CS5350: Machine Learning *Due 25 Nov 2008*

Project 4: Gaussian Classifiers

Introduction

In this project, we will build a generative classifier based on Gaussians. That is, we assume the following *generative story* for how our data came to be:

1. For each class $k = 1 \dots K$:
   (a) Choose a *mean* for this class: $\boldsymbol{\mu}_k$
   (b) Choose a *covariance* for this class: $\boldsymbol{\sigma}_k^2$
2. For each example $n = 1 \dots N$:
   (a) Choose the class $y_n \sim \mathcal{M}ult(\text{classPi})$
   (b) Generate the data point $\boldsymbol{x}_n$ from $\mathcal{N}or(\boldsymbol{\mu}_{y_n}, \boldsymbol{\sigma}_{y_n}^2)$

Your job is to estimate all these parameters: classPi, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ from data.

There are two easy, sample data sets in `testdata1.mat` and `testdata2.mat` that we will walk through, and then we'll run "real" experiments on the mnist data.

Gaussian Classifier Implementation 60%

Your job is to implement the classifier stub in `Gauss.m`. There are essentially five ways that this can be run, depending on whether (a) all classes are forced to have the same covariance structure and (b) whether the covariance should be identity (i.e., not learned), diagonal, or full. The first parameter (after the data) to `Gauss` specifies whether the covariance should be identity (0), diagonal (1) or full (2). The second parameter (after the data) specifies whether the covariance should be shared (1) or not (0).

The first thing we can do is load data, plot it (it's two dimensional) and look at the *true model* that generated it:

```
>> load testdata1
>> DrawGaussian(X,Y,trueModel);
```

Here, we see three classes (black, blue and red). The data points themselves are small dots. The Gaussians that generated them are drawn with contours at one and two standard deviations in the appropriate color. You can do the same for `testdata2` if you feel like it.

First, implement the case where the `classPi` variables are estimated, as are the `mu` variables, but ignore the covariances. We can now run (this tells is to use identity covariance for all the classes):

```
>> model = Gauss('train', X, Y, 0, 1)
model =
    classPi: [0.3333 0.3333 0.3333]
         mu: {[-1.3739 5.2221]  [-1.6813 1.1001]  [-8.4521 -3.0465]}
        si2: {[2x2 double]  [2x2 double]  [2x2 double]}

>> model.si2{1}
ans =
```

```
     1     0
     0     1

>> DrawGaussian(X,Y,model)
```

All your variables should match the above. When plotted, you should see completely round Gaussians centered roughly correctly (see the big figure at the bottom of this section for how all such figures should look).

We can also try predicting with this model:

```
>> mean(Gauss('predict', model, X) == Y)
ans =
    0.9233
```

Next, we can implement shared diagonal and full covariance:

```
>> model = Gauss('train', X, Y, 1, 1)
model =
    classPi: [0.3333 0.3333 0.3333]
         mu: {[-1.3739 5.2221]  [-1.6813 1.1001]  [-8.4521 -3.0465]}
        si2: {[2x2 double]  [2x2 double]  [2x2 double]}

>> model.si2{1}
ans =
    1.6001         0
         0    2.4068

>> mean(Gauss('predict', model, X) == Y)
ans =
    0.9333
```

Feel free to draw this one too. You should see axis aligned ellipses for the Gaussians, and they should all be the same. Now try full covariance:

```
>> model = Gauss('train', X, Y, 2, 1)
model =
    classPi: [0.3333 0.3333 0.3333]
         mu: {[-1.3739 5.2221]  [-1.6813 1.1001]  [-8.4521 -3.0465]}
        si2: {[2x2 double]  [2x2 double]  [2x2 double]}

>> model.si2{1}
ans =
    1.6001   -0.5758
   -0.5758    2.4068



>> mean(Gauss('predict', model, X) == Y)
ans =
    0.9367
```

Now, if you draw it, all the covariances should be the same, but no longer axis-aligned.

Next, we try estimating separate covariances for each class, but keep them diagonal:

```
>> model = Gauss('train', X, Y, 1, 0)
model =
    classPi: [0.3333 0.3333 0.3333]
         mu: {[-1.3739 5.2221]  [-1.6813 1.1001]  [-8.4521 -3.0465]}
        si2: {[2x2 double]  [2x2 double]  [2x2 double]}

>> model.si2{1}
ans =
    2.2572         0
         0    1.7040

>> model.si2{2}
ans =
    0.1025         0
         0    4.6308

>> model.si2{3}
ans =
    2.4405         0
         0    0.8856

>> mean(Gauss('predict', model, X) == Y)
ans =
    0.9767
```

If you plot these, the per-class covariances should now be different, but still axis-aligned.

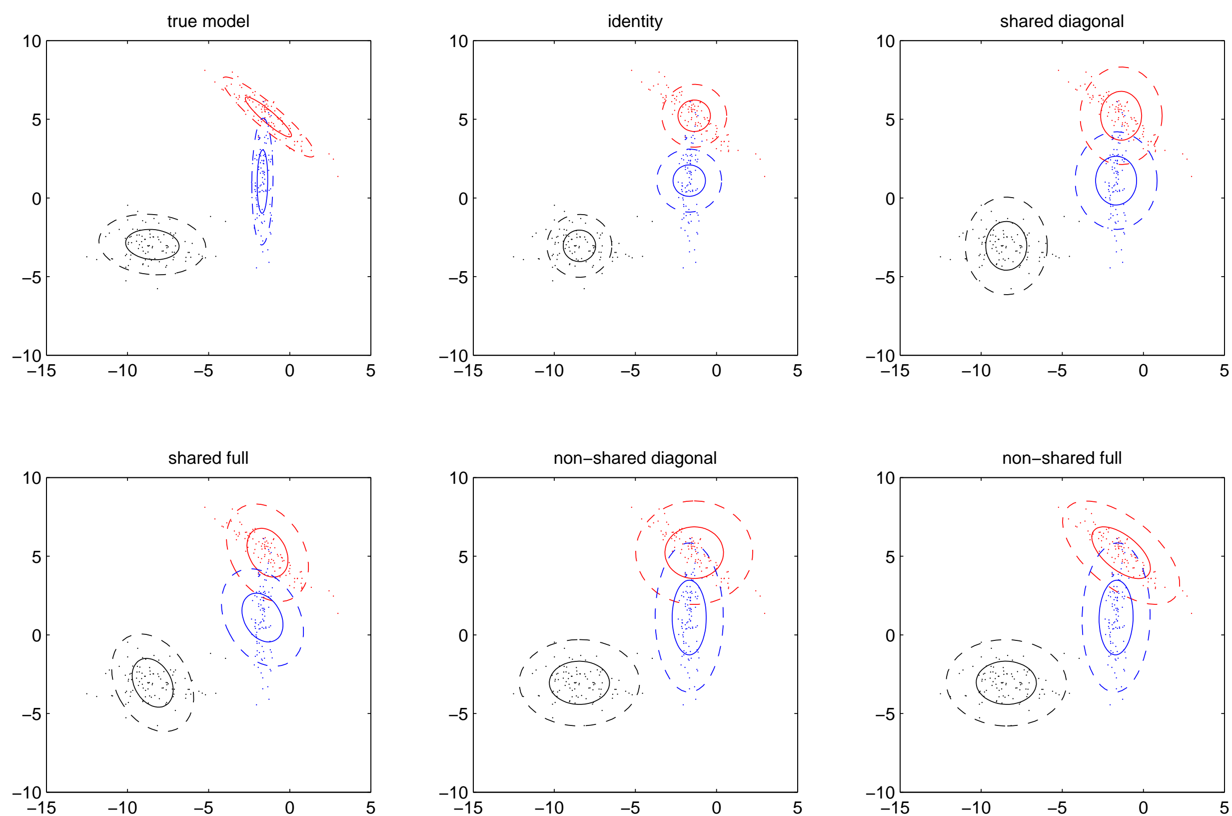
Finally, we try full covariance for each class:

```
>> model = Gauss('train', X, Y, 2, 0)
model =
    classPi: [0.3333 0.3333 0.3333]
         mu: {[-1.3739 5.2221]  [-1.6813 1.1001]  [-8.4521 -3.0465]}
        si2: {[2x2 double]  [2x2 double]  [2x2 double]}

>> model.si2{1}
ans =
    2.2572   -1.7801
   -1.7801    1.7040

>> model.si2{2}
ans =
    0.1025    0.0949
    0.0949    4.6308

>> model.si2{3}
ans =
    2.4405   -0.0422
   -0.0422    0.8856
```

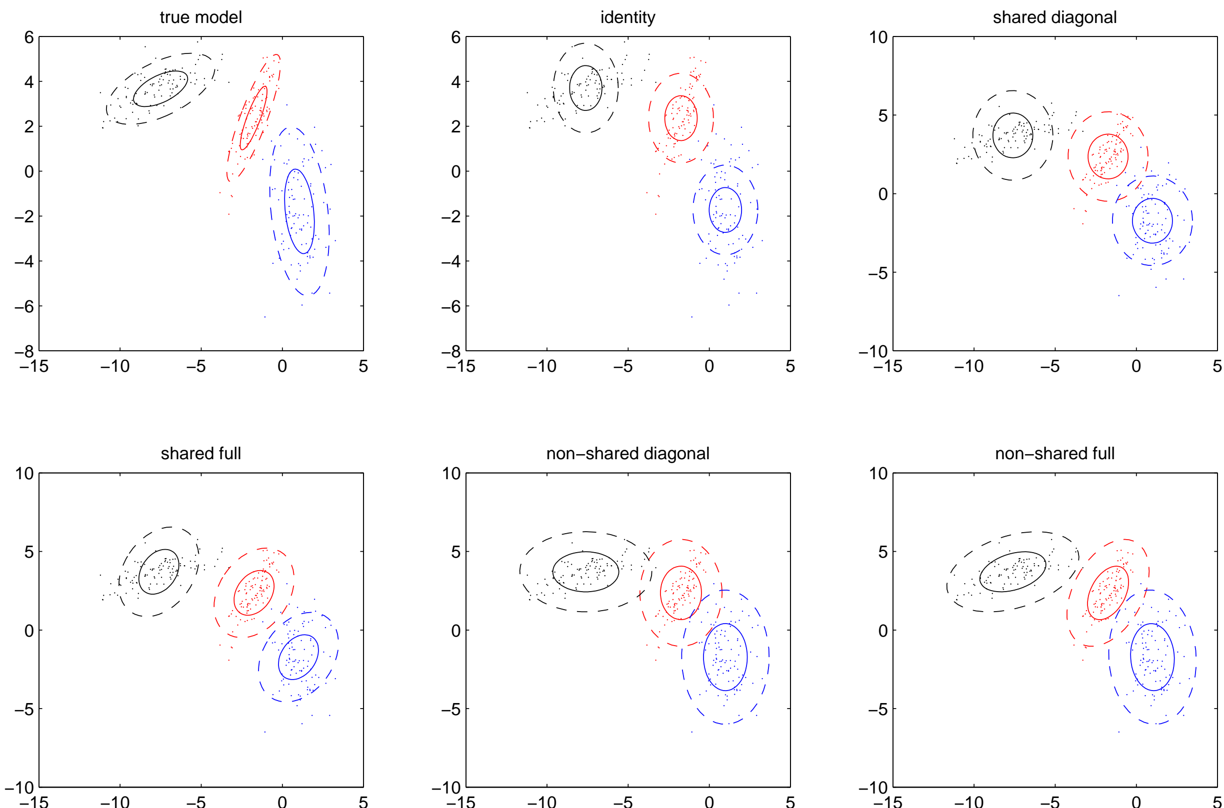
```
>> mean(Gauss('predict', model, X) == Y)
ans =
    0.9767
```

Drawing this should yeild different Gaussians for each class, not axis aligned.

The figures for all six runs for `testdata1` should look like:



For `testdata2` they should look like:



Analysis 40%

If you execute **run.m** it will run a whopping two experiments on mnist data:

1. In the first experiment, we run all five Gaussian models on the whole training set. The results you get should be:

```
0 1: identity, shared    ==> ... accuracy train = 0.792553, dev = 0.736842
1 1: diagonal, shared    ==> ... accuracy train = 0.797872, dev = 0.747368
2 1: full    , shared    ==> ... accuracy train = 0.968085, dev = 0.778947
1 0: diagonal, per-class ==> ... accuracy train = 0.771277, dev = 0.747368
2 0: full    , per-class ==> ... accuracy train = 1, dev = 0.873684
```

   This set of experiments takes 80 seconds for me.

   Please answer the following questions with respect to these results:

   (a) Do we see any evidence of overfitting in these examples?

   (b) What about underfitting?

   (c) Why do you suppose the diagonal/per-class underperforms everything else (including both full/per-class and diagonal-shared)?

   (d) Why do you suppose there's so little difference in performance between identity and diagonal/shared?

2. In the second experiment, we do the same thing but generate learning curves, for $N \in \{6, 12, 24, 47, 94, 188, 376\}$. This set of experiments take 6 minutes for me. It generates two figures. In Figure 1, we see a standard

learning curve for this data. It only plots development data accuracy (does not plot training data accuracy). Note that the x-axis is the log-number of training points, nt the number of training points.

In Figure 2, we see exactly the same data, but with accuracies divided by the performance of the identity covariance model. So, a value of 1.5 here means 50% better than the identity covariance; a value of 0.8 means 20% worse. The identity model is, of course, a horizontal line at 1.

Based on these figures, answer the following questions:

(a) Which curves show performance *worse* than the identity model? In what range of N do you see this? Why do you suppose this happens?

(b) You should see a spike around $\log N = 2.5$ (so $N = 12$). What do you think causes this spike?

(c) Are there fundamental difference between the ranking of the different models for different values of N? If so, why do you think this is; if not, why not?