LIGN 167: Problem Set 1

October 5, 2018

Collaboration policy: You may collaborate with up to two other students on this problem set. You must submit your work individually. When you submit your work, you must indicate who you worked with, and what each of your individual contributions were.

Throughout this problem set you should use the lign167 environment that you created in Anaconda. To use this environment, open up your Terminal (Mac/Linux) or Anaconda Prompt (Windows). Then type:

```
source activate lign167
```

This will activate your lign167 environment, allowing you to access all of the dependencies that you downloaded for the course.

After activating this environment, you should also download an additional dependency:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

This will install Matplotlib, a library for plotting that we will be using.

This problem set has two purposes: get you familiar with Python and NumPy, and reveal some important properties of statistical estimation and prediction.

In class we talked about *least squares regression*. In least-squares regression, we have a dataset that consists of two variables, $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$. We are trying to predict the values in Y from the values in X using the linear equation $y = a \cdot x + b$. More precisely, for each x_i, we use this equation to compute a predicted value:

$$\hat{y_i} = a \cdot x_i + b \tag{1}$$

Our goal is to minimize the total error of our predictions on the dataset. We want to find the values of a and b that minimize the quantity:

$$L = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 \tag{2}$$

$$= \sum_{i=1}^{n} (y_i - a \cdot x_i - b)^2 \tag{3}$$

In class, we found equations for the optimal values of the slope a and intercept b. These equations define *estimators* of the slope and intercept. The equations used the definition of the mean of a variable: $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$. The equations for the estimators of a and b that we

found were:

$$a(x,y) = \frac{\sum_i x_i \cdot y_i - n\bar{x}\bar{y}}{\sum_i x_i^2 - n\bar{x}^2} \tag{4}$$

$$b(x,y) = \bar{y} - a(x,y) \cdot \bar{x} \tag{5}$$

Note that we are treating these estimators as functions: $a(x,y)$ takes in the values of x and y, and returns an estimated value of the slope, and similarly for $b(x,y)$.

Problem 1. Write a Python function *compute_slope_estimator*, which takes in two input variables, x and y. The variables x and y should be 1-dimensional NumPy arrays that have the same length n. The function should return the optimal value of the slope from Equation 4.

(If you are new to Python or Numpy, see this great tutorial: http://cs231n.github.io/python-numpy-tutorial. Also, please don't hesitate to come to office hours.)

Problem 2. Write a Python function *compute_intercept_estimator*, which takes in two input variables, x and y. The variables x and y should be 1-dimensional NumPy arrays that have the same length n. The function should return the optimal value of the intercept from Equation 5.

Problem 3. Write a function *train_model*, which takes in two 1-dimensional NumPy arrays of the same length, x and y. It should use *compute_slope_estimator* and *compute_intercept_estimator*, and return a tuple of values: the optimal value of the slope and the optimal value of the intercept.

The elements in the array y can be considered our training set: we use them to estimate the optimal values of the slope and intercept.

Problem 4. Write a function *sample_linear_model* which takes four arguments: *x_vals*, a, b, and *sd*. The variable *x_vals* is a 1-dimensional NumPy array of length n. The function should return a NumPy array y of length n, where each element of y_i has been sampled from: $y_i = a \cdot x_i + b + \epsilon_i$. Here ϵ_i should follow a normal distribution with mean equal to 0 and standard deviation equal to *sd*.

This function describes the *generative model* that we believe our dataset was sampled from.

(You should use NumPy's built in functions for sampling from a normal distribution.)

Problem 5. Write a function *sample_datasets* which takes five arguments (the first four the same as in the previous problem): *x_vals*, a, b, *sd*, and n. It should return a list of n sampled datasets, where each dataset is constructed using the function *sample_linear_model* from the previous problem.

Problem 6. The true value of the slope and intercept are hidden from us; we use Equations 4 and 5 to define estimators of these quantities, i.e. to infer the best values of these quantities, given the information that we have. In the next problems we are going to examine the properties of these estimators.

There is a common technique for examining the properties of an estimator. First, make a hypothesis/guess about the true model parameters. Using these guessed model parameters, sample many (e.g. 1000) possible training sets. On each of these training sets, compute the

estimator for the model parameters. Finally, evaluate the distribution of the estimators: how far are they, on average, from the true value of the model parameters.

As a first step towards this, we are going to compute the average value of the estimated slope, for a given set of hypotheses about the true model parameters. Assume that $a = 1$, $b = 1$, and $sd = 1$. **These assumed parameter values will be used throughout the rest of the problem set.**

Using these parameter values and the function *sample_datasets* from Problem 5, sample 1000 training sets. (The function *sample_datasets* also requires a value of *x_vals* to be input; this will be defined below.) Denoting the i'th training set by $\mathbf{y_i}$, compute the average value of the estimated slope on these 1000 training sets:

$$\frac{1}{1000}\sum_{i=1}^{1000} a(x, \mathbf{y_i}) \tag{6}$$

Write a function *compute_average_estimated_slope* which takes a NumPy array *x_vals* as input, and returns the average estimated slope.

Problem 7. Let us now compute the average estimated slope for different values of *x_vals*. Using NumPy, you can easily create arrays containing n evenly spaced points between 0 and 1.

```
x_vals = np.linspace(0,1,num=n)
```

Using this function to create *x_vals*, call *compute_average_estimated_slope* with n equal to 5, 100, and 1000. What do you notice about average estimated slope that is returned?

Problem 8. We're going to next examine a different property of the estimated slope: its average error. Using the same procedure (and same model parameters) as in Problem 6, sample 1000 training sets. Denoting the i'th training set by $\mathbf{y_i}$, compute the average squared error of the estimated slope:

$$\frac{1}{1000}\sum_{i=1}^{1000}(1 - a(x, \mathbf{y_i}))^{\mathbf{2}} \tag{7}$$

(Note that in the above formula, we are taking the difference between $a(x, \mathbf{y_i})$ and 1 because the assumed true value of the slope is 1.)

Write a function *compute_estimated_slope_error*, which takes *x_vals* as input, and returns the average squared error of the estimated slope.

As in Problem 7, use *np.linspace* to try values of *x_vals* with n equal to 5, 100, and 1000. What do you notice about the average squared error as n increases?

Problem 9. Sample 1000 training sets as in the previous problems, and calculate the estimated value of the slope on each of the 1000 training sets. Collect these 1000 samples together in a NumPy array. Using Matplotlib, create a histogram of these samples.

Try this for different values of *x_vals*, as in Problems 7 and 8. What do you notice about these distributions?

Problem 10. Write a function *calculate_prediction_error*, which takes as input two NumPy arrays of the same size, *y* and *y_hat*. Using y_i to denote the i'th element of

the array y (and similarly for y_hat), and assuming that the length of the arrays is n, return the average square difference between the arrays:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{8}$$

Problem 11. In Equation 2, we define the error of a given slope and intercept on the training set. In the current question, we will be investigating the average magnitude of these errors, that is: after we fit a slope and intercept to a training set, how well, on average, will we be able to predict the values that occur in the training set?

Write a function *average_training_set_error*, which takes *x_vals* as input. It should sample 1000 training sets, as in previous problems. For each training set, it should calculate the estimator for the slope and intercept. It should then use the estimated slope and intercept to compute the predicted value $\hat{y}$, using Equation 1. Finally, it should use the function *calculate_prediction_error* from Problem 10 to compute the prediction error between $\hat{y}$ and the observed values in the training set.

The function will therefore be calculating 1000 prediction errors. Let p_i be the i'th prediction error. The function should return the average value of the errors:

$$\frac{1}{1000}\sum_{i=1}^{1000} p_i \tag{9}$$

Try this for different values of *x_vals*, as in Problems 7, 8, and 9. As the number of elements in *x_vals* increases, what happens to the average prediction error?

Problem 12. In the previous problem, you were asked to fit parameters to a training set, and evaluate the predictions of the resulting model *on that same training set*. This can give a biased (artificially low) estimate of the model's prediction error. The model may achieve a very low prediction error on a training set by *overfitting* that training set.

In order to better evaluate whether our model has learned a real pattern in the data, we can test its ability to generalize, i.e. predict data points that it has never observed before. More precisely, we will evaluate the model's prediction error on a test set which is sampled independently of the training set.

Write a function *average_test_set_error*, which takes *x_vals* as input. It should sample 1000 training sets, as in previous problems. For each training set, it should calculate the estimator for the slope and intercept. It should then use the estimated slope and intercept to compute the predicted value $\hat{y}$, using Equation 1.

In contrast to the previous problem, it should now sample a test set, by calling *sample_linear_model*. This will give us one test set for every training set. Finally, it should use the function *calculate_prediction_error* from Problem 10 to compute the prediction error between $\hat{y}$ and the observed values in the test set.

The function will therefore be calculating 1000 prediction errors (one for each of the 1000 test sets). Let p_i be the prediction error on the i'th test set. The function should return the average value of the errors:

$$\frac{1}{1000}\sum_{i=1}^{1000} p_i \tag{10}$$

Try this for different values of *x_vals*, as in previous problems. What do you notice when you compare the average value of the test set prediction error, to the average value of

the training set prediction error, as computed in Problem 11? What happens as the number of elements in x_vals increases?