Technical Report

Optimization Plugin for RapidMiner

Venkatesh Umaashankar
Sangkyun Lee

04/2012

technische universität dortmund

SFB 876 Verfügbarkeit von Information durch Analyse unter Ressourcenbeschränkung

Part of the work on this technical report has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project C1.

Speaker: Prof. Dr. Katharina Morik
Address: TU Dortmund University
Joseph-von-Fraunhofer-Str. 23
D-44227 Dortmund
Web: http://sfb876.tu-dortmund.de

Abstract

Optimization in general means selecting a best choice out of various alternatives, which reduces the cost or disadvantage of an objective. Optimization problems are very popular in the fields such as economics, finance, logistics, etc. Optimization is a science of its own and machine learning or data mining is a diverse growing field which applies techniques from various other areas to find useful insights from data. Many of the machine learning problems can be modelled and solved as optimization problems, which means optimization already provides a set of well established methods and algorithms to solve machine learning problems. Due to the importance of optimization in machine learning, in recent times, machine learning researchers are contributing remarkable improvements in the field of optimization. We implement several popular optimization strategies and algorithms as a plugin for RapidMiner, which adds an optimization tool kit to the list of existing arsenal of operators in RapidMiner.

Contents

1 Introduction

In mathematics, optimization, usually refers to maximizing or minimizing a function $f(x)$'s value by choosing input values from a valid set and finding the input value which results in the maximum or minimum value of the function. x could be either vector or a scalar. When x is scalar, it is an one dimensional optimization problem and in case of vector it becomes a multi dimensional optimization problem. From the perspective of machine learning, many learning problems can be solved as multidimensional unconstrained minimization problems.

1.1 Mathematical formulation

We will describe what a minimization problem is and also set the notations that will be used throughout this report. An optimization problem could be either constrained or unconstrained. Constrained optimization problems are the problems that have constraints (equality and non equality constraints) on the variables and unconstrained minimization problems are those that do not have any constraints. In this plugin we mainly focus only on unconstrained optimization problems and we implement some of the algorithms to solve them.

An unconstrained optimization problem can be expressed as

$$\theta^*, \arg\min_{\theta \in R^p} J(\theta, X)$$

the aim is to find the value of θ^* which minimizes the value of the objective function $J(\theta, X)$.

1.1.1 Notations from machine learning perspective

- X a matrix containing the training data of size $n \times p$, where n is the number of rows or observations and p is the number of attributes in each observation. Each row in X is a vector $x^{(i)} \in \mathbb{R}^p$, a row vector of size $1 \times p$, for $i = 1..n$. *Note: an addition column with value 1 is added as first column in the data matrix to handle the intercept term in the optimization problem.*

- y, a column vector of size $n \times 1$, which contains the labels for each observation $x^{(i)} \in X$ and $y^{(i)} \in \mathbb{R}$ for $i = 1..n$

- $\theta \in \mathbb{R}^p$, the optimization variable or weights of the learning problem, is a column vector of size $p \times 1$

- $J(\theta, X)$, the cost or the loss function or the objective function to be minimized

1.2 Smooth and non-smooth problems

An optimization problem can be classified in to two categories mentioned below.

Smooth: The cost function is continuous, differentiable and has a second derivate, the problem is called a smooth optimization problem. eg. linear least squares problem.

Non Smooth: The cost function is not differentiable. e.g. hinge loss.

In this plugin we implement algorithms that support both smooth and non smooth problems. The algorithms supported are explained in the coming chapters in detail.

1.3 Iterative optimization algorithms

The optimization algorithms implemented in the plugin are iterative in nature. The learning process starts with either a random value or a zero vector for the learning variables and during every iteration, the value of the learning variables are modified, such that the new value reduces the value of the objective function. The learning variables in each iteration are also called as iterates. The iterations are repeated until any of the following conditions are satisfied.

1. specified maximum number of iterations exceeded.
2. convergence is achieved.
3. no more improvements can be made.

In the following sections, we describe the algorithms specific to the section [1.2] that are implemented in the plugin.

1.4 Optimization based learning for RapidMiner

The optimization plugin for RapidMiner aims at introducing mathematical optimization based learning capability to RapidMiner. An unconstrained optimization problem, which is called learning (or) training in machine learning can be solved with different algorithms. Each algorithm has a set of configurable items, say, function to be optimized, choice of search direction, regularization type, choice of learning rate (line search) etc. There are different options available to perform the task pertaining to each configuration. This plugin aims at giving a clean separation of these configurable items, so that the user can configure these items before starting the learning. Thus, giving the user, an easily configurable optimization based learner.

1.5 Download

The optimization plugin for RapidMiner is available for download from the link https://bitbucket.org/venkatesh20/optimization-extension

2 Smooth optimization

In this chapter, we describe the algorithms that are implemented in the Optimization plugin that support smooth optimization. The algorithms for smooth optimization exactly follow the steps explained in the section 1.3 in the previous chapter. Smooth optimization algorithms generally use grandient and hessian information of the objective function to calculate the next iterate.There are two strategies available for calculating the next iterate

Line search: A search direction p_k is identified and we move along the search direction hoping to minimize the cost. We discuss the choices for search direction later in this chapter.

Trust region: A local simple model of the objective model is build and the local model is minimized in the region where it can be trusted.

We implement only Line search based algorithms in the plugin and trust region based algorithms are currently not implemented in the plugin, due to their sophisticated nature. Trust region based methods may be supported by the plugin, in the future.

2.1 Line search

In case of line search, the update rule for the next iterate is given by

$$\theta_{k+1} = \theta_k + \alpha p_k \tag{1}$$

where

- $p_k \in R^p$, a column vector which gives the search direction
- $\alpha \in R$, learning rate or step size of the line search.

2.2 Search direction

The plugin supports the two choices listed below for selecting the search direction.

2.2.1 Gradient descent

In case of gradient descent or steepest gradient, the negative of the gradient is chosen as the search direction.

$$p_k = -\nabla J(\theta, X) \tag{2}$$

Gradient descent has a linear rate of convergence and could be quite slow in some cases.

2.2.2 Newton method

In case of Newton's method, the search direction is given by

$$p_k = -(\nabla^2 J(\theta, X))^{-1} \nabla J(\theta, X) \tag{3}$$

where $-(\nabla^2 J(\theta, X))^{-1}$ is the inverse of the hessian matrix (second order derivatives) of the function $J(\theta, X)$.

To use the plain Newton's method, the hessian matrix should be positive definite. Newton's method has a quadratic rate of convergence if the learning is started with a good approximation of the learning variables which is near the minimum, otherwise it may fail to converge even. There are modified versions of Newton's method which make some modifications to the hessian matrix to make it positive definite and also to make it work for any starting point.

2.3 Smooth loss functions

This section describes the loss fuctions that are implemented in the plugin, which are available for the user to configure a smooth optimization problem. The same notations presented in the section 1.1.1 are used to describe the loss functions.

2.3.1 Least squares

The cost function for the least squares problem is given by

$$J(\theta, X) = 1/2n \sum_{i=1}^{n} (h(\theta, x^{(i)}) - y^{(i)})^2 \tag{4}$$

where $h(\theta, x^{(i)})$ is the hypothesis function that is used to predict the label and is given by the below equation.

$$h(\theta, x^{(i)}) = x^{(i)}\theta \tag{5}$$

Gradient for least squares The gradient for the least squares problem is given by

$$\nabla J(\theta, X) = \frac{1}{n} \sum_{i=1}^{n} (h(\theta, x^{(i)}) - y^{(i)}) x^{(i)} \tag{6}$$

Hessian for least squares The hessian for the least squares problem is given by

$$\nabla^2 J(\theta, X) = X^T X \tag{7}$$

2.3.2 Logistic loss

The cost function for the logisitic loss or logistic regression problem is given by

$$J(\theta, X) = \frac{1}{n}\sum_{i=1}^{n} -log\left\{\frac{1}{1+e^{(-y^{(i)}x^{(i)}\theta)}}\right\} \tag{8}$$

$h(\theta, x^{(i)})$ is the hypothesis function that is used to predict the label and is given by the below equation.

$$h(\theta, x^{(i)}) = \begin{cases} +1, & \text{if } 1/(1+e^{-x^{(i)}\theta}) \geq threshold(\text{usually } 0.5) \\ -1 & \text{otherwise} \end{cases} \tag{9}$$

Gradient for logistic loss The gradient for the logistic loss is given by

$$\nabla_{\theta_j} J(\theta, x^{(i)}) = \frac{1}{1+e^{(y^{(i)}x^{(i)}\theta)}}(-y^{(i)}x_j^{(i)}) \text{ where } \theta_j \text{ is } j^{th} \text{ compoment of the vector } \theta \tag{10}$$

Hessian for logistic loss The hessian for the logistic loss is given by

$$\nabla_j \nabla_k J(\theta, x^{(i)}) = x_j^{(i)} x_k^{(i)} \frac{e^{(y^{(i)}x^{(i)}\theta)}}{\left(1+e^{(y^{(i)}x^{(i)}\theta)}\right)^2} \tag{11}$$

2.4 Smooth regularizers

In an optimization problem, regularization is normally done to reduce overfitting of the model to the training data. The smooth optimization problem type supports only ℓ_2 regularization. The regularization term, $\psi(\theta)$ is usally added to the loss of the chosen loss function.

2.4.1 ℓ_2 regularizer

The loss function of the ℓ_2 regularization is given by

$$\psi(\theta) = \frac{\lambda}{2}\sum_{i=2}^{p} \theta_j^2 \tag{12}$$

Gradient of ℓ_2 regularization term

$$\nabla\psi(\theta_i) = \theta_j \tag{13}$$

Hessian of ℓ_2 regularization term

$$\nabla^2\psi(\theta_i) = 1 \tag{14}$$

The first term θ_1 (intercept or bias term) is not usually included in the regularization. L2 regularization is smooth and can be easily solved when summed with other smooth loss functions. L2 regularization keeps the values of the optimization variable θ small and it also reduces the overfitting.

2.4.2 Group ℓ_1 regularizer

Group ℓ_1 is similar to ℓ_1 at group level. Depending upon the the value for λ, a group of co-efficients could be dropped out of the model. If the group size is reduced to one, this is same as ℓ_1 regularization. Sometimes, attributes can be grouped based on the domain knowledge, in such cases group ℓ_1 could be very useful. The loss function of the group ℓ_1 regularization is given by

$$\psi(\theta) = \lambda \sum_{k=1}^{K} \|\theta_{G_k}\| = \lambda \sum_{k=1}^{K} \sqrt{\sum_{j=1}^{G_k} (\theta_{G_{kj}})^2} \tag{15}$$

where K is the number of groups, θ_{G_k} is a subvector of θ, which is called as group G_k. The elements with in the group are not allowed to overlap.

Gradient of group ℓ_1 regularization term

$$\nabla\psi(\theta_{G_{kj}}) = \begin{cases} \lambda \frac{\theta_{G_{kj}}}{\|\theta_{G_k}\|}, & \text{if } \theta_{G_{kj}} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

The first term θ_1 (intercept or bias term) is not usually included in the regularization. Group ℓ_1 regularization is smooth . For more details, please refer to the paper [4]

2.5 Line search algorithms

2.5.1 Backtracking Armijo

Armijo's condition is given by

$$J(\theta_k + \alpha p_k, X) \leq J(\theta_k, X) + c_1 \alpha \nabla J(\theta_k, X)^T p_k \tag{17}$$

This means that the reduction in the loss, should be proportional to the learning rate α and also to the directional derivate $\nabla J(\theta_k, X) p_k$. In backtracking Armijo line search, we start with a maximum value of the learning rate α and keep reducing it by a factor till we find a value of α, that satisfies the Armijo's condition. This condition is also called as sufficient decrease condition. For a detailed description of the algorithm refer to algorithm 3.1 of Jorge Nocedal and Stephen J. Wright [1].

Armijo's line search can work with any smooth loss function.

2.5.2 Wolfe's line search

$$\nabla J(\theta_k + \alpha p_k, X)^T p_k \geq c_2 \nabla J(\theta_k, X)^T p_k \tag{18}$$

The above equation is known as curvature condition. The Armijo's condition given in the equation. 17, and the above condition, both together are referred to as Wolfe conditions. In wolfe's line search, the selected learning rate alpha should satisfy both the wolfe conditions for constants $c_1 > 0$, $c_2 > 0$ and $0 < c_1 < c_2 < 1$. For a detailed description of the Wolfe line search algorithm refer to algorithm 3.5 of Jorge Nocedal and Stephen J. Wright [1].

Wolfe's strong line search can work with any smooth loss function and has generally faster convergence than that of back tracking Armijo.

3 Non-smooth optimization

3.1 Non-smooth Algorithms

In this section, we describe the algorithms for non smooth optimization that are implemented in the plugin.

3.2 Subgradient descent

In case of non smooth optimization, when the objective function or loss function is not differentiable, the derivative is replaced by its subgradient. g is a subgradient of a function f at x if

$$f(y) \geq f(x) + g^T(y - x) \; \forall y \in R^n \tag{19}$$

when a function is differentiable, the subgradient is equal to its gradient. Subdifferential $\partial f(x)$ of f at x is the set of all subgradients. Subgradient descent is same as the gradient descent or steepest descent algorithm, except for the fact that subgradient will be used instead of gradient and also subgradient descent may even converge slower than that of gradient descent. This is due to the reason that the direction provided by subgradient may not be strictly descreasing.

3.2.1 Non-smooth loss functions

This section describes the loss fuctions that are implemented in the plugin, which are available for the user to configure a non smooth optimization problem. The same notations presented in the section 1.1.1 are used to describe the loss functions.

3.2.2 Hinge loss

Hinge loss is a non differentiable, non smooth loss function. The cost function for the hinge loss is given by

$$J(\theta, X) = \frac{1}{m}\sum_{i=1}^{n} max\left\{0, 1 - y^{(i)}(\theta^T x^{(i)})\right\} \tag{20}$$

$h(\theta, x^{(i)})$ is the hypothesis function that is used to predict the label and is given by

$$h(\theta, x^{(i)}) = \begin{cases} +1, & \text{if } x^{(i)}\theta > 0 \\ -1 & \text{otherwise} \end{cases} \tag{21}$$

Subgradient for hinge loss The subgradient, $g \in \partial J(\theta, x^{(i)})$ for the hinge loss is given by

$$g = \begin{cases} -y^{(i)}x^{(i)}, & \text{if } 1 - (y^{(i)}x^{(i)}\theta) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

3.2.3 Nonsmooth regularizers

3.2.4 ℓ_1 regularizer

The loss function of the ℓ_1 regularization is given by

$$\psi(\theta) = \lambda \sum_{i=2}^{p} |\theta_j| \tag{23}$$

subgradient of ℓ_1 regularization term the subgradient $b \in \partial\psi(\theta)$ is given by

$$b_j = \begin{cases} -1, & \text{if } \theta_j < 0 \\ 1, & \text{if } \theta_j > 0 \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

The first term θ_1 (intercept or bias term) is not usually included in the regularization. ℓ_1 regularization is non smooth .ℓ_1 regularization keeps the values of the optimization variable θ sparse and it also reduces the overfitting.

3.2.5 Elasticnet regularizer

The loss function of the Elasticnet regularization is given by

$$\psi(\theta) = (\ell_1 + \ell_2) = \left(\lambda \sum_{i=2}^{p} |\theta_j| + \frac{\sigma}{2}\sum_{i=2}^{p} \theta_j^2\right) \tag{25}$$

subgradient of elasticnet regularization term the subgradient $b \in \partial\psi(\theta)$ is given by

$$b_j = \theta_j + \begin{cases} -1, & \text{if } \theta_j < 0 \\ 1, & \text{if } \theta_j > 0 \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

The first term θ_1 (intercept or bias term) is not usually included in the regularization. Elasticnet regularization is non smooth. Elasticnet regularization gives the combined effect of ℓ_1 and ℓ_2 regularization.

3.3 Stochastic gradient descent

The update rule for SGD is same as that of gradient descent, except for the fact that in SGD only one observation or example in the training data is processed per iteration, and the learning variable is updated every iteration. So the loss that is minimized is only the value of the loss function for current example. Hence it is a non smooth optimization problem. SGD supports all the loss functions [2.3.1] least squares, [2.3.2] Log Loss, [3.2.2] hinge loss and all the regularizers [3.2.5] L1, [2.4.1] L2, that we have seen so far.

3.4 ℓ_1 regularized dual averaging

The regularized dual averaging algorithm (RDA),based on the paper Lin Xiao [2] is implemented in the plugin. This implementation supports loss functions [2.3.1] least squares, [2.3.2] Log Loss, [3.2.2] hinge loss based on soft L1 regularization.

Each iteration of the RDA methods takes the form

$$\theta_{t+1} = \underset{\theta}{\operatorname{argmin}} \left\{ \frac{1}{t} \sum_{\tau=1}^{t} \langle g_\tau, \theta \rangle + \Psi(\theta) + \frac{\beta_t}{t} h(\theta) \right\} \tag{27}$$

where $h(\theta)$ is a strongly convex function, $\{\beta_t\}_{t \geq 1}$ is a non negative, non decreasing input sequence which determines the convergence property of the algorithm.

At each iteration, this method minimizes the sum of three terms: a linear function obtained by averaging all previous subgradients (the dual average), the original regularization function $\Psi(\theta)$ and an additional strongly convex regularization term $\frac{\beta_t}{t} h(\theta)$.

for $t = 1, 2, 3....$ **do**

1. Compute the subgradient g_t of the cost function w.r.t θ $\frac{\partial J(\theta)}{\partial \theta}$
2. Update the average subgradient
$$\bar{g}_t = \frac{t-1}{t} \bar{g}_{t-1} + \frac{1}{t} g_t$$
3. Now we update the θ as follows for soft L1 regularization, here the regularization function $\Psi(\theta) = \lambda \|w\|_1$ for some $\lambda > 0$ and the strongly convex function $h(\theta) = \frac{1}{2}\|w\|_2^2$

4. Loop through the elements of the average gradient vector $\bar{g}_t$, if $|g|_t^{(i)} \leq \lambda$, then set

$$\theta_{t+1}^{(i)} = 0$$

else

$$\theta_{t+1}^{(i)} = -\frac{\sqrt{t}}{\gamma}\Big(\bar{g}_t^{(i)} - \lambda \ \mathrm{sgn}(\bar{g}_t^{(i)})\Big)$$

We restrict ℓ_1 -regularization on part of the optimization variables only.The bias term is free of regularization. In this case, we can simply replace λ by 0 for the corresponding coordinates in the above update.

4 OptimLearner Operator and Configuration

The optimization plugin adds only one new operator called optimLearner. OptimLearner is highly configurable and supports the configuration of all the problem types, line searches and regularizers discussed earlier. A sample repository is made available along with the plugin. This repository contains many sample processes which show case several configurable options of the optimization plugin.

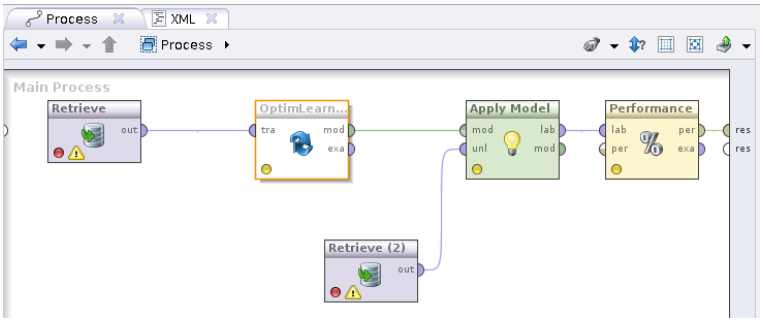

Figure 1: OptimLearner - Sample Process

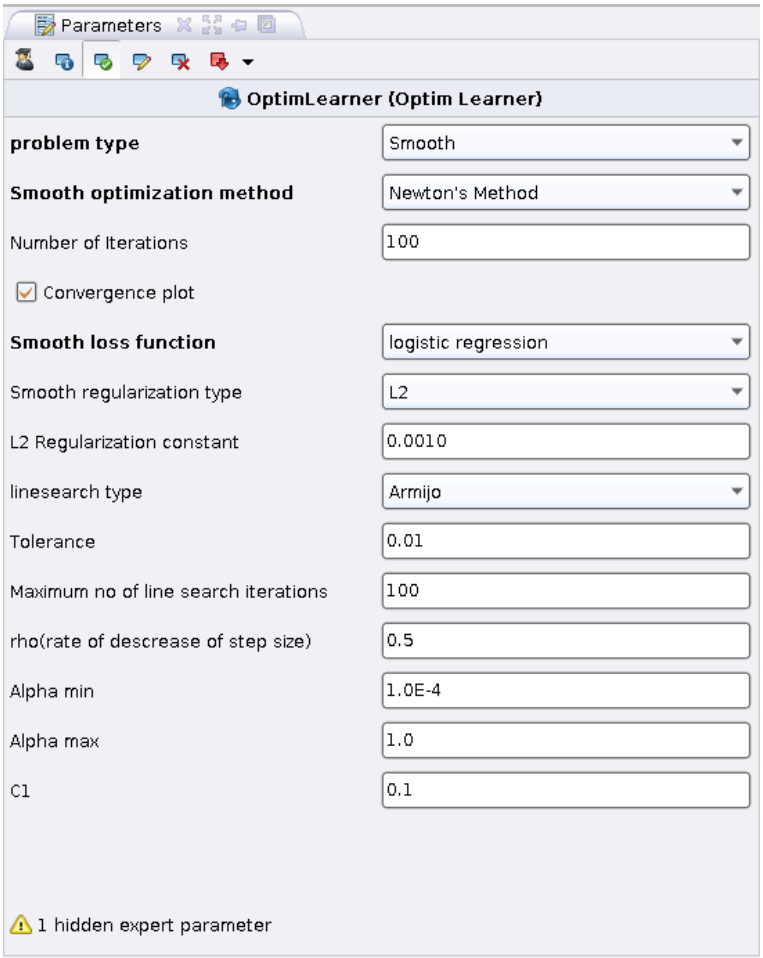

Figure 2: OptimLearner - Sample Configuration

5 Experiments

5.1 Datasets

1. **MNIST**: The MNIST dataset is a database of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples. We used the scaled version of mnist dataset available from libsvm datasets (http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multiclass.html). We extract the data pertaining to the numbers **6** and **8** to make the problem, a binominal classfication. From the extracted data, we considered 80%(9415) as training examples and 20%(2354) testing examples.

2. **Wine quality**: The wine quality data set is available from UCI machine learning repository (http://archive.ics.uci.edu/ml/datasets/Wine+Quality). We extracted the data pertaining to red wine only.we considered 80%(1279) as training examples and 20%(320) testing examples.

In the below experiments we train the model on the training set and validate the model performance on the test set and report the same.

5.2 First vs second order method

Here we compare the first order method *"Gradient descent"* and second order *"Newton method"*.

Experiment setup

- **Dataset**: Wine quality data set.
- **Regression type**: Least Squares , Linear Regression
- **Fixed tolerance**: tolerance was fixed at 1.0e-5 and both the methods were run until the stopping criterian is less than tolerance.

Table 1: First Vs second order - Performance metrics

Method	**RMSE**	**Absolute Error**
Gradient Descent	0.71339168 +/ -0.00000000	0.55016313 +/- 0.45414559
Newton Method	0.65892359 +/- 0.00000000	0.50665033 +/- 0.42129056

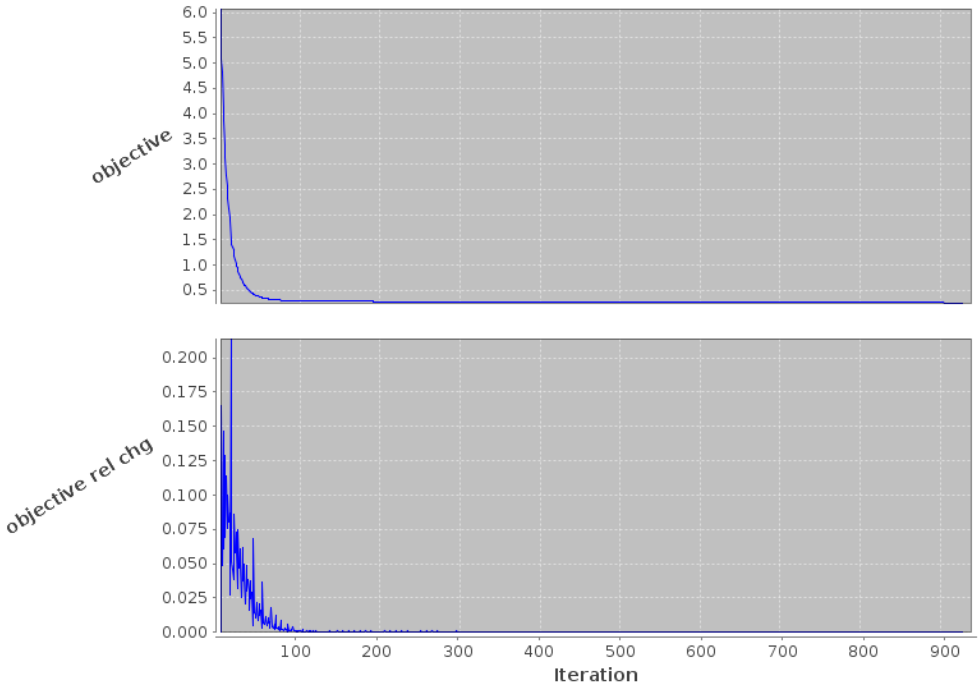

Figure 3: First vs second order - Gradient descent

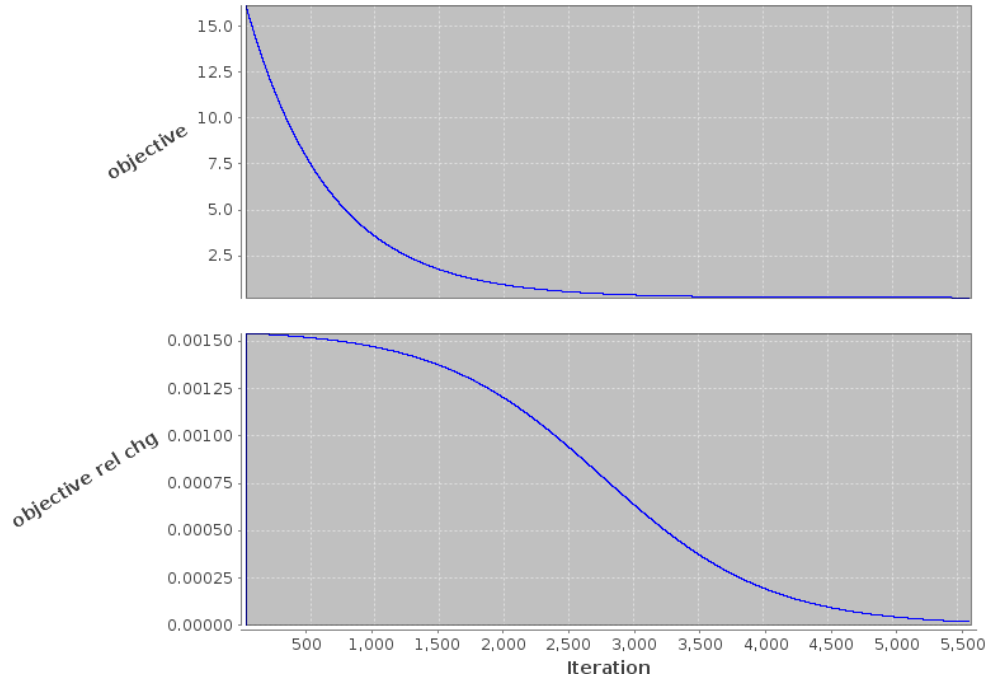

Figure 4: First vs second order - Newton method

We can observe from the plots that, when a point close to the minimum is reached, Newton method converges faster from that point than Gradient Descent. Theoritically, Newton method converges in quadratic rate near the solution.

5.3 Smooth vs non-smooth methods to nonsmooth problem

Here we compare the smooth method *"Gradient descent"* and nonsmooth method *"Stochastic gradient descent"* using a nonsmooth loss function, hinge loss as the objective function.

Experiment setup

- **Dataset**: MNIST.
- **Problem type**: Hinge loss with ℓ_2 regularization with $\lambda = 0.001$
- **Fixed time**: Both methods were run for a fixed time of 20 seconds. The time for objective computation is excluded for the both the methods.
- **Gradient descent**: fixed step size, $\alpha = 0.17$
- **SGD**: fixed step size, $\gamma = 2$

Table 2: Smooth vs non-smooth methods - Performance metrics

Method	**Accuracy (%)**	**F-Score (%)**	**No. of iterations**
Gradient Descent	78.12	79.26	113
SGD	98.85	98.87	225960

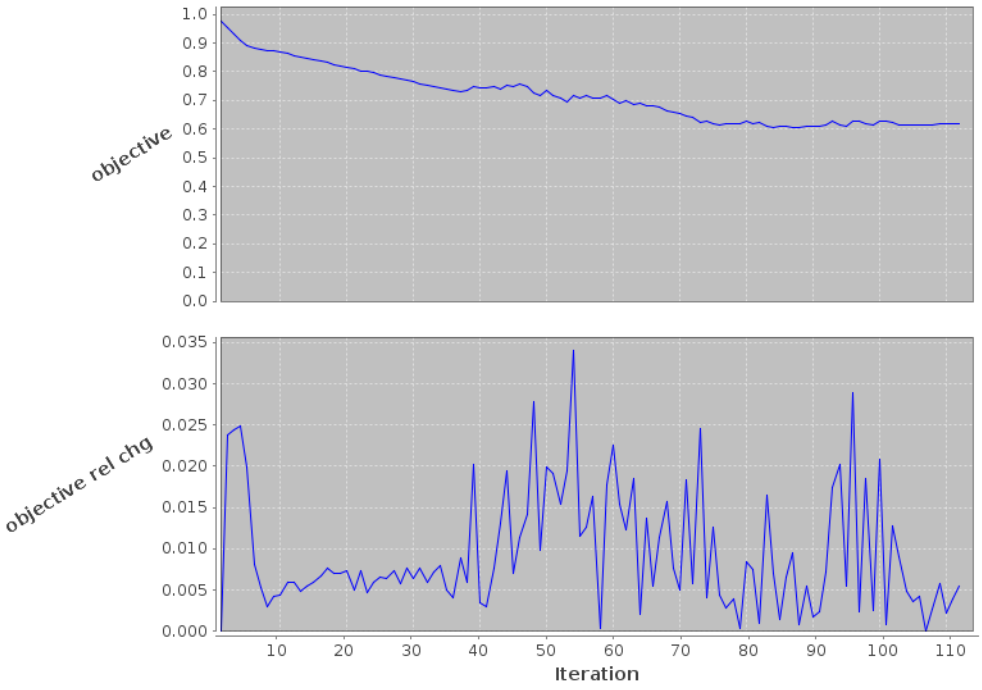

Figure 5: Smooth vs non-smooth - Gradient descent

We can observe from the plots that, in case of "Gradient Descent", the objective function value decreases gradually in each iteration, whereas in case of "Stochastic Gradient

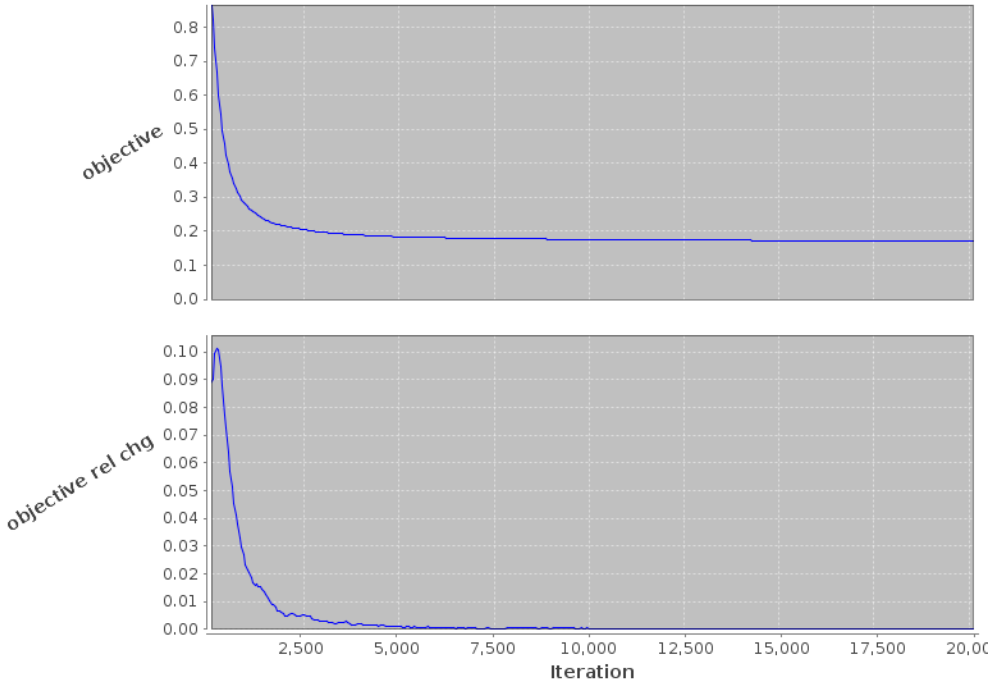

Figure 6: Smooth vs non-smooth - Stochastic gradient descent (only initial iterations)

Descent" the initial iterations make huge errors and the value of the objective function decreases drastically in the initial iterations and almost remains the same afterwards. And also it is very obvious that, in this case SGD converges faster than that of Gradient descent.

5.4 Batch vs online method

Here we compare the batch method *"Gradient descent"* and online method *"Stochastic gradient descent"* using Logistic loss as the objective function.

Experiment setup

- **Dataset**: MNIST.
- **Regression type**: Logistic regression with ℓ_2 regularization with $\lambda = 0.001$
- **Fixed time**: Both methods were run for a fixed time of 20 seconds. The time for objective computation is excluded for SGD, where as in case of Gradient descent objective value computation is a part of termination criterian.
- **Gradient descent**: *Armijo* line search, with $\alpha_{min} = 10^{-4}$, $alpha_{max} = 1.0$ and $c_1 = 10^{-4}$
- **SGD**: descreasing step size, $\gamma = 2$

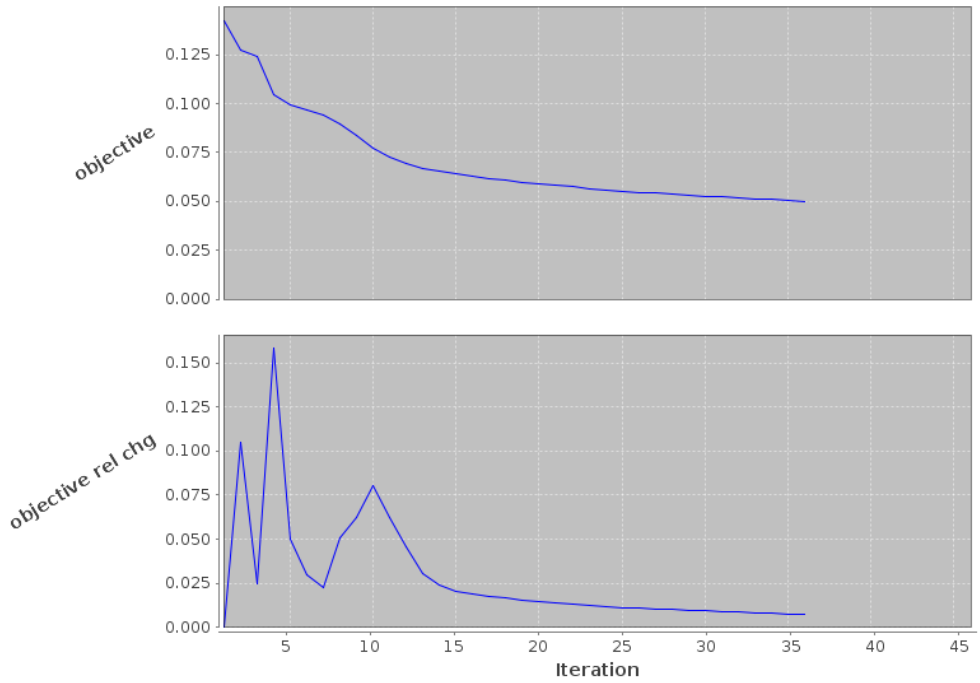

Figure 7: Batch vs online - Gradient descent

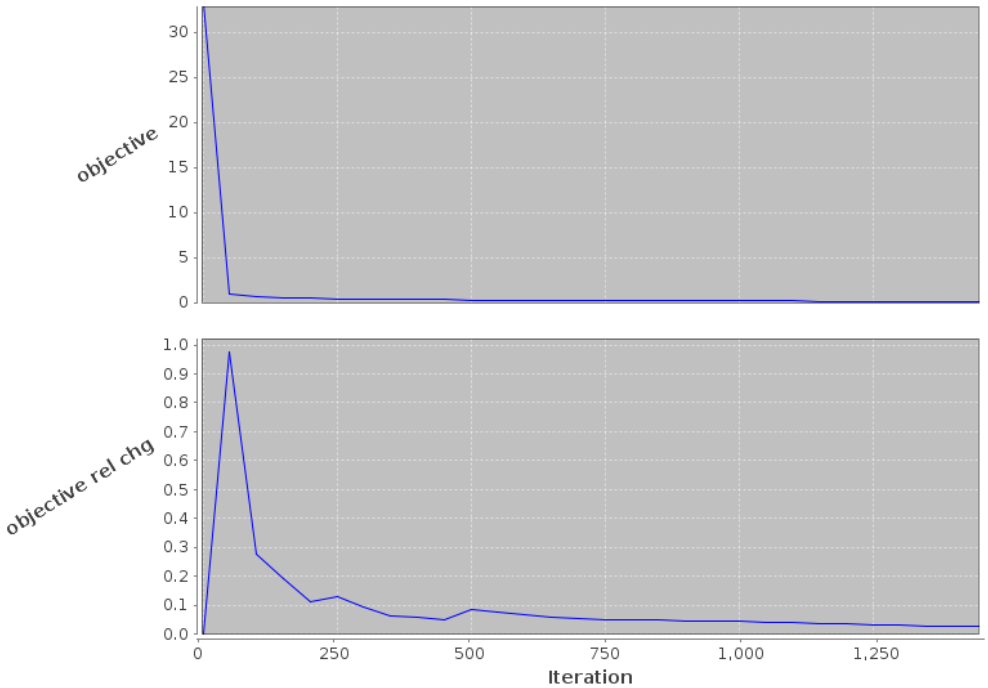

Figure 8: Batch vs online - Stochastic gradient descent (only initial iterations)

Table 3: Batch vs online Method - Performance metrics

Method	**Accuracy (%)**	**F-Score (%)**	**No. of iterations**
Gradient Descent	98.9	98.91	36
SGD	98.6	98.62	167460

We can observe from the plots that, the *"Stochastic gradient descent"* converged more faster than *"Gradient descent"*. Usually, when the dataset has good representative samples this happens and also, when the data set size is large, *"Stochastic gradient descent"* wins over *"Gradient descent"* most of the times.

5.5 Non-regularized vs regularized

Here, we compare the Non regularized versions of *SGD and RDA*, with regularized versions *SGD and RDA*, using Logistic loss as the objective function.

Experiment setup

- **Dataset**: MNIST.
- **Regression type**: Logistic regression.
- **Non regularized**: No regularizers applied.
- **Regularized**: ℓ_1 regularization for values of $\lambda = 0.0001, 0.001, 0.01, 0.1$
- **SGD**: descreasing step size, $\gamma = 2$
- **RDA**: $\gamma = 2$

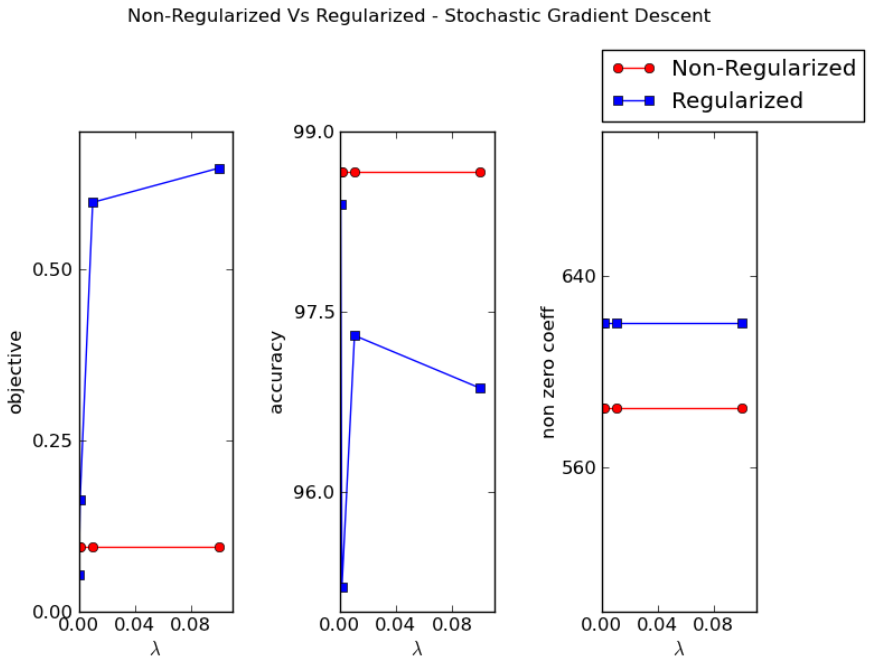

Figure 9: Non-regularized vs regularized - SGD

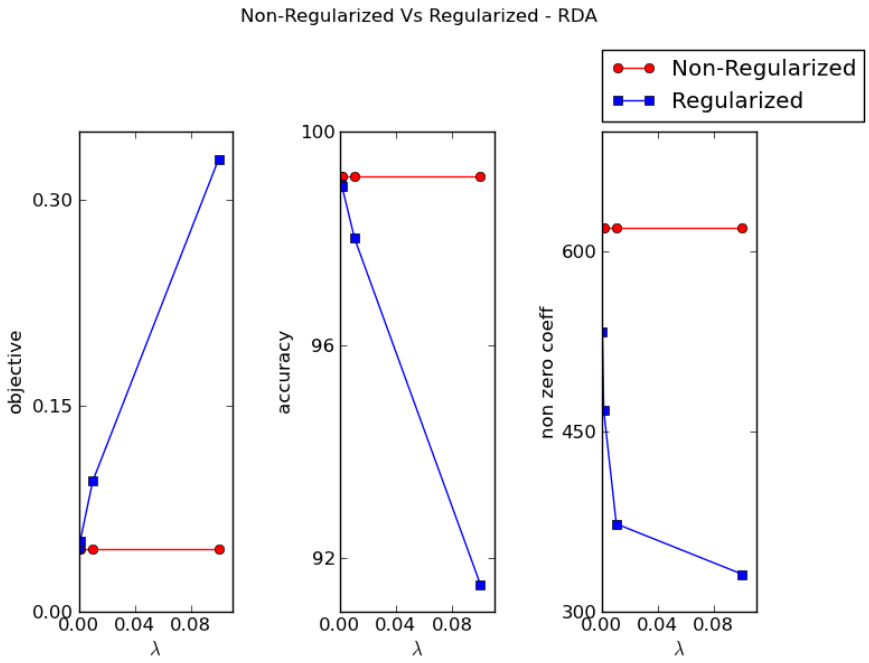

Figure 10: Non-regularized vs regularized - RDA

We can observe from the plots that, RDA produces higly sparse solutions than that of SGD, without compromising much on the accuracy. This could be very useful in high dimensional data space.

5.6 Optim-plugin vs Rapidminer vs Weka

Here, we compare the performance of operators from Optim-plugin, RapidMiner and Weka using the metrics run duration (dur), accuracy, fscore and number of non zeros in the coefficient vectors(non zeros).

Experiment setup

- **Dataset**: MNIST.
- **Regression type**: Logistic Regression.
- **Optim-plugin**:

  optim-GD: Gradient Descent from optimization plugin with logistic loss, ℓ_1 regularization, $\lambda = 0.01$, wolfe line search, tolerance = 10^{-4}.

  optim-SGD: $\gamma = 2$, number of epochs = 1, decreasing step size, with logistic loss, ℓ_1 regularization, $\lambda = 0.01$.

  optim-RDA: $\gamma = 2$, number of epochs = 1, with logistic loss, ℓ_1 regularization, $\lambda = 0.01$.

- **Rapidminer**:

  RM-LR: Logistic Regression operator from RapidMiner, Kernel type = dot, C = 0.0010, Convergence epsilon = 10^{-4}, Max iterations = 10.

- **Weka**:

  Weka-LR: Logistic learner from weka, ridge = 0.01, Max iterations = -1.

  Weka-SGD: SGD learner from weka, $\lambda = 0.01$, learning rate = 0.001.

Table 4: Optim-Plugin Vs Rapidminer Vs Weka - Performance metrics

Method	**Duration (sec)**	**Accuracy (%)**	**F-Score (%)**	**Nonzeros**
Optim-GD	62	97.88	97.91	620
optim-SGD	3	96.52	96.52	620
optim-RDA	3	98.09	98.12	419
RM-LR	200	97.88	97.87	620
Weka-LR	53	98.51	98.53	623
Weka - SGD	6	99.02	99.0	605

6 Conclusion

We presented a plugin to RapidMiner which provides algorithms to solve optimization problems. The operator implementing these algorithms is configurable by the user. Also, this plugin provides an effective configurable optimization based learner in RapidMiner. Optimization plugin also emphasises the fact, that many machine learning problems are in fact optimization problems. The advantage of various efficient optimization techniques allows us to solve these machine learning problems efficiently as well.

References

[1] Jorge Nocedal and Stephen J. Wright: *Numerical optimization*, Springer, New York, 2006.

[2] Lin Xiao *et al.*. 'Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization', Journal of Machine Learning Research 11

[3] Hauser, Raphael. "Line Search Methods for Unconstrained Optimisation ." Lecture 8. 2007.

[4] Jerome Friedman, Trevor Hastie and Robert Tibshirani. "A note on the group lasso and a sparse group lasso" Feb. 2010.