Data Analysis and Model Classification
Guidesheet I-2

Fumiaki Iwane   Ping-Keng Jao   Bastien Orset   Julien Rechenmann
Ricardo Chavarriaga   José del R. Millán

With the help of this guidesheet, you will familiarize yourselves with Linear and Quadratic Discriminant Analysis classifiers (usually referred to as LDA and QDA, respectively) and the basics of estimating the performance of classifiers.

LDA/QDA classifiers

In this section we will train two classifiers on all the samples of the training data. The dimensionality of the original features may be too large compared to the number of available samples. Therefore we will choose only a subset of the original features.[1] In the provided ERP dataset, you can select a subset of features by taking every fifth feature, starting from the first feature until the last by typing `features = trainLabels(:,1:10:end);`

Linear discriminant classifiers   These classifiers assume an equal covariance matrix for both classes. The covariance matrix can be either *full*, in which case the covariance between features is taken into account, or *diagonal*, in which case only the variance of the features is taken into account. This method will yield a linear separating hyperplane.

Quadratic discriminant classifiers   These classifiers assume that classes can have different covariance matrices. As before, the covariance matrices can be either full, in which case covariance between features is taken into account, or diagonal, in which case only the variance of the features is taken into account. The separating hyperplane is quadratic due to the additional quadratic term (obviously).

If you attended the course, you should already understand how to train those different classifiers (i.e. how to find the methods' parameters w).

The statistics toolbox from MATLAB has a function to train an LDA/QDA classifier, you can type `classifier = fitcdiscr(features, labels, 'discrimtype', type);`, and classify a set of *test* features: `yhat = predict(classifier, features);` The outputs for the commands are, respectively, the trained classifier and the predicted classes for each sample in the training set (next section provides the concept of training set).

From the output `yhat` of the `predict function`,[2] we can compute the *classification error* and *class error* introduced last week.

Hands on

- Use your whole dataset to train a classifier using the `fitcdiscr` function and also use the same whole dataset (`features`) for prediction with the `predict()` function. Compute the classification

---

[1] Later in this course we will show some methods for selecting the best features

[2] You will find a lot of functions named predict in MATLAB help. In this special case, `which predict` will not lead you to the correct path. Locate the hyperlink of `predict` at the bottom of `fitcdiscr` in help browser (click "Reference page for fitcdiscr" if you type `help fitcdiscr`.)

error. Check the MATLAB help for `fitcdiscr()`. The default classifier is *linear*. Do the same using *diaglinear*, *diagquadratic* and *quadratic* classifiers. Based on the classification error, which classifier would you choose? The quadratic classifier didn't work because the matrix could not be inverted. Therefore we used the pseudo-quadratic classifier. The smallest classification error is fould with the pseudo-quadratic classifier: we would use that one.

'Prior' sets the prior probability for each class. Setting prior as 'uniform' means assuming that all the classes have the same prior probability.

- Check the help function of `fitcdiscr()` to find out how to specify prior probabilities. Then, based on the chosen classifiers, repeat again by specifying uniform prior when calling `fitcdiscr()`. This time, look at both classification error and class error and compare the cases with and without uniform prior. What do you observe and how do you explain the values based on the prior. Also, regarding the classification error and the class error, would you argue that one is more useful than the other. Make some assumption if necessary, and choose one for the later part of this guide sheet.

Training and testing error

Now we will assess the *generalization* of the classifier on an *unseen* dataset or *test set*. We will be differentiating the **training** error (predict on the set of samples that are used to train a classifier) from the **testing** error, obtained using the unseen (part of) dataset.

Hands on

- Separate your dataset into two sets of equal size, *set1* and *set2*. Train a diaglinear classifier using the data from *set1* and compute the error on the same data (i.e. training error). Compare it to the **testing** error on *set2*. What happened? 💬

- Repeat the same using *linear*, *diagquadratic* and *quadratic* classifiers. Would you still choose the same classifier as in the last section? Why the improvement on the **training** error does not improve the **testing** error? Why can you not use the *quadratic* classifier? Please take a look at your course and the equation of quadratic discriminant function. 💬

- These classifiers use different types of models. Roughly, the complexity can be measured from the number of parameters that are needed to be estimated. How many parameters does each classifier have? Compare these numbers to the number of training samples you are using to train the classifier. Do you think these classifiers are robust, why? 💬

- Compute the **training** and **testing** error again, but train on *set2* and notice the variability of the performance.

- Modify 'Prior' parameter (c.f. documentation `fitcdiscr()`), does it affect your performances? Why? 💬

- Test on Kaggle

Cross-validation for performance estimation

In the previous section, we saw that it is vital to estimate the classification accuracy on an *unseen* dataset. However, when separating the dataset into training and testing sets, you reduce the amount of data used for training! On the other hand, can you think about a disadvantage of having too small testing sets?

If we had datasets containing millions of samples, we could afford "losing" samples for the test set, however, with a small dataset like ours, we want to use every single possible bit of data available, while still testing on unseen data. One approach to tackle this is k-fold cross-validation. In this case, you split your data into k subsets (i.e., folds) of the data of equal size.[3] Then use $k-1$ subsets to train a classifier and the remaining subset to test it, say the first subset. Then you iterate and select the second subset for testing and use subsets $\{1, 3, 4, \ldots\}$ for training, etc...

In MATLAB, you can use the `cvpartition` command to create a 10-fold cross-validation partition: either `cp = cvpartition(N,'kfold',10);` or `cp = cvpartition(labels,'kfold',10);`, with `N` the total number of samples. Use `cp.training(i)` and `cp.test(i)` to get the training and testing set.

---

[3]Obviously, the size can be different by one between folds, depending on the total number of samples and k.

When $k = N$, you have only one sample in each test subset. This special case is referred to as "leave-one-out" cross-validation.

Hands on

- Compare the partition for `cvpartition(N,'kfold',10)` and `cvpartition(trainLabels,'kfold',10)`. How many samples do you have per class and per fold? Which one would you recommend to use?

- Implement a $k = 10$-fold cross-validation to estimate the cross-validation error of all classifiers tested before (if possible), use a `for` loop to compute the test error for each fold. Your cross-validation error is the mean of the test errors obtained over the 10 folds. Notice that you can also compute the standard deviation of the cross-validation error. This is an important measure of how stable your performance is.

- Repeat cross-validation (10-fold), changing the partition using `repartition(cp)` each time. Does the result change for both cross-validations? Why? Now you should know how to get a more stable estimation of accuracy. Is there any advantages of having classification performances that vary or on the contrary that are always the same?

- Do you need to choose a model from the cross validation? Why?