Homework 2: Classification and Bias-Variance Trade-offs

Introduction

This homework is about classification and bias-variance trade-offs. In lecture we have primarily focused on binary classifiers trained to discriminate between two classes. In multiclass classification, we discriminate between three or more classes. Most of the material for Problem 1 and Problem 3, and all of the material for Problem 2 will be covered by the end of the Tuesday 2/8 lecture. The rest of the material will be covered by the end of the Thursday 2/10 lecture. We encourage you to read CS181 Textbook's Chapter 3 for more information on linear classification, gradient descent, classification in the discriminative setting (covers multiclass logistic regression and softmax), and classification in the generative setting. Read Chapter 2.8 for more information on the trade-offs between bias and variance.

As a general note, for classification problems we imagine that we have the input matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ (or perhaps they have been mapped to some basis $\mathbf{\Phi}$, without loss of generality) with outputs now "one-hot encoded." This means that if there are K output classes, rather than representing the output label y as an integer $1, 2, \ldots, K$, we represent $\mathbf{y}$ as a "one-hot" vector of length K. A "one-hot" vector is defined as having every component equal to 0 except for a single component which has value equal to 1. For example, if there are $K = 7$ classes and a particular data point belongs to class 3, then the target vector for this data point would be $\mathbf{y} = [0, 0, 1, 0, 0, 0, 0]$. We will define C_1 to be the one-hot vector for the 1st class, C_2 for the 2nd class, etc. Thus, in the previous example $\mathbf{y} = C_3$. If there are K total classes, then the set of possible labels is $\{C_1 \ldots C_K\} = \{C_k\}_{k=1}^K$. Throughout the assignment we will assume that each label $\mathbf{y} \in \{C_k\}_{k=1}^K$ unless otherwise specified. The most common exception is the case of binary classification ($K = 2$), in which case labels are the typical integers $y \in \{0, 1\}$.

In problems 1 and 3, you may use `numpy` or `scipy`, but not `scipy.optimize` or `sklearn`. Example code given is in Python 3.

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 'HW2'**. Remember to assign pages for each question. **You must include your plots in your writeup PDF.** The supplemental files will only be checked in special cases, e.g. honor code issues, etc.

Please submit your **LaTeX file and code files to the Gradescope assignment 'HW2 - Supplemental'**.

Problem 1 (Exploring Bias and Variance, 10 pts)

In this problem, we will explore the bias and variance of a few different model classes when it comes to logistic regression.

Consider the true data generating process $y \sim \text{Bern}(f(x))$, $f(x) = 0.4 \times \sin(1.2x) + 0.5$, where $x \in [-3, 3]$, and $y \in \{0, 1\}$. Recall that for a given x, bias and variance are defined in terms of expectations *over randomly drawn datasets* D from this underlying data distribution:

$$\text{Bias}[\hat{f}(x)] = \mathbb{E}_D[\hat{f}(x)] - f(x)$$
$$\text{Variance}[\hat{f}(x)] = \mathbb{E}_D[(\hat{f}(x) - \mathbb{E}_D[\hat{f}(x)])^2]$$

Here, $\hat{f}(x)$ is our estimator (learned through logistic regression on a given dataset D). We will directly explore the bias-variance trade-off by drawing multiple such datasets and fitting different logistic regression models to each. Remember that we, the modelers, do not usually see the true data distribution. Knowledge of the true $f(x)$ is only exposed in this problem to (1) make possible the simulation of drawing multiple datasets, and (2) to serve as a pedagogical tool in allowing verification of the true bias.

1. Consider the three bases $\phi_1(x) = [1, x]$, $\phi_2(x) = [1, x, x^2]$, $\phi_3(x) = [1, x, x^2, x^3, x^4, x^5]$. For each of these bases, generate 10 datasets of size $N = 30$ using the starter code provided, and fit a logistic regression model using $\text{sigmoid}(w^T \phi(x))$ to each dataset by using gradient descent to minimize the negative log likelihood. This means you will be running gradient descent 10 times for each basis, once for each dataset. Note that the classes are represented with 0's and 1's.

   Use random starting values of w, $\eta = 0.001$, take 10,000 update steps for each gradient descent run, and make sure to average the gradient over the data points (for each step). These parameters, while not perfect, will ensure your code runs in a reasonable amount of time. The emphasis of this problem is on capturing the bias-variance trade-off, so don't worry about attaining perfect precision in the gradient descent as long as this trade-off is captured in the final models.

   Note: Overflow RuntimeWarnings due to `np.exp` should be safe to ignore, if any. Also, to reduce stress from randomness in students' solutions (due to randomized weight initialization differences), in line 109 of the `T2_P1.py` starter code, we call `np.random.seed(1738)` to set a deterministic random seed. Please do not change this! In addition, please do not change the randomized weight initialization code in lines $42 - 46$.

2. Create three plots, one for each basis. Starter code is available which you may modify. By default, each plot displays three types of functions: (1) the true data-generating distribution $f(x)$ (the probability that $y = 1$ for different x). (2) all 10 of the prediction functions learned from each randomly drawn dataset, and (3) the mean of the 10 prediction functions. Moreover, each plot also displays 1 of the randomly generated datasets and highlights the corresponding prediction function learned by this dataset.

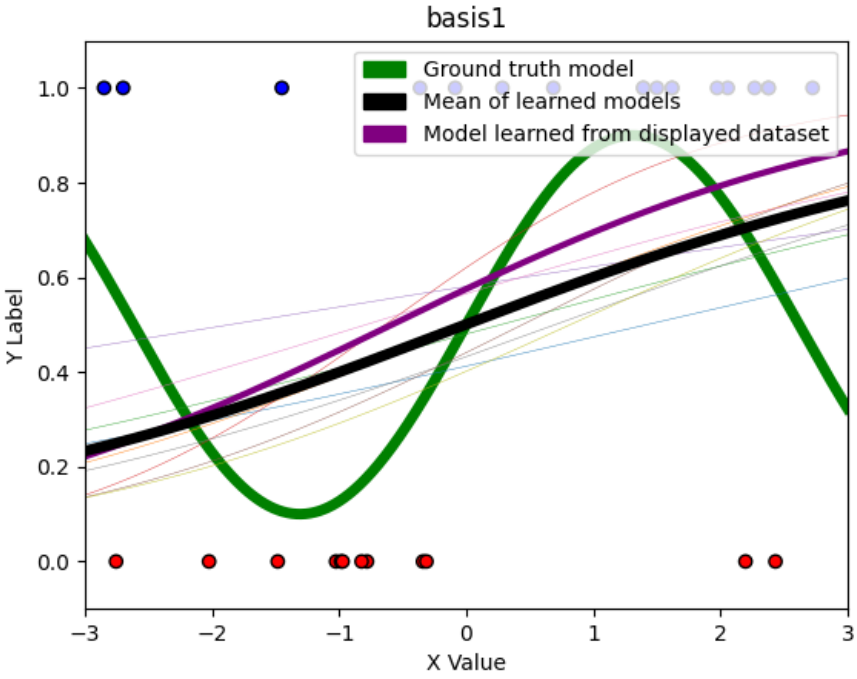
3. How are bias and variance reflected in the 3 types of curves on the graphs? How do the fits of the individual and mean prediction functions change? Keeping in mind that none of the model classes match the true generating process exactly, discuss the extent to which each of the bases approximates the true process.

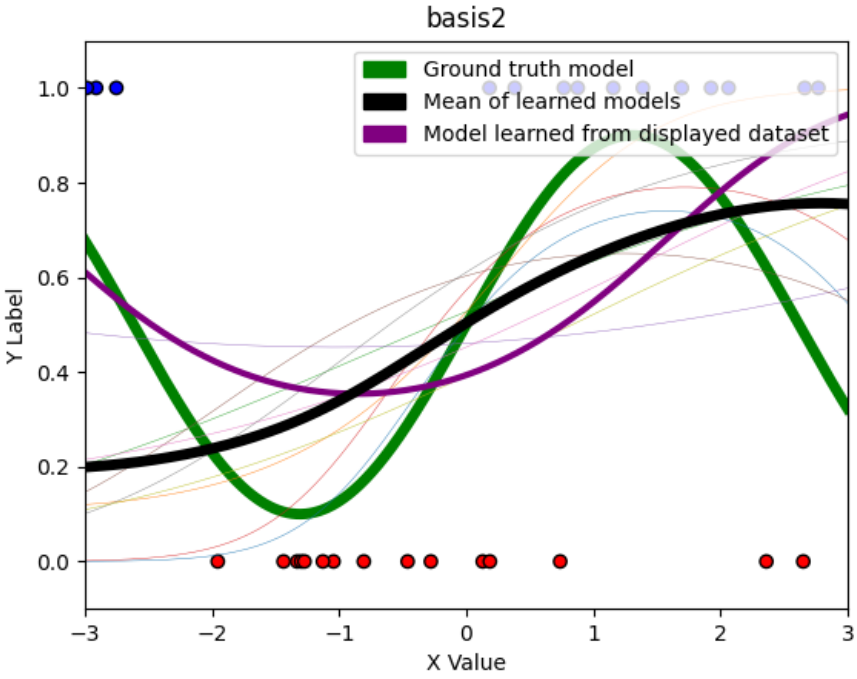
   Note: In this problem, we are not interested in whether the model is more biased for certain inputs x compared to other inputs x'. We are interested in the overall bias and variance of $\hat{f}(x)$ across the different basis choices. In other words, we want to investigate how the bias between $\hat{f}(x)$ and the ground truth as well as the variance of $\hat{f}(x)$ will be different over different basis choices.

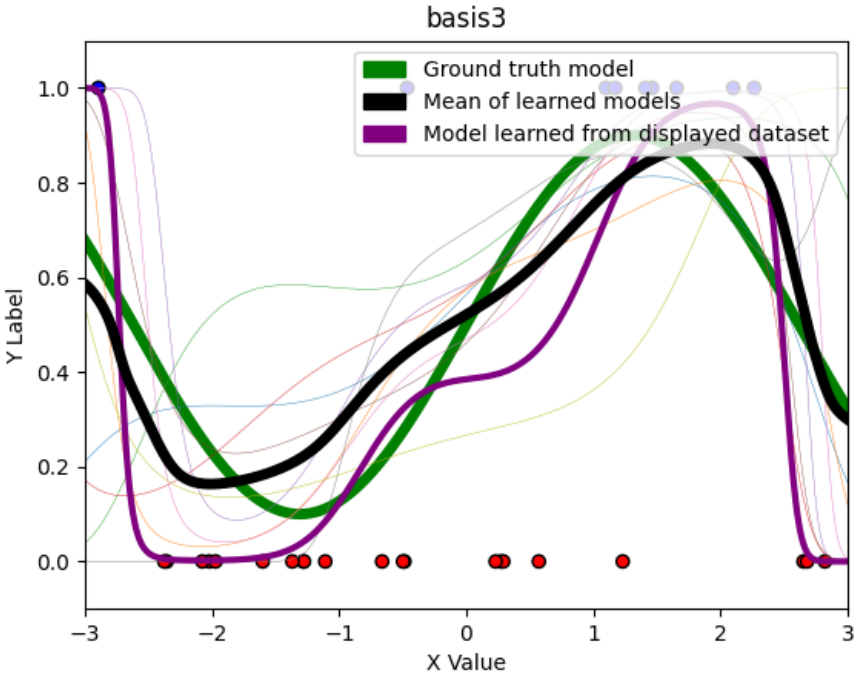
4. If we were to increase the size of each dataset drawn from $N = 30$ to a larger number, how would the variance change? The bias? Why might this be the case?

Solution

1. This part of the problem is implemented in T2_P1.py
2. Below are the three plots, each separated on a different page.


basis1
Ground truth model
Mean of learned models
Model learned from displayed dataset
Y Label
X Value
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3


basis2
Ground truth model
Mean of learned models
Model learned from displayed dataset
Y Label
X Value
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3


basis3
Ground truth model
Mean of learned models
Model learned from displayed dataset
Y Label
X Value
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3

3. **How are bias and variance reflected in the 3 types of curves on the graphs?**

   Bias and variance are reflected in the 3 types of curves on the graphs by

   1. How much models of a given type vary when new datasets are used (e.g., how different the thin lines on the graph are from each other). This visual component corresponds to the variance of the models.

   2. How a single given model fits the dataset (e.g., how well the purple line, which was fit using the displayed dataset, fit to the displayed dataset))). This visual component corresponds to the bias of models.

   How do the fits of the individual and mean prediction functions change?

   For the simpler models, such as those that use basis1, the individual predictions functions don't vary all that much. As the complexity increases when we move on to basis2 and basis3, individual prediction functions begin to vary a lot more from each other. This is due to the ability of the more complex models to fit their curves to correctly classify a given dataset.

   As for the mean prediction functions, they become significantly more accurate as we move on to more complex bases. The mean prediction function for models that use basis3 very closely resembles the ground truth model. It is clear in this case, however that it is only the mean prediction function that is accurate and not the individual models that are accurate, as the model learned from the displayed dataset (in purple in the plot for basis3) is very clearly fitting its dataset much more closely than it is resembling the ground truth model.

   Keeping in mind that none of the bases approximates the true generating process exactly, discuss the extent to which each of the bases approximates the true process.

   basis1 is unable to approximate the true process which much accuracy at all as it is simply unable to model the form of the true process with only non-quadratic terms of the x-values.

   basis2 is not too accurate for the same reason, however, it does have slightly less bias and more variance.

   basis3 is able to approximate the true process very well as it uses a more complete polynomial of $\mathbf{x}$ as its basis. It does tend, however, to fit more closely to the data points themselves than to the true process, which is the downside of this basis.

4. Increasing the dataset size would **decrease** the variance, given that models would be required to use their weights to fit across more points. This is the opposite of what would happen if the dataset size were to be decreased, which would enable the models (especially those fit using basis3) to over-fit their weights for the data points available. The loss for this scenario may even become zero, as basis3 could exactly fit a low enough given amount of data points.

   As for the bias, increasing the dataset size would **decrease** the bias as each model would be given more information from which to tune their weights. More technically speaking, gradient descent would be able to take advantage of much more accurate gradients when optimizing weights.

Problem 2 (Maximum likelihood in classification, 15pts)

Consider now a generative K-class model. We adopt class prior $p(\mathbf{y} = C_k; \boldsymbol{\pi}) = \pi_k$ for all $k \in \{1, \ldots, K\}$ (where π_k is a parameter of the prior). Let $p(\mathbf{x}|\mathbf{y} = C_k)$ denote the class-conditional density of features $\mathbf{x}$ (in this case for class C_k). Consider the data set $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ where as above $\mathbf{y}_i \in \{C_k\}_{k=1}^K$ is encoded as a one-hot target vector and the data are independent.

1. Write out the log-likelihood of the data set, $\ln p(D; \boldsymbol{\pi})$.
2. Since the prior forms a distribution, it has the constraint that $\sum_k \pi_k - 1 = 0$. Using the hint on Lagrange multipliers below, give the expression for the maximum-likelihood estimator for the prior class-membership probabilities, i.e. $\hat{\pi}_k$. Make sure to write out the intermediary equation you need to solve to obtain this estimator. Briefly state why your final answer is intuitive.

For the remaining questions, let the class-conditional probabilities be Gaussian distributions with the same covariance matrix

$$p(\mathbf{x}|\mathbf{y} = C_k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}), \text{ for } k \in \{1, \ldots, K\}$$

and different means $\boldsymbol{\mu}_k$ for each class.

3. Derive the gradient of the log-likelihood with respect to vector $\boldsymbol{\mu}_k$. Write the expression in matrix form as a function of the variables defined throughout this exercise. Simplify as much as possible for full credit.
4. Derive the maximum-likelihood estimator $\hat{\mu}_k$ for vector $\boldsymbol{\mu}_k$. Briefly state why your final answer is intuitive.
5. Derive the gradient for the log-likelihood with respect to the covariance matrix $\boldsymbol{\Sigma}$ (i.e., looking to find an MLE for the covariance). Since you are differentiating with respect to a *matrix*, the resulting expression should be a matrix!
6. Derive the maximum likelihood estimator $\hat{\Sigma}$ of the covariance matrix.

Hint: Lagrange Multipliers. Lagrange Multipliers are a method for optimizing a function f with respect to an equality constraint, i.e.

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } g(\mathbf{x}) = 0.$$

This can be turned into an unconstrained problem by introducing a Lagrange multiplier λ and constructing the Lagrangian function,

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}).$$

It can be shown that it is a necessary condition that the optimum is a critical point of this new function. We can find this point by solving two equations:

$$\frac{\partial L(\mathbf{x}, \lambda)}{\partial \mathbf{x}} = 0 \quad \text{and} \quad \frac{\partial L(\mathbf{x}, \lambda)}{\partial \lambda} = 0$$

Cookbook formulas. Here are some formulas you might want to consider using to compute difficult gradients. You can use them in the homework without proof. If you are looking to hone your matrix calculus skills, try to find different ways to prove these formulas yourself (will not be part of the evaluation of this homework). In general, you can use any formula from the matrix cookbook, as long as you cite it. We opt for the following common notation: $\mathbf{X}^{-\top} := (\mathbf{X}^\top)^{-1}$

$$\frac{\partial \mathbf{a}^\top \mathbf{X}^{-1} \mathbf{b}}{\partial \mathbf{X}} = -\mathbf{X}^{-\top} \mathbf{a}\mathbf{b}^\top \mathbf{X}^{-\top}$$
$$\frac{\partial \ln |\det(\mathbf{X})|}{\partial \mathbf{X}} = \mathbf{X}^{-\top}$$

Solution

1. For each data point $(\mathbf{x}, \mathbf{y}) \in \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ we have

$$\begin{aligned} p(\mathbf{x}_i, \mathbf{y}_i) &= p(\mathbf{x}_i|\mathbf{y}_i)p(\mathbf{y}_i) \\ &= \prod_{k=1}^{K} (p(\mathbf{x}_i|\mathbf{y}_i = C_k)\pi_k)^{\mathbf{y}_{ik}} \end{aligned}$$

where $\mathbf{y}_{ij}$ is the jth element of $\mathbf{y}_i$. We add this exponent as we only want the probability that a feature set $\mathbf{x}_i$ gets classified as $\mathbf{y}_i = C_k$ if that feature set does in fact get classified that way. This makes the likelihood of the data set

$$\begin{aligned} p(D; \boldsymbol{\pi}) &= p(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N; \pi) \\ &= \prod_{i=1}^{N} \prod_{k=1}^{K} (p(\mathbf{x}_i|\mathbf{y}_i = C_k)\pi_k)^{\mathbf{y}_{ik}} \end{aligned}$$

and the log-likelihood of the data set

$$\begin{aligned} \ln p(D; \boldsymbol{\pi}) &= \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{y}_{ik} (\ln(p(\mathbf{x}_i|\mathbf{y}_i = C_k)\pi_k) \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{y}_{ik} (\ln p(\mathbf{x}_i|\mathbf{y}_i = C_k) + \ln \pi_k) \end{aligned}$$

2. We want to find the following, which can be found using Lagrange Multipliers.

$$\min_{\boldsymbol{\pi}} \ln p(D; \boldsymbol{\pi}) \text{ s.t. } \sum_k \pi_k - 1 = 0$$

This can be turned into an unconstrained problem by introducing a Lagrange multiplier λ and constructing the Lagrangian function as follows

$$L(\boldsymbol{\pi}, \lambda) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{y}_{ik} (\ln p(\mathbf{x}_i|\mathbf{y}_i = C_k) + \ln \pi_k) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

Now we take the partial derivative with respect to to λ and with respect to π_k, to find the optimal $\boldsymbol{\pi}$. Taking the derivative with respect to π_k and setting it to zero yields

$$\frac{\partial}{\partial \pi_k} L(\boldsymbol{\pi}, \lambda) = \frac{1}{\pi_k} \sum_{i=1}^{N} y_{ik} + \lambda = 0$$

$$\implies \pi_k = -\frac{1}{\lambda} \sum_{i=1}^{N} y_{ik} = -\frac{N_k}{\lambda}$$

where N_k is the number of data points that belong to the class C_k. Taking the partial derivative with respect to λ yields the following

$$\frac{\partial}{\partial \lambda} L(\boldsymbol{\pi}, \lambda) = \sum_{k=1}^{K} \pi_k - 1 = 0 \implies \sum_{k=1}^{K} \pi_k = 1$$

in order to solve for λ we plug in our equation from before:

$$\sum_{k=1}^{K} \pi_k = 1 \implies \sum_{k=1}^{K} -\frac{N_k}{\lambda} = 1 \implies -\frac{N}{\lambda} = 1 \implies \lambda = -N$$

and finally, we can solve for our optimal π_k, which together gives us our optimal $\boldsymbol{\pi}$.

$$\pi_k = -\frac{N_k}{\lambda} = \frac{N_k}{N}$$

This final answer is intuitive as it is simply the fraction of all of our data points that belong to a class C_k.

3. Now that we have defined the class-conditional probabilities as Gaussian distributions, our log-likelihood becomes the following, where p is the number of variables/features in each $\mathbf{x}_i$ of $(\mathbf{x}_i, \mathbf{y}_i) \in \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$

$$\ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{y}_{ik} (\ln \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}) + \ln \pi_k)$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(\ln \left(\frac{1}{2\pi^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \right) \right) + \ln \pi_k \right)$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(-\frac{1}{2} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) + \ln \pi_k \right)$$

Taking the gradient with respect to $\boldsymbol{\mu}_k$ yields the following

$$\nabla_{\boldsymbol{\mu}_k} \ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(\boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \right)$$

4. In order to find the MLE of $\boldsymbol{\mu}_k$ we can use the gradient from above and set it to zero to find our optimal $\hat{\mu}_k$.

$$\nabla_{\boldsymbol{\mu}_k} \ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(\boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \right) = 0$$

$$\implies \hat{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} x_i$$

where, again, N_k is the number of data points that belong to the class C_k. This solution is intuitive as it is the mean of the $\mathbf{x}_i$ whose $\mathbf{y}_i = C_k$.

5. As before, our log-likelihood is the following

$$\ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(-\frac{1}{2} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) + \ln \pi_k + C \right)$$

where C is a constant. Taking the gradient with respect to $\boldsymbol{\Sigma}$ yields the following

$$\nabla_{\boldsymbol{\Sigma}} \ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(-\frac{1}{2} \nabla_{\boldsymbol{\Sigma}} \ln |\boldsymbol{\Sigma}| - \frac{1}{2} \nabla_{\boldsymbol{\Sigma}} (\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \right)$$

From equation 57 in the Matrix Cookbook we have the following

$$\nabla_{\boldsymbol{\Sigma}} \ln |\boldsymbol{\Sigma}| = -\nabla_{\boldsymbol{\Sigma}} \ln |\boldsymbol{\Sigma}^{-1}| = -\boldsymbol{\Sigma}^\top = -\boldsymbol{\Sigma}$$

and from equation 72 in the Matrix Cookbook we have the following

$$\nabla_{\boldsymbol{\Sigma}} (\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) = (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top$$

Using these two facts, our gradient becomes the following

$$\nabla_{\boldsymbol{\Sigma}} \ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \left(\frac{1}{2} \boldsymbol{\Sigma} - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \right)$$

$$= \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \boldsymbol{\Sigma} - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top$$

$$= \frac{N}{2} \boldsymbol{\Sigma} - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top$$

where, again, N_k is the number of data points classified as C_k.

6. Using the gradient from above and setting it to zero we get the following

$$\nabla_{\boldsymbol{\Sigma}} \ln p(D; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{N}{2} \boldsymbol{\Sigma} - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top = 0$$

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top$$

Problem 3 (Classifying Stars, 15pts)

You're tasked with classifying three different kinds of stars using their magnitudes and temperatures. See star.png for a plot of the data, adapted from http://astrosci.scimuze.com/stellar_data.htm and available as `data/hr.csv`, which you will find in the Github repository.

The CSV file has three columns: type, magnitude, and temperature. The first few lines look like this:

```
Type,Magnitude,Temperature
Dwarf,-5.8,-0.35
Dwarf,-4.1,-0.31
...
```

In this problem, you will code up 4 different classifiers for this task:

a) **A three-class generalization of logistic regression**, also known as softmax regression, in which you implement gradient descent on the negative log-likelihood. In Question 2 you will explore the effect of using different values for the learning rate η (`self.eta`) and regularization strength λ (`self.lam`). Make sure to include a bias term and to use L2 regularization. See CS181 Textbook's Chapter 3.6 for details on multi-class logistic regression and softmax. For your implementation, use the loss and gradient expressions provided there.

b) **A generative classifier with Gaussian class-conditional densities with a *shared covariance* matrix** across all classes. Feel free to re-use your Problem 2 results.

c) **Another generative classifier with Gaussian class-conditional densities , but now with a *separate covariance* matrix** learned for each class. (Note: The staff implementation can switch between the two Gaussian generative classifiers with just a few lines of code.)

d) **A kNN classifier** in which you classify based on the $k = 1, 3, 5$ nearest neighbors and the following distance function:

$$dist(star_1, star_2) = ((mag_1 - mag_2)/3)^2 + (temp_1 - temp_2)^2$$

where nearest neighbors are those with the smallest distances from a given point.

Note 1: When there are more than two labels, no label may have the majority of neighbors. Use the label that has the most votes among the neighbors as the choice of label.

Note 2: The grid of points for which you are making predictions should be interpreted as our test space. Thus, it is not necessary to make a test point that happens to be on top of a training point ignore itself when selecting neighbors.

After implementing the above classifiers, complete the following exercises:

1. Plot the decision boundaries generated by each classifier for the dataset. Include them in your PDF. Identify the similarities and differences among the classifiers. What explains the differences?

2. For logistic regression only, make a plot with "Number of Iterations" on the x-axis and "Negative Log-Likelihood Loss" on the y-axis for several configurations of the hyperparameters η and λ. Specifically, try the values 0.05, 0.01, and 0.001 for each hyperparameter. Limit the number of gradient descent iterations to 200,000. What are your final choices of learning rate (η) and regularization strength (λ), and why are they reasonable? How does altering these hyperparameters affect the ability to converge, the rate of convergence, and the final loss (a qualitative description is sufficient)? You only need to submit one plot for your final choices of hyperparameters.

   Note: The *likelihood* of the model is the probability of data given the model—it should not include the regularization term. The *objective* is the combination of the likelihood and the regularizer.

3. For both Gaussian generative models, report the negative log-likelihood loss. Which model has a lower loss, and why? For the separate covariance model, be sure to use the covariance matrix that matches the true class of each data point.

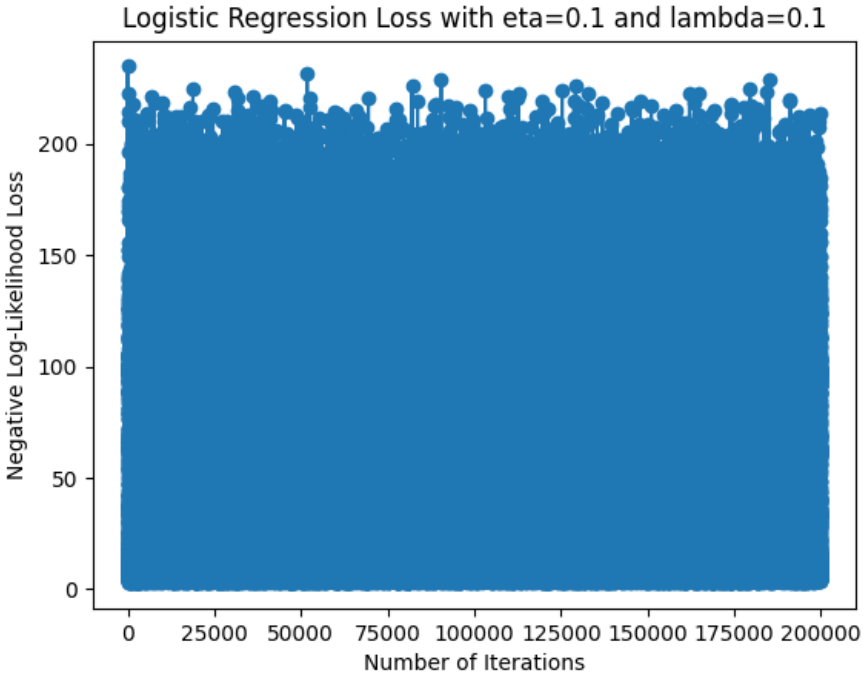
4. Consider a star with Magnitude 6 and Temperature 2. To what class does each classifier assign this star? Do the classifiers give any indication as to whether or not you should trust them?

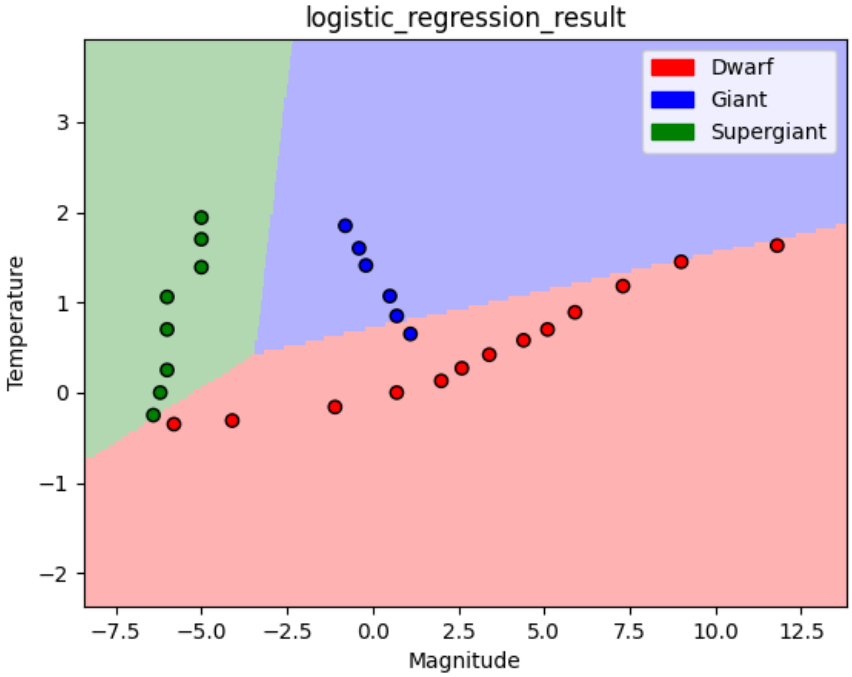
Problem 3 (cont.)

Implementation notes: Run the controller file, `T2_P3.py`, to test your code. Write the actual implementations in the `GaussianGenerativeModel`, `LogisticRegression`, and `KNNModel` classes, which are defined in the three `T2_P3_ModelName.py` files. These classes follow the same interface pattern as sklearn. Their code currently outputs nonsense predictions just to show the high-level interface, so you should replace their `predict()` implementations. You'll also need to modify the hyperparameter values in `T2_P3.py` for logistic regression.

Solution

1.


Logistic Regression Loss with eta=0.1 and lambda=0.1
Negative Log-Likelihood Loss
0
50
100
150
200
0
25000
50000
75000
100000
125000
150000
175000
200000
Number of Iterations


logistic_regression_result
Dwarf
Giant
Supergiant
Temperature
Magnitude
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

Name

Rodney Lafuente Mercado

Collaborators and Resources

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

I briefly collaborated with Henry Kuo on problem 3.

Calibration

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

More than 15.