CMSC 25025 / STAT 37601

Machine Learning and Large Scale Data Analysis

Assignment 2*

Out: Thursday, April 6, 2017
Due: Tuesday, April 18, 2017 at 1:30 pm.

This assignment consists of three problems. One is a "pencil and paper" problem on clustering and PCA. The second problem involves clustering of the SOU data, and the third asks you to work with the digits dataset. The second two problems are LSDA problems, and may be completed with a partner.

1. *Properties of k-means and PCA* (10 points)

   (a) The optimal population clustering C^* minimizes $R(C)$. Show by giving an example that $R(\widehat{C})$ being close to $R(C^*)$ does not imply that $\widehat{C}$ is close to C^*.

   (b) Let $R^{(k)}$ denote the minimal risk among all possible clusterings with k clusters. Show that $R^{(k)}$ is nonincreasing in k.

   (c) The problem of fitting the best k-dimensional subspace to data $x_1, \ldots, x_n \in \mathbb{R}^d$ can be written as the optimization

   $$\min_{\mu, \{\lambda_i\}, V_k} \sum_{i=1}^{n} \|x_i - \mu - V_k \lambda_i\|^2$$

   where V_k is an $d \times k$ orthogonal matrix. Show that an optimum over the variables $\mu \in \mathbb{R}^d$ and $\lambda_i \in \mathbb{R}^k$ is given by

   $$\widehat{\mu} = \overline{x}_n = \frac{1}{n} \sum_{i=1}^{n} x_i$$
   $$\widehat{\lambda}_i = V_k^T (x_i - \overline{x}_n).$$

   Show that $\widehat{\mu}$ is not unique, and characterize the set of possible solutions.

LSDA

2. *Friends, Romans, countrymen, lend me your vectors* (20 points)

   In this problem you will use the classic *vector space model* from information retrieval to find similar SOU addresses. Your code from Assignment 1 may come in handy here.

*Compiled 2017/04/14 at 20:10:30

In the vector space model, a document of words d is represented by a TF-IDF vector $\mathbf{w}(d) = (w_1(d), w_2(d), \ldots, w_V(d))$ of length V, where V is the total number of words in the vocabulary. The TF-IDF weights are given by

$$w_i(d) = n_i(d) \cdot \log\left(\frac{|\mathcal{D}|}{\sum_{d' \in \mathcal{D}} \mathbb{1}(t_i \in d')}\right)$$

where $n_i(d)$ is the number of times term t_i appears in document d, $\sum_{d' \in \mathcal{D}} \mathbb{1}(t_i \in d')$ is the number of documents that contain term t_i, and $|\mathcal{D}| = \sum_{d' \in \mathcal{D}} 1$ is the total number of documents in the collection $\mathcal{D}$. This weighting scheme favors terms that appear in few documents.

You will construct an IPython notebook that uses this representation to find similar SOUs.

(a) Compute the TF-IDF vectors for each SOU address. You should lower case all of the text, and remove punctuation. For example, you could use something like this:

```
s = s.lower().translate(string.maketrans("",""), string.punctuation)
```

You will have to make choices about the size of the term vocabulary to use—for example throwing out the 20 most common words, and words that appear fewer than, say, 50 times.

(b) A similarity measure between documents is

$$\text{sim}(d, d') = \frac{\mathbf{w}(d) \cdot \mathbf{w}(d')}{\|\mathbf{w}(d)\|\|\mathbf{w}(d')\|},$$

the cosine of the angle between the corresponding TF-IDF vectors. In terms of this measure, find the

- 50 most similar pairs of SOUs given by different Presidents.
- 50 most similar pairs of SOUs given by the same President.
- 25 most similar pairs of *Presidents*, averaging the cosine similarity over all pairs of their SOUs.

When you read the above speeches, do they indeed seem similar to you? (You can read the speeches in a more reader-friendly format here: `http://www.presidency.ucsb.edu/sou.php`) Comment on what you find, and describe what is needed to construct a better similarity measure between documents.

(c) Using this vector representation, cluster the speeches using k-means. To do this you can use the `mllib` implementation of k-means:

```
from pyspark.mllib.clustering import KMeans
```

then we can run the method `KMeans.train`:

```
clusters = KMeans.train(train_data, 10, maxIterations=50,
initializationMode="random")
```

The options here limit the number of iterations of kmeans to 50, the number of clusters to 10, the clusters are initialized randomly.

Experiment with different number of clusters, and display the clusters obtained (in some manner that you choose). Comment on the clustering results, and whether or not the results are interpretable.

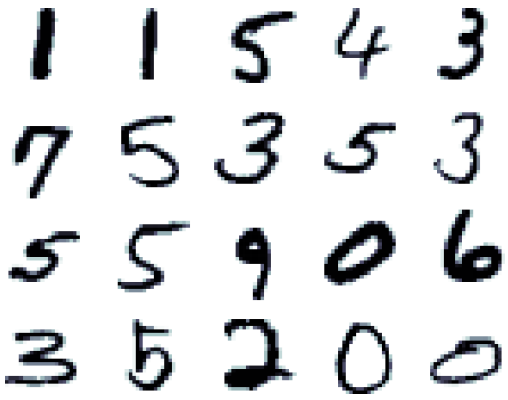
Although the SOU dataset is not very large, you should try to exploit parallelism whenever possible in order to become familiar with this paradigm.

Pass in your code and documentation as an IPython notebook `assn2_prob2.ipynb`, to run on a generic cluster in our RCC infrastructure.

LSDA

3. *Digital divide* (60 points)

For this extended problem you are going to work with the MNIST handwritten digits dataset. The dataset is made up of more than 70,000 images of the digits 0-9. The data are length 784 vectors ranging between [0,255] representing intensity values. The vectors are a flattened version of the 28×28 pixel scans of the digits. Some examples of these data are shown below:



Each data point also has a response value, corresponding to which digit it is.

Here is an example of how to read the data. The dataframe will have two columns : features and label.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("mnist").getOrCreate()
df = spark.read.json('/project/cmsc25025/mnist/data.json')
```

To display the images you can do something like this: store a few digits into a numpy array `subset`, whose rows are digits.

```
%matplotlib inline
import matplotlib.pyplot as plt

nrows = 4
ncols = 5
plt.figure(figsize=(ncols*2, nrows*2))

for i in xrange(nrows*ncols):
    plt.subplot(nrows, ncols, i+1)
    plt.imshow(subset[i].reshape((28,28)), cmap='gray')
    plt.axis('off')

plt.axis('off')
plt.show()
```

This problem has four subparts: (1) PCA, (2) k-means, (3) spectral clustering, and (4) classification.

Part 1: PCA

(a) *Extract principal components.*

After you have read in the training data, you will have to standardize the data, which means subtracting the mean and dividing by the standard deviation. Then, perform PCA to extract the principal components of the standardized training data. You may not use a library to do PCA for you, but you may use libraries (for example from numpy) to compute the singular value or eigenvalue decomposition of a matrix. Display the first 10 principal components as images. If you wish, you can then use the builtin PCA function in `pyspark.mllib` to compare to your results.

(b) *Plot variance*

Plot the variance of all of the principal components—this corresponds to the singular values. This should be monotonically decreasing.

(c) *Dimension reduction*

Take a data point in the test data set and project it onto the first m principal components. Then, transform that m-length vector back into a 784-length vector and display it as an image. Repeat this for several different values of m. Also, try it on different data points. Describe the results qualitatively. Does it give an accurate representation of the images? How do the results depend on m? Can you describe what the top principal components are capturing?

(d) *Plot reconstruction error*

Perform dimension reduction (as above) on the entire test data set. The reconstruction error for a data point is the sum of the squared residuals (the residual is the difference between the true pixel value and the reconstructed pixel value). Plot the reconstruction error against the number of components m used for the dimension reduction.

Part 2: k-means

Now use k-means to perform unsupervised clustering of the digit data, and try to assess how well the clusters capture the structure of the data. You can use the `mllib` implementation of k-means:

```
from pyspark.mllib.clustering import KMeans
```

The clustering is then done with the method `KMeans.train`:

```
clusters = KMeans.train(train_data, 10, maxIterations=50,
initializationMode="random")
```

The options here limit the number of iterations of kmeans to 50, the clusters are initialized randomly. To evaluate how closely the clusters capture the structure of the data, associate each cluster with the majority label.

Construct plots showing samples of the digits from each cluster, and comment on how well the clusters respect the true digit labels.

Part 3: Spectral clustering

Spectral clustering uses k-means on top of a low dimensional representation of data. It makes use of the spectrum (eigenvectors) of a particular similarity matrix of the data in order to perform the dimensionality reduction.

Construct the similarity matrix A by applying Gaussian kernel to squared distance:

$$A_{ij} = \exp(-\|x_i - x_j\|^2/h),$$

where x_i, x_j are digits. The value of the bandwidth h is crucial for good clustering performance. You should experiment with different values to get an appropriate scaling. Next, compute the normalized graph Laplacian matrix

$$L = I - D^{-1/2} A D^{-1/2}$$

where D is the diagonal matrix $D_{ii} = \sum_j A_{ij}$. How is the spectrum of L related to the spectrum of A? What's the last eigenvector of L?

Compute the bottom few eigenvectors of L except the last one. Read off the rows of them as new representation of digits. Run k-means. Typical choices of the reduced dimension are 2 or 3, but you can experiment with different ranges. Compare the result with standard k-means, and describe your findings.

Part 4: Classification

Next you will explore how well clustering and PCA work to organize or represent the digits according to their labels 0-9, using classification algorithms. To begin, split the data into training, development and testing sets where the proportions are approximately 4:1:1.

```
train, dev, test = df.rdd.randomSplit([4, 1, 1])
```

(a) *Multinomial logistic regression* is a linear model for multi-class classification. The conditional probabilities of the outcome class $k \in [K]$ are modeled using the softmax function

$$\mathbb{P}(y = k|x, \beta_k, b_k) = \frac{\exp(\beta_k^\top x + b_k)}{\sum_{j=1}^{K} \exp(\beta_j^\top x + b_j)}.$$

Using the raw features of `mnist` data, train the model on the training set, evaluate the model on development set, and calculate the error rate. You will need the `mllib` implementation of multiclass logistic regression:

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
```

Then you can run the method `LogisticRegression`:

```
model = LogisticRegressionWithLBFGS.train(points, iterations, numClasses)
```

The options here include the training data, the number of iterations of training, and the number of possible outcomes for the classification problem.

The next step is to determine out how well PCA works in terms of classification. To use PCA for classification, we first represent each data point in terms of its principal components. Then, we train a classifier to predict the class label given those principal components, using logistic regression. To predict the label for a new data point, we use its projection onto the principal eigenvectors.

After setting up the basic work flow you can use cross-validation to select the number of principal components.

(b) Train multinomial logistic regression model on training data, using the top k principal components. Predict the labels of the development data and compute the error rate. Plot the error as a function of k. How does the error change as the number k vary?

(c) Pick k that minimizes the error on the development data. Retrain the model using both training and development data. Compute the error rate on testing data. Compare the result to multinomial logistic regression using raw features, and describe your findings.

`chisubmit` instructions

Two `ASSIGNMENT_ID`s will be created in the `chisubmit` system this time, i.e. `a2` (pencil & paper) and `a2p` (sou & mnist). Please name your submissions `assn2_prob1.pdf` for `a2` and `assn2_prob2.ipynb` & `assn2_prob3.ipynb` for `a2p`. If you work with a partner, you need to register `a2p` using the `chisubmit --partner` option. Please see the details at `http://chi.cs.uchicago.edu/chisubmit/students.html`.