Machine Learning and Optimization

Andres Munoz

Courant Institute of Mathematical Sciences, New York, NY.

Abstract. This final project attempts to show the differences of machine learning and optimization. In particular while optimization is concerned with exact solutions machine learning is concerned with generalization abilities of learners. We present examples in the areas of classification and regression where this difference is easy to observe as well as theoretical reasons of why this two areas are different even when they seem similar at a first glance.

1 Introduction

Since the invention of computers people have been trying to answer the question of whether a computer can 'learn', and before we start talking about theorems and algorithms we should define what 'learning' means for a machine. Arthur Samuel in 1959 defined machine learning as the "field of study that gives computers the ability to learn without being explicitly programmed". At the beginning this field was mainly algorithmic and without much foundations, it was probably thanks to the work of Valiant [6] who introduced the framework of PAC learning (probably approximately correct learning) that a real theory of the learnable was established. Another milestone in the theory of learning was set by Vapnik in [7]. Vapnik casts the problem of 'learning' as an optimization problem allowing people to use all of the theory of optimization that was already given. Nowadays machine learning is a combination of several disciplines such as statistics, information theory, theory of algorithms, probability and functional analysis. But as we will see optimization is still at the heart of all modern machine learning problems. The layout of the paper is as follows. First we present the definitions and notation needed, then we give a mathematical definition of learning and we give the general optimization problem. In section 3 we present learning guarantees and describe the principle of empirical risk minimization. In the end we present experiments comparing the raw use of optimization versus a combination of optimization and learning bounds.

2 Definitions

We will consider the following spaces: a space of examples $\mathcal{X}$ a space of labels $\mathcal{Y}$ and a space of hypothesis $\mathcal{H}$ that contains functions mapping $\mathcal{X}$ to $\mathcal{Y}$. We will also consider a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and a probability measure P over the space $\mathcal{X} \times \mathcal{Y}$.

Definition 1. *For any hypothesis $h \in \mathcal{H}$ we define the risk of h with respect to L and P to be:*

$$\mathcal{L}_P(h) = \mathop{\mathrm{E}}_{P}(L(h(x), y))$$

The general problem of machine learning can be then cast as finding the hypothesis $h \in \mathcal{H}$ that solves the following optimization problem:

$$\min_{h \in \mathcal{H}} \mathcal{L}_P(h)$$

By having access only to a labeled sample $S = (x_i, y_i)_{i=1}^n \in \mathcal{X} \times \mathcal{Y}$. A simplification of this scenario is given when the distribution P is given only over the example space $\mathcal{X}$ and there exists a labeling function $f : \mathcal{X} \to \mathcal{Y}$ in that case we are trying to find a hypothesis h such that it solves the optimization problem:

$$\min_{h \in \mathcal{H}} \mathop{\mathrm{E}}_{P}(L(h(x), f(x))$$

Example 1. Consider $\mathcal{X} = \mathbb{R}^2$ and $Y = \{-1, 1\}$, suppose there's a probability distribution P_x over $\mathcal{X}$ and that the labeling function f is an indicator function for a rectangle $R \subset \mathbb{R}^2$,i.e:

$$f(x) = \begin{cases} 1 & x \in R \\ -1 & x \notin R \end{cases}$$

If the loss function is the $0-1$ *loss* i.e $L(y, y') = 1$ if $y \neq y'$ and 0 otherwise, then a good hypotheses set would be that of indicator functions of rectangles and the problem becomes that of finding the function that minimizes

$$\mathop{\mathrm{E}}_{P_x}[L(h(x), f(x))] = P_x(h(x) \neq f(x))$$

Of course in the previous example f is a solution to the problem but we clearly don't know f and the only thing we have access to is a labeled sample of points $(x_i, y_i)_{i=1}^n$ which is called the training data. And with it we need to approximate the real solution as much as possible. This introduces the definition of PAC learning [6]:

Definition 2. *We say a problem is agnostically PAC-learnable if given any (ϵ, δ) there exists an algorithm $\mathcal{A}$ such that after seeing a sample of size $n = n(\epsilon, \delta)$ the algorithm returns a hypothesis h such that*

$$\mathcal{L}_P(h) - \min_{h \in \mathcal{H}} \mathcal{L}_P(h) < \epsilon$$

with probability at least $1 - \delta$ over the sampling process and $n(\epsilon, \delta)$ is polynomial in $(\frac{1}{\epsilon}, \frac{1}{\delta})$.

Example 2. If L is the $0-1$ loss and we know that $\mathcal{H}$ contains a function that has risk 0 then after seeing n examples any hypothesis consistent with the examples verifies that with probability at least $1 - \delta$ [5]:

$$P(h(x) \neq y) < \frac{\log(|\mathcal{H}|/\delta)}{n}$$

So in this case we have a PAC-learnable problem with $n = \frac{1}{\epsilon}\log(|\mathcal{H}|/\delta)$ which is clearly polynomial on $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

The previous example has shown us that if $\mathcal{H}$ is finite and consistent with the examples then the problem is PAC-learnable and not only that but that the risk decreases in the order of $O(\frac{1}{n})$. Nevertheless asking that the hypothesis set is consistent requires normally that we have an infinite hypothesis set making the previous bound vacuous since it depends on $\log(|\mathcal{H}|)$. So can we give a bound of the same kind even if the number of hypotheses we consider is infinite?
In the previous example the algorithm chose a hypothesis that had empirical error equal to 0. The analogous of this algorithm when we don't know if there is a hypothesis consistent with the data is to find a hypothesis that has the minimal error on the training data h_n. We would hope that as the size of the sample increases we would have $h_n \to \operatorname{argmin}_{h \in H} \mathcal{L}_P(h)$ in some sense, and in fact under certain (reasonable) conditions:

Proposition 1. *If a hypothesis class $\mathcal{H}$ has finite VC-dimension [7] then it is true that $\mathcal{L}(h_n) - \min_{h \in \mathcal{H}} \mathcal{L}(h) \to 0$ in probability.*

We wont dwell on the definition and properties of the VC-dimension since that is just tangentially related to this paper. The previous proposition assures us that we can approximate our original problem by simply minimizing:

$$\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

This is known as empirical risk minimization (ERM) and in a sense is the raw optimization part of machine learning, as we will see we will require something more than that.

3 Learning Guarantees

Definition 3. *Given a set of functions $G = \{g : Z \to \mathbb{R}\}$ and a sample $S = (z_i)_{i=1}^{n}$ the empirical Rademacher complexity of G is defined by:*

$$\Re_S(G) = \operatorname*{E}_{\sigma} \left(\frac{1}{n} \sup_{g \in G} \sum_{i=1}^{n} g(z_i)\sigma_i \right)$$

Where σ_i is a uniform random variable taking values in $\{-1, 1\}$. The Rademacher complexity of G is simply the expectation of the empirical Rademacher complexity:

$$\Re_n(G) = \operatorname*{E}_{S} (\Re_S(G))$$

The Rademacher complexity measures how well the set G correlates with noise. In a sense the smaller the rademacher complexity is then the less expressive our function space is. The Rademacher complexity is in general in the order of $O(\sqrt{\frac{d}{n}}$. Where d is the VC-dimension.

Example 3. Let $\mathcal{X} = \{x \in \mathbb{R}^n | \|x\| \leq R\}$. The bounds on the Rademacher complexities of certain hypotheses classes $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{R}\}$ are given below:

$$\begin{aligned}\mathcal{H} = \{w^T x | w \in \mathbb{R}^n\} &\quad \Re_n(\mathcal{H}) \leq \sqrt{\tfrac{d+1}{n}} \\ \mathcal{H} = \{w^T x | \|w\| \leq \Lambda\} &\quad \Re_n(\mathcal{H}) \leq \sqrt{\tfrac{\Lambda R}{n}} \\ \mathcal{H} = \text{Rectangles in } \mathbb{R}^n &\quad \Re_n(\mathcal{H}) \leq \sqrt{\tfrac{2d}{n}}\end{aligned}$$

An interesting fact about the previous example is that the complexity of the second hypotheses set does not depend on the dimension of the space and just depends on the size of the vector w we will use this fact later on to construct learning algorithms.
In what follows we will restrict our attention to two different problems the one of classification which corresponds to the label space $\mathcal{Y} = \{-1, 1\}$ and loss function equal to the $0 - 1$ loss as we've seen before. The other problem is that of regression which corresponds to the label space $\mathcal{Y} = \mathbb{R}$ and loss equal to the square loss $L(y, y') = (y - y')^2$. For those problems we can argue the following:

Proposition 2. *If L is the $0 - 1$ loss function then given a sample $S = (x_i, y_i)_{i=1}^n$, for any hypothesis $h \in \mathcal{H}$ the following is true with probability at least $1 - \delta$ [4]:*

$$\mathcal{L}_P(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i) + \Re_n(\mathcal{H}) + \sqrt{\frac{\log(2/\delta)}{n}} \tag{1}$$

If L is the square loss and $L(h(x), y) \leq M$ for every (x, y) then

$$\mathcal{L}_P(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i) + 4M\Re_n(\mathcal{H}) + M\sqrt{\frac{\log(2/\delta)}{n}} \tag{2}$$

The above bounds express the ubiquitous problem in statistics and machine learning of bias-variance trade-off. In a sense if the set $\mathcal{H}$ is really big then the best empirical error can be made really small at the cost of increasing the value of the Rademacher complexity, on the other hand a small class would have a small Rademacher complexity but it would have to pay the price of increasing the best empirical error. So in a sense we need to find the best hypothesis set for the data that we are given and this is precisely the machine learning way of solving these problems.

4 Algorithms

4.1 Support Vector Machines

As we said in section 1 a natural way of solving a classification problem would be that of empirical risk minimization. That is minimize

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i)$$

Since we are dealing with the $0-1$ loss we can rewrite the objective function as: $\sum_{i=1}^{n} 1_{h(x_i) \neq y_i} = \sum_{i=1}^{n} 1_{1 \neq y_i h(x_i)}$ where 1_z is just the indicator function. This function is nevertheless non-differentiable and non-convex and is thus not easy to minimize. In fact this problem is NP-hard so it is pointless to try to minimize this function. To deal with this problem we introduce the hinge loss or margin loss given by $\phi(y, y') = \max(0, 1 - yy')$ this function is an upper bound to the $0 - 1$ loss as shown in the figure, it is also convex and it turns out that [1]:

$$\min_{h \text{ measurable}} \mathrm{E}(L(h(x), y)] = \min_{h \text{ measurable}} \mathrm{E}(\phi(h(x), y)$$

which is indeed a desirable property. Thus we can replace the minimization problem by that of

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \max(0, 1 - h(x_i) y_i)$$

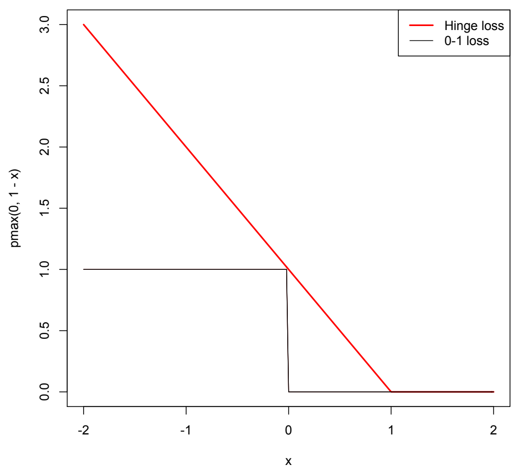


Fig. 1. Hinge loss is an upper bound for the $0 - 1$ loss

Now suppose the hypotheses class is given by linear functions $h(x) = w^T x + b$ for $(w, b) \in \mathbb{R}^{n+1}$ thus the problem becomes:

$$\min_{w} \sum_{i=1}^{n} \max(0, 1 - y_i(x_i^T w + b))$$

Or by adding some slack variables

$$\begin{aligned} &\min_{w} \sum_{i=1}^{n} \zeta_i \\ \text{subject } \; & y_i(x_i^T w + b) + \zeta_i \geq 1 \\ & \zeta_i \geq 0 \end{aligned}$$

This is an empirical risk minimization problem and it is a linear program so we can easily solve it using the simplex method. Nevertheless we said that we needed to take into account the complexity of the hypotheses class in particular we want to make the Rademacher complexity small but for vectors in R^d that term is on the order of $\sqrt{d+1}$ so if the dimension is really big we might risk over-fitting data. This is known as the curse of dimensionality. A way of controlling the complexity is by bounding the norm of the vector w as it was seen in example 3, so if we constrain $\|w\|^2 \leq \mathit{\Lambda}$ we get:

$$\begin{aligned} &\min_{w} \sum_{i=1}^{n} \zeta_i \\ \text{subject } \; & y_i(x_i^T w + b) + \zeta_i \geq 1 \\ & \zeta_i \geq 0 \\ & \|w\|^2 < \mathit{\Lambda} \end{aligned}$$

We can introduce a Lagrange multiplier $\lambda > 0$ for the last constrain and obtain

$$\begin{aligned} &\min_{w} \sum_{i=1}^{n} \zeta_i + \lambda \|w\|^2 - \lambda \mathit{\Lambda} \\ \text{subject } \; & y_i(x_i^T w + b) + \zeta_i \geq 1 \\ & \zeta_i \geq 0 \end{aligned}$$

Now here we assumed that we knew the optimal Lagrange multiplier for the problem which is of course not true, but since we also don't know what is the best $\mathit{\Lambda}$ we can just leave the problem as

$$\begin{aligned} &\min_{w} \sum_{i=1}^{n} \zeta_i + \lambda \|w\|^2 \\ \text{subject } \; & y_i(x_i^T w + b) + \zeta_i \geq 1 \\ & \zeta_i \geq 0 \end{aligned}$$

and this is the support vector machine (SVM) optimization problem. In practice the parameter λ is tuned using cross-validation on a held-out data set and that is what we did on this project.

4.2 Ridge Regression

If the loss function is the square loss then the natural optimization problem becomes

$$\min_{w,b} \sum_{i=1}^{n} ((w^T x_i + b) - y_i)^2.$$

Using a similar line of thought as in SVM we realize that if we want to control the complexity of the hypothesis space we need to bound the norm of w and so as before the optimization gets a 'regularization' parameter.

$$\min_{w,b} \lambda \|w\|^2 + \sum_{i=1}^{n} ((w^T x_i + b) - y_i)^2.$$

This algorithm is known as ridge regression. The regularization parameter λ controls over-fitting of outliers as seen in the picture below.

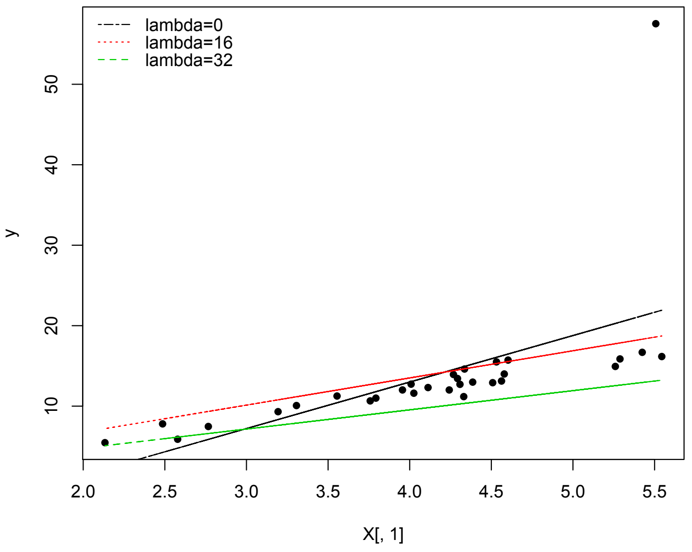


Fig. 2. Linear models for different values of λ. When $\lambda = 0$ the slope is over slanted as it tries to fit the outlier as much as possible.

4.3 Adaboost

Boosting is a way of combining simple hypotheses to create more complicated ones what it does is the following: at every iteration t it keeps a distribution D_t over the training data, it picks the best classifier $h_t \in \mathcal{H}$ and picks a weight α_t for this classifier, it then updates the distribution D_{t+1} by weighting more those points that were misclassified by h_t and after T rounds it returns the classifier $h_T = \sum_{i=1}^{n} \alpha_t h_t$. The algorithm is completely described below:

Algorithm 1 Adaboost pseudocode

```
Require: $(x_i, y_i)_{i=1}^n$
  $D_1(x_i) \leftarrow 1/n$
  for $i = 1 \rightarrow T$ do
      Find $h_t = \text{argmin}_{h \in \mathcal{H}} D_i(h(x) \neq y)$
      $\epsilon_t \leftarrow D_i(h_t(x) \neq y)$
      $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
      $D_{t+1}(x_i) \leftarrow \frac{D_t(x_i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$
  end for
  return $\sum_{t=1}^{T} \alpha_t h_t$
```

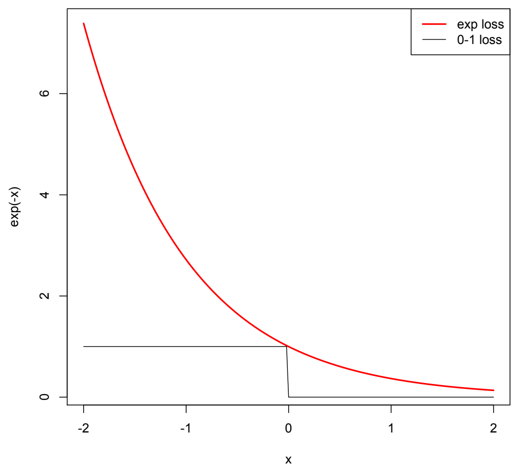


Fig. 3. The exponential loss is an upper bound for the $0-1$ loss.

The term Z_t appearing on the algorithm is simply a normalization factor. We now see how to cast Adaboost as an optimization problem. Consider the loss function $L(h(x), y) = \exp(-h(x)y)$ which is an upper bound for the $0-1$ loss as shown in the figure above. In [2] it is shown Adaboost is doing coordinate descent on the loss function just described where every iteration t is a step towards reducing the objective function. Although the prove of this is not hard it is rather long and is omitted, the interested reader can find the proof in [5]. Intuition tells us that we should let our algorithm iterate as much as possible nevertheless this might lead to overfitting (although not always), in practice what's done is early stopping which means to stop the iterations before achieving a minimum, in this way we try to control the complexity of the hypothesis we are creating. Other common practice to avoid overfitting is L_1 regularization, i.e. minimizing the function:

$$\sum_{i=1}^{n} \exp\left(-\sum_{t=1}^{T} \alpha_t h_t(x_i) y_i\right) + \|a\|_1$$

In the experiments we present the standard Adaboost and analyze it's generalization ability.

5 Experiments

5.1 Classification

For the problem of classification we consider the data set of Arrhythmia from the UCI repository (http://archive.ics.uci.edu/ml/datasets/Arrhythmia) the data set consists of 452 instances with 279 attributes, we used 352 for training and 100 for testing. We present two different results.One is obtained by using empirical risk minimization of the hingeloss. Since this is a linear program we used the simplex algorithm to find the best linear classifier, as it can be seen in the plot for different training sizes the training accuracy was always 1. Nevertheless the testing accuracy wasn't as good as the one given by SVM. For SVM since the data set was really small we did cross validation to tune the parameter λ only for the smallest training size (58) and then left that value of $\lambda = 16$ for the rest of the training sizes. As you can see the algorithm is really learning since the accuracy keeps steadily increasing as the training size increases as opposed to empirical risk minimization where the testing error varies wildly from training size to training size. Another thing worth noticing is that the training error is not as good as that of empirical risk minimization, this should be clear since SVM doesn't optimize this but the training and testing error seem to be converging as predicted by the bound (1)

The following experiment shows the use of Adaboost in the same data, as our base classifiers we used stump functions: for every coordinate $i \in \{1, \ldots, 279\}$ we define a set of hypotheses $H_i = \{h \in \mathbb{R} | h(x) = 1 \text{ if } x_i < h\}$ and $\mathcal{H} = \bigcup_i H_i$. Since the amount of data wasn't huge we let Adaboost run for 8 iterations and reported the results, after 4 iterations of Adaboost the testing error is already at its minimum and

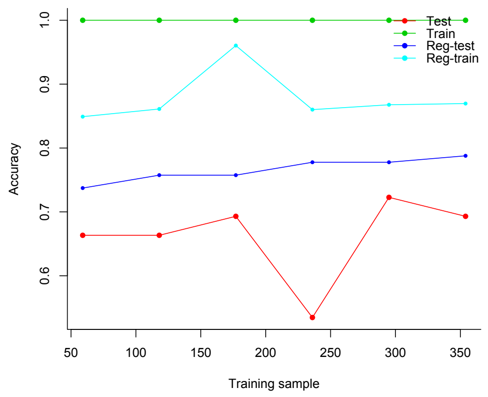


Fig. 4. Accuracy of empirical risk minimization and SVM. Reg-train and reg-test represent the accuracy of the regularized algorithm (SVM)

afterwards it starts to increase while the training error keeps decreasing, this is the overfitting effect that I described in the previous section and the reason why early stopping is necessary as a regularization technique.

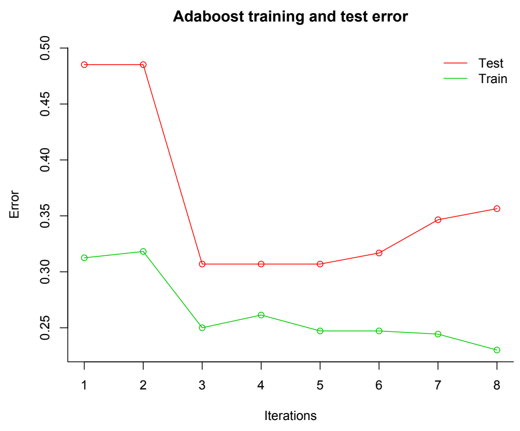
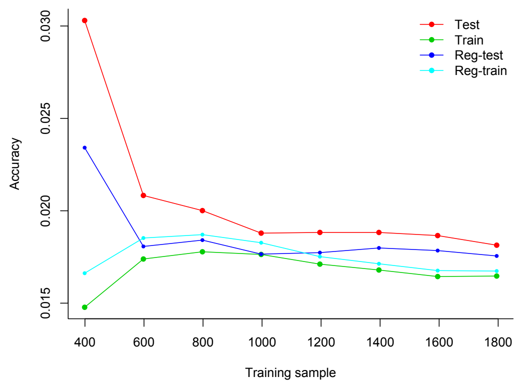


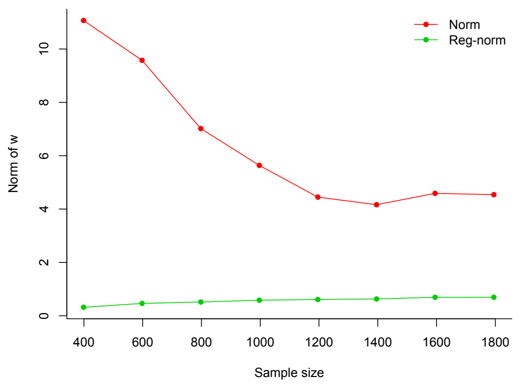
Fig. 5. Adaboost training and testing error

5.2 Regression

We present the results on two different regression tasks. Both from the UCI repository, the first one is the communities and crime data (http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime) that consists of 1994 points with 278 attributes each; we used 1800 for training and 194 for testing. As in the case of classification the λ parameter was tuned by cross validation on the smallest training sample (400) and left fixed to $\lambda = 6$. We plot the error of the hypotheses against the training sample size, the greatest advantage of ridge regression over simple regression is seen when the sample size is small where the error is 20% smaller the other plot shown is the square norm of the linear hypothesis w, big values on the norm of w mean that the vector was trying to explain the training data as good as possible, i.e it was overfitting.



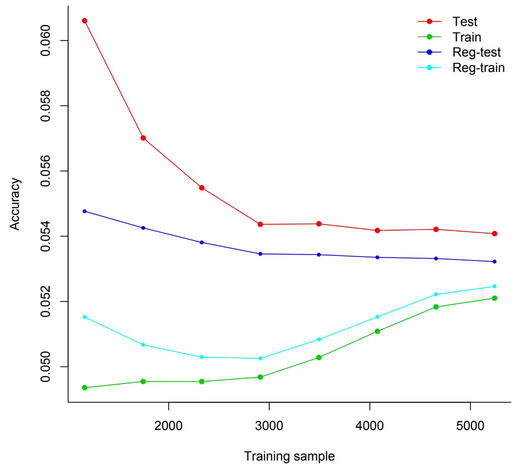
a)



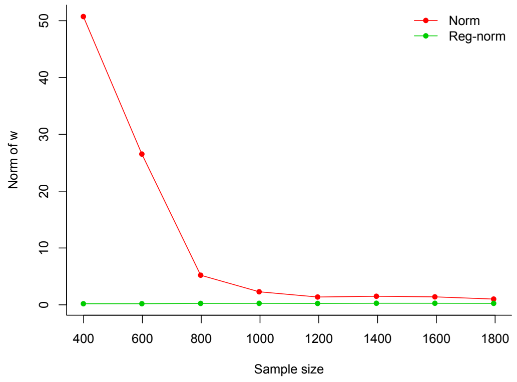
b)

Table 1. Communities data. a)Accuracy of the regularized and unregularized version of regression. b) Norm square of the vector w

The second data set is from the Insurance Company Benchmark which is a classification problem in reality but can still be seen as a regression problem, the size of the data set was of 9000 points with 86 attributes, after cleaning the data we were left with 5500 points for training and 1000 points for testing. In this particular challenge we can appreciate really well the advantages of regularization, the labels of the data were mostly 0 with maybe less than 10% of them being one, while the non-regularized regression tried to adapt to this labels, the regularized version treats them more as outliers and focuses on getting the 0 labels correctly as it can be seen in the norm plots where the regularized algorithm gives vectors with norms really close to 0 and the unregularized version is above 50 and it takes the algorithm to see 1000 points to realize that those points are outliers.



a)



b)

Table 2. Insurance Company Benchmark. a)Accuracy of the regularized and unregularized version of regression. b) Norm square of the vector w

6 Implementations and software

A note on the implementation of the algorithms. SVM training and testing was done with the lib-svm [3] library. The empirical risk minimization was done using the simplex method. Since the data set wasn't too big I used the implementation I had in my previous homeworks. Nevertheless for the biggest training set the algorithm was too slow and I ended up using the R implementation of the simplex method. The training of Adaboost was done in the whole training set, the implementation was done in R from scratch. Regression and ridge regression being unconstrained quadratic programs had a close form solution that was calculated exactly. Nevertheless in regular regression when the amount of data was too little compared to the dimensionality; positive semidefinite Hessians were obtained, in order to be able to deal with these problems we added $10^{-4}I$ to the Hessian to make the problem strictly convex.

7 Conclusions

The role of optimization in machine learning is crucial and it is always necessary, in fact without the developments on quadratic programming algorithms like SVM and ridge regression wouldn't be used in practice because of the size of the training samples. Nevertheless the use of raw optimization without the aid of statistics can lead to serious mistakes in experimental design as we showed in the experiments presented in this project. Although the problems treated here were of a modest size I believe that they still exemplify in a real way the problem of overfitting and how to control the problem with the use of regularization and early stopping. For really big data sets algorithms like SVM are no longer useful since training on millions or billions of points becomes infeasible, so people have turned to online learning which was something I wanted to treat on my project but the lack of time didn't allow me to.

References

1. Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe. Convexity, classification, and risk bounds. Technical report, JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION, 2003.
2. Leo Breiman. Prediction games and arcing algorithms, 1997.
3. Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.
4. Vladimir Koltchinskii. Rademacher penalties and structural risk minimization, 1999.
5. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning.* MIT Press, 2012.
6. L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.
7. Vladimir N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1998.