MATLAB Companion Script for *Machine Learning* ex3 (Optional)

Introduction

Coursera's *Machine Learning* was designed to provide you with a greater understanding of machine learning algorithms- what they are, how they work, and where to apply them. You are also shown techniques to improve their performance and to address common issues. As is mentioned in the course, there are many tools available that allow you to use machine learning algorithms *without* having to implement them yourself. This Live Script was created by MathWorks to help *Machine Learning* students explore the data analysis and machine learning tools available in MATLAB.

FAQ

Who is this intended for?

- This script is intended for students using MATLAB Online who have completed ex3 and want to learn more about the corresponding machine learning tools in MATLAB.

How do I use this script?

- In the sections that follow, read the information provided about the data analysis and machine learning tools in MATLAB, then run the code in each section and examine the results. You may also be presented with instructions for using a MATLAB machine learning app. This script should be located in the ex3 folder which should be set as your Current Folder in MATLAB Online.

Can I use the tools in this companion script to complete the programming exercises?

- No. Most algorithm steps implemented in the programming exercises are handled automatically by MATLAB machine learning functions. Additionally, the results will be similar, but not identical, to those in the programming exercises due to differences in implementation, parameter settings, and randomization.

Where can I obtain help with this script or report issues?

- As this script is not part of the original course materials, please direct any questions, comments, or issues to the *MATLAB Help* discussion forum.

Multi-Class Logistic Regression and Neural Network Prediction

In the first part of this Live Script, a multi-class (one-vs-all) logistic regression model is implemented using the `fitcecoc` function from the Statistics and Machine Learning Toolbox. In the second part, a pre-trained neural network created using tools from the Deep Learning Toolbox is used to classify handwritten digits.

Files needed for this script

- `ex3data1.mat` - Training set of hand-written digits
- `ex3_MATLAB.mat` - Data file containing a trained neural network variable

Table of Contents

Multi-Class Classification Using Logistic Regression

The following sections will guide you through multi-class classification in MATLAB using the `fitcecoc` function. An ECOC ('error correcting output codes') model in MATLAB consists of a single model variable which contains the individual one-vs-all models for a multiclass classification problem. In practice, this means that there is no need to train and compare the predictions of individual classifiers.

Load the data

Recall that `ex3data1.mat` contains a matrix `X` of 5000 training examples of handwritten digit images and a vector `y` containing the corresponding digit labels. The digit images consist of grayscale pixel intensities from 0 to 1 and the 20x20 pixel images which have been 'unwound' into 1x400 row vectors and stacked into the matrix `X`. The class labels range from 1 to 10 (with the 10 value corresponding to a '0' digit). Run the code in this section to load the data.

```
clear;
% Load saved matrices X and y from file
load('ex3data1.mat');
```

Train a one-vs-all multi-class logistic classifier using `fitcecoc`

In ex3 you implemented a one-vs-all model by training 10 separate classifiers. In this section we will use the `fitcecoc` function to train a single model containing the 10 classifiers *simultaneously*. First, we'll create a *classifier model template* variable which is used to store options for training multi-class models, such as the model type and regularization options. Additional options include the solver configuration settings- see the documentation for more information about the available options. After creating the template using the `templateLinear` function, we call `fitcecoc` to create and train the multi-class classifier model. The `fitcecoc` function will automatically detect the 10 classes (1 - 10) from the response vector variable (y) and add a bias term to the feature data before training the classifiers. The result is a `ClassificationECOC` model variable which contains the 10 classifier models and subsequently the information about each model.

```
opts = templateLinear('Regularization','ridge','Learner','logistic','Lambda',0.001,'Pas
multiLogMdl = fitcecoc(X,y,'Learners',opts)
```

```
multiLogMdl =
  CompactClassificationECOC
      ResponseName: 'Y'
        ClassNames: [1 2 3 4 5 6 7 8 9 10]
    ScoreTransform: 'none'
    BinaryLearners: {45×1 cell}
      CodingMatrix: [10×45 double]


  Properties, Methods
```

Compute the training accuracy

Run the code below to compute the training accuracy. We obtain the predicted class (class with the maximum probability) in the usual manner by calling the `predict` function and providing the model variable and feature data as inputs. The `predict` function will then return the predicted class for each example.

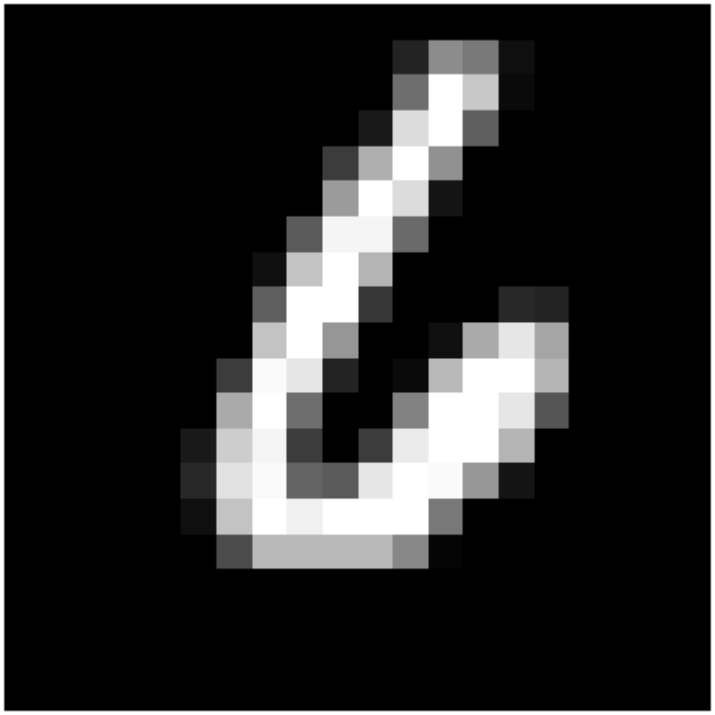
```
fprintf('Training accuracy: %g%%',100*sum(y == predict(multiLogMdl,X))/length(y))
```

```
Training accuracy: 93.26%
```

Predicting the class of a random training example

The code below will estimate the class of a random training example, compare with the correct class, and display the corresponding image. We obtain a vector of probabilities predicted for each class by requesting an additional output from `predict`. Rerun this section multiple times to repeat with different examples.

```
i = randi(length(y));
[class,~,~,prob] = predict(multiLogMdl,X(i,:));
imshow(reshape(X(i,:),20,20))
```



```
fprintf('True class: %d  |  Predicted class: %d | Probability of match: %.1f%%',y(i),cl
```

```
True class: 6  |  Predicted class: 6 | Probability of match: 93.3%
```

Digit Prediction Using a Trained Neural Network

In the second part of ex3 you used a pre-trained, feed-forward neural network to perform digit classification, where the weights were provided as two matrices. As we have seen preciously when using MATLAB machine learning functions and apps in companion scripts, model variables returned from functions like `fitlm`, `fitglm`, `fitclinear`, and `fitcecoc` contain both model and performance information in addition to the fitted model parameters. They can then be used with the `predict` function to predict values or classes of new or existing examples. The same is true with neural network model variables *except* the model variables themselves can be used for prediction.

In this section we load a pre-trained neural network model variable, explore its properties, and use it to predict digit values for image samples. In the companion script for ex4, we will actually create and train a neural network using tools and apps in the Deep Learning Toolbox.

Load the data and neural network variable

Run the code below to re-load the image data and labels as well as the trained neural network variable `net`.

```
clear;
load('ex3data1.mat');
load('ex3_companion.mat');
```

Explore the model variable

A trained MATLAB neural network model contains all of the information about the network, including the layer structure and weights. Run this section to display and explore some of the properties of the network variable, `net`. In particular, the network architecture is visualized using the `view` function, while the hidden and output layer weights and bias values are extracted from the `IW`, `LW`, and `b` properties in the `net` variable.

```
net % List the network variable properties
```

```
net =

    Neural Network

              name: 'Pattern Recognition Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 10160
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'crossentropy'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotconfusion', 'plotroc'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainscg'
```

```
    trainParam: .showWindow, .showCommandLine, .show, .epochs,
                .time, .goal, .min_grad, .max_fail, .sigma,
                .lambda

  weight and bias values:

              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs
```

```
view(net) % Visualize the network
net.IW{1} % View the hidden layer weights
```

```
ans = 25×395
   -0.0046    0.0042   -0.0015   -0.0043   -0.0043   -0.0032   -0.0027   -0.0033 ···
   -0.0250    0.0206    0.0009   -0.0204   -0.0231   -0.0194   -0.0169   -0.0151
   -0.0040    0.0029    0.0011   -0.0036   -0.0057   -0.0039   -0.0038   -0.0057
   -0.0038    0.0037   -0.0037   -0.0056   -0.0059   -0.0038   -0.0037   -0.0040
    0.0252   -0.0207    0.0013    0.0232    0.0255    0.0189    0.0175    0.0192
   -0.0249    0.0216   -0.0034   -0.0217   -0.0195   -0.0172   -0.0161   -0.0180
    0.0112   -0.0098    0.0014    0.0097    0.0102    0.0089    0.0077    0.0085
   -0.0116    0.0095    0.0017   -0.0073   -0.0085   -0.0084   -0.0057   -0.0081
    0.0128   -0.0125    0.0084    0.0136    0.0105    0.0068    0.0065    0.0081
    0.0152   -0.0138    0.0058    0.0135    0.0103    0.0099    0.0087    0.0095
     ⋮
```

Note: The neural network training process is optimized in MATLAB so that any variables which do not contribute training information are ignored. In the current dataset, 5 elements (pixels) of each training sample were found to have a constant value across all samples during training, and so these features were ignored. Therefore, the model input layer is size 25 x 395, as shown in the output above. As we will see in the next section, the trained network still expects inputs with 400 elements when predicting (as is indicated in the visual diagram of the model. The redundant pixels are automatically disregarded.

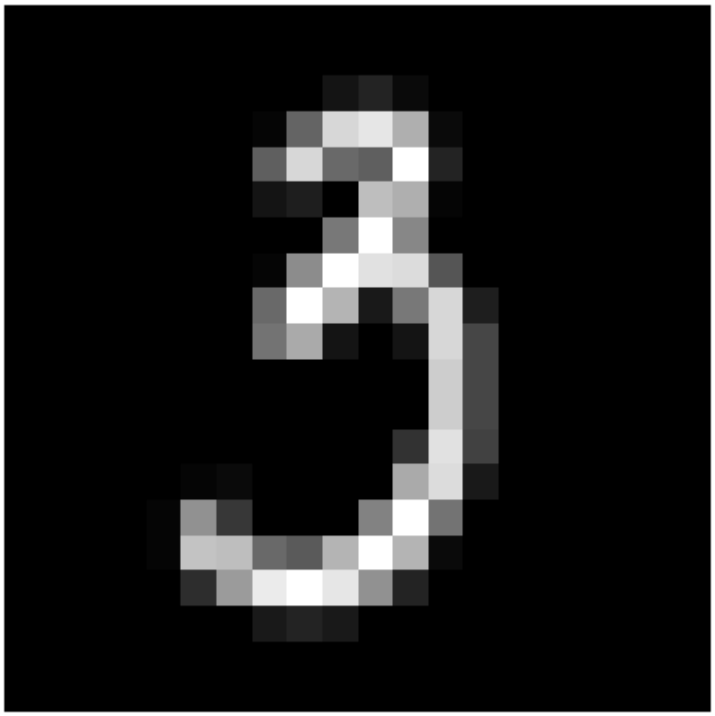
Predict the data

The analog to the `predict` function for linear classification and regression model variables is the `sim` function for neural network variables. However, the network variable can be used *directly* for prediction, in this case to predict the probabilities of a match for each digit, by inputting one or more images as input- see the code below. Run the code in this section to estimate the class of a random example, compare with the true class, and display the corresponding image using the trained network.

Note:

- When training or simulating a neural network in MATLAB, the preferred data organization is with observations *in columns,* and classes *in rows*. Therefore the image row vectors should be transposed before using them to train or predict as in the code below.
- The output of the prediction will consist of a 10 X *N* array of probabilities, where *N* is the number of images to be classified. The `max` function can then be used to find the maximum probability for each example and thus the predicted class.

```
i = randi(length(y));
ysim = net(X(i,:)');
[~,class] = max(ysim);
imshow(reshape(X(i,:),20,20))
```



```
fprintf('True class: %d  |  Predicted class: %d | Probability of match: %.1f%%',y(i),cl
```

```
True class: 3  |  Predicted class: 3 | Probability of match: 57.1%
```

Compute the training accuracy

The neural network model was trained using regularization with the same data provided in `ex3data1.mat`. Run the code below to compute the training accuracy of the model.

```
[~,ysim] = max(net(X'));
fprintf('Training accuracy: %g%%',100*sum(y == ysim')/length(y));
```

```
Training accuracy: 94.68%
```