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
```

```
data = 97×2 table
```

	Population	Profit
1	6.1101	17.5920
2	5.5277	9.1302

	Population	Profit
3	8.5186	13.6620
4	7.0032	11.8540
5	5.8598	6.8233
6	8.3829	11.8860
7	7.4764	4.3483
8	8.5781	12
9	6.4862	6.5987
10	5.0546	3.8166
11	5.7107	3.2522
12	14.1640	15.5050
13	5.7340	3.1551
14	8.4084	7.2258
15	5.6407	0.7162
16	5.3794	3.5129
17	6.3654	5.3048
18	5.1301	0.5608
19	6.4296	3.6518
20	7.0708	5.3893
21	6.1891	3.1386
22	20.2700	21.7670
23	5.4901	4.2630
24	6.3261	5.1875
25	5.5649	3.0825
26	18.9450	22.6380
27	12.8280	13.5010
28	10.9570	7.0467
29	13.1760	14.6920
30	22.2030	24.1470
31	5.2524	-1.2200
32	6.5894	5.9966
33	9.2482	12.1340
34	5.8918	1.8495
35	8.2111	6.5426

	Population	Profit
36	7.9334	4.5623
37	8.0959	4.1164
38	5.6063	3.3928
39	12.8360	10.1170
40	6.3534	5.4974
41	5.4069	0.5566
42	6.8825	3.9115
43	11.7080	5.3854
44	5.7737	2.4406
45	7.8247	6.7318
46	7.0931	1.0463
47	5.0702	5.1337
48	5.8014	1.8440
49	11.7000	8.0043
50	5.5416	1.0179
51	7.5402	6.7504
52	5.3077	1.8396
53	7.4239	4.2885
54	7.6031	4.9981
55	6.3328	1.4233
56	6.3589	-1.4211
57	6.2742	2.4756
58	5.6397	4.6042
59	9.3102	3.9624
60	9.4536	5.4141
61	8.8254	5.1694
62	5.1793	-0.7428
63	21.2790	17.9290
64	14.9080	12.0540
65	18.9590	17.0540
66	7.2182	4.8852
67	8.2951	5.7442
68	10.2360	7.7754

	Population	Profit
69	5.4994	1.0173
70	20.3410	20.9920
71	10.1360	6.6799
72	7.3345	4.0259
73	6.0062	1.2784
74	7.2259	3.3411
75	5.0269	-2.6807
76	6.5479	0.2968
77	7.5386	3.8845
78	5.0365	5.7014
79	10.2740	6.7526
80	5.1077	2.0576
81	5.7292	0.4795
82	5.1884	0.2042
83	6.3557	0.6786
84	9.7687	7.5435
85	6.5159	5.3436
86	8.5172	4.2415
87	9.1802	6.7981
88	6.0020	0.9270
89	5.5204	0.1520
90	5.0594	2.8214
91	5.7077	1.8451
92	7.6366	4.2959
93	5.8707	7.2029
94	5.3054	1.9869
95	8.2934	0.1445
96	13.3940	9.0551
97	5.4369	0.6170

```
summary(data)
```

```
Variables:

    Population: 97×1 double

        Values:
```

```
            Min        5.0269
            Median     6.5894
            Max        22.203

    Profit: 97×1 double

        Values:

            Min       -2.6807
            Median     4.5623
            Max        24.147
```

```
methods(data)
```

```
Methods for class table:

addprop       convertvars   inner2outer   isempty       issortedrows  movevars      outerjoin     rmprop
addvars       head          innerjoin     ismember      join          ndims         removevars    rowfun
cat           horzcat       intersect     isprop        mergevars     numel         renamevars    rows2var
```

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

```
help table.rowfun
```

```
  rowfun Apply a function to rows of a table or timetable.
     B = rowfun(FUN,A) applies the function FUN to each row of the table A, and
     returns the results in the table B.  B contains one variable for each output of
     FUN.  FUN accepts M separate inputs, where M is SIZE(A,2).

     B = rowfun(FUN,A, 'PARAM1',val1, 'PARAM2',val2, ...) allows you to specify
     optional parameter name/value pairs to control how rowfun uses the variables
     in A and how it calls FUN.  Parameters are:

        'InputVariables'      - Specifies which variables in A are inputs to FUN.
        'GroupingVariables'   - Specifies one or more variables in A that define groups
                                of rows.  Each group consists of rows in A that have the
                                same combination of values in those variables.  rowfun
                                applies FUN to each group of rows, rather than separately
                                to each row of A.  B has one row for each group when you
                                specify 'OutputFormat' as 'uniform' or 'cell'.  When you
                                specify 'OutputFormat' as 'table', the sizes of FUN's
                                outputs determine how many rows of B correspond to each
                                group.  For timetables, 'GroupingVariables' can be either
                                the row times vector, or one or more data variables,
                                but not a mix of the two.

     'GroupingVariables' and 'InputVariables' are each a positive integer, a
     vector of positive integers, a variable name, a cell or string array
     containing one or more variable names, a logical vector, or a pattern
     scalar. 'InputVariables' can also be a function handle that returns a
     logical scalar.  In this case, rowfun treats as data variables only
     those variables in A for which that function returns true.

        'SeparateInputs'      - Specifies whether FUN expects separate inputs, or one
                                vector containing all inputs.  When true (the default),
                                rowfun calls FUN with one argument for each data variable.
                                When false, rowfun creates the input vector to FUN by
                                concatenating the values in each row of A, and the data
                                variables in A must be compatible for that concatenation.
```

'ExtractCellContents' - When true, **rowfun** extracts the contents of cell variables in A and passes the values, rather than the cells, to FUN. Default is false. This parameter is ignored when SeparateInputs is false. For grouped computation, the values within each group in a cell variable must allow vertical concatenation.

'OutputVariableNames' - Specifies the variable names for the outputs of FUN.

'NumOutputs' - Specifies the number of outputs with which **rowfun** calls FUN. This may be less than the number of output arguments that FUN declares, and may be zero.

'OutputFormat' - Specifies the form in which **rowfun** returns the values computed by FUN. Choose from the following:

'uniform' - **rowfun** concatenates the values into a vector. All of FUN's outputs must be scalars with the same type.

'table' - **rowfun** returns a table with one variable for each output of FUN. For grouped computation, B also contains the grouping variables. 'table' allows you to use a function that returns values of different sizes or types. However, for ungrouped computation, all of FUN's outputs must have one row each time it is called. For grouped computation, all of FUN's outputs for one call must have the same number of rows. 'table' is the default OutputFormat if A is a table.

'timetable' - **rowfun** returns a timetable with one variable for each output of FUN. For grouped computation, B also contains the grouping variables. B's time vector is created from the row times of A. If these times do not make sense in the context of FUN, use 'table' OutputFormat. For example, when you calculate the means of groups of data, it might not make sense to return the first row time of each group as a row time that labels the group. If this is the case for your data, then return the grouped variables in a table. 'timetable' is the default OutputFormat if A is a timetable.

'cell' - B is a cell array. 'cell' allows you to use a function that returns values of different sizes or types.

'ErrorHandler' - a function handle, specifying the function **rowfun** is to call if the call to FUN fails. **rowfun** calls the error handling function with the following input arguments:

- a structure with fields named "identifier", "message", and "index" containing, respectively, the identifier of the error that occurred, the text of the error message, and the row or group index at which the error occurred.
- the set of input arguments at which the call to the function failed.

The error handling function should either throw an error, or return the same number and type and size of outputs as FUN. These outputs are then returned in B. For example:

```
function [A, B] = errorFunc(S, varargin)
warning(S.identifier, S.message); A = NaN; B = NaN;
```

If an error handler is not specified, **rowfun** rethrows the error from the call to FUN.

Examples:

Example 1 - Simulate a geometric brownian motion model for a range of parameters

```
mu = [-.5; -.25; 0; .25; .5];
sigma = [.1; .2; .3; .2; .1];
params = table(mu,sigma);
```

```
      stats = rowfun(@gbmSim,params, ...
                       'OutputVariableNames',{'simulatedMean' 'trueMean' 'simulatedStd' 'trueStd'});
      [params stats]

      function [m,mtrue,s,strue] = gbmSim(mu,sigma)
      % Discrete approximation to geometric Brownian motion
      numReplicates = 1000; numSteps = 100;
      y0 = 1;
      t1 = 1;
      dt = t1 / numSteps;
      y1 = y0*prod(1 + mu*dt + sigma*sqrt(dt)*randn(numSteps,numReplicates));
      m = mean(y1); s = std(y1);
      % Theoretical values
      mtrue = y0 * exp(mu*t1); strue = mtrue * sqrt(exp(sigma^2*t1) - 1);

   Example 2 - Compute the average difference between a pair of variables, by group.
      t = table(randi(3,15,1),randn(15,1),rand(15,1),'VariableNames',{'g' 'x' 'y'})
      rowfun(@(x,y) mean(x-y),t,'GroupingVariable','g', ...
                     'InputVariables',{'x' 'y'}, 'OutputVariableName','MeanDiff')

   Example 3 - Convert units of ozone air quality timeseries data
      Time = datetime(2015,12,31,20,0,0)+hours(0:8)';
      OzoneData = [32.5 32.3 31.7 32.0 62.5 61.0 60.8 61.2 60.3]';
      Unit = categorical([1 1 1 1 2 2 2 2 2]', [1 2], {'ppbv', 'ug_m3'});
      tt = timetable(Time, OzoneData,Unit);
      % The unit changes between years of data.  Create a function to convert
      % ug/m3 to ppbv.
      function dataPPBV = OzoneConcToMixingRatio(data,unit)
        if unit == 'ppbv'
            dataPPBV = data;
        elseif unit == 'ug_m3'
            % Assume standard T,P.
            dataPPBV = data./2.00;
        else
            dataPPBV = nan;
        end
      end

      % Use rowfun to make a timetable with consistent units for all data
      ttPPBV = rowfun(@OzoneConcToMixingRatio, tt, 'OutputVariableName','Ozone_ppbv')

   Example 4 - Find largest outlier from annual mean in terms of number of standard deviations
      % Make some sample data in a table
      Time = datetime(2016,3,1) + days(randi(1000,50,1));
      data = randn(50,1)+3;
      tt = timetable(Time,data);
      % Shift time to yearly
      tt.Time = dateshift(tt.Time,'start','year');
      % Use rowfun to group and normalize to zero mean and unit standard deviation.
      rf = rowfun(@(x) max(abs((x-mean(x))./std(x))), tt, 'GroupingVariables','Time', 'OutputVariableN

   See also varfun, cellfun, structfun, arrayfun.

 Help for table/rowfun is inherited from superclass tabular

   Documentation for table/rowfun
```

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.

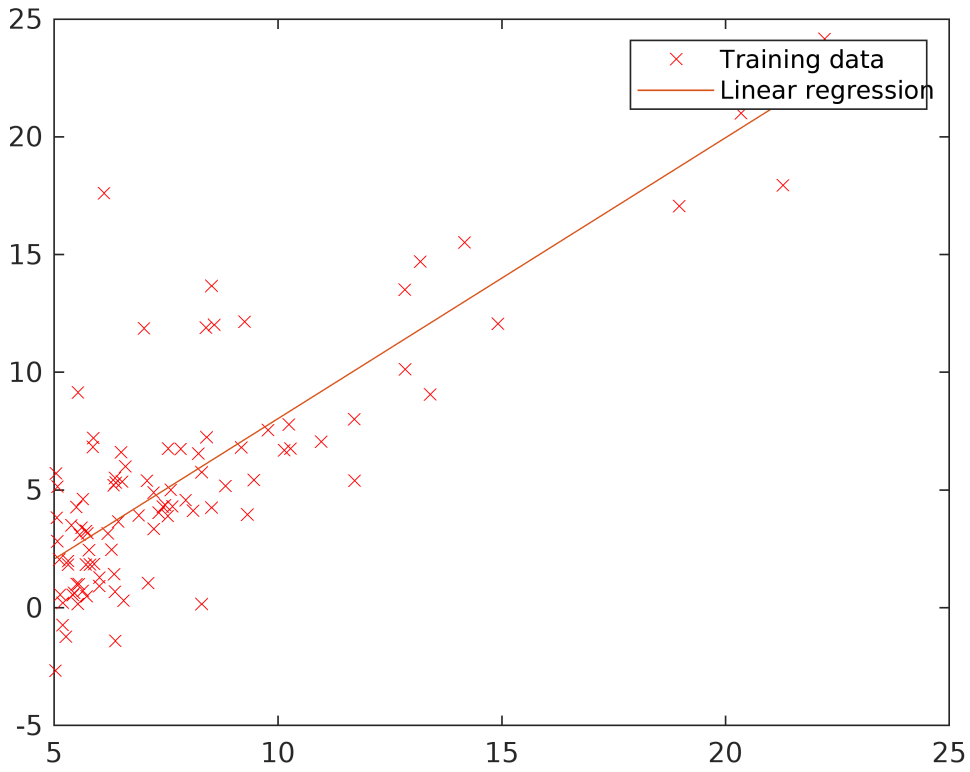
```
% Predict values for population sizes of 35,000 and 70,000
predict1 = predict(linMdl,3.5);
fprintf('For population = 35,000, we predict a profit of %f', predict1*10000);
```

```
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);
linMdl.Coefficients
```

ans = 3×4 table

	Estimate	SE	tStat	pValue
1 (Intercept)	8.9598e+04	4.1767e+04	2.1452	0.0375
2 Size	139.2107	14.7951	9.4092	0
3 Bedrooms	-8.7380e+03	1.5451e+04	-0.5655	0.5746

```
theta = linMdl.Coefficients.Estimate;
fprintf('Theta computed by fitlm: [%.3f; %.3f; %.3f]',theta(1),theta(2), theta(3))
```

```
Theta computed by fitlm: [89597.910; 139.211; -8738.019]
```

```
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 '**Resubstitution 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`.