Konstantin Burlachenko (burlachenkok@gmail.com, bruziuz@stanford.edu)

Report for CS229: Convex Optimization For Machine Learning (cvx4ml)

Abstract

"Humanity is a wandering fires in the fog. The appearance of breakthroughs through the fog from one flame to another can be called a miracle - A.N. Kolmogorov". Machine Learning connects engineering fields with usual people life. But I believe that Machine Learning can be improved by mathematical optimization, which has already become an important tool in many areas. Very important that there are effective algorithms. It is possible solve problem with n variables and m constraints in time:$\max(n^3, n^2m, F)$ [I, p.8].

Introduction

I had two goals in this work. Create real effective solvers based on Interior Point Method for several models in Machine Learning which are reduced to convex optimization:

- Support Vector Machine and Linear Classifier. Which works good for small number of features.
- Variatons of least norm with different regularization which penalize model complexity in different interesting ways.
- Logistic Regression. First method you should always try before trying Neural Networks.

I showed how classical Machine Learning can be enhanced by ideas from Convex Optimization:

- Exploiting iterating L_1 algorithm to improve quality of Support Vector Machine and Linear Classifier. This is even more stronger trick to solve combinatorial problems via convex optimization, which in fact arises in this two models.
- Instead separation points by affine surface in high dimensional space points can be separated in similar way by any surface which is linear in parameters describes. One interesting example is ellipsoid surface.
- Use robust fitting instead least squares – like L_1 and Huber regression.
- Show how apply worst case analysis and stochastic robust approximation. I wanted to show that minimizing sum of discrepancies by sum of squares from 1820 is not the last word on modern optimization.

Path to **private repository**: https://bitbucket.org/bruziuz/cvx4ml_private
Path to **public repository**: https://bitbucket.org/bruziuz/cvx4ml

Related Work

Relative to demonstrated ideas what I'm doing here completely known by people from "math optimization community", but I'm not sure that it's well known for people who do machine learning. I think in battle of cration models all reasonable options should be considered. Relative to existing solution I tried to be better in terms of speed and speed is still very important due to two reasons:

1. Feature selection is combinatorial problem. To improve our ability to solve it we should have very fast fitting method.

2. With fast method we can work with more number of features and with more train samples

Things from convex optimization which matter for existing ML algorithms

One example is that one practical problem in ML that during optimize loss (objective) we can stopped earlier. This can be solved in more robust way then $|\nabla J(\theta)|_2^2 \leq \mu$ via Newton Decrement, which is for arbitrarily convex $J(\theta)$ give approximate value between loss in current point and it's minimum value, but it is more-more robust. For arbitrarily function it's heuristic too.

For some class of convex function which has property $|J'''| \leq 2(J'')^{\frac{3}{2}}$ for $J(\theta)$ restricted to any line this stopping criteria is not heuristic any more it's starting to be precise. This function are self-concordant and was introduced by Nesterov and Nemerovskiy [1,p.497] And several function which are naturally arises in Interior Point Method are self concordant:

1. Functions with $f''' = 0$ is self-concordant – affine and quadratic
2. Negative logarithm –log(x)
3. $-\sum \log(b_i - a_i^T x)$ is self-concordant

Classical analysis of Newton method which has been done by Kantarovich [I, p.504,eq.9.53] and modern analysis for self-concordant functions done by Nesterov and Nemervoskiy [I, p.505,eq.9.57] shows that upper bound of number of iterations depend lineary of initial gap. If you have linear model then to find coefficients the most general you can do to stay in room of convex optimization is todo $min. \sum f(b_i - {a_i}^T x)$ where f is convex and it can be non-differentiable.

This penalization is described via algebra, if problem is not big dimension or if you can imagine all things in mind then hthis problem can be seen as Geometry problem. If fact there is no big boundaries between geometry and statistics. This optimization problem can be interpreted in other way as maximum likelihood estimation in case of additive error given be $p(z) = \frac{\exp(-f(z))}{\int_{-\infty}^{+\infty} \exp(-f(u))du}$

Gradient and subgradients methods are slow and it's better to avoid them if possible. Of course sometimes it's impossible, but in problems which I will consider in's possible.

Even it's possible to formulate SVM with Hinge Loss [3], but if use methods which use differentiable convex loss, then this is only notation trick. Really problem reformulation should be done, e.g. via one of methods mention mentioned in (A.3). Another alternative use methods which works with non-differentiable loss functions, e.g. subgradients methods.

Dataset and Features

In order to test and debug algorithms I downloaded couple datasets which contains enough numerical features, because models for which I created solvers works with numerical data, not categorial. If feature is real catgorial, bur represented numerically my solvers will not know about it. And it still will be considered as real numer.

[1] Regression https://www.kaggle.com/harlfoxem/housesalesprediction/data

[2] Binary classification https://www.kaggle.com/primaryobjects/voicegender/data

I created several scripts to work with csv files:

Path	Description
[private_repo]/public/data/cvx4ml_info.py	Evaluate simple metric on features from csv file.
[private _repo]/public/data/cvx4ml_ holdout.py	Prepare train/test sets
[private _repo]/public/data/cvx4ml_read_top_n_lines.py	Read top n line from file.
[private_repo]/private/cvxpy_and_sklearn_impl/data_manipulation.py	Manipulate with input files and convert into numpy arrays internally

Filename are specified as "<filename>:start:end" where:

filename – is usual name of text csv file with features/target variables

start:end – specify the indicies of features you're interested in lie in line segment [start,end]

During using Python (CvxPy/SkLearn) and both my C++ solver implementation features can separated by tab, by space or by comma in CSV file. During loading features and target variables I didn't rescale features explicitly.

Methods

All full documention with problem reformulation, formulas, and various interpretations is available in
[private_repo]/private/optimization_problems/<PROBLEM_NAME>/opt_problem.[pdf|docx]

I have failed with one models from Convex Optimization, which can not be well solved in Machine Learning due to use high dimensionality of the problem. So at least to it in one machine it's a problem.

If imagine that there are 20 features and each features is bounded and we distrtitized each feature to 10 slots then joint mass function is described by $p \epsilon R^{20^{10}}$ so it's 10 trillions of scalar variables. Even for distribute convex optimization it can't be done.

Problem	Reason of interest	Path to description
Discriminant analysis with non-parametric estimation of distribution	1. To consider general form of distribution. 2. Append knowledge of KL-divergence with some distribution. 3. Append knowledge of minimum entropy of distribution into account 4. Append lower bound of variance and bounding constraints on expectation	binary_classification_gda_to _parametric
Instead fit parameters of linear(affine) function in feature space fir convex function	What is very outstanding we can solve this problem, unfortunately to solve this problem in high dimensional space is hard computationaly.	extra_fitting_convex_functio n_to_linear_regression

Other problems for which all is fine are listed below

Problem	Reason of interest	Path to description
Unconstrained Binary	Popularity of this binary classifier as first method to apply.	binary_classification_logis

classification via logistic regression and L_2 norm square regularization.		tic_regression
Unconstrained Binary classification via support vector machine	Popularity of this binary classifier as a second method to apply after logistic regression. Before deep learning it was one of the main method in Computer Vision as a final tool to handle feature space.	binary_classification_svm
Unconstrained Binary classification via support vector machine	Seek ellipsoid surface to approximately classify points. Minimize misclassified points via L_1 heuristic	binary_classification_svm_with_ellipsoid_surface
Unconstrained Binary classification via linear classifier	Seek affine function to approximately classify points. Minimize misclassified points via L_1 heuristic. Very good heuristic to misclassify few. Is it minimum number of points? No! We don't know as in this algorithm as in SVM.	binary_classification_svm_without_slab_constraint
Unconstrained Binary classification improved with reweighting technic.	Seek affine function to approximately classify points. Minimize misclassified points via improved L_1 heuristic. It will a separate chapter described this technic. Similar technic can be applied (and I applied) to SVM. In fact the difference between SVM and linear classifier is that SVM has extra quadratic term. Minimizing $\|a\|_2$ it the same as maximizing $\frac{2}{\|a\|_2}$ which is the width of the slab $1 \le a^T x - b \le 1$	binary_classification_svm_without_slab_constraint_improved
Extra constraint to Linear regression	Models which I considered in my work are all convex. Of course ML is not restricted by such models. But for convex models any affine equality constraints can be append to modeling phenomen. Like explicit equality constraint for cruicial important sample, which somebody pushed you to classify in some specific way.	extra_constraints_to_linear_regression
Unconstrained Robust Linear regression with DeadZone	We use L_1 norm for robust approximation. But we know that additive errors which is smaller then "a" is fine. Our model consideration should not based on such errors.	linear_regression_dead_zone
Unconstrained Robust Linear regression with Huber	To not go crazy if outliers. For small residuals we have behavior as with L_2norm square, but then if residual is big (more then M) we consider it as outlier. A lot of fields still don't know about it even it's backtrace to 1960.	linear_regression_huber
Unconstrained Linear regression with L_1 norm (robust estimator problem)	We estimate x s.t. $y = ax + v$ where $v \sim Laplace(0,2\sigma^2)$ I.e. p.d.f. of density is $p(z) = \frac{1}{2a}\exp\left(\frac{\|z\|}{a}\right)$	linear_regression_L1
Unconstrained Linear regression with L_1 norm and with stochastic robust approximation	minimize $E[\|Ax - b\|_1]$ in assumption that we describe uncertanty in (A,b) by sampling or state that (A_k, b_k) has probability p_k	linear_regression_L1_with_stochastich_robust
Unconstrained Linear regression with L_2 norm square (a.k.a. least-squares)	We estimate parameters of linear model under assumption that nois is $v \sim N(0, \sigma^2)$ is additive, unbiased. Old technic introduced by Gauss in 1820 and translated by G.W.Stewart in 1995. Which was very popular during 20th century.	linear_regression_L2
Linear regression with L_2 norm square and L_1 regularization (a.k.a. lasso regression)	L1 norm can be used as heuristic to say that parameters of our model should be sparse	linear_regression_L2_with_L1regularization
Linear regression with L_2 norm square with L_2 norm square regularization (a.k.a. ridge regression or Tikhonov regularization)	If we have several solution to original optimization problem select solution with smallest vector of parameters measured by square of L2 norm.	linear_regression_L2_with_L2regularization
Unconstrained Linear regression with L_2 norm but with stochastic robust approximation	Minimize expectation of L2 norm of discrepancy for linear model.	linear_regression_L2_with_stochastich_robust
Linear regression with L_2 norm and with constraint that parameters should lie in L_2 norm ball.	Introduce trust region in form of ball in which we consider parameters	linear_regression_L2_with_trust_region
Unconstrained Linear regression with L_2 norm with worst case robust approximation considering	Minimize maximum discrepancy measure by L_2 norm for linear model. This model can handle multiplicative aerror.	linear_regression_L2_with_worst_case

Unconstrained Linear regression with L_∞ norm (Chebyshev optimization problem)	Esrimate parameters of linear model in assumption that noise is uniform distribution in some known range of values $[-a, a]$.	linear_regression_Linf
Unconstrained Linear regression with L_∞ norm with stochastic robust approximation	Minimize expectation of L_∞ norm of discrepancy for linear model.	linear_regression_Linf_with_stochastich_robust
Unconstrained Robust Linear regression with log barrier	But never and never residual during fitting should be in it's absolute value greater then some limit a. It can be needed for take some garantess from fitting process.	linear_regression_LogBarrier

Methods. Several minimization strategies and distribution of residual.

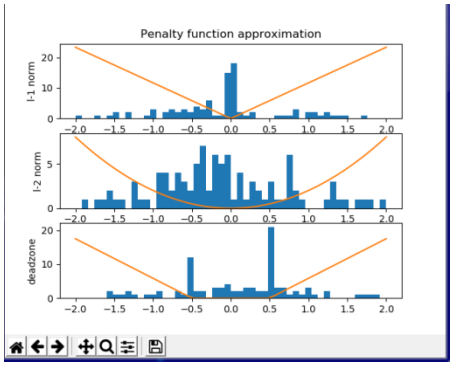

Image generated by **[private_repo]**/private/scripts_for_report/compare_penalties.py

If we fit linear model via minimizing empirical loss in form $\sum_{i=1}^{m} \mathcal{L}\left(x^{(i)}, y^{(i)}; \theta\right) = \sum_{i=1}^{m} \mathcal{L}\left(\theta^T x^{(i)} - y^{(i)}\right)$ with variable θ then usage of various norm for $\mathcal{L}$ if data can not be perfectly fitted lead to various distributions of residuals.

Methods. Future derivations

Due to limits of the report I provide several links to technical derivations and considerations.

Link	Description
cvx4ml_private\private\derivations\log_barrier_functions.pdf	Formulas for Hessian and gradient log barrier functions
cvx4ml_private\private\derivations\phase_1_derivations.pdf	Derivation of formulas for Phase-1 method for interior point with infeasible start. Derive it by myself because I couldn't find it in the web
cvx4ml_private\private\derivations\ problem_reformulation.pdf	Several common problem reformulation tricks to remove not-differentiablity from loss (objective)
cvx4ml_private\private\derivations\ fallacies_about_newton_method.pdf	Fallacies about unconstrained Newton Method
cvx4ml_private\private\derivations\ conjugate_gradient_algorithm.pdf	About fast Conjugate Gradient Algortihm. By the way which can potentionally diverge due to round off errors.
cvx4ml_private\private\derivations\ L1_heuristic.pdf	About L1 heuristic

Comparing various ways to create models for regression

Number of samples in train and dev is totally 21000. Train and dev obtained by 70/30 split. Error is measured as mean square error. For this dataset best performance is belong to linear_regression_L2_with_L2regularization and e.g. Huber is not outperform it.

Model name	Mean Square Error on Train	Mean Square Error on Test
linear_regression_L2	1782.84852355	2739.76809487
linear_regression_L2_with_L1regularization, lamba=0.1	1808.69561332	2741.37679232
linear_regression_L2_with_L2regularization,lambda=0.2	1783.91515791	2732.96556962
linear_regression_L2_with_stochastich_robust, buckets=20	1783.05980256	2736.0088867
linear_regression_L2_with_trust_region, D=2000,x0=0	1802.60661679	2735.21353917
linear_regression_L2_with_worst_case , buckets=20	1790.75216496	2775.87643092
linear_regression_Linf	4202.59961688	7795.18892507
linear_regression_Linf_with_stochastich_robust,buckets=50	2958.29507905	4971.24011347
linear_regression_Huber, MHuber = 500.0	1852.24555136	2818.73484157

linear_regression_L1	1852.16000858	2818.41414265
linear_regression_L1_with_stochastich_robust , buckets=20	2111.77019395	3214.15590754
linear_regression_L1_with_trust_region,D=2000,x0=0	1872.52507745	2828.63282284
linear_regression_deadzone	1852.16272262	2818.41181079

Comparing various ways to create models for binary classification

Number of samples in train and dev is totally 3100. Train and dev obtained by 70/30 split. Error measured as mean accuracy of correct classification. We see that for SVM and for linear classifier reweighting improve perfomance on test set.

Model name	Accuracy on train	Accuracy on test
binary_classification_logistic_regression	0.886175115207	0.27311827957
binary_classification_svm	0.928110599078	0.748387096774
binary_classification_svm with "20" reweighting iterations	0.920737327189	0.830107526882
binary_classification_svm_with_ellipsoid_surface	0.979723502304	0.852688172043
binary_classification_svm_without_slab_constraint	0.979262672811	0.851612903226
binary_classification_svm_without_slab_constraint with "20" reweighting iterations	0.98064516129	0.866666666667

Comparing my solvers with existing solvers in time in my Intel I7 2.6Ghz machine with Windows7.

Code for python scripts which implements all mentioned above with leveraging on cvxpy[4] and on SkLearn[10], when SkLearn supports model is given here: **[private_repo]/cvxpy_and_sklearn_impl/**. Code for C++ solvers is here **[private_repo]/tools/***, all this example share common code from **[private_repo]/** cvx4ml and **[private_repo]/** tools /cvx4ml_ common

Model	Time for my solver	Time for CVXPY/MOSEK	Time for CVXPY/ECOS	Time for CVXPY/SCS	Time for SkLearn	Mse For Train Error	Mse For Test Error	Size of train and test set together	Features (no intercep)	Extra info
linear_regression_L2	0,0120	0.43299	0.102445	4,6947	0.0309	1783.420567	2739.9720	21000	13	
linear_regression_L2_ with_L2regularization	0.0110	infeasible	0.744999	10.156000	0.0160	1783.915157	2732.9655	21000	13	gamma=0 .2
linear_regression_Linf	0.032	0.062	0.047	1.287999	not impl.	10932.13046	14071.547	1000	13	
lin_regression_deadzone	0.15400	0.108999	0.20300	2.3169999	not impl.	7978.67630	9926.5436	1000	13	AZone=1

Model	Time for my solver	Time for CVXPY/MOSEK	Time for CVXPY/ECOS	Time for CVXPY/SCS	Time for SkLearn	Accuracy of classify on train	Accuracy of classify on test	Size of train+test	Features (no intercept)
binary_classification_logistic_regression	0.0040	error	4.91700005	hang	0.01600	0.88617511	0.282796	3100	19
binary_classification_svm	0.8030	0.016	0.08472524	3.9890000	1.61999	0.674285	0.48	1000	19
binary_classification_svm_without_slab_co nstraint	0.13100	0.016	0.10999989	3.546999	not impl.	0.69857	0.51666	1000	19
binary_classification_svm_without_slab_i mproved	4.59000	0.8849999	1.248000	errors	not impl.	0.708571	0.533333	1000	19

Conclusion

Iterative reweighting improves performance. I would like to get more knowledge about this technic. In future I will visit classes of Emanuel Candess (https://statweb.stanford.edu/~candes/) to get more knowledge about L1 and compress sensing. Newton method works fine in theory. In practice I faced with several problems. Various ways to solve system of linear equation can failed because all numeric suff in computer is noisy due to roundoff and also not perfectly correct arithmetic in IEEE 754-1985. To handle it I always use three different algorithms and possible on scenario preconditioning. Fast conjugate gradient method sometimes diverge

Future Work

1. In my implementation of linear algebra I implemented via usual addition and multiplication, even I used AVX2 for optimize arithmetic. Also internally I has notion of light vector and light matrix to reduce not nessessary copy of data. But I think it can be interesting option to implement vectors from R^n not as elementwise vectors but as elemenents of some span, with dimension less then n.
2. I have not implemented SOCP and SDP solvers, so to solver be more complete I want to append solvers to solve problems in this popular cones.
3. I stored matrices as dense in my solver, even I worked with them via exploiting structure. I decided to do it because hardware more nice works with continuous memory in computer. In future I hope store structured matrices in some more better way then do it naively.
4. When I will completely finish with it I want to try build better models for Deep Learning based on non-convex optimization. Right now only momentum and stochastic gradients are widely popularized, but there are exist others (Particle method which even don't use derivative, Difference of convex function, Sequential Convex Optimization, Branch and Bound), [11]

References to literature and bibliography

[1] or [cvxbook] Convex Optimization 1st Edition by Stephen Boyd (Author), Lieven Vandenberghe (Author).

https://www.amazon.com/Convex-Optimization-Stephen-Boyd/dp/0521833787

https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

[2] Boris Polyak, Introduction to optimization

https://www.amazon.com/Introduction-Optimization-Translations-Mathematics-Engineering/dp/0911575146

[3] John Duchi, CS229 Supplemental Lecture notes about loss functions.

http://cs229.stanford.edu/extra-notes/loss-functions.pdf

[4] CvxPy: http://www.cvxpy.org/en/latest/

[5] MS&E 318/CME 338: Large-Scale Numerical Optimization

http://web.stanford.edu/class/msande318/notes/notes-first-order-smooth.pdf

[6] Lieven Vandenberghe, EE236C - Optimization Methods for Large-Scale Systems

http://www.seas.ucla.edu/~vandenbe/236C/

[7] EE364a: Convex Optimization I - Professor Stephen Boyd, Stanford University

http://stanford.edu/class/ee364a/

[8] EE364b: Convex Optimization II - Professor Stephen Boyd, Stanford University

https://stanford.edu/class/ee364b/

[9] CS229 lecture notes

http://cs229.stanford.edu/syllabus.html

[10] Packet Scy-Learn (http://scikit-learn.org/stable/)

[11] EE364B, 2008, L12,10:45:"In non-convex optimization there are lot of room for personal expression" - S.Boyd

(https://www.youtube.com/watch?v=cHVpwyYU_LY&feature=youtu.be&t=646)