6.891 Fall 1999: Problem Set Grading Guide

Note that there may be more than one way of obtaining a reasonable solution. Also, the decision boundaries may not exactly match those of the solutions since different training regimes will result in different classifiers. However, they should be qualitatively reasonable similar. The important thing is to see that work was put into the solution and that the logic of the solution makes sense.

For each part of each problem assign one of the following grades:

- $\surd+$: got correct answer and properly justified solution (i.e., "nailed the problem and answer completely correct").
- $\surd$: showed significant work and insight but may not have completely vsolved the part, or the answer was slightly off (i.e., "got pretty much everything right except made a small error or left off some justification").
- $\surd-$: showed significant work and insight but had a some gaps in justification or answer (i.e., "had the right idea, but got lost somewhere or had a major error").
- 0 : did not show significant work or insight, or answer was not applicable to the question.

6.891 Fall 1999: Problem Set #3 Solutions

Problem 1: Questions from the DHS

A: Problem 5.1 on page 58

The derivations below are $\forall j \neq i$:

$$g_i(x_1) > g_j(x_1) \quad \Rightarrow \quad w_i^T x_1 + w_{i0} > w_j^T x_1 + w_{j0} \tag{1}$$

$$g_i(x_2) > g_j(x_2) \quad \Rightarrow \quad w_i^T x_2 + w_{i0} > w_j^T x_2 + w_{j0} \tag{2}$$

so

$$
\begin{aligned}
\lambda w_i^T x_1 + \lambda w_{i0} &> \lambda w_j^T x_1 + \lambda w_{j0} & (3)\\
(1-\lambda) w_i^T x_2 + (1-\lambda) w_{i0} &> (1-\lambda) w_j^T x_2 + (1-\lambda) w_{j0} & (4)\\
\lambda w_i^T x_1 + (1-\lambda) w_i^T x_2 + \lambda w_{i0} &> \lambda w_j^T x_1 + (1-\lambda) w_j^T x_2 + \lambda w_{j0} & (5)\\
w_i^T [\lambda x_1 + (1-\lambda) x_2] + w_{i0} &> w_j^T [\lambda x_1 + (1-\lambda) x_2] + w_{j0} & (6)
\end{aligned}
$$

so

$$
g_i[\lambda x_1 + (1-\lambda)x_2] > g_j[\lambda x_1 + (1-\lambda)x_2] \tag{7}
$$

and $\lambda x_1 + (1-\lambda)x_2 \in R_i$ showing convexity.

B: Problem 5.29 parts A and B on page 63

By inspection: $w = (-1.5, 1, 1)^T$ where $x_0 = 1$ for each data point; the support vectors are $(1,0), (0,1), (2,0), (1,1)$.

C: Problem 6.1 on page 58

Let the output of the network be $y = W_2 h$ where $h = W_1 x$ are the linear outputs of the hidden units. Then $y = W_2(W_1 x) = Wx$ which is just a linear transformation so it is equivalent to a two-layer net. Since we know that linear networks can only solve linearly separable problems, it cannot solve XOR which is not linearly separable.

D: Problem 6.6 on page 59

The suggested view that the weight change be inversely proportional to $f'(net)$ is motivated by the observation that when the network's answer isn't changing much we need to make a big weight change to move to the right answer. However by looking at $f(\cdot)$ which is a sigmoid, it's derivative $f'(\cdot)$ is small at points away from the linear part where it starts to shift from 0 to 1, but at the shifting point the derivative is large. So when our network is close to the minimum, the output will not change much, then changing the weights inversely proportional to this small number will move us away from the minimum. Likewise, when we are far away from the minimum, $f'(net)$ may be large so we want to take a big step to go towards the minimum, but moving inversely proportional to this will take too small a step. So the weight change should be proportional to $f'(net)$.

Problem 2: Getting an Accurate and Reliable Classifier

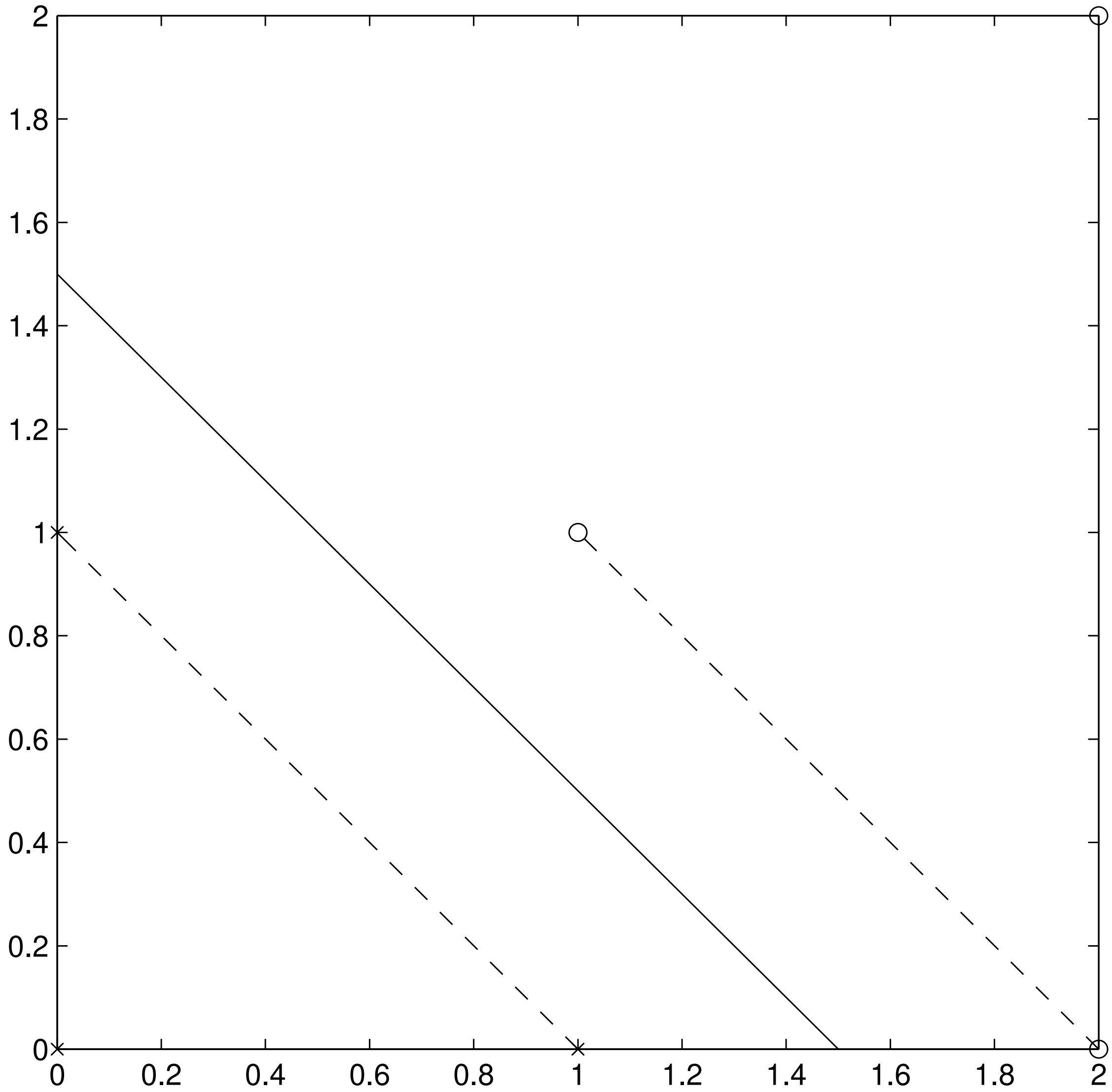

Figure 1: DHS 5.29a

A: Comment on the validity of the "hold-out" approach for estimating variance. What are the major potential pitfalls?

Since the purpose of machine learning is actually to do well on *unseen* data, minimizing the training error is not quite the right thing to do and can lead to over fitting. Using a hold-out set better approximates the performance on the unseen data. However, we now get to use less data to train (which makes the resulting classifier less accurate) and there is the potential of over fitting to the hold-out set unless we are careful about how we optimize in our cross-validation. In other words, in some sense, we have replaced one learning problem with the same overfitting problem one step removed (a meta-overfitting problem if you will).

B: What pitfalls does cross-validation address that were unaddressed by hold-out validation?

The problem of not using all of the data to train the network is slightly addressed by this problem since for the same number of total test examples, we get to train on more data. We might also hope that it would be harder to overfit to the hold-out data since we are trying different hold-out sets and averaging across them.

Problem 3: Return to the Gaussian Classifiers

A: This classifier will not work for one of these datasets. Which one? What part of the computation fails? What is the problem?

The complete Gaussian density estimator fails on the mystery dataset since the covariance matrix is singular (the data lies on a subspace of lower dimension). Looking at the data reveals that the last data entry is responsible (it is the sum of all the other columns). In Matlab, if you try `inv(sigma)`then Matlab will complain that it is singular. The code that we gave you for the Gaussian classifier doesn't complain because of the way we are computing $\sum^{-1} = \sum^{-1/2}\sum^{-1/2}$ causes Matlab to make gross approximations. If you multiply the resulting inverse with sigma you don't get anything resembling the identity matrix.

B: This problem does not arise for the spherically symmetric Gaussian classifier. Why? What is the key difference?

The covariance matrix of of spherically symmetric Gaussians is $\sum = \sigma^2 I$. So we can invert this matrix by taking $1/\sigma^2$ (unless $\sigma^2 = 0$ in which case our data would unvarying and not an interesting distribution at all).

Problem 4: Multiple Linear Discriminants

A: Compute the optimal linear discriminants for this dataset (using the squared error cost function).

We want $XW = T$ where X is the training data, W are the weights and T is the target matrix. We can do this directly with the pseudoinverse and solve it as a least squares problem. So $W = X^{\dagger}T$. See DHS 5.8 for a discussion of this.

B: Plot the decision boundaries generated by this classifier on the artificial data set.

Problem 5: Perceptrons

A: First write code which implements the Perceptron learning rule for two classes. Construct a Perceptron which separates the first and second classes of the data (don't worry about the other class right now).

See attached code.

B: Plot the decision boundaries generated by this classifier on the artificial data set.

C: When there are multiple classes, a single perceptron won't do. Like linear discriminants, Perceptrons can be generalized to perform multi-way classification. Note: this can be a bit tricky. One way is to train one classifier for each class so that it returns 1 for elements in the class and a 0 for other classes. Such a scheme can definitely lead to ambiguities (see DHS Figure 5.3). Propose a scheme to disambiguate these outputs.

I have disambiguated the output by choosing the maximum activation energy. Another method might be to choose the perceptron which classifies the example as positive and for which the example lies farthest from the classifier (if no classifier returns 1, then pick the classifier to which the point lies the closest).

D: Using this scheme, write code that trains a group of Perceptrons to perform multi-way classification.

See attached code.

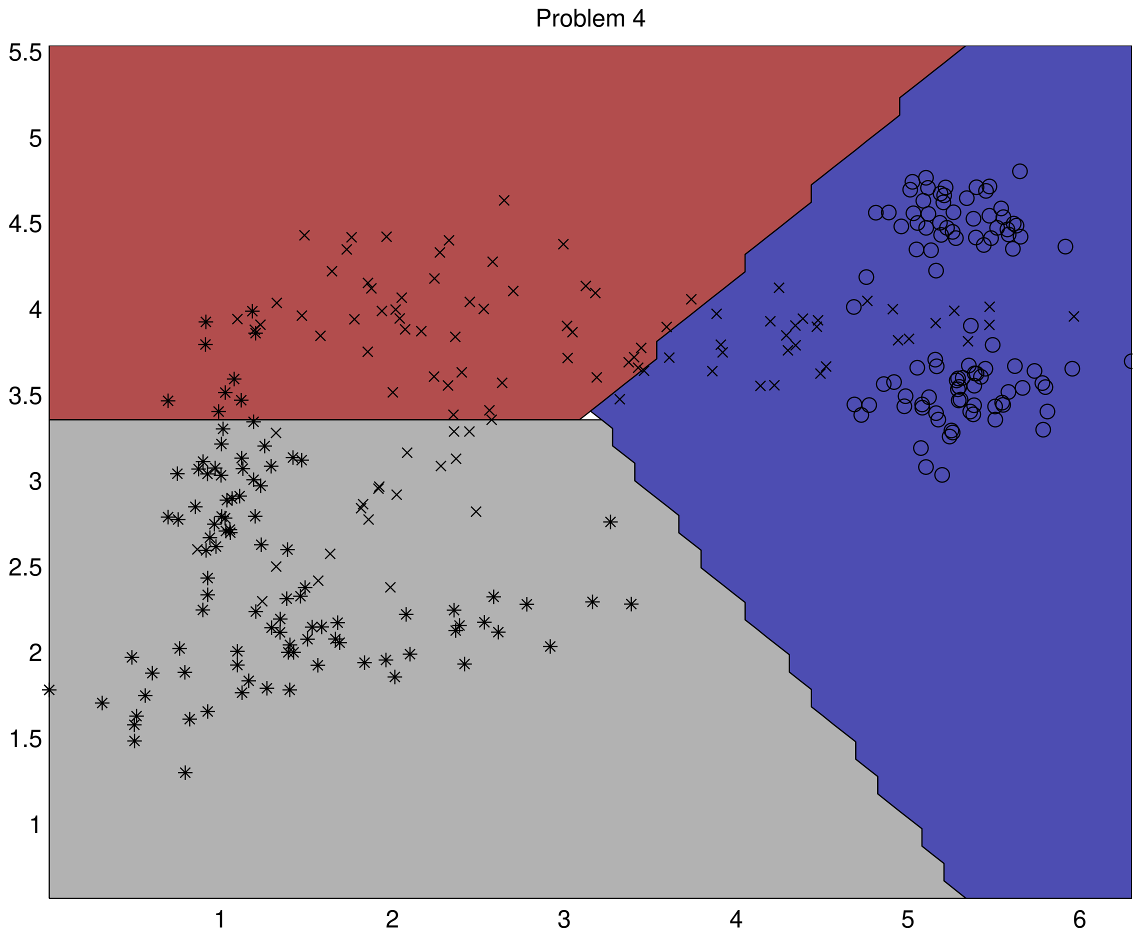

Figure 2: Linear Discriminant Classifier

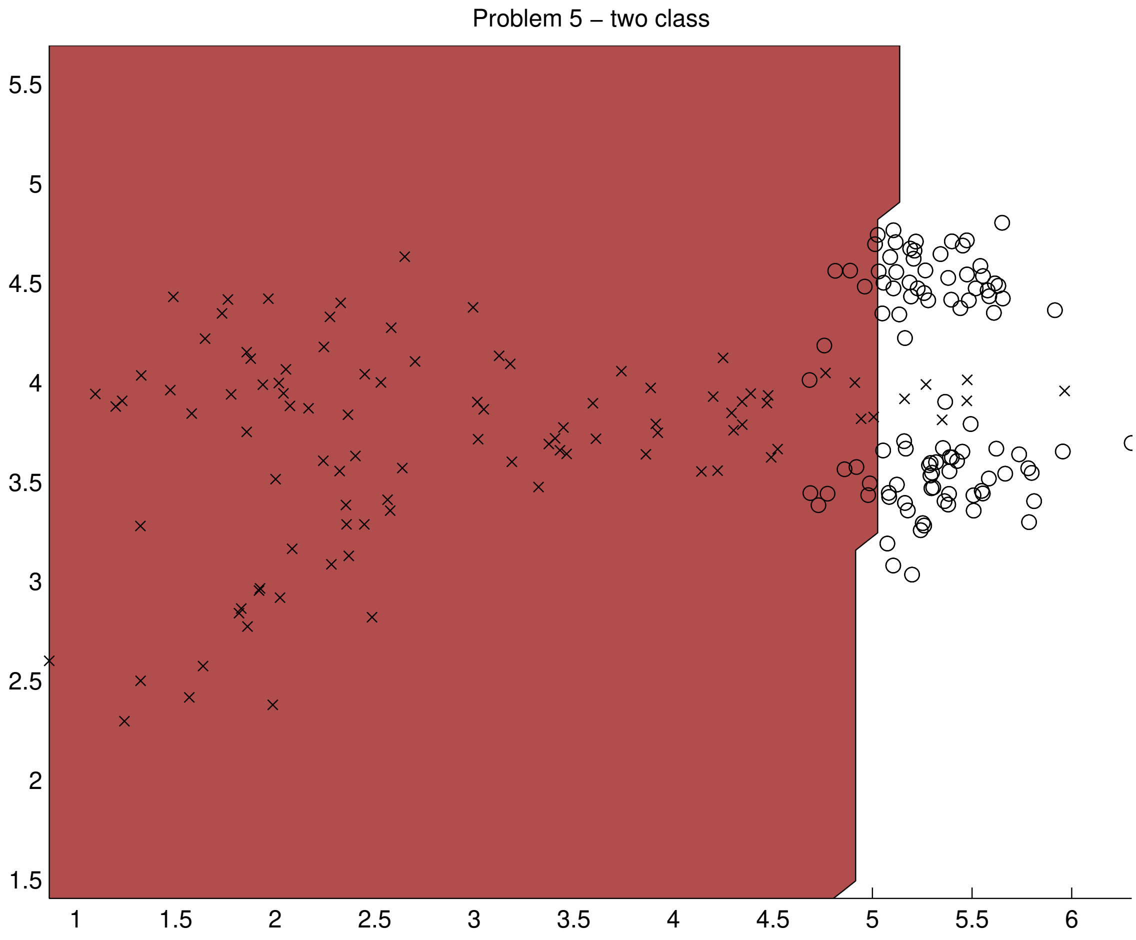

Figure 3: Perceptron Classifier – just classes 1 and 2

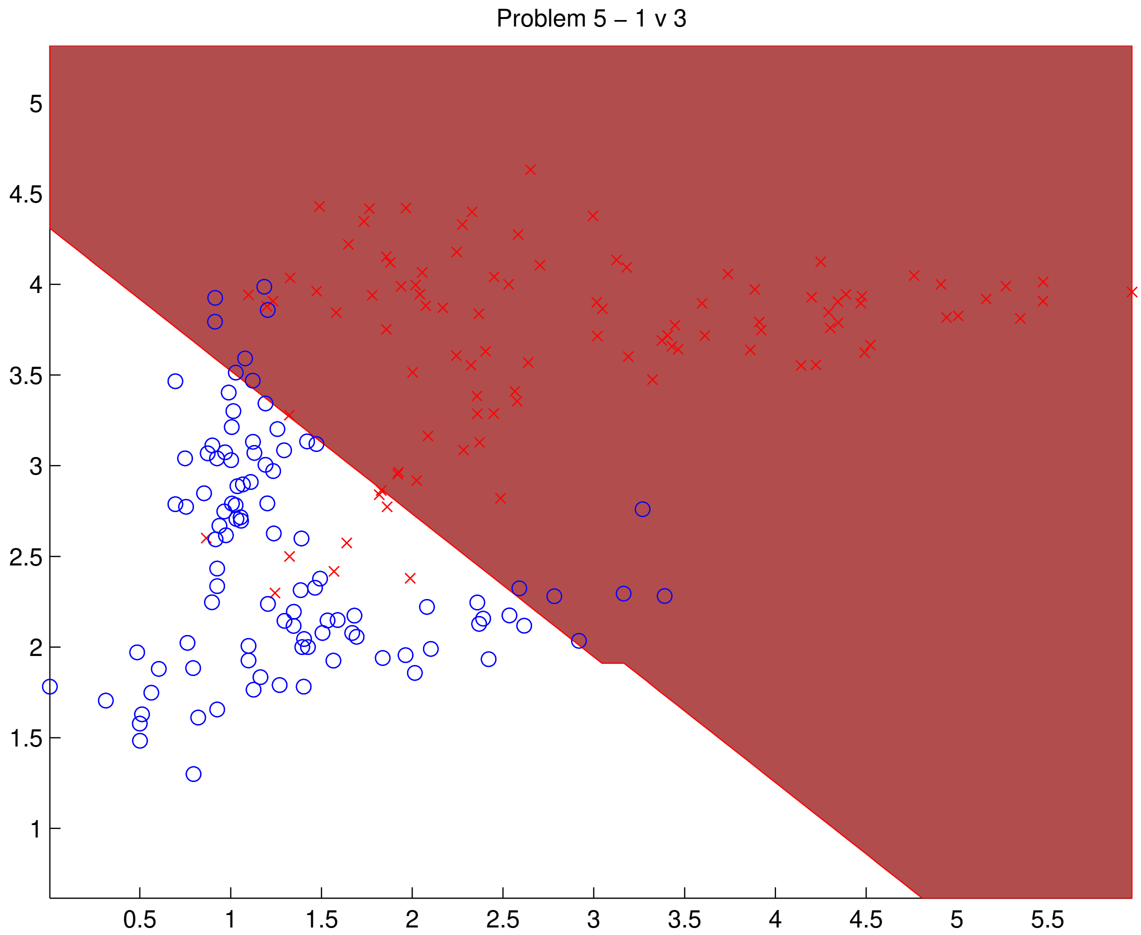

Figure 4: Perceptron Classifier - 1 v 3

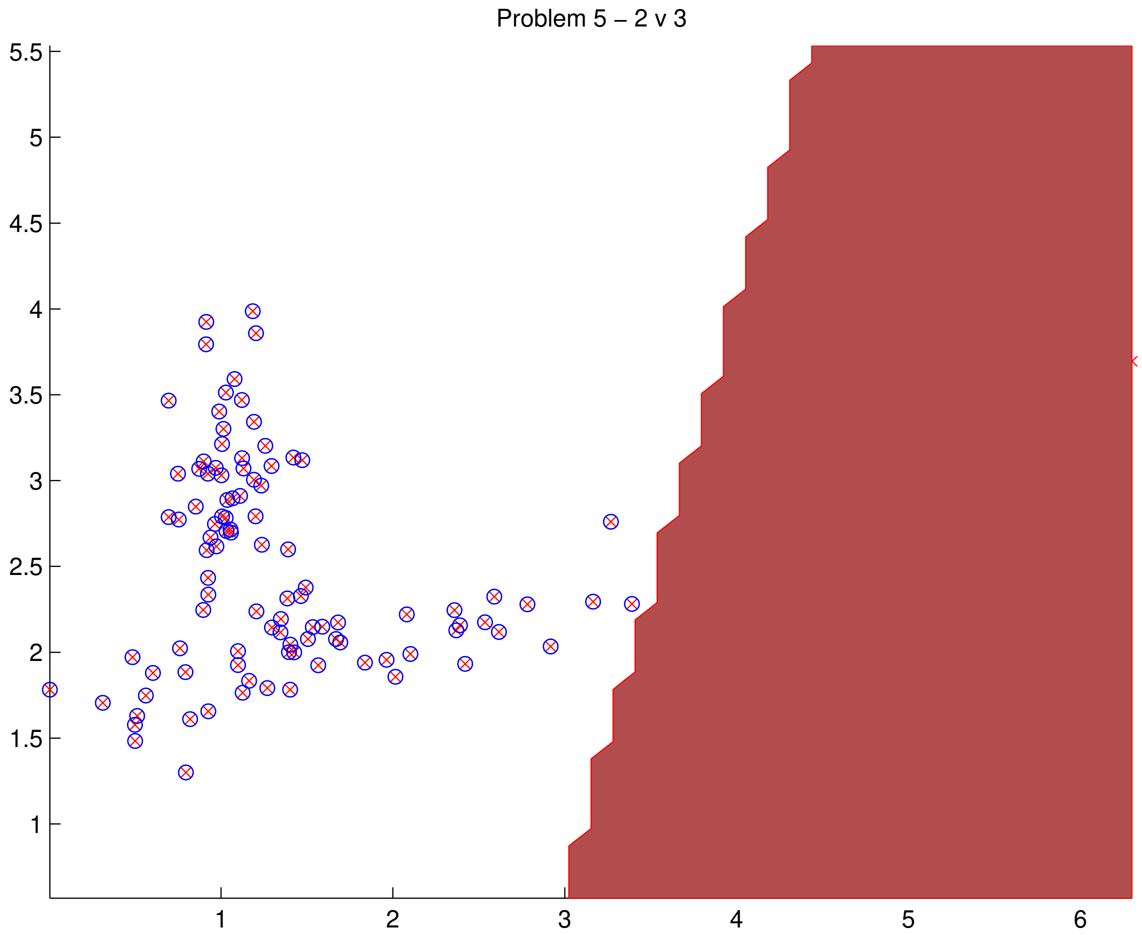

Figure 5: Perceptron Classifier - 2 v 3

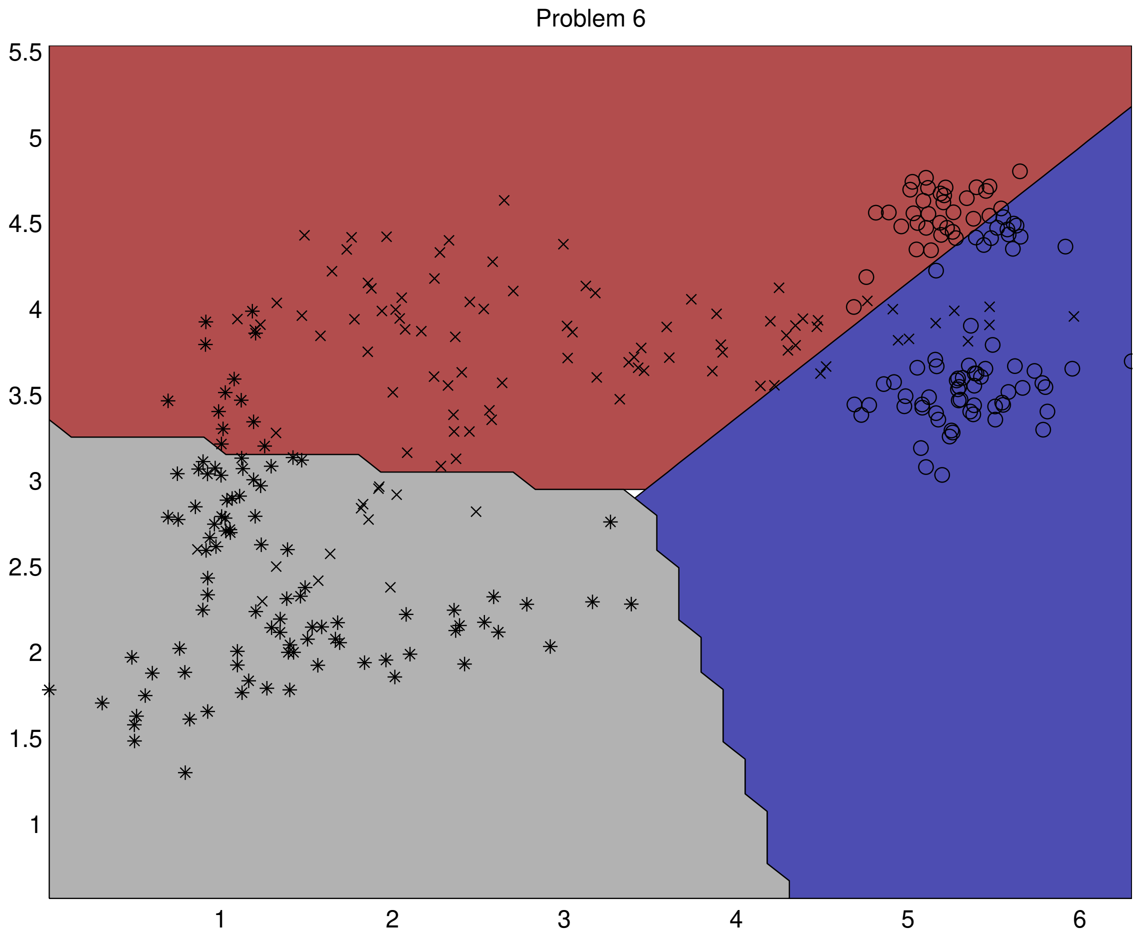

Figure 6: Perceptron Classifier

E: Plot the decision boundary for class 1 vs. class 2, class 1 vs. class 3, and class 2 vs. class 3 (three graphs total). Generate an additional graph that shows the three decision regions for the multi-class perceptron classifier.

See part (A) for first plot.

Problem 6: Generalized Linear Techniques

A: Propose and then add at least two additional non-linear features to the data. Describe these new features and then add them to `input`.

See attached code.

B: Compute the optimal linear discriminants for this data and plot the decision boundaries.

C: Train a multi-class perceptron for this data and plot the decision boundaries.

Problem 7: Multi-Layer Perceptron

A: Implement a code which can evaluate a MLP given a set of weights. Assume that the internal and hidden units of this network are sigmoids.

See attached code.

B: Implement code which trains a MLP to minimizes the squared error of the network. (Hint: Getting the matrix representations for the weights right will make the code pretty easy to write.)

See attached code.

C: Plot the decision boundaries generated by this classifier on the artificial data set.

Problem 8: Multi-Layer Perceptron Optimization

A: Describe each topology, its average error, and the variance of the error.

See attached code.

B: Plot the decision boundary generated by the best cross validated classifier.

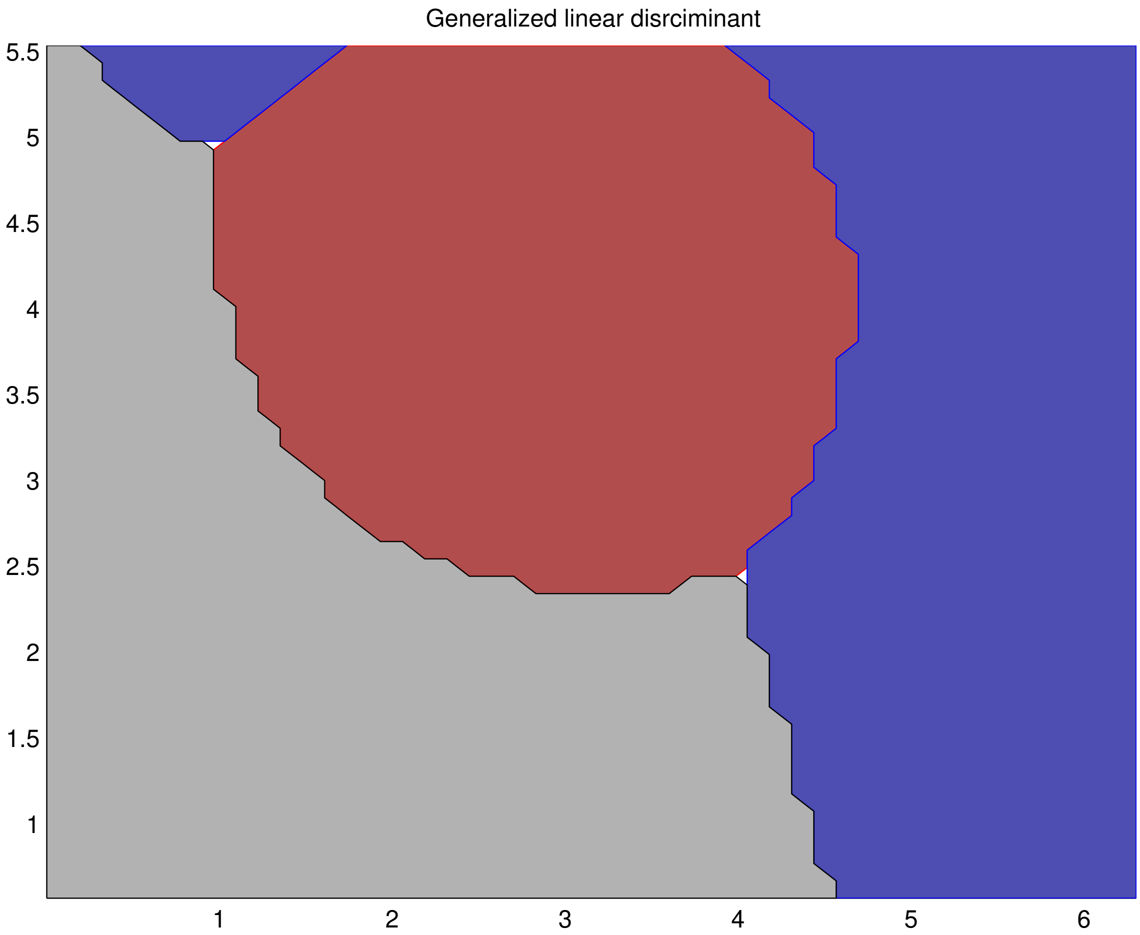

Figure 7: Generalized Linear Classifier

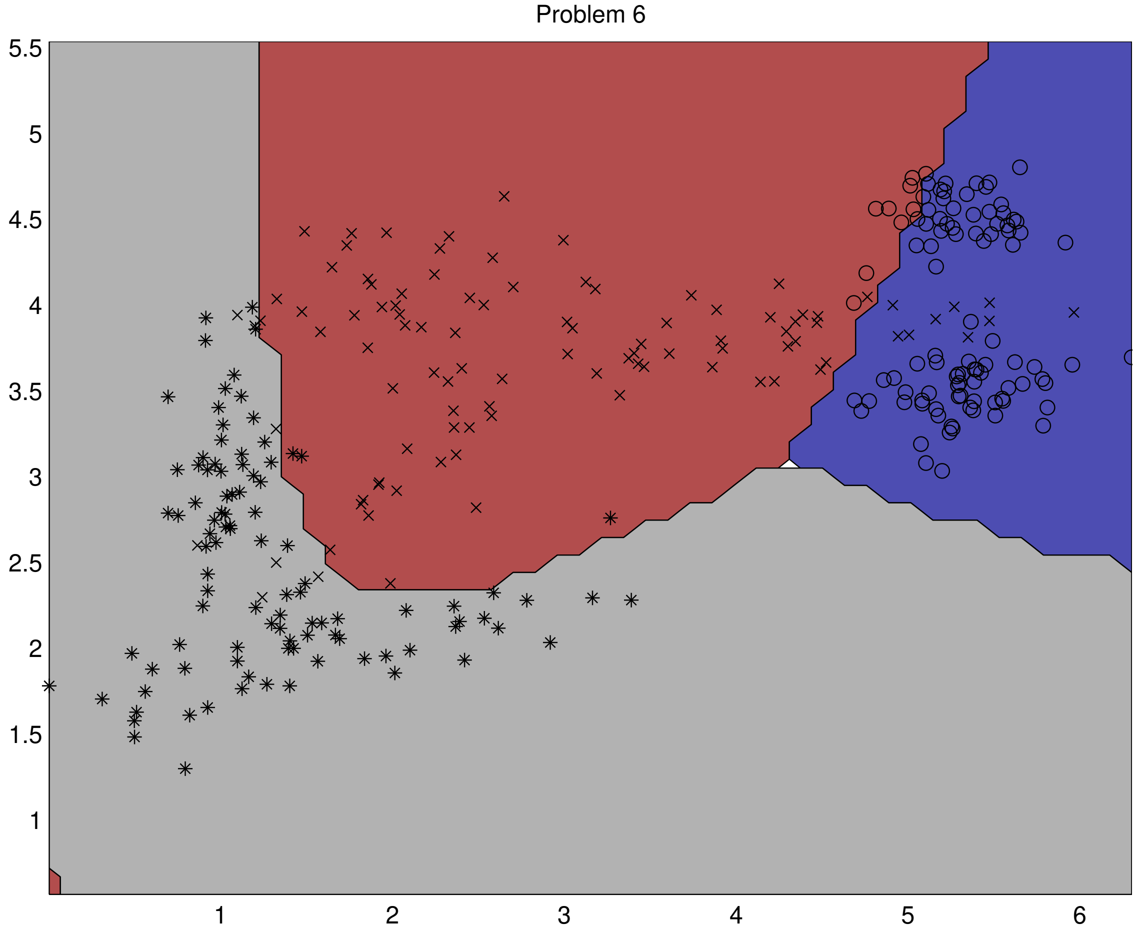

Figure 8: Generalized Perceptron Classifier

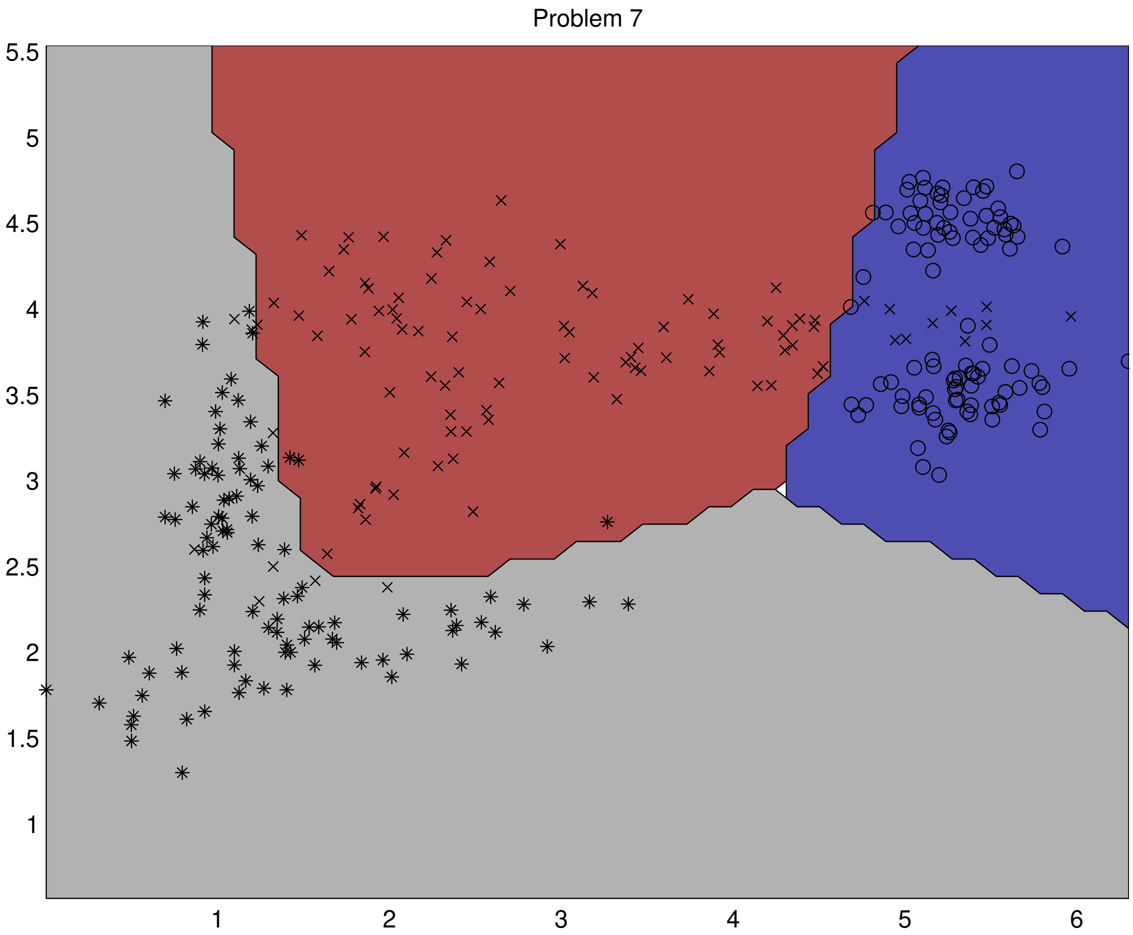

Figure 9: Multi-layered Perceptron with fixed 2-4-3 topology

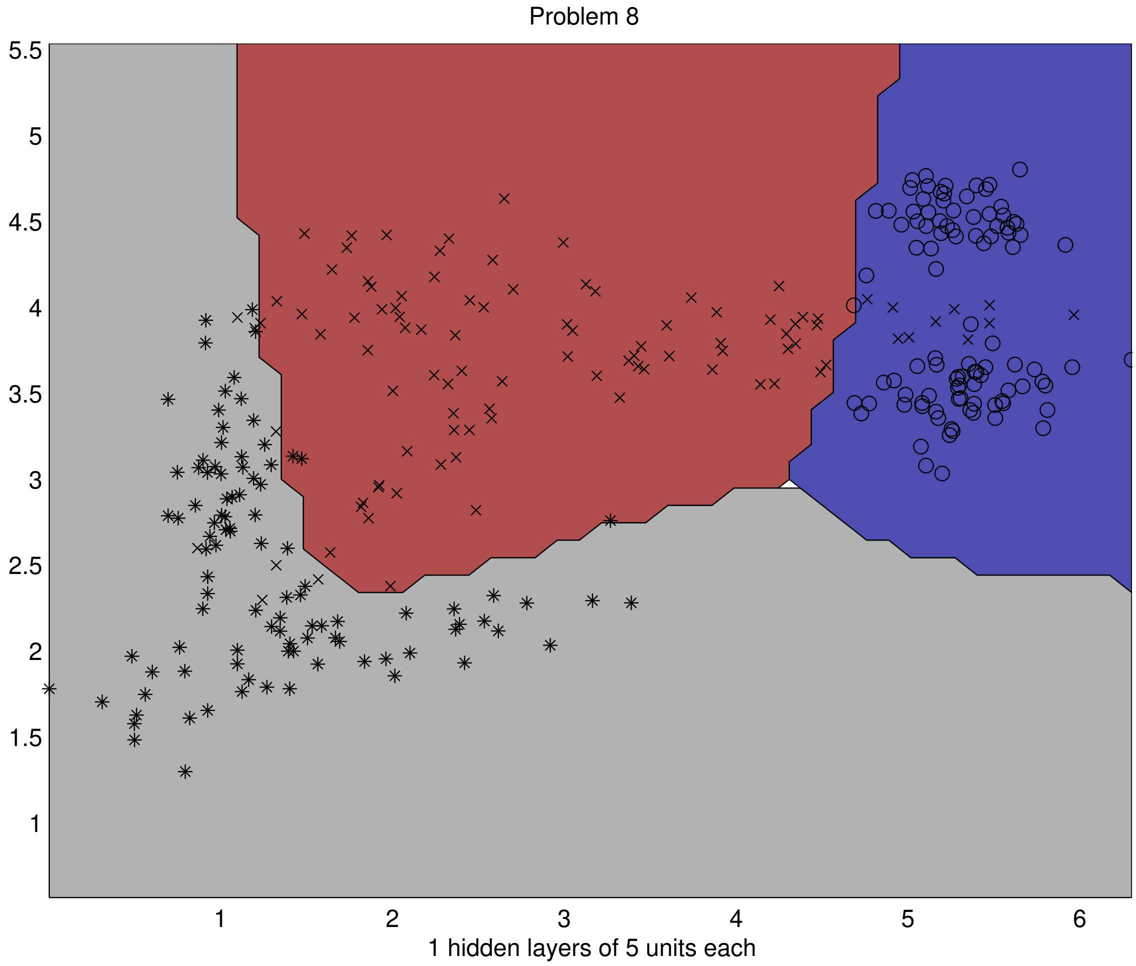

Figure 10: Multi-layered Perceptron with cross-validated topology