Classification of Fisher's Iris data

Andersson, Alexander (alexan@student·chalmers·se)
Källberg, Andreas (andkal@student·chalmers·se)
Sigurdhsson, Simon (ssimon@student·chalmers·se)

March 1, 2009

Abstract

Using statistics, specifically odds and Bayes theorem to construct a semi-intelligent "learning" algorithm that can classify data, in this case data from Fisher's Iris set.

1 The Problem

Suppose we have a huge data set, and each member of this data set has a number of properties. We need to classify these members, and because of certain circumstances, i.e. time, cost or similar, we can't do this by hand. Enter pattern recognition.

Pattern recognition is a useful tool in many different areas, ranging from genetics to OCR (Optical Character Recognition). Pattern recognition is often needed in image analysis. The most useful form of pattern recognition is so called "supervised learning" – where you teach the algorithm how to classify data using a given set of parameters. All you need is a large set of data to teach your algorithm with and some basic understanding of odds and Bayes theorem.

1.1 Behind the Scenes

Using Bayes formula in this context is simple; one easily understands that it is the priori/posteriori odds variety we need to use:

$$q_i^{post} = L(A_i) q_i^{prior}$$

Here we define A_i as the event that something belongs to a certain class, i.e. A_i ="character is a 'C' " or something along those lines, depending on what you want to classify. q_i is, as always, the odds for the event A_i.

One can easily spot a problem here; to use this equation we need the likelihood function $L(A_i) = P(X \approx x|A_i)$, which we clearly don't have. That's when supervised learning comes in handy. We estimate the likelihood function from a given set of data for which we *know* that A_i is true.

2 A Botanic Example

To illustrate the supervised learning method we will use MATLAB and Fisher's Iris data set, a data set containing four series of measurements on Iris flowers; it contains information about the sepal length, sepal width, petal length and petal width of the flowers, which are from the Setosa, Versicolor and Virginica species. This means that we have three classes (the species), and a data set with feature vectors in $\mathbb{R}^4$ for which we already know the class. Thus, we can use this data set to construct and teach an algorithm how to classify Iris flowers.

First, we explore our data. We plot flowers against sepal width and petal length, as shown in figure 1, to get a good idea of how the data is distributed

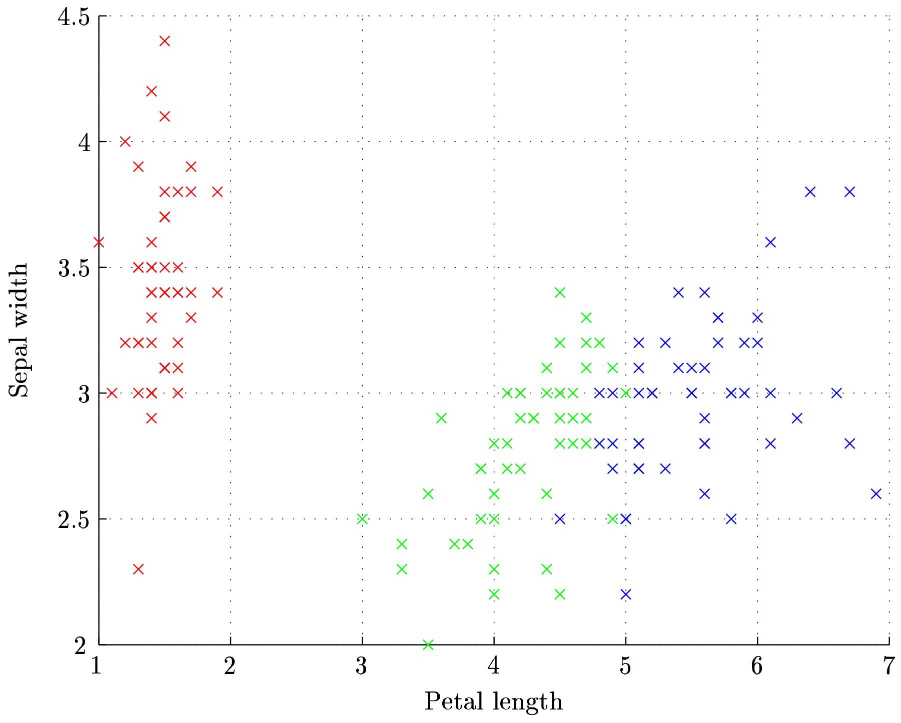

Figure 1: Classification of a subset of the Iris data, plotted against sepal width and petal length. **Legend:** Setosa, Versidor, Virginica.

over these two variables. The different species are clearly differentiated, even though Versidor and Virginica are overlapping a little. That means that we can use these variables to classify flowers with pretty high accuracy. In figure 2 lines have been added, separating the different classes with lines (note: axis has been flipped).

We still don't have a likelyhood function, but we can assume (it's not entirely correct) that the inner workings behind this function is normally distributed with the following multivariate probability density function:

$$f_i(x) = \frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)\right)$$

Σ_i is the covariance matrix for class i, and μ_i is the mean feature vector. We can assume that all classes have the same covariance matrix, that will make things easier. Now, if we test a feature vector using these functions ($f_i(x)$) we can easily determine what class the feature vector is likely to belong in. If, for example, $f_1(x) > f_2(x) > f_3(x)$, the given feature vector is more likely to belong to class 1, in our case the Setosa species.

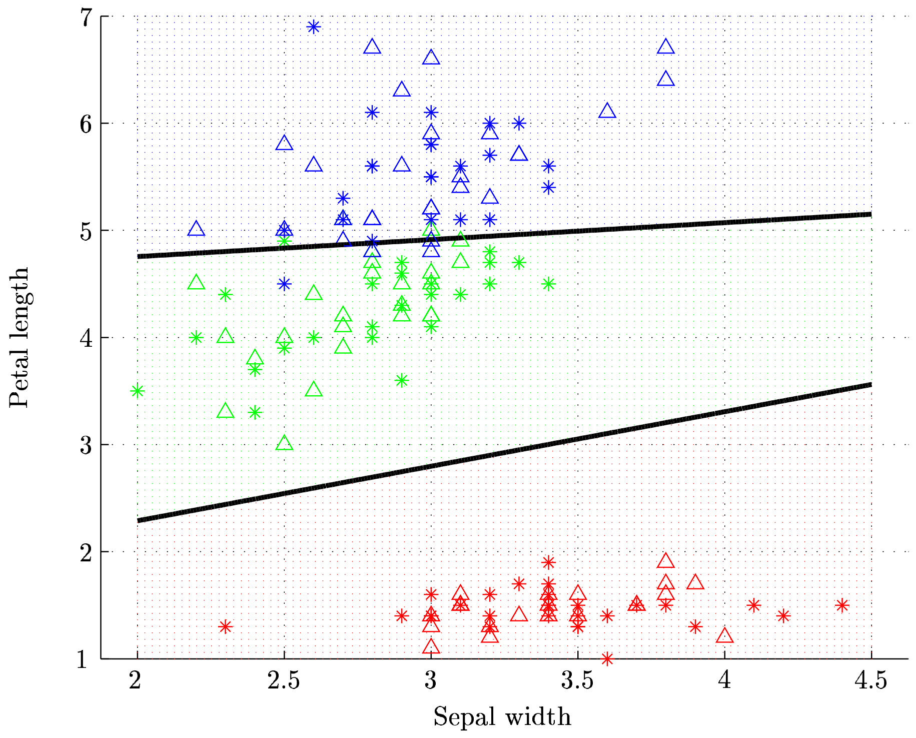

Figure 2: Classification of Iris data, with lines separating the areas and some test data plotted. Training data is plotted as stars, test data as triangles.

2.1 We Love MATLAB

Fortunately, MATLAB has a statistics toolbox, which contains nifty functions that help setting up the classifying function for us. And – drumroll – it's called `classify`! However, we have to set up some variables before we use this nice function.

```
randindex = randperm(length(class));
index1 = randindex(1:75);
index2 = randindex(76:150);
features_training = features(index1,:);
features_validation = features(index2,:);
class_training = class(index1);
class_validation = class(index2);
```

This divides the data into two sets; we will use one set for training and the other one to test our results. We'll explain the code further – first, `randperm` generates a random permutations of all positive integers less than or equal to the size of our data set. Then, we split this into two parts. This gives us two sets of integers, which we use to randomly pick feature vectors to put in our training and validation sets.

We can now use `classify` on our data. It's an easy procedure – the version of `classify` we're going to use takes four arguments; the data set we want to validate, our training data set, the corresponding classes of our training set and finally a parameter telling MATLAB how to adapt the data, in our case this parameter will be *linear*. We also calculate and display how much the algorithm's guesses diverge from the actual values (remember, we have a *validation* set, for which we generate guesses) and this gives us a measure of how good the algorithm is. Behold the MATLAB code:

```
allegedclass = classify(features_validation, ...
                        features_training, ...
                        class_training, 'linear');
sum(allegedclass==class_validation)/length(allegedclass)
```

The printed value (let's call it the "*perfection value*" of the algorithm) is 0.9733. The closer to 1 this value is, the better – it simply tells us how many percent of the algorithm's guesses were correct. In this case, the value we got is unacceptable. How can we perfect the algorithm?

Multiplying values with a constant will do nothing, since we have a linear model. Our best bet is to take the square of all values of both the training set and the validation set (and subsequently of all feature vectors we wish to qualify. Hopefully this will give us a better algorithm:

```
features_training2   = features_training.^2;
features_validation2 = features_validation.^2;
allegedclass = classify(features_validation2(:,2:4), ...
                        features_training2(:,2:4), ...
                        class_training, 'linear');
sum(allegedclass==class_validation)/length(allegedclass)
```

We get a perfection value of 1 now. Great, this means that our algorithm is really good – taking the square of all values clearly made a difference. Not surprising really, since this effectively increases the resolution of our training (and validation) set.

2.2 Further Investigation

We investigated different combinations of features to find a suitable one. The results are available in table 1, and quite clearly show that the method suggested was one of the better. One can also see a decline in perfection when more variables are introduced, with the optimal number of features around 2 or 3 – however, this might be a coincidence since the perfection value when all four features are considered doesn't deviate very much from the best combinations. What one clearly can see is that sepal length alone is a pretty bad feature, but combined with others it may prove very useful.

We also tried smaller training sets, and acheived surprisingly excellent results – this shows that pattern recognition may be a good technique even if you have limited training data.

3 Conclusions

Evidently, pattern recognition is a hard problem to solve, but if the problem is investigated properly and one finds an optimal combination of features and a good training set, great results can be acheived. This is something you might see more often than you think – since the technique is used in OCR, surveillance and similar things. We can imagine that most of the time, one does not have such an extensive training set as we do and that might be a problem, even though we noticed that limited training sets can give great results.

Table 1: Different seeds and including different properties give different "perfection values" for the algorithm. These values are without taking the square of the feature vectors. Notice that one value is 1, this is purely a coincidence.

Properties	**Seed 1**	**Seed 2**	**Seed 3**	**Average**
Sepal length	0.7467	0.7600	0.6800	0.7289
Sepal width	0.4800	0.6000	0.6133	0.5644
Petal length	0.9467	0.9733	0.9467	0.9556
Petal width	0.9600	0.9867	0.9333	0.9600
Sepal length Sepal width	0.7600	0.8267	0.8133	0.8000
Sepal length Petal length	0.9600	0.9600	0.9867	0.9689
Sepal length Petal width	0.9600	0.9867	0.9333	0.9600
Sepal width Petal length	0.9467	0.9733	0.9333	0.9511
Sepal width Petal width	0.9467	0.9867	0.9333	0.9556
Petal length Petal width	0.9467	0.9867	0.9333	0.9556
Sepal length Sepal width Petal length	0.9600	0.9467	0.9867	0.9645
Sepal length Sepal width Petal width	0.9467	0.9867	0.9333	0.9556
Sepal length Petal length Petal width	0.9733	0.9867	0.9467	0.9689
Sepal width Petal length Petal width	0.9333	1.0000	0.9333	0.9555
Sepal length Sepal width Petal length Petal width	0.9600	0.9867	0.9467	0.9645

A MATLAB-kod – projekt6.m

```
%%
load irisdata.mat

ca=3;
cb=2;

figure(1), clf, hold on
plot(features(class==1,ca),features(class==1,cb),'rx')
plot(features(class==2,ca),features(class==2,cb),'gx')
plot(features(class==3,ca),features(class==3,cb),'bx')
names=['Sepal length';
       'Sepal width ';
       'Petal length';
       'Petal width '];
xlabel(names(ca,:))
ylabel(names(cb,:))
legend('Setosa','Versicolor','Virginica')

%%

% How big portion of the data is used for training
% (The rest will be used for testing the model)
q_training=0.5;

% Create training and test-set.
randindex = randperm(length(class));
size_training=floor(length(class)*q_training)
index1 = randindex(1:size_training);
index2= randindex(size_training+1:end);
features_training = features(index1,:);
features_validation = features(index2,:);
class_training = class(index1);
class_validation = class(index2);

showModel(features_training,class_training,
    features_validation,class_validation,3)

%%

% Linear model based on all features
allegedclass = classify(features_validation,...
   features_training,class_training,'linear');
model1_result=sum(allegedclass==class_validation)/length
    (allegedclass)

% Linear model based on Sepal width, Petal length and
    Petal width
for n =1:15
explaining=retExplaining(n);
names(explaining,:)
allegedclass = classify(features_validation(:,explaining
    ),...
   features_training(:,explaining),class_training,'
       linear');
sum(allegedclass==class_validation)/length(allegedclass)
end

% Quadratic model  based on Sepal width, Petal length
    and Petal width
features_training2 = features_training.^2;
features_validation2 = features_validation.^2;
allegedclass = classify(features_validation2(:,2:4),...
    features_training2(:,2:4),class_training,'linear');
result2=sum(allegedclass==class_validation)/length(
    allegedclass)
```

B MATLAB-kod – showModel.m

```
function showModel(features_training,class_training,
    features_test,class_test,n)
%features=[features_training,features_test];
%class=[class_training,class_test];

ca=2;cb=3;cc=4;

figure(n),clf, hold on, grid on
plot3(features_training(class_training==1,ca),
    features_training(class_training==1,cb),
    features_training(class_training==1,cc),'r*')
plot3(features_training(class_training==2,ca),
    features_training(class_training==2,cb),
    features_training(class_training==2,cc),'g*')
plot3(features_training(class_training==3,ca),
    features_training(class_training==3,cb),
    features_training(class_training==3,cc),'b*')

plot3(features_test(class_test==1,ca),features_test(
    class_test==1,cb),features_test(class_test==1,cc),'r
    ^')
plot3(features_test(class_test==2,ca),features_test(
    class_test==2,cb),features_test(class_test==2,cc),'g
    ^')
plot3(features_test(class_test==3,ca),features_test(
    class_test==3,cb),features_test(class_test==3,cc),'b
    ^')


%xy_bounds=[min(features(:,ca)) max(features(:,ca)) min(
    features(:,cb)) max(features(:,cb))];
xy_bounds=axis
[X,Y] = meshgrid(linspace(xy_bounds(1),xy_bounds(2)),
    linspace(xy_bounds(3),xy_bounds(4)));
X = X(:); Y = Y(:);
[C,err,P,logp,coeff] = classify([X Y],features_training
    (:,[ca cb]),class_training,'linear');


K = coeff(1,2).const;
L = coeff(1,2).linear;
```

```
f1 = sprintf('0 = %g+%g*x+%g*y',K,L);
K = coeff(2,3).const;
L = coeff(2,3).linear;
f2 = sprintf('0 = %g+%g*x+%g*y',K,L);



gscatter(X,Y,C,'rgb','.',1,'off');

h2 = ezplot(f1,[xy_bounds(1),xy_bounds(2),xy_bounds(3),
    xy_bounds(4)]);
set(h2,'Color','k','LineWidth',2)
h2 = ezplot(f2,[xy_bounds(1),xy_bounds(2),xy_bounds(3),
    xy_bounds(4)]);
set(h2,'Color','k','LineWidth',2)


names=['Sepal length';
       'Sepal width ';
       'Petal length';
       'Petal width '];
xlabel(names(ca,:))
ylabel(names(cb,:))
zlabel(names(cc,:))
title('')
% legend('Setosa, training','Versicolor, training','
    Virginica, training','Setosa, validation','
    Versicolor, validation','Virginica, validation')
```