University of Notre Dame, Computer Science and Engineering

CSE 40647/60647 Data Mining — Assignment 2
Due Date: March 3rd, 2014 at 11:59pm ET

Association Analysis, Data Preprocessing, and Clustering

February 23, 2014

This assignment will require you to implement and interpret some of the data processing concepts that were introduced in class, such as association analysis, dimensionality reduction, and clustering. **An IPython Notebook with predefined functions to assist you with this assignment is available here. Additionally, the datasets used in this assignment can be downloaded here, here, and here.** Keep in mind that the main objective of this assignment is to highlight the insights that we can derive from applying these techniques—the coding aspect is secondary. Accordingly, you are welcome to consult any online documentation and/or code that has been posted to the course website, so long as all references and sources are properly cited. You are also encouraged to use code libraries, so long as you acknowledge any source code that was not written by you by mentioning the original author(s) directly in your source code (comment or header).

You are expected to submit a single IPython Notebook file following the same instructions and naming convention described in Assignment 0. Answers to the conceptual questions can be embedded in the Notebook as *markdown* cells, and you may use *heading* cells to further organize your document.

1 Association Rules (30 points)

The Data
For this portion of the assignment you will be using the *Extended Bakery dataset*, which describes transactions from a chain of bakery shops that sell a variety of drinks and baked goods. More information about the original unmodified dataset can be found here.

You may download the dataset here.

This particular dataset is made up of smaller subsets that contain 1,000, 5,000, 20,000 and 75,000 transactions each. Furthermore, each of these subsets is given in two different formats. You are free to use either one when writing your solutions. Below is a description of each:

1. Sparse Vector Format: `XXX-out1.csv` files. Each line of the file has the following format:
   - First column: transaction ID
   - Subsequent columns: list of purchased goods (represented by their ID code)

   Example:
   3,0,2,4,6
   Transaction 3 contained items 0, 2, 4, 6

2. Full Binary Vector Format: `XXX-out2.csv` files. Each line has the following format:
   - First column: transaction ID
   - 50 subsequent columns: A sequence of binary values that indicate if item i (represented in column i) has been purchased in that transaction.

   Example:
   3,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   Transaction 3 contained items 0, 2, 4, 6

The sparse vector format can be easily read into a list of transactions in Python by using the following code snippet:

```python
T = [] # list of transactions
infile = open('/path/to/file/file.csv','r')
for line in infile:
    # Append new transaction (skipping \n and transaction number)
    T.append(line.strip().split(',')[1:])
```

The Idea: Association Rule Learning
Given the database of transactions, where each transaction is a list of items, find rules that associate the presence of one set of items with that of another set of items. Ideally, we only want to find rules that are substantiated by the data; we want to avoid spurious associations.

What to Do
Table 1.1 provides a map between the product IDs and the actual items. Your task is to find the association rules that exceed specific values of *minimum support* and *minimum confidence*. You are free to experiment with different values until you find something that produces meaningful/interesting results.

Recall from our class discussions that this process requires two steps, namely finding the *frequent itemsets* and discovering the strong *association rules* within these. To do so, you may re-use some of the code that we provided for Apriori[1], write your own code, or use any Python modules that offer that functionality. Whatever option you choose to follow, be sure that your output is formatted in a clear and legible manner.

What to Provide

Your output should contain the following:

- For each frequent itemset:
  1. All items in it, described by the product names (the `products.csv` file may be helpful on mapping IDs → names).
  2. The support of the itemset.
- For each rule:
  1. All items on the left side of the rule.
  2. All items on the right side of the rule.
  3. The support of the rule.
  4. The confidence of the rule.

Given this output, respond to the following question:

1. Compare the rules that you obtained for each different subset (1,000–75,000 transactions). How does the number of transactions affect the results you observed?

ID	Product	ID	Product	ID	Product	ID	Product	ID	Product
0	Chocolate Cake	**10**	Almond Tart	**20**	Pecan Tart	**30**	Almond Croissant	**40**	Lemonade
1	Lemon Cake	**11**	Apple Pie	**21**	Ganache Cookie	**31**	Apple Croissant	**41**	Raspberry Lemonade
2	Casino Cake	**12**	Apple Tart	**22**	Gongolais Cookie	**32**	Apricot Croissant	**42**	Orange Juice
3	Opera Cake	**13**	Apricot Tart	**23**	Raspberry Cookie	**33**	Cheese Croissant	**43**	Green Tea
4	Strawberry Cake	**14**	Berry Tart	**24**	Lemon Cookie	**34**	Chocolate Croissant	**44**	Bottled Water
5	Truffle Cake	**15**	Blackberry Tart	**25**	Chocolate Meringue	**35**	Apricot Danish	**45**	Hot Coffee
6	Chocolate Eclair	**16**	Blueberry Tart	**26**	Vanilla Meringue	**36**	Apple Danish	**46**	Chocolate Coffee
7	Coffee Eclair	**17**	Chocolate Tart	**27**	Marzipan Cookie	**37**	Almond Twist	**47**	Vanilla Frappuccino
8	Vanilla Eclair	**18**	Cherry Tart	**28**	Tuile Cookie	**38**	Almond Bear Claw	**48**	Cherry Soda
9	Napoleon Cake	**19**	Lemon Tart	**29**	Walnut Cookie	**39**	Blueberry Danish	**49**	Single Espresso

Table 1.1: Products in the dataset and their corresponding codes.

[1] http://nbviewer.ipython.org/github/cse40647/cse40647/blob/sp.14/10%20-%20Apriori.ipynb

2 Data Preprocessing (30 points)

The Data
For this portion of the assignment you will be using a set of images of the University of Notre Dame campus. The dataset consists of 30 color images capturing the campus during the spring, fall, and winter (10 images for each season). Each image consists of 62,500 pixels (250x250 pixels). Each pixel is defined by three color values varying between 0 and 255 that specify the degree or red, green, and blue present at that point.

You may download the dataset here.

The Idea: PCA with Image Histograms
Your objective here will be to perform dimensionality reduction on the dataset to view the similarity of the images within some lower-dimensional (feature) space. Specifically, we would like to perform a transformation on the images so that they can be viewed in a 2-dimensional space, which we can then visualize via a plot. This can be accomplished by performing principal component analysis (PCA) on some property of the images and retaining only the top 2 principal components. Ideally, we want to apply PCA to a property of the images that can be used to capture some notion of the "similarity" between images.

Intuitively, since we typically view an image as a collection of pixels, we might consider performing PCA on the set of pixels, reducing an image from a collection of 62,500 pixel values to a new 2-dimensional space. In other words, we would consider each pixel as a feature of an image, and try to transform an image into a new set of 2 features. Ideally, images that have a similar set of pixels should also be similar in this 2-dimensional space (i.e., they should share similar feature values). However, due to the magnitude of this reduction, a significant amount of information would likely be lost in the transformation, meaning that pictures with similar pixel values many not appear similar in the the new 2-dimensional space.

A better approach might be to reduce the histogram of color values. Each color has 256 possible values, resulting in a histogram of 768 (256*3) color values distributed over the entire range of 65,000 pixels. Each value of the histogram is the number of pixels in the image with the corresponding color value. Here, we would consider each histogram value as a feature of an image. By performing PCA on an image histogram, we are only reducing an image from a set of 768 unique histogram values (rather than the 65,000 unique pixel values) to a new 2-dimensional space. As a result of this transformation, images that have a similar histogram should also be similar in this 2-dimensional space.

What to Do
The IPython Notebook provided with this assignment includes functions to compute the histograms and plot the images within the transformed (2-dimensional) space (`load_images` and `plot_image_space`, respectively). There are also functions to generate and plot the color palettes associated with each image (`cluster_image_colors` and `plot_color_palette`, respectively); the palettes are generated via (k-means) clustering of the pixel color values, and may be investigated at your own leisure—they are not needed to complete the assignment.

The images can be loaded and the histograms generated by running the following code snippet (which imports pandas as `pd`). Please ensure that the directory provided to the `load_images` function is correct. For example, if you have placed all the images in your base IPython Notebook directory in a folder labeled `images`, with the campus images in a subfolder labeled `campus`, then the (relative) path to the campus images would be `images/campus`.

```python
import pandas as pd
# Load images and generate color histograms
images = load_images('/path/to/campus_images')
X = pd.DataFrame([im.histogram() for im in images])
```

Once you have a DataFrame of image histograms, the PCA transformation of this data can be computed and plotted in Python by running the following code snippet (which imports `decomposition` from scikit-learn):

```python
from sklearn import decomposition
# Generate PCA transformation and plot the results
estimator = decomposition.PCA(n_components=2)
X_proj = estimator.fit_transform(X)
plot_image_space(images, X_proj, 'title_of_plot')
```

The plot generated by this code snippet will display the images according to the top 2 principal components (or eigenvectors) found by the PCA transformation of the image histogram.

What to Provide

Your output should contain the following:

- The PCA projection of the image color histograms in 2 dimensions. Using the provided `plot_image_space` function, this should be displayed as thumbnail images distributed within a 2-dimensional plot.

Given this output, respond to the following questions:

1. What does it mean for two images to be close together in this plot? What does it mean for two images to be far apart?
2. Do images corresponding to one of the seasons tend to group together more closely than others? Why might this be the case?

3 Clustering (40 points)

The Data
For this portion of the assignment you will be using the *Iris flower dataset*. The dataset consists of 50 samples from each of three species of Iris, with four features measured from each sample. scikit-learn provides a function to load the dataset (no download required).

The Idea: Choosing k for k-means
Your objective here will be to assess the performance of k-means clustering on the Iris dataset. Recall that the number of clusters, k, is an input parameter to the k-means algorithm. A variety of measurements are available for estimating the optimal value of k. For this assignment, you will look at the sum of squared deviation (SSQ) and the gap statistic. Both of these criteria make use of the intuition that k-means tries to minimize variance (the distance or deviation of each point from each of the k clusters) by iteratively assigning points to their nearest clusters.

Choosing k with SSQ
The SSQ criterion is a direct application of the intuition that k-means tries to minimize variance. Recall that the SSQ criterion sweeps over a range of possible k values, with each value of k associated with a degree of deviation (the distance of each point from each of the k clusters). These deviations can be squared and summed to arrive at the "sum of squared deviation" (SSQ) for each value of k. Larger values of k are expected to continue to reduce the SSQ (because there are more clusters for points to be near, reducing their deviation). However, one could expect a leveling-off in the SSQ once the value of k exceeds the true number of clusters, as this would result in true clusters (that is, clusters actually present in the data) being separated. If, then, one plots the SSQ over a range of k values, this leveling-off point may produce a noticeable "elbow" in the plot. By this criterion, the estimated optimal value of k is that which occurs at this elbow point. While simple, the difficulty with this criterion is that often the elbow point is not distinctive or well-defined.

Choosing k with the Gap Statistic
The gap statistic provides a criterion that produces a quantifiable estimate of the optimal value of k over a range of possible k values. The intuition here is that there is an expected degree of deviation associated with clustering any given dataset. We want the number of clusters, k, that displays the largest "gap" between the deviation we expect, given the dataset and the number of clusters, and the deviation we estimate or observe. Thus, rather than simply considering estimated deviation by itself, we can standardize the estimated deviation for a possible value of k by comparing it with the expected deviation under an appropriate null reference distribution of the data (e.g., a uniform distribution). This difference or gap between the expected deviation and the estimated deviation is termed the gap statistic. Maximizing this gap statistic then corresponds to minimizing the estimated deviation relative to what would be expected. To ensure that we do not needlessly posit additional clusters (i.e., larger values of k), we only consider the value $k+1$ if its gap statistic (minus any measurement error) is higher than that for k. By this criterion, the lowest value of k with a corresponding gap statistic higher than or equal to the gap statistic of $k+1$ is the estimated optimal value of k.

What to Do
The provided assignment IPython Notebook includes functions to compute and plot the gap statistic (`gap_statistics` and `plot_gap_statistics`) and the sum of squared deviation (`ssq_statistics` and `plot_ssq_statistics`).

The Iris flower dataset can be loaded in Python by using the following code snippet (with `datasets` imported from scikit-learn):

```
from sklearn import datasets
# Load the Iris flower dataset
iris = datasets.load_iris()
data = iris.data
```

Then the sum of squared deviations (SSQ) can be easily run and the results plotted by using the following code snippet:

```
# Generate and plot the SSQ statistics
ssqs = ssq_statistics(data, ks=range(2,11+1))
plot_ssq_statistics(ssqs)
```

Similarly, the gap statistic can be easily run and the results plotted by using the following code snippet:

```
# Generate and plot the gap statistics
gaps, errs, difs = gap_statistics(data, nrefs=20, ks=range(2,11+1))
plot_gap_statistics(gaps, errs, difs)
```

Both the SSQ and gap statistic code snippets require a variable `ks`, which defines the range of k values (the "ks") to evaluate. For example, if you would like to evaluate k values between 2 an 11 (inclusive), you could set `ks` as `range(2,11+1)`.

The function `plot_gap_statistics` generates two plots. The first plot simply displays the gap statistic for each k value evaluated. The second plot displays the difference between the gap statistic for value k and that computed for value $k+1$. On the second plot, the first non-negative value, or gap difference, is the optimal number of clusters estimated by the gap statistic criterion.

What to Provide

Your output should contain the following:

- The SSQs computed for k values between 1 and 10 (inclusive). There should be one plot corresponding to the SSQs.
- The gap statistics computed for k values between 1 and 10 (inclusive). There should be two plots corresponding to the gap statistics.

Given this output, respond to the following questions:

1. Where did you estimate the elbow point to be (between what values of k)? What value of k was typically estimated as optimal by the gap statistic? To adequately answer this question, consider generating both measures several times, as there may be some amount of variation in the value of k that they each estimate as optimal.
2. How close are the estimates generated by the elbow point and gap statistic to the number of species of Iris represented in the dataset?
3. Assuming we are trying to generate one cluster for each Iris species represented in the dataset, does one measure seem to be a consistently better criterion for choosing the value of k than the other? Why or why not?

4 Graduate Student Portion (20 points / +10 points for undergraduates)

The Data
We know what can improve this assignment: cute kittens. So for this portion of the assignment you will be using a set of (cute) kitten images. The dataset consists of 25 color images, each preprocessed so that the kittens' faces are similar in size and orientation. Each image consists of 62,500 pixels (250x250 pixels).

You may download the dataset here.

The Idea: Dimensionality Reduction in Face Recognition
Your job, if you accept it (and if you're a graduate student, you must), is to generate face clusters (via k-means clustering) and eigenfaces (via PCA) from this set of kitten face images.

Generating Clusters of Faces
Intuitively, if we wanted to reduce the number of images in a dataset, we could try to cluster these images into groups. If each group is defined by a cluster centroid (that is, an image that represents all members of the cluster) as in k-means clustering, then we could simply retain these centroid images and discard the remaining ones. In this way, we are considering each image as a dimension, and reducing the dimensionality of the set of images, much to the same effect as PCA. A drawback of this method, however, is that there may not be any natural clustering of the images, resulting in a potentially significant loss of information if we retain only the centroid images.

Generating Eigenfaces
Recall that the Eigenface approach to facial recognition is the use of PCA to reduce a collection of face images to a smaller set of reference images. The term typically refers to the use of this approach on human faces, but the same concept is applicable to kitten faces as well. The intuition here is that the eigenfaces will constitute a set of "standardized face ingredients," which may be used to match subsequent face images. We are considering each image as a dimension, and reducing the dimensionality of the set of images.

Note that both the face clustering and Eigenface approaches perform dimensionality reduction on the set of images (that is, we reduce the number of images, where each image is a dimension). This contrasts with performing dimensionality reduction on individual images by reducing the set of pixels to a smaller set of component features (where each pixel is a dimension).

What to Do
The provided assignment IPython Notebook includes functions to load and plot the (kitten) images (`load_images` and `plot_gallery`, respectively).

The kitten images can be easily loaded in Python by using the following code snippet, which makes use of our provided `load_images` function (and assumes that NumPy has been imported as `np`). Please ensure that the directory provided to the `load_images` function is correct. For example, if you have placed all the images in your base IPython Notebook directory in a folder labeled `images`, with the kitten images in a subfolder labeled `cute_kittens`, then the (relative) path to the kitten images would be `images/cute_kittens`.

```
# Load an array of kitten images
kittens = load_images('/path/to/kitten_images', grayscale=True)
kittens = np.array(kittens) # transform to a NumPy array
n_samples, n_features = kittens.shape # number of rows and columns
image_shape = (100, 100) # the image dimensions (width and height)
```

Before performing dimensionality reduction (our objective in generating clusters and eigenfaces), some additional data preprocessing is needed. Generally, dimensionality reduction techniques require centering the data globally (i.e., subtracting the mean from each data point for each feature), because otherwise they may capture spurious fluctuations in the data. Though not strictly needed, we may also center the data locally (i.e., subtracting the mean from each data point for each image) to enhance the visualization of the images.

The following code snippet centers the kitten data and displays the first few images in the centered dataset by making use of our provided `plot_gallery` function. The snippet assumes that the number of retained components (clusters or eigenfaces) has been defined as `n_components` (an integer) and that the number of columns and rows of images to display has been defined as `n_col` and `n_row` (also integers), respectively. Note that `n_components` must equal the product of `n_col` and `n_row`, or you may otherwise receive an error.

```
# Global centering
kittens_centered = kittens - kittens.mean(axis=0)
# Local centering
kittens_centered -= kittens_centered.mean(axis=1).reshape(n_samples,-1)
# Plot the first few centered kitten images
assert n_components == n_col*n_row
plot_gallery("title_of_plot", kittens[:n_components],n_col, n_row)
```

The following code snippet performs k-means clustering on the centered kitten data, thus generating face clusters, and plots the results (with the number of clusters defined as `n_components` and the number of columns and rows of images to display defined as `n_col` and `n_row`, respectively):

```
# Compute k-means clustering on the dataset
estimator = sklearn.cluster.KMeans(n_clusters=n_components, \
                                   tol=1e-3, max_iter=50)
estimator.fit(kittens_centered)
# Plot the resulting kitten face clusters
assert n_components == n_col*n_row
plot_gallery("title_of_plot", \
             estimator.cluster_centers_[:n_components],n_col,n_row)
```

By using the following code snippet, PCA can be performed on the centered kitten data, thus generating eigenfaces, and results plotted (with the number of retained components defined as `n_components` and the number of columns and rows of images to display defined as `n_col` and `n_row`, respectively). Note that, while we have already explicitly centered the data, the PCA function used here actually performs global centering for us.

```
# Compute PCA on the dataset
estimator = decomposition.RandomizedPCA(n_components=n_components, \
                                        whiten=True)
estimator.fit(kittens_centered)
# Plot the resulting PCA eigenfaces
assert n_components == n_col*n_row
plot_gallery("title_of_plot", \
             estimator.components_[:n_components], n_col, n_row)
```

What to Provide

Your output should contain the following:

- The face clusters computed for at least 2 values of `n_components`. The number of displayed face clusters should correspond to this value.
- The eigenfaces computed for at least 2 values of `n_components`. The number of displayed eigenfaces should correspond to this value.

Given this output, respond to the following questions:

1. For each eigenface, darker regions indicate pixels that contribute more to that eigenface and lighter regions indicate pixels that contribute less; in this sense, the further a feature is from a neutral color—in this case gray—the more that feature is "captured" by the eigenface. Briefly describe the sort of cute kitten facial features captured (i.e., those facial features represented by noticeably lighter or darker tones) by the first few eigenfaces. As the first eigenvector should capture the most variance, the first eigenface should correspond to the most general facial features.
2. How do the face clusters (generated by k-means clustering) compare to the eigenfaces (generated by PCA)? As the kitten images have no obvious clusterings, it may be more difficult to interpret the facial features to which each cluster corresponds.