Amey Mahajan

Briton Park

STAT 251

The Exponential Screening Estimator:

Implementation, Analysis and Application

**Introduction**

We study the linear regression problem in high dimensional settings with the assumption that the linear combination of covariates is sparse. Several methods have been developed in order to cope with this issue, including LASSO regression and forward stepwise regression. Rigollet and Tsybakov (2010) proposed the exponential screening (ES) estimator, which results in a sparse combination of approximating functions where sparsity is defined as the number of zero entries (L0 norm).

For this project, we implement the ES algorithm using a Metropolis-Hastings approximation, analyze its convergence, and compare its MSE in various settings against ordinary least squares (OLS), LASSO regression, and forward stepwise regression using synthetic data.

We originally had hoped to apply our implementation to a stock market dataset generated by the machine-learning based hedge fund, Numer.ai. However, the data they provide are 51 columns of continuous predictors being used to calculate a binary output (whether or not the stock returns more than its trading cost). Because Exponential Screening is designed to be built out of OLS estimators, not logistic regression estimators, we move to another application.

Our new dataset of choice is the famous Boston dataset, which explores median housing values in suburbs of Boston. We first examine the performance of our estimator on the pure dataset against LASSO. Then, we perturb the dataset by concatenating several columns of random noise (extending the number of columns) to test the power of the ES estimator. If the promise of theory holds true, it should drop the coefficients on those columns nearly to zero, while keeping higher coefficients on the “true” predictors, thereby creating a sparse estimator that reduces the “junk” that we have created -- accomplishing the goal of ES.

**Model**

We start with a collection of couples Z = {(x1, y1), … , (xn, yn)} where (xi*,*yi) ∈ IRM x IR, xi are the covariates, and yi are the response values. We assume the regression model yi = g(xi) + ξi for i ∈ (1,...,n) where g: IRM → IR and ξi are random errors following the Gaussian distribution with mean 0 and standard deviation 1. Let H = {f1,..., fM} be a dictionary of given approximating functions functions such that f: IRM→ IR. The ES algorithm approximates the regression function using a sparse linear combination fθ = θ1f1 + … + θMfM with weights θi. Each θ is computed according to the regression problem below using the design matrix, denoted by X, where X**i,j =** fj(xi).

To capture the sparsity of the estimator we use p ∈ *P* := {0,1}M, a sparsity pattern which may be thought of as an M-dimensional hypercube. Using p we can indicate the presence (pj = 1) and absence (pj = 0) of a covariate indexed by j ∈ (1,...,M). We let |p| denote the number of ones in the sparsity pattern and IRp denote the space defined by {θ · p : θ ∈ IRM} ⊂ IRM where θ · p ∈ IRM denotes the Hadamard product defined by the vector (θ · p)j = θjpj , j = 1,..., M. Note that for any p ∈ *P*, the least squares estimator θp ∈ IRp and is defined by θp = argminθ ∈IRM |Y − Xθ|22.

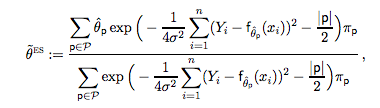
While running the Metropolis-Hastings method, the algorithm uses the following prior for each θp.



R denotes the rank of the design matrix X and H is a normalization constant given by:



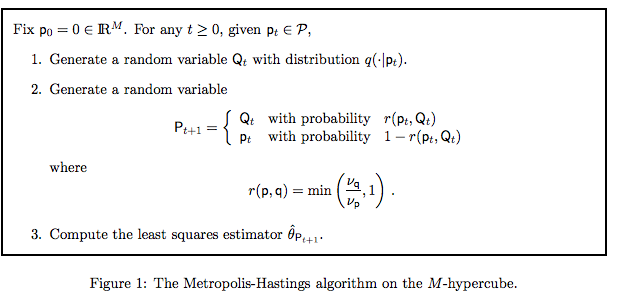
Using this model and notation, the ES solution is given by:



**Metropolis Hastings Approximation**

In order to precisely compute the ES estimator, 2R - 1 individual least squares estimators must be computed. Due to this large computational cost, the Metropolis Hastings algorithm (MH) can be used to efficiently approximate the ES estimator. The MH algorithm uses the aforementioned notion of a hypercube, which is a graph *G* with vertices given by the sparsity patterns in *P*. Each of the 2M sparsity patterns in *P* is connected to all other patterns with a different value in exactly one entry. In other words, each p ∈ *P* has neighbors q ∈ *P,* such that |p - q| = 1. The instrumental distribution q(.| p) is defined by the uniform distribution on each p ∈ *P*. Because each vertex or sparsity pattern has the same number of neighbors, q(p| q’) = q(q| p) for p, q ∈ P.

Using this formulation, the MH algorithm to compute the ES estimator is given by:



\*Rigollett and Tsybakov (2010)

where



and



with *w* = |q| - |p| ∈ {-1, 1} (meaning p and q are neighbors).

Using the least squares estimators, the ES estimator is computed:



with a predetermined, fixed T0 and T.

We approximate the ES estimator using the Metropolis Hastings algorithm in Python (see Appendix).

**Experiments and Discussion**

**A) Rigollet and Tsybakov (2010) experiment**

To test our implementation of the Metropolis-Hastings algorithm to approximate ES, we ran the same experiments as Rigollet and Tsybakov (2010). While Rigollet and Tsybakov (2010) ran each experiment 500 times, we ran the experiments once due to our time constraints.

The first model considered in the experiment was Y = Xθ\* + σ\*ξ where X is an n by M matrix of standard Gaussian entries and ξ ∈ IRn is a vector of standard Gaussian random variables, which is independent of X. The second model considered was of the same formulation, but instead using the Rademacher distribution to fill in the entries of X instead of the standard Gaussian distribution.

For both experiments, the chosen vector for θ\*was given by θ\*= 1I(*j* ≤ *S*) for a fixed *S*. The variance chosen was σ2 = S/9, following Candes and Tao (2007). (n, M, S) was set to be (100, 200, 10) and (200, 500, 20), while T0 and T were set to be 3000 and 7000, respectively.

The error was calculated for each experiment, following the authors’ methodology, using X(θ − θ\*)|22 / n where θ is the estimator output from the Metropolis-Hastings approximation of the ES estimator.

Table 1: Rigollet and Tsybakov (2010) experiment

|  |  |  |  |
| --- | --- | --- | --- |
| (n, M, S) | Distribution | Error\* (mean ± standard error) | Error (our implementation) |
| (100, 200,10) | Gaussian | 0.12 ± 0.07 | 0.17 |
| (100, 200,10) | Rademacher | 0.12 ± 0.06 | 0.16 |
| (200, 500, 20) | Gaussian | 0.24 ± 0.10 | 0.26 |
| (200, 500, 20) | Rademacher | 0.24 ± 0.09 | 0.21 |

\*Values reported by Rigollet and Tsybakov (2010)

**B) Synthetic data**

Synthetic data that we generated was used to test the effectiveness of the ES algorithm. We generated multiple datasets using y = X*B* + *w*. In this formulation, X is an *n* by *M* matrix with i.i.d. rows, *w* is an i.i.d. standard Gaussian vector, and *B* is the (true) regression vector such that *B* ∈ {0, 1}M. We considered the cases where (n, M) = {(50, 50), (25, 50), (100, 50)}.

For each of the (n, M) cases, we conducted trials with the Gaussian and the Rademacher distributions to fill in the rows of X. For the Gaussian distribution, we used N(0, 1), N(2, 2), N(-2, 0.5). We also varied the sparsity levels for *B*, where |*B*| = {5, 10, 20} for each dataset where the entries *i* for which *Bi =* 1 were the first |B| entries.

Overall, this resulted in 27 datasets using the Gaussian distribution and 9 datasets using the Rademacher distribution. We compared the ES estimator to ordinary least squares (OLS), forward stepwise regression and LASSO.

Table 2: Numerical Experiments (MSE) on the standard Gaussian distribution.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | p | σ | sparsity | μdata | σdata | OLS | Stepwise | LASSO | ES |
| 50 | 50 | 0.75 | 5 | 0 | 1 | 67.73 | 0.61 | **0.52** | 0.70 |
| 50 | 50 | 0.75 | 5 | 2 | 2 | 20.28 | **0.61** | 0.82 | 0.67 |
| 50 | 50 | 0.75 | 5 | -2 | 0.5 | 18.35 | 0.87 | 0.92 | **0.77** |
| 50 | 50 | 1.05 | 10 | 0 | 1 | 17.37 | 2.81 | 2.67 | **1.75** |
| 50 | 50 | 1.05 | 10 | 2 | 2 | 127.38 | 9.66 | 7.47 | **1.97** |
| 50 | 50 | 1.05 | 10 | -2 | 0.5 | 5.37 | **1.83** | 1.93 | 2.01 |
| 50 | 50 | 1.49 | 20 | 0 | 1 | 369.11 | **7.94** | 19.34 | 8.08 |
| 50 | 50 | 1.49 | 20 | 2 | 2 | 173.33 | 11.89 | 6.27 | **4.97** |
| 50 | 50 | 1.49 | 20 | -2 | 0.5 | 82.77 | **4.34** | 10.56 | 7.82 |
| 25 | 50 | 0.75 | 5 | 0 | 1 | 2.79 | 0.82 | **0.61** | 0.62 |
| 25 | 50 | 0.75 | 5 | 2 | 2 | 9.94 | **0.96** | 1.58 | 1.00 |
| 25 | 50 | 0.75 | 5 | -2 | 0.5 | 1.35 | **0.94** | 1.21 | 1.73 |
| 25 | 50 | 1.05 | 10 | 0 | 1 | 13.27 | **13.13** | 18.22 | 18.73 |
| 25 | 50 | 1.05 | 10 | 2 | 2 | 19.65 | **11.85** | 30.59 | 116.47 |
| 25 | 50 | 1.05 | 10 | -2 | 0.5 | 4.02 | **2.89** | 3.03 | 4.47 |
| 25 | 50 | 1.49 | 20 | 0 | 1 | 14.58 | **12.59** | 22.45 | 38.43 |
| 25 | 50 | 1.49 | 20 | 2 | 2 | 118.13 | **45.77** | 74.79 | 91.34 |
| 25 | 50 | 1.49 | 20 | -2 | 0.5 | 12.80 | **7.01** | 9.00 | 10.52 |
| 100 | 50 | 0.75 | 5 | 0 | 1 | 1.18 | **0.73** | 0.92 | **0.73** |
| 100 | 50 | 0.75 | 5 | 2 | 2 | 1.03 | **0.51** | 0.60 | **0.52** |
| 100 | 50 | 0.75 | 5 | -2 | 0.5 | 1.73 | **0.58** | 0.68 | **0.59** |
| 100 | 50 | 1.05 | 10 | 0 | 1 | 2.70 | 1.66 | 2.04 | **1.64** |
| 100 | 50 | 1.05 | 10 | 2 | 2 | 1.68 | 1.24 | 1.20 | **1.16** |
| 100 | 50 | 1.05 | 10 | -2 | 0.5 | 2.44 | 1.68 | **1.59** | 1.69 |
| 100 | 50 | 1.49 | 20 | 0 | 1 | 4.29 | 3.35 | 3.67 | **2.50** |
| 100 | 50 | 1.49 | 20 | 2 | 2 | 4.25 | 3.90 | 3.99 | **3.84** |
| 100 | 50 | 1.49 | 20 | -2 | 0.5 | 4.51 | 4.69 | 4.85 | **3.90** |

Table 3: Numerical Experiments (MSE) on the Rademacher Distribution.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n | p | σ | sparsity | OLS | Stepwise | LASSO | ES |
| 50 | 50 | 0.75 | 5 | 35.18 | 0.56 | 0.71 | **0.54** |
| 50 | 50 | 1.05 | 10 | 18.60 | **2.48** | 3.96 | 3.39 |
| 50 | 50 | 1.49 | 20 | 287.89 | **9.53** | 7.95 | 13.35 |
| 25 | 50 | 0.75 | 5 | 2.47 | **0.57** | 1.14 | 0.94 |
| 25 | 50 | 1.05 | 10 | 11.93 | 10.23 | 14.30 | **8.74** |
| 25 | 50 | 1.49 | 20 | 17.18 | **16.73** | 19.57 | 28.16 |
| 100 | 50 | 0.75 | 5 | 1.28 | **0.48** | 0.50 | 0.50 |
| 100 | 50 | 1.05 | 10 | 2.38 | 1.31 | 1.30 | **1.21** |
| 100 | 50 | 1.49 | 20 | 4.59 | 3.92 | 3.28 | **2.95** |

For each of the experiments, we implemented the Metropolis Hastings approximation for the ES estimator, ordinary least squares regression (OLS), LASSO regression with cross validation of the regularization parameter, and a variant of stepwise regression. Stepwise regression was done by first calculating an F score for each variable using linear regression and ordering the variables by *p*-values. Then a linear regression model was creating using the first *k* variables for *k* from 1 to M. The MSE for each linear model was computed and the minimum was taken to be the MSE for the stepwise regression model.

In the Gaussian setting, the ES method performs comparably with stepwise regression, adjusting for sampling error. We notice that stepwise dominates in the (n,M) = (25,50) case, while in the other cases, they perform comparably. In the Rademacher setting, the ES estimator again performs comparably with stepwise regression. Stepwise regression dominates in the (n, M) = (50, 50) and (n, M) = (25, 50) cases, while the ES estimator dominates in the (n, M) = (100, 50) case. This pattern extends to higher values of n, as well. We find that the advantage of ES over stepwise is that it amalgamates all sparsity patterns in the M-space, rather than outright dropping covariates, as stepwise regression does. This ensures that it takes account of all the provided data, which may prove important in practice. Even in cases where the ES method loses to stepwise regression, the results are fairly competitive and our implementation comes in second place for most of the above examples. Meanwhile, ES performs significantly better than both OLS and LASSO regressions.

**Convergence**

Convergence of the Metropolis Hastings approximation of ES estimator was analyzed using the dataset generated using y = X*B* + *w* where (n, M, S) = (50, 50,10), the entries of *w* are characterized by the standard Gaussian distribution, the rows of *X* are vectors drawn from the standard Gaussian distribution and θ\*= **1I**{*j* ≤ *S*}, and |*B*| = S with entries randomly chosen to be 1 and setting the other entries to be 0.

We applied the Metropolis Hastings algorithm and computed the L2 norm of the difference between the approximation from each step *t* and the previous step *t-1*. The results are plotted below.

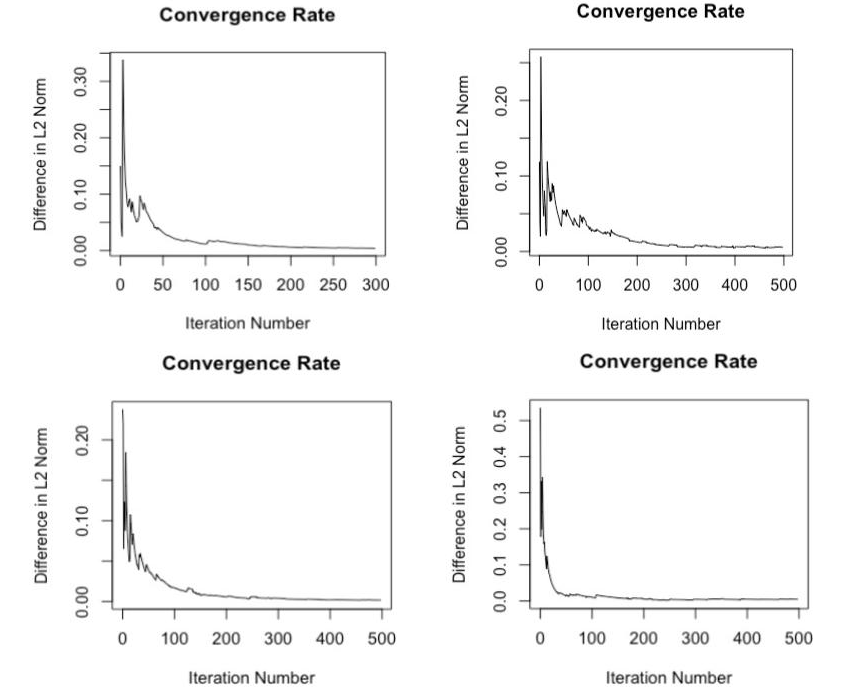


Figure 1: Convergence behavior of Metropolis-Hastings approximation using sparsity S = 10. Top-left: Standard Gaussian distribution, (n, M) = (50, 50). Top-right: Standard Gaussian distribution, (n, M) = (25, 50). Bottom-left: Rademacher distribution, (n, M) = (50, 50). Bottom-right: Rademacher distribution, (n, M) = (25, 50).

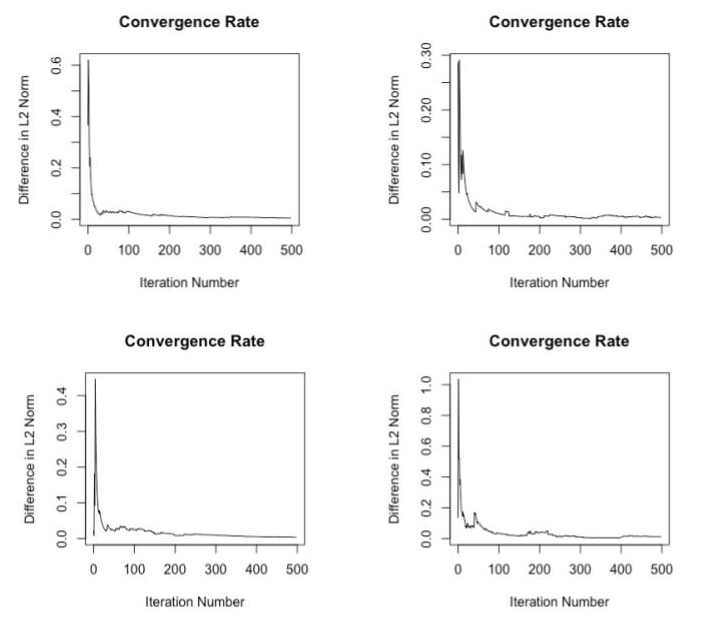


Figure 2: Convergence behavior of Metropolis-Hastings Approximation using sparsity S = 20. Top-left: Standard Gaussian distribution, (n, M) = (50, 50). Top-right: Standard Gaussian distribution, (n, M) = (25, 50). Bottom-left: Rademacher distribution, (n, M) = (50, 50). Bottom-right: Rademacher distribution, (n, M) = (25, 50).

From the Figures 1 and 2, it is evident that the convergence pattern is relatively robust across different types of distributions and both well- and ill-conditioned data matrices. We observed the same pattern in data matrices with higher n, M, and S. These results also held on fewer iterations of the Metropolis-Hastings approximation (e.g. 300 iterations rather than 500), but due to the rapid speed of the convergence, we see no reason not to use the higher number of iterations.

**Application**

We test our implementation on the famous Boston housing dataset. The data has 505 rows and 12 predictors used to estimate the median home value of various suburbs of Boston.

We apply the ES algorithm with 1000 iterations and a σ parameter of 0.05 (σchosen to be low because we know that all the columns should have significance in prediction). The ES estimator achieves an MSE of 21.99, while LASSO achieves an MSE 24.8. These serve as baseline scores on the unaltered dataset.

We progress by perturbing the dataset by adding columns of standard Gaussian noise. These columns, randomness aside, should have little to no correlation with the outcome variable - causing them to be dropped by the ES estimator and penalized by LASSO. We add 5, 10 and 15 columns of Gaussian noise and iterate over several values of the σ parameter as well.

Table 4: Boston Dataset Results

|  |  |  |  |
| --- | --- | --- | --- |
| Columns added | σ | ES | LASSO |
| 5 | 0.05 | **16.68** | 18.06 |
| 10 | 0.05 | **36.91** | 40.43 |
| 15 | 0.05 | 43.53 | **28.20** |
| 5 | 0.5 | **27.41** | 34.6 |
| 10 | 0.5 | 26.27 | **25.54** |
| 15 | 0.5 | **24.87** | 26.6 |
| 5 | 1 | 27.77 | **26.41** |
| 10 | 1 | 42.7 | **38.15** |
| 15 | 1 | 37.01 | **23.16** |
| 5 | 5 | 45.91 | **34.92** |
| 10 | 5 | **23.33** | 27.34 |
| 15 | 5 | 33.05 | **32.23** |

As is visible from the above table, both estimators perform reasonably when bogus noise columns are added to the dataset. Importantly, we find that it is critical to tune over possible values of σ - σ should be tuned over many values for each number of added junk data. We found that conducting this process improved the performance of the estimator, and would be a worthwhile future direction of work in any implementation of this estimator.

In total, our results generally support the claims made by Rigollet and Tsybakov that the Exponential Screening estimator is useful when attempting to achieve optimal rates of sparse estimation. In particular we point to our numerical experiments, in which the ES method performs comparably with or better than many well-known methods.