Discriminant Analysis

What Is Discriminant Analysis?

Discriminant analysis is a classification method. It assumes that different classes generate data based on different Gaussian distributions.

- To train (create) a classifier, the fitting function estimates the parameters of a Gaussian distribution for each class (see Creating a Classifier Using ClassificationDiscriminant.fit).
- To predict the classes of new data, the trained classifier finds the class with the smallest misclassification cost (see How the predict Method Classifies).

Linear discriminant analysis is also known as the Fisher discriminant, named for its inventor, Sir R. A. Fisher [2].

Create Discriminant Analysis Classifiers

To create the basic types of discriminant analysis classifiers for the Fisher iris data:

1. Load the data:

   ```
   load fisheriris;
   ```

2. Create a default (linear) discriminant analysis classifier:

   ```
   linclass = ClassificationDiscriminant.fit(meas,species);
   ```

   To visualize the classification boundaries of a 2-D linear classification of the data, see Linear Discriminant Classification — Fisher Training Data.

3. Classify an iris with average measurements:

```
4. meanmeas = mean(meas);

5. meanclass = predict(linclass,meanmeas)

6.

7. meanclass =

        'versicolor'
```

8. Create a quadratic classifier:

```
9. quadclass = ClassificationDiscriminant.fit(meas,species,...

        'discrimType','quadratic');
```

To visualize the classification boundaries of a 2-D quadratic classification of the data, see Quadratic Discriminant Classification — Fisher Training Data.

10. Classify an iris with average measurements using the quadratic classifier:

```
11. meanclass2 = predict(quadclass,meanmeas)

12.

13. meanclass2 =

         'versicolor'
```

Creating a Classifier Using ClassificationDiscriminant.fit

The model for discriminant analysis is:

- Each class (`Y`) generates data (`X`) using a multivariate normal distribution. In other words, the model assumes `X` has a Gaussian mixture distribution (`gmdistribution`).
  - For linear discriminant analysis, the model has the same covariance matrix for each class; only the means vary.
  - For quadratic discriminant analysis, both means and covariances of each class vary.

Under this modeling assumption, `ClassificationDiscriminant.fit` infers the mean and covariance parameters of each class.

- For linear discriminant analysis, it computes the sample mean of each class. Then it computes the sample covariance by first subtracting the sample mean of each class from the observations of that class, and taking the empirical covariance matrix of the result.
- For quadratic discriminant analysis, it computes the sample mean of each class. Then it computes the sample covariances by first subtracting the sample mean of each class from the observations of that class, and taking the empirical covariance matrix of each class.

The `fit` method does not use prior probabilities or costs for fitting.

Weighted Observations

The `fit` method constructs weighted classifiers using the following scheme. Suppose *M* is an *N*-by-*K* class membership matrix:

$M_{nk} = 1$ if observation *n* is from class *k*

$M_{nk} = 0$ otherwise.

The estimate of the class mean for unweighted data is

$$\hat{\mu}_k = \frac{\sum_{n=1}^{N} M_{nk} x_n}{\sum_{n=1}^{N} M_{nk}}.$$

For weighted data with positive weights *w*_n_, the natural generalization is

$$\hat{\mu}_k = \frac{\sum_{n=1}^{N} M_{nk} w_n x_n}{\sum_{n=1}^{N} M_{nk} w_n}.$$

The unbiased estimate of the pooled-in covariance matrix for unweighted data is

$$\hat{\Sigma} = \frac{\sum_{n=1}^{N} \sum_{k=1}^{K} M_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T}{N - K}.$$

For quadratic discriminant analysis, the `fit` method uses *K* = 1.

For weighted data, assuming the weights sum to 1, the unbiased estimate of the pooled-in covariance matrix is

$$\hat{\Sigma} = \frac{\sum_{n=1}^{N} \sum_{k=1}^{K} M_{nk} w_n (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T}{1 - \sum_{k=1}^{K} \frac{W_k^{(2)}}{W_k}},$$

where

- $W_k = \sum_{n=1}^{N} M_{nk} w_n$ is the sum of the weights for class *k*.
- $W_k^{(2)} = \sum_{n=1}^{N} M_{nk} w_n^2$ is the sum of squared weights per class.

How the predict Method Classifies

There are three elements in the `predict` classification algorithm:

- Posterior Probability
- Prior Probability
- Cost

`predict` classifies so as to minimize the expected classification cost:

$$\hat{y} = \underset{y=1,\dots,K}{\arg\min} \sum_{k=1}^{K} \hat{P}(k \mid x) C(y \mid k),$$

where

- $\hat{y}$ is the predicted classification.
- K is the number of classes.
- $\hat{P}(k \mid x)$ is the posterior probability of class *k* for observation *x*.
- $C(y \mid k)$ is the cost of classifying an observation as *y* when its true class is *k*.

The space of `X` values divides into regions where a classification `Y` is a particular value. The regions are separated by straight lines for linear discriminant analysis, and by conic sections (ellipses, hyperbolas, or parabolas) for quadratic discriminant analysis. For a visualization of these regions, see Create and Visualize a Discriminant Analysis Classifier.

Posterior Probability

The posterior probability that a point *x* belongs to class *k* is the product of the prior probability and the multivariate normal density. The density function of the multivariate normal with mean μ_k and covariance Σ_k at a point *x* is

$$P(x \mid k) = \frac{1}{\left(2\pi \left|\Sigma_k\right|\right)^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right),$$

where $\left|\Sigma_k\right|$ is the determinant of Σ_k, and Σ_k^{-1} is the inverse matrix.

Let $P(k)$ represent the prior probability of class *k*. Then the posterior probability that an observation *x* is of class *k* is

$$\hat{P}(k \mid x) = \frac{P(x \mid k)P(k)}{P(x)},$$

where $P(x)$ is a normalization constant, namely, the sum over *k* of $P(x|k)P(k)$.

Prior Probability

The prior probability is one of three choices:

- `'uniform'` — The prior probability of class `k` is 1 over the total number of classes.
- `'empirical'` — The prior probability of class `k` is the number of training samples of class `k` divided by the total number of training samples.
- A numeric vector — The prior probability of class `k` is the `j`th element of the `prior` vector. See `ClassificationDiscriminant.fit`.

After creating a classifier `obj`, you can set the prior using dot addressing:

```
obj.Prior = v;
```

where `v` is a vector of positive elements representing the frequency with which each element occurs. You do not need to retrain the classifier when you set a new prior.

Cost

There are two costs associated with discriminant analysis classification: the true misclassification cost per class, and the expected misclassification cost per observation.

True Misclassification Cost per Class. `Cost(i,j)` is the cost of classifying an observation into class `j` if its true class is `i`. By default, `Cost(i,j)=1` if `i~=j`, and `Cost(i,j)=0` if `i=j`. In other words, the cost is `0` for correct classification, and `1` for incorrect classification.

You can set any cost matrix you like when creating a classifier. Pass the cost matrix in the `Cost` name-value pair in `ClassificationDiscriminant.fit`.

After you create a classifier `obj`, you can set a custom cost using dot addressing:

```
obj.Cost = B;
```

`B` is a square matrix of size `K`-by-`K` when there are `K` classes. You do not need to retrain the classifier when you set a new cost.

Expected Misclassification Cost per Observation. Suppose you have `Nobs` observations that you want to classify with a trained discriminant analysis classifier `obj`. Suppose you have `K` classes. You place the observations into a matrix `Xnew` with one observation per row. The command

```
[label,score,cost] = predict(obj,Xnew)
```

returns, among other outputs, a cost matrix of size `Nobs`-by-`K`. Each row of the cost matrix contains the expected (average) cost of classifying the observation into each of the `K` classes. `cost(n,k)` is

$$\sum_{i=1}^{K} \hat{P}(i \mid Xnew(n)) C(k \mid i),$$

where

- *K* is the number of classes.
- $\hat{P}(i \mid Xnew(n))$ is the posterior probability of class *i* for observation *Xnew*(*n*).

- $C(k|i)$ is the cost of classifying an observation as *k* when its true class is *i*.

Create and Visualize a Discriminant Analysis Classifier

This example shows both linear and quadratic classification of the Fisher iris data. The example uses only two of the four predictors to enable simple plotting.

1. Load the data:

```
load fisheriris
```

2. Use the petal length (`PL`) and petal width (`PW`) measurements:

```
PL = meas(:,3);

PW = meas(:,4);
```

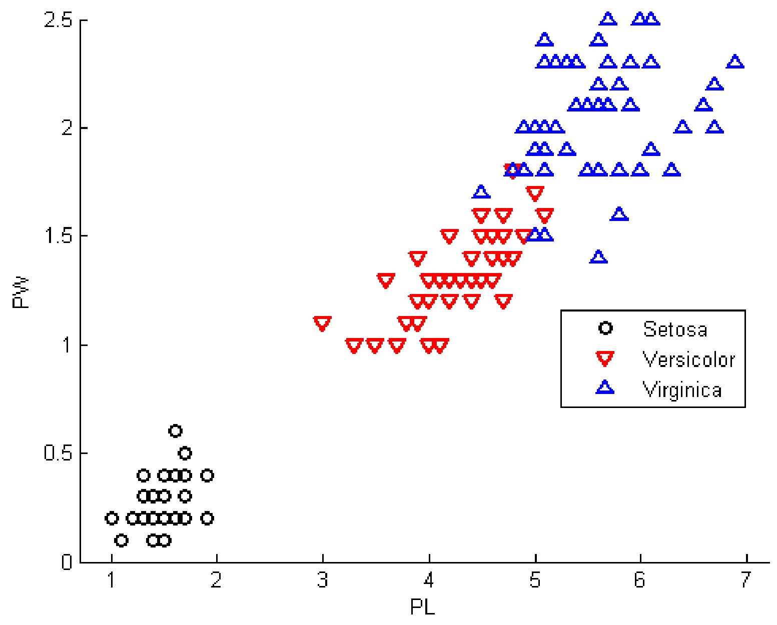
3. Plot the data, showing the classification:

```
h1 = gscatter(PL,PW,species,'krb','ov^',[],'off');

set(h1,'LineWidth',2)

legend('Setosa','Versicolor','Virginica',...

       'Location','best')
```



4. Create a linear classifier:

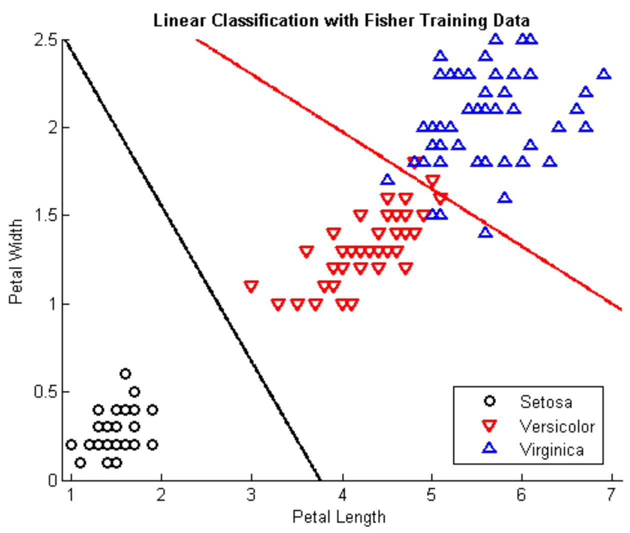
```
X = [PL,PW];
cls = ClassificationDiscriminant.fit(X,species);
```

5. Plot the classification boundaries:

```
hold on
K = cls.Coeffs(2,3).Const;
L = cls.Coeffs(2,3).Linear;
% Plot the curve K + [x,y]*L  = 0:
f = @(x1,x2) K + L(1)*x1 + L(2)*x2;
h2 = ezplot(f,[.9 7.1 0 2.5]);
set(h2,'Color','r','LineWidth',2)

K = cls.Coeffs(1,2).Const;
L = cls.Coeffs(1,2).Linear;
% Plot the curve K + [x1,x2]*L  = 0:
f = @(x1,x2) K + L(1)*x1 + L(2)*x2;
h3 = ezplot(f,[.9 7.1 0 2.5]);
set(h3,'Color','k','LineWidth',2)
axis([.9 7.1 0 2.5])
xlabel('Petal Length')
ylabel('Petal Width')
title('{\bf Linear Classification with Fisher Training Data}')
```

Linear Discriminant Classification — Fisher Training Data



6. Create a quadratic discriminant classifier:

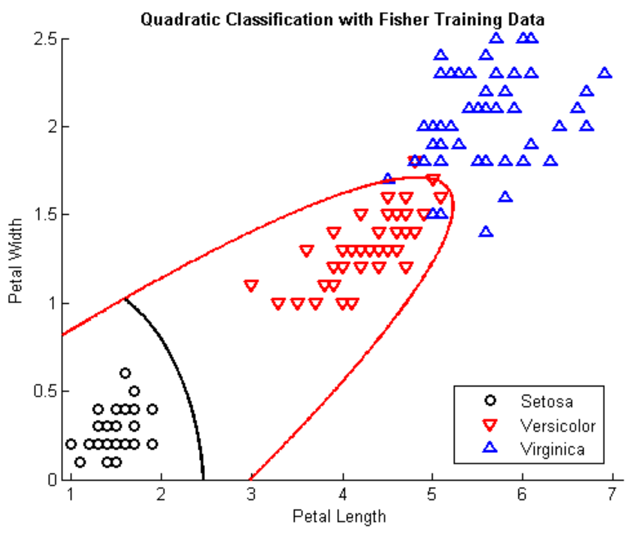
```
cqs = ClassificationDiscriminant.fit(X,species,...
      'DiscrimType','quadratic');
```

7. Plot the classification boundaries:

```
  delete(h2); delete(h3) % remove the linear plots
K = cqs.Coeffs(2,3).Const;
L = cqs.Coeffs(2,3).Linear;
Q = cqs.Coeffs(2,3).Quadratic;
% Plot the curve K + [x1,x2]*L + [x1,x2]*Q*[x1,x2]'=0:
f = @(x1,x2) K + L(1)*x1 + L(2)*x2 + Q(1,1)*x1.^2 + ...
    (Q(1,2)+Q(2,1))*x1.*x2 + Q(2,2)*x2.^2;
h2 = ezplot(f,[.9 7.1 0 2.5]);
set(h2,'Color','r','LineWidth',2)

K = cqs.Coeffs(1,2).Const;
L = cqs.Coeffs(1,2).Linear;
Q = cqs.Coeffs(1,2).Quadratic;
% Plot the curve K + [x1,x2]*L + [x1,x2]*Q*[x1,x2]'=0:
f = @(x1,x2) K + L(1)*x1 + L(2)*x2 + Q(1,1)*x1.^2 + ...
    (Q(1,2)+Q(2,1))*x1.*x2 + Q(2,2)*x2.^2;
h3 = ezplot(f,[.9 7.1 0 1.02]); % plot the relevant
                                % portion of the curve
set(h3,'Color','k','LineWidth',2)
axis([.9 7.1 0 2.5])
xlabel('Petal Length')
ylabel('Petal Width')
title('{\bf Quadratic Classification with Fisher Training Data}')
hold off
```

Quadratic Discriminant Classification — Fisher Training Data



Improve a Discriminant Analysis Classifier

- Deal with Singular Data
- Choose a Discriminant Type

- Examine the Resubstitution Error and Confusion Matrix
- Cross Validation
- Change Costs and Priors

Deal with Singular Data

Discriminant analysis needs data sufficient to fit Gaussian models with invertible covariance matrices. If your data is not sufficient to fit such a model uniquely, `ClassificationDiscriminant.fit` fails. This section shows methods for handling failures.

Tip To obtain a discriminant analysis classifier without failure, set the `DiscrimType` name-value pair to `'pseudoLinear'` or `'pseudoQuadratic'` in `ClassificationDiscriminant.fit`.

"Pseudo" discriminants never fail, because they use the pseudoinverse of the covariance matrix Σ_k (see `pinv`).

8 . **Example: Singular Covariance Matrix.** When the covariance matrix of the fitted classifier is singular,

`ClassificationDiscriminant.fit` can fail:

```
load popcorn
X = popcorn(:,[1 2]);
X(:,3) = 0; % a zero-variance column
Y = popcorn(:,3);
ppcrn = ClassificationDiscriminant.fit(X,Y);

Error using ClassificationDiscriminant (line 635)
Predictor x3 has zero variance. Either exclude this predictor or set
'discrimType' to
'pseudoLinear' or 'diagLinear'.

Error in classreg.learning.FitTemplate/fit (line 243)
            obj=this.MakeFitObject(X,Y,W,this.ModelParams,fitArgs{:});

Error in ClassificationDiscriminant.fit (line 296)
            this = fit(temp,X,Y);
```

To proceed with linear discriminant analysis, use a `pseudoLinear` or `diagLinear` discriminant type:

```
ppcrn = ClassificationDiscriminant.fit(X,Y,...
    'discrimType','pseudoLinear');
meanpredict = predict(ppcrn,mean(X))


meanpredict =
    3.5000
```

Choose a Discriminant Type

There are six types of discriminant analysis classifiers: linear and quadratic, with diagonal and pseudo variants of each type.

Tip To see if your covariance matrix is singular, set `discrimType` to `'linear'` or `'quadratic'`. If the matrix is singular, the `ClassificationDiscriminant.fit` method fails for `'quadratic'`, and the `Gamma` property is nonzero for `'linear'`.

To obtain a quadratic classifier even when your covariance matrix is singular, set `discrimType` to `'pseudoQuadratic'` or `'diagQuadratic'`.

```
obj = ClassificationDiscriminant.fit(X,Y,...
    'discrimType','pseudoQuadratic') % or 'diagQuadratic'
```

Choose a classifier type by setting the `discrimType` name-value pair to one of:

- `'linear'` (default) — Estimate one covariance matrix for all classes.
- `'quadratic'` — Estimate one covariance matrix for each class.
- `'diagLinear'` — Use the diagonal of the `'linear'` covariance matrix, and use its pseudoinverse if necessary.
- `'diagQuadratic'` — Use the diagonals of the `'quadratic'` covariance matrices, and use their pseudoinverses if necessary.
- `'pseudoLinear'` — Use the pseudoinverse of the `'linear'` covariance matrix if necessary.
- `'pseudoQuadratic'` — Use the pseudoinverses of the `'quadratic'` covariance matrices if necessary.

`ClassificationDiscriminant.fit` can fail for the `'linear'` and `'quadratic'` classifiers. When it fails, it returns an explanation, as shown in Deal with Singular Data.

`ClassificationDiscriminant.fit` always succeeds with the diagonal and pseudo variants. For information about pseudoinverses, see `pinv`.

You can set the discriminant type using dot addressing after constructing a classifier:

```
obj.DiscrimType = 'discrimType'
```

You can change between linear types or between quadratic types, but cannot change between a linear and a quadratic type.

Examine the Resubstitution Error and Confusion Matrix

The *resubstitution error* is the difference between the response training data and the predictions the classifier makes of the response based on the input training data. If the resubstitution error is high, you cannot expect the predictions of the classifier to be good. However, having low resubstitution error does not guarantee good predictions for new data. Resubstitution error is often an overly optimistic estimate of the predictive error on new data.

The *confusion matrix* shows how many errors, and which types, arise in resubstitution. When there are `K` classes, the confusion matrix `R` is a `K`-by-`K` matrix with

`R(i,j)` = the number of observations of class `i` that the classifier predicts to be of class `j`.

Example: Resubstitution Error of a Discriminant Analysis Classifier. Examine the resubstitution error of the default discriminant analysis classifier for the Fisher iris data:

```
load fisheriris
obj = ClassificationDiscriminant.fit(meas,species);
resuberror = resubLoss(obj)
```

```
resuberror =
        0.0200
```

The resubstitution error is very low, meaning `obj` classifies nearly all the Fisher iris data correctly. The total number of misclassifications is:

```
resuberror * obj.NObservations
```

```
ans =
    3.0000
```

To see the details of the three misclassifications, examine the confusion matrix:

```
R = confusionmat(obj.Y,resubPredict(obj))

R =
    50     0     0
     0    48     2
     0     1    49

obj.ClassNames


ans =
    'setosa'
    'versicolor'
    'virginica'
```

`R(1,:) = [50 0 0]` means `obj` classifies all 50 setosa irises correctly.

`R(2,:) = [0 48 2]` means `obj` classifies 48 versicolor irises correctly, and misclassifies two versicolor irises as virginica.

`R(3,:) = [0 1 49]` means `obj` classifies 49 virginica irises correctly, and misclassifies one virginica iris as versicolor.

Cross Validation

Typically, discriminant analysis classifiers are robust and do not exhibit overtraining when the number of predictors is much less than the number of observations. Nevertheless, it is good practice to cross validate your classifier to ensure its stability.

Example: Cross Validating a Discriminant Analysis Classifier. Try five-fold cross validation of a quadratic discriminant analysis classifier:

1. Load the Fisher iris data:

```
load fisheriris
```

2. Create a quadratic discriminant analysis classifier for the data:

```
quadisc = ClassificationDiscriminant.fit(meas,species,...
'DiscrimType','quadratic');
```

3. Find the resubstitution error of the classifier:

```
qerror = resubLoss(quadisc)
qerror =
    0.0200
```

   The classifier does an excellent job. Nevertheless, resubstitution error can be an optimistic estimate of the error when classifying new data. So proceed to cross validation.

4. Create a cross-validation model:

```
cvmodel = crossval(quadisc,'kfold',5);
```

5. Find the cross-validation loss for the model, meaning the error of the out-of-fold observations:

```
cverror = kfoldLoss(cvmodel)
cverror =
    0.0333
```

   The cross-validated loss is nearly as low as the original resubstitution loss. Therefore, you can have confidence that the classifier is reasonably accurate.

Change Costs and Priors

Sometimes you want to avoid certain misclassification errors more than others. For example, it might be better to have oversensitive cancer detection instead of undersensitive cancer detection. Oversensitive detection gives more false positives (unnecessary testing or treatment). Undersensitive detection gives more false negatives (preventable illnesses or deaths). The consequences of underdetection can be high. Therefore, you might want to set costs to reflect the consequences.

Similarly, the training data `Y` can have a distribution of classes that does not represent their true frequency. If you have a better estimate of the true frequency, you can include this knowledge in the classification `prior` property.

Change Costs and Priors

Sometimes you want to avoid certain misclassification errors more than others. For example, it might be better to have oversensitive cancer detection instead of undersensitive cancer detection. Oversensitive detection gives more false positives (unnecessary testing or treatment). Undersensitive detection gives more false negatives (preventable illnesses or deaths). The consequences of underdetection can be high. Therefore, you might want to set costs to reflect the consequences.

Similarly, the training data `Y` can have a distribution of classes that does not represent their true frequency. If you have a better estimate of the true frequency, you can include this knowledge in the classification `prior` property.

Example: Setting Custom Misclassification Costs. Consider the Fisher iris data. Suppose that the cost of classifying a versicolor iris as virginica is 10 times as large as making any other classification error. Create a classifier from the data, then incorporate this cost and then view the resulting classifier.

1. Load the Fisher iris data and create a default (linear) classifier as in Example: Resubstitution Error of a Discriminant Analysis Classifier:

```
load fisheriris
obj = ClassificationDiscriminant.fit(meas,species);
resuberror = resubLoss(obj)
resuberror =
    0.0200
R = confusionmat(obj.Y,resubPredict(obj))
R =
    50     0     0
     0    48     2
     0     1    49
obj.ClassNames
ans =
    'setosa'
    'versicolor'
    'virginica'
```

`R(2,:) = [0 48 2]` means `obj` classifies 48 versicolor irises correctly, and misclassifies two versicolor irises as virginica.

2. Change the cost matrix to make fewer mistakes in classifying versicolor irises as virginica:

```
obj.Cost(2,3) = 10;
R2 = confusionmat(obj.Y,resubPredict(obj))
R2 =
    50     0     0
     0    50     0
     0     7    43
```

`obj` now classifies all versicolor irises correctly, at the expense of increasing the number of misclassifications of virginica irises from `1` to `7`.

Example: Setting Alternative Priors. Consider the Fisher iris data. There are 50 irises of each kind in the data. Suppose that, in a particular region, you have historical data that shows virginica are five times as prevalent as the other kinds. Create a classifier that incorporates this information.

1. Load the Fisher iris data and make a default (linear) classifier as in Example: Resubstitution Error of a Discriminant Analysis Classifier:

```
load fisheriris
obj = ClassificationDiscriminant.fit(meas,species);
resuberror = resubLoss(obj)
resuberror =
    0.0200
R = confusionmat(obj.Y,resubPredict(obj))
R =
50     0     0
0     48     2
0      1    49
obj.ClassNames
ans =
     'setosa'
```

```
      'versicolor'
      'virginica'
```

`R(3,:) = [0 1 49]` means `obj` classifies 49 virginica irises correctly, and misclassifies one virginica iris as versicolor.

2. Change the prior to match your historical data, and examine the confusion matrix of the new classifier:

```
obj.Prior = [1 1 5];
R2 = confusionmat(obj.Y,resubPredict(obj))
R2 =
    50      0      0
     0     46      4
     0      0     50
```

The new classifier classifies all virginica irises correctly, at the expense of increasing the number of misclassifications of versicolor irises from 2 to 4.

Regularize a Discriminant Analysis Classifier

To make a more robust and simpler model, try to remove predictors from your model without hurting its predictive power. This is especially important when you have many predictors in your data. Linear discriminant analysis uses the two regularization parameters, Gamma and Delta, to identify and remove redundant predictors. The cvshrink method helps you identify appropriate settings for these parameters.

1. Load data and create a classifier.

Create a linear discriminant analysis classifier for the ovariancancer data. Set the SaveMemory and FillCoeffs options to keep the resulting model reasonably small.

```
load ovariancancer
obj = ClassificationDiscriminant.fit(obs,grp,...
    'SaveMemory','on','FillCoeffs','off');
```

2. Cross validate the classifier.

Use 30 levels of `Gamma` and 30 levels of `Delta` to search for good parameters. This search is time consuming. Set `Verbose` to `1` to view the progress.

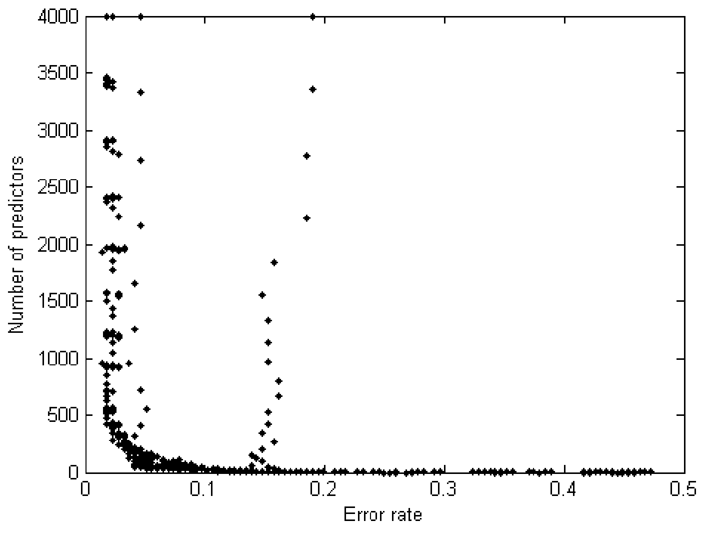
```
rng(8000,'twister') % for reproducibility
[err,gamma,delta,numpred] = cvshrink(obj,...
    'NumGamma',29,'NumDelta',29,'Verbose',1);
```

```
Done building cross-validated model.
Processing Gamma step 1 out of 30.
Processing Gamma step 2 out of 30.
Processing Gamma step 3 out of 30.
%%% (many lines removed) %%%
Processing Gamma step 28 out of 30.
Processing Gamma step 29 out of 30.
Processing Gamma step 30 out of 30.
```

3. Examine the quality of the regularized classifiers.

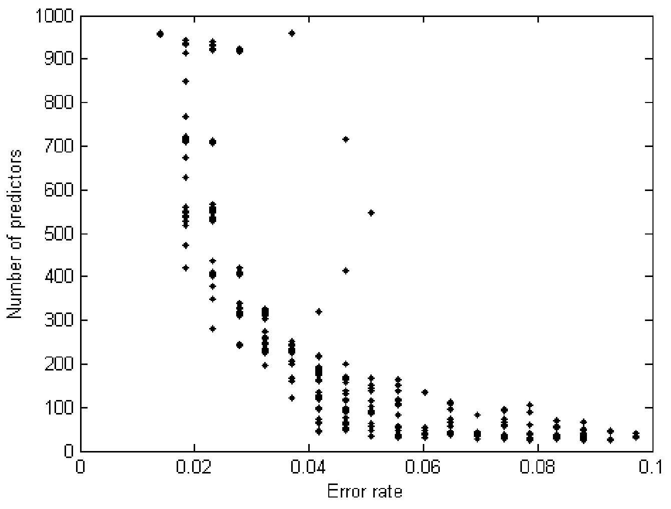
Plot the number of predictors against the error.

```
figure;
plot(err,numpred,'k.')
xlabel('Error rate');
ylabel('Number of predictors');
```



Examine the lower-left part of the plot more closely.

```
axis([0 .1 0 1000])
```



There is a clear tradeoff between lower number of predictors and lower error.

4. Choose an optimal tradeoff between model size and accuracy.

Find the values of `Gamma` and `Delta` that give minimal error

```
minerr = min(min(err));
[p,q] = find(err == minerr)
```

```
p =

    24
    25


q =

     8
     8
```

Two points have the same minimal error: `[24,8]` and `[25,8]`, which correspond to

```
[gamma(p(1)),delta(p(1),q(1))]
```

```
ans =

    0.8436    0.1463
```

```
[gamma(p(2)),delta(p(2),q(2))]
```

```
ans =

    0.8697    0.1425
```

These points correspond to about a quarter of the total predictors having nonzero coefficients in the model.

```
numpred(p(1),q(1))
```

```
ans =

   957
```

```
numpred(p(2),q(2))
```

```
ans =

   960
```

To further lower the number of predictors, you must accept larger error rates. For example, to choose the `Gamma` and `Delta` that give the lowest error rate with 250 or fewer predictors:

```
low250 = min(min(err(numpred <= 250)))
```

```
low250 =

    0.0278
```

```
lownum = min(min(numpred(err == low250)))
```

```
lownum =

   243
```

You need 243 predictors to achieve an error rate of 0.0278, and this is the lowest error rate among those that have 250 predictors or fewer. The `Gamma` and `Delta` that achieve this error/number of predictors:

```
[r,s] = find((err == low250) & (numpred == lownum));
gamma(r)
```

```
ans =

    0.7133
```

```
delta(r,s)
```

```
ans =

     0.2960
```

5. Set the regularization parameters.

To set the classifier with these values of `Gamma` and `Delta`, use dot addressing.

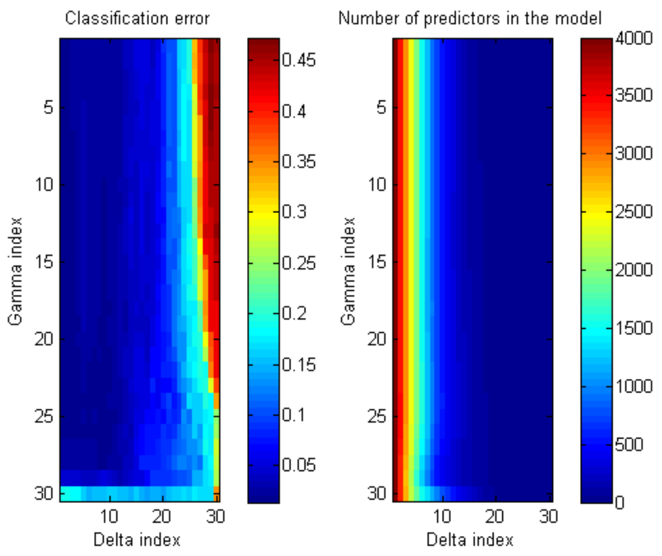
```
obj.Gamma = gamma(r);
obj.Delta = delta(r,s);
```

6. Heat map plot.

To compare the `cvshrink` calculation to that in Guo, Hastie, and Tibshirani [3], plot heat maps of error and number of predictors against `Gamma` and the index of the `Delta` parameter. (The `Delta` parameter range depends on the value of the `Gamma` parameter. So to get a rectangular plot, use the `Delta` index, not the parameter itself.)

```
% First create the Delta index matrix
indx = repmat(1:size(delta,2),size(delta,1),1);
figure
subplot(1,2,1)
imagesc(err);
colorbar;
title('Classification error');
xlabel('Delta index');
ylabel('Gamma index');

subplot(1,2,2)
imagesc(numpred);
colorbar;
title('Number of predictors in the model');
xlabel('Delta index');
ylabel('Gamma index');
```



You see the best classification error when `Delta` is small, but fewest predictors when `Delta` is large.

Examine the Gaussian Mixture Assumption

Discriminant analysis assumes that the data comes from a Gaussian mixture model (see Creating a Classifier Using ClassificationDiscriminant.fit). If the data appears to come from a Gaussian mixture model, you can expect discriminant analysis to be a good classifier. Furthermore, the default linear discriminant analysis assumes that all class covariance matrices are equal. This section shows methods to check these assumptions:

- Bartlett Test of Equal Covariance Matrices for Linear Discriminant Analysis
- Q-Q Plot
- Mardia Kurtosis Test of Multivariate Normality

Bartlett Test of Equal Covariance Matrices for Linear Discriminant Analysis

The Bartlett test (see Box [1]) checks equality of the covariance matrices of the various classes. If the covariance matrices are equal, the test indicates that linear discriminant analysis is appropriate. If not, consider using quadratic discriminant analysis, setting the `DiscrimType` name-value pair to `'quadratic'` in `ClassificationDiscriminant.fit`.

The Bartlett test assumes normal (Gaussian) samples, where neither the means nor covariance matrices are known. To determine whether the covariances are equal, compute the following quantities:

- Sample covariance matrices per class σ_i, $1 \le i \le k$, where k is the number of classes.
- Pooled-in covariance matrix σ.
- Test statistic V:

$$V = (n-k)\log(|\Sigma|) - \sum_{i=1}^{k}(n_i - 1)\log(|\Sigma_i|)$$

where n is the total number of observations, and n_i is the number of observations in class i, and $|\Sigma|$ means the determinant of the matrix Σ.

- Asymptotically, as the number of observations in each class n_i become large, V is distributed approximately χ^2 with $kd(d + 1)/2$ degrees of freedom, where d is the number of predictors (number of dimensions in the data).

The Bartlett test is to check whether V exceeds a given percentile of the χ^2 distribution with $kd(d + 1)/2$ degrees of freedom. If it does, then reject the hypothesis that the covariances are equal.

Example: Bartlett Test for Equal Covariance Matrices. Check whether the Fisher iris data is well modeled by a single Gaussian covariance, or whether it would be better to model it as a Gaussian mixture.

```
load fisheriris;
prednames = {'SepalLength','SepalWidth',...
    'PetalLength','PetalWidth'};
L = ClassificationDiscriminant.fit(meas,species,...
    'PredictorNames',prednames);
Q = ClassificationDiscriminant.fit(meas,species,...
    'PredictorNames',prednames,'DiscrimType','quadratic');
D = 4; % Number of dimensions of X
Nclass = [50 50 50];
N = L.NObservations;
K = numel(L.ClassNames);
SigmaQ = Q.Sigma;
SigmaL = L.Sigma;
logV = (N-K)*log(det(SigmaL));
for k=1:K
    logV = logV - (Nclass(k)-1)*log(det(SigmaQ(:,:,k)));
end
nu = (K-1)*D*(D+1)/2;
pval = 1-chi2cdf(logV,nu)

pval =
     0
```

The Bartlett test emphatically rejects the hypothesis of equal covariance matrices. If `pval` had been greater than `0.05`, the test would not have rejected the hypothesis. The result indicates to use quadratic discriminant analysis, as opposed to linear discriminant analysis.

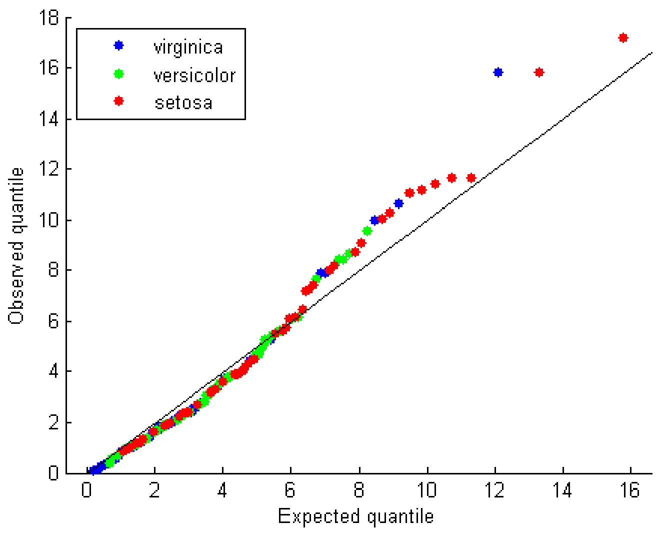
Q-Q Plot

A Q-Q plot graphically shows whether an empirical distribution is close to a theoretical distribution. If the two are equal, the Q-Q plot lies on a 45° line. If not, the Q-Q plot strays from the 45° line.

Check Q-Q Plots for Linear and Quadratic Discriminants. For linear discriminant analysis, use a single covariance matrix for all classes.

```
load fisheriris;
prednames = {'SepalLength','SepalWidth',...
    'PetalLength','PetalWidth'};
L = ClassificationDiscriminant.fit(meas,species,...
    'PredictorNames',prednames);
N = L.NObservations;
K = numel(L.ClassNames);
```

```
mahL = mahal(L,L.X,'ClassLabels',L.Y);
D = 4;
expQ = chi2inv(((1:N)-0.5)/N,D); % expected quantiles
[mahL,sorted] = sort(mahL); % sorted obbserved quantiles
figure;
gscatter(expQ,mahL,L.Y(sorted),'bgr',[],[],'off');
legend('virginica','versicolor','setosa','Location','NW');
xlabel('Expected quantile');
ylabel('Observed quantile');
line([0 20],[0 20],'color','k');
```

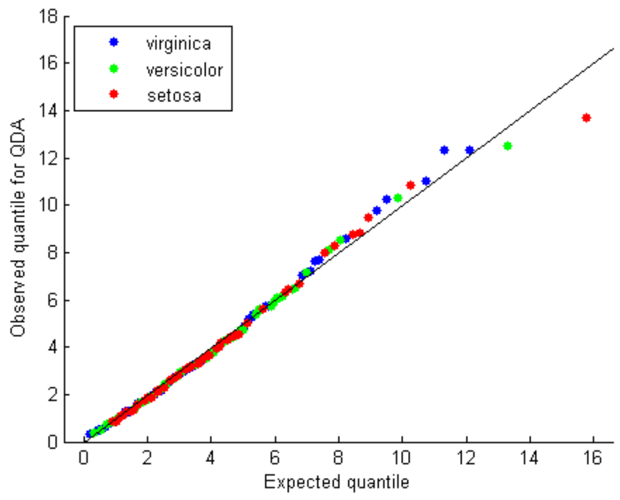


Overall, the agreement between the expected and observed quantiles is good. Look at the right half of the plot. The deviation of the plot from the 45° line upward indicates that the data has tails heavier than a normal distribution. There are three possible outliers on the right: two observations from class `'setosa'` and one observation from class `'virginica'`.

As shown in Bartlett Test of Equal Covariance Matrices for Linear Discriminant Analysis, the data does not match a single covariance matrix. Redo the calculations for a quadratic discriminant

```
load fisheriris;
prednames = {'SepalLength','SepalWidth',...
    'PetalLength','PetalWidth'};
Q = ClassificationDiscriminant.fit(meas,species,...
    'PredictorNames',prednames,'DiscrimType','quadratic');
Nclass = [50 50 50];
N = L.NObservations;
K = numel(L.ClassNames);
mahQ = mahal(Q,Q.X,'ClassLabels',Q.Y);
expQ = chi2inv(((1:N)-0.5)/N,D);
[mahQ,sorted] = sort(mahQ);
figure;
gscatter(expQ,mahQ,Q.Y(sorted),'bgr',[],[],'off');
legend('virginica','versicolor','setosa','Location','NW');
```

```
xlabel('Expected quantile');
ylabel('Observed quantile for QDA');
line([0 20],[0 20],'color','k');
```



The Q-Q plot shows a better agreement between the observed and expected quantiles. There is only one outlier candidate, from class `'setosa'`.

Mardia Kurtosis Test of Multivariate Normality

The Mardia kurtosis test (see Mardia [4]) is an alternative to examining a Q-Q plot. It gives a numeric approach to deciding if data matches a Gaussian mixture model.

In the Mardia kurtosis test you compute M, the mean of the fourth power of the Mahalanobis distance of the data from the class means. If the data is normally distributed with constant covariance matrix (and is thus suitable for linear discriminant analysis), M is asymptotically distributed as normal with mean $d(d + 2)$ and variance $8d(d + 2)/n$, where

- d is the number of predictors (number of dimensions in the data).
- n is the total number of observations.

The Mardia test is two sided: check whether M is close enough to $d(d + 2)$ with respect to a normal distribution of variance $8d(d + 2)/n$.

Example: Mardia Kurtosis Test for Linear and Quadratic Discriminants. Check whether the Fisher iris data is approximately normally distributed for both linear and quadratic discriminant analysis. According to Bartlett Test of Equal Covariance Matrices for Linear Discriminant Analysis, the data is not normal for linear discriminant analysis (the covariance matrices are different). Check Q-Q Plots for Linear and Quadratic Discriminants indicates that the data is well modeled by a Gaussian mixture model with different covariances per class. Check these conclusions with the Mardia kurtosis test:

```
load fisheriris;
prednames = {'SepalLength','SepalWidth',...
    'PetalLength','PetalWidth'};
L = ClassificationDiscriminant.fit(meas,species,...
    'PredictorNames',prednames);
mahL = mahal(L,L.X,'ClassLabels',L.Y);
D = 4;
```

```
N = L.NObservations;
obsKurt = mean(mahL.^2);
expKurt = D*(D+2);
varKurt = 8*D*(D+2)/N;
[~,pval] = ztest(obsKurt,expKurt,sqrt(varKurt))

pval =
    0.0208
```

The Mardia test indicates to reject the hypothesis that the data is normally distributed.

Continuing the example with quadratic discriminant analysis:

```
Q = ClassificationDiscriminant.fit(meas,species,...
    'PredictorNames',prednames,'DiscrimType','quadratic');
mahQ = mahal(Q,Q.X,'ClassLabels',Q.Y);
obsKurt = mean(mahQ.^2);
[~,pval] = ztest(obsKurt,expKurt,sqrt(varKurt))

pval =
    0.7230
```

Because `pval` is high, you conclude the data are consistent with the multivariate normal distribution.

Bibliography

[1] Box, G. E. P. *A General Distribution Theory for a Class of Likelihood Criteria.* Biometrika 36(3), pp. 317–346, 1949.

[2] Fisher, R. A. *The Use of Multiple Measurements in Taxonomic Problems.* Annals of Eugenics, Vol. 7, pp. 179–188, 1936. Available at http://digital.library.adelaide.edu.au/dspace/handle/2440/15227.

[3] Guo, Y., T. Hastie, and R. Tibshirani. *Regularized Discriminant Analysis and Its Application in Microarray.* Biostatistics, Vol. 8, No. 1, pp. 86–100, 2007.

[4] Mardia, K. V. *Measures of multivariate skewness and kurtosis with applications.* Biometrika 57 (3), pp. 519–530, 1970.