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

MATLAB Companion Script for Machine Learning ex1 (Optional) 1
Introduction 1
FAQ 1
Linear Regression 1
Files needed for this script 1
Linear Regression with One Variable 2
Load the data into a table 2
Fit a linear model using the fitlm function 8
Making predictions using the LinearModel variable 8
Linear Regression with Multiple Variables 9
Load and preview the data 9
Fit a linear model using fitlm and estimate housing prices 9
Using the Regression Learner App 10
Open the app and select the variables 10
Select and train the model 10
Evaluate the model 10
Export the model 11

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
3	8.5186	13.6620

	Population	Profit
4	7.0032	11.8540
5	5.8598	6.8233
6	8.3829	11.8860
7	7.4764	4.3483
8	8.5781	12.0000
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
36	7.9334	4.5623
37	8.0959	4.1164

	Population	Profit
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
69	5.4994	1.0173
70	20.3410	20.9920
71	10.1360	6.6799

	Population	Profit
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       head          innerjoin     issortedrows  ndims         rmprop        setxor        stack
addvars       height        intersect     join          numel         rowfun        size          summary
cat           horzcat       isempty       mergevars     outerjoin     rows2vars     sortrows      table
convertvars   inner2outer   ismember      movevars      removevars    setdiff       splitvars     tail
```

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.
    C = outerjoin(A, B) creates the table C as the outer join between the
    tables A and B.  If A is a timetable, then B can be either a table or a
    timetable, and outerjoin returns C as a timetable for either
    combination of inputs. An outer join includes the rows that match
    between A and B, and also unmatched rows from either A or B.

    outerjoin first finds one or more key variables.  A key is a variable
    that occurs in both A and B with the same name.  If both A and B are
    timetables, the key variables are the time vectors of A and B.
    outerjoin then uses those key variables to match up rows between A and
    B.  C contains one row for each pair of rows in A and B that share the
    same combination of key values.  In general, if there are M rows in A
    and N rows in B that all contain the same combination of key values, C
    contains M*N rows for that combination.  C also contains rows
    corresponding to key combinations in A (or B) that did not match any
    row in B (or A).  outerjoin sorts the rows in the result C by the key
    values.

    C contains all variables from both A and B, including the keys.  If A and B
    contain variables with identical names, outerjoin adds a unique suffix
    to the corresponding variable names in C.  Variables in C that came from A
    (or B) contain null values in those rows that had no match from B (or A).

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

            'Keys'      - specifies the variables to use as keys.
            'LeftKeys'  - specifies the variables to use as keys in A.
            'RightKeys' - specifies the variables to use as keys in B.

    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.

    When joining two timetables, 'Keys', or 'LeftKeys' and 'RightKeys',
```

must be the time vector names of the timetables.

'MergeKeys' - specifies if **outerjoin** should include a single variable in C for each key variable pair from A and B, rather than including two separate variables. **outerjoin** creates the single variable by merging the key values from A and B, taking values from A where a corresponding row exists in A, and from B otherwise. Default is false. If both A and B are timetables, 'MergeKeys' must always be true.

'LeftVariables' - specifies which variables from A to include in C. By default, **outerjoin** includes all variables from A.

'RightVariables' - specifies which variables from B to include in C. By default, **outerjoin** includes all variables from B.

'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, C contains rows corresponding to keys in A (or B) that did not match any in B (or A), but not vice-versa. By default, **outerjoin** does a full outer join, and includes unmatched rows from both A and B.

[C,IA,IB] = **outerjoin**(A, B, ...) returns index vectors IA and IB indicating the correspondence between rows in C and those in A and B. **outerjoin** constructs C by horizontally concatenating A(IA,LEFTVARS) and B(IB,RIGHTVARS). IA or IB may also contain zeros, indicating the rows in C that do not correspond to rows in A or B, 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.
  a = table({'a' 'b' 'c' 'e' 'h'}',[1 2 3 11 17]','VariableNames',{'Key1' 'Var1'})
  b = table({'a' 'b' 'd' 'e'}',[4 5 6 7]','VariableNames',{'Key1' 'Var2'})

  % Combine a and b 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.
  cfull = outerjoin(a,b,'key','Key1')

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

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


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

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

```
       % Join a and b, ignoring rows in b whose key values do not match any
       % rows in a.
       cleft = outerjoin(a,b,'Type','left')

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

Help for table/outerjoin is inherited from superclass tabular

    Reference page for table/outerjoin
```

Fit a linear model using the `fitlm` function

There are many available functions 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);
```

```
Undefined function or variable 'LinMdl'.
```

```
fprintf('For population = 35,000, we predict a profit of %f', predict1*10000);
```

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

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

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 '**Workspace Variable'**, select '**data**' (if not already selected).
4. Under '**Response**' select '**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 visual 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`.