MATLAB Companion Script for *Machine Learning* ex7 (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 ex7 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 ex7 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.

K-Means Clustering and Dimensionality Reduction Using PCA

In this Live Script, we use functions and apps from the Statistics and Machine Learning Toolbox to perform cluster analysis and PCA for data compression and visualization.

Files needed for this script

- `ex7data1.mat` - Example Dataset for PCA
- `ex7data2.mat` - Example Dataset for K-means
- `ex7faces.mat` - Faces Dataset

- bird_small.png - Example Image

Table of Contents

K-means clustering

In this section we group data using the `kmeans` function. We then visualize the and evaluate the resulting clusters.

Load the data

Run the code below to load the example dataset in `ex7data2.mat` which contains a matrix, X, of two-dimensional data points.

```
clear;
load ex7data2.mat;
```

Cluster using kmeans

The kmeans function automatically performs all steps of the clustering algorithm you implemented in ex7, including random initialization. Note that additional options such as the maximum number of iterations or the preferred distance metric can be set by providing additional inputs. Run the code below to cluster the data into 3 groups. Note that kmeans returns two outputs: a vector containing the assigned centroid for each data point, idx, and a matrix of centroid locations, C. We also print the algorithm progress at each step by setting the Display option to iter.

```
[idx,C] = kmeans(X,3,'Display','iter');
```

Visualize the clusters using gscatter

Run the code in this section to plot the data points using the gscatter function, which provides a convenient way to visualize data by group or category.

```
gscatter(X(:,1),X(:,2),idx,'','','','off'); hold on;
plot(C(:,1),C(:,2),'kx','MarkerSize',10,'MarkerEdgeColor','k','LineWidth',3);
legend(["Cluster "+(1:3),'Centroids']); hold off;
```

Run k-means using multiple random initializations

If you were to rerun the previous sections multiple times, you may occasionally encounter clustering result where one of the three fairly distinct clusters are split by two centroids. In that case, the algorithm converged to a local minimum value of the cost metric. Because of this possibility, it is often advisable to repeat the algorithm multiple times using different initial centroids, which we can accomplish using the 'Replicates' option. The code below will run k-means using 5 different random initializations. Note that only the centroid locations and indices corresponding to the lowest cost are returned as the output.

```
[idx,~] = kmeans(X,3,'Display','iter','Replicates',5);
```

Analyze the groups using grpstats

When a dataset contains categorical labels, or if the data has been clustered into groups as above, it is common to compute statistics for each variable to help determine the differences and similarities between groups. Below we use the grpstats function to compute summary statistics for each variable in the dataset by cluster. The grpstats function will output the individual results for each statistic requested. However, as we are requesting several statistics, it's easier to view them if they are organized as a table (which is the default output of grpstats for table inputs). Run the code below to compute and display a table of summary statistics on the clustered data. The first two columns in the table contain the group number and the number of elements in each group. The remaining columns contain the summary stats - one for each statistic and variable - by group.

```
tblstats = grpstats(array2table([X,idx],'VariableNames',{'X1','X2','Group'}),'Group',{'
```

Determine the optimal number of clusters using silhouette plots

As discussed in the course, it can be difficult to determine the 'best' number of centroids to use when clustering data. A visual inspection of the current data set would lead most people to choose 3 centroids, but this method may prove difficult or impossible for higher dimensional data or larger numbers of groups. One way to help choose the appropriate number of clusters is to create a *silhouette plot*. The 'silhouette' of a data point is a measure of how close that point is to other points in neighboring clusters. This measure ranges from -1 to 1 where:

- Values close to +1 imply the point is very distant from neighboring clusters.
- Values close to 0 imply the point is not distinctly in its assigned cluster or another one.
- Values close to -1 imply the point has most likely been misclassified and belongs to a different cluster.

In general, 'weaker' clusters can be identified as having a larger proportion of points possessing small or even negative silhouette values. The mean silhouette value over all data points can therefore be used to compare the performance of different clustering models. Run the code below to cluster the data using two different numbers of centroids and create a silhouette plot for each using the `silhouette` function, then examine the resulting plots.

```
idx = kmeans(X,3,'Replicates',5);
[s4,~] = silhouette(X,idx);
title(sprintf('Mean silhouette value for 3 centroids: %g',mean(s4)))
idx = kmeans(X,4,'Replicates',5);
[s3,~] = silhouette(X,idx);
title(sprintf('Mean silhouette value for 4 centroids: %g',mean(s3)))
```

As expected, the 3-centroid model appears to perform better than the 4-centroid model based on its larger mean silhouette value. Also note the presence of the two weaker clusters in the 4-silhouette model where a larger proportion of their data points have relatively low silhouette values.

Evaluate the number of clusters using `evalclusters`

The silhouette criterion is just one method of evaluating the performance of a clustering model. The `evalclusters` function allows one to quickly obtain the optimal number of clusters based on a particular criterion- see the documentation for the list of available criteria and their descriptions. Run the code below to call `evalclusters` using the `'silhouette'` criterion over the range of centroid values $k = 2, 3, \ldots, 6$ (provided using the `'Klist'` option) and compare the result with the previous section. Note that `evalclusters` returns an object containing the optimal number of clusters in the `OptimalK` property.

```
clsteval = evalclusters(X,'kmeans','silhouette','Klist',2:6)
fprintf('The optimal number of clusters found for this dataset is %d.',clsteval.Optimal
```

Image Compression with *K*-means

Recall that in ex7 you used k-means to cluster the pixel color values of an example image into 16 seperate clusters. You then compressed the image by replacing the RGB pixels with the closest centroid color. This only required 4 bits per pixel to map to one of the 16 centroids, resulting in significant storage savings over the original set of 3, 8-bit RGB values per pixel. In this section, we will repeat this analysis using functions from the Statistics and Machine Learning Toolbox.

Load, format, and visualize the image data

Run the code below to load and format the image into a double array for use with `kmeans`. A 3D scatter plot will be generated using `scatter3` to visualize the color data. Use the Zoom, Pan, and Rotate figure controls to explore the color content of the image. To view the data in a particular color plane (e.g. red-blue), hover the cursor over the image and click the 'Rotate' icon when it appears. Then right-click the plot and select the desired axis view using the 'Go to' view options.

```
clear;
A = imread('bird_small.png'); % A is a 128x128x3 uint8 image array
X = double(reshape(A(:),length(A(:))/3,3)); % X is a (128*128)x3 double matrix
scatter3(X(:,1),X(:,2),X(:,3),12,X/255,'filled');
xlabel('Red'); ylabel('Green'); zlabel('Blue')
```

Compress the image data using kmeans

Choose a k value using the control below to cluster the color data into k colors and compress the image. The original and compressed images will be displayed along with the compression ratio. The grouped data is also plotted generated for comparison with the figure in the previous section. You may want to launch the figure into a separate window and maximize the figure for closer inspection (hover the cursor over the figure and click the 'arrow' control when it appears).

```
k = 32;
replicates = 10;
[idx,C] = kmeans(X,k,'Replicates',replicates,'MaxIter',200);
% Map the pixels to the centroids and convert to uint8
Xcomp = C(idx,:);
Acomp = uint8(reshape(Xcomp,size(A)));

% Original and compressed scatter plots
subplot(2,2,1)
scatter3(X(:,1),X(:,2),X(:,3),12,X/255);
title(sprintf('Original: %d colors',length(unique(X,'rows'))));
xlabel('Red'); ylabel('Green');zlabel('Blue')
subplot(2,2,3);
scatter3(X(:,1),X(:,2),X(:,3),12,Xcomp/255);
xlabel('Red'); ylabel('Green');zlabel('Blue');
title(sprintf('Compressed: %d colors.', k));

% Original and compressed images
subplot(2,2,2);
imshow(A);
title('Original');
axis square
subplot(2,2,4);
imshow(uint8(Acomp))
title(sprintf('Compressed size: %0.2g%% of original',100*log2(k)/24))
axis square
```

Principal Component Analysis

In ex7 you used principal component analysis to compress data, remove duplicate information, and increase the efficiency of machine learning algorithms. In this section you will use the MATLAB `pca` function from the Statistics and Machine Learning Toolbox to obtain the principal components of the sample dataset used in ex7.

Load the example 2D dataset and rename the ex7 pca function

Run the code below to load the example data matrix X. **Since the pca function from ex7 shares the same name as the MATLAB version, the code below will also rename your pca function to pca_ex7.** Make sure to run the code at the end of this section to undo this change before trying to use ex7 or submit the exercise again.

```
clear;
movefile pca.m pca_ex7.m;
load('ex7data1.mat');
```

Use pca to obtain principle components

The `pca` function takes a matrix as input and returns a matrix as output. The columns of the output matrix contain the principal components sorted in order of decreasing variance. The `pca` function automatically centers the data about the mean before computing the principal components. Additional information from the analysis can be obtained by requesting additional outputs, including the row scores (the data coordinates in the principal component space), column means, and the principal component variances and variance percentages- see the documentation for details. Run the code below to compute the principal components of X. You should see that the two components account for about 87% and 13% of the variance of the data in X, respectively.

```
[coeff,scr,~,~,varpct,mu] = pca(X)
```

Visualize the data in the principal component space

The code below uses the `biplot` function to visualize the data *in the principle component space* using the coefficient and score outputs from `pca`.

```
figure;
biplot(coeff,'Scores',scr)
```

It's clear from the plot that the data values vary the most along the first principal component (x-axis)

Visualize the principal components and reconstruct the data in the original space

The score output contains the coefficients of the original data when projected onto the principal components, so there is no need to write an additional function to project the data as in ex7. Run the code in this section to plot the data points and principal components in the original space. The 'reconstructed' data is also plotted using the projection onto the first principle component.

```
% Plot the data
```

```
figure; hold on;
scatter(X(:,1),X(:,2));
% Plot the mean location
plot(mu(1),mu(2),'kx','MarkerSize',12,'LineWidth',3);
% Plot the principal component directions
m = coeff(2,:)./coeff(1,:);
plot([min(X(:,1)),max(X(:,1))],[m(1)*(min(X(:,1))-mu(1))+mu(2),m(1)*(max(X(:,1))-mu(1)
plot([min(X(:,1)),max(X(:,1))],[m(2)*(min(X(:,1))-mu(1))+mu(2),m(2)*(max(X(:,1))-mu(1)
% Plot the reconstructed data
Xrec = scr(:,1)*coeff(:,1)'+mu;
plot(Xrec(:,1),Xrec(:,2),'b.','MarkerSize',12)
legend({'Data','$\mu$','PC 1','PC 2','Projected Data'},'Location','northwest','Interpre
axis tight; axis equal; hold off;
```

Use PCA to Compress Facial Image Data

In the last part of ex7 you ran PCA on face images to compress the data values by projecting onto a subset of principal components. In the next two sections we reproduce those results using the `pca` function.

Load the face image data and calculate the principal components

Run the code below to load the matrix X of 5000 32x32 grayscale images used in ex7. (Recall that the face images have been 'unwound' into 1024-element row vectors and stacked into X.) The `pca` function is then used to obtain the principal components, scores, variance percentages, and column means of X.

```
clear;
load('ex7faces.mat');
[coeff,scr,~,~,varpct,mu] = pca(X);
```

Select the number of principal components using a variance cutoff

In this section we implement a method below for choosing the minimum number of principal components, k, which account for the chosen variance percentage, v, in the data. Run the code below to calculate the required number of components needed to account for the variance and plot a random image alongside its compressed counterpart. Use the control to increase or decrease the percentage of variance that is accounted for by the projected data.

```
% Find k and project onto first k components
v = 95;
if v == 100
    k = 1024;
else
    k = find(cumsum(varpct)>v,1);
end
Xcomp = scr(:,1:k)*coeff(:,1:k)'+mu;

% Plot images side by side
figure;
colormap(gray);
idx = randi(5000);
subplot(1,2,1);
```

```
imagesc(reshape(X(idx,:),32,32));
xlabel(sprintf('Original image #%d',idx));
axis square
subplot(1,2,2);
imagesc(reshape(Xcomp(idx,:),32,32));
xlabel(sprintf('Reconstructed using %d\n principal components',k));
title(sprintf('Variance cutoff: %d%%',v))
axis square
```

Use PCA with the Regression and Classification Learner Apps

As discussed in the lectures, PCA can help improve the performance of machine learning algorithms by reducing data redundancy prior to training. In the following sections we provide instructions for using PCA with the Regression or Classification Learner Apps.

Generate training data

The code below uses a subset of 20% of the bird image data loaded in the previous section. Artificial class labels are then generated using a hypersphere as a decision surface with some noise near the boundary added in for a more realistic (less well-separated) example. The features and labels are then combined in the matrix `data` and a 3D scatterplot is generated to help visualize the labeled data.

```
clear;
A = imread('bird_small.png'); % A is a 128x128x3 uint8 image array
X = double(reshape(A(:),length(A(:))/3,3)); % X is a (128*128)x3 double matrix
m = length(X);
tind = rand(m,1) < 0.2;
Xtrain = X(tind,:);
yt = (sum(Xtrain.^2,2)<1e5 + 1e4*randn(length(Xtrain),1));
data = [Xtrain,yt];
figure;
scatter3(Xtrain(:,1),Xtrain(:,2),Xtrain(:,3),10,[yt,0*yt,~yt]);
```

To train, export, and utilize an SVM classifier model using PCA on the example data, follow the steps in the next few sections.

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 Classification Learner App and select the data

1. In the **MATLAB Apps tab**, select the **Classification Learner** app from the Machine Learning section.
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 '**column_4**' (if not already selected).
5. Under '**Predictors**' select the remaining columns (if not already selected).
6. Under '**Validation**' select '**No Validation**'.

7. Click the '**Start Session**' button.

Select the model and PCA options and train the classifier

1. Expand the model list and select '**Quadratic SVM**' from the '**Support Vector Machines**' list.
2. Click on the '**PCA**' button to open the **Advanced PCA Options**' menu.
3. Check the '**Enable PCA**' box.
4. Expand the '**Component reduction criterion**' drop-down list and select '**Specify number of components**'.
5. Change the '**Number of numeric components**' to **2**.
6. Close the '**Advanced PCA options**' menu.
7. Click the '**Train**' button.

Export and extract the model and PCA coefficients

1. Select '**Export Model' -> 'Export Model**'.
2. Select the default output variable name ('trainedModel') and click '**OK**'.
3. Close the app.
4. Run the code below to extract the `classificationSVM` model and the PCA information.

```
quadSVMmdl = trainedModel.ClassificationSVM
coeff = trainedModel.PCACoefficients
mu = trainedModel.PCACenters
```

When PCA is used before training the model in the Regression and Classification Learner Apps, the principal components and variable means are included in the `PCACoefficients` and `PCACenters` properties of the `trainedModel` variable. Note that only the principal components used in training are returned.

Predict labels using the model variable and the PCA coefficients

Depending on how you are going to use the trained model, you may work either projected data values or the original data values. In this section we work with projected data, while in the next section we work with the original data, i.e. without projecting data onto the components before predicting. Since the `classificationSVM` model was trained on the projected data, you must project new data, such as a test set, onto the principal components before using the `predict` function, as the original data will contain more features then the model was trained on.

Run the code below to project the remaining pixel data (not used for training) onto the first two principal components and predict classes using the `predict` function. The resulting data points and their predicted classes are then plotted using the original and projected coordinates for comparison.

```
% Center and project the original data onto the principal components
Xtest = (X(~tind,:)-mu)*coeff;
% Classify the projected data
ytest = predict(quadSVMmdl,Xtest);
```

```
% Plot the results in the original coordinates
figure;
scatter3(X(~tind,1),X(~tind,2),X(~tind,3),10,[ytest,0*ytest,~ytest]);
title('Original coordinates');
% Plot the results in the principal coordinates
figure;
gscatter(Xtest(:,1),Xtest(:,2),ytest,'br','o',10,'off')
title('Principal component coordinates');
axis square
```

Predict class labels without projecting using `predictFcn`

If after training *you do not want to work with in the principal component space*, you can predict class labels using feature data in the original coordinate space by using the `predictFcn` function. This function takes feature data from the original space as input and returns a vector of class labels as output- the projection into the component space is handled automatically before prediction. The `predictFcn` is included as a property of all `trainedModel` variables exported from the Regression or Classification Learner Apps. Run the code below to predict the labels using the held-out data values and recreate the 'original coordinates' scatter plot from the previous section.

```
% Use predictFcn
Xtest = X(~tind,:);
ytest = trainedModel.predictFcn(Xtest);
% Plot the results
figure;
scatter3(Xtest(:,1),Xtest(:,2),Xtest(:,3),10,[ytest,0*ytest,~ytest]);
title('Original coordinates')
```

Re-rename your pca function

Run the code below to change the name of your pca function back to 'pca.m' from 'pca_ex7.m' if you plan to work on or resubmit ex7 in the future.

```
movefile pca_ex7.m pca.m
```