Assignment #2

Due: 7:59pm EST, Feb 25th, 2022

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 $\boldsymbol{\Phi}$, 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 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.

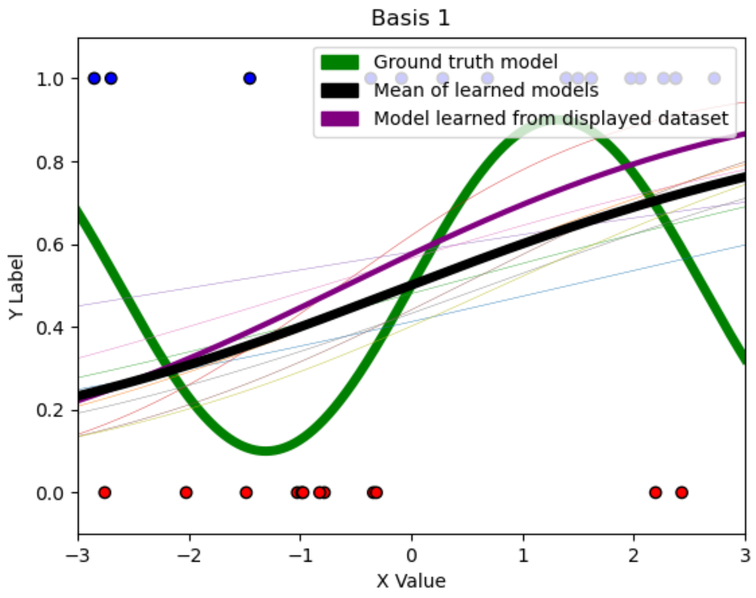
3. How are bias and variance reflected in the 3 types of curves on the graphs? How do the fits of the individual and mean prediction functions change? Keeping in mind that none of the model classes match the true generating process exactly, discuss the extent to which each of the bases approximates the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

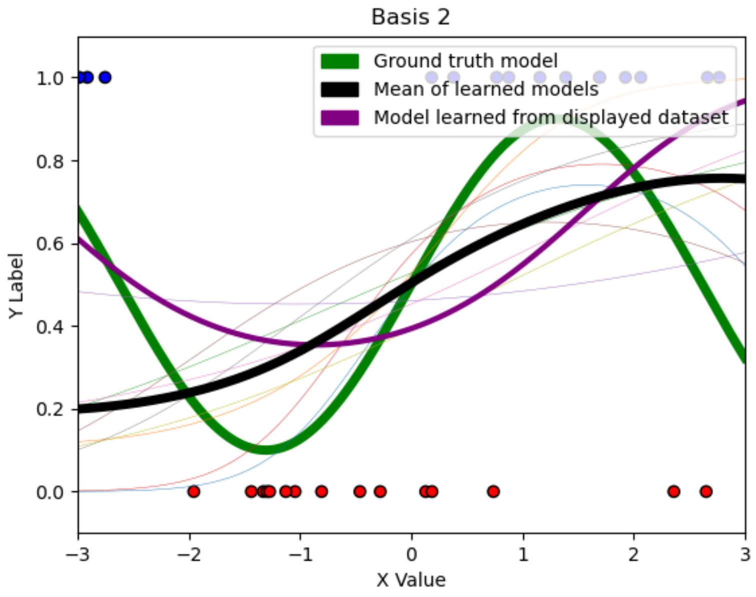
   Note: In this problem, we are not interested in whether the model is more biased for certain inputs x compared to other inputs x'. We are interested in the overall bias and variance of $\hat{f}(x)$ across the different basis choices. In other words, we want to investigate how the bias between $\hat{f}(x)$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

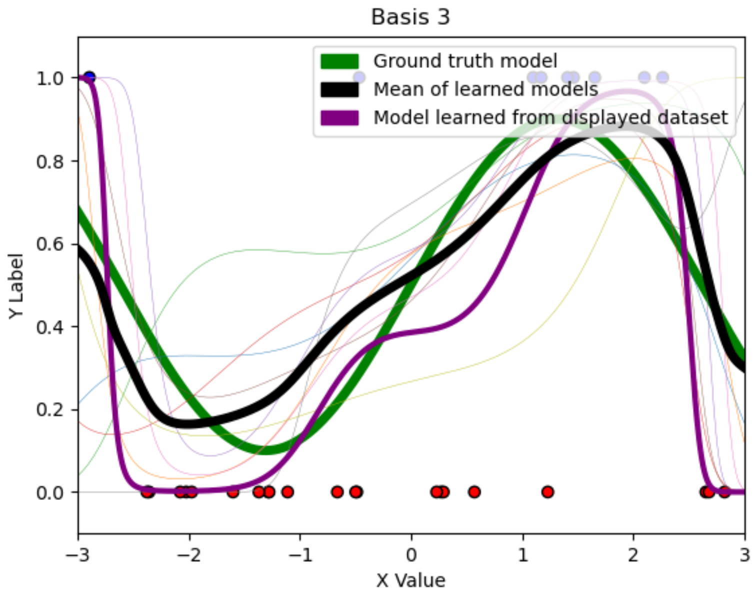
4. If we were to increase the size of each dataset drawn from $N = 30$ to a larger number, how would the variance change? The bias? Why might this be the case?

Solution

2. The plots for the bases are:







3. The third basis has a better bias but a worse variance. Unlike the linear and quadratic bases, it has enough polynomial terms with a high enough degree that it can come close to mimicking the ground truth sinusoidal model. However, because it has more terms with higher degree, it is also able to overfit for individual data samples and therefore has higher variance. On the other hand, the first and second bases are incapable of emulating the ground truth model since lines and parabolas just can't really resemble one period of a sinusoidal curve. As a result, they have high bias. At the same time, these bases are more consistent over different data inputs since they're less flexible and can't adapt to each new input — resulting in lower variance.

   The fit of the first basis (to the ground truth model) is bad for all inputs, while the fit of the second basis is middle of the pack and consistent for all inputs. The fit of the third basis changes depending on the input with some inputs coming fairly close to the ground truth model. Taking the mean of the third basis prediction accounts for the higher variance and produces a model that fits the ground truth model well.

   The second basis provides a mediocre fit with and without averaging, while the third basis provides a good fit when we average the models produced from different data samples. Without any extra techniques, the third basis provides an inconsistent estimate (sometimes bad, sometimes very close), and the first basis provides a bad fit with and without averaging.

4. The bias of the three models should not change since the bias reflects their ability to mimic the ground truth, and this does not change when we add data points: the third basis is still the most capable, and the first and second bases won't be able to get very close. The variance for the first and second bases likely won't change much since they don't have much of an ability to overfit anyway (too few terms in each basis). On the other hand, the variance for the third basis will decrease since it'll become harder to overfit as the number of data points increase, and the size of the dataset becomes much larger than the number of features.

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 $k \in \{1, \dots, K\}$ (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

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

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:

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

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 2

1. By factoring, our likelihood is

$$p(D;\boldsymbol{\pi}) = \prod_{i=1}^{n} p(\mathbf{x}_i, \mathbf{y}_i; \boldsymbol{\pi}) = \prod_{i=1}^{n}\prod_{k=1}^{K} \big(p(\mathbf{x}_i|\mathbf{y}_i = C_k)\pi_k\big)^{I(\mathbf{y}_i = C_k)}$$

where $I(\mathbf{y}_i = C_k)$ is an indicator variable of $\mathbf{y}_i$ being in class k. We take the log of both sides to get

$$\ln p(D;\boldsymbol{\pi}) = \sum_{i=1}^{n}\sum_{k=1}^{K} \ln\left(\big(p(\mathbf{x}_i|\mathbf{y}_i = C_k)\pi_k\big)^{I(\mathbf{y}_i = C_k)}\right) = \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\Big(\ln p(\mathbf{x}_i|\mathbf{y}_i = C_k) + \ln \pi_k\Big).$$

For later use, we'll distribute $I(\mathbf{y}_i = C_k)$ and regroup this expression as

$$\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln p(\mathbf{x}_i|\mathbf{y}_i = C_k) + \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln \pi_k.$$

2. We write our Lagrangian as follows:

$$L(\boldsymbol{\pi}, \lambda, D) = \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln p(\mathbf{x}_i|\mathbf{y}_i = C_k) + \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln \pi_k + \lambda\Big(\sum_k \pi_k - 1\Big).$$

Taking the partial derivative with respective to π_k yields:

$$\frac{\partial L(\boldsymbol{\pi}, \lambda, D)}{\partial \pi_k} = \sum_{i=1}^{n} \frac{I(\mathbf{y}_i = C_k)}{\pi_k} + \lambda$$

since $\sum_k \pi_k$ includes a single π_k term, which has a derivative of 1, and taking the partial derivative with respective to λ yields:

$$\frac{\partial L(\boldsymbol{\pi}, \lambda, D)}{\partial \lambda} = \sum_k \pi_k - 1.$$

Setting both equal to 0 and rearranging yields

$$\sum_{i=1}^{n} \frac{I(\mathbf{y}_i = C_k)}{\pi_k} + \lambda = 0 \implies \frac{1}{\pi_k}\sum_{i=1}^{n} I(\mathbf{y}_i = C_k) = -\lambda \implies \frac{1}{-\lambda}\sum_{i=1}^{n} I(\mathbf{y}_i = C_k) = \pi_k.$$

From here we'll define a_k as $\sum_{i=1}^{n} I(\mathbf{y}_i = C_k)$ or more simply the number of data points in class k. It follows that we can write our previous result as $\pi_k = \frac{a_k}{-\lambda}$, and since this holds for all $k \in \{1, \dots, K\}$, we can substitute it into our second equation:

$$\frac{\partial L(\boldsymbol{\pi}, \lambda, D)}{\partial \lambda} = \sum_k \pi_k - 1 = 0 \implies \sum_k \frac{a_k}{-\lambda} = 1 \implies \frac{1}{-\lambda}\sum_k a_k = 1 \implies \sum_k a_k = -\lambda.$$

Since a_k is defined as the number of data points in class k, and we're summing over all k, $\sum_k a_k$ is simply the size of our dataset. Plugging this back into our previous equality yields

$$\hat{\pi}_k = \frac{a_k}{-\lambda} = \frac{a_k}{\sum_j a_j}.$$

We can interpret the MLE $\hat{\pi}_k$ as the empirical probability of being in class k i.e. the number of data points in class k over the total size of our dataset.

3. We plug $p(\mathbf{x}|\mathbf{y} = C_k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$ into our log likelihood function to see that

$$\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k) \ln f_{\mathbf{x}|\mathbf{y}=C_k}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}) + \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k) \ln \pi_k.$$

Since we're taking the gradient with respect to $\boldsymbol{\mu}_k$, we can ignore all terms without $\boldsymbol{\mu}_k$, and we're left with

$$\sum_{i=1}^{n} I(\mathbf{y}_i = C_k) \ln f_{\mathbf{x}|\mathbf{y}=C_k}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}).$$

For simplicity, we'll rewrite the sum over $i \in \{1, \ldots, n\}$ s.t. $\mathbf{y}_i = k$ since $I(\mathbf{y}_i = C_k) = 0$ for all other i:

$$\sum_{i|\mathbf{y}_i=C_k} \ln f_{\mathbf{x}|\mathbf{y}=C_k}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}).$$

Next we'll plug in the multivariate normal PDF

$$f_{\mathbf{x}|\mathbf{y}=C_k}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)}$$

to get

$$\sum_{i|\mathbf{y}_i=C_k} \ln f_{\mathbf{x}|\mathbf{y}=C_k}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}) = \sum_{i|\mathbf{y}_i=C_k} \ln\left(\frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)}\right)$$

and manipulate using log rules to get

$$-\frac{1}{2}\sum_{i|\mathbf{y}_i=C_k} \ln(\det(2\pi\boldsymbol{\Sigma})) + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k).$$

Since we're taking the gradient with respect to $\boldsymbol{\mu}_k$, we can ignore the first term in the summation. Note that I'll be citing results from the Matrix Cookbook using "(cookbook n)" where n is the number of the result. From here, we'll group $(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}$ and $(\mathbf{x}_i - \boldsymbol{\mu}_k)$ to use the product rule for matrices (cookbook 37):

$$-\frac{1}{2}\sum_{i|\mathbf{y}_i=C_k} \frac{\partial}{\partial \boldsymbol{\mu}_k}\left((\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}\right)(\mathbf{x}_i - \boldsymbol{\mu}_k) + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} \frac{\partial}{\partial \boldsymbol{\mu}_k}(\mathbf{x}_i - \boldsymbol{\mu}_k).$$

Off the bat, we can note that $\frac{\partial}{\partial \boldsymbol{\mu}_k}(\mathbf{x}_i - \boldsymbol{\mu}_k) = -\mathbf{I}$ where I is the $n \times n$ identity matrix and n is the length of $\boldsymbol{\mu}_k$. Next we'll use the result $\frac{\partial \mathbf{x}^T \mathbf{A}}{\partial \mathbf{x}} = \mathbf{A}$ from cookbook 69 and the chain rule to see that

$$\frac{\partial}{\partial \boldsymbol{\mu}_k}\left((\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}\right) = -\boldsymbol{\Sigma}^{-1}$$

(I think the cookbook result is for vectors $\mathbf{a}$ instead of matrices $\mathbf{A}$, but the logic extends, and in my opinion, the result is fairly clear from the matrix multiplication formula). Combining our results yields

$$-\frac{1}{2}\sum_{i|\mathbf{y}_i=C_k} -\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(-\mathbf{I}) = \frac{1}{2}\sum_{i|\mathbf{y}_i=C_k} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}.$$

4. To solve for the MLE, we'll set the gradient from part 3 equal to $\mathbf{0}$ (the vector of all 0's), break up the sum, and drop the $\frac{1}{2}$ multiplier:

$$\frac{1}{2}\sum_{i|\mathbf{y}_i=C_k} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} = \mathbf{0} \implies \sum_{i|\mathbf{y}_i=C_k} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) + \sum_{i|\mathbf{y}_i=C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} = \mathbf{0}.$$

Note that the whole quantity will be $\mathbf{0}$ if both summations are equal to $\mathbf{0}$. We'll assume that this is the case and proceed. Next we can use the distributive property of matrix multiplication over matrix addition to see that

$$\sum_{i|\mathbf{y}_i=C_k} \mathbf{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) = \mathbf{\Sigma}^{-1} \sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i - \boldsymbol{\mu}_k = \mathbf{\Sigma}^{-1}\Bigg(\bigg(\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i\bigg) - a_k\boldsymbol{\mu}_k\Bigg),$$

and

$$\sum_{i|\mathbf{y}_i=C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1} = \Bigg(\sum_{i|\mathbf{y}_i=C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)^T\Bigg)\mathbf{\Sigma}^{-1} = \Bigg(\bigg(\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i^T\bigg) - a_k{\boldsymbol{\mu}_k}^T\Bigg)\mathbf{\Sigma}^{-1}.$$

Note that the final equalities hold since $\boldsymbol{\mu}_k$ does not depend on i and a_k is the scalar defined in part 2 i.e. a_k is the number of data points for which y_i is in class k — clearly this is the same as the number of $\boldsymbol{\mu}_k$'s in our summation. Finally, we can manipulate the second equality to see that

$$\bigg(\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i^T\bigg) - a_k{\boldsymbol{\mu}_k}^T = \mathbf{0} \implies \sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i^T = a_k{\boldsymbol{\mu}_k}^T \implies \bigg(\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i\bigg)^T = a_k{\boldsymbol{\mu}_k}^T \implies \sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i = a_k\boldsymbol{\mu}_k$$

where the middle implication holds since the sum of transposes is the transpose of the sum (cookbook 4). Similarly, the first equality yields

$$\bigg(\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i\bigg) - a_k\boldsymbol{\mu}_k = \mathbf{0} \implies \sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i = a_k\boldsymbol{\mu}_k$$

and both equalities give us

$$\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i = a_k\boldsymbol{\mu}_k \implies \boldsymbol{\mu}_k = \frac{1}{a_k}\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i.$$

Thus both terms in the expression

$$\sum_{i|\mathbf{y}_i=C_k} \mathbf{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) + \sum_{i|\mathbf{y}_i=C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}$$

can simultaneously be $\mathbf{0}$ as we hoped, and $\hat{\mu}_k = \frac{1}{a_k}\sum_{i|\mathbf{y}_i=C_k} \mathbf{x}_i$. We can interpret the MLE $\hat{\boldsymbol{\mu}}_k$ as the average of our data points x_i with y_i in class k or the empirical mean of $\mathbf{x}|\mathbf{y} = C_k$.

5. As in part 3, we plug $p(\mathbf{x}|\mathbf{y} = C_k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma})$ into our log likelihood function to see that

$$\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln f_{\mathbf{x}|\mathbf{y}=C_k}(\mathbf{x}_i;\boldsymbol{\mu}_k,\mathbf{\Sigma}) + \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln \pi_k.$$

Since we're differentiating with respect to $\mathbf{\Sigma}$, we can drop the second tern. Next we'll plug in the normal PDF and expand the log as in part 3 to get

$$-\frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\Bigg(\ln\left(\det\left(2\pi\mathbf{\Sigma}\right)\right) + (\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)\Bigg).$$

Next we can distribute $I(\mathbf{y}_i = C_k)$ and split up the sums:

$$-\frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)\ln\left(\det\left(2\pi\mathbf{\Sigma}\right)\right) - \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k).$$

Note that for each $\mathbf{y}_i$, $I(\mathbf{y}_i = C_k) = 1$ for only 1 k and 0 for all other k values. Thus

$$\sum_{k=1}^{K} I(\mathbf{y}_i = C_k) \ln(\det(2\pi\mathbf{\Sigma})) = \ln(\det(2\pi\mathbf{\Sigma})).$$

From here, we use cookbook 19 to see that $\det(2\pi\mathbf{\Sigma}) = (2\pi)^n \det(\mathbf{\Sigma})$ where n is the length of $\mathbf{x}$, and log rules to see that $\ln(\det(2\pi\mathbf{\Sigma})) = n\ln(2\pi) + \ln(\det(\mathbf{\Sigma}))$. Since the first term does not involve $\mathbf{\Sigma}$, we can drop it when differentiating. Thus we are left with

$$-\frac{1}{2}\sum_{i=1}^{n} \ln(\det(\mathbf{\Sigma})) - \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k).$$

We use the provided cookbook formulas to see that differentiating with respect to $\mathbf{\Sigma}$ will yield

$$-\frac{1}{2}\sum_{i=1}^{n} \mathbf{\Sigma}^{-T} - \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k) - \mathbf{\Sigma}^{-T}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-T}.$$

6. We set this equal to $\mathbf{0}$ where $\mathbf{0}$ is $n \times n$ matrix of all zeroes and multiply through by -2. Next we left multiply by $\mathbf{\Sigma}^T$ and use the distributive property of matrix multiplication over addition to get

$$\sum_{i=1}^{n} \mathbf{I} - \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-T} = \mathbf{0}.$$

Then we right multiply by $\mathbf{\Sigma}^T$ and apply the distributive property once again to get

$$\sum_{i=1}^{n} \mathbf{\Sigma}^T - \sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T = \mathbf{0}.$$

Rearranging and solving yields

$$\mathbf{\Sigma}^T = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} I(\mathbf{y}_i = C_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$$

where n is the size of our dataset. Since covariance matrices are symmetric, $\mathbf{\Sigma} = \mathbf{\Sigma}^T$, and the above expression is the MLE $\hat{\mathbf{\Sigma}}$.

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.

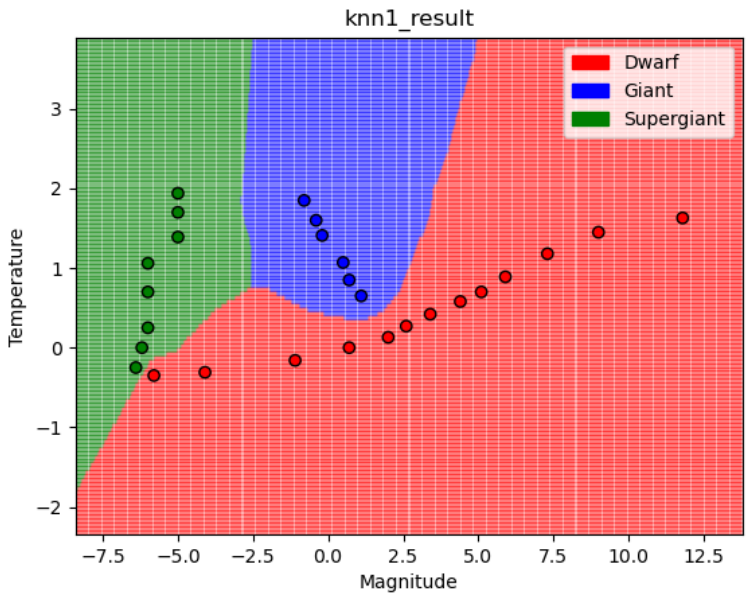
4. Consider a star with Magnitude 6 and Temperature 2. To what class does each classifier assign this star? Do the classifiers give any indication as to whether or not you should trust them?

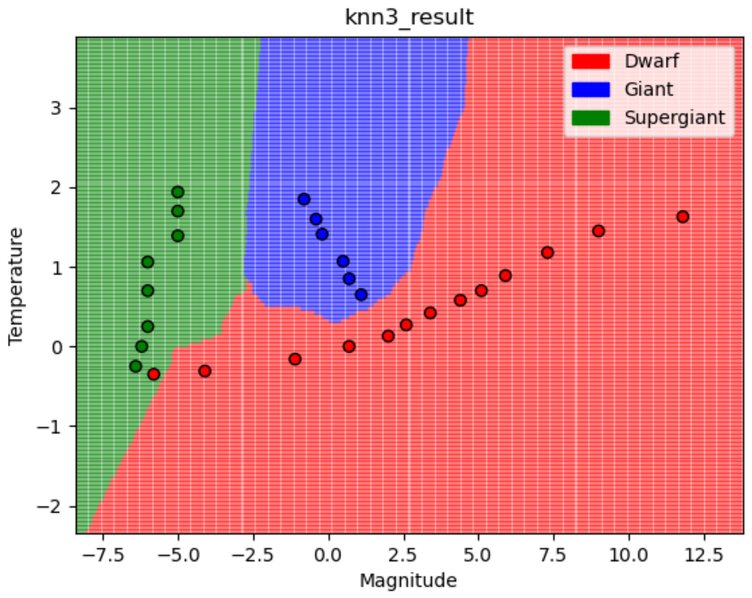
Problem 3 (cont.)

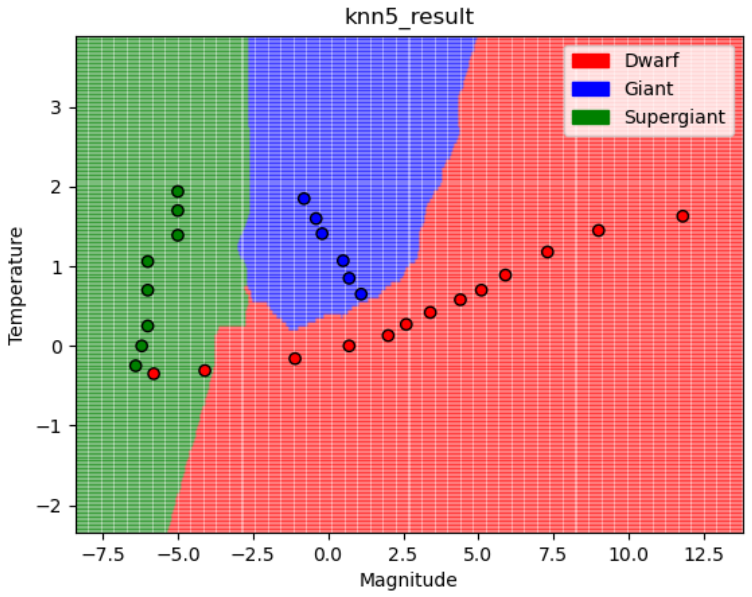
Implementation notes: Run the controller file, `T2_P3.py`, to test your code. Write the actual implementations in the `GaussianGenerativeModel`, `LogisticRegression`, and `KNNModel` classes, which are defined in the three `T2_P3_ModelName.py` files. These classes follow the same interface pattern as 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.

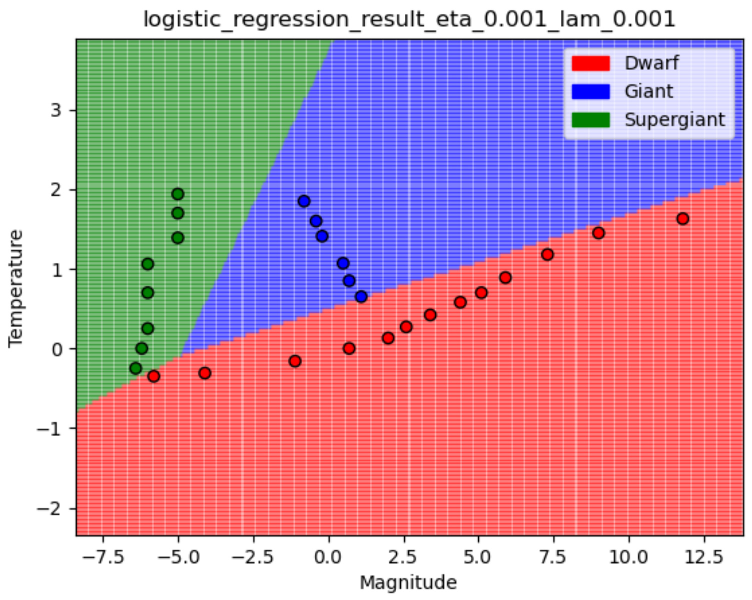
Solution 3

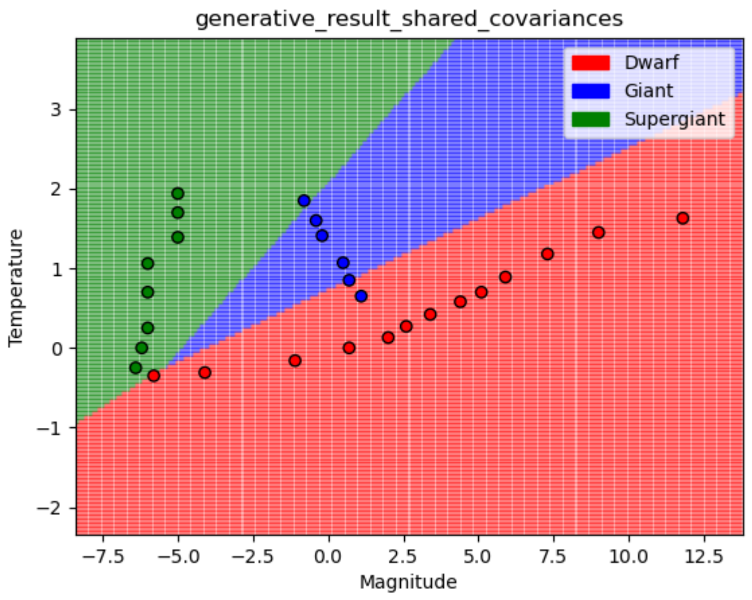
1. The plots are:

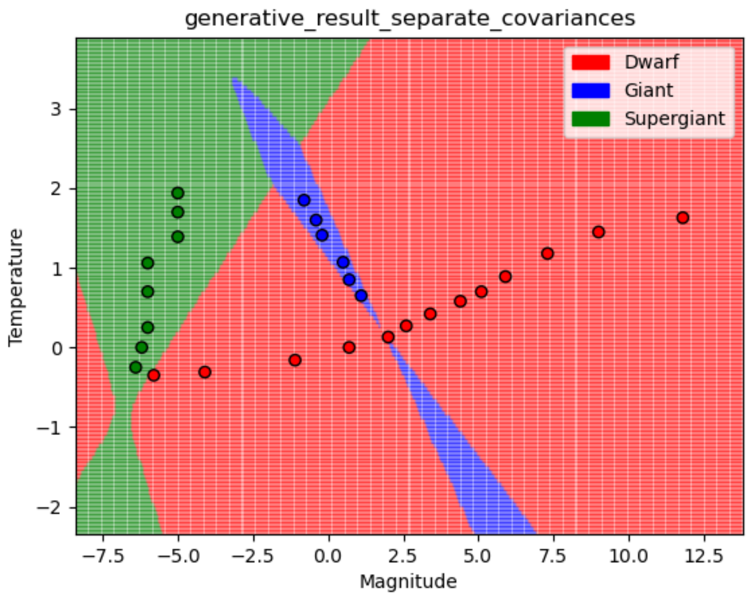



knn3_result
Dwarf
Giant
Supergiant
Temperature
Magnitude
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
−2
−1
0
1
2
3


knn5_result
Dwarf
Giant
Supergiant
Temperature
Magnitude
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
−2
−1
0
1
2
3

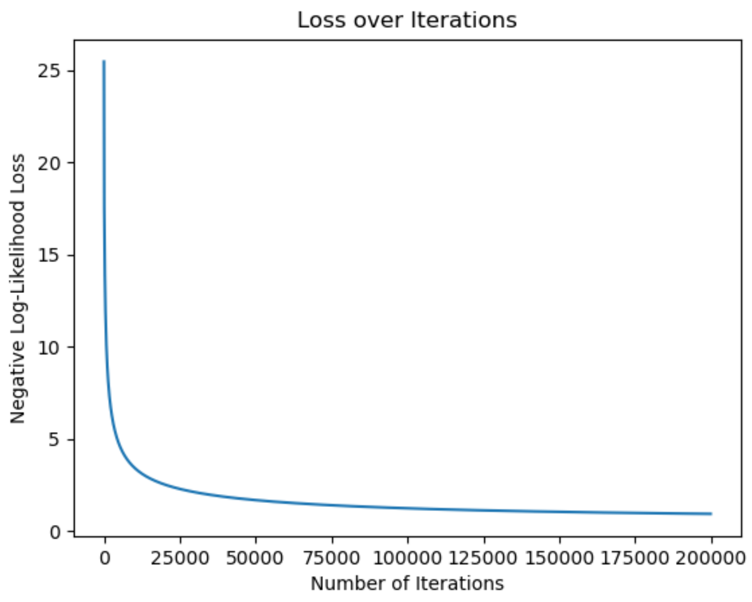

logistic_regression_result_eta_0.001_lam_0.001
Dwarf
Giant
Supergiant
Temperature
Magnitude


generative_result_shared_covariances
Dwarf
Giant
Supergiant
Temperature
Magnitude



2. All combinations of lambda and eta (with the exception of 0.05 and 0.05) seem to perform similarly in terms of final loss, and they all perform very well. When both hyperparameters are equal to 0.5, the loss oscillates up and down over training iterations. In general, as you decrease lambda, the loss seems to converge earlier but they all level out very quickly. I think this is because the model is punished less for higher weights and is willing to move towards the minimum faster. Raising eta seems to increase the rate of convergence as well, and this makes sense since we're telling the model to take bigger steps, so it can get to the minimum faster. However, when both parameters are equal to 0.05, I think we're giving the model too much free reign with the large step size and punishing it too much for higher magnitude weights, so it jumps around a lot and isn't able to commit to/find the minimum. This prevents it from converging unlike the other cases. I think a reasonable step size is 0.001 since we converge fairly quickly anyway, and a low step size allows us to avoid jumping back and forth (while missing the minimum) — this isn't relevant for this particular dataset but may be relevant for another sample. Similarly I think a reasonable lambda is also 0.001 since there doesn't seem to be much of a point with regularizing this particular model (the resulting decision boundaries are all very similar).

   The loss over iterations for these hyperparameters is:



3. The negative log-likelihood loss with separate covariance matrices is -25.855489867624136, and the negative log-likelihood loss with a shared covariance matrix is -91.11263678374112.

   My guess is that when you use separate covariance matrices, the model cares a lot more about the shape of the data in each class i.e. what does x_i look like when you consider the relationship between magnitude and temperature. The probability of a particular data point is then linked to how closely it follows the shape for that class. On the other hand, when you have a shared covariance matrix, the model cares more about the mean of the data in each class, and the probability of a data point is more influenced by how close it is to the mean of that class. I think this is reflected in the decision boundaries for each classifier with the separate covariance classifier having more elaborate shapes that match the shape of the data and the shared covariance classifier having wider patches that seem to be dominated by the nearest class.

   The shared covariance matrix is better, and I think that's because (for this dataset) the distance from the mean is a better predictor of class than how closely the shape of the data point (direction from the mean) resembles the shape of the other data points in the same class.

4. The separate covariance Gaussian Model and the three kNN models predict class 0, while the shared covariance Gaussian model and logistic regression models predict class 1.

   Both the logistic regression and Gaussian generative models provide probabilities, and we can interpret the kNN classifier's suggestion as a probability as well by considering how many of the k nearest neighbors were in the predicted class. We can say that the classifier is more sure for higher probabilities in the first two cases, and the classifier is more sure when more of the neighbors are in the predicted class for the kNN case.

Name

Christy Jestin

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes?
I worked with David Qian and Emil Liu.

Calibration

Approximately how long did this homework take you to complete (in hours)?
The homework took about 16 hours.