HW2

Jinhong Du - 12243476

April 18, 2020

Contents

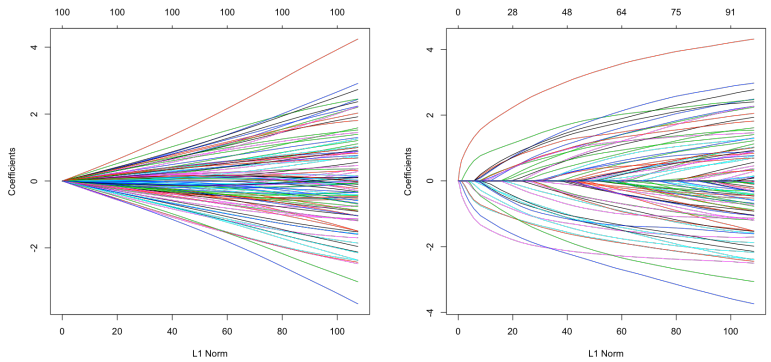
Problem A: Regularized Regression

The goal of this exercise is to familiarize yourself with the `glmnet` package, and ridge regression (L2-regularlization) and lasso (L1-regularization), by applying it to some simple simulated data.

i) Install the `glmnet` package and run the code in https://github.com/stephens999/stat34800/blob/master/analysis/glmnet_intro.Rmd. Add some additional code chunks of your own to investigate further and check your understanding. For example, you could:

- Simulate some independent test data from the same model and check that the prediction error of different methods is comparable with the CV results.
- Plot the (non-intercept) coefficients obtained from ridge regression and lasso against the true values used in the simulation and discuss the "shrinkage" that is occurring.
- Plot the estimated (non-intercept) coefficients against the "theoretical" expectations you would expect if the predictors were orthogonal, e.g. the "soft thresholding" property for the Lasso. (Note that the predictors here are not orthogonal, so the theory will certainly not hold precisely - does it hold approximately?) - Check that indeed the sum of absolute values of the coefficients is decreasing along the lasso path.
- When you have finished, write a brief summary of what the code is doing, what you examined, and what you learned.

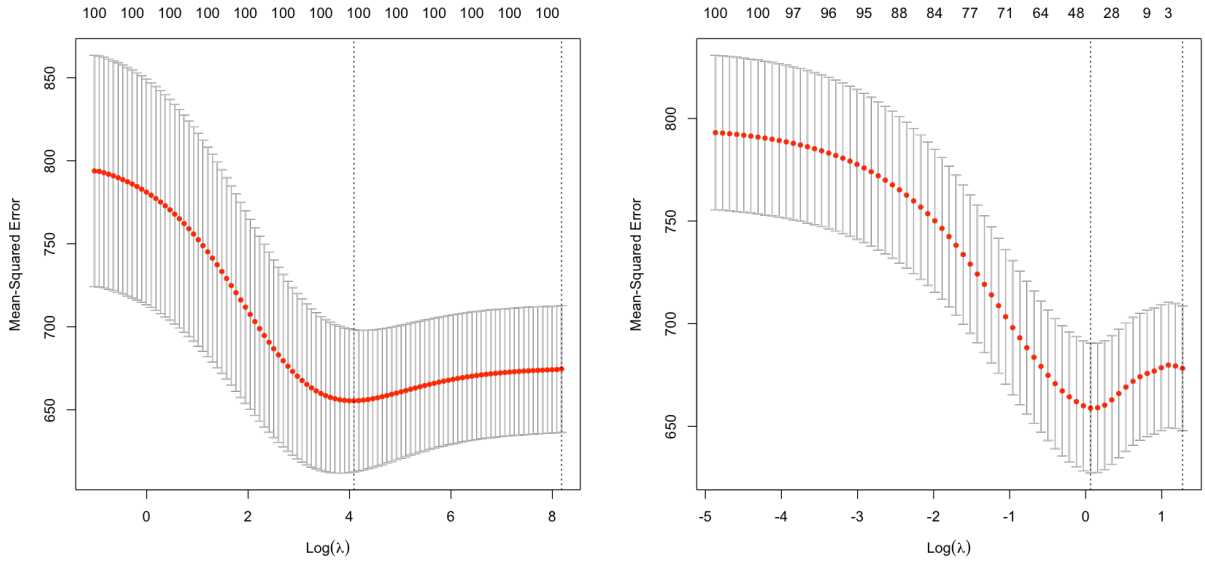
[1]:
```
library(glmnet)
# generate data
set.seed(1)
p <- 100
n <- 500
X <- matrix(rnorm(n*p),ncol=p)
b <- rnorm(p)
e <- rnorm(n,0,sd=25)
Y <- X %*% b + e
# fit ols, ridge regression and lasso, and see some basic plots
Y.ols = lm(Y~X)
Y.ridge = glmnet(X,Y,alpha=0)
plot(Y.ridge)
Y.lasso = glmnet(X,Y,alpha=1)
plot(Y.lasso)
```



Each curve corresponds to a variable. It shows the path of its coefficient against the l_1-norm of the whole coefficient vector as λ varies. The axis above indicates the number of nonzero coefficients at the current λ, which is the effective degrees of freedom (df) for the lasso. We can also annotate the curves by setting `label = TRUE` in the plot command.

[3]:
```
# Use cross-validation
library('lmvar')
cv.ols <- cv.lm(lm(Y~X, x=TRUE, y = TRUE))

cv.ridge <- cv.glmnet(X,Y,alpha=0)
plot(cv.ridge)
cat(cv.ridge$lambda.min, cv.ridge$lambda.1se)

cv.lasso <- cv.glmnet(X,Y,alpha=1)
plot(cv.lasso)
cat(cv.lasso$lambda.min, cv.lasso$lambda.1se)
```

```
59.58265 3571.886
1.065725 3.571886
```



Each plot includes the cross-validation curve (red dotted line), and upper and lower standard deviation curves along the λ sequence (error bars). Two selected λ's are indicated by the vertical dotted lines. `lambda.min` is the value of λ that gives minimum mean cross-validated error. The other λ saved is `lambda.1se`, which is the largest value of λ that gives the regularized model with error within one standard error of the minimum.

[4]:
```
# Extract coefficients from best cv fits.
b.ridge <- predict(Y.ridge, type="coefficients", s = cv.ridge$lambda.min)
b.lasso <- predict(Y.lasso, type="coefficients", s = cv.lasso$lambda.min)
b.ols <- as.vector(Y.ols$coefficients)
```

Note that the fits include an intercept. When regress Y on 1 without regularization, the coefficient of the intercept is equal to the mean of Y.

[5]:
```
length(b.lasso)
b.lasso[1]
b.ols[1]
lm(Y~1)$coefficients[1]
mean(Y)
```

```
101
-1.38243963081785
-1.37837157682698
-1.2339568943442
-1.2339568943442
```

[6]:
```
btrue = c(0,b) # Here the 0 is the intercept (true value 0)
sum((btrue-0)^2) # This is error if we just estimate 0 for everything and ignore␣
 ↪data. It is better than OLS!
sum((btrue-b.ols)^2)
sum((btrue-b.ridge)^2)
sum((btrue-b.lasso)^2)
```

```
85.3041069979209
138.271481192587
57.7274845383333
68.7211972516889
```

Next we simulate some independent test data from the same model and check that the prediction error of different methods is comparable with the CV results.

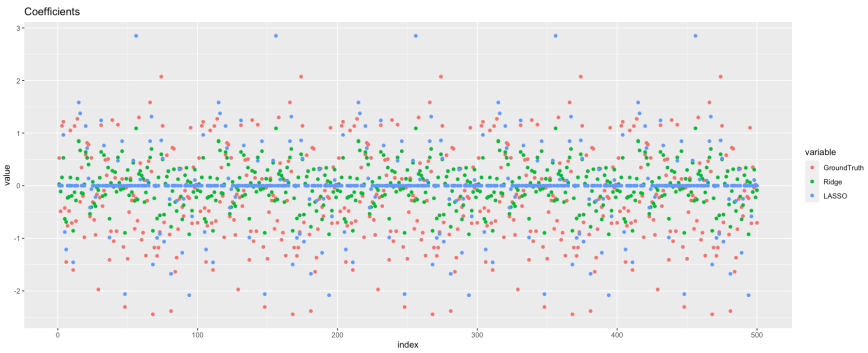
[7]:
```
# generate testing set
n_test <- 250
X_test <- matrix(rnorm(n_test*p),ncol=p)
e_test <- rnorm(n_test,0,sd=25)
Y_test <- X_test %*% b + e_test
# mean cross-validated error
cat(cv.ols$MSE$mean,
   cv.ridge$cvm[cv.ridge$lambda==cv.ridge$lambda.min],
   cv.lasso$cvm[cv.lasso$lambda==cv.lasso$lambda.min],'\n')

Y_test_pred.ridge <- predict(Y.ridge, newx = X_test, s = cv.ridge$lambda.min)
Y_test_pred.lasso <- predict(Y.lasso, newx = X_test, s = cv.lasso$lambda.min)
Y_test_pred.ols <- predict(Y.ridge, newx = X_test, s = 0)
cat(mean((Y_test-Y_test_pred.ols)^2), mean((Y_test-Y_test_pred.ridge)^2),␣
 ↪mean((Y_test-Y_test_pred.lasso)^2))
```

```
822.5425 655.4441 658.8492
734.4624 659.0557 676.1751
```

In the cross-validation, Ridge Regression and OLS achieve smallest and largest mean MSEs respectively. The performance of LASSO is slightly worse than Ridge Regression. Then we refit the models on the whole training set based on λ chosen from the CV results, and use these models to predict in the testing set. The results in the testing set are consistent to the CV results.

Next we plot the (non-intercept) coefficients obtained from ridge regression and lasso against the true values used in the simulation.

[8]:
```
library("ggplot2")
library("reshape2")
coefficients <- data.frame(GroundTruth=b,
                           Ridge=b.ridge[-1],
                           LASSO=b.lasso[-1],
                           index=c(1:500)
                          )
coefficients <- melt(coefficients, id="index")
options(repr.plot.width=15, repr.plot.height=6)
ggplot(data=coefficients,
       aes(x=index, y=value, colour=variable)) +
       geom_point() + ggtitle('Coefficients')
```

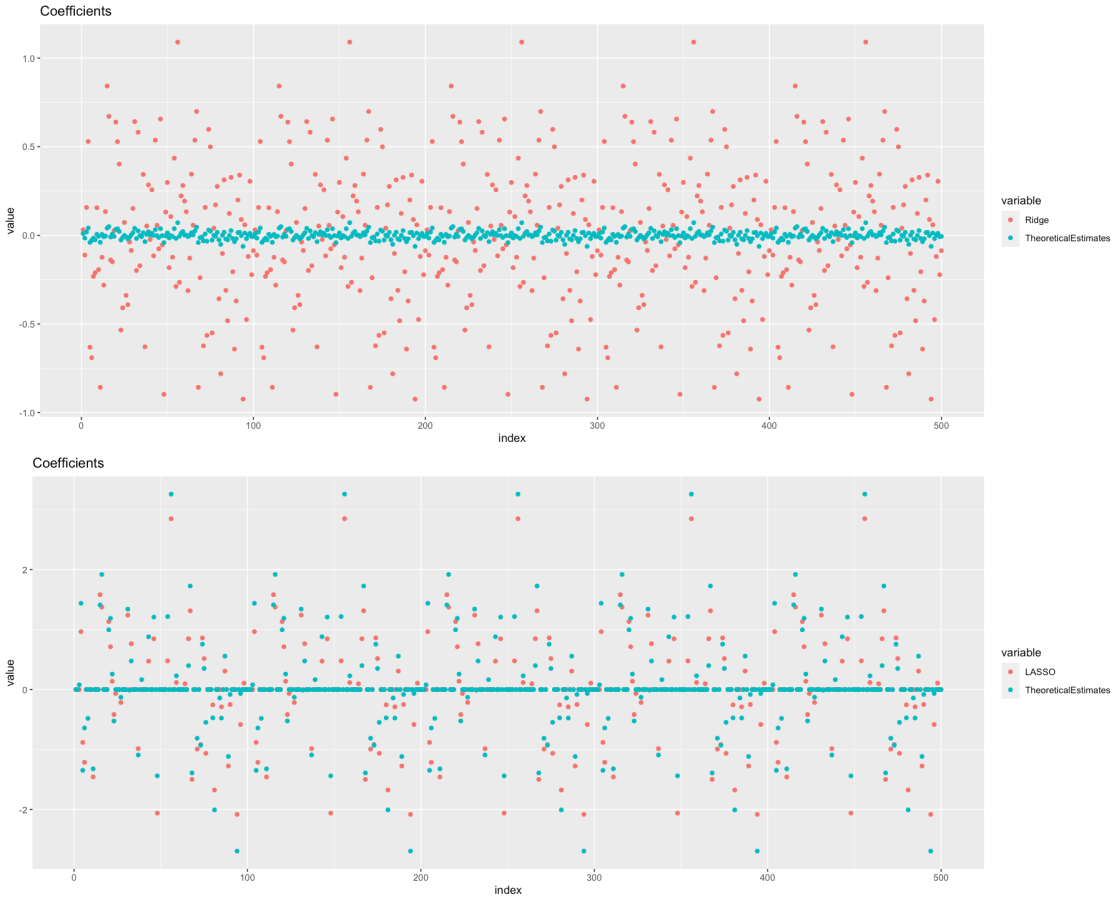


As we can see, the estimated coefficients of Ridge Regression and LASSO are shrank to zero in general. Specifically, most coefficients of LASSO are zero, that is the estimated coefficients are sparse for LASSO.

Let's look at the estimated (non-intercept) coefficients and the "theoretical" expectations if the predictors were orthogonal.

[9]:
```
coefficients <- data.frame(Ridge=b.ridge[-1],
                           TheoreticalEstimates=b.ols[-1]/(1+cv.ridge$lambda.min),
                           index=c(1:500)
                          )
coefficients <- melt(coefficients, id="index")
options(repr.plot.width=15, repr.plot.height=6)
ggplot(data=coefficients,
   aes(x=index, y=value, colour=variable)) +
   geom_point() + ggtitle('Coefficients')
```

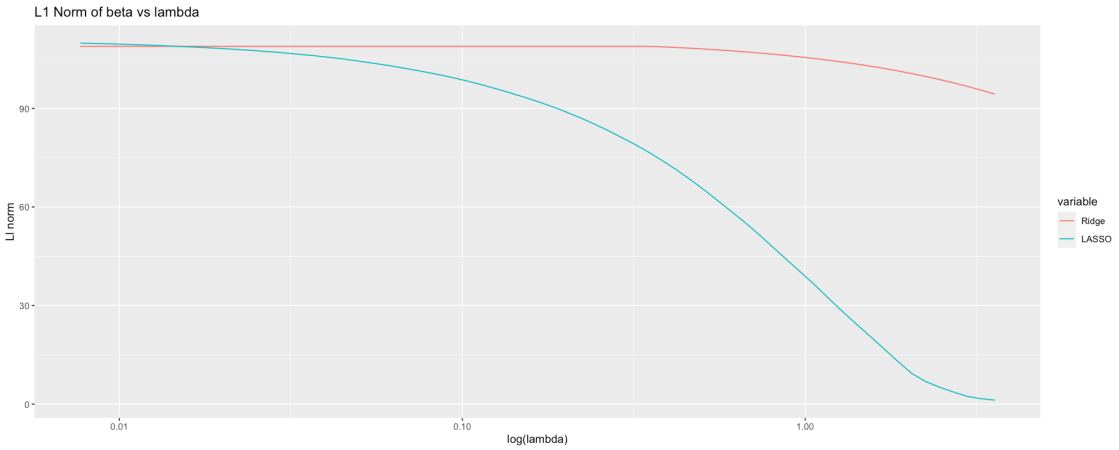
[10]:
```
coefficients <- data.frame(LASSO=b.lasso[-1],
                           TheoreticalEstimates=sign(b.ols[-1])*pmax(abs(b.
 ↪ols[-1])-cv.lasso$lambda.min, 0),
                           index=c(1:500)
                          )
coefficients <- melt(coefficients, id="index")
options(repr.plot.width=15, repr.plot.height=6)
ggplot(data=coefficients,
   aes(x=index, y=value, colour=variable)) +
   geom_point() + ggtitle('Coefficients')
```



As we can see, the theretical estimators of β is similar to the LASSO estimators. While the Ridge Regression estimators seem to have larger magnitude than the theoretical ones.

Finally, we plot the sum of absolute values of the coefficients along the lasso path.

[11]:
```
betas.ridge <- predict(Y.ridge, type="coefficients", s = cv.lasso$lambda)
betas.lasso <- predict(Y.lasso, type="coefficients", s = cv.lasso$lambda)
l1norm <- data.frame(Ridge=apply(abs(betas.ridge), 2, sum),
                     LASSO=apply(abs(betas.lasso), 2, sum), lamda=cv.lasso$lambda)
l1norm <- melt(l1norm, id="lamda")
ggplot(data=l1norm,
       aes(x=lamda, y=value, colour=variable)) +
       geom_line() + ggtitle('L1 Norm of beta vs lambda') +
       xlab('log(lambda)') + ylab('Ll norm') +
       scale_x_continuous(trans='log10')
```



Summary:

- I use `glmnet` to fit and compare results for OLS, Ridge Regression and LASSO.
- I use cross-validation to determine to best λ for Ridge Regression and LASSO respectively.
- I compare these with results in a new testing set. The testing results are similar to the CV results.
- I compare estimated coefficients of the three models. Compared to OLS, both Ridge Regression and LASSO shrink estimated coefficients to zero. Furthermore, LASSO induces sparsity.
- I compare Ridge estimators and LASSO estimators with theoretical estimators when the columns of X are orthonormal. The latter looks similar even though X is not orthonormal.
- I plot the L_1 norm of LASSO estimators versus λ. The sum of absolute values of the coefficients is decreasing along the lasso path.

ii) Note that the simulation in i) involves a non-sparse setting: every predictor has an effect on Y. This might be expected to favor ridge regression over Lasso since ridge regression tends to produce non-sparse solutions, whereas Lasso tends to produce sparse solutions. So now modify the simulation in i) to simulate a sparse scenario, where only 10 of the 100 predictors actually affect Y. (Note that you may or may not have to modify the residual variance to make the problem "not too easy" and "not too hard".) Investigate whether ridge regression or lasso provide better predictions in this setting.

First we generate the training and testing set. To make β sparse, we keep the first 10 entries of β and set the remains to be zero. Also, since the noises have very large variance, we multiply β by 10, which is equivalent to sample β from $\mathcal{N}(0, 100)$ and then make it sparse.

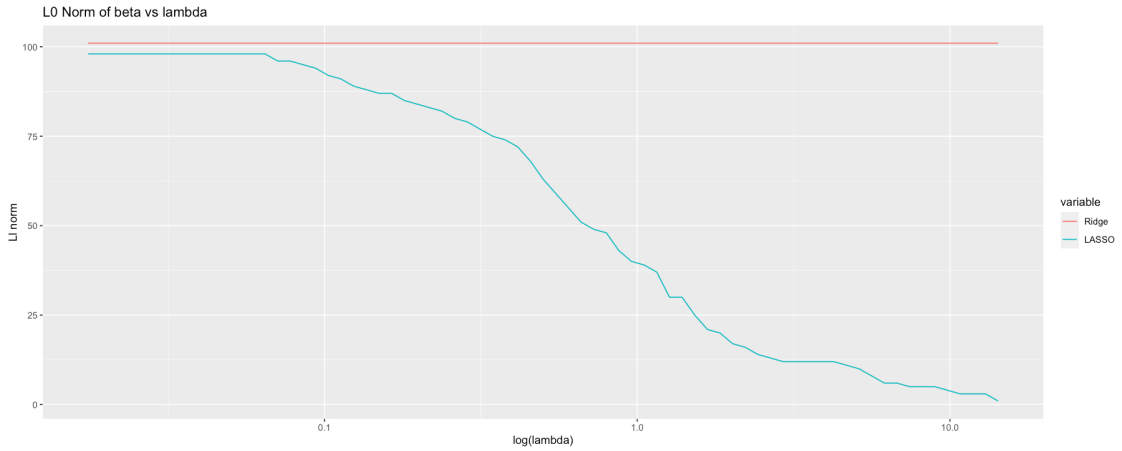
[12]:
```
set.seed(1)
p <- 100
n <- 500
X <- matrix(rnorm(n*p),ncol=p)
b <- rnorm(p)*10
b[-c(1:10)] <- 0
e <- rnorm(n,0,sd=25)
Y <- X %*% b + e
n_test <- 250
X_test <- matrix(rnorm(n_test*p),ncol=p)
e_test <- rnorm(n_test,0,sd=25)
Y_test <- X_test %*% b + e_test
```

[13]:
```
Y.ols = lm(Y~X)
Y.ridge = glmnet(X,Y,alpha=0)
Y.lasso = glmnet(X,Y,alpha=1)
cv.ols <- cv.lm(lm(Y~X, x=TRUE, y = TRUE))
cv.ridge <- cv.glmnet(X,Y,alpha=0)
cv.lasso <- cv.glmnet(X,Y,alpha=1)
b.ridge <- predict(Y.ridge, type="coefficients", s = cv.ridge$lambda.min)
b.lasso <- predict(Y.lasso, type="coefficients", s = cv.lasso$lambda.min)
b.ols <- as.vector(Y.ols$coefficients)
# mean cross-validated error
cat(cv.ols$MSE$mean,
   cv.ridge$cvm[cv.ridge$lambda==cv.ridge$lambda.min],
   cv.lasso$cvm[cv.lasso$lambda==cv.lasso$lambda.min],'\n')

Y_test_pred.ols <- predict(Y.ridge, newx = X_test, s = 0)
Y_test_pred.ridge <- predict(Y.ridge, newx = X_test, s = cv.ridge$lambda.min)
Y_test_pred.lasso <- predict(Y.lasso, newx = X_test, s = cv.lasso$lambda.min)
cat(mean((Y_test-Y_test_pred.ols)^2), mean((Y_test-Y_test_pred.ridge)^2),␣
 ↪mean((Y_test-Y_test_pred.lasso)^2))
```

```
792.5945 780.4756 655.5385
729.897 698.1997 668.735
```

LASSO achieves smallest MSEs in both training set and testing set.

[14]:
```
betas.ridge <- predict(Y.ridge, type="coefficients", s = cv.lasso$lambda)
betas.lasso <- predict(Y.lasso, type="coefficients", s = cv.lasso$lambda)

l1norm <- data.frame(Ridge=apply(betas.ridge!=0, 2, sum),
                     LASSO=apply(betas.lasso!=0, 2, sum),
                    lamda=cv.lasso$lambda
                    )
l1norm <- melt(l1norm, id="lamda")
ggplot(data=l1norm,
       aes(x=lamda, y=value, colour=variable)) +
       geom_line() + ggtitle('L0 Norm of beta vs lambda') +
       xlab('log(lambda)') + ylab('Ll norm') +
       scale_x_continuous(trans='log10')
```



As expected, the number of nonzero entries of $\hat{\beta}$ of LASSO decreases as λ inceases. While for Ridge Regression, almost all entries of $\hat{\beta}$ are nonzero. Therefore, LASSO performs better in sparsity setting.

Problem B: Conjugate Bayesian inference problems

Preparation: read through the introductory vignettes on fiveMinuteStat:

- https://stephens999.github.io/fiveMinuteStats/bayes_beta_binomial.html.
- https://stephens999.github.io/fiveMinuteStats/bayes_conjugate.html.

i) Do the exercise at the end of https://stephens999.github.io/fiveMinuteStats/bayes_conjugate.html, i.e. show that the Gamma distribution is conjugate for estimating a Poisson rate.

Proof. Suppose that $X|\mu \sim \text{Poisson}(\mu)$ and $\mu \sim \Gamma(\alpha, \beta)$, then

$$f_{X|\mu}(x) = \frac{\mu^x e^{-\mu}}{x!},$$
$$f_{\mu}(t) = \frac{\beta^\alpha}{\Gamma(\alpha)} t^{\alpha-1} e^{-\beta t}.$$

So

$$\begin{aligned} f_{\mu|X}(\mu) &\propto f_{X|\mu}(X|\mu) f_{\mu}(\mu) \\ &\propto \mu^{(\alpha+X)-1} e^{-(\beta+1)\mu}, \end{aligned}$$

i.e.,

$$\mu|X \sim \text{Gamma}(\alpha + X, \beta + 1).$$

Furthermore, if we are given multiple observation

$$X_1, \ldots, X_n \overset{iid}{\sim} \text{Poisson}(\mu),$$

then

$$\mu|X_1, \ldots, X_n \sim \text{Gamma}(\alpha + \sum_{i=1}^{n} X_i, \beta + n).$$

□

ii) Suppose you observe data $X \sim \mathcal{N}(\theta, 1/\tau)$ (so τ is the inverse of the variance, also known as the "precision"). Assume a prior distribution for θ of $\theta \sim \mathcal{N}(\mu_0, 1/\tau_0)$.

- Show that the posterior distribution for θ is $\theta|X \sim \mathcal{N}(\mu_1, 1/\tau_1)$ where
  - $\mu_1 = wX + (1-w)\mu_0$
  - $\tau_1 = \tau + \tau_0$
  - $w = \tau/\tau_1$
- Explain how this can be interpreted as providing a "shrinkage" estimate of θ, with shrinkage towards the prior mean.
- Notes:
  - The posterior precision is the sum of the prior precision and data precision, so collecting data always increases the precision of your information about θ.
  - The posterior mean, μ_1, has a very intuitive form, which is the weighted sum of the prior mean and the data, where the weight depends on the precisions. If the data are very precise compared with the prior then the data dominates. If the data are very imprecise then the prior dominates.
  - Working with the precision, instead of the variance, is a standard trick in Bayesian inference to make the algebra a bit easier to manage.

Proof. Since $X|\theta \sim \mathcal{N}(\theta, \frac{1}{\tau})$ and $\theta \sim \mathcal{N}(\mu_0, \frac{1}{\tau_0})$, we have

$$f_{X|\theta}(x) = \sqrt{\frac{\tau}{2\pi}} e^{-\frac{\tau}{2}(x-\theta)^2}$$
$$f_{\theta}(t) = \sqrt{\frac{\tau_0}{2\pi}} e^{-\frac{\tau_0}{2}(t-\mu_0)^2}.$$

So

$$\begin{aligned} f_{\theta|X}(\theta) &\propto f_{X|\theta}(X) f_{\theta}(\theta) \\ &\propto e^{-\frac{\tau}{2}(X-\theta)^2 - \frac{\tau_0}{2}(\theta-\mu_0)^2} \\ &\propto e^{-\frac{\tau+\tau_0}{2}\theta^2 + (\tau X + \tau_0\mu_0)\theta} \\ &\propto e^{-\frac{\tau+\tau_0}{2}\left(\theta - \frac{\tau X + \tau_0\mu_0}{\tau+\tau_0}\right)^2} \end{aligned}$$

which is a normal distribution with mean $\mu_1 = \frac{\tau}{\tau+\tau_0}X + \frac{\tau_0}{\tau+\tau_0}\mu_0$ and precision $\tau_1 = \tau + \tau_0$. Therefore, $\theta|X \sim \mathcal{N}(\mu_1, \frac{1}{\tau_1})$.

As we observe more and more data, e.g. $X_1, \ldots, X_n$, then $\theta|X_1, \ldots, X_n \sim \mathcal{N}\left(\mu_n, \frac{1}{\tau_n}\right)$ where $\mu_n = w_n \sum_{i=1}^n X_i + (1-w_n)\mu_0$, $\tau_n = n\tau + \tau_0$ and $w_n = \frac{\tau}{\tau_n}$. Notice that the posterior mean is a weighetd sum of the prior mean and the data. Since the precision τ_n gets increasing as we observe more data, the weight of the data w_n gets decreasing as n increases. Therefore, μ_n are shrunk to the prior mean μ_0 as n increases. □

Problem C: Empirical Bayes Shrinkage

The goal here is to implement Empirical Bayes shrinkage for the normal means problem with normal prior: $X_j|\theta_j, s_j \sim N(\theta_j, s_j^2)$ with $\theta_j|\mu,\sigma \sim N(\mu,\sigma^2)$, $j = 1,\dots,n$.

a) Derive an expression for the log-likelihood $l(\mu,\sigma) = \sum_j \log p(X_j|\mu,\sigma,s_j)$.

Since

$$\begin{aligned}
p(X_j|\mu,\sigma,s_j) &= \int_{\mathbb{R}} p(X_j,\theta_j|\mu,\sigma,s_j)\mathrm{d}\theta_j \\
&= \int_{\mathbb{R}} p(X_j|\theta_j,s_j)p(\theta_j|\mu,\sigma)\mathrm{d}\theta_j \\
&= \int_{\mathbb{R}} \frac{1}{2\pi s_j\sigma} e^{-\frac{1}{2s_j^2}(X_j-\theta_j)^2} e^{-\frac{1}{2\sigma^2}(\theta_j-\mu)^2}\mathrm{d}\theta_j \\
&= \frac{1}{2\pi s_j\sigma} e^{-\frac{1}{2s_j^2}X_j^2-\frac{1}{2\sigma^2}\mu^2+\frac{1}{2s_j^2\sigma^2(s_j^2+\sigma^2)}(X_j\sigma^2+\mu s_j^2)^2} \int_{\mathbb{R}} e^{-\frac{1}{2}\frac{s_j^2+\sigma^2}{s_j^2\sigma^2}\left(\theta_j-\frac{X_j\sigma^2+\mu+s_j^2}{s_j^2+\sigma^2}\right)^2}\mathrm{d}\theta_j \\
&= \frac{1}{\sqrt{2\pi(s_j^2+\sigma^2)}} e^{-\frac{1}{(s_j^2+\sigma^2)}(X_j-\mu)^2}
\end{aligned}$$

i.e., $X_j|\mu,\sigma,s_j \sim \mathcal{N}(\mu, s_j^2+\sigma^2)$, the log-likelihood is given by

$$\begin{aligned}
l(\mu,\sigma) &= \sum_{j=1}^n \log p(X_j|\mu,\sigma,s_j) \\
&= \sum_{j=1}^n \left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(s_j^2+\sigma^2) - \frac{1}{2(s_j^2+\sigma^2)}(X_j-\mu)^2\right]
\end{aligned}$$

Setting

$$\begin{aligned}
\frac{\partial l}{\partial \mu} &= -\sum_{j=1}^n \frac{1}{s_j^2+\sigma^2}(\mu - X_j) = 0 \\
\frac{\partial l}{\partial \sigma^2} &= \sum_{j=1}^n \left[-\frac{1}{2(s_j^2+\sigma^2)} + \frac{1}{(s_j^2+\sigma^2)^2}(X_j-\mu)^2\right] = 0,
\end{aligned}$$

yields the MLEs. While the MLEs have no explicit expression, we can use numerical optimization algorithm to solve for approximate solutions.

b) Write down the expression for the posterior mean $E(\theta_j|X_j,\mu,\sigma)$.

Since

$$\begin{aligned} p(\theta_j|X_j,\mu,\sigma) &\propto p(X_j|\theta_j)p(\theta|\mu,\sigma) \\ &\propto e^{-\frac{1}{2s_j^2}(X_j-\theta_j)^2-\frac{1}{2\sigma^2}(\theta_j-\mu)^2} \\ &\propto e^{-\frac{1}{2}\frac{s_j^2+\sigma^2}{s_j^2\sigma^2}\left[\theta_j-\left(\frac{\sigma^2}{s_j^2+\sigma^2}X_j+\frac{s_j^2}{s_j^2+\sigma^2}\mu\right)\right]^2} \end{aligned}$$

i.e., $\theta_j|X_j,\mu,\sigma \sim \mathcal{N}\left(\frac{\sigma^2}{s_j^2+\sigma^2}X_j+\frac{s_j^2}{s_j^2+\sigma^2}\mu, \frac{s_j^2\sigma^2}{s_j^2+\sigma^2}\right)$, the posterior mean is given by

$$\mathbb{E}(\theta_j|X_j,\mu,\sigma) = \frac{\sigma^2}{s_j^2+\sigma^2}X_j+\frac{s_j^2}{s_j^2+\sigma^2}\mu.$$

c) Use these expressions to implement a function `ebnm_normal` to do EB shrinkage. Your function should input $x=(x_1,\ldots,x_n)$ and $s=(s_1,\ldots,s_n)$ and output the maximum likelihood estimates $(\hat{\mu},\hat{\sigma})$ for μ,σ and the vector of posterior means $E(\theta_j|X_j,\hat{\mu},\hat{\sigma})$. [You will need to use a numerical optimizer, like the R function `optim`, to optimize $l(\mu,\sigma)$; it may be better to optimize over $\eta := \log\sigma$ to avoid the non-negative constraint on σ.]

First we need to define the negative log-likelihood function as follows.

[15]:
```
import numpy as np
from scipy.stats import norm
from scipy.optimize import minimize

def neg_log_likelihood(pars, x, s):
    '''
    Evaulate log-likelihood l(mu,sigma).
    Params:
        pars - params to be optimized
        x    - observation
        s    - posterior standard deviation of x
    Returns:
        -L   - negative log-likelihood
    '''
    mu = pars[0]
    sigma = np.exp(pars[1])
    L = np.sum(norm.logpdf(x, mu, np.sqrt(s**2+sigma**2)))
    return -L
```

Then we can define the `ebnm_normal` function to obtain Empirical Bayesian estimators by numerically minimizing the negative log-likelihood.

[16]:
```
def ebnm_normal(x,s):
    '''
    Params:
        x       - array of size [n,], observation
        s       - array of size [n,], posterior standrad deviation of x
    Returns:
        mu      - MLE of mu
        sigma   - MLE of sigma
        E_theta - posterior mean of theta_j
    '''
    res = minimize(neg_log_likelihood, np.zeros(2),
                        args=(x, s),
                        method='L-BFGS-B')
    mu, sigma = res.x
    sigma = np.exp(sigma)
    E_theta = (sigma**2*x+s**2*mu)/(sigma**2+s**2)
    return mu, sigma, E_theta
```

d) Demonstrate the code on a simulation to show that

i) the estimates $(\hat{\mu}, \hat{\sigma})$ are sensible compared with the true values μ, σ;

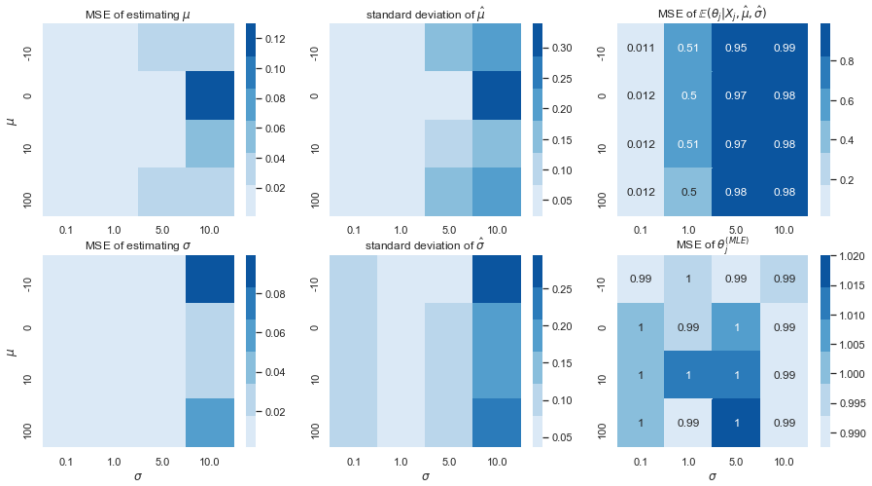
ii) the posterior means $\bar{\theta}_j := E(\theta_j | X_j, \hat{\mu}, \hat{\sigma})$ provide more accurate estimates of θ_j than do the maximum likelihood estimates $\hat{\theta}_j := X_j$.

We choose four values for μ and σ and each pair of (μ, σ) we simulate data for 10 times.

[17]:
```
np.random.seed(0)
mu_list = np.array([-10, 0, 10, 100])
sigma_list = np.array([0.1, 1, 5, 10])

mu_hat = np.zeros((4,4,10))
sigma_hat = np.zeros((4,4,10))
MSE_theta_posterior = np.zeros((4,4,10))
MSE_theta_MLE = np.zeros((4,4,10))
n = 1000
for i,mu in enumerate(mu_list):
for j,sigma in enumerate(sigma_list):
    for k in range(10):
        theta = np.random.normal(mu,sigma,n)
        s = np.ones(n)
        x = np.random.normal(theta,s,n)
        res = ebnm_normal(x,s)
        mu_hat[i,j,k] = res[0]
        sigma_hat[j,i,k] = res[1]
        MSE_theta_posterior[i,j,k] = np.mean((theta - res[2])**2)
        MSE_theta_MLE[i,j,k] = np.mean((theta - x)**2)
```

With the simulation results recorded, we are able to present the following plots.

[18]:
```
MSE_mu = np.mean((mu_hat-np.expand_dims(np.expand_dims(mu_list,-1),-1))**2, -1)
MSE_sigma = np.mean((sigma_hat-np.expand_dims(np.
 ↪expand_dims(sigma_list,-1),-1))**2,-1)
sd_mu = np.std(mu_hat, -1)
sd_sigma = np.std(sigma_hat, -1)
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()
fig, axes = plt.subplots(2,3,figsize=(15,8))
axes[0,0] = sns.heatmap(MSE_mu, cmap=sns.color_palette("Blues"),
            xticklabels=sigma_list, yticklabels=mu_list, ax=axes[0,0])
axes[0,0].set(ylabel='$\mu$', title='MSE of estimating $\mu$')
axes[1,0] = sns.heatmap(MSE_sigma.T, cmap=sns.color_palette("Blues"),
            xticklabels=sigma_list, yticklabels=mu_list, ax=axes[1,0])
axes[1,0].set(xlabel='$\sigma$', ylabel='$\mu$', title='MSE of estimating␣
 ↪$\sigma$')
axes[0,1] = sns.heatmap(sd_mu, cmap=sns.color_palette("Blues"),
            xticklabels=sigma_list, yticklabels=mu_list, ax=axes[0,1])
axes[0,1].set(title='standard deviation of $\hat{\mu}$')
axes[1,1] = sns.heatmap(sd_sigma.T, cmap=sns.color_palette("Blues"),
            xticklabels=sigma_list, yticklabels=mu_list, ax=axes[1,1])
axes[1,1].set(xlabel='$\sigma$', title='standard deviation of $\hat{\sigma}$')
axes[0,2] = sns.heatmap(np.mean(MSE_theta_posterior,-1), cmap=sns.
 ↪color_palette("Blues"),
          annot=True, xticklabels=sigma_list, yticklabels=mu_list, ax=axes[0,2])
axes[0,2].set(title='MSE of $\mathbb{E}␣
 ↪(\\theta_{j}|X_{j},\hat{\mu},\hat{\sigma})$')
axes[1,2] = sns.heatmap(np.mean(MSE_theta_MLE,-1),cmap=sns.
 ↪color_palette("Blues"),
          annot=True, xticklabels=sigma_list, yticklabels=mu_list, ax=axes[1,2])
axes[1,2].set(xlabel='$\sigma$', title='MSE of $\\theta_{j}^{(MLE)}$')
plt.show()
```



The first column of the above plots shows the average estimated error for μ and σ, respectively. The second column shows the standard deviation of $\hat{\mu}$ and $\hat{\sigma}$ for multiple simulations. The last column shows the MLE of using $\mathbb{E}(\theta_j|X_j, \hat{\mu}, \hat{\sigma})$ and X_j to estimate θ_j, respectively. As we can see,

1. The estimates $(\hat{\mu}, \hat{\sigma})$ are sensible compared with the true values. When σ is larger, the estimate errors of both $\hat{\mu}$ and $\hat{\sigma}$ are larger, and the volatility of them are larger as expected. When μ is more near 0, the estimated $\hat{\mu}$ are worse.
2. When σ is large, estimating θ_j by both $\mathbb{E}(\theta_j|X_j, \hat{\mu}, \hat{\sigma})$ and X_j are comparable. But when σ is small, the estimate error of the former is much smaller than the latter.

e) Apply your code to the "8-schools problem" at http://andrewgelman.com/2014/01/21/everything-need-know-bayesian-statistics-learned-eight-schools/.

[19]:
```
x = np.array([28,8,-3,7,-1,1,18,12])
s = np.array([15,10,16,11,9,11,10,18])
mu_hat, sigma_hat, E_theta_hat = ebnm_normal(x,s)
print(E_theta_hat)
```

```
[7.68578326 7.6857597  7.68574757 7.68575732 7.68572986 7.68574392
7.68578674 7.68576245]
```

We can see that Empirical Bayes Algorithm shrinks all the estimates to almost the same value, meaning the data are most consistent with no variation in effect.