \tag{51}$$



(a)



(b)

Figure 7: Two types of Convex Functions

We have two probable convex functions that we depict in the formula and in the figures above. The

[22]A variation, called the **Coordinate Descent**, in which initially a single variable whose gradient is highest is decreased till another variable's gradient becomes of equal value and then both these variables's gradient values are decreased till a third variable's gradient becomes equal and so on. Do note in this case all the constraints are fixed, and this is yet another optimization technique.

[23]**dom(f)** refers to the domain of f.

strictly convex version has a unique global minima but the other version has a trough at the bottom with possible multiple global minimas.

Geometrically, we can interpret the inequalities in (51) as the line segment between $(x, f(x))$ and $(y, f(y))$ always lying above the graph of f (with the by-product that the tangent at any point $(x, f(x))$ always lies below or along $f(x)$, meaning that (46) can be written as $f(y) \geq f(a) + \nabla f(a)(y-a)$). We show this in Figures 7a and 7b. This implies the existence of a global under-estimator[24] (and thereby a global minima) of the function using the 1st order Taylor approximation (if $\nabla f(a) = 0$, then for all $y \in \textbf{dom}\, f,\ f(y) \geq f(x)$). Similarly, the 2nd order conditions imply that the hessian is a positive semidefinite for convex functions (if we are moving towards the minimum), that is $\nabla^2 f(x) \geq 0$. For more details refer to either Boyd *et al.* [4] or Rockafellar *et al.* [37] or Nesterov *et al.* [33]. Recapping our discussion – convex functions have a global minima, $\nabla f(x^*) = 0$ at the minima x^* and $\nabla^2 f(x) \geq 0$ if we are correctly moving towards the minimum.

Note that we plot the strictly convex function and the (just) convex function in Figure 7. There is just one minima for strictly convex function as shown in Figure 7a. Though there might be multiple equivalent minimas for the (just) convex function case as shown in Figure 7b.

3.2 Regularized Models Formulation

The equivalence of the formulations shown in (52) was discussed earlier in Section 2.1.3. Let us now discuss about an optimization approach whose principles are behind the design of fast L^1 penalty based methods. We will be mostly discussing in terms of the first formulation in (52).

$$\min_{\beta} \sum_i L(x_{:,i}, y_i, \beta) + \lambda ||\beta||_p \iff \min_{\beta} \sum_i L(x_{:,i}, y_i, \beta),\ \ ||\beta||_p \leq t \tag{52}$$

3.2.1 Homotopy based Optimization :

A homotopy based approach to Constrained optimization problems entails on using **a known solution to a problem as an initial estimate and then solving a similar problem with slightly different constraints**. The intuition: if there are small variations in the original problem, the characteristics of the problem are unchanged and the solution to the modified problem lies nearby the original problem. **The goal now is to find a trajectory from the solution of the old problem to the solution of the new problem.**

Osborne *et al.* [35] formulated conditions for homotopy based optimized for lasso regression in which a

[24]a function that underestimates the current function at each point.

trajectory from the old solution to the new can be found when the λ or t in (52) is changed slightly. Breaks in the trajectory occur when certain conditions turn false and thus the trajectory needs to be recalculated. Their algorithm is precursor to the Least Angle Regression(LARS) algorithm and drives many ideas behind it. Before discussing how LARS, by Efron *et al.* [19], uses a computationally simple method to solve lasso Regression, let us discuss the fact that the LARS based method can be generalized into a method that can be applied to all double-differentiable loss functions with the L^1 penalty. To that end, we first discuss piecewise linearity, and come up with mathematical justification for why the solution is optimal for the entire regularization path.

3.2.2 Piecewise Linearity :

Consider that the loss function $L(X, y, \beta)$, shown in (52), is a double differentiable loss function [e.g. hinge^2 loss (41) and least squares.] then the optimal path denoted by $\beta(\lambda)$ is piecewise linear[25] as a function of λ when the penalty is lasso penalty. This means that $\exists \lambda_0 = 0 < \lambda_1 < ... < \lambda_m = \infty$ and $\gamma_0, \gamma_1, ...\gamma_m \in R^p$ such that $\beta(\lambda) = \beta(\lambda_k) + (\lambda - \lambda_k)\gamma_k$ for $\lambda_k \leq \lambda \leq \lambda_{k+1}$. Now, one can generate the whole regularized path $\beta(\lambda)$, for $0 \leq \lambda \leq \infty$ simply by sequentially calculating the "step sizes" between consecutive λ values and the directions $\gamma_1, ...\gamma_m$. In a nutshell, when solving for a λ value, the value of the weights β move linearly to the next λ value (when the loss in question is double-differentiable and the penalty involved is the lasso penalty). The question now is: ***how to calculate this 'next λ' value?***

3.2.3 Recap: What Conditions are needed for Calculating the Entire Regularization path?

Let us now formulate conditions that are needed to calculate the entire regularization path (that is, solve for $0 \leq \lambda \leq \infty$). First, we can combine the ideas from the newton method and steepest descent method (50). The newton method has a quadratic speed of convergence whereas the steepest descent method shows a simple way to evaluate for L^1 penalty problems. Intuitively, we can combine the above methods as:

a The step size h_k of steepest descent can be substituted with the Newton step $[\nabla^2 L(\beta_k)]^{-1}$.

b Use the homotopy based argument to evaluate for not only a fixed λ_k but for $0 \leq \lambda \leq \infty$.

c Come up with an argument that this is optimal for all values.

In our case, due to the existence of a global minima, the newton step will have $\nabla^2 L \geq 0$ if we are moving correctly. Next, we will come up with arguments for steps (b) and (c).

[25]between eligible λ values, the magnitude of β_i changes linearly.

3.3 Necessary and Sufficient Conditions for Optimality for the Whole Regularization path

Let us now concern ourselves only with double differentiable loss functions[26] and the lasso penalty as shown in (53).

$$\text{Lasso penalized algorithms:} \quad \min_{\beta} \sum_i L(x_{:,i}, y_i, \beta) + \lambda ||\beta||_1 \tag{53}$$

We already discussed the KKT conditions, in Section 1.3.2, which can serve as the basis to prove optimality for our designed algorithms. Formulating (53) by breaking up[27] β_j as $\beta_j = \beta_j^+ - \beta_j^-$, and $\beta_j^+ \geq 0, \beta_j^- \geq 0$. We also substitute $L(y_i, (\beta^+, \beta^-)x_{:,i}) = L(x_{:,i}, y_i, \beta)$:

$$\text{Primal Formulation} : \min_{\beta^+,\beta^-} \sum_i L(y_i, (\beta^+, \beta^-)x_{:,i}) + \lambda \sum_j (\beta_j^+ + \beta_j^-) \text{ such that, } \beta^+ \geq 0,\ \beta^- \geq 0, \forall j \tag{54}$$

$$\text{Dual Formulation} : \min_{\beta^+,\beta^-} \sum_i L(y_i, (\beta^+, \beta^-)x_{:,i}) + \lambda \sum_j (\beta_j^+ + \beta_j^-) - \sum_j \gamma_j^+ \beta_j^+ - \sum_j \gamma_j^- \beta_j^- \tag{55}$$

-ve values as the constraints are usually ≤ 0

Let us now list the KKT conditions that need to be true at all optimal points for the optimization function in (55). We will now use a shorthand notation for the loss function L as the proof is agnostic of the exact formulation.

$$\text{For inequality constraints (from (14)):} \quad \gamma_j^+ \beta_j^+ = 0 \ \& \ \gamma_j^- \beta_j^- = 0$$

$$\text{Breaking (15), per variable:} \quad [\nabla L(\beta)]_j + \lambda - \gamma_j^+ = 0 \ \& \ -[\nabla L(\beta)]_j + \lambda - \gamma_j^- = 0 \tag{56}$$

We show, in (56), an additional constraint that will make sure that if there are similar features in the model, only one of them will be entered into the model. At optimal, for fixed $\lambda > 0$ (for $\lambda = 0$, $\beta = 0_{1..j}$) as:

$$\text{if } \beta_j^+ > 0, \lambda > 0 \quad \Longrightarrow \gamma_j^+ = 0 \Longrightarrow [\nabla L(\beta)]_j = -\lambda < 0 \Longrightarrow \gamma_j^- = 0 \Longrightarrow \beta_j^- = 0 \tag{57}$$

$$\text{if } \beta_j^- > 0, \lambda > 0 \quad \Longrightarrow \gamma_j^- = 0 \Longrightarrow [\nabla L(\beta)]_j = \ \ \lambda > 0 \Longrightarrow \gamma_j^+ = 0 \Longrightarrow \beta_j^+ = 0 \tag{58}$$

From (57) and (58), we can surmise that either of β_j^+ or β_j^- (and similarly between γ_j^+ or γ_j^-) is non-zero. Similarly, $|[\nabla L(\beta)]_j| = \lambda$ means that variable 'j' can have non-zero coefficients for $\beta(\lambda)$, only if $|[\nabla L(\beta)]_j| = \lambda$. Thus, for each λ, we have sets of variables that can be divided on the basis of $|[\nabla L(\beta)]_j| = \lambda$ (Active set $A = \{j \in \{1..m\} : \beta(\lambda) \neq 0\}$) and $|[\nabla L(\beta)]_j| < \lambda$ (Active Complementary set A^c).

[26]The KKT conditions work for singly-differentiable loss function also but then one can not use the Newton Step.
[27]it helps to reduce the 2^D constraints with 2D+1 constraints.

$$j \in A \qquad \Longrightarrow |[\nabla L(\beta)]_j| = \lambda, sign[\nabla L(\beta)]_j = -sign(\beta(\lambda))_j \tag{59}$$

$$j \notin A (or j \in A^c) \qquad \Longrightarrow |[\nabla L(\beta)]_j| < \lambda \tag{60}$$

We need to move $\beta(\lambda)$ such that $|[\nabla L(\beta)]_j| = \lambda, j \in A$ and $|[\nabla L(\beta)]_j| < \lambda, j \in A^c$. This concludes the proof that an algorithm devised, to behave like the Active set based homotopy method shown above, will find the optimal point at each part in the regularization path provided it never violates the KKT conditions of (57),(58),(59) and (60). Let us now discuss on ***how to move*** $\beta(\lambda)$.

$$\text{Choose} \qquad \beta_0 = \{0,..,0\} \in R^n, \tag{61}$$

$$\text{According to Newton's Method,} \qquad \beta_{k+1} = \beta_k - [\nabla^2 f(\beta_k)]^{-1} \nabla f(\beta_k), \quad k = 0,1... \tag{62}$$

$$\text{where } \nabla f(\beta_k) = -\nabla L(X,Y,\beta) \text{ or } \lambda\, \nabla J(\beta) \quad \text{as } \nabla L + \lambda \nabla J = 0$$

$$\text{Substituting L and J in (61),} \qquad \beta_{k+1} = \beta_k - [\nabla^2 L(\beta_k)]^{-1} \nabla J(\beta_k),$$

$$\text{Thus,} \qquad \frac{\partial \beta(\lambda)_k}{\partial \lambda} = -[\nabla^2 L(\beta_k)]^{-1} \nabla J(\beta_k) = -[\nabla^2 L(\beta_k)]^{-1} Sign(\beta(\lambda)), \quad k \in A \tag{63}$$

So in summary:[28]

1. KKT conditions will be violated if a new variable (not in Active set) achieves $|[\nabla L(\beta)]_j| = \lambda$.
2. If a variable hits 0, the KKT conditions will be violated as $sign[\nabla L(\beta)]_j = -sign(\beta(\lambda))_j$.
3. If the loss function is not totally differentiable everywhere (e.g. Hinge^2 Loss (41), Huber loss (64)) then the direction becomes invalid, whenever such a transition occurs, requiring that the direction is recalculated.

3.3.1 Algorithm: For "doubly differentiable" Loss with L^1 Penalty

We show an algorithm below that surmises the KKT conditions of the earlier subsection.

1. Initialize $\beta = 0_{1..D}, A = \max_j |\nabla L(\beta)|_j, \gamma_A = -sign(\nabla L(\beta))_A, \;\; \gamma_{A^c} = 0$
2. While $(\max|\nabla L(\beta)| > 0)$

   a) $d_1 = \min\{d > 0 : |\nabla L(\beta + d\gamma|)_j| = |\nabla L(\beta + d\gamma|)_A|, j \notin A\}$ (Adding the Variable to A).

[28]Note: we can use $\nabla f(\beta_k) = -\nabla L(X,Y,\beta)$, the reason being that $J(\beta(\lambda)) - J(\beta(\lambda+\epsilon)) \approx J(\beta(\lambda)) + f_1(\beta(\lambda) - \beta(\lambda+\epsilon))\nabla J(\beta(\lambda)) - J(\beta(\lambda)) = f_1(\beta(\lambda) - \beta(\lambda+\epsilon))\nabla J(\beta(\lambda)), f_1(\beta(\lambda) - \beta(\lambda+\epsilon)) \geq 0$ and $L(\beta(\lambda)) - L(\beta(\lambda+\epsilon)) \approx L(\beta(\lambda)) - f_2(\beta(\lambda) - \beta(\lambda+\epsilon))\nabla L(\beta(\lambda)) - L(\beta(\lambda)) = -f_2(\beta(\lambda) - \beta(\lambda+\epsilon))\nabla L(\beta(\lambda)), f_2(\beta(\lambda) - \beta(\lambda+\epsilon)) \geq 0$, $L(\beta)$ monotonically decreases and $J(\beta)$ monotonically increases every iteration, with $\lambda \to 0$, which is the justification for the $+ve$, $-ve$ signs for the difference accumulated in J and L. Thus $\dfrac{L(\beta(\lambda)) - L(\beta(\lambda+\epsilon))}{f_2(\beta(\lambda) - \beta(\lambda+\epsilon))} = -\nabla L(\beta(\lambda)) \propto \nabla J(\beta(\lambda))$ $[\because \nabla L + \lambda \nabla J = 0]$

b) $d_2 = \min\{d > 0 : (\beta + d\gamma|)_j = 0, j \in A\}$ (Deleting the Variable from A).

c) $d_3 = \min\{d > 0 : r(y_i, (\beta + d\gamma)x_i) = 0$ hits a "knot",$i = 1 \dots N\}$ (Recalculate the gradients).

d) set d=min(d_1,d_2,d_3)

e) $\beta = \beta + d\gamma$

f) $\gamma_A = |\nabla^2 L(\beta_A)|^{-1}(-sign(\beta_A))$, $\gamma_{A^c} = 0$

In step 2c above, we use r() function to define distance of a training example to a knot point (non-differentiable point). Henceforth, when we talk about 'steps', we will be referring to the above algorithm.

Knot points: Consider the Huber loss shown in (64), the loss function changes between the limits set via 't'. Both ∇L & $\nabla^2 L$ are changed when we move across the point 't', which happens when a training example moves from the quadratic loss region to the linear loss region (or vice versa). This is taken care in step 2c.

$$\text{Huber Loss:} \quad L(x_{:,i}, y_i, \beta) = \begin{cases} (y_i - x_{:,i}\beta)^2 & \text{if } |y_i - x_{:,i}\beta| \le t \\ 2t|y_i - x_{:,i}\beta| - t^2 & \text{Otherwise} \end{cases} \tag{64}$$

$\beta_i(\lambda)$ **touching 0:** KKT conditions are violated whenever any variable's loss function touches 0 (due to the requirement that the sign of the ∇L do not flip). We will though later see that Least Angle Regression (LARS) relaxes this requirement.

3.3.2 Losses' and Corresponding Algorithms for Solving the entire Regularization Path

In this section, we show the derivations of the piecewise algorithm for different loss functions (including Least Squares (denoted by LSE+L^1) and Squared Hinge Loss (41) (denoted by SVM^2)) and lasso penalty . We will then derive and differentiate LARS from the LSE+L^1 Algorithm. We will assume that y is a (Nx1 vector), β is a (Dx1 vector) and X is a (NxD matrix), with $\{y_i, x_{:,i}\}$ denoting an example and x_j denoting the j^{th} column of X.

- Consider the lasso regression formulation (there is no β_0 as y is centered ($\sum_i y_i = 0$)) as

$$\text{Lasso Regression:} \quad \min_\beta ||y - X\beta||_2^2 \quad + \quad \lambda||\beta||_1^1 \tag{65}$$

$$\nabla L_j = \nabla[(y - X\beta)^2] = -2(y - X\beta)^T x_j \qquad \nabla^2 L_{j,j} = 2(x_j^T x_j),\ \nabla^2 L_{j,k} = 2(x_j^T x_k) \tag{66}$$

$$\begin{aligned} \text{For finding } d_1, |\nabla L(\beta + d\gamma)|_A &= |\nabla L(\beta + d\gamma)|_j \implies \nabla L(\beta + d\gamma)_A = \pm \nabla L(\beta + d\gamma)_j \\ \implies (y - X(\beta + d\gamma))^T x_A &= \pm(y - X(\beta + d\gamma))^T x_j \\ \implies y^T x_A - (X\beta)^T x_A \mp y^T x_j \pm (X\beta)^T x_j &= ((X\gamma)^T x_A \mp (X\gamma)^T x_j)d \end{aligned}$$

$$\Longrightarrow \text{Step size } d_1 = \min\left\{\frac{A\_Correlation - J\_Correlation}{(X\gamma)^T(x_A - x_j)}, \frac{A\_Correlation + J\_Correlation}{(X\gamma)^T(x_A + x_j)}\right\} \tag{67}$$

$$\text{where, } A\_Correlation = (y - X\beta)^T x_A, J\_Correlation = (y - X\beta)^T x_j$$

$$\text{Similarly for step 2b, } \beta_j + d_2\gamma_j = 0 \Longrightarrow \underline{d_2 = -\beta_j/\gamma_j} \tag{68}$$

Note that we have a smooth loss function for least squares and so there is no Knot step (step 2c) in this case. We can now substitute the derivations into the generic algorithm discussed earlier.

- Consider the Hinge Loss for SVM with L^1 penalty as:

$$\text{SVM+L}^1\text{:}\quad \min_{\beta,\beta_0} \left\{\lambda||\beta||_1 + \sum_i (1 - y_i * (x_{:,i}\beta + \beta_0))_+,\ \ (v)_+ = max(v,0)\right\} \tag{69}$$

$$\text{Equivalent formulation} \Longrightarrow \min_{\beta,\beta_0} \left\{||\beta|| + C\sum_i \xi_i, \text{ such that } y_i(x_{:,i}\beta + \beta_0) \geq 1 - \xi_i,\ \xi_i \geq 0\right\} \tag{70}$$

$$\text{SVM}^2\text{+L}^1\text{:}\quad \min_{\beta,\beta_0} \left\{\lambda||\beta||_1 + \sum_i (1 - y_i * (x_{:,i}\beta + \beta_0))_+^2,\ \ (v)_+ = max(v,0)\right\} \tag{71}$$

SVM loss, as shown in (69), is not a doubly differentiable function and thus we consider the hinge squared loss as shown in (71). We will represent the dot product as '$*$' and 1_n is a nx1 vector (when there are n examples having positive loss).

$$\begin{aligned} L &= (1_n - y * (X\beta) - y * (\beta_0))_+^2 \\ \text{For each feature '}j\text{', } \nabla L_j &= -2(1_n - y * (X\beta - \beta_0))_+^T(y * x_j) \end{aligned}$$

$$\nabla^2 L_{j,j} = 2(-y * x_j)^T(-y * x_j) = 2x_j^T x_j \quad , \quad \nabla^2 L_{j,k} = 2(-y * x_k)^T(-y * x_j) = 2x_j^T x_k$$

$$\nabla^2 L_{j,\beta_0} = 2(-yx_j)^T(-y) = 2\sum_k x_{jk} \quad , \quad \nabla^2 L_{\beta_0,\beta_0} = 2(y)^T(y) = 2y^T y$$

$$\begin{aligned} \text{For finding } d_1, |\nabla L(\beta + d\gamma)|_A &= |\nabla L(\beta + d\gamma)|_j \Rightarrow \nabla L(\beta + d\gamma)_A = \pm\nabla L(\beta + d\gamma)_j \\ \Rightarrow (1_n - y * (X\beta + \beta_0 + d\gamma_0 + Xd\gamma))^T(y * x_A) &= \pm(1_n - y * (X\beta + \beta_0 + d\gamma_0 + Xd\gamma))^T(y * x_j) \end{aligned}$$

$$\Rightarrow (1_n - y * (X\beta + \beta_0))^T(y * x_A) \mp (1_n - y * (X\beta + \beta_0))^T(y * x_j)$$

$$= \mp y * ((\gamma_0 + X\gamma)(y * x_j) + (\gamma_0 + X\gamma)(y * x_A))d$$

$$\Longrightarrow \text{Step size } d_1 = \min\left\{\frac{A\_Correlation - J\_Correlation}{A\_val - J\_val}, \frac{A\_Correlation + J\_Correlation}{A\_val + J\_val}\right\} \tag{72}$$

where, $A\_Correlation = (1_n - y * (X\beta + \beta_0))^T(y * x_A), J\_Correlation = (1_n - y * (X\beta + \beta_0))^T(y * x_j)$

$$A\_val = (\gamma_0 + X\gamma)(x_A), \quad J\_val = (\gamma_0 + X\gamma)(x_j) \quad (\because y * y = \sum 1_n) \tag{73}$$

In the above derivation, γ is a Dx1 vector and γ_0 a single value, with a direction value for β and β_0 respectively. Note that, for both the SVM^2 and LSE version with L^1 penalty, the step size 'd_1' is just the minimum of the ratio of difference between the generalized correlations and difference of directional estimates ($X\gamma x_j$).

- **Handling Offset** β_0 **:** Do note the fact that according to the generic piecewise algorithm, that we discussed earlier, all the variables in the active set A have the same gradient value. And thus we cannot directly substitute the gradient of the offset β_0 (it is also complicated by the fact that the gradient of β_0 is always 0, which we show below). Though there are clues in the algorithm that will help find the offset.

  First consider that $\gamma_i = \gamma + \gamma_0$, where γ is the derivation which we found earlier and γ_i is the corrected direction. If we find the value of γ_0, in terms of γ than we don't have to worry about the mess that happens because of $\nabla_{\beta_0} L = 0$. We thus show the derivation below.

$$\nabla_{\beta_0} L = \nabla_{\beta_0}(1_n - y*(X\beta) - y*(\beta_0))_+^2 = (1_n - y*(X\beta) - y*(\beta_0))^T(-y) \quad (74)$$

$$\text{And we know that, } \nabla_{\beta_0}(L + \lambda J) = 0, \text{at every iteration}$$

$$\Rightarrow (1_n - y*(X\beta) - y*(\beta_0))^T(-y) + \lambda(0) = 0$$

$$\Rightarrow (1_n - y*(X\beta) - y*(\beta_0))^T(-y) = 0$$

$$\text{which also implies that} \quad (1_n - y*(X\beta) - y*(\beta_0) - y*(Xd\gamma) - y*(d\gamma_0))^T(-y) = 0$$

$$\Rightarrow (y*(X\gamma + \gamma_0))^T(y) = 0 \Rightarrow (y*(X\gamma))^T y + (y*\gamma_0)^T(y) = 0$$

$$\Rightarrow \sum X\gamma + n\gamma_0 = 0 \Rightarrow \boxed{\gamma_0 = -\frac{\sum X\gamma}{n}} \quad (75)$$

  Thus we substitute γ_i instead of γ in the generic algorithm as $\boxed{\gamma_i = \gamma - \frac{\sum X\gamma}{n}}$. Next, we will discuss the derivation when the elastic penalty is concerned.

- For the elastic net formulation, which we discussed Section 2.3.1, the difference when compared to the LSE+L^1's d_1 step shown in (67), is that the L^2 penalty will also influence the step size as shown in (77), thus shrinking high valued parameters and spreading the magnitude to similar but lesser valued parameters. When solved for a SVM^2+L^1+L^2 based formulation we will have a similar effect of shrinkage+thresholding. This can also be seen in the generalized correlation for the elastic net as shown in (77) in which if the magnitude of $\beta_k(k = A \text{ or } j)$ is greater the generalized correlation is smaller, thus having the effect of shrinkage.

$$\begin{aligned}
\text{Elastic Net:} \quad \min_{\beta} ||(y - X\beta)||_2^2 + &\quad \delta||\beta||_2^2 + \lambda||\beta||_1^1 && (76)\\
\nabla L_j = &\quad \nabla[(y - X\beta)^2 + \delta||\beta||_2^2] = -2(y - X\beta)^T x_j + 2\delta\beta_j \\
\Rightarrow |\nabla L(\beta + d\gamma)|_A = &\quad |\nabla L(\beta + d\gamma)|_j \\
\Rightarrow (y - X(\beta + d\gamma))^T x_A - \delta(\beta_A + d\gamma_A) = &\quad \pm\left((y - X(\beta + d\gamma))^T x_j - \delta(\beta_j + d\gamma_j)\right) \\
\Rightarrow (y - X\beta)^T x_A - \delta\beta_A \mp ((y - X\beta)^T x_j - \delta\beta_j) = &\quad (\pm\delta\gamma_j \mp X\gamma x_j - \delta\gamma_A + X\gamma x_A)d \\
d = \frac{A\_Correlation - J\_Correlation}{X\gamma(x_A - x_j) + \delta\gamma_j - \delta\gamma_A}, &\quad \frac{A\_Correlation + J\_Correlation}{X\gamma(x_A + x_j) + \delta\gamma_j + \delta\gamma_A} && (77)\\
\text{Where,} \ A\_Correlation = (y - X\beta)^T x_A - \delta\beta_A, &\quad J\_Correlation = (y - X\beta)^T x_j - \delta\beta_j
\end{aligned}$$

The original paper for LARS [19] modifies, at each step, the correlation (more intuitive to a statistician) instead of the β estimates (as shown above for the LSE+L^1 algorithm). In the appendix we prove that the LSE+L^1 algorithm's '2a' step, as shown earlier, and as taken by LARS are equivalent.

3.4 Difference between LSE+L^1 and Least Angle Regression (LARS)

We will now discuss the differences between LSE+L^1 (discussed in Section 3.3.2) and Least Angle Regression (LARS). The original paper, by Efron *et al.* [19], was an important paper as they showed that one can compute the whole regularization path for lasso regression very efficiently (by using the piecewise algorithm). The paper was motivated for statisticians and thus primarily discussed regression in terms of correlation and other statistical metrics. Earlier we gave an interpretation in terms of the β estimates and as the reader can observe in the appendix, both the interpretations are equivalent.

There is a difference between LSE+L^1 based algorithm discussed here and LARS. It is due to the fact that LARS just obeys the step size d_1. That is, one does not drop the variable when it touches 0. Thus if we allow that only step 2a is taken in the generic piecewise algorithm then our described algorithm (LSE+L^1) and LARS will generate the same regularization path.

$$\text{LARS:} \qquad \text{for every iteration } |(Y - X\beta)X_i|_\infty \le s, \quad 0 \le s \le \infty \tag{78}$$

$$\text{From Osborne } et\ al. \text{ [35] (LASSO):} \qquad \min_{\beta} \beta^T X^T X\beta, \text{ subject to } |(Y - X\beta)X_i|_\infty \le s, \quad 0 \le s \le \infty \tag{79}$$

In (78), we show a representative formulation (because it is not technically a L^1 penalty based formulation

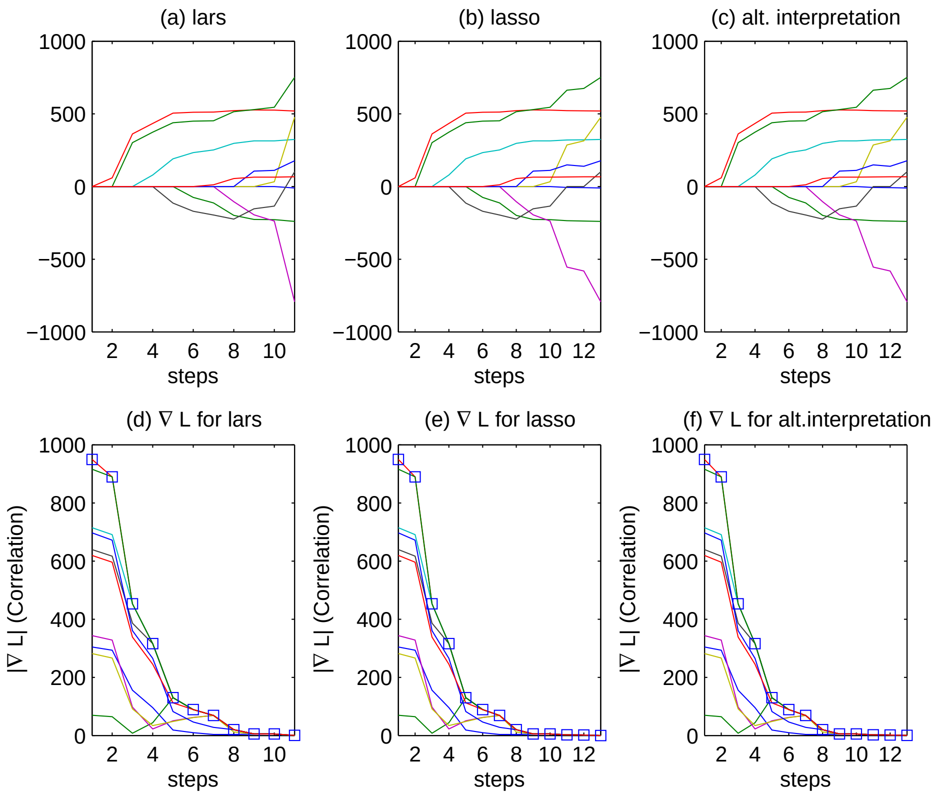


Figure 8: Plots for (a),(d) - LARS, (b),(e) - lasso regression and (c),(f) - alt. interpretation

but behaves as one). In Osborne *et al.* [35], a similar interpretation was given for the lasso regression, shown in (79), to identify what really happens in the LARS method[29]. Do note that for (79), if we consider the orthogonal design case ($X^T X = I$), then it has the same interpretation that we gave for (78).

In LARS, we start with $\beta = 0$ and identify the variable having the maximum correlation with 'y'. We then, increase/decrease the value of first variable (depending on the sign of the variable), which will then reduce the residual. We basically move along (increase/decrease the value of β added first) till another variable has equal correlation with it. Again, we will increase/decrease the magnitudes of both these variables till a third variable gets equal correlation and so on... Technically this seems a lot like the LSE+L^1 model that we discussed earlier, except that there is no dropping of the variable. The original LARS paper formulated optimality in terms of statistics, and showed that the algorithm will always guarantedly reduces the maximum correlation as it moves along the regularization path. Their proof on why LARS will always converge was also based on the fact that the path taken by LSE+L^1 is roughly modeled by LARS, but with a faster rate. Here, we will now explain this in terms of violations of KKT conditions.

[29] Does not this look a lot like the Dantzig Selector from Section 2.3?

Consider the formulation for LSE+L^1 given in (65), and for the piecewise algorithm in Section 3.3.1: LARS will violate $sign[\nabla L(\beta)]_j = -sign(\beta(\lambda))_j$ by not stopping when the β path touches 0. So what LARS does is to weaken the condition of lasso regression, by ***requiring that every new variable entering the Active set, enters it in the "correct" direction***. That means that the ***older variables already in the Active set can violate the sign requirements***. Thus, in cases where variables never meet 0, LARS is equivalent to lasso regression. But, in cases where variables do meet 0, LARS solves almost like the lasso regression.

We show the results of these algorithms in Figure 8 for the **Pima Indians Diabetes** dataset [3]. For LARS, in Figure 8a, the regularization path finishes in 11 iterations while for the lasso regression and alternative interpretation, shown in 8b and 8c respectively, we finish in 13 iterations. This happens as the latter two algorithms drop a variable at step 11 (which can be seen in the figure). Also note that the regularization path for lasso regression (solved via LARS), shown in 8b, and the path for the alternate algorithm (described earlier), shown in 8c, are the same. We also show the value of the correlation for all three algorithms in Figures 8d, 8e and 8f. For all the three algorithms, we see that the absolute correlation always decreases as the number of iterating steps of the algorithm increases. Note that we mark blue squares denoting the addition of a new variable into the model.

We can see in (67) that the maximum correlation ($\hat{C}_k$) declines with each step and it is also derived in (80). Note that even if we violate the sign changing condition, we will keep on decreasing the absolute correlation. Thus, LARS has the advantage of always terminating in fixed number of steps (compared to lasso regression which might have more steps due to dropping of a variable).

$$\begin{aligned}
|\hat{C}_k| &= max\{|C_j|\} = max\{|X_j^T(Y - X\beta_k)|\}, \;\; j = 1...N \\
&= max\{|X_j^T(Y - X(\beta_{k-1} + d\gamma_{k-1}))|\} \quad \because \beta_k = \beta_{k-1} + d\gamma_{k-1} \\
&= max\{|X_j^T(Y - X\beta_{k-1}) - X_j^T X d\gamma_{k-1}|\} \\
&= max\{|\hat{C}_{k-1} - X_j^T X d\gamma_{k-1}|\} \quad \because \text{of all possible } C \text{ the last } \hat{C}_{k-1} \text{ was the max} \qquad (80) \\
\Rightarrow |\hat{C}_k| &< |\hat{C}_{k-1}| \quad (\because sign(X_j^T X_j d\gamma_{k-1}) \propto d\nabla L_j \propto sign(\hat{C}_{k-1}), \text{ if } j \text{ is the new added variable})
\end{aligned}$$

Let us now discuss the number of steps required to finish the LARS algorithm for D variables and N examples. For $D < N$, **after D variables are entered into set A (after D steps)**, (67) will be not defined and thus d_1=0 and we completely fit the model $y = X\beta$). For $D > N$, LARS terminates after N-1 variables have been entered into the model because $X_A'X_A$ now has a rank of N-1 (due to the centering of the columns of X, A is the Active set). Think of this in terms of the number of equations required to fully represent K variables (the answer is K). Thus, ***LARS*** would have at most ***maximum(D,N-1) steps*** and the final model will have a ***maximum(D,N-1) selected variables***. This brings us to an important conclusion: even the LSE+L^1 algorithm will have a maximum(D,N-1) selected variables, but there might be more iteration steps involved (because there might be steps in which variables are dropped from the model). **The advantage of LARS is that it will always terminate in maximum(*D*,*N*-1) steps.**

3.5 Modification to the Formulations

In this subsection we will discuss additional formulations that can take advantage of the piecewise algorithm.

- **Positive Lasso** is formulated, as shown in (81) and discussed in detail in Hastie *et al.* [28], so that there is an additional constraint compared to the lasso regression formula. This additional constraint is that $\beta \geq 0$, which will require that no step is taken that makes 'β' negative. That is why there is a single value for step d_1 (compared to the two probable values for LSE+L^1 algorithm), which ensures that only positive β estimates are used.

$$\text{Positive Lasso:} \quad \min_{\beta} \sum_i (Y_i - X_i\beta)^2 \text{ such that } ||\beta||_1 \leq t \text{ and } \beta_j \geq 0 \tag{81}$$
$$\Longrightarrow \text{Solution: Step size } d_1 = \frac{A\_Correlation - J\_Correlation}{X\gamma(X_A - X_j)}$$
$$where,\ A\_Correlation = (Y - X\beta)^T X_A,\ \ J\_Correlation = (Y - X\beta)^T X_j$$

- **Monotone Lasso** is formulated so that the β values never decrease; that is, the weights are always monotonically increasing as mentioned in Hastie *et al.* [28].

- **Fused Lasso** formulation, as shown in (82) and discussed in detail in Tibshirani *et al.* [40], has been used in protein mass spectroscopy . The intuition over here is that the nearby predictors are expected to be more correlated than distant ones and so one should penalize the model higher if nearby predictors differ by a larger value.

$$\min_{\beta} ||Y - X\beta||^2 + \lambda_1 \sum_j |\beta_j| + \lambda_2 \sum_j |\beta_j - \beta_{j-1}| \tag{82}$$

- **Elastic Net** formulation, as shown in (76), is an additional formulation that tries to get more than min(N-1,D) variables into the final model (as discussed before, lasso regression limits its final model to min(N,D-1) variables). The L^2 penalty is used in conjunction with the L^1 penalty and thus we can derive an efficient piecewise algorithm to solve it. We will like to remind the reader that the properties of elastic penalty were already covered in Section 2.3.1.

- **Weighted Lasso Regression** formulation has theoretical underpinnings as discussed in Section 2.2. We can solve (37) via the generic piecewise algorithm, by first substituting $x_j^* = x_j/W_j$ and solving for entire path as:

$$\min_{\beta} ||Y - X^*\beta||^2 + \lambda \sum_j |\beta_j|, \text{ with final } \beta_j^w = \beta_j/W_j \tag{83}$$

That is in terms of the gradients:

$$\nabla L_j \approx (Y - X^*\beta)^T(X_j^*) = (Y - \frac{X}{W}\beta)^T(\frac{X_j}{W_j}) = (Y - X\beta^w)^T(\frac{X_j}{W_j}) = \frac{(Y - X\beta^w)^T X_j}{W_j} \tag{84}$$

$$\text{Alternatively, } \nabla L_j = \lambda W_j, j \in A(\text{ instead of } \nabla L = \lambda) \Rightarrow \nabla L_j/W_j = \lambda \tag{85}$$

Consider (84) and (85): All we are doing is rescaling the gradient by the weight. Thus, if the variable is weighted more, the gradient will decrease slowly, and the algorithm will have to increase the β estimates by a higher factor to acceptably decrease the gradient (for larger parameters). In this way, the weighted lasso penalty overcomes the requirement of unbiasness for variables that have larger true magnitudes.

- ***'Fast Sparse Classification and Regression'*:** In a recent paper, Friedman [23] proposed an optimization technique for solving regression and classification problems based on various penalties but with emphasis on $J(\beta) = \mathrm{L}^p, p \leq 1$.

Consider again the loss + penalized formulation that we discussed in Section 2 and reproduced below.

$$\min \quad L(X, y, \beta) + \lambda J(\beta) \tag{86}$$

He proposes a path (gradient) following algorithm that calculates the gradient λ_j, where j stands for iteration steps, as follows:

$$g(\beta_j) = -\frac{\partial L}{\partial \beta} \quad \& \quad p(\beta_j) = \frac{\partial J}{\partial |\beta|} \quad \& \quad \lambda(\beta_j) = \frac{g(\beta_j)}{p(\beta_j)} = -\frac{\partial L/\partial \beta}{\partial J/\partial |\beta|} \tag{87}$$

His path following algorithm takes the same types of steps like the piecewise path algorithm except that there is no nice closed form solution for the next feature or point to be added/deleted. The above formulation are much intuitive if thought in relation to the L^1 penalty and for which we give an alternate explanation.

Assume $J(\beta) = ||\beta||_p^p$ in (86) and introduce $\alpha_j = |\beta|_j^p, j = 1..D$ for D features. Thus $J(\beta) = ||\beta||_p^p = ||\alpha||_1$.

Thus (86) turns into (88).

$$\min \quad L(X, y, \beta) + \lambda||\alpha||_1, \text{ with } \alpha_j = |\beta_j|^p \tag{88}$$

$$\left|\frac{\partial L(X, y, \alpha)}{\partial \alpha}\right| = \lambda \Rightarrow \left|\frac{\partial \beta}{\partial \alpha}\frac{\partial L(X, y, \alpha)}{\partial \beta}\right| = \lambda \Rightarrow \left|\frac{\partial L(X, y, \beta)/\partial \beta}{\partial \alpha/\partial \beta}\right| = \lambda \Rightarrow \left|\frac{\partial L(X, y, \beta)/\partial \beta}{\partial J(\beta)/\partial \beta}\right| = \lambda \tag{89}$$

$$\text{For L}^1 \text{ penalty, } \left|\frac{\partial L(X, y, \beta)/\partial \beta}{\partial J(\beta)/\partial \beta}\right| = \left|\frac{\partial L(X, y, \beta)/\partial \beta}{1}\right| = \lambda \text{ (Equivalent to (59))} \tag{90}$$

$$\text{For L}^p \text{ penalty, } \left|\frac{\partial L(X, y, \beta)/\partial \beta}{\partial J(\beta)/\partial \beta}\right| = \left|\frac{\partial L(X, y, \beta)/\partial \beta}{p\beta^{p-1}}\right| = \lambda \tag{91}$$

$$\text{For L}^p, p > 1, \left|\frac{\partial L(X, y, \beta)/\partial \beta}{p\beta^{p-1}}\right| = \lambda \quad \text{and for L}^p, p < 1, \left|\frac{\beta^{1-p}}{p}\right| |\partial L(X, y, \beta)/\partial \beta| = \lambda \tag{92}$$

As we discussed earlier that for many loss functions, (88) can be solved via the active set algorithm (which was also discussed in detail in this paper). The equation that worked for the active set algorithm in (59) is also valid here and shown in (89). One can see an uncanny similarity between (87) and (89), which means that through a transformation of variables we can easily use a L^1 penalty based algorithm to solve for L$^p, p < 1$ penalty, with the major exception that all the closed form solutions for distance till the next point/feature (which we used for the piecewise paths) become invalid. An additional problem is that L$^p, p < 1$ have derivatives that are discontinuous; Friedman [23] tackles discontinuity with the elastic penalty. We will not go into the details and the reader is encouraged to refer to Friedman [23].

Also do note, from (90) and (91), that all we are doing is rescaling the gradient based on the current feature magnitude. If we are working with L$^p, p < 1$ penalty, the gradient decrements at a faster rate if the magnitude of the corresponding feature is higher, thus leading to an even greater magnitude for the feature. This is the reason for increasing sparsity as we approach L^0 penalty. On the other hand, L$^p, p < 1$ penalty will cause the gradient to decreases at a lower rate if the magnitude of the corresponding feature is higher and that accounts for the 'shrinkage' property that was discussed in details with the ridge penalty (and which exists for L$^p, p > 1$ penalties in general). L^1 penalty is a special case as we do not have a scaling factor of β in the denominator. This is also the reason for the existence of linear piecewise paths when certain loss functions are used.

Table 1: Complexity Table for the Active Set algorithm

	Additions	Deletions	Knot points	Overall
$N >> D$	$O(D * DN)$	$O(D)$	$O(N)$	$O(D^2N + D + N) \approx O(D^2N)$
$D >> N$	$O(N * DN)$	$O(N)$	$O(N)$	$O(N^2D + D + N) \approx O(N^2D)$

Next, we will discuss the complexity order for calculating the piecewise paths.

3.6 Complexity Order of Calculations in Piecewise Paths

Consider the complexity of each step: The number of possible variable additions in Step 2a is directly related to 'D' or 'N', whichever is less, and there will be 'D' comparisons at every iterations. In each of these comparison there will be 'N' points accessed. The number of possible deletions in Step 2b is also directly related to 'D' or 'N' whichever is less. And for Step 2c, the number of knots will be totally dependent on number of points; as whenever a point is encountered, ones needs to recalculate the gradients.

As seen in Table 1, the algorithms will depend to a linear level on $maximum(N, D)$ and to a quadratic level to $minimum(N, D)$. For medium sized datasets, which are either large in N or D, this algorithm will give a fast way to measure the entire Regularization path. For small sized datasets, this algorithm is a no-brainer. For datasets which are large in N and D (large≈tens of thousands), there is still the question on whether the calculation of the entire regularization path being possible or not in a reasonable time-frame (which we will later show is solvable in a reasonable time). There are two main calculations involved, step size and the calculation of the inverse of gradient^2($[\nabla^2 L]^{-1}$). Step size can easily become the greater calculation as one can update or downdate the inverse using the Sherman-Morrison update.

3.7 So What Really happens in 1-norm based SVM problems?

We solve for the regularization parameter from $\lambda = \infty$ to $\lambda = 0$, thus correspondingly C changes from 0 to ∞ in (40). There is an infinite width margin at the start (margin width $= 2/||\beta||$) at $\lambda = \infty$ and all examples as support vector's (SV). As we slowly decrease λ (increasing C), the margin starts to shrink (but only for those features that are in the model) and also some SV's are removed from the model. Thus, sparsity of features decreases and sparsity of SV's increases as we continue onto the regularization path. At $\lambda = 0$, almost all features will be affected by the shrinkage and their corresponding margins will be ultra thin (assuming β's has increased), and the most SV's in this case will lie outside the margin (if it's a separable case). Something that needs to be considered for 1-norm based SVM problems is the fact that the sparsity exists both in the loss function and the penalty, and it would be interesting to find the relation between both the sparsity types.

4 Results & Conclusions

In this section we will discuss some of the results to exhibit that the algorithms can scale to large datasets reasonably. Our goal is to first show that the piecewise algorithms are reasonably fast and to that end we compare results from contemporary implementations. Next, we will show how we can make our algorithms even faster by using non-traditional hardware and show results on matrix-matrix multiplication and kernel evaluations.

4.1 Traditional Hardware

In this section we discuss results for traditional hardware. We term our general PC based architecture as 'traditional hardware' as we are well versed on using it for all our general computing needs. We will later discuss non-traditional hardware, which are being used currently to create supercomputers with almost a magnitude more power than current traditional hardware. The hardware used in our case are the x86 architecture based Core 2 Duo, from Intel, with each processor having 2 individual cores. Each core can perform 4 floating point calculation per clock cycle if they are packed as vector instructions. The also sport highly efficient subsystems for handling instruction branching, and prefetching, among others.

We use Matlab for all our experiments. We modified all our code so that the major part of the algorithm is decomposed to matrix-matrix and matrix-vector operations . The reason of doing so is to allow Matlab to call on optimized matrix-matrix and matrix-vector code directly via Basic Linear Algebra Subroutines (BLAS) and Automatically Tuned Linear Algebra Subroutines (ATLAS) libraries. We will go into details of these libraries more in the next subsection when we discuss comparative libraries for non-traditional hardware.

4.1.1 Classification Results

We will restrict ourselves to linear classification algorithms as they usually take more time than linear regression algorithms. We will first compare the Hinge^2 loss based L^1 penalized algorithm (represented with SVM^2) with the results of the usual SVM algorithm with L^2 penalty. We use the libsvm [9] software to get the results for the SVM algorithm with L^2 penalty. Note that we tried using Hinge loss based L^1 penalized algorithm, but testing datasets with more than 100 features was impractical.

Table 2: Comparison of the SVM^2 algorithm to SVM in terms of Accuracy and Timings

Training Size	SVM^2 Accuracy in % (# of features)	libsvm Accuracy in % (# of features)	Time in Secs (svm^2/libsvm)
N=10, D=10	72 (3.51)	69 (10)	(0.009/0.012)
N=10, D=100	63 (4.84)	56 (100)	(0.015/0.043)
N=100, D=10	84.6(3.69)	82.6 (10)	(0.066/0.039)
N=100, D=100	84 (4)	71.4 (100)	(0.13/0.22)
N=100, D=1K	83.6 (3.89)	60.6 (1000)	(0.20/2.64)
N=100, D=10K	83.3 (6.67)	52.9 (10K)	(1.15/39.62)
N=1K, D=10	84.8 (3.67)	84.9 (10)	(0.39/0.66)
N=1K, D=100	85.2 (12.89)	81 (100)	(1.16/21.57)
N=1K, D=1K	85.5 (19.33)	71.2 (1K)	(4.11/39.14)
N=1K, D=10K	84.7 (47.75)	59.4 (10K)	(28.45/560.26)
N=10K, D=10	85.4 (4.25)	85.4 (10)	(14.46/40.45)
N=10K, D=100	85 (34.75)	85 (100)	(58.17/639.54)
N=10K, D=10K	85.8 (21.25)	-*	(255.63/-*)

* - Did not finish in a reasonable time

In Table 2, we show the results for a synthetic dataset of different sizes. We depict the number of training examples as N and the number of dimensions/features as D. We additionally had 1000 validation and 1000 test examples. We create multiple models from the training examples and find the best model on the validation set. We then report the results in the table for 100 randomized experiments for smaller datasets <1K training examples & 1K features and 10 randomized experiments for the rest. The synthetic dataset had 2 relevant dimensions and the rest are noisy dimensions, all of which have a correlation of 0.3 with the two relevant dimensions. Class labels were assigned as: if feature-1 > feature-2, class label ={+1} else class label ={-1}. Additionally, noise is added to the features so as there is no total separation between the classes.

We create 5 models for libsvm each corresponding to different cost (C) values C={0.1, 0.5, 1, 2, 10}. We show the (average) number of features in brackets along with the accuracy figure (which ranges between 0-100%). Note that the number of features for the model chosen for SVM^2 algorithm always remains nominal whereas libsvm chooses all the features. The SVM^2 algorithm is also not affected that much on increasing the number of examples or dimensions which is not the case with libsvm, which suffers from decreasing accuracy when the number of features increase. Also note that the execution time ramps up slower for SVM^2 compared to libsvm when increasing either N or D.

In Table 3, we also show results for some standard datasets for which 100 randomized experiments were done with 5-fold cross validation. We set C={0.1, 0.5, 1, 2, 10} for libsvm and the remaining algorithms used the entire regularization path. These datasets are small both in dimensions and number of examples.

Table 3: Comparison of various variants of the Hinge loss

Dataset	Hinge loss+L^1 [from Ji Zhu] % Accuracy(# of features)	Hinge^2 loss+L^1 % Accuracy(# of features)	Hinge loss+L^2 [via libsvm] % Accuracy(# of features)
Two norm (N= 300, D=20)	96.71 (18.78)	96.81 (19.02)	96.98 (20)
Breast Cancer Wisconsin* (N= 683, D=10)	92.80 (4.29)	95.66 (7.96)	96.68 (10)
Breast Cancer (Diagnostic) (N= 569, D=30)	96.63 (12.01)	96.46 (10.23)	97.51 (30)
Pima Indian Diabetes (N=768, D=8)	75.80 (5.76)	75.82 (5.67)	76.34 (8)
Ionosphere* (N=351, D=34)	78.53 (3.56)	87.32 (14.46)	87.61 (33)
Sonar* (N=208, D=60)	74.84 (26.17)	74.24 (19.48)	76.81 (60)
SPAM$^+$ (N=3065, D=57)	78.10 (7.78)	92.24 (52.67)	92.89 (57)

N= # of examples, D= # of features.
$^+$ SPAM had a separate test set of 1536.
* from libsvm site, rest from UCI Repository [3].

Our experimental setup for each randomized experiment was as follows – we divided the total data into 5 folds and then used 4 folds for training and validation and the remaining 1 fold for held out testing. We found the best (and simplest) performing model parameter for the training by using the validation set. We then used both the training and validation set to create the model with the best performing parameter and then reported the accuracy on the test set.

As can be seen that the accuracy of Hinge loss+L^2 (SVM+L^2) is higher compared to the L^1 penalty variants, though the Hinge^2 variant loses by a small margin. We can not explain the drop in accuracy for the 1-norm SVM (Hinge loss+L^1 or SVM+L^1) by Zhu *et al.* [46] for the Breast Cancer dataset and Ionosphere dataset. In general the behavior of the 1-norm SVM is to be most frugal when it comes to the number of features which also results in a larger drop in accuracy. Do note the competitiveness of Hinge^2 loss with the other algorithms, and if the earlier results on noise/speed are taken into account, than it is a strong contender for consideration for real world applications.

An argument that has come up many times in literature is '*when are L^1 methods better than L^2 methods?*' A related question is '*how much do we lose if we do not include a feature?*' We can limitedly infer through the above discussions that when there is noise or when there are many features then L^1 penalized methods are better suited to handle the task. We can speculate, for small datasets, that having as much features as possible in the model do seem to help the accuracy. But for large datasets with noisy features, selecting relevant features makes the final model less susceptible to noise compared to a model with all features.

4.2 Non-traditional Hardware

In recent years, there has been an explosive growth in terms of non-traditional hardware architectures. These architectures are termed non-traditional as they have been designed for specific tasks like signal processing or vector processing or other uses. Such architectures have slowly become ubiquitous and have permeated into common day usage. Some popular examples include consoles like the Playstation 3 (PS3) and the graphic cards (termed as Graphics Processing Units (GPU's)) residing in many personal computers.

The authors, in an earlier work [31], showed a speedup of 4 times (compared to a reasonable dual core PC) on the Latent Dirichlet Allocation (LDA) algorithm by using the PS3 console. The PS3 console sports a 7 core CPU, which has 6 dedicated digital signal processing cores and an additional core for scheduling, each of which can execute up-to 4 floating point operation per cycle and run at 3.2 GHz. Each of these core can be thought to be equivalent in processing power to a single core of a Core 2 Duo machine. This led to further study with other competitive architectures including some contemporary graphics cards from companies like ATI and AMD. We will now briefly discuss the advantages of GPU's vs. CPU's on typical workload that is expected with machine learning architecture.

We use the ATI Radeon 4830, a graphics card from AMD-ATI worth 100 USD, for experimentation. This graphics card has 640 stream processors each working at 575 MHz and has 512 MB of on-board RAM. These stream processors are very simple ALU processors, good at crunching data but bad at executing branches ($if - else$). These processors are organized as groups of processors each running some thousands of threads which are interleaved so as to hide memory and ALU latencies within the threads. For more information refer to AMD-ATI guide [2].

The general flow of programming for such GPU's is as follows and shown in Figure 9:

Step 1 copy the required datasets onto the on-board memory of the GPU.

Step 2 generate threads, utilizing the stream processors, that does required operations on the dataset.

Step 3a, 3b copy back the results to the CPU memory.

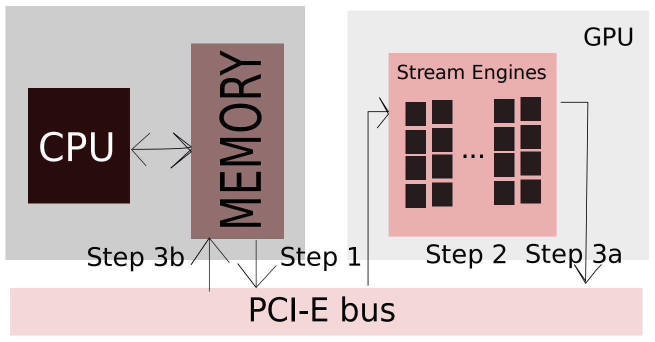


Figure 9: CPU-GPU data transfer.

We can note that there are two bottlenecks involved: copying the data to and fro between the GPU and CPU and performing ALU operations. Typical GPU's today have either DDR3 or DDR5 memory[30] which supports 60GB/s and 115 GB/s data transfer rate respectively. On the other hand the number of floating point operations that can be done on the GPU can be calculated as follows:

Theoretical Peak performance for ALU operations[31]:

$$\begin{aligned}
\text{GPU flops (Single Precision)} &= 2 \text{ x } (\text{\# of stream processors}) \text{ x } (\text{frequency of processor}) \\
&= 2 \text{ x } 640 \text{ x } 575\text{Mhz} = 736 \text{ Gflops (Single Precision)} \quad (93) \\
\text{GPU flops (Double Precision)} &= 2 \text{ x } (\text{\# of stream processors})/5 \text{ x } (\text{frequency of processor}) \\
&= 2 \text{ x } 128 \text{ x } 575\text{Mhz} = 147 \text{ Gflops (Double Precision)} \quad (94) \\
\text{CPU flops (Single Precision)} &= 2 \text{ x } (\text{\# of vector ops. per cycle}) \text{ x } (\text{frequency of processor}) \\
&= 2 \text{ x } 4 \text{ x } 2.4\text{GHz} \approx 20 \text{ Gflops (Single Precision)} \quad (95) \\
\text{CPU flops (Single Precision)} &= 2 \text{ x } (\text{\# of vector ops. per cycle}) \text{ x } (\text{frequency of processor}) \\
&= 2 \text{ x } 1 \text{ x } 2.4\text{GHz} \approx 5 \text{ Gflops (Double Precision)} \quad (96)
\end{aligned}$$

Note that we talk about single precision and double precision floating numbers, which are usually 32-bit and 64-bit respectively. Single precision floats are sometimes not suited where accuracy is of high concern. From the above derivations we can see that there is a magnitude difference between the power of a GPU compared to a single processor of a normal 2.4GHz Core 2 Duo. In practice, such performance cannot be extracted for all algorithms due to the copying of data involved. Also consider that we have a 60GB/s or 115GB/s memory on-board of the GPU that is being fed via at 250MB/s or 500MB/s[32] link from the CPU. Thus if the GPU is data-starved, then there will be no measurable performance gain. To achieve the theoretical peak one will need to perform at least $\frac{736 \text{ Gflops}}{1/4 * 60 \text{ GB/s}} \approx 48$ operations per byte. Thus if there is trivial amount of computation being performed on the data, one will not see any performance gains.

[30]DDR=Double Data Rate memory
[31]we have a factor of 2 below as we have as input - 2 floats and as output - 1 float
[32]via PCI-E bus.

The graphics card used in our experiments, ATI Radeon 4830, is rated to work at 732 Gflops (93) for single precision and 147 Gflops (94) for double precision calculations, which include additions and multiplications. Transcendental functions like *sin, cos, log,* and *exp*, among others will take more than a single cycle to be calculated. The card doesn't fully support the IEEE 754 floating point standard but comes close to it. The card has $1/5^{th}$ of the rated flops when double precision are concerned[33]. On the other hand, a single core of a Core 2 Duo, from Intel, can perform 4 floating point operations per cycle via vector operations done through SSE[34] instructions and whose processing power is noted in (95) and (96). Note that there is more than a magnitude difference between the processing power involved.

Before going on further we will like to mention that many processor companies have developed linear algebra subroutines. These subroutines, called Basic Linear Algebra Subroutine (BLAS) and Automatically Tuned Linear Algebra Subroutine (ATLAS), are tuned (sometimes via hand coding in assembly) for specific processors. Matlab uses such subroutines, via the Math Kernel Library (MKL) by Intel and AMD core Math Library (ACML) by AMD[35], for matrix operations. The speedup is almost linear with increasing the number of processor and many a times one can reach 80% of the theoretical Gflops.

The routines that we are most interested are for matrix-matrix and matrix-vector multiplication. They are exposed via GEMM() and GEMV() which stand for General Matrix Matrix and General Matrix Vector Multiply respectively. Most BLAS libraries expose these subroutines: via *sgemm*() and *dgemm*() for single and double precision matrix-matrix multiplication and via *sgemv*() and *dgemv*() for single and double precision matrix-vector multiplication.

It is comparatively cheaper to calculate matrix-vector than the matrix-matrix operation ($O(mn)$ Vs. $O(mnk)$ for multiplying a m x n matrix with a n x 1 or n x k matrix respectively) and thus we have not considered in this study. In Figure 10, we show comparative Gflops for matrix-matrix operations for CPU vs. GPU for both double and single precision operations. On the X-axis we plot for various matrix sizes[36] and on the Y-axis we plot the Gflops obtained. For these experimentation we used a 2.4 GHz dual core machine (Core 2 Duo 2.4 GHz, 4GB RAM). We can see that for these dense matrix operations a GPU consistently performs 10 times better than a dual core machine.

In a set of related experiment on how GPU's can be used in a machine learning context we considered a generic kernel based model as shown in (97) which can be evaluated for different kernels, shown in (98).

[33]Matlab uses double precision for all calculations unless specified.
[34]SSE=Streaming SIMD (Single Instruction Multiple Data) Extensions
[35]Depending on the processor type. Also, Matlab still doesn't have inbuilt support for GPU's.
[36]we calculate multiplication of an n x n matrix with another n x n matrix.

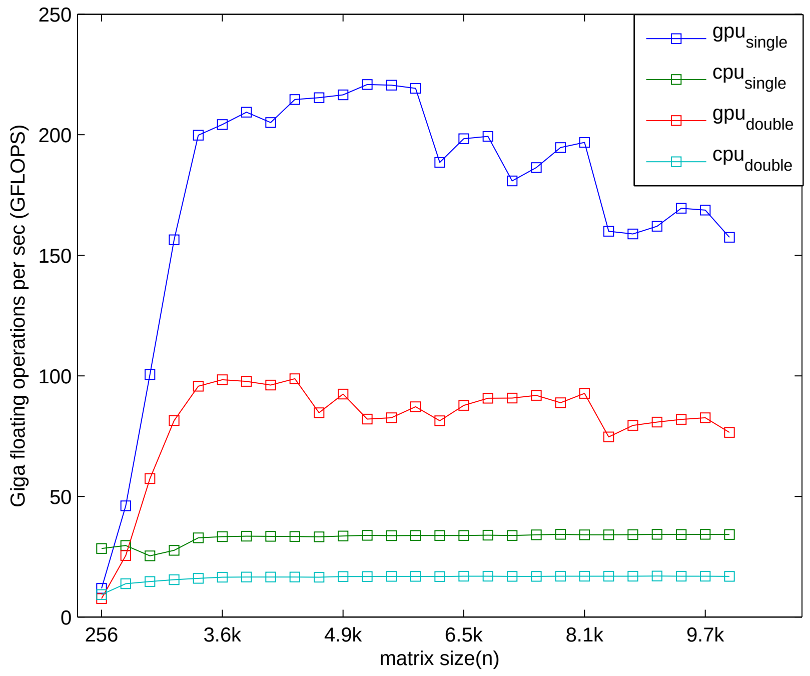


Figure 10: Single (sgemm()) & Double (dgemm()) precision comparison for CPU (depicted by CPU$_{\text{single}}$ & CPU$_{\text{double}}$ respectively) and GPU on Matrix-Matrix Multiplication (depicted by GPU$_{\text{single}}$ & GPU$_{\text{double}}$ respectively) via BLAS.

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + \beta_0 \tag{97}$$

$$\text{Kernels} \begin{cases} \text{Linear:} & K(u,v) = u^T v \\ \text{Polynomial:} & K(u,v) = (1 + u^T v)^D, D = \text{ degree} \\ \text{Radial Basis:} & K(u,v) = exp(-||u - v||^2) \\ \text{Sigmoid:} & K(u,v) = tanh(1 + \gamma u^T v) \end{cases} \tag{98}$$

Non-linear kernels involve transcendental functions which are expensive to calculate. Consider that the ATI Radeon 4830 has 128 stream processors each of which can compute a transcendental function like *sin*, *cos*, *tanh*, *exp*, and so on. We earlier mentioned that this card has 640 stream processors out of which 128 can additionally do transcendental operations whereas the remaining 512 can only do simple ALU operations.

For our experiment we use the kernel based SVM (via libsvm [9] software) to create kernel models and then evaluate it on the CPU and GPU. We also modified the testing code of libsvm [9] to take into account multi-core CPU's. We also implemented the same testing code for the ATI Radeon 4830 GPU via the Brook+ API [2] for C/C++.

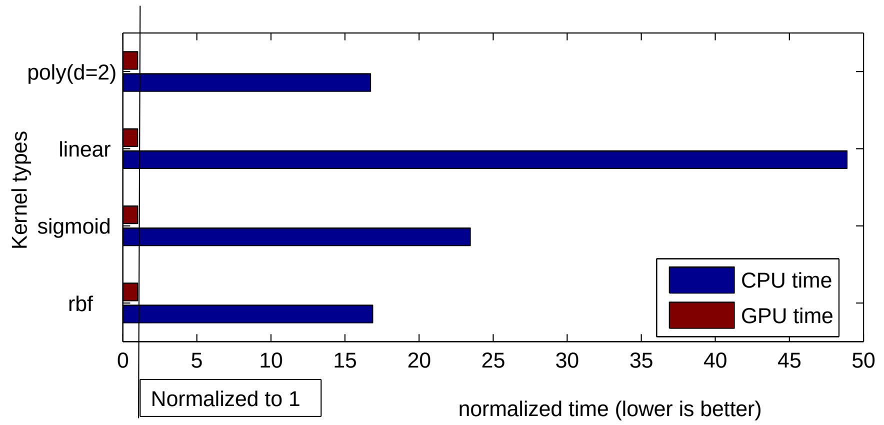


Figure 11: Normalized kernel evaluation time.

In Figure 11, we show timings for evaluating different kernels on the GPU and the CPU. We use the $ijcnn1$ classification dataset from the libsvm site and evaluate for 8000 test examples with about 3000 or so training support vectors. So, for each test example one needs to evaluate kernels for all the 3000 or so support vectors. It was reported that, for the said dataset, the kernel evaluation takes 80% of the training time [10]. We can observe, from the figure, that there is a magnitude difference in the kernel evaluation time between the CPU and GPU. To give a rough idea of timings, the GPU time was under 0.3 secs for all the cases. Due to time constrains we use single precision calculations and the simplest kernel code (no optimization done). Further optimization of the GPU code can lead to an even larger speedup and is a future goal of the authors.

Note that in the current form our CPU code for SVM^2 (Hinge^2 loss + L^1 penalty) uses a large amount of matrix-vector operations that are fast enough on a normal CPU. Our results show that we can achieve 15-20% speed improvement on using the GPU based matrix-vector operations. There is much promise in using GPUs for kernel evaluation as we can easily get more than a magnitude increase in speed. Similar results for kernel evaluations are reported by Catanzaro *et al.* [8].

4.3 Conclusions

Solving L^1 penalty can be tough combinatorially and the piecewise path algorithm is a computationally cheap way to get multiple models. Though not perfect in all senses (asymptotically), L^1 penalty can be beefed up and can be modified to behave as an oracle property. We have not discussed why one should use this, compared to using a fixed value of λ. The advantages are straightforward: In a normal constrained optimization setting, we will basically start from a feasible solution and move towards the minimum, which is also done in piecewise paths (starting from (0,...,0) albeit with a little more work by making sure we do not violate the KKT conditions[37]). Thus, with a little more effort instead of getting 'a' solution, we get all possible solutions. Contemporary SVM implementations as such do not have a fixed convergence time when 1000's of examples are concerned. Though the piecewise implementation is more dependable in that respect as the amount of work can be roughly estimated. One could always stop any time and still have part of the regularization path, which can always serve as a seed for future runs. Thus, the probability of getting some useful models is always there, which is not the case with the contemporary SVM implementations that solve for fixed λ solutions. And as always, the idea of getting a bunch of possible useful models seems more promising than getting a single model.

We also seen that the advent of faster hardware can bring out a change in the way we view and act on large datasets. Consider that a 100 USD graphics card has the processing power of a (scaled) 20+ core processor. Such a powerful machine, the latter one, might not be in the reach of the researcher but the former can be easily bought off the shelf. This totally changes the scenario as we can reasonably tackle problems of the size of 100K+ examples and 100K features on a desktop machine (offsetted by enough memory of-course) and not be concerned about obtaining a cluster of machines.

We hope to have reasonably demonstrated to the reader that nowadays we have enough compute power and mathematical formulations to tackle large sized linear models. The advantage of the homotopy based approach is the ability to generate entire paths in a reasonable amount of time. Also, today's state of the art computer hardware allows us to tackle problems on a desktop machine which were earlier in the realms of clusters of machines working in tandem.

Appendix

Proof of why Piecewise and LARS step size are equivalent (The matrix math below has been intentionally butchered to keep stuff simple.)

[37]State of the Art optimization methods (like interior point) use a bunch of additional assumptions to quickly move towards the optimum.

LARS: (from page 7 in [19]) update correlation $\mu_{t+1} = \mu_t + d_l U_A$ $(\Rightarrow X\beta_{t+1} = X\beta_t + d_l U_A)$

$$\text{where } \mu_t = X\beta_t, \quad \text{step size } d_l = \frac{C \pm c_j}{A_A \pm A_j}, \quad U_A = X_A W_A \tag{99}$$

$$\text{Additionally } G_A = X_A^T X_A, \ \ A_A = (G_A^{-1})^{-1/2} = ||X_A^T X_A||, \ \ W_A = A_A G_A^{-1} = \frac{||X_A^T X_A||}{X_A^T X_A} \tag{100}$$

$$\Rightarrow X_A^T U_A = A_A, \ \ a_j = X_j^T U_A$$

$$\text{For piecewise, } \beta_t = \beta_t + d_p \gamma_A, \text{ where } d_p = \frac{C \pm c_j}{(X_\gamma)^T (X_A \pm X_j)}, \ \ \gamma_A = [\nabla^2 L]^{-1} = \frac{1}{X_A^T X_A} \tag{101}$$

$$\text{To prove equivalence } d_l U_A = d_p X \gamma_A \Rightarrow d_l X_A W_A = d_p X_A \gamma_A \Rightarrow d_l W_A = d_p \gamma_A \Rightarrow d_l ||X_A^T X_A|| = d_p \tag{102}$$

Consider the **denominator** from (101): $(X_\gamma)^T (X_A \pm X_j) = (X_A \gamma_A)^T (X_A \pm X_j) = \gamma_A^T X_A^T X_A \pm \gamma_A^T X_A^T X_j$

$$= \frac{X_A^T X_A}{X_A^T X_A} \pm \frac{X_A^T X_j}{X_A^T X_A} = \left\{ \frac{X_A^T X_A ||X_A^T X_A||}{X_A^T X_A} \pm \frac{X_A^T X_j ||X_A^T X_A||}{X_A^T X_A} \right\} \frac{1}{||X_A^T X_A||} \tag{103}$$

$$= \left\{ \frac{X_A^T X_A A_A}{G_A} \pm \frac{X_A^T X_j A_A}{G_A} \right\} \frac{1}{||X_A^T X_A||} = (X_A^T X_A W_A \pm X_J^T X_A W_A) ||X_A^T X_A||^{-1} \tag{104}$$

$$= (X_A^T U_A \pm X_J^T U_A) ||X_A^T X_A||^{-1} = (A_A \pm a_j) ||X_A^T X_A||^{-1} \tag{105}$$

$$\text{On substituting above denominator in (101), } d_p = \frac{C \pm c_j}{(A_A \pm a_j)} ||X_A^T X_A|| = d_l ||X_A^T X_A|| \qquad \text{(Q.E.D)}$$

Note: In above we can go from $X\gamma$ to $X_A \gamma_A$ as only for $j \in A, \gamma_A \neq 0$. Also note that the main difference arising here is due to LARS always making sure that the direction is a unit vector whereas piecewise based algorithm does not enforce the condition.

References

[1] A. E. Albert. *Regression and the Moore-Penrose pseudoinverse.* Academic Press, New York,, 1972.

[2] AMD. Stream computing guide. 2008. `http://ati.amd.com/technology/streamcomputing/Stream_Computing_User_Guide.pdf`.

[3] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[4] S. P. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, Cambridge, UK; New York, 2004.

[5] T. Brants and A. Franz. Web 1t 5-gram version 1. Technical report, september 2006.

[6] A. Brondsted. *An Introduction to Convex Polytopes.* Springer-Verlag, 1983.

[7] E. Candes and T. Tao. The dantzig selector: statistical estimation when p is much larger than n, 2005.

[8] B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 104–111, New York, NY, USA, 2008. ACM.

[9] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] C.-C. Chang and C.-J. Lin. *LIBSVM FAQ*, 2001. FAQ at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html#f432`.

[11] S. Chen and D. Donoho. Basis pursuit. Technical report, 1995.

[12] R. Davidson and J. G. MacKinnon. *Estimation and Inference in Econometrics.* Oxford University Press, 1993.

[13] M. de Kunder. Technical report, 2008. `http://www.worldwidewebsize.com/`.

[14] D. L. Donoho. Compressed sensing. *Stanford University - Technical Report*, 2004.

[15] D. L. Donoho. Thresholds for the recovery of sparse solutions via l1 minimization. In *In Proc. Conf. on Inofrmation Sciences and Systems*, 2006.

[16] D. L. Donoho and M. Elad. Maximal sparsity representation via l1 minimization. In *the Proc. Nat. Aca. Sci. 100*, pages 2197–2202, 2003.

[17] D. L. Donoho and J. Tanner. Neighborliness of randomly-projected simplices in high dimensions. In *Proc. National Academy of Sciences*, pages 9452–9457, 2005.

[18] D. L. Donoho and J. Tanner. Sparse nonnegative solutions of underdetermined linear equations by linear programming. In *Proceedings of the National Academy of Sciences*, pages 9446–9451, 2005.

[19] B. Efron, T. Hastie, L. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

[20] B. Efron, T. Hastie, and R. Tibshirani. Discussion of 'the dantzig selector', 2007.

[21] J. Fan and R. Li. Variable selection via penalized likelihood. *Journal of American Statistical Association*, 96:1348–1360, 2001.

[22] I. E. Frank and J. H. Friedman. A statistical view of some chemometrics regression tools. In *Technometrics*, volume 35(2), pages 109–135, may 1993.

[23] J. Friedman. Fast sparse regression and classification. 2008.

[24] W. Fu. Penalized regressions: the bridge vs the lasso. *JCGS*, 7(3):397–416, 1998.

[25] W. Fu and K. Knight. Asymptotics for lasso-type estimators. In *Ann. Statist.*, volume 28(5), pages 1356–1378, 2000.

[26] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, 286(5439):531–537, 1999.

[27] B. Grunbaum. *Convex polytopes.* John Wiley & Sons, 1967.

[28] T. Hastie, J. Taylor, R. Tibshirani, and G. Walther. Forward stagewise regression and the monotone lasso. *Electronic Journal of Statistics*, 1:2007, 2007.

[29] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2001.

[30] A. E. Hoerl. Application of ridge analysis to regression problems. *Chemical Engineering Progress*, 58:54–59, 1962.

[31] A. Jaiantilal. Achieving speedup on Latent Dirichlet Allocation through the Cell B.E. architecture. *Available through request*, 2007.

[32] E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26:394–395, 1920.

[33] Y. Nesterov. *Introductory Lectures on Convex Optimization.* Springer-Verlag, 2004.

[34] J. Nodecal and S. J. Wright. *Numerical Optimization.* Springer, 1999.

[35] M. Osborne, B. Presnell, and B. Turlach. A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis*, 20(3):389–403, 2000.

[36] R. Penrose. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.

[37] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, 1997.

[38] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5:941–973, 2004.

[39] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

[40] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of Royal Statistical Society*, 67(1):91–108, 2005B.

[41] A. Tikhonov. On the stability of inverse problems. *Dokl. Akad. Nauk SSSR*, 39(5):195–198, 1943.

[42] A. M. Vershik and P. V. Sporyshev. Asymptotic behavior of the number of faces of random polyhedra and the neighborliness problem. In *Selecta Math. Soviet.*, volume 11(2), pages 181–201, 1992.

[43] L. Wang, J. Zhu, and H. Zou. The doubly regularized support vector machine. In *Statistica Sinica*, volume 16, pages 589–615, 2006.

[44] S. Weisberg. *Applied Linear Regression.* Wiley, 2005.

[45] D. W.Marquardt and R. D.Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, feb 1975.

[46] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. Technical report, Stanford University, 2003.

[47] G. M. Ziegler. *Lectures on Polytopes.* Springer, 1995.

[48] H. Zou. An improved 1-norm svm for simultaneous classification and variable selection. *AISTATS*, 2:675–681, 2007.

[49] H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101:1418–1429(12), December 2006.

[50] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, 67:301–320, 2005.

[51] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.