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.

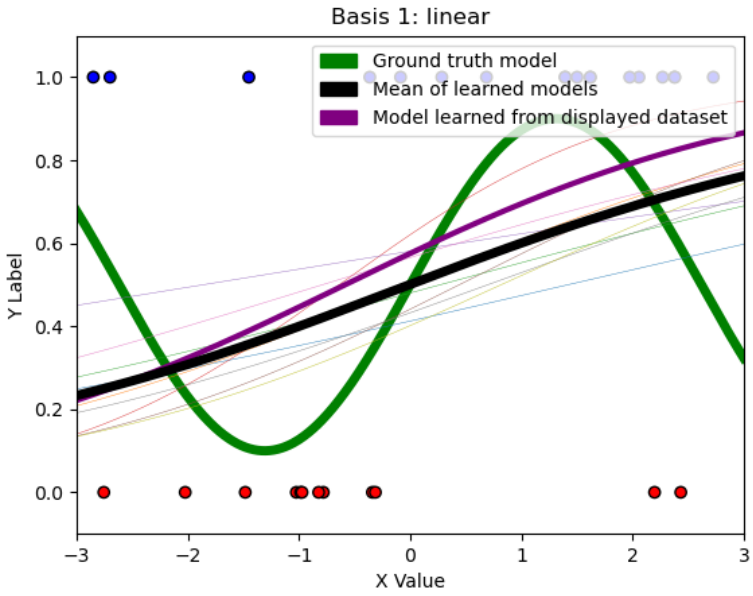
   Note: In this problem, we are not interested in whether the model is more biased for certain inputs x compared to other inputs x'. We are interested in the overall bias and variance of $\hat{f}(x)$ across the different basis choices. In other words, we want to investigate how the bias between $\hat{f}(x)$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

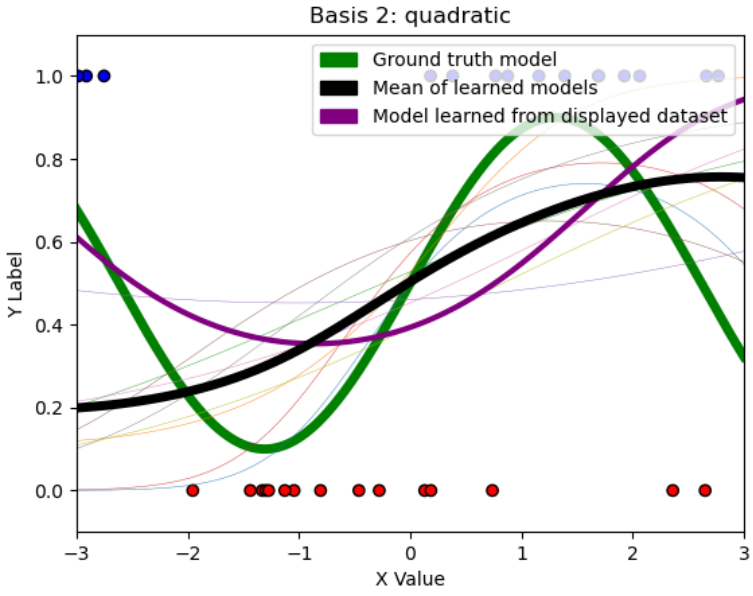
4. If we were to increase the size of each dataset drawn from $N = 30$ to a larger number, how would the variance change? The bias? Why might this be the case?

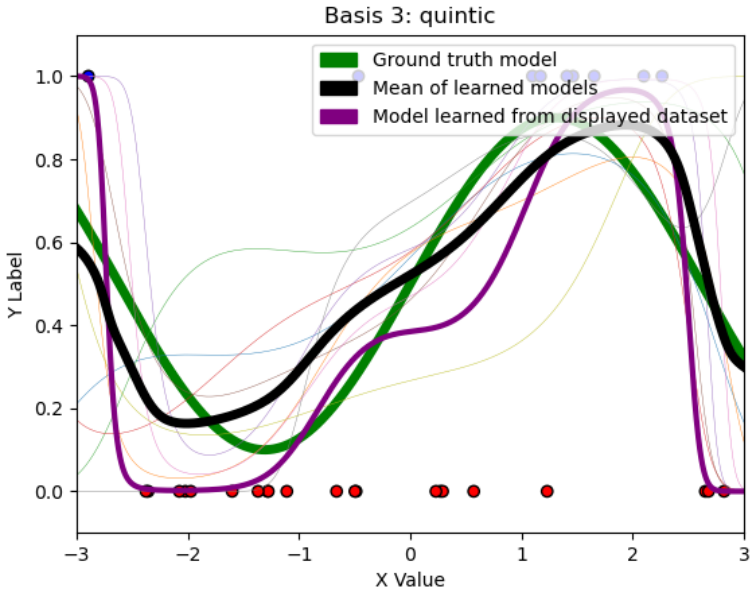
Solution

Answer:

1. See gradient descent implementation as well as functions to create bases in the code file.
2. See the three plots below:







3. As predicted by information from lecture, more expressive models (with higher-order polynomials) lead to less bias but more variance. To see this, note that the best-fit curves for the linear basis are all relatively close together (relatively little variation between data sets) but all are relatively biased (their mean distance from the true GDP model is high), since it's impossible to fit the sine function using a linear model.

   The quadratic basis $\phi(x) = [1, x, x^2]$ lies somewhere in between–it manages to capture some curvature in the true DGP at the cost of significantly different coefficients for different draws of the data set D: notice for example the highlighted purple curve and the brown curve; at some points their slopes are basically opposite.

   Finally, the fifth-order polynomials exhibit very little bias: the mean of the learned models approximates the true DGP very closely. However, the variance is very high given that the model estimates 6 parameters with only 30 data points. For example, notice how far the green curve is from the highlighted purple fit.

4. If we took $N >> 30$, variance would decline for all models: we would have x samples drawn from the entire range of of x values and the probability that two data sets drawn from the same distribution would be vastly different would decrease. This would benefit the more expressive models ϕ_2 and ϕ_3 the most since they suffered from the greatest variance.

   However, increasing the size of the data set would do nothing to eliminate bias. Even asymptotically, the linear model ϕ_1 is not able to capture the features of the true DGP, while a fifth-order polynomial gets quite close. This makes intuitive sense: the variance captures how different the weights fitted on different data sets will be from each other, and if there are only small differences between large data sets drawn from the same ground truth model, the models fitted on these data sets will be quite similar. However, even an asymptotically optimal model of the ground truth whose functional form is restricted to 2 or 3 parameters will on average be far away from the ground truth; this does not change with the size of the data set.

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

Answer:

1. Start by expressing the likelihood as a product of the conditional distribution and the class priors (LOTP) and use independence between individual data points drawn from the distribution (we'll carry this iid assumption trhoughout):

$$p(D;\pi) = \prod_{i=1}^{n} p((x_i, y_i); \pi)$$

$$= \prod_{i=1}^{n} p(x_i|y_i = C_k)p(y_i = C_k|\pi)$$

$$= \prod_{i=1}^{n} p(x_i|y_i = C_k)\pi_k$$

Next, to get the log-likelihood, we take the log which turns the product into a sum:

$$\log p(D;\pi) = \sum_{i=1}^{n} [\log(p(x_i|y_i = C_k)) + \log(\pi_k)]$$

$$= \sum_{i=1}^{n} \log(p(x_i|y_i = C_k)) + \sum_{i=1}^{n} \log(\pi_k) I[y_i = C_k]$$

2. First, we use the provided constraint to set up a Largangian function:

$$\mathcal{L}(D, \pi, \lambda) = \log p(D;\pi) + \lambda \left(\sum_k \pi_k - 1 \right) = \sum_{i=1}^{n} \log(p(x_i|y_i = C_k)) + \sum_{i=1}^{n} \log(\pi_k) + \lambda \left(\sum_k \pi_k - 1 \right)$$

Now taking first order conditions with respect to π_k, the parameter of interest, and λ, we get:

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \frac{1}{\pi_k} \sum_{i=1}^{n} I[y_i = C_k] + \lambda = 0$$

where we added an indicator that is one iff the i-th y is category C_k in order to separate the sums.

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \sum_k \pi_k - 1 = 0$$

Letting N_k be the number of observations with label C_k, we have:

$$\frac{N_k}{\pi_k} = \frac{N_m}{\pi_m}$$

for all m. Then substituting into the second FOC, we get:

$$\sum_m \frac{N_m}{N_k} \pi_k = 1$$

Now, let $\sum_m N_m = n$ be number of observations:

$$\pi_k n = N_k$$

$$\hat{\pi}_k = \frac{N_k}{n}$$

This makes a lot of intuitive sense! Our best estimate for the prior probability of a class is just the proportion of observations from that class!

3. To derive the gradient, we can plug in the Multivariate Gaussian PDF and take the derivative with respect to μ_k, ignoring terms that don't depend on μ_k. This includes the priors as well as all terms where $y_i \neq C_k$, since μ_k does not enter into the density at these points:

$$\log p(D;\pi) = \sum_{i=1}^{n} \log \left(\det(2\pi\mathbf{\Sigma})^{-1/2} \exp(-\frac{1}{2}(x_i - \mu_m)^T \Sigma^{-1}(x_i - \mu_m)) | y_i = C_m \right) + \sum_{i=1}^{n} \log(\pi_m)$$

$$= \sum_m \sum_{i=1}^{n} \log \left(\det(2\pi\mathbf{\Sigma})^{-1/2} I[y_i = C_m] + \log \left(\exp(-\frac{1}{2}(x_i - \mu_m)^T \mathbf{\Sigma^{-1}}(x_i - \mu_m)) \right) I[y_i = C_m] + \sum_{i=1}^{n} \log(\pi_m) \right.$$

$$\frac{\partial \log p(D;\pi)}{\partial \mu_k} = \frac{1}{2} \sum_{y_i = C_k} \left(\mathbf{\Sigma^{-1}\Sigma^{-T}} \right) (x_i - \mu_k)$$

Note that the above was obtained by (1) ignoring the prior and the determinant term since they don't depend on μ_k, (2) using formula (81) from the Matrix Cookbook to take the derivative with respect to the vector $(x_i - \mu_k)$, and (3) chain-ruling the -1 to differentiate wrt μ_k.

4. Next, to derive the MLE, we can simply set the gradient expression found above equal to zero (first order condition) and solve for $\hat{\mu}_k$ (noting that $\mathbf{\Sigma}$ is a nonzero constant matrix):

$$\frac{1}{2} \sum_{y_i = C_k} \left(\mathbf{\Sigma^{-1}\Sigma^{-T}} \right) (x_i - \mu_k) = 0$$

$$\sum_{y_i = C_k} (x_i - \mu_k) = 0$$

Using notation as above:

$$\sum_{y_i = C_k} x_i - N_k \mu_k = 0$$

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{y_i = C_k} x_i = \frac{\sum_{i=1}^{n} x_i I[y_i = C_k]}{\sum_{i=1}^{n} I[y_i = C_k]}$$

This makes intuitive sense: the MLE for the mean in category k is just the sample mean of all observations with labels in category k, just as one woudl expect.

5. For this part, we can again write down the log-likelihood in a suitable form and differentiate with respect to $\mathbf{\Sigma}$, ignoring constant terms.

$$\log p(D;\pi) = \sum_m \sum_{i=1}^{n} \log \left(\det(2\pi\mathbf{\Sigma})^{-1/2} I[y_i = C_m] + \log \left(\exp(-\frac{1}{2}(x_i - \mu_m)^T \mathbf{\Sigma^{-1}}(x_i - \mu_m)) \right) I[y_i = C_m] + \sum_{i=1}^{n} \log(\pi_m) \right.$$

$$\frac{\partial \log p(D;\pi)}{\partial \mathbf{\Sigma}} = \frac{\partial}{\partial \mathbf{\Sigma}} \sum_m \sum_{i=1}^{n} \log \left(\det(2\pi\mathbf{\Sigma})^{-1/2} I[y_i = C_m] + \frac{\partial}{\partial \mathbf{\Sigma}} \sum_m \sum_{i=1}^{n} \log \left(\exp(-\frac{1}{2}(x_i - \mu_m)^T \mathbf{\Sigma^{-1}}(x_i - \mu_m)) \right) I[y_i = C \right.$$

$$= \frac{\partial}{\partial \mathbf{\Sigma}} \sum_m \sum_{i=1}^n \log\left(\det(2\pi\mathbf{\Sigma})^{-1/2} I[y_i = C_m] - \frac{1}{2}\frac{\partial}{\partial \mathbf{\Sigma}} \sum_{y_i = C_k} (x_i - \mu_k)^T \mathbf{\Sigma}^{-1}(x_i - \mu_k) I[y_i = C_m]\right.$$

Now applying the two cookbook formulas above to differentiate these two terms:

$$= -\frac{1}{2}\sum_m N_m (2\pi\mathbf{\Sigma})^{-T} + \frac{1}{2}\sum_m \sum_{i=1}^n \mathbf{\Sigma}^{-T}(x_i - \mu_k)(x_i - \mu_k)^T \mathbf{\Sigma}^{-T} I[y_i = C_k]$$

$$= -\frac{1}{2} n \mathbf{\Sigma}^{-T} + \frac{1}{2}\Sigma^{-T}\left(\sum_m \sum_{i=1}^n I[y_i = C_k](x_i - \mu_k)(x_i - \mu_k)^T\right)\Sigma^{-T}$$

6. Now all that's left is setting the expression above equal to 0 and solving for the MLE $\hat{\mathbf{\Sigma}}$:

$$0 = -\frac{1}{2} n \mathbf{\Sigma}^{-T} + \frac{1}{2}\Sigma^{-T}\left(\sum_m \sum_{i=1}^n I[y_i = C_k](x_i - \mu_k)(x_i - \mu_k)^T\right)\Sigma^{-T}$$

$$n\mathbf{\Sigma}^{-T} = \Sigma^{-T}\left(\sum_m \sum_{i=1}^n I[y_i = C_k](x_i - \mu_k)(x_i - \mu_k)^T\right)\Sigma^{-T}$$

Multiplying by $\mathbf{\Sigma}^T$ from the left and using the properties of inverse matrices:

$$n = \left(\sum_m \sum_{i=1}^n I[y_i = C_k](x_i - \mu_k)(x_i - \mu_k)^T\right)\Sigma^{-T}$$

And finally, multiplying by $\mathbf{\Sigma}^T$ from the right and also using the fact that $\mathbf{\Sigma}$ is symmetric and so $\mathbf{\Sigma} = \mathbf{\Sigma}^T$:

$$\hat{\mathbf{\Sigma}} = \hat{\mathbf{\Sigma}}^T = \frac{1}{n}\sum_m \sum_{i=1}^n I[y_i = C_k](x_i - \mu_k)(x_i - \mu_k)^T$$

This makes intuitive sense, since it's just an average of covariance matrices for the classes k weighted by the number of observations in each class.

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?

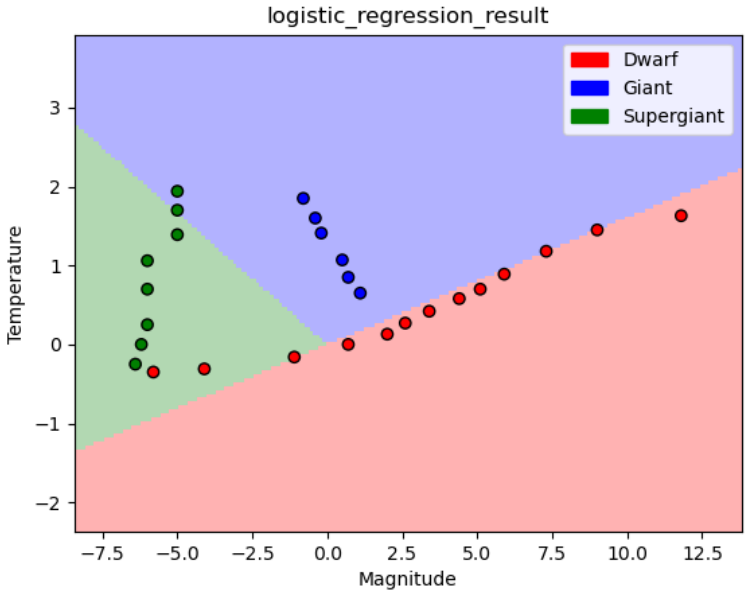
Problem 3 (cont.)

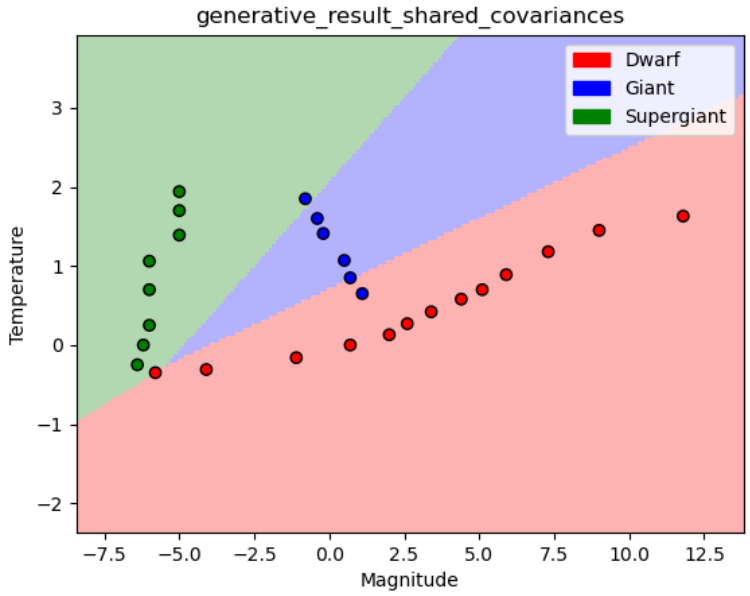
Implementation notes: Run the controller file, `T2_P3.py`, to test your code. Write the actual implementations in the `GaussianGenerativeModel`, `LogisticRegression`, and `KNNModel` classes, which are defined in the three `T2_P3_ModelName.py` files. These classes follow the same interface pattern as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

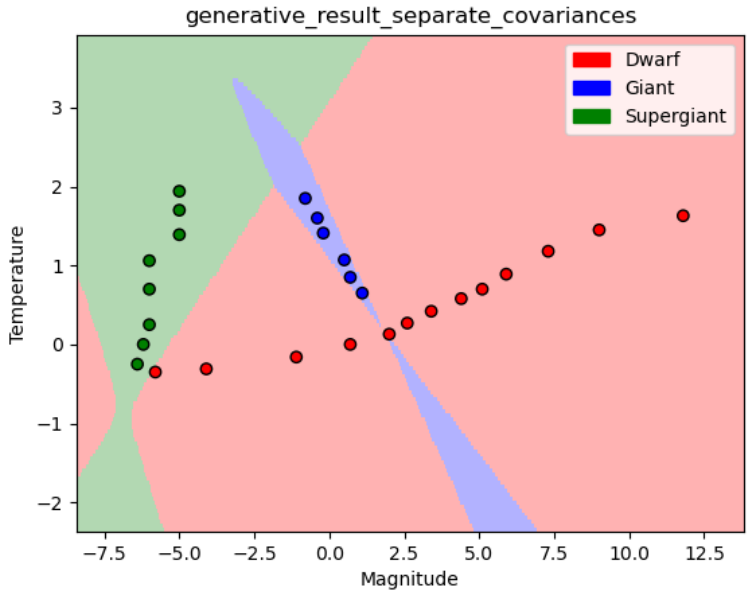
Solution

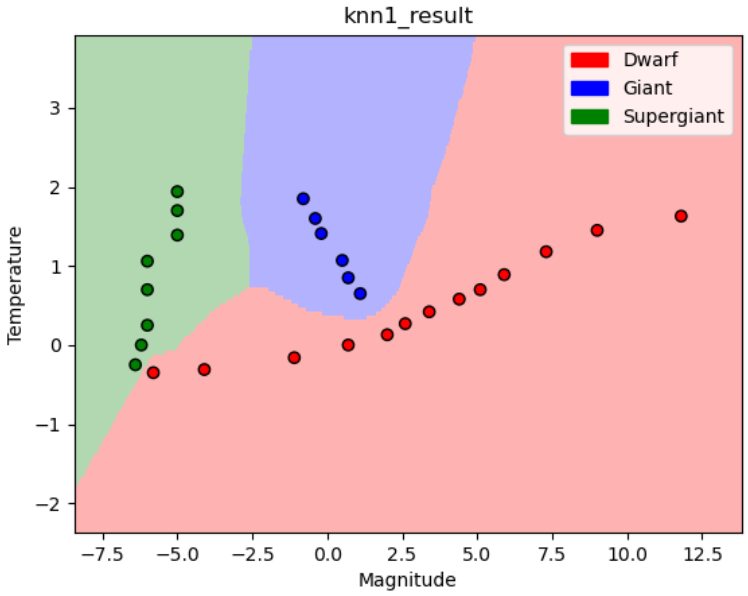
Answer:

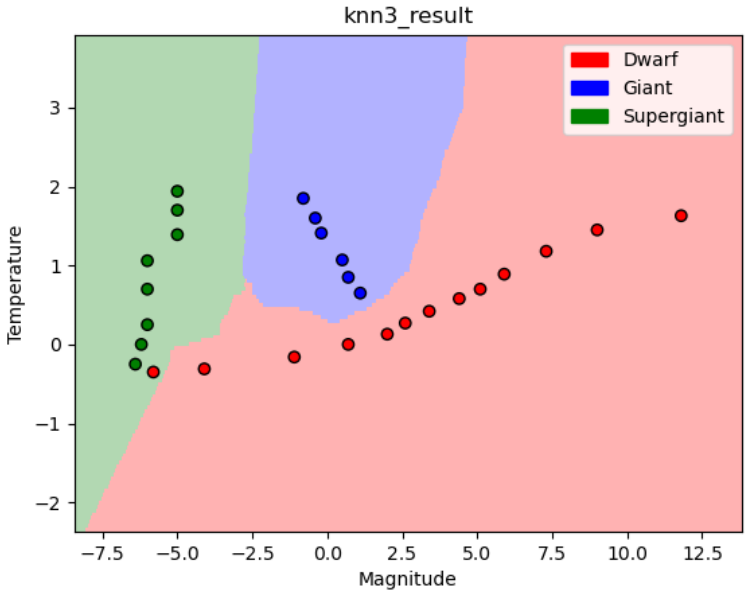
1. See the plots below:

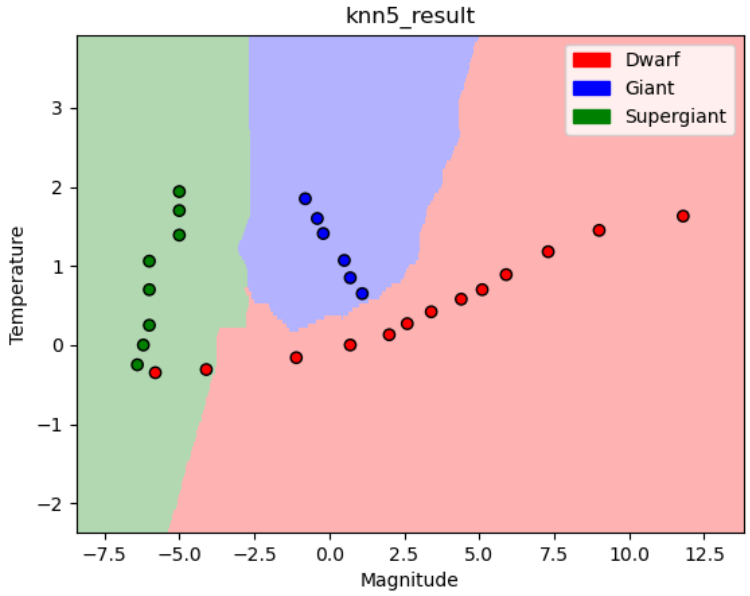



generative_result_shared_covariances
Dwarf
Giant
Supergiant
Temperature
Magnitude
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5


generative_result_separate_covariances
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


knn1_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


knn3_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



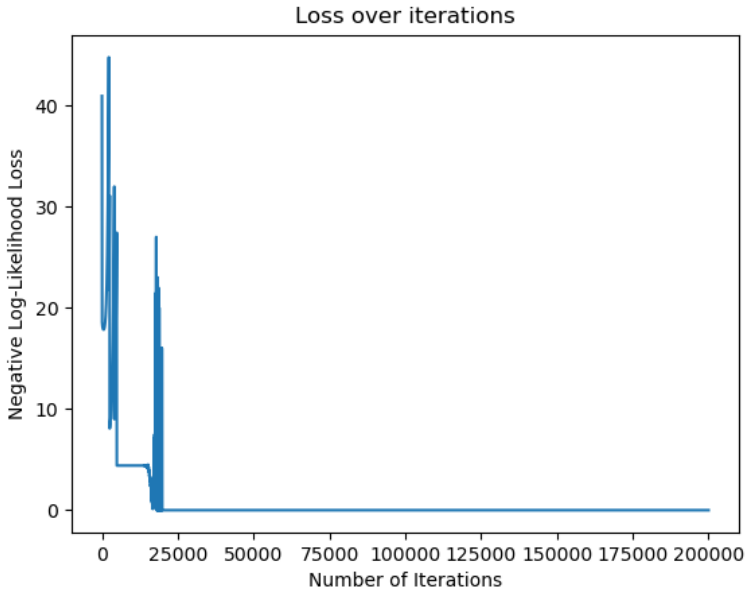
While there are exceptions, most models are able to separate the training set into the 3 categories or stars successfully. Notable exceptions include the logistic regression which includes some "red" points in "green" territory and the generative model with shared covariances which misclassifies some "blue" points.

Notice that both logistic regression and the generative model with shared covariances produce straight-line decision boundaries. Including separate covariances in the generative model leads to parabola-shaped boundaries, as predicted. Finally, the decision boundaries for kNN are not a simple function of the x, y variables.

2. Here are the results for various hyperparameters:

	$\lambda = 0.05$	$\lambda = 0.01$	$\lambda = 0.001$
$\eta = 0.05$	converges (slow convergence)	converges (fast convergence)	converges (slow convergence)
$\eta = 0.01$	converges (slow convergence)	converges (slow convergence)	converges (slow convergence)
$\eta = 0.001$	converges (fast convergence)	converges (fast convergence)	converges (slow convergence)

I chose $\eta = 0.001$ and $\lambda = 0.001$ for the final model; the learning rate is low enough to find an optimum and the regularization term is not too large (else the model might steer far away from an optimum). Here's a plot:



3. The negative log-likelihood loss for the shared covariance model is 89.95505685356703 and for the separate covariance model, it is 39.15901944439807. This makes sense: since the separate-covariance model has more degrees of freedom, it will fit the data better than a model which coerces the covariance matrix to be the same for all classes.

   However, overfitting may be an issue here, especially for small datasets like this one where each category may only have a handful of observations. I'd worry about the generalizability of the separate covariance model beyond the given data set.

4. Here are the classifications:

Classifier	Classification
Logistic regression	1
Generative (shared)	1
Generative (separate)	0
kNN (1)	0
kNN (3)	0
kNN (5)	0

   The kNN models do not give any indication of uncertainty since they simply classify based on the nearest points without fitting a probabilistic model. Logistic regression and the generative model both model a distribution for the data, so we can look at the relative magnitudes of their predictions for all 3 classes to see how "sure" each model is about its prediction:

Classifier	Probabilities
Logistic regression	[0. 1. 0.]
Generative (shared)	[1.57694899e-03 5.01896193e-03 5.01861864e-05]
Generative (separate)	[9.0021604e-14 0.0000000e+00 0.0000000e+00]

   This shows that the logistic regression and the generative model with separate covariances are completely sure about their predictions (even though these are different!) but the generative model with shared covariance places significant probability on both 0 and 1 classes.

Name

Matej Cerman

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes? I used the Matrix Cookbook and also this Wikipedia page for the multivariate Gaussian PDF.

I worked with Arpit Bhatte.

Calibration

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

20+ hours (way, way too long because singificant parts of the pset weren't clear so I had to wade through Ed clarifications).