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:

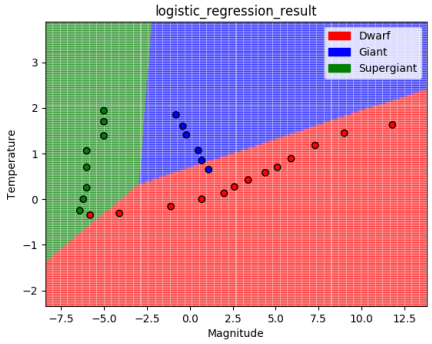
1. Plot the decision boundaries generated by each classifier for the dataset. Include them in your PDF. Identify the similarities and differences among the classifiers. What explains the differences?
2. For logistic regression only, make a plot with "Number of Iterations" on the x-axis and "Negative 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?

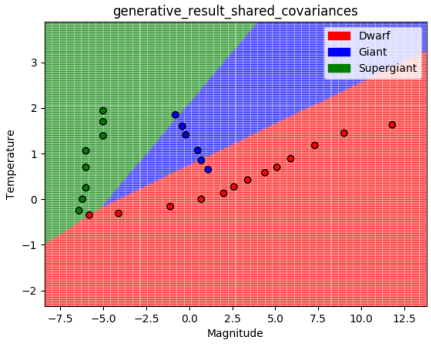
Problem 3 (cont.)

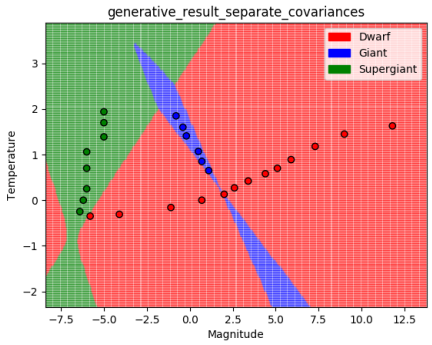
Implementation notes: Run the controller file, `T2_P3.py`, to test your code. Write the actual implementations in the `GaussianGenerativeModel`, `LogisticRegression`, and `KNNModel` classes, which are defined in the three `T2_P3_ModelName.py` files. These classes follow the same interface pattern as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

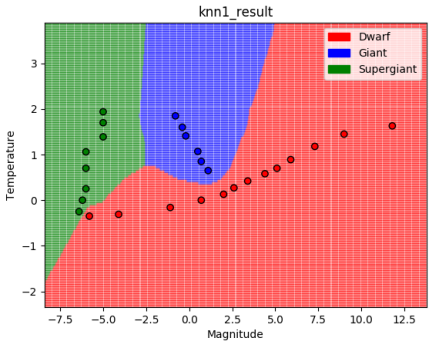
Solution

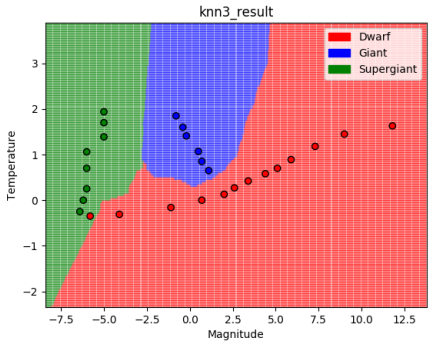
1. The decision boundaries for each of the classifiers are shown below:

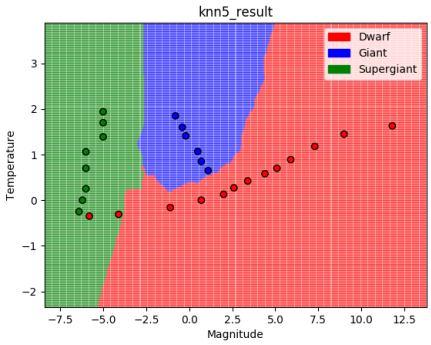










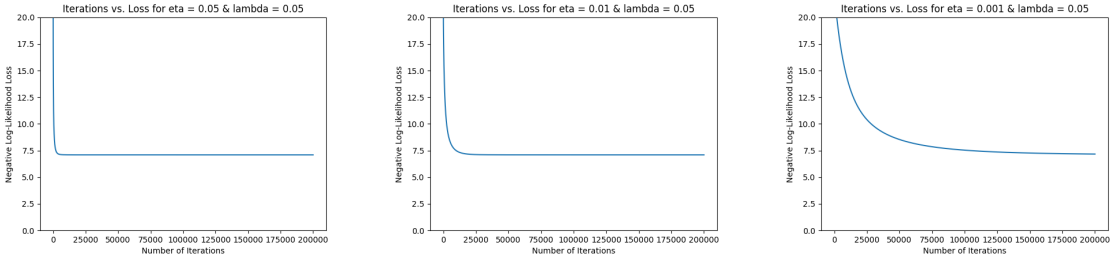


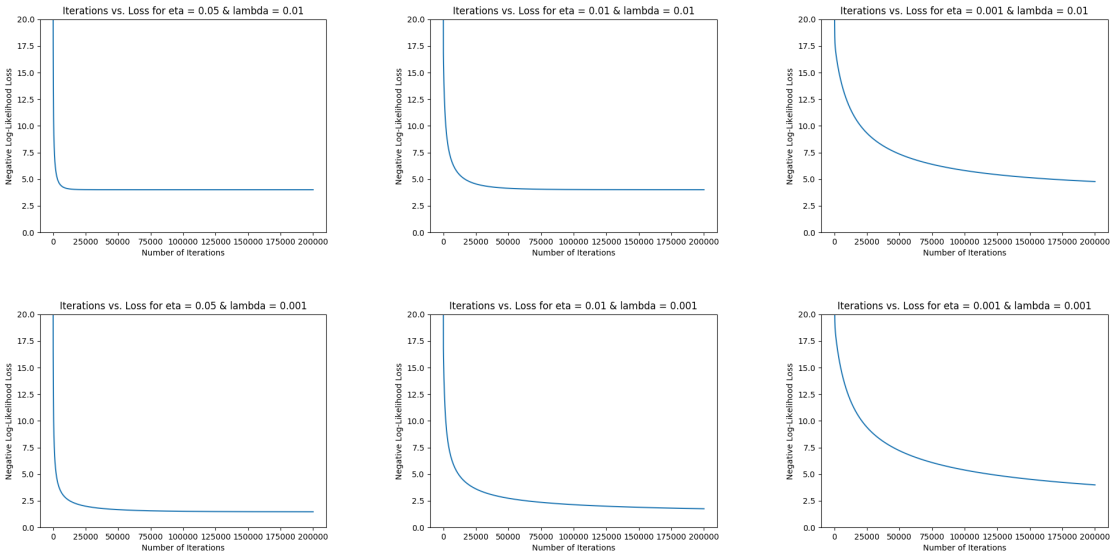
The plot for the Logistic Regression and for the Generative Gaussian model (with shared covariances) look extremely similar and actually misclassify the exact same three data points. This is because both of these models can only yield linear decision boundaries, and observing our plot of points, there is no way to perfectly classify all of our data points using only linear boundaries. One difference between the two plots is that, although the regions classified as Dwarfs are roughly the same between the two models, the Generative Gaussian model predicts a significantly larger region to be Supergiant than the Logistic Regression does. This is caused by the differences in how they optimize their parameters. Logisitic Regression uses gradient descent, and therefore will place the decision boundaries between Supergiants and Giants at the midpoints between the data points since that will minimize the loss. In comparison, in the Generative Gaussian model, because we have modeled a distribution for our class labels, our decision boundaries are not at the midpoints between the data points.

Now comparing the Generative Gaussian models with shared and separate covariances, we find that the model with separate covariances is able to perfectly classify all of our data points, and it is able to do so because it can create non-linear decision boundaries, unlike the model with shared covariances. Intuitively, this is because the model with separate covariances allows us to model differently shaped distributions for each of the classes. For any point, the predicted class of that point is the class which has the highest pdf at that point. The decision boundaries in the model with separate covariances is formed where the pdfs of two different classes tie in probabilities, but because the distributions of each of the classes are of different shapes (because each class has different covariances) then that allows the model to yield nonlinear decision boundaries.

The parametric models (Logistic Regression and both Generative Gaussian models) had smooth decision boundaries, whereas the non-parametric KNN models all have non-smooth decision boundaries that appear jagged. This is because as a nonparametric model, KNN will create the decision boundaries at the midpoints between data points of different classes (where the midpoint is defined by whatever distance function we use in KNN). In comparing the three KNN models between each other, we see that KNN 1 perfectly classifies all of the data points, which makes intuitive sense because KNN 1 predicts solely from the nearest data point. In contrast, KNN 3 misclassifies one point and KNN 5 misclassifies two points. Moreover, the decision boundaries appear less smooth for KNN 3 than KNN 1, and the decision boundary for KNN 5 appear the least smooth. This is because KNN models that use a higher number of neighbors will have less stable decision boundaries, as the furthest neighbor used to classify will change often as we move throughout our region of data points, therefore yielding less smooth decision boundaries.

2. The nine plots for our different combinations of η and λ values are shown below:





Observing the plot, we find that our choice of λ affects what values of loss that gradient descent will converge to, regardless of what the value of η is. Smaller λ allows us to converge to a smaller log-likelihood loss. Specifically, for $\lambda = 0.05$, our losses converge to approximately 7.5, for $\lambda = 0.01$, our losses converge to approximately 5, and for $\lambda = 0.001$, our losses converge to approximately 2.5.

We find that η affects the rate of convergence (shown by the shape of the plots above). We find that smaller η means that gradient descent will take longer to converge. In particular, $\eta = 0.05$ converges quickly within approximately 25,000 iterations, $\eta = 0.01$ converges within approximately 100,000 iterations, and $\eta = 0.001$ converges within approximately 200,000 iterations.

Observing the plots, we would likely choose $\eta = 0.01$ and $\lambda = 0.001$ as our hyperparameters (the plot is shown above) because the choice of $\eta = 0.01$ allows gradient descent to have smoother convergence than the choice of $\eta = 0.05$, and converge in a reasonable number of iterations unlike the choice of $\eta = 0.001$. For our choice of λ, we choose $\lambda = 0.001$ because it allows gradient descent to converge to a lower loss of approximately 2.5 than compared to the choice of $\eta = 0.05$ and $\eta = 0.01$.

3. We find that the Generative Gaussian model with shared covariances had a negative log-likelihood loss of 116.57191681902043, while the Generative Gaussian model with separate covariances had a negative log-likelihood loss of 64.17309552244006. We find that the model with separate covariances has a lower negative log-likelihood loss than the model with shared covariances.

   This makes intuitive sense, because the model with separate covariances is more generalizable than the model with shared covariances. In particular, the model with separate covariances is more generalizable (and therefore provides a better fit to our data and thus a lower negative log-likelihood loss) because it can fit nonlinear decision boundaries (how the model does so is discussed above in 3.1). Just by observing our plots in 3.1, we can immediately see that the model with shared covariances misclassifies three points, while the model with separate covariances perfectly classifies all of our data points, so just by inspecting the plots we can see that we expect the separate covariance model to have lower negative log-likelihood loss.

4. We find that the three KNN models and the Generative Gaussian model with separate covariances classifies this star as a Dwarf (class 0), while the Logistic Regression model and the Generative Gaussian model with shared covariances classifies this star as a Giant (class 1).

   Observing the decision regions, we find that we may not trust the Generative Gaussian model with separate covariances because the region classified as Giant stars (the blue region) is extremely small,

and it seems the model may be suffering from overfitting.

Moreover, as we discussed above in 3.1, the KNN 5 model has non-smooth decision boundaries, which is because using 5 neighbors is not very stable (the explanation is discussed in 3.1), so that would be a reason not to trust the KNN 5 model.

Examining the Logistic Regression model, we would also find it less trustworthy than the Generative Gaussian model with shared covariances. Although both models utilize linear decision boundaries, and therefore suffer the limitations of linear boundaries being unable to perfectly fit the data, we would prefer the Generative Gaussian model because we fit the data using maximum likelihood estimation rather than gradient descent like the Logistic Regression model. As noted in 3.1, we can see this limitation of the Logistic Regression model by examining the difference in the decision boundary for Giant and Supergiant (the green and blue decision boundary). As described previously, the Logistic Regression merely fits this boundary at approximately the midpoint between the Supergiant and Giant points, whereas the Guassian Generative model chooses the decision boundaries that will maximize the joint distribution of data points.