CSCI 5521: Machine Learning Fundamentals (Spring 2021)

Homework 2

(Due Thu, Mar. 4, 11:59 PM central)

1. **(15 points)** Let $\mathbf{x}^i \in \mathbb{R}^d$ and $\mathcal{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ be a set of N samples drawn i.i.d. from a multivariate Gaussian distribution in $\mathbb{R}^d$ with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. Recall that the density function of a multivariate Gaussian distribution is given by:

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right] .$$

   (a) **(10 points)** Derive the maximum likelihood estimates for the mean μ and covariance Σ based on the sample set $\mathcal{X}$.[1]

   (b) **(2 points)** Let $\hat{\mu}$ be the maximum likelihood estimate of the mean. Is $\hat{\mu}$ a biased estimate of the true mean μ? Clearly justify your answer by computing $E[\hat{\mu}]$.

   (c) **(3 points)** Let $\hat{\Sigma}$ be the maximum likelihood estimate of the covariance matrix. Is $\hat{\Sigma}$ a biased estimate of the true covariance Σ? Clearly justify your answer by computing $E[\hat{\Sigma}]$.

Programming assignment:

2. **(40 points)** In this problem, we will use the `Boston50` dataset[2] and consider two parametric classifiers by modeling each class's conditional distribution $p(\mathbf{x}|C_i)$ as multivariate Gaussians with (a) full covariance matrix Σ_i and (b) diagonal covariance matrix Σ_i. In particular, using the training data, we will compute the class prior probabilities $p(C_i)$ and the class conditional probabilities $p(x|C_i)$ based on the maximum likelihood estimates of the mean $\hat{\mu}_i$ and the (full/diagonal) covariance $\hat{\Sigma}_i$ for each class C_i. The classification will be done using the following discriminant function:

$$g_i(\mathbf{x}) = \log p(C_i) + \log p(\mathbf{x}|C_i) .$$

   Write code for a class `MultiGaussClassify` in file `MultiGaussClassify.py` with two key functions:

   `MultiGaussClassify.fit(self,X,r)` and `MultiGaussClassify.predict(self,X)`.

   For the class, the `__init__(self,k,d,diag)` function can initialize the parameters for each class to be uniform prior, zero mean, and identity covariance, i.e., $p(C_i) = 1/k$, $\mu_i = \mathbf{0}$ and $\Sigma_i = \mathbb{I}$, $i = 1, \ldots, k$. Here, the number of classes k and the dimensionality d of features is passed as an argument to the constructor of `MultiGaussClassify`. Further, `diag` is boolean (True or False) which indicates whether the estimated class covariance matrices should be a full matrix (`diag`=False) or a diagonal matrix (`diag`=True).

[1] You can use material from the `Matrix Cookbook` on Canvas and/or the textbook for your derivation.

[2] This is the same dataset from homework 1 and we provide code to create this dataset in hw2q2.py.

For `fit(self,X,r)`, the inputs (X, r) are the feature matrix and class labels respectively, and the function will learn (estimate) the parameters for each class, i.e., $p(C_i)$, μ_i, and Σ_i, $i = 1, \ldots, k$.

For `predict(self, X)`, the input X is the feature matrix corresponding to the validation or test set and the output should be the predicted labels for each point in the validation or test set.

We will compare the performance of three models:

(i) `MultiGaussClassify` with full class covariance matrices,

(ii) `MultiGaussClassify` with diagonal covariance matrices, and

(iii) `LogisticRegression`[3]

applied to the `Boston50` dataset. Using the code provided for `my_cross_val()` with 5-fold cross-validation, report the error rates in each fold as well as the mean and standard deviation of error rates across folds for the three models.

You will have to submit (a) **code** and (b) **summary of results**:

(a) **Code**: You will have to submit code for `MultiGaussClassify` as well as a wrapper script `hw2q2.py`. For the class, please use the following template:

```
class MultiGaussClassify:

def __init__(self, k, d, diag):
     ...
def fit(self, X, r):
     ...
def predict(self, X):
     ...
```

Your class `MultiGaussClassify` **should not** inherit any base class in `sklearn`. Again, the three functions you must implement in the `MultiGaussClassify` class are `__init__`, `fit`, and `predict`.

The wrapper script `hw2q2.py` is used to prepare the dataset, and make calls to `my_cross_val`(`method`,X,r,k) to generate the error rate results. The `method` argument is an object of MultiGaussClassify (e.g., `method = MultiGaussClassify(k, d, True)`).

The results should be printed to terminal (not generating an additional file in the folder). Make sure the calls to `my_cross_val`(`method`,X,r,k) are made in the following order and add a print to the terminal before each call to show which method and dataset is being used:

1. `MultiGaussClassify` with full covariance matrix on `Boston50`,
2. `MultiGaussClassify` with diagonal covariance matrix on `Boston50`, and
3. `LogisticRegression` on `Boston50`.

[3]You should use `LogisticRegression` from `scikit-learn`, similar to Homework 1.

For example, the first call to `my_cross_val(method,`X`,`r`,`k`)` should result in the following output:

```
Error rates for MultiGaussClassify with full covariance matrix on Boston50:
Fold 1:  ###
Fold 2:  ###
...
Fold 5:  ###
Mean:  ###
Standard Deviation:  ###
```

(b) **Summary of results**: For each method, report the validation set error rates for each of the $k = 5$ folds, the mean error rate over the k folds, and the standard deviation of the error rates over the k folds. Make a table to present the results for each method (3 tables in total). Each column of the table represents a fold, and add two columns at the end to show the overall mean error rate and standard deviation over the k folds. For example:

Error rates for MGC with full cov matrix on Boston50						
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	SD
#	#	#	#	#	#	#

3. (**20 points**) Write code to implement PCA and apply it to the Digits dataset by projecting the data to $\mathbb{R}^2$ using the first two principal components. Remember to center your data (by subtracting off the mean) before computing the principal components and before projecting the data points. Plot all samples in the projected space and label each data point with the corresponding digit in 10 different colors for the 10 types of digits.

   You will have to submit (a) **code** and (b) **summary of results**:

   (a) **Code**: You will have to submit file `myPCA.py` which contains two functions `myPCA(X,k)` and `ProjectDatapoints(X,W,`$\hat{\mu}$`)`.

   `myPCA` function takes two **inputs**: (1) the original data $X \in \mathbb{R}^{N\times d}$ from the Digits dataset, and (2) an integer $k = 2$ indicating how many principle components used for projection. It returns (1) the projection matrix $W \in \mathbb{R}^{d\times 2}$, and (2) the estimated mean $\hat{\mu} \in \mathbb{R}^{d\times 1}$ of the Digits data. You have to implement your own PCA, but you are allowed to use numpy functions such as mean, cov and linalg.eig.

   `ProjectDatapoints` function takes three **inputs**: (1) the original data $X \in \mathbb{R}^{N\times d}$, (2) the projection matrix $W \in \mathbb{R}^{d\times 2}$ learnt from `myPCA`, and (3) the estimated mean $\hat{\mu} \in \mathbb{R}^{d\times 1}$ returned by `myPCA`. It returns the projected data $\hat{X} \in \mathbb{R}^{N\times 2}$ used to plot.

   (b) **Summary of results**: Plot all samples in the projected space and label each data point with the corresponding digit in 10 different colors for the 10 types of digits using the provided code (`hw2q3.py`).

4. **(25 points)** In this problem, we consider k-means for image segmentation. We can use k-means to cluster pixels with similar (color) values together to generate a segmented or compressed version of the original image. Write code to implement your own version of the k-means algorithm, using squared Euclidean distance, and apply it to the provided image

"stadium.png". For each $k = \{3, 5, 7\}$, generate a segmented image and compute the distortion measure J from the lecture notes (i.e., reconstruction error E from the book).

You will have to submit (a) **code** and (b) **summary of results**:

(a) **Code**: You will have to submit files `my_kmeans.py` which contains the function `kmeans(X,k)` and `hw2q4.py`.
`kmeans` function takes two **inputs**: (1) pixels $X \in \mathbb{R}^{N \times 3}$ of the provided image, and (2) the number of clusters k. It returns: (1) the estimated labels $r \in \mathbb{R}^{N \times k}$ where $r_k^t = 1$ if pixel t belongs to cluster k, otherwise 0, (2) the estimated k centroids $\mu \in \mathbb{R}^{k \times 3}$, and (3) the distortion measure J (i.e., reconstruction error E Eq. (7.3) in textbook) during training. It must also print out the distortion measure (reconstruction error) at each iteration during training to the terminal.

`hw2q4.py` script is used to load image, prepare the data, make calls to the function `kmeans(X, k)`, generate the segmented or compressed version of the original image, and plot the resulting image for each k.

(b) **Summary of results**: For each value of $k = \{3, 5, 7\}$, report the final (i.e., after k-means has converged) segmented image and a plot of the distortion measure (reconstruction error) during training. (This will be 3 segmented images and 3 plots of the loss where the x-axis is the training iteration number and y-xais is the distortion or error value.)

Additional instructions: Code can only be written in Python 3.6+; no other programming languages will be accepted. One should be able to execute all programs from the Python command prompt or terminal. Please specify instructions on how to run your program in the README file.

Each function must take the inputs in the order specified in the problem and display the textual output via the terminal and plots/figures should be included in the report.

For each part, you can submit additional files/functions as needed. In your code, you can only use machine learning libraries such as those available from scikit-learn as specified in the problem description. You may use libraries for basic matrix computations and plotting such as numpy, pandas, and matplotlib. Put comments in your code so that one can follow the key parts and steps in your code.

Your code must be runnable on a CSE lab machine (e.g., csel-kh1260-01.cselabs.umn.edu). One option is to SSH into a machine. Learn about SSH at these links: `https://cseit.umn.edu/knowledge-help/learn-about-ssh`, `https://cseit.umn.edu/knowledge-help/choose-ssh-tool`, and `https://cseit.umn.edu/knowledge-help/remote-linux-applications-over-ssh`.

Instructions

Follow the rules strictly. If we cannot run your code, you will not get any credit.

- **Things to submit**

1. hw2.pdf: A document which contains the solution to Problems 1, 2, 3, and 4 including the summary of results for problems 2, 3, 4. This document must be in PDF format (no word, photo, etc. is accepted). If you submit a scanned copy of a hand-written document, make sure the copy is clearly readable, otherwise no credit may be given.

2. Python code for Problem 2 (must include the required `MultiGaussClassify.py` and `hw2q2.py` files).
3. Python code for Problem 3 (must include the required `myPCA.py` and `hw2q3.py` files).
4. Python code for Problem 4 (must include the required `my_kmeans.py` and `hw2q4.py` files).
5. README.txt: README file that contains your name, student ID, email, instructions on how to run your code, any assumptions you are making, and any other necessary details.
6. Any other files, except the data, which are necessary for your code (such as package dependencies like a requirements.txt or yml file).

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems, but each person must write up the final solutions individually. You need to list in the README.txt which problems were a collaborative effort and with whom. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help on online,
- Look up things/post on sites like Quora, StackExchange, etc.