CS181 Problem Set 1

Lev Kruglyak

Due: February 10, 2024

Collaborators: *AJ LaMotta*

Problem 1. (Setting up the Regression) Your goal is to predict temperature variation given the year. Before we start deriving and coding up our regressions, we will interrogate the set-up of our problem.

a. These data were derived from ice core samples in Antarctica. Take a brief look at the original data file. Briefly discuss how the data were processed: what kinds of decisions and corrections had to be made? We know that different places on earth have different temperatures: what does the temperature in the temperature column correspond to?

Reading the source, we can see that several corrections were made. For example, temperatures were corrected for sea-water isotropic composition and for ice sheet elevation. Also, temperatures are not recorded on an absolute scale but as a change in temperature from a rolling average of previous changes.

b. Even before doing any formal regressions, we see that there is some periodicity in the data: there are years that are warmer, and years that are cooler. Suppose you are a government official advising on how much to worry about climate change. Would it be reasonable to use this pattern as evidence that the earth will cool down again? Why or why not, or to what extent?

While there does appear to be some peridocity, using the dataset to predict anything too far into the future would be tricky, since temperature variation is an incredibly complex system with lots of factors. For example, volcanic erruptions or human CO2 emissions could alter the temperature on small timeframes, breaking any pattern deduced by the regression.

c. In the problems below, we will focus on interpolating temperatures for years not provided in the training set. What kind of application would such a regression be useful for?

The main application would be predicting future temperature variations, or making estimates of temperature variations so long ago that measuring directly would be infeasible.

Problem 2. (Optimizing a Kernel) Kernel-based regression techniques are similar to nearest-neighbor regressors: rather than fit a parametric model, they predict values for new data points by interpolating values from existing points in the training set. In this problem, we will consider a kernel-based regressor of the form:

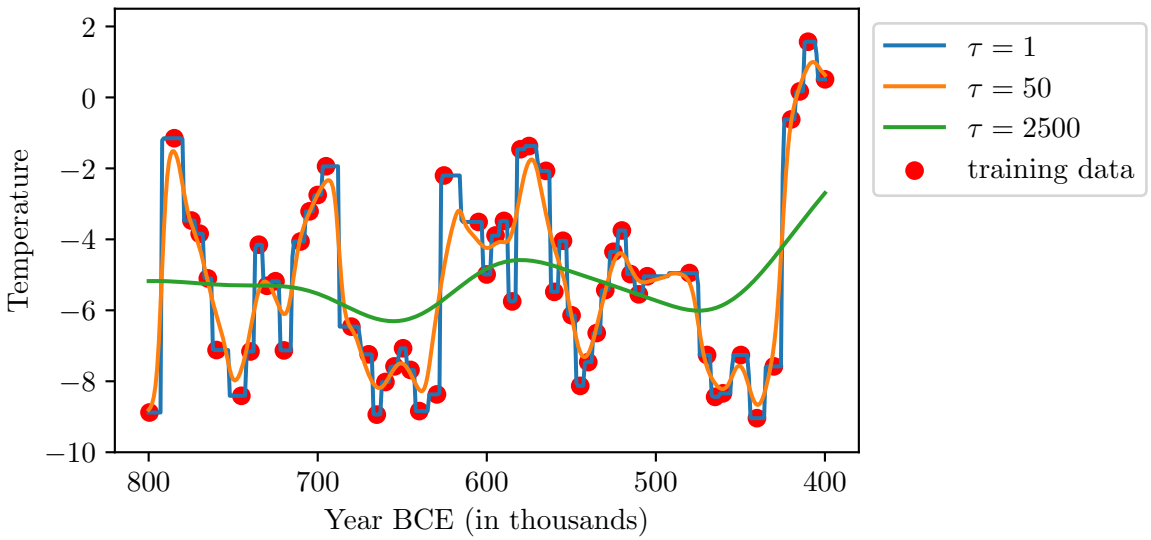
$$f_\tau(x^*) = \frac{\sum_n K_\tau(x_n, x^*) y_n}{\sum_n K_\tau(x_n, x^*)}$$

where $\mathcal{D}_{\texttt{train}} = \{(x_n, y_n)\}_{n=1}^N$ are the training data points, and x^* is the point for which you want to make the prediction. The kernel $K_\tau(x, x')$ is a function that defines the similarity between two inputs x and x'. A popular choice of kernel is a function that decays as the distance between the two points increases, such as

$$K_\tau(x, x') = \exp\left\{-\frac{(x - x')^2}{\tau}\right\}$$

where τ represents the square of the lengthscale (a scalar value that dictates how quickly the kernel decays). In this problem, we will consider optimizing what that (squared) lengthscale should be.

a. Let's first take a look at the behavior of the fitted model for different values of τ. Implement the `kernel_regressor` function in the notebook, and plot your model for years in the range 800,000 BC to 400,000 BC at 1000 year intervals for the following three values of τ: $1, 50, 2500$. Since we're working in terms of thousands of years, this means you should plot $(x, f_\tau(x))$ for $x = 400, 401, \ldots, 800$. Describe how the fits change with τ.



As we can see, as τ decreases, the curve fits the training data points closer and closer. As τ increases, the curve generalizes more, is smoother, and shows the general trends of the data.

b. Now, we will evaluate the quality of each model *quantitatively* by computing the error on some test set $\mathcal{D}_{\texttt{test}} = \{(x'_m, y'_m)\}_{m=1}^M$. Write down the expression for MSE of f_τ over the test set as a function of the training set and test set. Your answer may include $\{(x'_m, y'_m)\}_{m=1}^M$, $\{(x_n, y_n)\}_{n=1}^N$, and K_τ, but not f_τ.

Using the mean squared error formula and the definition for our estimator, we can write

$$\text{MSE} = \frac{1}{M} \sum_{1 \le m \le M} \left(y'_m - f_\tau(x'_m)\right)^2 = \frac{1}{M} \sum_{1 \le m \le M} \left(y'_m - \frac{\sum_{n=1}^N K_\tau(x_n, x'_m) y_n}{\sum_{n=1}^N K_\tau(x_n, x'_m)}\right)^2.$$

c. Suppose we used the training set as our test set, that is, we evaluated the expression above with $\mathcal{D}_{\texttt{test}} = \mathcal{D}_{\texttt{train}}$, and chose the value of τ which gave the smallest loss. What value of τ would be picked? Why is setting $\mathcal{D}_{\texttt{test}} = \mathcal{D}_{\texttt{train}}$ a bad idea?

Suppose we set $\mathcal{D}_{\texttt{test}} = \mathcal{D}_{\texttt{train}}$, then the expression for MSE is

$$\text{MSE} = \frac{1}{N} \sum_{1 \le m \le N} \left(y_m - \frac{\sum_{n=1}^N K_\tau(x_n, x'_m) y_n}{\sum_{n=1}^N K_\tau(x_n, x'_m)}\right).$$

Notice that as $\tau \to 0$, we have the convergence $K_\tau(x, x^*) \to \mathbb{I}_{\{x\}}(x^*)$ of the kernel function to an indicator variable. This means that $f_\tau(x^*)$ converges to the step function

$$\widetilde{f}_\tau(x^*) = y_{i(x^*)} \quad \text{where} \quad i(x^*) = \text{argmin}_{1 \le n \le N} |x^* - x_n|.$$

In particular, for any $(x, y) \in \mathcal{D}_{\texttt{test}}$ we have $\widetilde{f}_\tau(x) = y$. In other words, the estimator is perfect on the "test" dataset in the limit as $\tau \to 0$. The optimal τ then must be extremely small. (A value of $\tau = 0$ is impossible, unless the expressions in the definition of the estimator are replaced with their limit as $\tau = 0$.)

This is clearly not very useful – our model has completely overfit the data and became a k-means clustering estimator with $k = 1$, losing the beneficial features of the kernel estimator.

d. Let us compute the MSE on the provided test set (that is, not the training set). Write Python code to compute the MSE with respect to the same lengthscales as in (a). Which model yields the lowest test set MSE?

Calculating the MSE for the data we plotted in (a), we obtain the following data:

τ	MSE
1	1.947
50	1.858
2500	8.334

We can see that the model with $\tau = 50$ yields the lowest error. This makes sense if we look at the graph in (a) – when τ is very low, the model overfits to the points in the training data set and might deviate too much from the points in the test data set.

e. Describe the time and space complexity of kernelized regression with respect to the size of the training set N. How, if at all, does the size of the model—everything that needs to be stored to make predictions—change with the size of the training set N? How, if at all, do the number of computations required to make a prediction for some input x^* change with the size of the training set N?

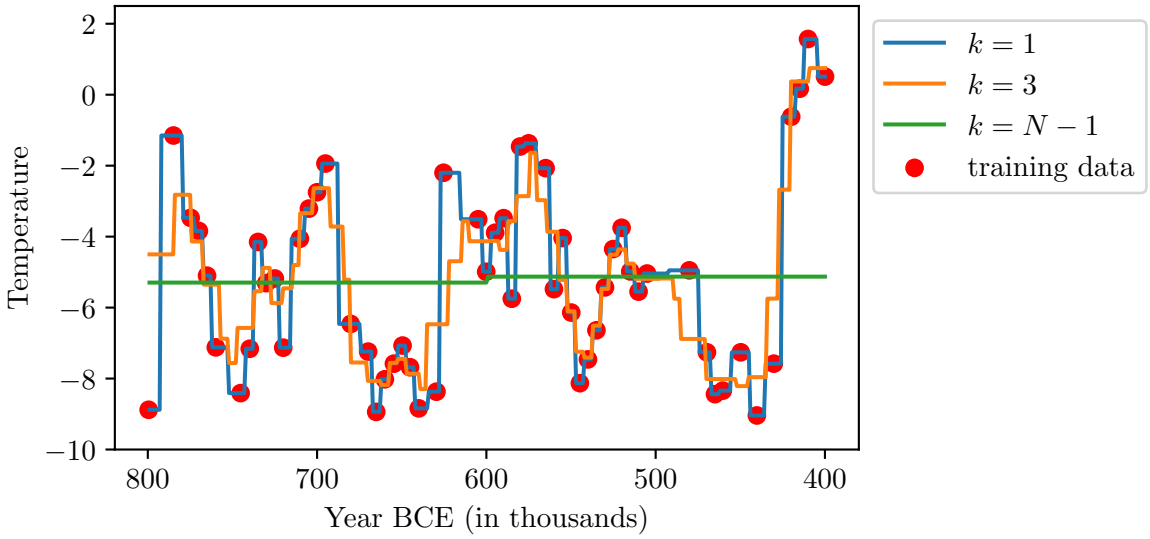
There are a variety of optimizations we could make to improve space and time complexity, however for the purposes of this problem let's assume no optimizations were made. For storage, we need to store the entire training dataset, or $2N$ numbers. This means storage complexity is $O(N)$. Similarly, assuming the kernel is constant time to evaluate, the time complexity to perform an estimation is $O(N)$. Thus, both factors scale linearly with the size of the training set.

Problem 3. (Kernels and kNN) Now, let us compare the kernel-based approach to an approach based on nearest-neighbors. Recall that kNN uses a predictor of the form

$$f(x^*) = \frac{1}{k}\sum_n y_n \mathbb{I}(x_n \texttt{ is one of k-closest to } x^*)$$

where $\mathbb{I}$ is an indicator variable. For this problem, you will use the **same dataset as in Problem 1**. Write your own tests!

a. The kNN implementation **has been provided for you** in the notebook. Run the cells to plot the results for $k = \{1, 3, N-1\}$, where N is the size of the dataset.
Describe what you see: what is the behavior of the functions in these three plots? How does it compare to the behavior of the functions in the three plots from Problem 1? In particular, are there situations in which kNN and kernel-based regression interpolate similarly?



The behavior of the k-means clustering estimator is quite similar to the behavior of the kernel estimator, however the kernel estimator is smoother. The observations about increasing/decreasing τ in the kernel estimator correspond to increasing/decreasing k.

b. Compute the MSE on the test set for each value of k. Which solution has the lowest MSE?

Doing the same calculations as before, we see that the best model has $k = 1$.

k	MSE
1	1.741
3	3.891
56	9.529

c. As in Problem 1, describe the space and time complexity of kNN. How does what is stored to compute predictions change with the size of the training set N? How does the computation needed to compute the prediction for a new input depend on the size of the training set N? (For the latter, justify based off of your implementation.)

As before, the space complexity is $O(N)$ since we need to store all of the training data. In the given implementation, the time complexity is $O(N \log N)$ since we have to sort the entire array of distances, however this is far from optimal.

In fact, the time complexity could also be made to be $O(N)$ since finding the k-nearest neighbors requires searching all points in the dataset and finding the k smallest elements. We could also account for the space and time complexity of storing and sorting the k smallest elements, however this is independent of N assuming k is. In particular, we don't have to sort the entire array of N data points to get the k smallest elements – rather we can keep some data structure of size complexity $O(k)$ and insert values into it with time complexity $O(k)$ for each iteration through the training data set.

Problem 4. (Modeling Climate Change 800,000 Years Ago) Our last regression will be linear regression. We currently only have a one dimensional input, the year. To create a more expressive linear model, we will introduce basis functions.

a. We will first implement the four basis regressions below. Note that we introduce an addition transform f to address concerns about numerical instabilities.

(a) $\phi_j(x) = f(x)^j$ for $j = 1, \ldots, 9$. $f(x) = \frac{x}{1.81 \cdot 10^2}$.

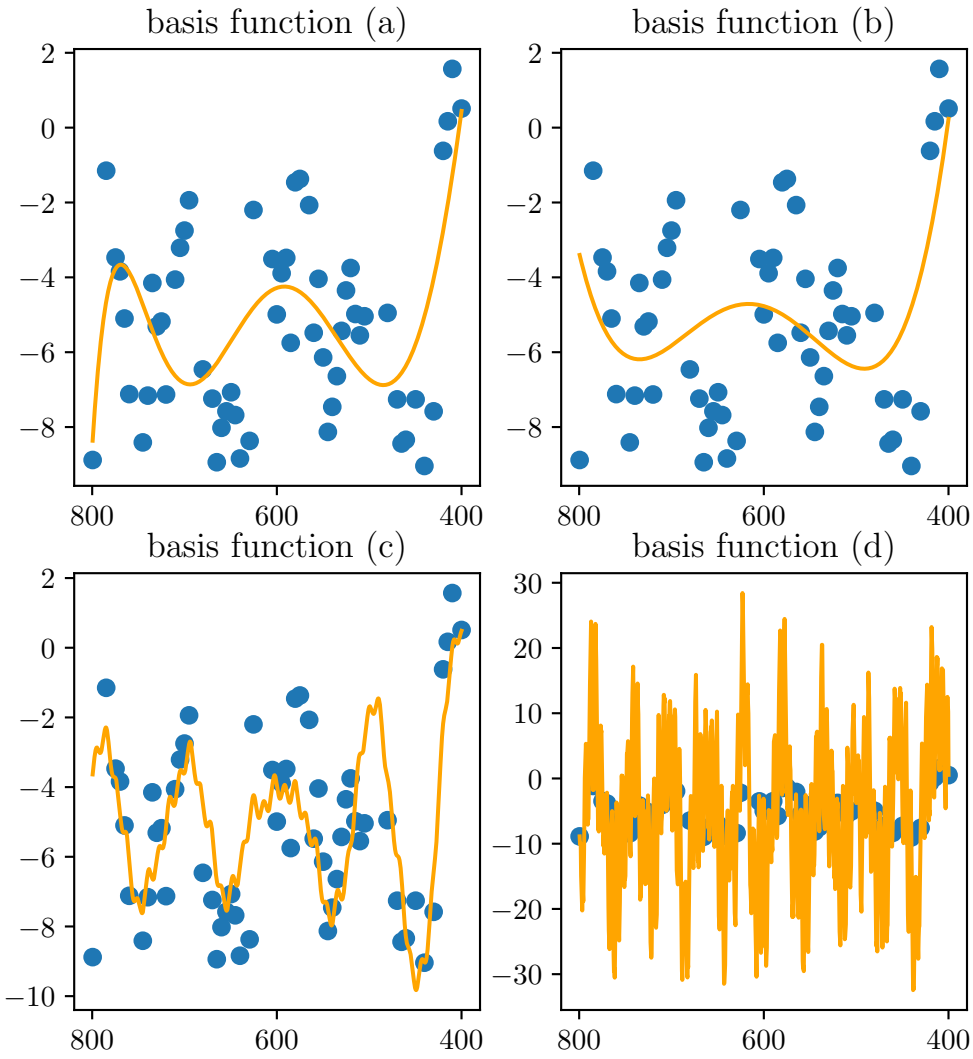
(b) $\phi_j(x) = \exp\left\{-(f(x) - \mu_j)^2/5\right\}$ for $\mu_j = \frac{j+7}{8}$ with $j = 1, \ldots, 9$. $f(x) = \frac{x}{4.00 \cdot 10^2}$.

(c) $\phi_j(x) = \cos(f(x)/j)$ for $j = 1, \ldots, 9$. $f(x) = \frac{x}{1.81}$.

(d) $\phi_j(x) = \cos(f(x)/j)$ for $j = 1, \ldots, 49$. $f(x) = \frac{x}{1.81 \cdot 10^{-1}}$.

For each basis create a plot of your code graphing the least squares regression line trained on your training data against a scatter plot of the training data. Boilerplate plotting code is provided in the notebook.

Using the standard equations for linear regression with basis functions, we get the following graphs:



b. Now we have trained each of our basis regressions. For each basis regression, compute the MSE on the test set. Discuss: do any of the bases seem to overfit? Underfit? Why?

Computing the MSE (for both training and testing data) gives us the following table:

basis function	MSE_{test}	$\text{MSE}_{\text{train}}$
(a)	7.956	4.825
(b)	8.708	5.530
(c)	5.967	2.879
(d)	58.912	0.643

As we can confirm from the plots in the previous part, the basis function (c) has the lowest error on the testing data. Looking at the training MSE as well, we see that (d) overfits the most by far, whereas (a) and (b) underfit. (c) lies at a perfect balance between both metrics.

c. Briefly describe what purpose the transforms f serve: why are they helpful?

The transforms help fight numeric instability present due to the limitations of floating point numbers.

d. As in Problems 1 and 2, describe the space and time complexity of linear regression. How does what is stored to compute predictions change with the size of the training set N? How does the computation needed to compute the prediction for a new input depend on the size of the training set N? How do these complexities compare to those of the kNN and kernelized regressor?

For a linear regression with basis functions, both the storage and time complexity are $O(1)$, since we only need to store the constant size weight matrix, unlike the previous two estimators which required us to store the whole dataset. This makes linear regression much more efficient, however it might not be as accurate for dataset with exceptionally complex features.

e. Briefly compare and contrast the different regressors: kNN, kernelized regression, and linear regression (with bases). Are some regressions clearly worse than others? Is there one best regression? How would you use the fact that you have these multiple regression functions?

I think as with many such problems, it depends on the dataset. If the dataset follows a simple linear or polynomial, or even periodic distribution, linear regression will be able to very efficiently extrapolate a pattern, and generalize it quite well. This would also take up much less space that the other two regressors, and be more efficient in calculation. On the other hand, if we have some very unpredictable dataset such as a stock valuation history, a kNN or kernelized regression might perform better at capturing the fine details. Between those two, if the input points are not evenly distributed, then kernelized regression might perform worse due to the kernel not picking up nearby points, whereas kNN would simply select the k nearest points. In most scenarios, it's good to try out different regressors and compare them using MSE or other metrics to select the best one.

Problem 5. (Deriving Linear Regression) In the previous problems, you focused on implementing regressions and exploring their fits on data. Now we will turn to some more analytic work. Specifically, the solution for the least squares linear regressions "looks" kind of like a ratio of covariance and variance terms. In this problem, we will make that connection more explicit.

Let us assume that our data are tuples of scalars (x, y) that are described by some joint distribution $p(x, y)$. We will consider the process of fitting these data from this distribution with the best linear model possible, that is a linear model of the form $\hat{y} = wx$ that minimizes the expected squared loss $\mathbb{E}_{x,y}[(y - \widehat{y})^2]$.

a. Derive an expression for the optimal w, that is, the w that minimizes the expected squared loss above. You should leave your answer in terms of moments of the distribution, e.g. terms like $\mathbb{E}_x[x]$, $\mathbb{E}_x[x^2]$, $\mathbb{E}_y[y]$, $\mathbb{E}_y[y^2]$, $\mathbb{E}_{x,y}[xy]$ etc.

Let's first expand the loss function:

$$\begin{aligned}\mathbb{E}(\mathcal{L}) = \mathbb{E}_{x,y}[(y - \widehat{y})^2] = \mathbb{E}_{x,y}[(y - wx)^2] &= \mathbb{E}_{x,y}[y^2 - 2wxy + 2w^2x^2] \\ &= \mathbb{E}_y[y^2] - 2w\mathbb{E}_{x,y}[xy] + 2w^2\mathbb{E}_x[x^2]\end{aligned}$$

Next, to minimise this loss we can take the derivative with respect to the weights and set it to zero:

$$\partial_w \mathbb{E}(\mathcal{L}) = -2\mathbb{E}_{x,y}[xy] + 2w\mathbb{E}_x[x^2] = 0 \quad \Longrightarrow \quad w^* = \frac{\mathbb{E}_{x,y}[xy]}{\mathbb{E}_x[x^2]}.$$

This an the expression for the optimal weights.

b. Provide unbiased and consistent formulas to estimate $\mathbb{E}_{x,y}[xy]$ and $\mathbb{E}_x[x^2]$ given observed data $\{(x_n, y_n)\}_{n=1}^N$.

These formulas can be given by the standard unbiased/consistent estimators for moments:

$$\mathbb{E}_{x,y}[xy] = \frac{1}{N}\sum_{i=1}^{N} x_i y_i, \quad \text{and} \quad \mathbb{E}_x[x^2] = \frac{1}{N}\sum_{i=1}^{N} x_i^2.$$

c. In general, moment terms like $\mathbb{E}_{x,y}[yx]$, $\mathbb{E}_{x,y}[x^2]$, $\mathbb{E}_{x,y}[yx^3]$, $\mathbb{E}_{x,y}[\frac{x}{y}]$, etc. can easily be estimated from the data (like you did above). If you substitute in these empirical moments, how does your expression for the optimal w^* in this problem compare with the optimal w^* that we see in Section 2.6 of the cs181-textbook?

Using the formulas for the empirical moments, we see that

$$w^* = \frac{\mathbb{E}_{x,y}[xy]}{\mathbb{E}_x[x^2]} = \frac{(1/N)\cdot\sum_{1\le i\le N} x_i y_i}{(1/N)\cdot\sum_{1\le i\le N} x_i^2} = \frac{\sum_{1\le i\le N} x_i y_i}{\sum_{1\le i\le N} x_i^2}.$$

Recall from the textbook that the optimal weights had the expression

$$w^* = (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbf{y}.$$

However, note that $\mathbf{X}^\intercal\mathbf{y} = \sum_{1\le i\le N} x_i y_i$ and $\mathbf{X}^\intercal\cdot\mathbf{X} = \sum_{1\le i\le N} x_i^2$, so the two expressions for w^* are equivalent.

d. Many common probabilistic linear regression models assume that variables x and y are jointly Gaussian. Did any of your above derivations rely on the assumption that x and y are jointly Gaussian? Why or why not?

We made no such assumptions in the problem, all we assumed was that x and y are random variables which have second moments.