Ver. 1.0.2

IAML – INFR11182 (LEVEL 11): Assignment #2

Due on Monday, November 23, 2020 @ 16:00

NO LATE SUBMISSIONS

IMPORTANT INFORMATION

N.B. *This document is best viewed on a screen as it contains a number of (highlighted) clickable hyperlinks.*

It is very important that you read and follow the instructions below to the letter. You will be deducted marks for not adhering to the advice below.

Good Scholarly Practice: Please remember the University requirement regarding all assessed work for credit. Details about this can be found at:

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Specifically, this assignment should be your own individual work. We will employ tools for detecting misconduct. Moreover, please note that Piazza is **NOT** a forum for discussing the solutions of the assignment. You may, in exceptional circumstances, ask **private** questions to the instructors if you deem that something may be incorrect, and if we feel that the issue is justified, we will send out an announcement.

General Instructions

- There are two versions of this assignment. One for INFR10069 (level 10) and the other for INFR11182 (level 11). The level 11 version has some additional parts. **MAKE SURE you are doing the assignment that corresponds to the course you are registered on; you can check this on EUCLID.**
- We will use the IAML Learn page for any announcements, updates, and FAQs on this assignment. Please visit the page frequently to find the latest information.

- You should use Python for implementing your solutions as this will standardise the output and also provide a consistent experience with the labs. Set up your environment as specified in the Labs. **It is VERY IMPORTANT that you use the exact same package versions as those specified in the requirements file from the labs!** Using the correct environment (i.e. `py3iaml`) is necessary to ensure that your outputs are consistent with the expected solutions. The correct package versions are specified here.

- If running `import sklearn; print(sklearn.__version__)` in your Jupyter Notebook does not print the package version `0.19.1`, then you are *not* using the correct environment.

- This assignment consists of multiple questions. **MAKE SURE to use the correct dataset for each question.**

- This assignment accounts for 30% of your final grade for this course and is graded based on a written report (compiled from a latex template which we provide). You should submit not only the report, but also the code you wrote for the assignment. Since this is not a programming course, your code is not graded, but those answers in your report without corresponding code will not be graded.

- Some of the topics in this coursework are covered in weeks 7 and 8 of the course. Focus first on questions on topics that you have covered already, and come back to the other questions as the lectures progress.

- The criteria on which you will be judged include the quality of the textual answers and/or any plots asked for.

- Read the instructions carefully, answering what is required and only that. Keep your answers brief and concise. Specifically, **for textual answers**, the size of the text-box in the latex template will give you an idea of the **maximum** length of your answer. You do **not need** to fill in the whole text-box but you will be **penalised** if you go over. This does not apply to figure-based answers.

- For answers involving figures, make sure to clearly label your plots and provide legends where necessary. You will be penalised if the visualisations are not clear.

- For answers involving numerical values, use correct units where appropriate and format floating point values to an appropriate number of decimal places.

- Some experiments may take long time (e.g. more than 10 minutes) to finish. It will be a good idea that you test your code with a small number of samples for debugging. You should use the whole data when your write your report.

Submission Mechanics

Important: ***You must submit this assignment by Monday, 23 November 2020 at 16:00. We do not accept Late Submissions for this coursework, except with an extension approved by the Extensions and Special Circumstances (ESC)***

team. No extensions will be granted, except where specifically permitted by a student's Schedule of Learning Adjustments. Please see the ITO Website for further details.

- Your submission consists of a report (in PDF) and code. Marking is done based on the report submitted, whereas submitted code is mainly used to check your own individual work.

- We will use the Gradescope submission system for uploading PDF report files, and the Learn system for uploading code files. Information describing how to upload your completed assignment will be made available on the IAML Learn page.

- You should clone or download the Assignment Repository from https://github.com/uoe-iaml/INFR11182-2020-CW2.

  This contains:

  1. The data you will need for the third part of assignment under the `data` directory.
  2. The helper functions under `helpers` directory, which you should use to load data in your program.
  3. The Python template files under `templates` directory. They are `iaml01cw2_q1.py`, `iaml01cw2_q2.py`, `iaml01cw2_q3.py`, and optionally `iaml01cw2_my_helpers.py`, which define the functions you should write in the assignment.
  4. Two `tex` files, `Assignment_2.tex` and `style.tex`. These provide the template for you to fill out the assignment questions. In particular, the template forces your answers to appear on separate pages and also controls the length of textual answers.

- You should modify the `Assignment_2.tex` template by:

  1. Uncommenting and specifying your student number at the top of the document (compilation will automatically fail if you forget to do this). Remove the '%' and enter your student number e.g.

     ```
     \newcommand{\assignmentAuthorName}{s1234567}
     ```

  2. Filling in the answers in the provided `answerbox` environment.

  DO NOT modify anything else in the template and certainly **DO NOT** edit the style file. **We reserve the right to not mark assignments which do not adhere to the template**.
  Note that the questions shown in `Assignment_2.tex` may not be the full copy of the original ones in this assignment instruction document, but just short versions of them to save space.

- Your should submit three files of code, one for each of Questions 1, 2 and 3, using the template files specified in the following table. Note that you should use the function names specified in the templates for each question. You should not change the function names or file names. In case you use helper functions of your own, you should also submit them as a single file with the file name specified in the table. Jupyter Notebook files (.ipynb) can be submitted instead of Python (.py) files. If it is the case, you should replace “`.py`” with “`.ipynb`”.

File name	Descriptions
`iaml01cw2_q1.py`	Code for Question 1
`iaml01cw2_q2.py`	Code for Question 2
`iaml01cw2_q3.py`	Code for Question 3
`iaml01cw2_my_helpers.py`	Code for helper functions (optional)

Latex Tips

- To fill in text answers, you can modify the text inside the `answerbox`:

  ```
  \begin{answerbox}{5em}
    Your answer here
  \end{answerbox}
  ```

  with your answer (replacing ‘Your answer here’):

  ```
  \begin{answerbox}{5em}
    Steam locomotives were first developed in the United Kingdom
    during the early 19th century and used for railway transport
    until the middle of the 20th century.
  \end{answerbox}
  ```

  which, when compiled gives:

  > Steam locomotives were first developed in the United Kingdom during the early 19th century and used for railway transport until the middle of the 20th century.

- To add an image, you can use:

  ```
    \begin{answerbox}{18em}
      This image shows a train.
      \begin{center}
        \includegraphics[width=0.7\textwidth]{stock_image.jpg}
      \end{center}
    \end{answerbox}
  ```

  which will be compiled to:

This image shows a train.

Make sure that you specify the correct path to your image. For example. if your image was stored in a directory called `results`, you would change the relevant line to read:

```
\includegraphics[width=0.7\textwidth]{results/stock_image.jpg}
```

You can find more information about inserting images into latex documents here.

- You can also add two images side-by-side:

```
\begin{answerbox}{18em}
  Below we see two trains.

  \begin{center}
    \begin{tabular}{ll}
      \includegraphics[width=0.4\textwidth]{stock_image.jpg}
      &
        \includegraphics[width=0.4\textwidth]{stock_image.jpg}
    \end{tabular}
  \end{center}
\end{answerbox}
```

which will be compiled to:

Below we see two trains.

- To add an inline equation, you can use the ‘$’ symbol to write:

```
\begin{answerbox}{3em}
  I am using the following model, $y = \mathbf{x}^T\mathbf{w}$.
\end{answerbox}
```

which compiles to:

I am using the following model, $y = \mathbf{x}^T\mathbf{w}$.

- To add a table for numerical results you can use:

```
\begin{answerbox}{7em}
  Results are presented in the table below.

\begin{center}
  \begin{tabular}{ |c|c|c| } \hline
    Parameter Value & Train Accuracy & Test Accuracy \\ \hline
    1 & 10.1\% & 9.1\% \\
    2 & 12.5\% & 10.1\% \\\hline
  \end{tabular}
\end{center}
\end{answerbox}
```

which compiles to:

Results are presented in the table below.

Parameter Value	Train Accuracy	Test Accuracy
1	10.1%	9.1%
2	12.5%	10.1%

You can find more information about tables in latex here.

- For a small number of questions we may ask you to show your code in your report. You can include code as an image, but if you prefer you can use the following command:

  ```
  \begin{answerbox}{5em}
  \begin{verbatim}
  import numpy as np
  mean_time = 10.0
  print('mean time', mean_time)
  \end{verbatim}
  \end{answerbox}
  ```

  which, when compiled gives:

  ```
  import numpy as np
  mean_time = 10.0
  print('mean time', mean_time)
  ```

- Once you have filled in all the answers, compile the latex document to generate the PDF that you will submit. You can use Overleaf, your favourite latex editor, or just run `pdflatex Assignment_2.tex` twice on a DICE machine to compile the PDF.

week8

Question 1 : (41 total points) Image data analysis with PCA

In this question we employ PCA to analyse image data

Here we will investigate the use of principal component analysis (PCA) for image data. Image data are made up of H-by-W pixels, where H and W denote the height and width, respectively. For simplicity, we assume a gray-scale image. Let p_{ij} denote the pixel value at a grid point (i, j), $1 \leq i \leq H$, $1 \leq j \leq W$, where p_{11} corresponds to the the pixel at the top-left corner and p_{HW} to the one at the bottom-right corner. We assume that p_{ij} takes an integer value between 0 and 255 (i.e. 8-bit coding). In computers, we store an image of $\{p_{ij}\}$ in a D-dimensional vector, $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, where $D = H \times W$, and x_1 corresponds to p_{11} and x_D to p_{HW}.
We here use the Fashion-MNIST[1] data set of images – there are 10 classes, [2] 6000 training samples and 1000 test samples per class. Each sample is a 28-by-28 gray-scale image. At first, download the data set to your local directory from either the original GitHub repository, 'fashion-mnist/data/fashion',[3] or the mirrored directory[4] in the Informatics DICE. Note that the data set consists of four files,

```
t10k-images-idx3-ubyte.gz    t10k-labels-idx1-ubyte.gz
train-images-idx3-ubyte.gz   train-labels-idx1-ubyte.gz
```

Once you have downloanded them to your local directory, load the data set in the following manner in your code.

```
Xtrn, Ytrn, Xtst, Ytst = load_FashionMNIST(DataPath)
```

where you should replace *DataPath* with the actual path (i.e. directory) where you keep the data set files. `load_FashionMNIST()` can be found in the helper file, `iaml01cw2_helpers.py`. `Xtrn` and `Ytrn` are training data (60000 samples) and corresponding labels, whereas `Xtst` and `Ytst` are test data (10000 samples) and labels. There are 10 classes represented as an integer value between 0 and 9. As you will see, `Xtrn` is a numpy.ndarray of 60000-by-784. The i-th row vector of `Xtrn` corresponds to the pixel values of i-th image of 28-by-28 pixels, where i-element in `Ytrn` shows the corresponding class label.

Before you work on the tasks shown below, apply the following normalisation of 4 steps to the data. Please note that we do not employ the commonly used standardisation or Z-score normalisation for PCA in this coursework.

Step 1 Make a back up of `Xtrn` and `Xtst` by copying `Xtrn` to `Xtrn_orig` and `Xtst` to `Xtst_orig`.

Step 2 Divide each element of `Xtrn` and `Xtst` by 255.0. (NB: `Xtrn` and `Xtst` are over-written as a result.)

[1] https://github.com/zalandoresearch/fashion-mnist

[2] Class labels (in integer values) and their class names are as follows. 0:T-Shirt/Top, 1:Trouser, 2:Pullover, 3:Dress, 4:Coat, 5:Sandals, 6:Shirt, 7:Sneaker, 8:Bag, 9:Ankle boots.

[3] https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion

[4] https://ifile.inf.ed.ac.uk/?path=/afs/inf.ed.ac.uk/group/teaching/iaml/cwk2/fashion-mnist-data

Question 1 continued on next page...

Step 3 Calculate the mean value of Xtrn for each dimension (i.e. column) and store the result in a vector, Xmean, whose shape is (784,).

Step 4 Subtract Xmean from each row of Xtrn and Xtst, and store the result in Xtrn_nm and Xtst_nm, respectively.

1.1 (3 points) Once you have applied the normalisation from Step 1 to Step 4 above, report the values of the first 4 elements for the first training sample in Xtrn_nm, i.e. Xtrn_nm[0,:] and the last training sample, i.e. Xtrn_nm[-1,:].

1.2 (4 points) Using Xtrn and Euclidean distance measure, for each class, find the two closest samples and two furthest samples of that class to the mean vector of the class.
Display the images of the mean vectors and samples you found in a 10-by-5 grid, where the top row corresponds to the images for class 0 and the bottom to those for class 9. Each column corresponds to the image of mean vector, the images of the 1st and 2nd closest and 2nd and 1st furthest samples to the mean vector for that class, respectively (from left to right). For each image sample, you should provide the class number and the sample number in the data set. Note that we use 0-based indexing.
Explained your findings briefly.
Hint: To display an image of a vector, reshape the vector to a 28-by-28 array, and use matplotlib.pyplot.imshow() with 'gray_r' colormap. To display a total of 50 images in a 10-by-5 grid, you could use subplot(), but it is also fine that you plot each image and save it in a file separately, and you import them in latex and arrange them using the latex tabular environment when you write a report.

1.3 (3 points) Apply Principal Component Analysis (PCA) to the data of Xtrn_nm using sklearn.decomposition.PCA, and report the variances of projected data for the first five principal components in a table. Note that you should use Xtrn_nm instead of Xtrn.

1.4 (3 points) Plot a graph of the cumulative explained variance ratio as a function of the number of principal components, K, where $1 \leq K \leq 784$. Discuss the result briefly.

1.5 (4 points) Display the images of the first 10 principal components in a 2-by-5 grid, putting the image of 1st principal component on the top left corner, followed by the one of 2nd component to the right. Discuss your findings briefly.

Question 1 continued on next page. . .

1.6 (5 points) Using `Xtrn_nm`, for each class and for each number of principal components $K = 5, 20, 50, 200$, apply dimensionality reduction with PCA to the first sample in the class, reconstruct the sample from the dimensionality-reduced sample, and report the Root Mean Square Error (RMSE) between the original sample in `Xtrn_nm` and reconstructed one.

You should report this using a table, in which each row corresponds to class and each column corresponds to a value of K.

1.7 (4 points) Display the image for each of the reconstructed samples in a 10-by-4 grid, where each row corresponds to a class and each row column corresponds to a value of $K = 5,\ 20,\ 50,\ 200$.

Note that you should add `Xmean` to each reconstructed sample to display the corresponding image. Discuss your findings briefly.

1.8 (4 points) Plot all the training samples (`Xtrn_nm`) on the two-dimensional PCA plane you obtained in Question 1.3, where each sample is represented as a small point with a colour specific to the class of the sample. Use the 'coolwarm' colormap for plotting.

Give comments on the separation of the classes, and explain your findings briefly.

1.9 (4 points) We here compare PCA with Discrete Cosine Transform (DCT), which is another technique for dimensionality reduction with cosine functions at different frequencies. PCA transforms a D-dimensional vector, $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$ to a K-dimensional vector, $\boldsymbol{z} = (z_1, \ldots, z_K)$ by projecting $\boldsymbol{x}$ to a vector space spanned by K eigenvectors (i.e. principal components), where $1 \leq K \leq D$. The k-th element of $\boldsymbol{z}$ is given by the inner (dot) product between $\boldsymbol{x}$ and $\boldsymbol{v}_k$:

$$z_k = \boldsymbol{x} \cdot \boldsymbol{v}_k = \sum_{i=1}^{D} x_i v_{ki}$$

where $\boldsymbol{v}_k$ is the k-th eigenvector.[5] In DCT, $\boldsymbol{v}_k$ is replaced with a basis vector $\boldsymbol{c}_k$, whose i-th element is given as

$$c_{ki} = \cos\left(\frac{\pi(k-1)(2i-1)}{2D}\right)$$

where $1 \leq k \leq K$ and $1 \leq i \leq D$,[6] so that the projection of $\boldsymbol{x}$ with $\boldsymbol{c}_k$ is given as

$$z_k^{(DCT)} = \boldsymbol{x} \cdot \boldsymbol{c}_k = \sum_{i=1}^{D} x_i c_{ki}\ .$$

We are interested in how the vector, $\boldsymbol{c}_k$, looks like as we did for eigenvectors in Question 1.5. Letting $D = 784$, display the image of $\boldsymbol{c}_k$ for $k = 1, 2, 3$, where you display each vector as a 28-by-28 image. Are they similar to those of principal components? Explain your findings briefly.

[5] $\boldsymbol{x}, \boldsymbol{v}_k$, and $\boldsymbol{z}$ are row vectors.

[6] The original DCT-II for vector transformation is considered in this question for simplicity rather than 2D DCT for two-dimensional data.

Question 1 continued on next page...

1.10 (4 points) Apply DCT to `Xtrn_nm` and calculate the variance, $\mathrm{Var}(z_k^{(DCT)})$, for $k = 1, \ldots, D$.
To compare DCT with PCA, on a single graph, plot the cumulative explained variance ratio for both DCT and PCA, using 'red' for DCT and 'blue' for PCA. (Note that you should have basically the same graph for PCA as the one obtained in Question 1.4.)
Hint: Var($z_k^{(DCT)}$) is the variance of the samples transformed or projected with DCT's k-th basis vector $\boldsymbol{c}_k$.

1.11 (3 points) Based on the comparison between DCT and PCA above, discuss their advantages and disadvantages in terms of data compression.

Wk4 and wk5

Question 2 : (25 total points) Logistic regression and SVM

In this question we will explore classification of image data with logistic regression and support vector machines (SVM) and visualisation of decision regions.

Here we will look into logistic regression as an example of linear classifiers and SVM as an example of nonlinear classifiers. The SVM was originally for binary classification, but we consider an extended version to multi-class classification by forming one-to-rest (OvR) classifiers, in which each classifier is trained in such a way that it separates the data of the target class from those of the other classes.
Apart from basic classification experiments, we will visualise decision regions using PCA, in which we consider a two-dimensional plane in the original vector space and visualise how the points on the plane are classified.
We use the same image data set as the one for Question 1. Make sure that you apply the four steps of normalisation described in Question 1 and use the normalised data, i.e. `Xtrn_nm` for training and `Xtst_nm` for testing in this question.

2.1 (3 points) Carry out a classification experiment with multinomial logistic regression, and report the classification accuracy and confusion matrix (in numbers rather than in graphical representation such as heatmap) for the test set.
You should use `sklearn.linear_model_LogisticRegression()` with the default parameters. Make sure that you use `Xtrn_nm` for training and `Xtst_nm` for testing.

2.2 (3 points) Carry out a classification experiment with SVM classifiers, and report the mean accuracy and confusion matrix (in numbers) for the test set.
You should use `sklearn.svm.SVC()` with the default parameters (kernel='rbf', $C = 1.0$, gamma='auto').

2.3 (6 points) We now want to visualise the decision regions for the logistic regression classifier we trained in Question 2.1.
Since it is not possible to visualise the original vector of 784 dimensions, we consider a two-dimensional plane, i.e. cross section, in the original space and visualise decisions regions for the points on that plane, ignoring points outside the plane. We here employ the plane spanned by the first two principal components, where we assume that the plane shares the same origin as the one of the original vector space. Using matplotlib.pyplot.counterf() and 'coolwarm' colormap, plot the decisions regions for the rectangular area of 100-by-100 grid points whose left-upper corner is $(-5\sigma_1, 5\sigma_2)$ and right-bottom corner is $(5\sigma_1, -5\sigma_2)$, where σ_1 and σ_2 denote the standard deviations for the first principal component and second principal component, respectively. You should also use matplotlib.colorbar() to display a colorbar.
Report your findings briefly.
Hint: The projected point $\boldsymbol{z} = (z_1, \ldots, z_D)$ with PCA is given as $\boldsymbol{z} = \boldsymbol{x}V^T$, where V is a D-by-D square matrix, whose i-th row corresponds to the i-th eigenvector. Note that both

$\boldsymbol{z}$ and $\boldsymbol{x}$ represent the same point with different coordinate systems, the original vector space and the one spanned with D eigenvectors, and the two coordinate systems share the same origin. Any points on the 2D-plane spanned with the first two principal components can be expressed as $\boldsymbol{z} = (z_1, z_2, 0, \ldots, 0)$. You can find the corresponding $\boldsymbol{x}$ in the original vector space by $\boldsymbol{x} = \boldsymbol{z}V$.

2.4 (4 points) Using the same method as the one above, plot the decision regions for the SVM classifier you trained in Question 2.2. Comparing the result with that you obtained in Question 2.3, discuss your findings briefly.

2.5 (6 points) We used default parameters for the SVM in Question 2.2. We now want to tune the parameters by using cross-validation. To reduce the time for experiments, you pick up the first 1000 training samples from each class to create `Xsmall`, so that `Xsmall` contains 10,000 samples in total. Accordingly, you create labels, `Ysmall`.

By using a 3-fold cross validation and `Xsmall` only, estimate the classification accuracy of an SVM classifier with RBF kernel, while you vary the penalty parameter C in the range 10^{-2} to 10^{3} (use 10 values spaced equally log space). Set the kernel coefficient parameter gamma to auto for this question.

Plot the mean cross-validated classification accuracy against the regularisation parameter C by using a log-scale for the x-axis. Report the highest obtained mean accuracy score and the value of C which yielded it.

2.6 (3 points) Train the SVM classifier on the whole training set by using the optimal value of C you found in Question 2.5.

Report the classification accuracy on the training set and test set. (Note that you do not use cross validation for this question.)

week7

Question 3 : (32 total points) Clustering and Gaussian Mixture Models

In this question we will explore K-means clustering, hierarchical clustering, and GMMs.

We will look into clustering and mixture models using a multilingual speech data set, in which we cluster the natural language from a given speech sample. Compared with image data whose size is fixed, speech data is sequential and its length is variable by nature, we thus cannot directly apply commonly used clustering algorithms. To tackle the problem, we drastically simplify the task by representing each speech sample by the fixed-dimensional mean vector of parameterised speech features for that sample.
The speech data we use here is a subset of CoVoST2, in which speech data of 22 languages is available. We have converted each audio speech sample (which is normally a few seconds in length) to the simplified representation, i.e. a mean vector, which has 26 elements.[7] The resultant data have been stored in two files, `speech_trn.npz` and `speech_tst.npz`, which are training set and test set, respectively.
Now, load the data sets in the following manner.

```
Xtrn,Ytrn,Xtst,Ytst = load_CoVoST2(DataPath)
```

where you should replace *DataPath* with the actual path (i.e. directory) where you keep the dataset files, `load_CoVoST2()` can be found in the helper file, `iaml01cw2_helpers.py`. The training data, `Xtrn`, is a numpy.ndarray, whose shape is $(22000, 26)$. The i-th row vector represents the feature vector of 26 elements for the i-th speech sample. The same format (but with different numbers of rows) applies to the test data, `Xtst`.[8] `Ytrn[i]` is the label for `Xtrn[i]`, representing the language name coded in an integer value between 0 and 21. The correspondence between language labels and language names can be found in `data/languages.txt`. Instead of using language names, we use "Language ℓ" to denote the language whose label code is ℓ, where $0 \leq \ell \leq 21$.

Note that you should NOT change the order of samples in the data set and you should NOT standardise the data.
When you use Sklearn's function for k-means clustering and Gaussian mixture models, use the parameter `random_state=1`.
Due to the extreme simplification applied to data, you may not be able to obtain good results. You do not need to worry if the result of clustering analysis does not match your intuitions.

3.1 (3 points) Apply k-means clustering on `Xtrn` for $k = 22$, where we use sklearn.cluster.KMeans with the parameters `n_clusters=22` and `random_state=1`. Report the sum of squared distances of samples to their closest cluster centre, and the number of samples for each cluster.

[7] MFCC (mel-frequency cepstral coefficients) analysis was employed. A feature vector of 26 dimensions consist of 12 MFCCs, energy, and 12 delta-MFCCs and delta-energy. Understanding the details of this preprocessing is NOT required in order to answer this question.

[8] Different classes have different numbers of samples from each other.

Question 3 continued on next page. . .

3.2 (3 points) Using the training set only, calculate the mean vector for each language, and plot the mean vectors of all the 22 languages on a 2D-PCA plane, where you apply PCA on the set of 22 mean vectors without applying standardisation. On the same figure, plot the cluster centres obtained in Question 3.1.

You should format the figure nicely so that language and cluster information is clear. Language information should be shown in terms of name, its abbreviation, or number.

Are the mean vectors and cluster centres similar? Discuss your findings briefly.

3.3 (3 points) We now apply hierarchical clustering on the training data set to see if there are any structures in the spoken languages.

To avoid the analysis being cluttered, we represent the data of each language by the mean vector for that language, so that we only use 22 mean vectors for this analysis instead of using all the data in the training set. Now, using scipy.cluster.hierarchy, carry out hierarchical clustering with the Ward's linkage and display the dendrogram with an option of orientation='right'. Provide appropriate labels to indicate the language of each leaf node.

Discuss your findings.

3.4 (5 points) We here extend the hierarchical clustering done in Question 3.3 by using multiple samples from each language.

To that end, apply k-means clustering with the parameters `n_clusters=3` and `random_state=1` to the training data for each language to find 3 cluster centres, which will result in $3 \times 22 = 66$ vectors in total.

Carry out hierarchical clustering with each of the linkage methods, 'ward', 'single', and 'complete' on the 66 vectors.

Plot a dendrogram for each method, and discuss the results briefly.

3.5 (6 points) We now consider Gaussian mixture model (GMM), whose probability distribution function (pdf) is given as a linear combination of Gaussian or normal distributions, i.e.,

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} p_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

where $N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is a Gaussian distribution of $\boldsymbol{x}$ for k-th component with the mean vector $\boldsymbol{\mu}_k = (\mu_{k1}, \ldots, \mu_{kD})$ and covariance matrix $\boldsymbol{\Sigma}_k = (\sigma_{ij}^{(k)})$, p_k is the weight for k-th distribution or component such that $\sum_{k=1}^{K} p_k = 1$, and K is the number of mixture components. $N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is given as

$$N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\right).$$

The covariance matrix $\boldsymbol{\Sigma}$ is a D-by-D symmetric matrix, whose (i, j) element, σ_{ij}, is the covariance between variables x_i and x_j. If the number of training samples is not as sufficiently large as D, $\boldsymbol{\Sigma}$ becomes singular and non-invertible, which makes it impossible to calculate the pdf. *Naive Bayes* is one of the techniques to mitigate the problem, in

Question 3 continued on next page…

which you assume conditional independence among variables $\boldsymbol{x}$. As a result, the covariance $\sigma_{ij} = 0$ for $i \neq j$, meaning all non-diagonal elements of $\boldsymbol{\Sigma}$ are zeros and only the diagonal elements can be non zeros. Such a covariance matrix is referred to as "*diagonal-covariance matrix*" as opposed to "*full-covariance matrix*" that does not use the assumption.
We now employ Gaussian Mixture Model (GMM) in sklearn to model the data for a language.
Using the data for Language 0 only, for each type of covariance matrix, 'diagonal covariance' and 'full covariance', and for each number of mixture components, $K = 1, 3, 5, 10, 15$, train a GMM on the training data, and obtain the per-sample average log-likelihood on the training data and test data for Language 0.
Report the result in a single graph and a table, and discuss your findings.

3.6 (5 points) The training of GMM employs the Expectation-Maximisation (EM) algorithm, which incrementally optimises model parameters - mean vectors, covariance matrices, and component weights,
Here were are interested in how effective the optimisation is compared to a simple Gaussian mixture model that does not use the EM, but it finds the mean vector and covariance matrix of Gaussian distribution for each class based on the k-means clustering, and estimates the weights of mixture components based on the number of samples assigned to each cluster.
Write code for the following two functions for this simple Gaussian mixture model, and show the code in your report. For simplicity, use diagonal covariance instead of full covariance. Note that we fit a Gaussian distribution to the samples assigned to each cluster, using the maximum likelihood parameter estimation. Use the parameter `random_state=1` when you use k-means clustering. For a multivariate Gaussian distribution, use scipy.stats.multivariate_normal.

`SimpleGMM_train(X, n_components)` :
Estimate model parameters from samples in `X`, and return mean vectors (ndarray, shape (n_components, n_features)), a vector of diagonal elements of covariance matrix (ndarray, shape (n_components, n_features)), and weights of mixture components (ndarray, shape (n_components,)).

`SimpleGMM_eval(X, Ms, Dcovs, Pk)` :
Compute the log-likelihood of each sample in `X`, and return the log-likelihoods in an ndarray (shape (n_samples,)). `Ms`, `Dcovs`, and `Pk` correspond to the return values of `SimpleGMM_train()`.

3.7 (3 points) Now train the SimpleGMM on the training set for Language 0, and report the weights of mixture components for each $K = 1, 5, 10$.

3.8 (4 points) Using the data for Language 0 and using the SimpleGMM you obtained in Question 3.7, report, in a table, per-sample average log-likelihood on the training data and test data for $K = 1, 3, 5, 10, 15$.
Discuss the results you obtained, comparing with those obtained in Question 3.5.