# CAP 6610 Machine Learning, Spring 2020

Homework 1 Solution

1. (10 points) The uniform distribution for a continuous variable $x$ is

$$p(x;a,b) = \frac{1}{b-a}, \qquad a \le x \le b.$$

Verify that this distribution is normalized (integrats to one), and find expressions for its mean and variance.

**Solution.**

$$\int_a^b p(x;a,b)dx = \int_a^b \frac{1}{b-a}dx = \frac{x}{b-a}\Big|_a^b = \frac{b}{b-a} - \frac{a}{b-a} = 1.$$

$$\mathrm{E}[x] = \int_a^b \frac{x}{b-a}dx = \frac{x^2}{2(b-a)}\Big|_a^b = \frac{b^2}{2(b-a)} - \frac{a^2}{2(b-a)} = \frac{a+b}{2}.$$

$$\mathrm{E}[x^2] = \int_a^b \frac{x^2}{b-a}dx = \frac{x^3}{3(b-a)}\Big|_a^b = \frac{b^3}{3(b-a)} - \frac{a^3}{3(b-a)} = \frac{a^2+ab+b^2}{3}.$$

Therefore

$$\mathrm{var}(x) = \mathrm{E}[x^2] - (\mathrm{E}[x])^2 = \frac{4a^2+4b^2+4ab-3(a+b)^2}{12} = \frac{(a-b)^2}{12}.$$

2. (10 points) Recall that the PMF of a Poisson random variable is

$$p(x;\lambda) = \frac{\lambda^x e^{-\lambda}}{x!},$$

with one parameter $\lambda$. Given i.i.d. samples $x_1, ..., x_n \sim \mathrm{Pois}(\lambda)$, derive the MLE for $\lambda$.

**Solution.** Maximum likelihood tries to solve the following problem

$$\underset{\lambda}{\text{minimize}} \quad \sum_{i=1}^n \left(x_i \log\lambda - \lambda - \log x_i!\right).$$

Taking derivative with respect to $\lambda$ and set it equal to zero gives

$$\sum_{i=1}^n \left(\frac{x_i}{\lambda} - 1\right) = 0.$$

The result is

$$\lambda = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

again the sample average of the data set.

3. (10 points) The function `randn(d,1)` generates a multivariate normal variable $\boldsymbol{x} \in \mathbb{R}^d$ with zero mean and covariance $\boldsymbol{I}$. Describe how to generate a random variable from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. *Hint.* If $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top)$.

   **Solution.** Find a "square-root decomposition" of $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$. This can be done by, for example, the Cholesky decompositon or the eigen-decomposition $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ by setting $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}$. Let $\boldsymbol{z}$ be generated from $\mathcal{N}(0, \boldsymbol{I})$, then $\boldsymbol{L}^\top\boldsymbol{z} + \boldsymbol{\mu}$ follows the distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

4. (20 points) Consider a data set in which each data sample $i$ is associated with a weighting factor $r_i > 0$, and we instead try to minimize the weighted MSE function

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} r_i(y_i - \boldsymbol{\phi}_i^\top\boldsymbol{\theta})^2.$$

   Find an expression for the solution $\widehat{\boldsymbol{\theta}}$ that minimizes this loss function.

   **Solution.** The gradient with respect to one component function $r_i(y_i - \boldsymbol{\phi}_i^\top\boldsymbol{\theta})^2$ is

$$\boldsymbol{\phi}_i r_i(\boldsymbol{\phi}_i^\top\boldsymbol{\theta} - y_i).$$

   Therefore, the gradient for the whole function is

$$\sum_{i=1}^{n} \boldsymbol{\phi}_i r_i(\boldsymbol{\phi}_i^\top\boldsymbol{\theta} - y_i) = \boldsymbol{\Phi}^\top\boldsymbol{R}(\boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y}),$$

   where $\boldsymbol{R}$ is a diagonal matrix with the $(i,i)$th entry equal to $r_i$. Setting it equal to zero gives

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^\top\boldsymbol{R}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{R}\boldsymbol{y}.$$

5. (50 points) 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.

(a) One effective classifer by Tom Mitchell is an instance of the naive Bayes model. First, the actual word count is ignored in the input data; we only consider whether a word $j$ appears in doc $i$ or not. Then each feature in $\boldsymbol{x}$ can be viewed as a Bernoulli random variable. Furthermore, the naive Bayes assumption states that each of these Bernoulli random variables are conditionally independent given the label $y$, i.e., $p(\boldsymbol{x}|y) = \prod p(x_j|y)$. Each of the $p(x_j|y)$ can be easily estimated using the training data.

(b) To incorporate the word count, we can impose a rather different probabilistic model. 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.

(c) 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 classifer 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.

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 three classifiers using the training set, and report their prediction accuracy on the test set.

Submit your code and make sure they are executable. We may or may not check your code.

**Solution.** To make the prediction, you will need $p(y)$ and $p(\boldsymbol{x}|y)$. For all models, $p(y)$ is simply obtained from counting the number of documents of a specific class divided by the total number of docs, i.e.,

$$p(c) = \pi_c = \frac{n_c}{\sum_{j=1}^{k} n_j}.$$

Now we explain how to estimate $p(\boldsymbol{x}|y)$ for each class.

(a) Define the feature representation $\boldsymbol{\phi}_i$ for the $i$th document as

$$\boldsymbol{\phi}_i(j) = \begin{cases} 1 & \boldsymbol{x}_i(j) \neq 0 \\ 0 & \boldsymbol{x}_i(j) = 0. \end{cases}$$

For this model, we assume that each $\boldsymbol{\phi}_i(j)$ follows a Bernoulli distribution, and the conditional joint completely factors. The probability that term $j$ appears in a doc from class $c$ is $p(\phi_j|c) = p_{cj}^{\phi_j}(1-p_{cj})^{1-\phi_j}$ with the parameter $p_{cj}$, which can be estimated by

$$p_{cj} = \frac{1}{n_c} \sum_{i \in \text{class } c} \boldsymbol{\phi}_c(j).$$

For a new document represented as a 0-1 vector $\boldsymbol{\phi}$, the probability that it belongs to class $c$ is proportional to

$$\pi_c \prod_{j=1}^{m} p_{cj}^{\phi_j}(1-p_{cj})^{1-\phi_j}.$$

To prevent underflow, we can take the log of that quantity. The resulting classifier takes the following form:

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

where

$$\boldsymbol{w}_c(j) = \log \frac{p_{cj}}{1-p_{cj}}, \qquad \beta_c = \log \pi_c + \sum_{j=1}^m \log(1-p_{cj}).$$

The prediction accuracy is 74.51% on the test set.

(b) 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

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

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.

(c) 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

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

For the 20 Newsgroup data set, there are more terms than docs, so the $\boldsymbol{\Sigma}$ matrix obtained before is not directly invertible. An easy fix is to add a small diagonal matrix $\lambda\boldsymbol{I}$ before inverting it. Here's the prediction errors for a small selection of $\lambda$'s:

| $\lambda$ | $10^{-10}$ | $10^{-5}$ | $10^{-3}$ | $10^{-2}$ | $10^{-1}$ | 1 |
|---|---|---|---|---|---|---|
| prediction accuracy | 32.38% | 45.57% | 52.26% | 65.09% | 74.60% | 72.26% |

Notice that all three classifiers have a linear form $\arg\max_c(\boldsymbol{w}_c^\top\boldsymbol{x} + \beta_c)$.