Economics 691-06: Assignment 5

Nihar Shah

Solutions are marked in blue.

Problem: Shrinkage

The data scientists at Youtube have been running a series of week-long experiments to improve time spent on the platform. Specifically, the Youtube team has run fifty experiments over the last few months, each of which includes 2,000 users, split between 1,000 treated and 1,000 control users. For each experiment and for each user in that experiment, Youtube records the change in time spent on the website (in minutes) for that week, relative to the user's previous week. The data scientists have of course analyzed each experiment in isolation and using standard frequentist approaches; but they have asked you to explore if shrinkage analysis can potentially improve on the estimates of the treatment effect.

The raw data from these fifty experiments is in the file data_assignment5_1.csv. The file has 100,000 rows. Each row corresponds to a separate user in a separate experiment, for which there are three columns. The first column is the change in time spent on the website. The second column is an indicator for whether they are in the treated or control group. The third column indicates which experiment (1 - 50) that the user is in.

While the true treatment effects are generally unknown in practice, I have provided them in the file data_assignment5_2.csv to illustrate the improvements in the various techniques. This file has 50 rows. Each row corresponds to a separate experiment, for which there are two columns: the true treatment effect in the first column, and the experiment number in the second column. Note that this file should *only* be used for assessing the mean-squared error of your estimated treatment effects, and not for generating those estimates in the first place.

```
# Load libraries: numpy, pandas, linear regression, and Ridge / LASSO
import numpy as np
import pandas as pd
import statsmodels.formula.api as sm
from sklearn.linear_model import Ridge, Lasso

# Load in data and true parameters
from google.colab import files
import io
uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['data_assignment5_1.csv']))
uploaded = files.upload()
truth = pd.read_csv(io.BytesIO(uploaded['data_assignment5_2.csv']))
```

1. **(10 points)** Using the experimental data only, estimate the effect of each of the fifty experiments per standard (i.e. frequentist) regression analysis. Report the mean-squared error of your estimates across the fifty experiments, by comparing them to the true treatment effects.

   This is straightforward: we iterate through each experiment, and regress the outcomes on treatment. With our estimates in hand, we compute the mean-squared error versus the true treatment effects, and find a value of 1.06.

```
# Estimate OLS regression for a given experiment
def regression(experiment_number):
  i = np.where(data['experiment_number'] == experiment_number)[0]
  result = sm.ols(formula = 'outcome ~ treatment', data = data.iloc[i,]).fit()
  return(result.params[1])

# Get the MSE for OLS regression across all experiments
estimates = np.array([regression(number) for number in range(1, 51)])
print("The MSE for the frequentist regression analysis is: " +
      str(round(np.mean((estimates - truth['true_effect'])**2), 4)))

The MSE for the frequentist regression analysis is: 1.0606
```

2. **(15 points)** Using the James-Stein estimator, adjust your estimates from Question 1. Report the shrinkage factor that you are using. Then report the mean-squared error of your estimates, by comparing them to the true treatment effects.

   - As a hint, remember that you will have to compute the variance of the observed treatment effects, as drawn from the true treatment effect. To compute this, think about why the observed treatment effect can differ from the true treatment effect; and compute the variance accordingly.

   To get the James-Stein estimate, we need to find the appropriate shrinkage factor. Most of the shrinkage factor is standard, but we need to estimate σ^2. This is the variance of the observed treatment effect, as drawn from the true treatment effect. In other words, we need the variance of $e = \hat{\beta} - \beta$. This is not the same as the variance of $\hat{\beta}$, as each $\hat{\beta}$ has a different mean.

   One way to do this is to subtract the observed treatment effect from the true treatment effect, and estimate the variance of those residuals. That is illegal, however, as we are not supposed to use the true treatment effects for any purpose outside of assessing the estimates at the end of the problem. A second way to do this is to get the standard error from the linear regression models in Question 1, and average across experiments. This works as long as the standard errors is approximately constant across experiments. In this answer, I will illustrate a third approach, which is to solve from first principles. Each estimate is the difference in means between the treated and control group, and thus has the following specification:

$$\hat{\beta} = \frac{1}{1000} \sum_{i=1}^{1000} (\beta + \epsilon_i) - \frac{1}{1000} \sum_{i=1001}^{2000} \epsilon_i$$

   We can compute the variance of this easily:

$$\mathbb{V}(\hat{\beta}) = \left(\frac{1}{1000} + \frac{1}{1000} \right) \mathbb{V}(\epsilon_i)$$

In turn, the variance of the residuals can be estimated by estimating the variance in each treatment and control group for each experiment separately. (The two groups cannot be estimated together because they have different means.) So we estimate, for each experiment and each subgroup, the variance of the outcomes; and that value (which is approximately 400 in each split) is fed into the expression above to get the variance of the treatment effects.

From that point, we can apply the regular James-Stein estimator and compute the mean-squared error. We get a shrinkage factor of 0.73, and find a mean-squared error of 0.63.

```
# To construct the factor, we need the variance of treatment effects
# First get the individual-level variances
individual_var = np.mean(data.groupby(by = ['experiment_number',
                           'treatment']).agg(lambda x: np.var(x))['outcome'])
print("The variance for the errors is: " + str(round(individual_var, 4)))
The variance for the errors is: 399.0867

# Construct the variance for the treatment effect
treatment_var = individual_var/1000 + individual_var/1000

# Construct the factor
factor = 1 - (len(estimates) - 2) * (treatment_var)/(np.sum(estimates**2))
print("The adjustment factor is: " + str(round(factor, 4)))
The adjustment factor is: 0.731

# Evaluate the adjusted estimates
js_estimates = estimates * factor
print("The MSE for the James-Stein analysis is: " +
      str(round(np.mean((js_estimates - truth['true_effect'])**2), 4)))
The MSE for the James-Stein analysis is: 0.6254
```

3. **(15 points)** Using the split-sample empirical Bayesian estimator, adjust your estimates from Question 1. Report the parameters of the shrinkage model you are using. Then report the mean-squared error of your estimates, by comparing them to the true treatment effects.

   To implement the split-sample estimator, we split each experiment into two sub-experiments (with balanced treatment and control groups). Within each experiment, we estimate a direct treatment effect. Once we have processed all fifty experiments, we regress one set of treatment effects on the second to get a shrinkage model. The shrinkage model we estimate here has $\beta_0 = 0.19$ and $\beta_1 = 0.62$. We can then apply that to shrink our direct estimates, and get a mean-squared error of 0.61.

```
# Initialize an empty data frame for the split-sample treatment effects
split_sample = pd.DataFrame(np.nan, index= range(1,51),
                            columns=['treatment1', 'treatment2'])

# Set a random seed
np.random.seed(1)
```

```
# Iterate through each of the fifty experiments
for experiment_number in range(1, 51):

  # Identify the treated and control data for each experiment, and split into
  # two equally-sized splits of treated/control indices
  i = np.where((data['experiment_number'] == experiment_number) &
               (data['treatment'] == 1))[0]
  j = np.where((data['experiment_number'] == experiment_number) &
               (data['treatment'] == 0))[0]
  index1 = np.random.choice(i, round(len(i)/2), replace = False)
  index1 = np.append(index1,
                     np.random.choice(j, round(len(j)/2), replace = False))
  index2 = list(set(np.append(i, j)) - set(index1))

  # Run regressions for each of the two split samples, and save the parameter
  result = sm.ols(formula = 'outcome ~ treatment',
                  data = data.iloc[index1,]).fit()
  split_sample.loc[experiment_number,'treatment1'] = result.params[1]
  result = sm.ols(formula = 'outcome ~ treatment',
                  data = data.iloc[index2,]).fit()
  split_sample.loc[experiment_number,'treatment2'] = result.params[1]

# Run the regression of the second treatment effect on the first to get the
# appropriate shrinkage model
result = sm.ols(formula = 'treatment2 ~ treatment1', data = split_sample).fit()

# Report details on the model
print("The intercept for the split-sample shrinkage model is: " +
      str(round(result.params[0], 4)))
print("The coefficient for the split-sample shrinkage model is: " +
      str(round(result.params[1], 4)))
The intercept for the split-sample shrinkage model is: 0.1852
The coefficient for the split-sample shrinkage model is: 0.6211

# Using this model, adjust the original estimates; and evaluate the MSE
ss_estimates = result.params[0] + estimates * result.params[1]
print("The MSE for the split-sample analysis is: " +
      str(round(np.mean((ss_estimates - truth['true_effect'])**2), 4)))
The MSE for the split-sample analysis is: 0.6084
```

4. **(20 points)** Using LASSO, create new estimates of treatment effects for each experiment. In order to tune the parameter α, demonstrate two different cross-validation approaches, both using the mean-squared error in the predicted versus actual response values as the metric. First, set the optimal α within each experiment, defined as the alpha corresponding to the lowest mean-squared error for that experiment. Second, find the optimal α across all experiments, defined as the alpha corresponding to the lowest average mean-squared error across all experiments. For this second approach, report the optimal α you find. Under both approaches, you should use five cross-validation folds. Once you have generated your set of

estimates for each of the two tuning approaches, report the mean-squared error for that set of estimates, by comparing them to the true treatment effects. Thus, you should have two mean-squared errors for this question.

- You should search $\alpha \in [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]$. You also should not use any inbuilt functions to search for the optimal α, such as GridSearchCV (Python); but should code from scratch, as we did in class.
- As a hint, remember that you fit each experiment and compute each metric (e.g. mean-squared error between $\hat{y}$ and y) individually under both approaches. The difference between computing a per-experiment or pooled α only comes once you have all metrics. At that point, you can either find the minimum metric and corresponding α for each experiment; or you can find the average minimum metric and corresponding α across all experiments.
- Given the number of folds and the variation in inputs, you are strongly encouraged to write a modular function to keep your code efficient and readable.

To make this and the subsequent question easier, we first write a full cross-validation function. This function takes inputs on the number of folds, the model type (LASSO versus Ridge), the types of metrics (MSE in predicted y values, versus squared error in predicted coefficients), and whether the α values are computed in a pooled or per-experiment manner. That function is written below.

```
# Define the cross validation function, which takes in a set of inputs on
# the model type, the number of folds, the metric type, and whether we are
# computing the best overall or best-per-experiment alpha; and returns a list
# of alphas
def cross_validate(model_type = 'lasso', folds = 5,
                   metric_type = 'mse', pooled = False):

  # Define potential alphas
  alphas = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]
  results = np.empty((50, len(alphas)))

  # Iterate through each experiment, and iterate through each value of alpha
  for experiment_number in range(1, 51):

    # Get relevant data
    subset = data.iloc[np.where(data['experiment_number'] ==
                            experiment_number)[0],]

    # Split treatment and control data into three folds
    treatment_i = np.where(subset['treatment'] == 1)[0]
    control_i = np.where(subset['treatment'] == 0)[0]
    treatment_i = np.random.choice(treatment_i, len(treatment_i),
                               replace = False)
    control_i = np.random.choice(control_i, len(control_i), replace = False)
    treatment_i = np.array_split(treatment_i, folds)
    control_i = np.array_split(control_i, folds)
    # Save each fold as a set of indices
    indices = []
    for i in range(folds):
      indices.append(np.append(treatment_i[i], control_i[i]))

    # Iterate through each value of alpha
    j = 0
    for alpha in alphas:

      # Iterate through each of the folds
      metrics = []
      for i in range(folds):

        # Define relevant indices
        training_indices = np.concatenate(indices[:i] + indices[(i+1):])
        prediction_indices = indices[i]

        # Train model
        if model_type == 'lasso':
          model = Lasso(alpha = alpha, normalize = True, fit_intercept = True)
        else:
          model = Ridge(alpha = alpha, normalize = True, fit_intercept = True)
        model.fit(subset.iloc[training_indices,][['treatment']],
                  subset.iloc[training_indices,]['outcome'])

        # Compute metric: either the difference in y values, or the
        # difference between the shrunk and unshrunk coefficients
        if metric_type == 'mse':
          y = model.predict(subset.iloc[prediction_indices,][['treatment']])
          metric = np.mean((y - subset.iloc[prediction_indices,]['outcome'])**2)
        else:
          ols = sm.ols(formula = 'outcome ~ treatment',
                       data = subset.iloc[prediction_indices,]).fit()
          metric = ((model.coef_[0] - ols.params[1])**2)

        # Save metric value
        metrics.append(metric)

      # Once all folds have been computed for a given alpha, save the average
      # metric for that alpha
      results[experiment_number - 1,j] = np.mean(metrics)
      j = j + 1

  # Compute the best alpha by experiment or across all experiments
  if pooled:
    alpha = alphas[np.argmin(np.mean(results, axis = 0))]
    parameters = np.repeat(alpha, 50)
```

```
    print("The optimal pooled alpha is: " + str(alpha))
  else:
    parameters = np.array(alphas)[np.argmin(results, axis = 1)]

  # Return parameters
  return(parameters)
```

With this function, we can then get the optimal α values corresponding to each of the two approaches. We in turn use those cross-validated α values to generate LASSO estimates (using all the data in each experiment), and compute a mean-squared error versus the true treatment effects. The treatment effects that have per-experiment α values do poorly, generating an MSE of 0.96. The treatment effects that have a pooled α value (estimated at 0.01) do well, generating an MSE of 0.65. This divergence is discussed further in the final question.

```
# Estimate penalized regression for a given experiment, alpha, and model_type
def penalized_regression(experiment_number, alpha, model_type = 'lasso'):

  i = np.where(data['experiment_number'] == experiment_number)[0]

  if model_type == 'lasso':
    model = Lasso(alpha = alpha, normalize = True, fit_intercept = True)
  else:
    model = Ridge(alpha = alpha, normalize = True, fit_intercept = True)

  model.fit(data.iloc[i,][['treatment']], data.iloc[i,]['outcome'])

  return(model.coef_[0])

# Generate two sets of LASSO estimates
alphas = cross_validate(metric_type = 'mse', pooled = False)

lasso_estimates1 = np.array([penalized_regression(number, alphas[number - 1])
                             for number in range(1, 51)])
print("The MSE for the per-experiment LASSO variant is: " +
      str(round(np.mean((lasso_estimates1 - truth['true_effect'])**2), 4)))
The MSE for the per-experiment LASSO variant is: 0.9558

alphas = cross_validate(metric_type = 'mse', pooled = True)
The optimal pooled alpha is: 0.01

lasso_estimates2 = np.array([penalized_regression(number, alphas[number - 1])
                             for number in range(1, 51)])
print("The MSE for the pooled LASSO variant is: " +
      str(round(np.mean((lasso_estimates2 - truth['true_effect'])**2), 4)))
The MSE for the pooled LASSO variant is: 0.65
```

5. **(10 points)** Using Ridge Regression, create new estimates of treatment effects for each experiment, utilizing a third cross-validation approach. Continue to use five cross-validation

folds, and continue to find the optimal α across all experiments (rather than within each experiment) as in the second approach previously. However, now use the mean-squared difference between the estimated and OLS coefficient as the metric. Report the optimal α you find. Moreover, report the mean-squared error of your estimates, by comparing them to the true treatment effects.

Using the cross-validation function written in the previous question, we can get the optimal Ridge regression α, pooled across all experiments; and in turn use those to generate treatment effects. The optimal pooled $\alpha = 0.5$, and this generates estimates have an MSE of 0.58.

```
# Generate Ridge estimates
alphas = cross_validate(model_type = 'ridge', metric_type = 'beta',
                        pooled = True)
The optimal pooled alpha is: 0.5

ridge_estimates = np.array([penalized_regression(number, alphas[number - 1],
                            model_type = 'ridge') for number in range(1, 51)])
print("The MSE for the Ridge Regression is: " +
      str(round(np.mean((ridge_estimates - truth['true_effect'])**2), 4)))
The MSE for the Ridge Regression is: 0.5823
```

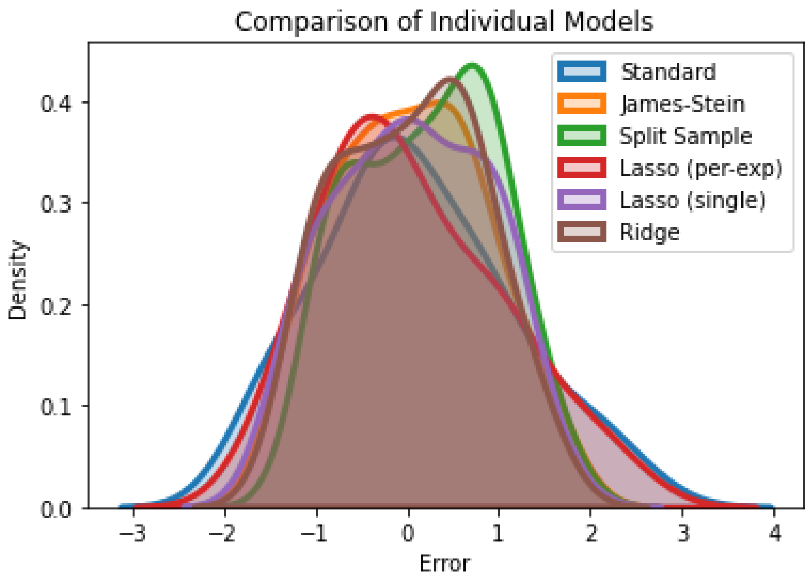
6. **(10 points)** You now have six sets of estimates from six distinct approaches: the simple frequentist estimates, the James-Stein estimates, the split-sample estimates, the LASSO estimates with the per-experiment α, the LASSO experiments with the single α, and the Ridge Regression estimates. On a single histogram or density plot, graph the the differences between the estimated treatment effect and the true treatment effect, across all fifty experiments and using different colors for each approach. As always, points will be awarded on the basis of the ease of understanding your visualization.

The first plot is below, with the six histograms. This plot is sufficient for full credit, although I personally find it a bit cluttered. As such, there is a second plot below that plots the simple versus average of all shrinkage approaches, to drive home the difference in average error between the two classes of approaches.

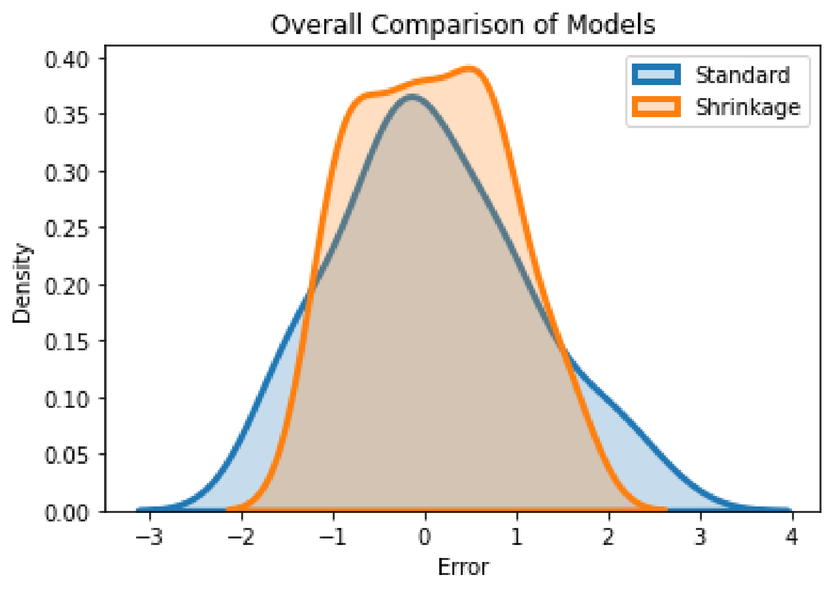
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(estimates - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Standard')
sns.distplot(js_estimates - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'James-Stein')
sns.distplot(ss_estimates - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Split Sample')
sns.distplot(lasso_estimates1 - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Lasso (per-exp)')
```

```
sns.distplot(lasso_estimates2 - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Lasso (single)')
sns.distplot(ridge_estimates - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Ridge')
plt.title('Comparison of Individual Models')
plt.xlabel('Error')
plt.ylabel('Density')
plt.legend()
```



```
sns.distplot(estimates - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Standard')
shrinkage_estimates = (js_estimates + ss_estimates + lasso_estimates1 +
                       lasso_estimates2 + ridge_estimates)/5
sns.distplot(shrinkage_estimates - truth['true_effect'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Shrinkage')
plt.title('Overall Comparison of Models')
plt.xlabel('Error')
plt.ylabel('Density')
plt.legend()
```



7. **(10 points)** Discuss the advantages and disadvantages of each of the six approaches. Which would you recommend the Youtube data scientists use going forward? Limit your entire answer to approximately three hundred words.

   In general, all the shrinkage approaches do comparably well, and far better than the standard frequentist approaches, with one exception: the LASSO regression with per-experiment α values. This approach does poorly, and that is likely because of the tremendous noise in trying to fit so many parameters with so little data per parameter. By contrast, penalized regression approaches that estimate a pooled α benefit from more stable estimation procedures; and thus they lead to estimates with low MSE.

   As such, we should consider other dimensions when comparing the shrinkage approaches. Each approach has its advantages and disadvantages. The James-Stein estimator has the advantage of having low data requirements (the means, rather than the underlying observations) and being computationally quick. The split-sample estimator is most intuitive, and works best if assumptions like normality of the data are suspect. The penalized regression estimators are very computationally intensive, but they are potentially very promising. In this example, they are performing best; and that is with only some rough and unrefined tuning of the α parameter!

   Any answer that discusses these components and argues in favor of one approach will receive full credit. Personally, I believe the split-sample estimator strikes the best balance between being intuitive and having moderate computational burdens.