Assignment 10 for Statistical Computing and Empirical Methods

Dr. Henry WJ Reeve

Teaching block 1 2021

Introduction

This document describes your tenth assignment for Statistical Computing and Empirical Methods (Unit EMATM0061) on the MSc in Data Science. Before starting the assignment it is recommended that you first watch video lectures 24 and 25.

1 Basic concepts in regression

(Q) Write down your explanation of each of the following concepts. Give an example where appropriate.

1. A regression model

(A) A regression model is a map $\phi : \mathcal{X} \to \mathbb{R}$ from a feature space to a real valued output. If a regression model ϕ performs well, it will map a feature vector X taking values in the feature space $\mathcal{X}$ to an output $\phi(X)$ which should be close to a corresponding real-valued labels Y. As an example, consider a problem where our goal is to estimate the weight of a fish based Y based on the image of the fish X, represented as a vector.

2. Training data

(A) In the context of regression problems, the training data set $\mathcal{D}$ is a set of the form $((X_1, Y_1), \cdots, (X_n, Y_n))$ where each X_i is a feature vector taking values in the feature space and Y_i is an associated continuous label taking values in $\mathbb{R}$. Typically we assume that our regression problem corresponds to a probability distribution P on $\mathcal{X} \times \mathbb{R}$. Moreover, we typically assume that the pairs (X_i, Y_i) are distributed independently and identically distributed from P. In our example of predicting the weight of a fish, our data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ consists of sequence of pairs (X_i, Y_i) where X_i is the image of a fish and Y_i is its weight.

3. Mean squared error on the test data

(A) Suppose we have a regression problem with a distribution P over pairs (X, Y) where X is a feature vector and Y is a corresponding label. The mean squared error on the test data for a regression model $\phi : \mathcal{X} \to \mathbb{R}$ is the quantity,

$$\mathbb{E}_{(X,Y)\sim \mathrm{P}} \left[(\phi(X) - Y)^2 \right].$$

This is often referred to as the "out-of-sample" prediction error. We often estimate this by computing the average squared error on a specific test data set. This a common objective for regression problems. However, it cannot be computed from the training data alone.

4. Mean squared error on the training data

(A) Suppose we have a training data set $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ and a regression model $\phi : \mathcal{X} \to \mathbb{R}$, the mean squared error on the training data $\mathcal{D}$ is the empirical average,

$$\hat{\mathcal{R}}_{\mathrm{MSE}}(\phi) = \frac{1}{n} \sum_{i=1}^{n} \left(\phi(X_i) - Y_i\right)^2 .$$

Unlike the test error this can be computed directly based on the training data. However, it is important to be aware that the training error and testing error of a trained regression model $\hat{\phi} : \mathcal{X} \to \mathbb{R}$ can be very different, especially if $\hat{\phi}$ is trained to minmise the mean squared error on the training data.

5. Supervised learning

(A) Supervised learning is the general challeng of trying to learn a function $\phi : \mathcal{X} \to \mathcal{Y}$ from a feature space $\mathcal{X}$ to an output space $\mathcal{Y}$. In supervised learning, this is done via a labelled training data set $((X_1, Y_1), \cdots, (X_n, Y_n))$. Special cases include classification, where $\mathcal{Y}$ is a finite set of discrete class labels, and regression, where $\mathcal{Y} = \mathbb{R}$.

6. Linear regression model

(A) A linear regression model is a regression model $\phi_{w,w^0} : \mathbb{R}^d \to \mathbb{R}$ of the form

$$\begin{aligned} \phi_{w,w^0}(x) &= w^1 \cdot x^1 + \cdots + w^d \cdot x^d + w^0 \\ &= x \; w^\top + w^0, \end{aligned}$$

where $w = (w^1, \cdots, w^d) \in \mathbb{R}^d$ is a weight vector and $w^0 \in \mathbb{R}$ is a bias vector.

2 Basic concepts in regularisation

(Q) Write down your explanation of each of the following concepts. Give an example where appropriate.

1. Regularisation

(A) Regularisation refers to the general technique within supervised learning of modifying an objective in some way so as to reduce the gap between test error and training error. Typically, this will increase the error on the training data. However, by reducing the gap between test and training error can often improve performance. Examples include ℓ_2 regularisation in the context of ridge regression.

2. Hyper-parameter

(A) Hyper-parameter is a parameter that is not optimised by the learning algorithm itself, but is used to control the overall learning process. For example the parameter λ in ridge regression is a hyper-parameter which controls the degree of regularisation. Note that if we tuned λ to minimise the regularised loss function itself we would always end up setting $\lambda = 0$.

3. The Euclidean norm

(A) Let $a = (a_1, \cdots, a_d) \in \mathbb{R}^d$ be a vector. The Euclidean norm is given by

$$\|a\|_2 = \sqrt{(a_1)^2 + \cdots + (a_d)^2}.$$

Note that this is the "standard" notion of distance also referred to as the ℓ_2 norm.

4. The ℓ_1 norm

(A) Let $a = (a_1, \cdots, a_d) \in \mathbb{R}^d$ be a vector. The ℓ_1 norm is given by

$$\|a\|_1 = |a_1| + \cdots + |a_d|.$$

This is also known as the "Manhattan metric".

5. Validation data

(A) Validation data is a subset of the data - distinct from both the training data and the test data - which is used to select hyper-parameters controlling regularisation performance. That is we choose our regularisation hyper-parameters to minimse error on the validation data.

6. The train-validation-test split

(A) The train-validation-test split is a standard procedure for splitting a data set into three sub-samples.

(a) The train sample is used for training the model.
(b) The validation sample ise used for optimising hyper-parameters.
(c) The test sample is used for testing the performance of the optimised model in an unbiased way.

7. Ridge regression

(A) Ridge regression is an approach to learning a linear regression model $\phi_{w,w^0} : \mathbb{R}^d \to \mathbb{R}$ to minimise the regularised loss function

$$\begin{aligned}\hat{\mathcal{R}}_{0,\lambda}(\phi) &= \frac{1}{n}\sum_{i=1}^{n}\left(\phi_{w,w^0}(X_i) - Y_i\right)^2 + \lambda \cdot \|w\|_2^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(X_i\, w^\top + w^0 - Y_i\right)^2 + \lambda \cdot \|w\|_2^2,\end{aligned}$$

based on training data $((X_1, Y_1), \cdots, (X_n, Y_n))$.

8. The Lasso

(A) The Lasso is an approach to learning a linear regression model $\phi_{w,w^0} : \mathbb{R}^d \to \mathbb{R}$ to minimise the regularised loss function

$$\begin{aligned}\hat{\mathcal{R}}_{0,\lambda}(\phi) &= \frac{1}{n}\sum_{i=1}^{n}\left(\phi_{w,w^0}(X_i) - Y_i\right)^2 + \lambda \cdot \|w\|_1 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(X_i\, w^\top + w^0 - Y_i\right)^2 + \lambda \cdot \|w\|_1,\end{aligned}$$

based on training data $((X_1, Y_1), \cdots, (X_n, Y_n))$.

3 An investigation into ridge regression for high-dimensional regression

In this question we consider a high-dimensional regression problem. We consider a problem of predicting the melting point of a chemical compound from a relatively high-dimensional feature vector of chemical descriptors.

To do this we shall use data from the "QSARdata" data library. Begin by installing the "QSARdata" library as follows.

```
install.packages("QSARdata")
```

Next load the "QSARdata" library and loading the "MeltingPoint" data set.

```
library(QSARdata)
data(MeltingPoint)
```

You will find a data frame called "MP_Descriptors". The rows of the data frame correspond to different examples of chemical compounds and the coloumns correspond to various chemical descriptors. In addition you will find a vector call "MP_Outcome" which contains the corresponding melting point for each of the examples. Begin by combining the dataframe of feature vectors "MP_Descriptors" together with the coloumn vector of melting points "MP_Outcome". Combine these together into a single data frame entitled "mp_data_total" as follows.

```
mp_data_total<-MP_Descriptors%>%
  add_column(melting_pt=MP_Outcome)
```

(Q) How many variables are in your data frame? How many examples?

(A)

```
dim(mp_data_total)
```

```
## [1] 4401  203
```

There are 203 variables and 4401 examples.

(Q) Next carry out a train-validate-test split of the "mp_data_total" data frame. You should use about 50% of the data to train the algorithm, about 25% to validate and about 25% to train.

(A)

```
num_total<-mp_data_total%>%nrow()
num_train<-floor(0.5*num_total)
num_validate<-floor(0.25*num_total)
num_test<-num_total-num_train-num_validate

set.seed(1) # set random seed for reproducibility
mp_test_inds<-sample(seq(num_total),num_test)
# random sample of test indicies
mp_validate_inds<-sample(setdiff(seq(num_total),mp_test_inds),num_validate)
# random sample of validate inds
```

```
mp_train_inds<-setdiff(seq(num_total),union(mp_validate_inds,mp_test_inds))
# random sample of validate inds



mp_train_data<-mp_data_total%>%filter(row_number() %in% mp_train_inds) # train data
mp_validate_data<-mp_data_total%>%filter(row_number() %in% mp_validate_inds) # train data
mp_test_data<-mp_data_total%>%filter(row_number() %in% mp_test_inds) # test data



mp_train_x<-mp_train_data%>%select(-melting_pt) # train feature vector
mp_train_y<-mp_train_data%>%pull(melting_pt) # train labels

mp_validate_x<-mp_validate_data%>%select(-melting_pt) # validate feature vector
mp_validate_y<-mp_validate_data%>%pull(melting_pt) # validate labels

mp_test_x<-mp_test_data%>%select(-melting_pt) # train feature vector
mp_test_y<-mp_test_data%>%pull(melting_pt) # train labels
```

Our goal is to find a linear regression model $\phi_{w,w^0} : \mathbb{R}^d \to \mathbb{R}$ by $\phi_{w,w^0}(x) = w\, x^\top + w^0$ which estimates the melting point based upon the chemical descriptors.

(Q) Create a function which takes as input training data (a matrix of features for training data and a vector of labels for training data), validation data (a matrix of features for validation data and a vector of labels for validation data) and a hyper-parameter λ. The function should train a ridge regression model using the specified value of λ, then compute the validation error and output a single number corresponding to the validation error. Your function should not be specific to this particular regression problem, but should apply to ridge regression problems in general.

(A)

```
library(glmnet)

compute_validation_error_ridge<-function(train_x,train_y,validate_x,validate_y,lambda){

  glmRidge = glmnet(x=train_x, y=train_y, alpha = 0, standardize = TRUE,lambda=lambda)

  validate_y_est<-predict(glmRidge,newx=validate_x)

  validate_error = mean((validate_y-validate_y_est)^2)

  return(validate_error)

}
```

(Q) Next generate a sequence of candidate hyper-parameters called "lambdas". The sequence should begin with 10^{-8} and increase geometrically in multiples of 1.25 and should be of length 100. That is, "lambdas" should contain the numbers

$$10^{-8}, 10^{-8} \times 1.25, 10^{-8} \times 1.25^2, \cdots, 10^{-8} \times 1.25^{99}, 10^{-8} \times 1.25^{100}.$$

(A)

```
lambda_min=1e-8
lambdas=lambda_min*(1.25^seq(100))
```

(Q) Now use your function to estimate the mean squared error on the validation data for a ridge regression model for the problem of predicting the melting point based upon the chemical descriptors. Consider all of the hyper-parameter values within your vector "lambda". Store the results of this procedure in a data frame.

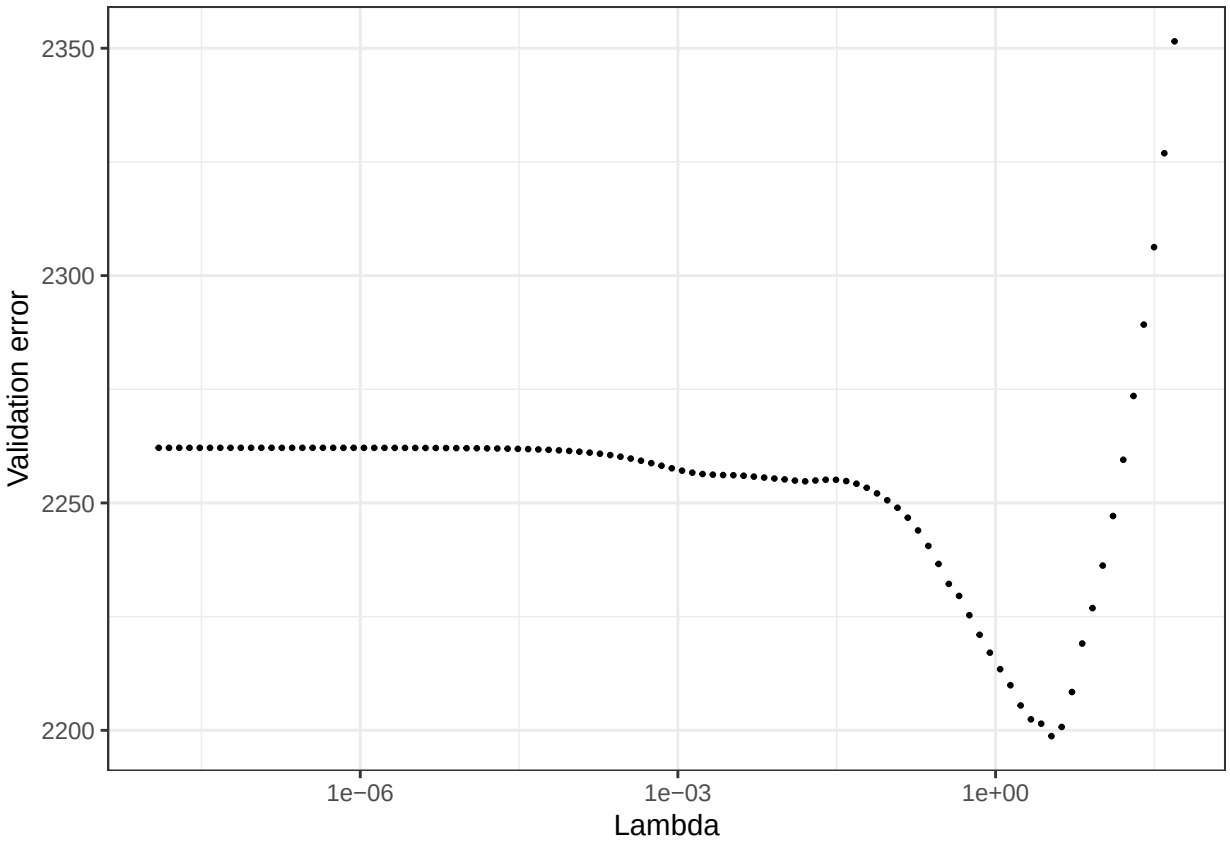
(A)

```
ridge_results_df<-data.frame(lambda=lambdas)%>%
  mutate(validation_error=map_dbl(lambda,
                                  ~compute_validation_error_ridge(
                                    train_x=mp_train_x%>%as.matrix()
                                    ,train_y=mp_train_y,
                                    validate_x=mp_validate_x%>%as.matrix(),
                                    validate_y=mp_validate_y,lambda=.x)))
```

(Q) Plot the validation error as a function of the hyper-parameter λ. Use the **scale_x_continuous()** function to plot the λ coordinate on a logarithmic scale.

(A)

```
ridge_results_df%>%
  ggplot(aes(x=lambda,y=validation_error))+
  geom_point(size=0.5)+theme_bw()+scale_x_continuous(trans='log10')+
  labs(x="Lambda",y="Validation error")
```



(Q) Now use your results data frame to determine the hyper-parameter λ with the lowest validation error.

(A)

```
min_val_error<-ridge_results_df%>%
  pull(validation_error)%>%min()

optimal_lambda<-ridge_results_df%>%
  filter(validation_error==min_val_error)%>%
  pull(lambda)%>%mean()


optimal_lambda
```

```
## [1] 3.373503
```

(Q) Retrain your ridge regression model with your selected value of the hyper-parameter λ and estimate the test error by computing the mean squared error on the test data. Does this lead to a biased estimate?

(A)

```
final_ridge_model = glmnet(x=mp_train_x%>%as.matrix(), y=mp_train_y,
                           alpha = 0,lambda=optimal_lambda)

mp_test_ridge_predicted_y<-predict(final_ridge_model,newx=mp_test_x%>%as.matrix())
```

```
ridge_test_error<-mean((mp_test_y-mp_test_ridge_predicted_y)^2)

ridge_test_error
```

```
## [1] 2179.659
```

Note that this is an unbiased estimate of performance on unseen data. Crucially, the test data was not used for either training the model or selecting the hyper-parameter.

(Q) Why can't we use the mean squared error on validation data for the ridge regression model with the selected hyper-parameter as an estimate of the mean squared error on test data?

(A) We selected the hyper-parameter to minimise error on the validation data. Hence, the error on the validation data for the optimised parameter is likely to under-estimate the error on unseen data.

(Q) Observe that the ridge regression model with the selected choice of λ has actually been trained twice: It was trained in order to compute the validation error, and then again to compute the test error. Comment on the computational efficiency of this procedure. Could it be easily improved? What memory implications would this have?

(A) Rather than retraining the same model we could store the trained parameters for each of the different values of λ. This would remove the requirement to retrain the model. Hence, this would lead to a reduction in the total time of the procedure. However, the memory cost would be greater.

4 Comparing ℓ_1 and ℓ_2 regularisation for logistic regression

As an optional extra consider regularised logistic regression. Our goal is to learn a linear classifier $\phi_{w,w^0} : \mathbb{R}^d \to \{0,1\}$ of the form $\phi_{w,w^0}(x) = \mathbf{1}\left\{w\, x^\top + w^0\right\}$.Suppose we have a set of training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ for a binary classification setting with feature vectors X_i taking values in $\mathbb{R}^d$ and binary labels Y_i taking values in $\{0,1\}$. The ℓ_2 regularised logistic regression minimises the following objective

$$\mathcal{R}^{\mathsf{bn}}_{0,\lambda}\left(\phi_{w,w^0}\right) = \frac{1}{n}\sum_{i=1}^{n}\left\{-\log \mathrm{S}\left((2Y_i - 1)\cdot(wX_i^\top + w^0)\right)\right\} + \frac{\lambda}{2}\|w\|_2^2,$$

over weights $w \in \mathbb{R}^d$ and a bias $w^0 \in \mathbb{R}$, where $\lambda > 0$ is a hyper-parameter and $\|w\|_2$ denotes the ℓ_2 norm (also) known as the Euclidean norm.

The ℓ_1 regularised logistic regression minimises the following objective

$$\mathcal{R}^{\mathsf{bn}}_{1,\lambda}\left(\phi_{w,w^0}\right) = \frac{1}{n}\sum_{i=1}^{n}\left\{-\log \mathrm{S}\left((2Y_i - 1)\cdot(wX_i^\top + w^0)\right)\right\} + \lambda\cdot\|w\|_1,$$

over weights $w \in \mathbb{R}^d$ and a bias $w^0 \in \mathbb{R}$, where $\lambda > 0$ is a hyper-parameter and $\|w\|_1$ denotes the ℓ_1 norm.

Regularisation by penalising an ℓ_1 or ℓ_2 norm is often referred to as "weight-decay".

(Q) Use your formulas for the derivatives to justify this name.

(A)

First let $\mathcal{R}^{\mathsf{bn}}\left(\phi_{w,w^0}\right)$ denote the negative log-likelihood cost given by

$$\hat{\mathcal{R}}\left(\phi_{w,w^0}\right) = \frac{1}{n}\sum_{i=1}^{n}\left\{-\log \mathrm{S}\left((2Y_i-1)\cdot(wX_i^\top + w^0)\right)\right\}.$$

Observe that

$$\mathcal{R}^{\mathsf{bn}}_{0,\lambda}\left(\phi_{w,w^0}\right) = \hat{\mathcal{R}}\left(\phi_{w,w^0}\right) + \lambda\cdot\frac{1}{2}\|w\|_2^2$$
$$\mathcal{R}^{\mathsf{bn}}_{1,\lambda}\left(\phi_{w,w^0}\right) = \hat{\mathcal{R}}\left(\phi_{w,w^0}\right) + \lambda\cdot\|w\|_1.$$

Based on our discussion from the previous lecture we have

$$\frac{\partial}{\partial w}\hat{\mathcal{R}}\left(\phi_{w,w^0}\right) = \frac{1}{n}\sum_{i=1}^{n}\left\{(1-2Y_i)\log \mathrm{S}\left((1-2Y_i)\cdot(wX_i^\top + w^0)\right)\right\}X_i.$$

Hence, to compute the derivatives it suffices to compute the derivatives

$$(\ell_2)\qquad \frac{\partial}{\partial w}\left\{\frac{1}{2}\|w\|_2\right\} \qquad\qquad (\ell_1)\qquad \frac{\partial}{\partial w}\left\{\|w\|_1\right\}.$$

In the case of ℓ_2 regularisation we have $\frac{\partial}{\partial w}\left\{\frac{1}{2}\|w\|_2\right\} = w$. Hence, it follows that

$$\frac{\partial}{\partial w}\mathcal{R}^{\mathsf{bn}}_{0,\lambda}\left(\phi_{w,w^0}\right) = \frac{\partial}{\partial w}\hat{\mathcal{R}}\left(\phi_{w,w^0}\right) + \lambda w.$$

To train our parameters we attempt to minimise our regularised loss function $\mathcal{R}^{\mathsf{bn}}_{0,\lambda}\left(\phi_{w,w^0}\right)$ by taking steps in the direction of the negative gradient,

$$w_{t+1} = w_t - \alpha\cdot\left\{\frac{\partial}{\partial w}\hat{\mathcal{R}}\left(\phi_{w,w^0}\right)\Big|_{w_t} + \lambda w_t\right\}.$$

Thus, we see that the effect of increasing λ is to modify the procedure to make steps in the negative direction of the weights. Hence, the weights are "decayed" towards zero. Note that the magnitude of the step depends linearly on the size of the weight's coordinates themselves.

The case of ℓ_1 regularisation is slightly more complex. First define a function $\mathrm{sign}:\mathbb{R}\to\{-1,0,1\}$ by

$$\mathrm{sign}(a) = \begin{cases} 1 & \text{if } a>0 \\ 0 & \text{if } a=0 \\ -1 & \text{if } a<0. \end{cases}$$

We extend sign to a function $\mathrm{sign}:\mathbb{R}^d\to\{-1,0,1\}^d$ by $(sign)(w) = (\mathrm{sign}(w_1),\cdots,\mathrm{sign}(w_d))$. Now we have $\frac{\partial}{\partial w}\left\{\|w\|_1\right\} = \mathrm{sign}(w)$. Hence, it follows that

$$\frac{\partial}{\partial w}\mathcal{R}^{\mathsf{bn}}_{1,\lambda}\left(\phi_{w,w^0}\right) = \frac{\partial}{\partial w}\hat{\mathcal{R}}\left(\phi_{w,w^0}\right) + \lambda\mathrm{sign}(w).$$

To train our parameters for the ℓ_1 method we attempt to minimise our regularised loss function $\mathcal{R}^{\mathsf{bn}}_{1,\lambda}\left(\phi_{w,w^0}\right)$ by taking steps in the direction of the negative gradient,

$$w_{t+1} = w_t - \alpha\cdot\left\{\frac{\partial}{\partial w}\hat{\mathcal{R}}\left(\phi_{w,w^0}\right)\Big|_{w_t} + \lambda\mathrm{sign}(w_t)\right\}.$$

Thus, we see that the effect of increasing λ is to modify the procedure to make steps in the negative direction of the weights. Hence, the weights are again "decayed" towards zero. However, now the magnitude of the step no longer depends on the size of the weight's coordinates themselves.

(Q) What does it mean for a high-dimensional vector $w \in \mathbb{R}^d$ to be sparse?

(A) A high-dimensional vector $w = (w^1, \cdots, w^d) \in \mathbb{R}^d$ is said to be sparse if there is some small subset $\mathcal{S} \subseteq \{1, \cdots, d\}$ with $|\mathcal{S}| = s \ll d$ such that for all $j \notin \mathcal{S}$ we have $w_j = 0$.

(Q) Do the formulas for the derivatives help explain why ℓ_1 regularisation is more likely to give rise to sparse solutions than ℓ_2 regularisation?

(A) The key point here is that the magnitude of the negative step in the ℓ_1 form of regularisation depends only upon the sign of the weights and not on their magnitude. This encourages the weights to be sparse by continuing to make steps towards zero, even when coordinates are already very small in magnitude. Conversely, for ℓ_2 regularisation the magnitude of the step reduces with the magnitude of the coordinates, making it much less likely for the weights to actually become sparse.

You can learn more about the the glmnet approach to logistic regression here.