MATLAB Companion Script for *Machine Learning* ex1 (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 ex1 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 ex1 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.

Linear Regression

In this script, linear regression models for both single and multiple variables are implemented using the `fitlm` function from the Statistics and Machine Learning Toolbox. A quick tutorial is also included on using the *Regression Learner App,* which provides a graphical interface for creating regression models.

Files needed for this script

- ex1data1.txt - Dataset for linear regression with one variable
- ex1data2.txt - Dataset for linear regression with multiple variables

Table of Contents

Linear Regression with One Variable

This section covers the MATLAB implementation of single variable linear regression corresponding to the first part of ex1. Recall that the file ex1data1.txt contains two columns of data: the first column corresponds to the populations of cities and the second column contains the profit of food trucks in those cities.

Load the data into a `table`

The `table` datatype is now the preferred datatype for most data analysis and machine learning tasks in MATLAB. In this script we will use tables instead of vectors and matrices. A `table` consists of rows and columns where each column corresponds to a variable, and each row corresponds to an observation.

Run the code below to:

- Read the profit data into a `table` using the `readtable` function
- Add descriptive names to the `table` variables
- Compute summary statistics on each variable
- Display the methods available for working with `table` variables.

```
clear;
data = readtable('ex1data1.txt'); % read tabular data into a table
data.Properties.VariableNames = {'Population','Profit'} % name the variables
summary(data)
methods(data)
```

Use the `help` control below to view descriptions of the functions for working with `table` variables diplayed above.

```
help table.outerjoin
```

```
 outerjoin Outer join between two tables or two timetables.
    T = outerjoin(TLEFT, TRIGHT) creates the table T as the outer join
    between the tables TLEFT and TRIGHT.  If TLEFT is a timetable, then
    TRIGHT can be either a table or a timetable, and outerjoin returns T as
    a timetable for either combination of inputs.  An outer join includes
    the rows that match between TLEFT and TRIGHT, and also unmatched rows
    from either TLEFT or TRIGHT.
 
    outerjoin first finds one or more key variables.  If both TLEFT and TRIGHT
    are tables, or if TLEFT is a timetable and TRIGHT is a table, the default
    keys are the variables that occur in both TLEFT and TRIGHT with the same
    names.  If both TLEFT and TRIGHT are timetables, the default keys are the row
    time vectors of TLEFT and TRIGHT.  outerjoin then uses those key variables
    to match up rows between TLEFT and TRIGHT.  T contains one row for each
    pair of rows in TLEFT and TRIGHT that share the same combination of key
    values. In general, if there are M rows in TLEFT and N rows in TRIGHT that
    all contain the same combination of key values, T contains M*N rows for
    that combination.  T also contains rows corresponding to key combinations
    in TLEFT (or TRIGHT) that did not match any row in TRIGHT (or TLEFT).
    outerjoin sorts the rows in the result T by the key values.
 
    T contains all variables from both TLEFT and TRIGHT, including the
    keys.  If TLEFT and TRIGHT contain variables with identical names,
    outerjoin adds a unique suffix to the corresponding variable names in
    T.  Variables in T that came from TLEFT (or TRIGHT) contain null values
    in those rows that had no match from TRIGHT (or TLEFT).
 
    T = outerjoin(TLEFT, TRIGHT, 'PARAM1',val1, 'PARAM2',val2, ...) allows
    you to specify optional parameter name/value pairs to control how
    outerjoin uses the variables in TLEFT and TRIGHT.  Parameters are:
 
            'Keys'       - specifies the variables to use as keys.
            'LeftKeys'   - specifies the variables to use as keys in TLEFT.
            'RightKeys' - specifies the variables to use as keys in TRIGHT.
 
    You may provide either the 'Keys' parameter, or both the 'LeftKeys' and
    'RightKeys' parameters.  The value for these parameters is a positive
    integer, a vector of positive integers, a variable name, a cell array
    of character vectors or string array containing one or more variable
    names, or a logical vector.  'LeftKeys' or 'RightKeys' must both
    specify the same number of key variables, and the left and right keys
    are paired in the order specified.  If TLEFT and TRIGHT are tables,
    then you may specify their row names as keys.  If they are timetables,
    then you may specify their row times as keys.
 
            'MergeKeys' - specifies if outerjoin should include a single variable
                          in T for each key variable pair from TLEFT and
                          TRIGHT, rather than including two separate
                          variables.  outerjoin creates the single variable
                          by merging the key values from TLEFT and TRIGHT,
                          taking values from TLEFT where a corresponding
                          row exists in TLEFT, and from TRIGHT otherwise.
                          Default is false.  If both TLEFT and TRIGHT are
                          timetables, 'MergeKeys' must always be true.
       'LeftVariables'  - specifies which variables from TLEFT to include in T.
                          By default, outerjoin includes all variables from TLEFT.
       'RightVariables' - specifies which variables from TRIGHT to include in T.
                          By default, outerjoin includes all variables from TRIGHT.
 
    'LeftVariables' or 'RightVariables' can be used to include or exclude
    key variables as well as data variables.  The value for these
    parameters is a positive integer, a vector of positive integers, a
```

```
    variable name, a cell array of character vectors or string array
    containing one or more variable names, or a logical vector.

                 'Type' - specifies the type of outer join operation, either
                          'full', 'left', or 'right'.  For a left (or
                          right) outer join, T contains rows corresponding
                          to keys in TLEFT (or TRIGHT) that did not match
                          any in TRIGHT (or TLEFT), but not vice-versa.  By
                          default, outerjoin does a full outer join, and
                          includes unmatched rows from both TLEFT and
                          TRIGHT.

    [T,ILEFT,IRIGHT] = outerjoin(TLEFT, TRIGHT, ...) returns index vectors
    ILEFT and IRIGHT indicating the correspondence between rows in T and
    those in TLEFT and TRIGHT.  outerjoin constructs T by horizontally
    concatenating TLEFT(ILEFT,LEFTVARS) and TRIGHT(IRIGHT,RIGHTVARS). ILEFT
    or IRIGHT may also contain zeros, indicating the rows in T that do not
    correspond to rows in TLEFT or TRIGHT, respectively.

    Examples:

      % Create two tables that both contain the key variable 'Key1'.  The
      % two arrays contain rows with common values of Key1, but each array
      % also contains rows with values of Key1 not present in the other.
      Tleft = table({'a' 'b' 'c' 'e' 'h'}',[1 2 3 11 17]','VariableNames',{'Key1' 'Var1'})
      Tright = table({'a' 'b' 'd' 'e'}',[4 5 6 7]','VariableNames',{'Key1' 'Var2'})

      % Combine Tleft and Tright with an outer join.  This matches up rows with
      % common key values, but also retains rows whose key values don't have
      % a match.  Keep the key values as separate variables in the result.
      Tfull = outerjoin(Tleft,Tright,'key','Key1')

      % Join Tleft and Tright, merging the key values as a single variable in the result.
      Tfullmerge = outerjoin(Tleft,Tright,'key','Key1','MergeKeys',true)

      % Join Tleft and Tright, ignoring rows in Tright whose key values do not match any
      % rows in Tleft.
      Tleftmerge = outerjoin(Tleft,Tright,'key','Key1','Type','left','MergeKeys',true)


      % Create two timetables Tleft and Tright. The time vector of each timetable
      % contains some overlapping times, but also includes times that are not
      % present in the other timetable.
      Tleft = timetable(seconds([1;2;4;6]),[1 2 3 11]')
      Tright = timetable(seconds([2;4;6;7]),[4 5 6 7]')

      % Combine Tleft and Tright with an outer join.  This matches up rows with
      % common times, but also retains rows whose times don't have
      % a match.
      Tfull = outerjoin(Tleft,Tright)

      % Join Tleft and Tright, ignoring rows in Tright whose key values do not match any
      % rows in Tleft.
      Tleftjoin = outerjoin(Tleft,Tright,'Type','left')

    See also innerjoin, join, horzcat, sortrows,
             union, intersect, ismember, unique, inner2outer, rows2vars.

Help for table/outerjoin is inherited from superclass tabular

    Documentation for table/outerjoin
```

Fit a linear model using the `fitlm` function

There are many functions available in the Statistics and Machine Learning Toolbox for fitting a model to data. To fit a linear regression model to the data in our `table`, we'll use the `fitlm` function. As we will see, there is no need to add a column of ones to the data for a bias term, to create a separate cost function, or to implement gradient descent to converge on the model parameters as in ex1- these tasks are automatically taken care of by `fitlm`. The output of `fitlm` is a `LinearModel` variable which contains all of the information about the model.

Run the code in this section to fit the linear model. The model coefficients (θ) are extracted from the model variable and printed out for you below- compare to your results from ex1. After running, double-click on the variable `linMdl` in the MATLAB workspace to examine its properties further. Variable properties can also be extracted and displayed using the '**.**' operator, as in the code below used to extract the `Coefficients` property. The result is a `table` of the model coefficients and additional statistical information.

```
linMdl = fitlm(data);
linMdl.Coefficients
```

ans = 2×4 table

	Estimate	SE	tStat	pValue
1 (Intercept)	-3.8958	0.7195	-5.4147	4.6079e-07
2 Population	1.1930	0.0797	14.9608	1.0232e-26

```
theta = linMdl.Coefficients.Estimate;
fprintf('Theta computed by fitlm: [%f; %f]',theta(1),theta(2))
```

```
Theta computed by fitlm: [-3.895781; 1.193034]
```

Making predictions using the `LinearModel` variable

Below we use the `predict` function to predict profit for different populations using the model trained in the previous section. Note that the first input to `predict` is a trained model variable, while the second input is a feature value or list of values in the form of a vector, matrix or table. Run the code in this section and compare with your results from ex1.

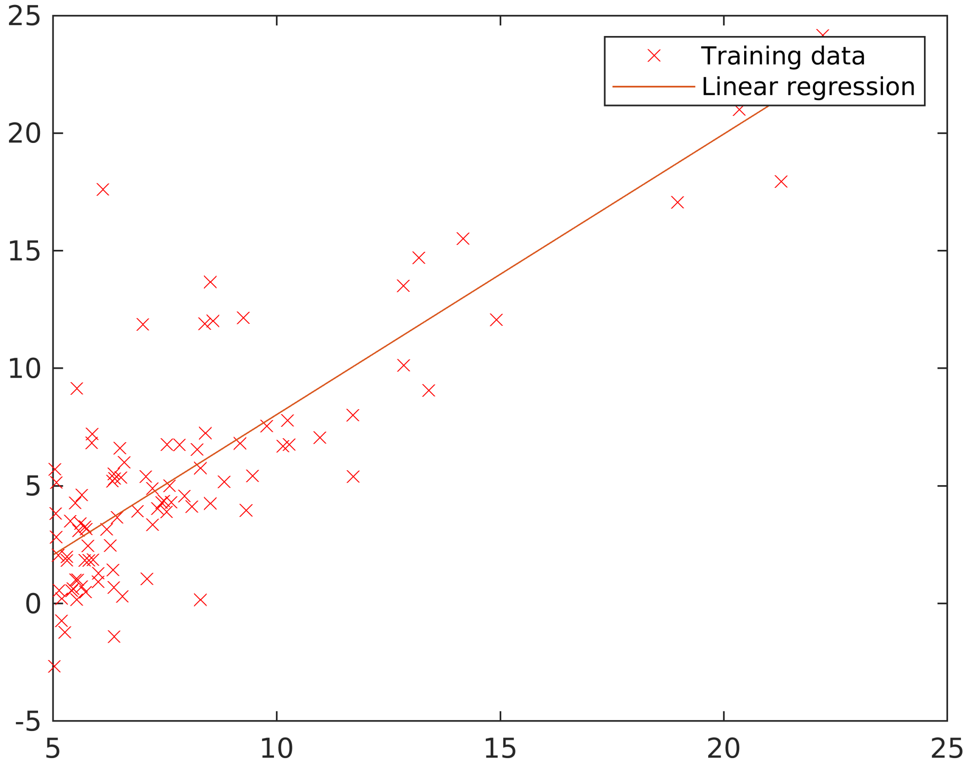
```
% Predict values for population sizes of 35,000 and 70,000
predict1 = predict(linMdl,3.5);
fprintf('For population = 35,000, we predict a profit of %f', predict1*10000);
```

```
For population = 35,000, we predict a profit of 2798.368764
```

```
predict2 = predict(linMdl,7);
fprintf('For population = 70,000, we predict a profit of %f', predict2*10000);
```

```
For population = 70,000, we predict a profit of 44554.546310
```

```
% Plot the linear fit
plot(data.Population,data.Profit,'rx'); hold on
profit = predict(linMdl,data);
plot(data.Population, profit, '-')
legend('Training data', 'Linear regression'); hold off
```



Linear Regression with Multiple Variables

Load and preview the data

Recall that the file ex1data2.txt contains a training set of housing prices in Portland, Oregon. Each row corresponds to a house where the element in the first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house. Run this section to load the data into a `table`. After a `table` is displayed as output you can perform basic sort and filter operations on a given columns as follows:

1. Hover the mouse pointer over the desired variable name (column heading)
2. Click on the down arrow when it appears
3. Choose the desired sort option to sort all rows in the table based on the value in that variable OR
4. Enter a specific range in the boxes provided to select only rows whose value for the selected variable falls inside that range
5. The MATLAB code required to perform 3 and/or 4 is automatically displayed below the `table`. If desired, the code can be copied to the clipboard or added to the script using the 'Copy' and 'Update' code buttons.

After running this section, experiment with sorting and filtering the data `table` below. (Since we want to use all the observations in `data`, there is no need to copy the code or add it to the script.)

```
clear;
% Load Data
```

```
data = readtable('ex1data2.txt');
data.Properties.VariableNames = {'Size', 'Bedrooms', 'Price'}
```

data = 47×3 table

	Size	Bedrooms	Price
1	2104	3	399900
2	1600	3	329900
3	2400	3	369000
4	1416	2	232000
5	3000	4	539900
6	1985	4	299900
7	1534	3	314900
8	1427	3	198999
9	1380	3	212000
10	1494	3	242500
11	1940	4	239999
12	2000	3	347000
13	1890	3	329999
14	4478	5	699900
15	1268	3	259900
16	2300	4	449900
17	1320	2	299900
18	1236	3	199900
19	2609	4	499998
20	3031	4	599000
21	1767	3	252900
22	1888	2	255000
23	1604	3	242900
24	1962	4	259900
25	3890	3	573900
26	1100	3	249900
27	1458	3	464500
28	2526	3	469000
29	2200	3	475000
30	2637	3	299900
31	1839	2	349900

	Size	Bedrooms	Price
32	1000	1	169900
33	2040	4	314900
34	3137	3	579900
35	1811	4	285900
36	1437	3	249900
37	1239	3	229900
38	2132	4	345000
39	4215	4	549000
40	2162	4	287000
41	1664	2	368500
42	2238	3	329900
43	2567	4	314000
44	1200	3	299000
45	852	2	179900
46	1852	4	299900
47	1203	3	239500

Fit a linear model using `fitlm` and estimate housing prices

The `fitlm` function can be used to fit a regression model with multiple variables as well. Note that the last column (variable) in the input table is considered the response variable by `fitlm` by default while all other variables are considered predictors. A different response variable can be specified, if desired- see the `fitlm` documentation for more information. Run this section to fit a linear model, display the model the coefficients, and predict the price of a 1650 sqft, 3-bedroom house. Compare with your results from ex1.

```
linMdl = fitlm(data);
theta = linMdl.Coefficients.Estimate;
fprintf('Theta computed by fitlm: [%f; %f]',theta(1),theta(2))
```

```
Theta computed by fitlm: [89597.909543; 139.210674]
```

```
price = predict(linMdl,[1650 3]); % Enter your price formula here
fprintf('Predicted price of a 1650 sq-ft, 3 br house: $%f', price);
```

```
Predicted price of a 1650 sq-ft, 3 br house: $293081.464335
```

Using the Regression Learner App

In this section, we reproduce the results of the previous section while providing an introduction to the *Regression Learner App*. This app offers a graphical interface for building, training, and evaluating linear regression models. Run the code below to clear the workspace and reload the housing data, then follow the steps in the next few sections.

```
clear;
% Load Data
data = readtable('ex1data2.txt');
data.Properties.VariableNames = {'Size', 'Bedrooms', 'Price'};
```

Note: If you have difficulty reading the instructions below while the app is open in MATLAB Online, export this script to a pdf file which you can then use to display the instructions in a separate browser tab or window. To export this script, click on the 'Save' button in the 'Live Editor' tab above, then select 'Export to PDF'.

Open the app and select the variables

1. In the MATLAB Apps tab, select the **Regression Learner** app from the Machine Learning section (you may need to expand the menu of available apps).
2. Select '**New Session -> From Workspace**' to start a new interactive session.
3. Under '**Data Set Variable'**, select '**data**' (if not already selected).
4. Under '**Response**' select '**From data set variable**' and '**Price**' (if not already selected).
5. Under '**Predictors**' select '**Size**' and '**Bedrooms**' (if not already selected).
6. Under '**Validation**' select '**No Validation**'
7. Click the '**Start Session**' button.

Select and train the model

There are many available models to choose from (the default model is 'Fine Tree'). To train a regession model corresponding to the one in the previous section obtained using `fitlm`:

1. Expand the model list and select '**Linear**' from the '**Linear Regression Models**' list.
2. Select '**Train**' to fit the model.

Evaluate the model

After training, there are several options available for evaluating model performance.

- The training results, including error values, prediction speed, and training time for the selected model is shown in the '**Current Model**' pane.
- The model predictions for the training data are visualized in the '**Response Plot**', which is shown by default. To plot the response by variable, select the desired feature variable from the choices in '**X-axis**'.
- The '**Predicted vs. Actual**' and '**Residuals**' plots offer additional means of visualizing and evaluating model performance.

Export the model

To export a linear regression model variable from the app into the workspace:

1. Select '**Export Model -> Export Model**'.
2. Select the default output variable name ('trainedModel').
3. Extract the linear model to the variable `LinMdl` by running the command below

```
linMdl = trainedModel.LinearModel
```

`linMdl` variable now contains the linear regression model. It can be used to predict housing prices using the `predict` function using the methods provided earlier for working with `LinearModel` variables returned from `fitlm`.