Programming Project 1
Machine Learning, Spring 2019

Introduction

In this assignment you will use Naive Bayes to classify celestial objects as stars or quasars based on photometric data (colors of light) collected by the Sloan Digital Sky Survey. The classification will be based on four color features. For more information on how similar techniques and data have been used by astronomers see the attached research papers.

What to submit

Submit one text document (.doc or .pdf) with your answers to all parts. For full credit this must be well formatted, well organized, and easily understood. I will primarily grade your written report. It should be polished and well written and should thoroughly address all the questions posed in each part.

Also submit .py files that include your code (you may submit one file or one file for each part). For full credit your code must be well organized and any unclear parts must be commented. I will only be running your code if I have a question about your written report.

Loading the data

Use the following to load the data

```
data = np.load('sdss_data.npy').item()
X_train = data['X_train']
X_test = data['X_test']
y_train = data['y_train']
y_test = data['y_test']
feature_names = data['feature_names']
class_labels = data['class_labels']
```

Part 1: Exploring the data

Begin by exploring the training data (i.e. calculate statistics and create plots).

Describe the training data set and include relevant tables or figures to support your description. In particular, consider the assumptions that you make when you use Gaussian Naive Bayes and how well this data set does or does not adhere to them.

Do **not** include every plot you make or every value you calculate! Include only the most relevant and informative figures and statistics to support your discussion. If you produce many similar plots, include only a few that are representative of what you wish to discuss.

Part 2: Naive Bayes

Fit a Gaussian Naive Bayes classifier to the training set. Report performance on both the training set and the test set. Do **not** rely on total accuracy alone (% correct) as a measure of error, as this can be very misleading. Discuss your results. Is this a good classifier for this data set?

Part 3: Model selection

In this part you will select and test a more complex (and perhaps not naive) Bayesian classifier.

Begin by recreating the Gaussian Naive Bayes classifier. Use GaussianMixture models with a single component and diagonal covariance matrix. Recall that the following may be used to fit Gaussian distributions to the features and find the log likelihood of each instance in the training set (see iris.py from class):

```
c_0 = GaussianMixture(n_components=1, covariance_type='diag')
c_0.fit(X_train[y_train == 0])
loglikelihood = c_0.score_samples(X_test)
```

The output of the score_sample function above is the log likelihood of belonging to class 0 for each example in the test set. So the t[th] value in `loglikelihood` is

$$\log P(X = \boldsymbol{x^t} | C = 0)$$

where $\boldsymbol{x^t}$ is the t[th] instance in the test set (which consists of four features). The above log likelihood may be used in the following discriminant function:

$$g_o(\boldsymbol{x^t}) = \log P(X = \boldsymbol{x^t} | C = 0) + \log P(C = 0)$$

Recall: we classify $\boldsymbol{x^t}$ as belonging to class i if $g_i(\boldsymbol{x^t}) = \max_k g_k(\boldsymbol{x^t})$

The above classification rule is equivalent to selecting the class with the highest probability given the evidence, $P(C = i | X = \boldsymbol{x^t})$. Make sure you see why this is true.

By comparing the discriminant functions for each class, you should be able to classify the test set. Test your classifier by fitting with the training set and testing on the test set. Your results should match your results from part 2 to within .01 (they may not match exactly due to the particular numerical methods used in the implementations). The parameters of the distributions should also match. You do not need to include anything in your written report for this step, but do not continue with part 3 until you have replicated your results on the test data from part 2.

Next test several models of varying complexity and select the best one. You can vary the complexity of your model in two ways:

1. Change the form of the covariance matrix. Options are:
    - 'diag': This means all off diagonal terms in the covariance matrix are zero (this implies the features are independent) and each class has its own covariance matrix
    - 'tied': This means that there will be a single covariance matrix for all classes (in this case the two classes are star and quasar), but it may have non-zero off-diagonal terms
    - 'full': This means that each class will have its own covariance matrix and the matrices may have non-zero off-diagonal terms

    full > tied > diag with regards to number of parameters (model complexity)

2. Change the number of components (the number of Gaussian distributions being used to model the features). More components means more parameters and a more complex model.

To compare multiple models, you will need to divide the training data set into two parts:

- A training set for fitting the model
- A validation set for comparing models

For each model you consider, fit with the first subset of the training data, and measure how well the model generalizes with the second subset (the validation set).

Describe the model you select. How many components? What form for the covariance matrix? Why did you choose this one? Include data to support your choice.

Once you have decided on a model, use your model to classify the test set and report the results (remember, do not rely on percentage correct as your only measure of error!).