CAP 6610 Machine Learning, Spring 2025

Homework 2 Solution

The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The data set can be downloaded here: `<http://qwone.com/~jason/20Newsgroups/>`. For simplicity, we will just focus on the "bag-of-words" representation of the documents given in the Matlab/Octave section. In this case, the input $\boldsymbol{x}_i$ is a vector of word histogram of doc i, and the output y_i is the news group that it belongs to.

The data has been divided into a training set and test set. You will build a model from the training set, and evaluate its performance on the test set. The vocabulary size is more than 60,000, which is bigger than the number of documents. (Also, you will find that the `test_data` matrix has more rows than the `train_data` matrix, which means some words in the test data never appear in the training data.) To simplify computation, you will prune the data first by keeping only the words that have appeared more than 1,000 times in the training data (which is approximately 300) and keep only those rows in both the `train_data` matrix and the `test_data` matrix.

1. Assume each doc is a huge multinomial random variable, with cardinality equal to the vocabulary size and the total number of draws is the length of that doc, given the label. In other words, $p(\boldsymbol{x}|y)$ is multinomial; for each class c, the parameter for the multinomial distribution is a nonnegative vector $\boldsymbol{p}_c$ that sums to one. Describe how to estimate $\boldsymbol{p}_c$. Write the resulting classifier in the form $\widehat{y} = \max_c(\boldsymbol{w}_c^\top \boldsymbol{x} + \beta_c)$. Explain how to obtain $\boldsymbol{w}_c$ and β_c from $p(y)$ and $p(\boldsymbol{x}|y)$.

   Remark. When calculating the likelihoods for multinomials, the expression $\prod_i p_i^{x_i}$ may give extremely small numbers that could cause underflow. We can overcome this issue by instead calculating $\sum_i x_i \log(p_i)$, which does not change which one is the maximum. However, beware of not taking $\log(0)$.

   Solution. If we directly model a document as a multinomial distribution for each class, we need to estimate a vector $\boldsymbol{p}_c$ for each class. This can be done as

$$\boldsymbol{p}_c = \frac{1}{\sum_{i\in\text{class } c} \boldsymbol{1}^\top \boldsymbol{x}_i} \sum_{i\in\text{class } c} \boldsymbol{x}_i.$$

   $p(y)$ is simply obtained from counting the number of documents of a specific class divided by the total number of docs, i.e.,

$$p(y) = \pi_c = n_c/n.$$

   The probability that a new document $\boldsymbol{x}$ belongs to class c is proportional to

$$\pi_c \prod_{j=1}^{m} p_{cj}^{x_j},$$

   where the term that involve the factorials are the same for all classes, therefore inconsequential in making the predictions. Again, to prevent underflow, we take the log, resulting in the following classifier:

$$\begin{aligned}\widehat{y} &= \arg\max_c \left(\log \pi_c + \sum_{j=1}^{m} x_j \log p_{cj} \right) \\ &= \arg\max_c (\boldsymbol{w}_c^\top \boldsymbol{x} + \beta_c),\end{aligned}$$

where

$$\boldsymbol{w}_c(j) = \log p_{cj}, \qquad \beta_c = \log \pi_c.$$

The prediction accuracy is 75.90% on the test set.

2. We can also assume each $p(\boldsymbol{x}|y)$ follows a multivariate normal distribution. Here we assume their covariance matrices are the same, which means the classifier is equivalent to the linear discriminant analysis. Of course, most people don't believe that bag-of-words actually follows a normal distribution, so some pre-processing is used. Do a Google search of TF-IDF and apply that to the data set before training your LDA model.

   Solution. Let's first consider the TF-IDF preprocessing. There is no unique definition, but here's one reasonable choice. Term frequency (TF) is simply taken as $\log(x_{ij}+1)$, so zero remains zero, but bigger values becomes less significant as their raw counts. Inverse document frequency (IDF) is defined for each term over the training set: the total number of documents divided by the number of documents that contains term j, and then we take the log of it. Using the $\boldsymbol{\phi}_i$ vectors that we defined before, then it is $\log(n/\sum_{i=1}^n \phi_i(j))$. As a result, the training data, term j in document i is modified to be

$$\log(x_{ij}+1) \log \frac{n}{\sum_{i=1}^n \phi_i(j)}.$$

   We will apply the same normalization to the test sets. The term frequency is again log of the word count plus one. We will use the IDF obtained from the training data. In practice, new docs may come one at a time, and we should use a fixed normalization scheme to keep the model consistent.

   Now let's talk about LDA. The probability that a doc belongs to one class is proportional to

$$\pi_c \det(2\pi\boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_c).\right)$$

   Again we take the log and ignore terms that are common for all classes, and the resulting classifier is

$$\widehat{y} = \arg\max_c \left(\boldsymbol{\mu}_c^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_c^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_c + \log \pi_c\right),$$

   where

$$\boldsymbol{\mu}_c = \frac{1}{n_c}\sum_{i\in\text{class } c} \boldsymbol{x}_i, \qquad \boldsymbol{\Sigma} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^\top - \frac{1}{n}\sum_{c=1}^k n_c \boldsymbol{\mu}_c\boldsymbol{\mu}_c^\top.$$

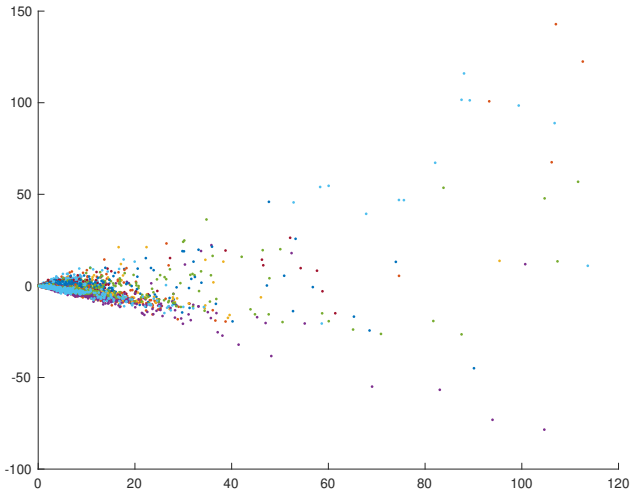
   The prediction accuracy is 72.26% on the test set.

For each case, derive the mathematical expressions for the corresponding classifiers. Be specific about how to calculate each and every model parameter from data. Pick a language that you like and program these two classifiers using the training set, and report their prediction accuracy on the test set.

Now let us apply some of the unsupervised methods by ignoring their labels. We will only consider the training data. You are required to code the algorithms by yourselves in the language of your choice.

3. *LSI/PCA via orthogonal iteration.* Implement the orthogonal iteration algorithm that finds the PCA projection matrix $\boldsymbol{\Theta}$ of a data matrix $\boldsymbol{\Phi}$. You are allowed to use a pre-existing function of QR. Apply tf-idf to the term-document matrix to obtain $\boldsymbol{\Phi}$ and feed it into your orthogonal iteration algorithm. Remember to use sparse matrix operations to avoid unneccessary memory/computational complexities. Set $k = 2$ and let the algorithm run until $\boldsymbol{\Theta}$ doesn't change much. Then get $\boldsymbol{Y} = \boldsymbol{\Theta}^\top\boldsymbol{\Phi}$. Each column of $\boldsymbol{Y}$ is a two-dimensional vector that you can plot on a plain. Plot all the documents on a two-dimensional plain, and use a different color for each point that belong to different news groups.

   Solution. The figure is given below. It looks like PCA embedding does not reveal much clustering structure from data.



4. *GMM via EM.* Implement the EM algorithm for the Gaussian mixture model (with different means and covariances for each Gaussian component). The data matrix $\boldsymbol{\Phi}$ is obtained from LSI with $k_{\text{LSI}} = 100$ using the previous orthogonal iteration algorithm. Run the EM algorithmn for GMM with $k_{\text{GMM}} = 20$ until convergence. For each Gaussian component $\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$, calculate $\boldsymbol{\Theta}\boldsymbol{\mu}_c$ where $\boldsymbol{\Theta}$ is the PCA projection; the vector $\boldsymbol{\Theta}\boldsymbol{\mu}_c$ should be element-wise nonnegative. For each cluster c, show the 10 terms that have the highest value in $\boldsymbol{\Theta}\boldsymbol{\mu}_c$. The index-term mapping can be found here `<http://qwone.com/~jason/20Newsgroups/vocabulary.txt>`. Does the result make sense?

   Solution. The top-10 keywords for all 20 clusters seem to be the same, even though a k-means++ initialization is applied. This shows the limitation of EM and the difficulty of estimating the parameters of GMM.

   Here's the top-10 words appearing in all clusters. They are all commonly used words and thus not very informative:

   - as
   - they
   - was
   - you
   - we
   - he
   - are
   - by
   - will
   - not