CSE 6740 HW2

Archit Sengupta: 904013900

September 2024

1 PCA

Let $\mathbf{v}$ be a unit-length vector and $\mathbf{x}_i$ be the data points. Let $f_{\mathbf{v}}(\mathbf{x})$ be the projection of $\mathbf{x}$ onto the vector $\mathbf{v}$:

$$f_{\mathbf{v}}(\mathbf{x}) = \arg\min_{\mathbf{u} \in V} \|\mathbf{x} - \mathbf{u}\|^2,$$

where $V = \{\alpha \mathbf{v} : \alpha \in R\}$.

The projection $f_{\mathbf{v}}(\mathbf{x})$ can be written as:

$$f_{\mathbf{v}}(\mathbf{x}) = (\mathbf{x}^\top \mathbf{v})\mathbf{v},$$

since the projection of $\mathbf{x}$ onto $\mathbf{v}$ is given by the dot product $\mathbf{x}^\top \mathbf{v}$ in the direction of $\mathbf{v}$.

The mean squared error between projected points and original points is:

$$\begin{aligned} MSE &= \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - f_{\mathbf{v}}(\mathbf{x}_i)\|^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - (\mathbf{x}_i \cdot \mathbf{v})\mathbf{v}\|^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i \cdot \mathbf{x}_i) - 2(\mathbf{x}_i \cdot \mathbf{v})^2 + (\mathbf{x}_i \cdot \mathbf{v})^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i\|^2 - (\mathbf{x}_i \cdot \mathbf{v})^2 \end{aligned}$$

The first term is constant with respect to $\mathbf{v}$, so minimizing MSE is equivalent to maximizing:

$$\frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i \cdot \mathbf{v})^2 = \mathbf{v}^T \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T \right) \mathbf{v}$$

The term in parentheses is the sample covariance matrix $\mathbf{S}$. So, we want to maximize:

$$\mathbf{v}^T \mathbf{S} \mathbf{v}$$

subject to $\|\mathbf{v}\| = 1$. This is the definition of the first principal component, which is the eigenvector of $\mathbf{S}$ with the largest eigenvalue.

So the unit-length vector $\mathbf{v}$ that minimizes the MSE between projected points and original points is the first principal component of the data.

2 Density Estimation

Consider a histogram-like density model where the space $\mathbf{x}$ is divided into fixed regions, and the density $p(\mathbf{x})$ takes a constant value h_i over the i-th region. The volume of region i is denoted by Δ_i, and we have the constraint:

$$\sum_i h_i \Delta_i = 1.$$

Let N represent the total number of observations, and n_i represent the number of observations that fall in region i.

2.1 (a) Log-likelihood Function

The probability of observing a data point in region i is proportional to $h_i \Delta_i$. Likelihood function L is:

$$L(h_1, h_2, \ldots, h_k) = \prod_i (h_i \Delta_i)^{n_i}.$$

The log-likelihood function ℓ is:

$$\ell(h_1, h_2, \ldots, h_k) = \log L = \sum_i n_i \log(h_i \Delta_i).$$

$$\ell(h_1, h_2, \ldots, h_k) = \sum_i n_i \log h_i + \sum_i n_i \log \Delta_i.$$

The second term is independent of h_i:

$$\ell(h_1, h_2, \ldots, h_k) = \sum_i n_i \log h_i + c.$$

2.2 (b) Maximum Likelihood Estimator for h_i

To find the maximum likelihood estimator for h_i,the total probability sums to 1:

$$\sum_i h_i \Delta_i = 1.$$

maximize the log-likelihood using a Lagrange multiplier λ.

Objective function:

$$\mathcal{L} = \sum_i n_i \log h_i + \lambda \left(1 - \sum_i h_i \Delta_i\right).$$

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{n_i}{h_i} - \lambda \Delta_i = 0.$$

$$h_i = \frac{n_i}{\lambda \Delta_i}.$$

But, $\sum_i h_i \Delta_i = 1$:

$$\sum_i \frac{n_i}{\lambda \Delta_i} \Delta_i = 1,$$

$$\frac{1}{\lambda} \sum_i n_i = 1.$$

Thus, $\lambda = N$, where $N = \sum_i n_i$ is the total number of observations.

So, MLE for h_i is:

$$h_i = \frac{n_i}{N \Delta_i}.$$

3 Bernoulli Mixture Model (BMM)

$$P(\mathbf{x}|i) = \prod_{d=1}^{D} \theta_{id}^{x_d} (1 - \theta_{id})^{1 - x_d}.$$

3.1 (a) Log-Likelihood Function and Latent Variables

The log-likelihood for N observations using the BMM is given by:

$$L(\pi, \theta) = \log P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N | \pi, \theta).$$

Using the mixture model, the probability of each observation is:

$$P(\mathbf{x}_n | \pi, \theta) = \sum_{i=1}^{K} \pi_i P(\mathbf{x}_n | i).$$

Thus, the log-likelihood becomes:

$$L(\pi, \theta) = \sum_{n=1}^{N} \log \left(\sum_{i=1}^{K} \pi_i \prod_{d=1}^{D} \theta_{id}^{x_{nd}} (1 - \theta_{id})^{1 - x_{nd}} \right).$$

If we use the EM algorithm to find the maximum likelihood estimate (MLE), the latent variables are the hidden component assignments z_{ni}, where $z_{ni} = 1$ if the n-th data point is assigned to component i, and $z_{ni} = 0$ otherwise. These latent variables are not observed, but they indicate the membership of each data point to the mixture components.

3.2 (b) Lower Bound of Log-Likelihood and E-Step

To derive the lower bound of the log-likelihood, we use Jensen's inequality:

$$\log\left(\sum_{i=1}^{K}\pi_i P(\mathbf{x}_n|i)\right) \geq \sum_{i=1}^{K} q_{ni}\log\left(\frac{\pi_i P(\mathbf{x}_n|i)}{q_{ni}}\right),$$

where q_{ni} is a probability distribution over the latent variables z_{ni}, such that $\sum_{i=1}^{K} q_{ni} = 1$.

The lower bound is:

$$L(\pi,\theta) \geq \sum_{n=1}^{N}\sum_{i=1}^{K} q_{ni}\left(\log\pi_i + \sum_{d=1}^{D}[x_{nd}\log\theta_{id} + (1-x_{nd})\log(1-\theta_{id})] - \log q_{ni}\right).$$

For the E-step, we maximize the lower bound with respect to q_{ni}, with the constraint that $\sum_{i=1}^{K} q_{ni} = 1$. The optimal q_{ni} is given by the posterior probability of the latent variable z_{ni}:

$$q_{ni} = P(z_{ni} = 1|\mathbf{x}_n,\pi,\theta) = \frac{\pi_i\prod_{d=1}^{D}\theta_{id}^{x_{nd}}(1-\theta_{id})^{1-x_{nd}}}{\sum_{j=1}^{K}\pi_j\prod_{d=1}^{D}\theta_{jd}^{x_{nd}}(1-\theta_{jd})^{1-x_{nd}}}.$$

3.3 (c) M-Step

In the M-step, we maximize the lower bound with respect to the parameters π_i and θ_i.

$$\sum_{n=1}^{N}\sum_{i=1}^{K} q_{ni}\log\pi_i.$$

Using the constraint that $\sum_{i=1}^{K}\pi_i = 1$, we introduce a Lagrange multiplier λ and maximize:

$$\mathcal{L} = \sum_{n=1}^{N}\sum_{i=1}^{K} q_{ni}\log\pi_i + \lambda\left(1 - \sum_{i=1}^{K}\pi_i\right).$$

Taking the derivative with respect to π_i and setting it to zero:

$$\frac{\partial\mathcal{L}}{\partial\pi_i} = \frac{\sum_{n=1}^{N} q_{ni}}{\pi_i} - \lambda = 0.$$

Solving for π_i, we get:

$$\pi_i = \frac{\sum_{n=1}^{N} q_{ni}}{N}.$$

Maximizing with respect to θ_{id}:

We maximize:

$$\sum_{n=1}^{N} q_{ni} \left(x_{nd} \log \theta_{id} + (1 - x_{nd}) \log(1 - \theta_{id}) \right).$$

Taking the derivative with respect to θ_{id} and setting it to zero:

$$\frac{\partial}{\partial \theta_{id}} \sum_{n=1}^{N} q_{ni} \left(x_{nd} \log \theta_{id} + (1 - x_{nd}) \log(1 - \theta_{id}) \right) = 0.$$

This leads to the update rule:

$$\theta_{id} = \frac{\sum_{n=1}^{N} q_{ni} x_{nd}}{\sum_{n=1}^{N} q_{ni}}.$$

Thus, the M-step update rules are:

$$\pi_i = \frac{\sum_{n=1}^{N} q_{ni}}{N}, \quad \theta_{id} = \frac{\sum_{n=1}^{N} q_{ni} x_{nd}}{\sum_{n=1}^{N} q_{ni}}.$$

4 Feature Selection

4.1 (a)

$$H(X, Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x, y)$$

Using the chain rule for entropy:

$$H(X, Y) = H(X) + H(Y|X)$$

$$H(X, Y) = - \sum_{x \in X} p(x) \log p(x) - \sum_{x \in X} p(x) H(Y|X = x)$$

conditional entropy:

$$H(Y|X) = - \sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \log p(y|x)$$

By Jensen's inequality, since the logarithm is a concave function, we have:

$$H(Y|X) \leq - \sum_{x \in X} p(x) \log \left(\sum_{y \in Y} p(y|x) \right)$$

conditional probabilities sum to 1:

$$\sum_{y \in Y} p(y|x) = 1$$

Applying the inequality $\ln x \leq x - 1$ for $x = \sum_{y \in Y} p(y|x)$:

$$H(Y|X) \leq -\sum_{x \in X} p(x) \log 1 = 0$$

Thus

$$H(X, Y) \leq H(X) + H(Y)$$

4.2 (b)

The equality $H(Z) = H(X) + H(Y)$ holds when X and Y are independent, but independence is not necessary.

Sufficient Condition: If X and Y are independent, the joint entropy is:

$$H(X, Y) = H(X) + H(Y)$$

Thus, for $Z = X + Y$, we have:

$$H(Z) = H(X) + H(Y)$$

Hence, independence is a sufficient condition for $H(Z) = H(X) + H(Y)$.

Not Necessary: However, independence is not a necessary condition. In some cases, even if X and Y are dependent, the equality $H(Z) = H(X) + H(Y)$ can still hold. Consider two random variables X and Y such that $Y = X$. Clearly, X and Y are dependent. However, let X be uniformly distributed over $\{0, 1\}$.

- The entropy of X is:

$$H(X) = -\sum_{x} p(x) \log p(x) = -\left(\frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2}\right) = \log 2$$

- Since $Y = X$, we also have:

$$H(Y) = H(X) = \log 2$$

- Now, $Z = X + Y$ can take the values $\{0, 2\}$, both with probability $\frac{1}{2}$. Hence, the entropy of Z is:

$$H(Z) = -\left(\frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2}\right) = \log 2$$

Thus, despite X and Y being dependent, we have:

$$H(Z) = H(X) = H(Y) = \log 2$$

5 Naive Bayes

5.1 (a)

Given that $\sigma_{ik} = \sigma$ (variance is independent of the class variable and X_i), the likelihood for M training samples is:

$$L(\mu_{ik}) = \prod_{j=1}^{M} P(X_i = x_{ij} \mid Y = y_k) = \prod_{j=1}^{M} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\left(x_{ij} - \mu_{ik}\right)^2}{2\sigma^2}\right)$$

Taking the log-likelihood:

$$\log L(\mu_{ik}) = \sum_{j=1}^{M} \left(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(x_{ij} - \mu_{ik})^2}{2\sigma^2}\right)$$

$$\log L(\mu_{ik}) = -\frac{M}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=1}^{M}(x_{ij} - \mu_{ik})^2$$

To maximize, take the derivative with respect to μ_{ik} and set it to 0:

$$\frac{\partial}{\partial \mu_{ik}} \log L(\mu_{ik}) = \frac{1}{\sigma^2}\sum_{j=1}^{M}(x_{ij} - \mu_{ik}) = 0$$

$$\hat{\mu}_{ik} = \frac{1}{M}\sum_{j=1}^{M} x_{ij}$$

the MLE estimator is:

$$\hat{\mu}_{ik} = \frac{1}{M}\sum_{j=1}^{M} x_{ij}$$

5.2 (b)

The likelihood for the Bernoulli distribution is:

$$L(\theta_{1k}) = \prod_{j=1}^{M} \theta_{1k}^{x_{1j}} \left(1 - \theta_{1k}\right)^{1 - x_{1j}}$$

log-likelihood:

$$\log L(\theta_{1k}) = \sum_{j=1}^{M} \left(x_{1j}\log\theta_{1k} + (1 - x_{1j})\log(1 - \theta_{1k})\right)$$

To maximize, take derivative with respect to θ_{1k} and set to 0:

$$\frac{\partial}{\partial \theta_{1k}} \log L(\theta_{1k}) = \sum_{j=1}^{M} \left(\frac{x_{1j}}{\theta_{1k}} - \frac{1 - x_{1j}}{1 - \theta_{1k}} \right) = 0$$

$$\frac{1}{\theta_{1k}} \sum_{j=1}^{M} x_{1j} = \frac{1}{1 - \theta_{1k}} \sum_{j=1}^{M} (1 - x_{1j})$$

$$\hat{\theta}_{1k} = \frac{1}{M} \sum_{j=1}^{M} x_{1j}$$

the MLE estimator is:

$$\hat{\theta}_{1k} = \frac{1}{M} \sum_{j=1}^{M} x_{1j}$$

6 Logistic regression

6.1 (a) Log-odds of Success

$$\ln \left(\frac{P[Y = 1 \mid X = x]}{P[Y = 0 \mid X = x]} \right)$$

Substitute the expression for $P[Y = 1 \mid X = x]$ and $P[Y = 0 \mid X = x] = 1 - P[Y = 1 \mid X = x]$:

$$\ln \left(\frac{\frac{\exp\left(w_0 + w^T x\right)}{1 + \exp(w_0 + w^T x)}}{1 - \frac{\exp(w_0 + w^T x)}{1 + \exp(w_0 + w^T x)}} \right)$$

Simplify:

$$\ln \left(\frac{\exp\left(w_0 + w^T x\right)}{1} \right) = w_0 + w^T x$$

Thus, the log-odds of success is a linear function of X:

$$\ln \left(\frac{P[Y = 1 \mid X = x]}{P[Y = 0 \mid X = x]} \right) = w_0 + w^T x$$

6.2 (b) Convexity of Logistic Loss

logistic loss:

$$\ell(z) = \log(1 + \exp(-z))$$

To show that $\ell(z)$ is convex, we compute the second derivative.

First derivative:

$$\frac{d}{dz}\ell(z) = \frac{-\exp(-z)}{1 + \exp(-z)} = -\sigma(z)$$

where $\sigma(z) = \frac{1}{1+\exp(-z)}$ is the sigmoid function.

Second derivative:

$$\frac{d^2}{dz^2}\ell(z) = \sigma(z)(1 - \sigma(z))$$

Since $\sigma(z)(1 - \sigma(z)) \geq 0$ for all z, the second derivative is non-negative, so $\ell(z)$ is a convex function.

7 Programming

7.1 Document clustering using EM

Expectation-Maximization (EM) algorithm for clustering documents using a mixture of multinomial distributions. The dataset (data.mat) contains a word count matrix, and 4 clusters were made.

Initialization:
Multiple random initializations were used for the word-topic probabilities , ensuring normalization. EM Steps: The E-step calculated document-cluster probabilities, while the M-step updated the word-topic distributions.
Results:
The algorithm was run with different random initializations, giving accuracies of 0.8750, 0.8150, 0.8025, and 0.8200 on testing with true labels, reflecting variability in convergence but overall effective clustering performance.
Conclusion:
The EM algorithm successfully clustered documents with accuracy ranging from 80.25% to 87.5%, demonstrating its robustness across different initializations.

7.2 Realistic Topic Models

Words in each document could belong to multiple topics. The conditional probability matrix, $P(w|z)$, representing the probability of word w given topic z, is learned through EM steps. Experimented with different values of k (number of topics) to determine the most interpretable clustering structure.

The algorithm initializes word-topic probabilities $P(w|z)$ randomly, followed by iterative EM updates. The process repeats until convergence. After learning, the top words associated with each topic are extracted using a provided utility function.

7.2.1 Results

Number of Topics: 2

The first classification identifies broad themes in learning and neural networks, capturing foundational principles. However, while it offers a general overview, the lack of specificity may lead to overlapping terms. The second topic focuses on performance metrics and training methodologies, highlighting important aspects of model evaluation. Yet, the generality of the first topic may dilute its effectiveness.

Number of Topics: 3

In the three-topic classification, the first topic combines neural network architecture with training processes. The second emphasizes algorithmic approaches and data handling. The third highlights model evaluation and practical outcomes. However, some redundancy in the second and third topics may still exist, indicating a need for more distinct definitions.

Number of Topics: 4

The four-topic classification offers a clearer division, with the first addressing the dynamics between learning algorithms and system behavior. The second explores architectural components, while the third concentrates on information processing in networks. The fourth focuses on error metrics and performance assessment. There is potential overlap, particularly between the first and fourth topics, which could impact interpretability.

Number of Topics: 5

With five topics, this classification provides the most detailed view. Each topic has a distinct focus, which enhances the granularity of the analysis. However, this increased has led to redundancy and overlap, especially between closely related topics. Balancing complexity and coherence is key.

Conclusion:

Best Classification: 3 or 4 Topics

Reasoning: The three-topic classification is simpler and more focused, but the four-topic classification offers greater detail without excessive overlap. It balances coherence and granularity. Five topics might introduce too much complexity and redundancy, making it less interpretable.

HW2 of CSE 6740

Programming: Text Clustering

In this problem, we will explore the use of EM algorithm for text clustering. Text clustering is a technique for unsupervised document organization, information retrieval. We want to find how to group a set of different text documents based on their topics. First we will analyze a model to represent the data.

Bag of Words

The simplest model for text documents is to understand them as a collection of words. To keep the model simple, we keep the collection unordered, disregarding grammar and word order. What we do is counting how often each word appears in each document and store the word counts into a matrix, where each row of the matrix represents one document. Each column of matrix represent a specific word from the document dictionary. Suppose we represent the set of nd documents using a matrix of word counts like this:

$$D_{1:n_d} = \begin{pmatrix} 2 & 6 & \dots & 4 \\ 2 & 4 & \dots & 0 \\ \vdots & & \ddots & \end{pmatrix} = T$$

This means that word W_1 occurs twice in document D_1. Word W_{n_w} occurs 4 times in document D_1 and not at all in document D_2.

Multinomial Distribution

The simplest distribution representing a text document is multinomial distribution (Bishop Chapter 2.2). The probability of a document D_i is:

$$p\left(D_i\right) = \prod_{j=1}^{n_w} \mu_j^{T_{ij}}$$

Here, μ_j denotes the probability of a particular word in the text being equal to w_j, T_{ij} is the count of the word in document. So the probability of document D_1 would be $p\left(D_1\right) = \mu_1^2 \cdot \mu_2^6 \cdot \ldots \cdot \mu_{n_w}^4$.

Mixture of Multinomial Distributions

In order to do text clustering, we want to use a mixture of multinomial distributions, so that each topic has a particular multinomial distribution associated with it, and each document is a mixture of different topics. We define $p(c) = \pi_c$ as the mixture coefficient of a document containing topic c, and each topic is modeled by a multinomial distribution $p\left(D_i \mid c\right)$ with parameters μ_{jc}, then we can write each document as a mixture over topics as

$$p\left(D_i\right) = \sum_{c=1}^{n_c} p\left(D_i \mid c\right) p(c) = \sum_{c=1}^{n_c} \pi_c \prod_{j=1}^{n_w} \mu_{jc}^{T_{ij}}$$

EM for Mixture of Multinomials

In order to cluster a set of documents, we need to fit this mixture model to data. In this problem, the EM algorithm can be used for fitting mixture models. This will be a simple topic model for documents. Each topic is a multinomial distribution over words (a mixture component). EM algorithm for such a topic model, which consists of iterating the following steps:

Expectation

Compute the expectation of document D_i belonging to cluster c:

$$\gamma_{ic} = \frac{\pi_c \prod_{j=1}^{n_w} \mu_{jc}^{T_{ij}}}{\sum_{c=1}^{n_c} \pi_c \prod_{j=1}^{n_w} \mu_{jc}^{T_{ij}}}$$

Maximization

Update the mixture parameters, i.e. the probability of a word being W_j in cluster (topic) c, as well as prior probability of each cluster.

$$\mu_{jc} = \frac{\sum_{i=1}^{n_d} \gamma_{ic} T_{ij}}{\sum_{i=1}^{n_d} \sum_{l=1}^{m_w} \gamma_{ic} T_{il}}$$

$$\pi_c = \frac{1}{n_d} \sum_{i=1}^{n_d} \gamma_{ic}$$

Task

Implement the algorithm and run on the toy dataset `data.mat`. Observe the results and compare them with the provided true clusters each document belongs to. Report the evaluation (e.g. accuracy) of your implementation. Hint: We already did the word counting for you, so the data file only contains a count matrix like the one shown above. For the toy dataset, set the number of clusters n_c = 4. You will need to initialize the parameters. Try several different random initial values for the probability of a word being W_j in topic c, μ_{jc}. Make sure you normalized it. Make sure that you should not use the true cluster information during your learning phase.

```
import scipy.io
import itertools
import numpy as np

def acc_measure(idx, Y):
    """
    Function for evaluate the accuracy
    :param idx:
        numpy array of (num_doc)
    :param Y:
        numpy array of (num_doc)
    :return:
        accuracy
    """
    # rotate for different idx assignments
    best_acc = 0
    for idx_order in itertools.permutations([1, 2, 3, 4]):

        for ind, label in enumerate(idx_order):
            Y[(ind)*100:(ind+1)*100] = label

        acc = (Y == idx).sum() / Y.shape[0]
        if acc > best_acc:
            best_acc = acc

    return best_acc

# Example:
#     acc_measure(np.array([1]*100 + [3]*100 + [2]*100 + [4]*100))

def cluster(bow, K, num_iters = 1000, epsilon = 1e-12):
    """
    # TODO: Finish the cluster function
    :param bow:
        bag-of-word matrix of (num_doc, V), where V is the vocabulary size
    :param K:
        number of topics
    :return:
        idx of size (num_doc), idx should be 1, 2, 3 or 4
    """


    num_docs, num_words = bow.shape

    # Initialize parameters
    pi = np.random.rand(K)
    pi /= np.sum(pi)

    mu = np.random.rand(K, num_words)
    mu /= mu.sum(axis=1, keepdims=True)


    for _ in range(num_iters):
        # E-Step
        gamma = pi * np.prod(mu ** bow[:, np.newaxis, :], axis=2)  # Shape: (num_docs, K)
        gamma /= gamma.sum(axis=1, keepdims=True)  # Normalize
        old_pi = pi.copy()
        old_mu = mu.copy()
        # M-Step: Update pi and mu
        pi = gamma.sum(axis=0) / num_docs  # Shape: (K,)

        mu = (gamma.T @ bow)  # Shape: (K, num_words)
        mu /= mu.sum(axis=1, keepdims=True)  # Normalize across words for each topic
        #check for convergence
        if np.linalg.norm(pi - old_pi) < epsilon and np.linalg.norm(mu - old_mu) < epsilon:
            print('Converged')
            break
```

```
    # Assign each document
    idx = np.argmax(gamma, axis=1) + 1  # Add 1 to make the labels 1-based
    return idx

# Evaluation
mat = scipy.io.loadmat('data.mat')
mat = mat['X']
X = mat[:, :-1]
Y = mat[:, -1]
for i in range (4):
    idx = cluster(X, 4)
    acc = acc_measure(idx, Y)
    print(f'{i} : accuracy %.4f' % (acc))
```

```
Converged
0 : accuracy 0.8750
Converged
1 : accuracy 0.8150
Converged
2 : accuracy 0.8025
Converged
3 : accuracy 0.8200
```

Extra Credit: Realistic Topic Models

The above model assumes all the words in a document belongs to some topic at the same time. However, in real world datasets, it is more likely that some words in the documents belong to one topic while other words belong to some other topics. For example, in a news report, some words may talk about “Ebola” and “health”, while others may mention “administration” and “congress”. In order to model this phenomenon, we should model each word as a mixture of possible topics.

Specifically, consider the log-likelihood of the joint distribution of document and words

$$\mathcal{L} = \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} T_{dw} \log P(d, w)$$

where T_{dw} is the counts of word w in the document d. This count matrix is provided as input. The joint distribution of a specific document and a specific word is modeled as a mixture

$$P(d, w) = \sum_{z \in \mathcal{Z}} P(z) P(w \mid z) P(d \mid z)$$

where $P(z)$ is the mixture proportion, $P(w \mid z)$ is the distribution over the vocabulary for the z-th topic, and $P(d \mid z)$ is the probability of the document for the z-th topic. And these are the parameters for the model.

The E-step calculates the posterior distribution of the latent variable conditioned on all other variables

$$P(z \mid d, w) = \frac{P(z) P(w \mid z) P(d \mid z)}{\sum_{z'} P(z') P(w \mid z') P(d \mid z')}$$

In the M-step, we maximizes the expected complete log-likelihood with respect to the parameters, and get the following update rules

$$P(w \mid z) = \frac{\sum_d T_{dw} P(z \mid d, w)}{\sum_{w'} \sum_d T_{dw'} P(z \mid d, w')}$$

$$P(d \mid z) = \frac{\sum_w T_{dw} P(z \mid d, w)}{\sum_{d'} \sum_w T_{d'w} P(z \mid d', w)}$$

$$P(z) = \frac{\sum_d \sum_w T_{dw} P(z \mid d, w)}{\sum_{z'} \sum_{d'} \sum_{w'} T_{d'w'} P(z' \mid d', w')}$$

Task

Implement EM for maximum likelihood estimation and cluster the text data provided in the nips.mat file you downloaded. You can print out the top key words for the topics/clusters by using the show `display_topics` utility. It takes two parameters: 1) your learned conditional distribution matrix, i.e., $P(w|z)$ and 2) a cell array of words that corresponds to the vocabulary. You can find the cell array wl in the nips.mat file. Try different values of k and see which values produce sensible topics. In assessing your code, we will use another dataset and observe the produces topics.

```
import scipy.io
import itertools
import numpy as np
```

```
def cluster_extra(T, K, num_iters = 50, epsilon = 1e-12):
    """
    TODO: Finish the function of clustering.
    :param bow:
        bag-of-word matrix of (num_doc, V), where V is the vocabulary size
    :param K:
        number of topics
    :return:
        word-topic matrix of (V, K)
    """
    T=T.toarray()
    num_docs, vocab_size = T.shape
    Word_topic_matrix = np.random.rand(vocab_size, K)  # P(w|z)
    Doc_topic_probs = np.random.rand(num_docs, K)   # P(d|z)
    Topic_probabilities = np.random.rand(K)  # P(z)

    # Normalize the initial distributions
    Word_topic_matrix /= Word_topic_matrix.sum(axis=0)
    Doc_topic_probs /= Doc_topic_probs.sum(axis=0)
    Topic_probabilities /= Topic_probabilities.sum()

    # EM loop
    for iteration in range(num_iters):

        # E-Step: Compute responsibilities P(z | d, w)

        Responsibility_matrix = Topic_probabilities * Doc_topic_probs[:, np.newaxis, :] * Word_topic_matrix[np.newaxis, :, :]
        Responsibility_matrix /= Responsibility_matrix.sum(axis=2, keepdims=True)  # Normalize

        # M-Step
        matrix = (T[:, :, np.newaxis] * Responsibility_matrix)
        # Update P(w|z)
        New_word_topic_matrix= matrix.sum(axis=0)
        New_word_topic_matrix /= New_word_topic_matrix.sum(axis=0)
        # Update P(d|z)
        New_doc_topic_probs = matrix.sum(axis=1)
        New_doc_topic_probs /= New_doc_topic_probs.sum(axis=0)
        # Update P(z)
        New_topic_prob = matrix.sum(axis=(0, 1))
        New_topic_prob /= New_topic_prob.sum()

        # Check for convergence
        x= np.linalg.norm(New_topic_prob-Topic_probabilities)
        if x < 1e-4:
            print("Converged")
            break

        # Update parameters
        Word_topic_matrix, Doc_topic_probs, Topic_probabilities = New_word_topic_matrix, New_doc_topic_probs, New_topic_prob

    return Word_topic_matrix



def display_topics(W, wl):
    """

    :param W:
        word-topic matrix of size (V, K)
    :param wl:
        array of str of size (V)
    :return:
    """
    top_n_words = 10
    ind_mat = np.argsort(-W.T, axis=1)[:, :top_n_words]
    for k_ind in range(ind_mat.shape[0]):
        w_ls = [wl[ind, 0][0] for ind in ind_mat[k_ind]]
        print('topic %i: %s' % (k_ind, ' '.join(w_ls)))


## Evaluation for displaying topics
n_topics = 5 # TODO specify num topics yourself
cell = scipy.io.loadmat('nips.mat')

mat = cell['raw_count'] # sparse mat of size (num_doc, num_words)
wl = cell['wl']
```

```
for n_topics in range(2,6):
    print(f'\n Number of topics: {n_topics}')
    W = cluster_extra(mat, n_topics)
    display_topics(W, wl)
```

```
 Number of topics: 2
topic 0: learning model input neural networks data network figure set time
topic 1: network output training neural point parameters error examples function learning

 Number of topics: 3
topic 0: model neural time training set figure data learning output function
topic 1: learning set network input algorithm neural time system number data
topic 2: network model learning figure input function output networks results data

 Number of topics: 4
topic 0: learning neural training algorithm data function model based system order
topic 1: model input network set time figure output networks training state
topic 2: network information time neural set learning input networks hidden model
topic 3: learning network neural function figure data error model system networks

 Number of topics: 5
topic 0: network input model learning data output figure training state set
topic 1: networks neural model function network time number figure input learning
topic 2: network learning neural function data algorithm set training system information
topic 3: time model learning set figure neural data hidden training input
topic 4: network networks error time figure learning training algorithm weights number
```