Assignment #3
Due: 11:59pm March 6, 2020

Homework 3: Bayesian Methods, Neural Networks, and Practical Supervised Learning

Introduction

This homework is about Bayesian methods, Neural Networks, and practical supervised learning. Section 2.9 in the textbook as well as reviewing MLE and MAP will be useful for Q1. Chapter 4 in the textbook will be useful for Q2.

Please type your solutions after the corresponding problems using this LaTeX template, and start each problem on a new page.

Please submit the **writeup PDF to the Gradescope assignment 'HW3'**. Remember to assign pages for each question.

Please submit your **LaTeXfile and code files to the Gradescope assignment 'HW3 - Supplemental'**.

You can use a **maximum of 2 late days** on this assignment. Late days will be counted based on the latest of the two submissions.

Problem 1 (Bayesian Methods)

This question helps to build your understanding of making predictions with a maximum-likelihood estimation (MLE), a maximum a posterior estimator (MAP), and a full posterior predictive.

Consider a one-dimensional random variable $x \sim \mu + \epsilon$, where it is known that $\epsilon \sim N(0, \sigma^2)$. Suppose we have a prior $\mu \sim N(0, \tau^2)$ on the mean. Use $\sigma^2 = 1$ and $\tau^2 = 5$.

In this problem, we see 15 independent samples of x to yield data

$$D = 3.3, 3.5, 3.1, 1.8, 3.0, 0.74, 2.5, 2.4, 1.6, 2.1, 2.4, 1.3, 1.7, 0.19$$

Make sure to include all required plots in your PDF.

1. We are interested in the predictive distribution of a new datapoint x^* given our observed data D, $p(x^*|D)$. Write down the expression for the full posterior predictive distribution:

$$p(x|D) = \int p(x|\mu)p(\mu|D)d\mu$$

   There are two ways to do this problem. One involves actually doing out this integral, which has been well studied and there are resources that can guide you through it (http://www.tina-vision.net/docs/memos/2003-003.pdf). The second involves using the representation of x and calculating the posterior of μ.

2. The full posterior predictive distribution is often difficult to calculate because we need to marginalize out the parameters (here, the parameter is μ). We can mitigate this problem by plugging in a point estimate of μ. Find the estimates of $p(x|D) \approx p(x|\mu^*)$ for $\mu^* = \mu_{MLE}$ and $\mu^* = \mu_{MAP}$.

3. Plot how the above 3 distributions change after each data point is gathered. You will have a total of 15 plots (they can be small, e.g. in a 3x5 grid). The x-axis of each plot will be the x value and the y-axis the density. You can make one plot for each estimator, or combine all three estimators onto one plot with a different colored line for each estimator.

4. How do the means of the predictive distributions vary with more data? How do the variances vary? Interpret the differences you see between the three different estimators.

5. Does the ordering of the data matter for the final predictive distributions?

6. Compute the marginal likelihood of the training data $p(D)$. Hint: Note that the required integral looks like an unnormalized Gaussian distribution, and take advantage of the fact that integrating over a normalized Gaussian distribution is equal to 1.

7. Now consider an alternate model in which we were much more sure about the mean: $\mu \sim N(0, \tau^2)$, where $\tau^2 = 0.1$. Compute the marginal likelihood $p(D)$ for this model. Which of the two models has a higher marginal likelihood? Interpret this result.

Solution

Problem 2 (Backprop)

In this problem, we will implement backprop for a simple MLP. The MLP will consist of a first fully connected layer with a sigmoid activation, followed by a one-dimensional, second fully connected layer with a sigmoid activation to get a prediction for a binary classification problem. Assume bias has not been merged. Let:

- $\mathbf{W}_1$ be the weights of the first layer, $\mathbf{b}_1$ be the bias of the first layer.
- $\mathbf{W}_2$ be the weights of the second layer, $\mathbf{b}_2$ be the bias of the second layer.

The described architecture can be written mathematically as:

$$\hat{y} = \sigma(\mathbf{W}_2 \left[\sigma \left(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1\right)\right] + \mathbf{b}_2)$$

where $\hat{y}$ is a scalar output of the net when passing in the single datapoint $\mathbf{x}$, the additions are element wise additions, and the sigmoid is an element wise sigmoid.

1. Let:
   - N be the number of datapoints we have
   - M be the dimensionality of the data
   - H be the size of the hidden dimension of the first layer. Here, hidden dimension is used to describe the dimension of the resulting value after going through the layer. Based on the problem description, the hidden dimension of the second layer should be 1.

   Write out the dimensionality of each of the parameters, and of the intermediate variables:

   $$\mathbf{a}_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1, \qquad \mathbf{z}_1 = \sigma(\mathbf{a}_1)$$
   $$a_2 = \mathbf{W}_2 \mathbf{z}_1 + \mathbf{b}_2, \qquad \hat{y} = z_2 = \sigma(a_2)$$

   and make sure they work with the mathematical operations described above. Examining shapes is one of the key ways to debug your code, and can be done using .shape after any numpy array.
2. We will derive the gradient for each of the parameters before we implement backprop. For this question, assume there is only one datapoint $\mathbf{x}$, and that our loss is $L = y \log(\hat{y}) + (1-y) \log(1-\hat{y})$. For all questions, the chain rule will be useful.
   (a) Find $\frac{\partial L}{\partial b_2}$.
   (b) Find $\frac{\partial L}{\partial W_2^h}$, where W_2^h represents the hth element of $\mathbf{W}_2$.
   (c) Find $\frac{\partial L}{\partial b_1^h}$, where b_1^h represents the hth element of $\mathbf{b}_1$. (*Hint: Note that only the hth element of $\mathbf{a}_1$ and $\mathbf{z}_1$ depend on b_1^h - this should help you with how to use the chain rule.)
   (d) Find $\frac{\partial L}{\partial W_1^{h,j}}$, where $W_1^{h,j}$ represents the element in row h, column j in $\mathbf{W}_1$.

Problem 2 (Backprop Cont.)

Continued from previous page.

3. Implement the gradient updates for the MLP with $H = 75$ with the given $\mathbf{X} \in \mathbb{R}^{N \times M}$ and $\mathbf{y} \in \mathbb{R}^N$ generated in the skeleton code for Q2 provided in the notebook. You will need to complete the forward and get_grad functions. The get_grad function should return the sum of each gradient over all (N) points. Feel free to use helper functions. Additionally, there is a PyTorch implementation of the gradient updates. PyTorch is Python library that supports automatic gradients (autograd), so the output of the torch implementation can be treated as a ground truth to compare to. Along with your code, include some screenshot showing your gradients match the torch autograd ones.

   If you are iterating through each datapoint and updating the parameters one element at a time (which is a good way to start!), note the time the updates takes. There are two places where we can vectorize our code to make it faster.

   (a) For each specific datapoint, we can update the entire gradient at once. To get started on this, generalize your expression for $\frac{\partial L}{\partial W_2^h}$ to write an explicit expression for $\frac{\partial L}{\partial \mathbf{W}_2}$. Can you do similar things for the other gradient updates?

   (b) Then, we can batch gradient updates across all the datapoints.

   Update your code to be vectorized if it was not originally. For this question, the bulk of the speedup should come from the first vectorization and is the only part required. The second vectorization is bonus (and good practice!). How much speedup do you get (per one pass through the data)? If you don't have a baseline to compare to, the staff no vectorize solution took about 17s.

Solution:

Problem 3 (Practical Supervised Learning)

In this problem, we will try several methods on a real data set about housing prices in Boston. Each datapoint represents a town. There are 13 features including per-capita crime rate and teacher-student ratio. The target is the median value of a house in the town (in thousands). To measure performance, we will use MSE. We provide code to get the data and split the data into a train set and a test set in T3.py. Use the train split for all development (including validation and tuning). Use the test set to report final test performance (i.e. only for part 8).

You may (and probably should) use existing software packages (e.g. sklearn) for this assignment.

1. It's always good to start simple! Fit a linear regression to the train data and report train MSE. Then report a 5-fold cross validation MSE.
2. Fit a ridge regression to the data and report train and cross-validation performance. Did this improve performance compared to linear regression? What does this tell you about our data and linear model?
3. Examine the dataset. What sorts of data are there? Do you think a simple linear model models the data well? Hypothesize if more complex models that include non-linearities would improve performance.
4. Fit a MLP Regressor to the dataset. Start with a 2-hidden-layer MLP with hidden sizes 75 each (i.e. pass in [75,75,] to the corresponding argument for the sklearn model). Report train and cross-validation performance. Compare the difference between train and validation performance for the MLP and for the previous linear models. Is this expected?
5. Tune the MLP Regressor. One way you can tune is to do a grid search over certain hyperparameters. Report the relevant hyperparameters of your best model and the train and validation performance.
6. Fit a Random Forest Regressor to the data and report train and validation performance. Both the Random Forest and MLP can capture non-linearities – why do you think the difference in performance is what it is (some things to consider: dataset size, ease of training, tuning).
7. Tune for k and report the train and validation performance of your best kNN regressor. Why do you think kNN cannot match the performance of a MLP or Random Forest? How does kNN compare computationally to parametric methods during train time (especially as data set size increases)? What about during test time?
8. Finally, report test performance for all the models mentioned previously. Comment briefly on validation vs. test performance.

Solution:

Name

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes?

Calibration

Approximately how long did this homework take you to complete (in hours)?