CS181-S21

Assignment #2

Due: 7:59pm EST, Feb 19th, 2021

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 2/9 lecture. The rest of the material will be covered by the end of the Thursday 2/11 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 $y \sim \text{Bern}(f(x)), f(x) = \sigma(\sin x)$, where $\sigma(z)$ is the sigmoid function $\sigma(z) = (1 + \exp[-z])^{-1}$, $x \in \mathbb{R}$, 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 $\phi_1(x) = [1, x]$, $\phi_2(x) = [1, x, x^2, x^3]$, $\phi_3(x) = [1, x, x^2, x^3, x^4, x^5]$. For each of these bases, generate 10 datasets of size $N = 10$ 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. Note that the classes are represented with 0's and 1's. This means you will be running gradient descent 10 times for each basis, once for each dataset.

   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.

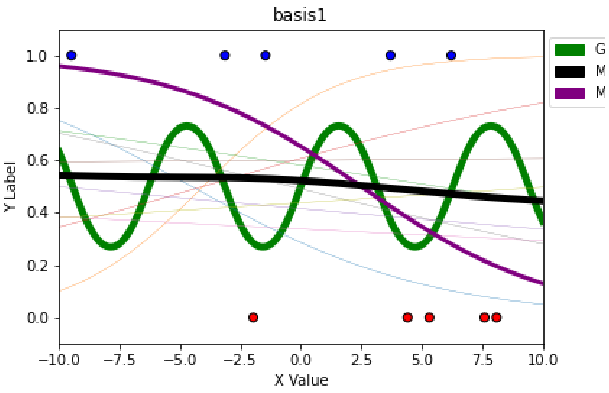
2. Create three plots, one for each basis. Starter code is available which you may modify. By default, each plot displays three types of functions: 1) the true data-generating distribution, 2) all 10 of the prediction functions learned from each randomly drawn dataset, and 3) the mean of the 10 prediction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

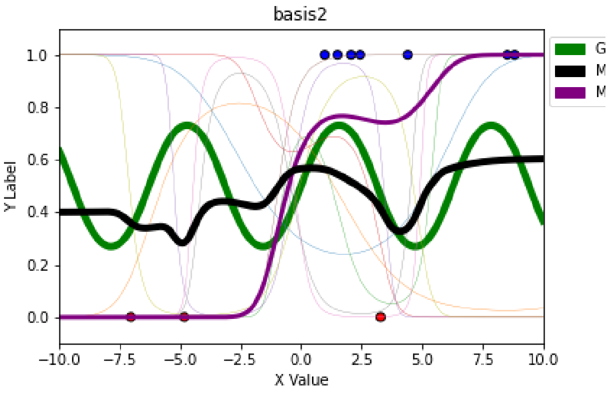
3. Explain what you see in terms of the bias-variance trade-off. How do the fits of the individual and mean prediction functions change? Keeping in mind that none of the model classes match the true generating process exactly, discuss the extent to which each of the bases approximates the true process.

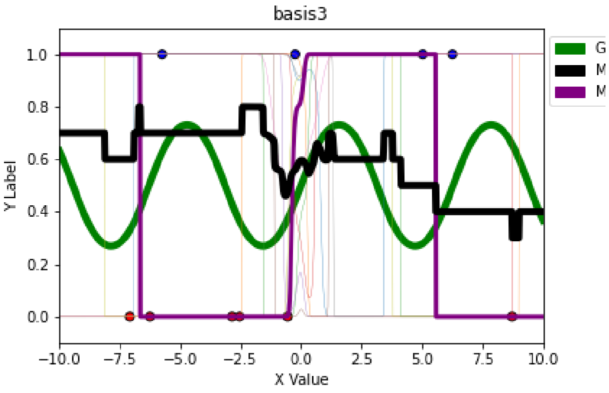
4. If we were to increase the size of each dataset drawn from $N = 10$ to a larger number, how would the variance change? Why might this be the case?

Solution

1. The code has been attached in the supplementary materials.
2. The three figures are shown below.







3. From basis 1 to basis 3, as the basis become increasingly complex, the bias decrease, and the variance between models increase. The fit of the individual dataset becomes "better" from basis 1 to basis 3. For the mean prediction functions, for basis 1, it is almost a straight line, for basis2, it has more turning points, for basis3, it has many turning points, but none of the model classes catch the true generating process quite good.

Ideally, we would expect a sin function could be perfectly caught by a polynomial of infinity order. I would expect basis 3 has the best potential to catch the ground truth. However, from the results, I would say basis 3 is not significantly better than basis 2, and basis 1 even worse. Obviously basis 1 is under-fitted, which means it is not able to catch the character of the generate process. Basis 3 is over-fitted, which means the model is too closely fitted to a limited number of the data, making the variance between models quite large.

4. I increase the number of N and see that the variance between smaller. It seems when increasing N, the effect is similar to averaging multiple regression models in the same class with smaller N. This "average" effect makes the variance between models smaller for larger N.

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 negative 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 negative 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 negative 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

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

2. Lagrangian function is
$$L = -\ln p + \lambda(\sum k\pi_k - 1).$$
Taking partial derivative we have
$$\frac{\partial L}{\partial \hat{\pi}} = \frac{\sum_k \mathbb{I}(\mathbf{y}_i = C_k)}{\lambda}$$
And also
$$\sum_k \pi_k - 1 = 0$$
Then we can get
$$\hat{\pi_k} = \frac{\sum_k \mathbb{I}(\mathbf{y}_i = C_k)}{n}$$
This is intuitive because we see the numerator is the total number of y_i in the class k, and the denominator is the total number of the data. Intuitively we will estimate the probability of $y_i = C_k$ with that.

3. Put the Gaussian distribution into the equation we derived in question 2.1, I get
$$-\ln p = -\sum_{i=1}^{n}\sum_{k=1}^{K} \ln(\pi_k \mathcal{N}(x|\mu_k, \Sigma))\mathbb{I}(y_i = C_k)$$
And after some simplification I get
$$-\ln p = \frac{n}{2}\log(|\Sigma|(2\pi)^K) - \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{I}(y_i = C_k)[-\frac{1}{2}(x_i - \mu_k)^T\Sigma^{-1}(x_i - \mu_k) + \ln(\pi_k)]$$
$$\frac{\partial(-\ln p)}{\partial \mu_k} = -\sum_{i=1}^{n}[\mathbb{I}(y_i = C_k)\Sigma^{-1}(x_i - \mu_k)]$$
Here Σ,x_i,μ_k are vectors.

4. Now that we know the gradient in the last question, we can directly let it equals to 0 to get the minimum of the negative log likelihood.

   We then get
$$\hat{\mu}_k = \frac{\sum_{i=1}^{n} \mathbb{I}(y_i = C_k)x_n}{\sum_{i=1}^{n} \mathbb{I}(y_i = C_k)} = \bar{x}_k$$
We see the answer equals to the sample mean of x in class k, which is intuitive, because if we use classical probability model to estimate it we will get the same answer. It is intuitive.

5. Using the two equations from the cookbook formulas, I have
$$\frac{\partial(-\ln p)}{\partial \Sigma} = \frac{n}{2}\Sigma^{-T} - \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{I}(y_i = C_k)[\frac{1}{2}\Sigma^{-T}(x_i - \mu_k)(x_i - \mu_k)^T\Sigma^{-T}]$$

6. Let the gradient in the last question equals to 0, we have

$$\hat{\Sigma} = \frac{\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{I}(y_i = C_k)(x_i - \mu_k)(x_i - \mu_k)^T}{n}$$

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.

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.

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.

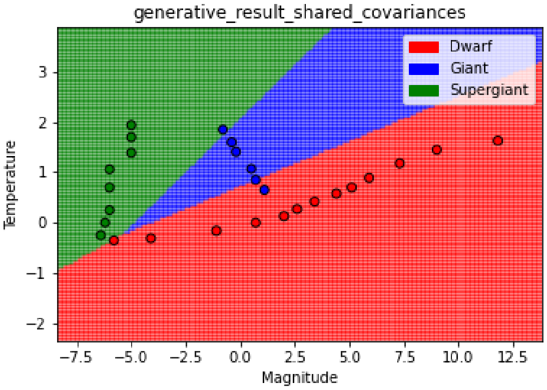
4. Consider a star with Magnitude 6 and Temperature 2. To what class does each classifier assign this star? Do the classifiers give any indication as to whether or not you should trust them?

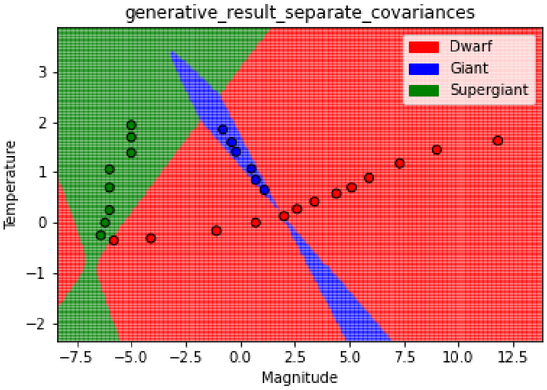
Problem 3 (cont.)

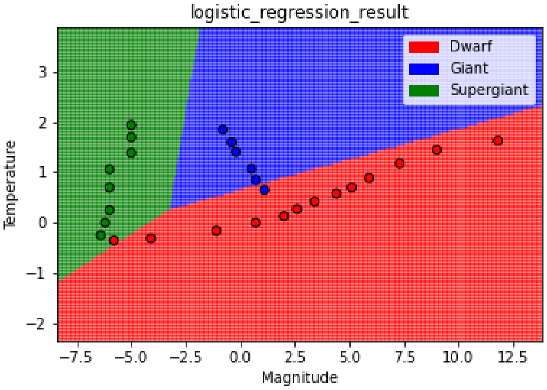
Implementation notes: Run the controller file, `T2_P3.py`, to test your code. Write the actual implementations in the `GaussianGenerativeModel`, `LogisticRegression`, and `KNNModel` classes, which are defined in the three `T2_P3_ModelName.py` files. These classes follow the same interface pattern as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

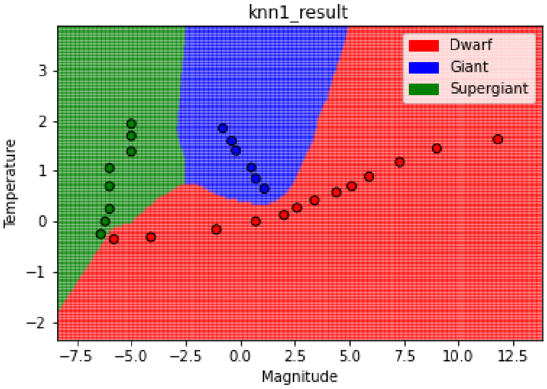
Solution

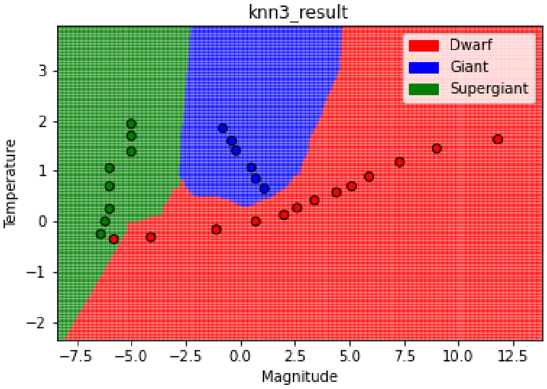
1. The plots are shown below. We see that for generative_shared_covariance, logistic_regression and the knn methods, the space is split into 3 regions. But for the generative_separate_covariance, the space is split into 6 different regions, due to the different covariances used in each cluster. The boundaries in the two generative methods and the logisitic regression method are straight, but the boundaries in knn methods are curl lines, that is because knn classification specifically depend on the distance to the neighboring points. The three knn methods are similar, but due to the difference in the k parameter, there are some small difference among them. When k is larger, the boundary is collectively decided by a few number of points, which leads to change to the classification of the training points near the boundary. More details will be further discussed below on why the results for each classification method are different.

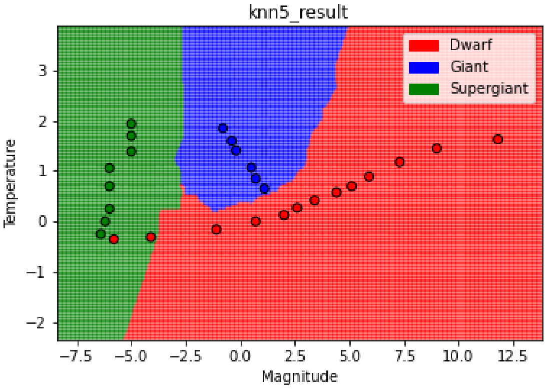





logistic_regression_result


knn1_result


knn3_result

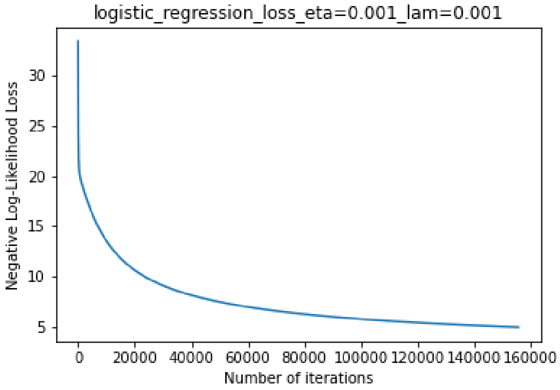


2. I finally choose $\eta = 0.001$ and $\lambda = 0.001$. For η it is because the ability to converge is good, and also because the learning rate is still reasonable. For λ it is because this choice gives the maximum likelihood, and also because the classification boundaries looks most reasonable when looking at the data. The plot is shown below.

   Here is a detailed discussion on hyperparameters.For

   $$\eta$$

   , when η is too large (say 0.5), the ability to converge is bad, and the final likelihood result will jump between a range of value. When η is smaller, the ability to converge is no longer a problem. The smaller η is, the smaller the rate of convergence will be. However even with $\eta = 0.001$, the number of iteration is still reasonable as shown below (I set the threshold to stop as the change of negative log likelihood is smaller than 1e-5). For λ, the property of converge does not seem to be largely influenced by λ, but the smaller λ is, the final negative log likelihood will be smaller. Based on our knowledge from the class, larger λ means larger regularization, which leads to smaller variance between different models. In turn, because the magnitude of the parameter w is limited, the likelihood will naturally be smaller (negative log likelihood is larger). Here I choose to use a small λ because base on the pattern of the data, I think the small λ is most physically reasonable. The small λ does not seem to lead to any overfit.



3. Shared covariance: 116.39. Separate covariance: 63.97. The separate one have a lower loss. That's intuitive because we have more parameters for the separate covariance one. More parameter means more freedom to make the fit result close to the training data. We can also see from the training boundary plot, that all the training data points are put into the "right" class for the separate covariance one, but for the shared covariance one, some blue training points are put into wrong classes.

4. Classification result: Separate Covariance Gaussian Model: 0(red,Dwarf). Shared Covariance Gaussian Model:1(blue, Giant). Logistic regression: 1(blue,Giant). All three Knn models with k=1,3,5: 0 (red,Dwarf).

   I think I would rather believe it should be classified as class 0(red) when looking at the boundaries of each model. We see that the blue points (Giants) are almost on a straight line with negative slope, and the red points (Dwarf) are almost on a straight line with positive slope. What's more the Magnitude of the Giants are limited, according to the training data. So I would believe that the boundary between Giants (blud) and Dwarfs(red) should limit the blue region to a smaller magnitude, like smaller than 3. But when looking at the boudaries in the Logistic regression model and the Shared Covariance Gaussian Model, the boundary does not strictly constrain blue region's Magnitude ,which does not seems physically reasonable.

Name

Collaborators and Resources

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

Calibration

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