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

**In this question we employ PCA to analyse image data**

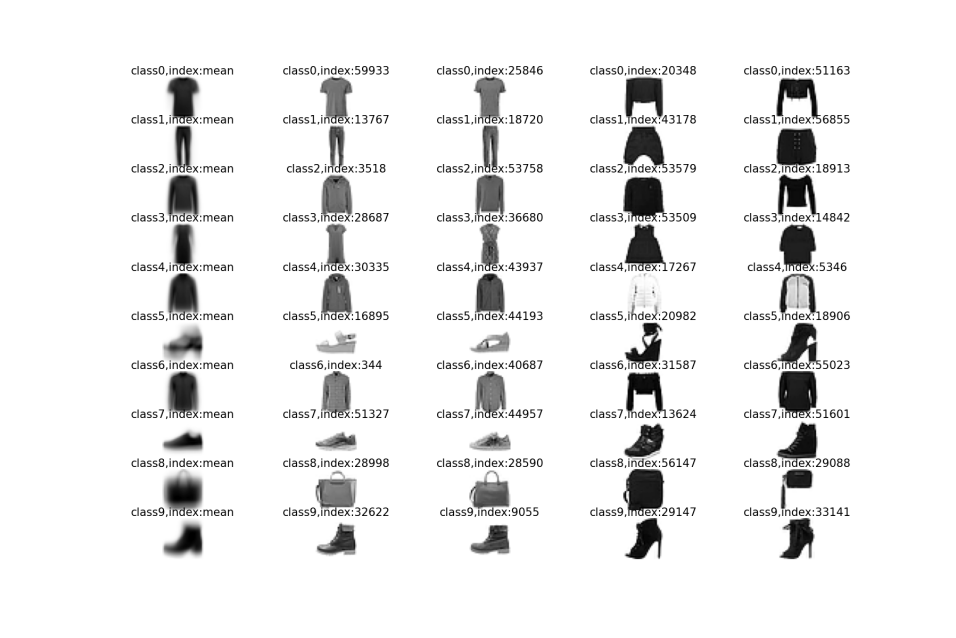
**1.1** (3 points) Once you have applied the normalisation from Step 1 to Step 4 above, report the values of the first 4 elements for the first training sample in `Xtrn_nm`, i.e. `Xtrn_nm[0,:]` and the last training sample, i.e. `Xtrn_nm[-1,:]`.

The first 4 elements for the first training sample in `Xtrn_nm[0,:]`:
array([-3.137e-06, -2.268e-05, -1.180e-04, -4.071e-04])

The first 4 elements for the last training sample:
array([-3.137e-06, -2.268e-05, -1.180e-04, -4.071e-04])

*Question 1 continued on next page...*

**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.




observations:
Every row's samples in the image are similar as they are in the same class.
The samples correspond to the image of mean vector are blurred, as these samples are not really existed in the training data.

**1.3** (3 points) Apply Principal Component Analysis (PCA) to the data of `Xtrn_nm` using sklearn.decomposition.PCA, and find the cumulative explained variance.

The first five principal components of explained variances are

| | | | | |
|---|---|---|---|---|
| 19.810 | 12.112 | 4.106 | 3.382 | 2.625 |

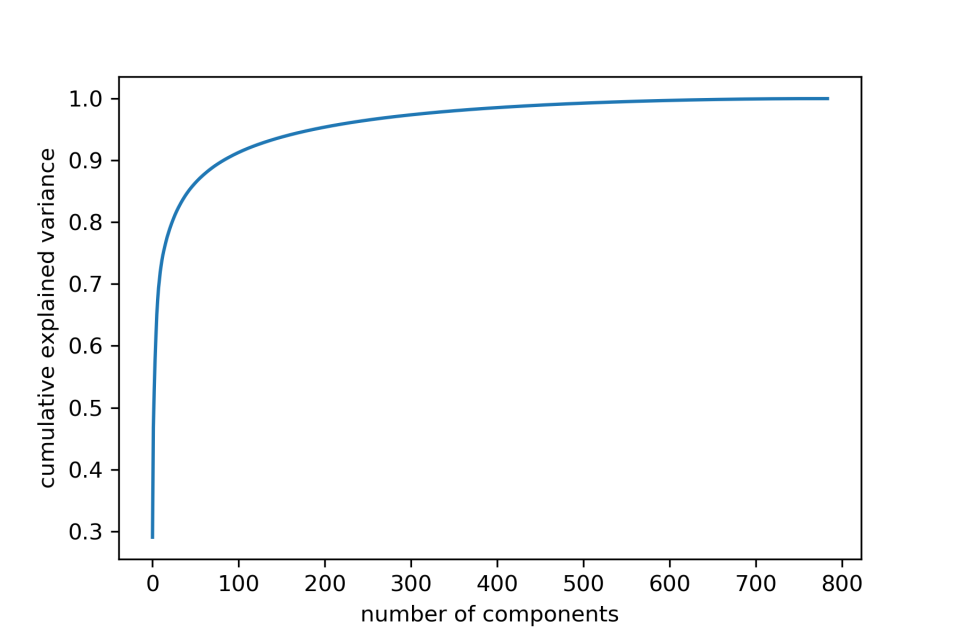
Hence, the cumulative explained variance of the first five principal components is 42.035 As when we run the code

```
pca.explained_variance_.cumsum()[0:5]
```

we got cumulative explained variances [19.810 31.922 36.028 39.410 42.035]

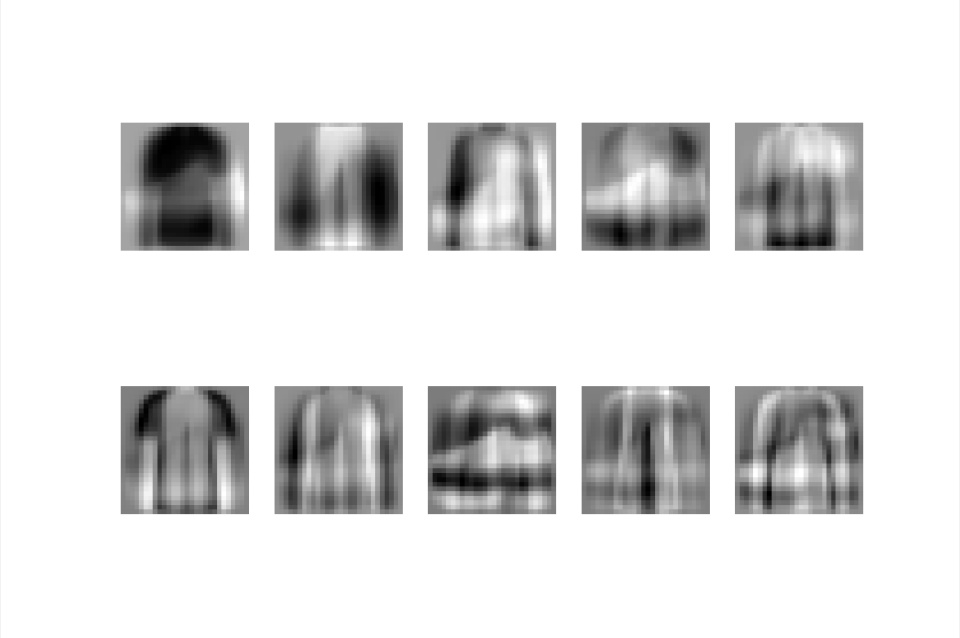
*Question 1 continued on next page...*

**1.4** (3 points) Plot a graph of the cumulative explained variance ratio. Discuss the result briefly.




This curve quantifies how much of the total 784-dimensional variance ratio is contained. We see that the first 100 components contain approximately 90% of the variance, which means that we could preserve about 90% of information when doing 100-dimensional projection. This curve shows the data information preserved when doing projection to different dimensions.

**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.



We know that these images could be called as "eigenfaces". They are represent vector as a linear combination of principal components. These eigenvectors seem to be associated with the shirt and shoes or in other certain features like pants etc.

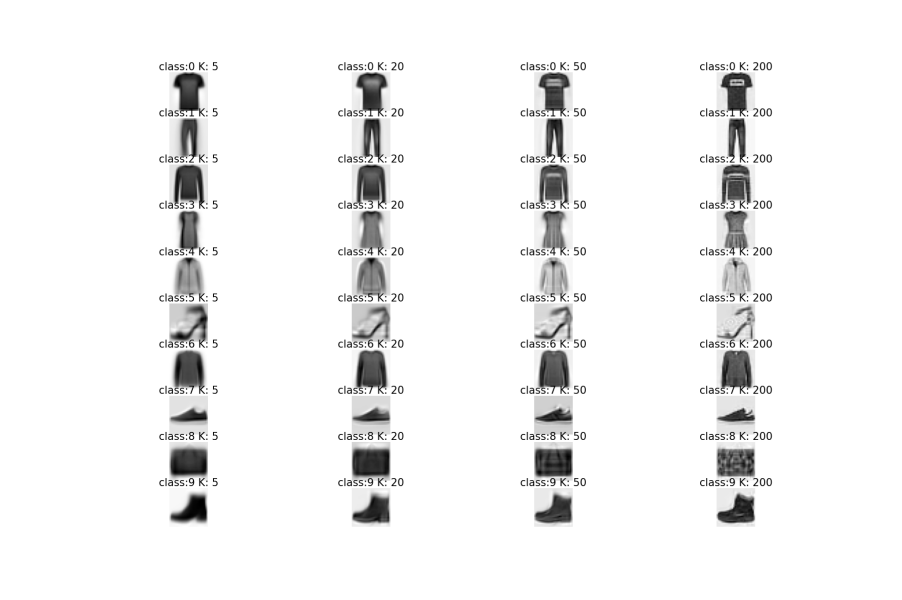
**1.6** (5 points) Using `Xtrn_nm`, for each class and for each number of principal components $K = 5, 20, 50, 200$, apply dimensionality reduction with PCA to the first sample in the class, reconstruct the sample from the dimensionality-reduced sample, and report the Root Mean Square Error (RMSE) between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

The RMSE values under different class and principal components are below

| K / class numbers | 5 | 20 | 50 | 200 |
|---|---|---|---|---|
| 0 | 0.256 | 0.150 | 0.127 | 0.061 |
| 1 | 0.198 | 0.140 | 0.095 | 0.038 |
| 2 | 0.199 | 0.146 | 0.124 | 0.080 |
| 3 | 0.146 | 0.107 | 0.083 | 0.056 |
| 4 | 0.118 | 0.103 | 0.088 | 0.047 |
| 5 | 0.181 | 0.159 | 0.143 | 0.089 |
| 6 | 0.129 | 0.096 | 0.072 | 0.046 |
| 7 | 0.166 | 0.128 | 0.107 | 0.064 |
| 8 | 0.223 | 0.145 | 0.124 | 0.091 |
| 9 | 0.184 | 0.151 | 0.122 | 0.072 |

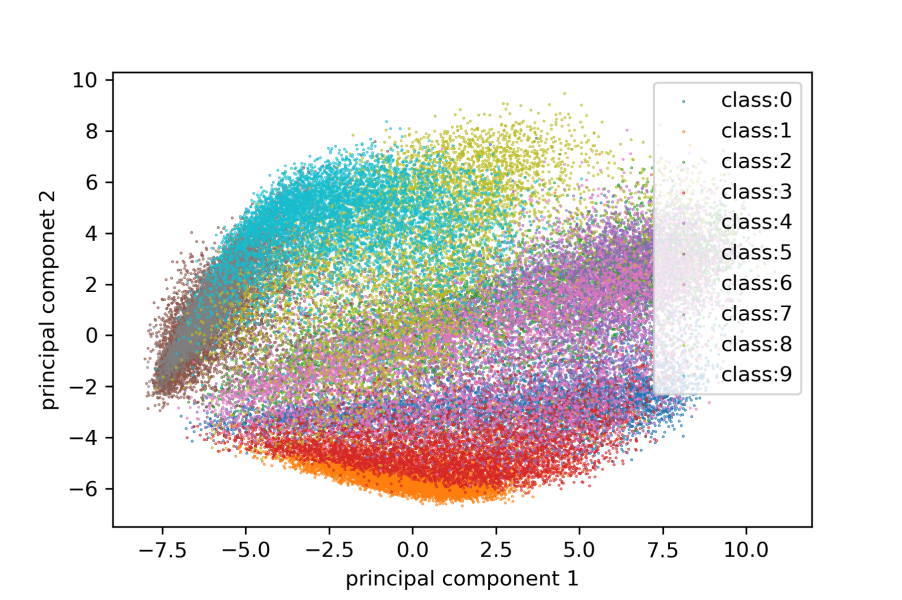
*Question 1 continued on next page...* 

**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 $K = 5,\ 20,\ 50,\ 200$.



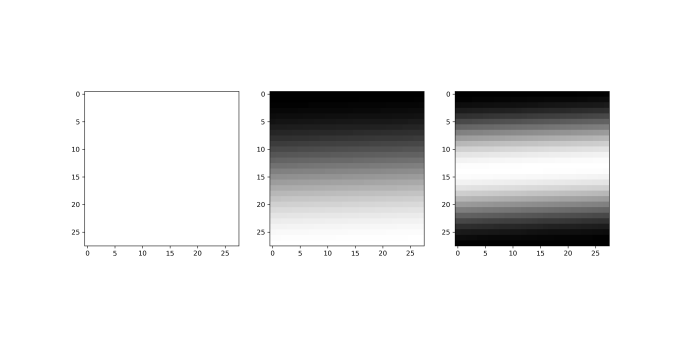

Observation: the reconstructed images for each class with low **K** are blurred and keep low resolution. What's more, with the increase of **K**, the resolution of images is increasing and those images are clearer. As discussed in 1.4, for **K** with low value, when doing projection, it will preserve less data information compared with high **K**. Hence those high **K** images will be clearer.

**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.



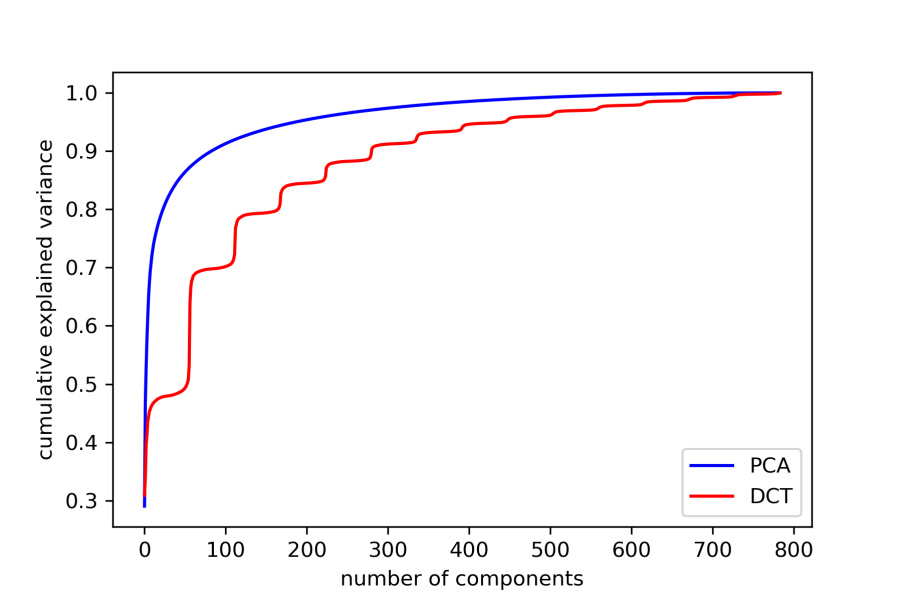

separation of the classes: These points corresponding to each class are in some specific area in this axes system, each class' points are in their specific area, and these area are different from each other. The class separation is based on previous analysis.
findings: as discussed above, every class corresponds to some specific area, they are different but still have some similarity. For example, class 5 and class 9 both represent shoes for different categories, their areas overlap in small part and have little difference. However, for class 5 and class 3, one represents shoes, another represents lady's coat, the area their points stay are obvious different.

**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.



From the images comparison for 1.9 and 1.5, the images for $c_k$ with different k are different with principal components images in 1.5.
DCT has the properties of energy concentration and separability. When K=1, from the formula we know that c_k is always 1, so the image is blank. When K=2, the data is concentrated in the area of upper part of the image. For K=3, the data is concentrated on the upper and bottom part of the image.
Compared with the principal components in PCA,the c_k in DCT is not decided by the data value.

**1.10** (4 points) Apply DCT to `Xtrn_nm` and calculate the variance, $\text{Var}(z_k^{(DCT)})$, for $k = 1, \ldots, D$.



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

PCA pros: -it allows estimating probabilities in high-dimensional data
- it process data compression fast, takes small storage consume.
PCA cons: -it is disastrous for tasks with fine-grained class
-it is too expensive for many applications(Twitter,web)
DCT pros: -It has the ability to pack most information in fewest coefficients, the DCT calculation speed is faster than PCA
-It has been implemented in single integrated circuit.
DCT cons: -It is not lossless transformation, because the DCT coefficients are some irrational Numbers.
-The compression ratio may cause distortion when quantizing

# 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.**

**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.

The classfication accuracy is:0.843

The confusion matrix is below:

| True \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 819 | 3 | 15 | 50 | 7 | 4 | 90 | 1 | 11 | 0 |
| 1 | 5 | 953 | 4 | 27 | 5 | 0 | 3 | 1 | 2 | 0 |
| 2 | 27 | 4 | 731 | 11 | 133 | 0 | 82 | 2 | 9 | 1 |
| 3 | 31 | 15 | 14 | 866 | 33 | 0 | 37 | 0 | 4 | 0 |
| 4 | 0 | 3 | 115 | 38 | 760 | 2 | 72 | 0 | 10 | 0 |
| 5 | 2 | 0 | 0 | 1 | 0 | 911 | 0 | 56 | 10 | 20 |
| 6 | 147 | 3 | 128 | 46 | 108 | 0 | 539 | 0 | 28 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 936 | 1 | 31 |
| 8 | 7 | 1 | 6 | 11 | 3 | 7 | 15 | 5 | 945 | 0 |
| 9 | 0 | 0 | 0 | 1 | 0 | 15 | 1 | 42 | 0 | 941 |

*Question 2 continued on next page...*

**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.

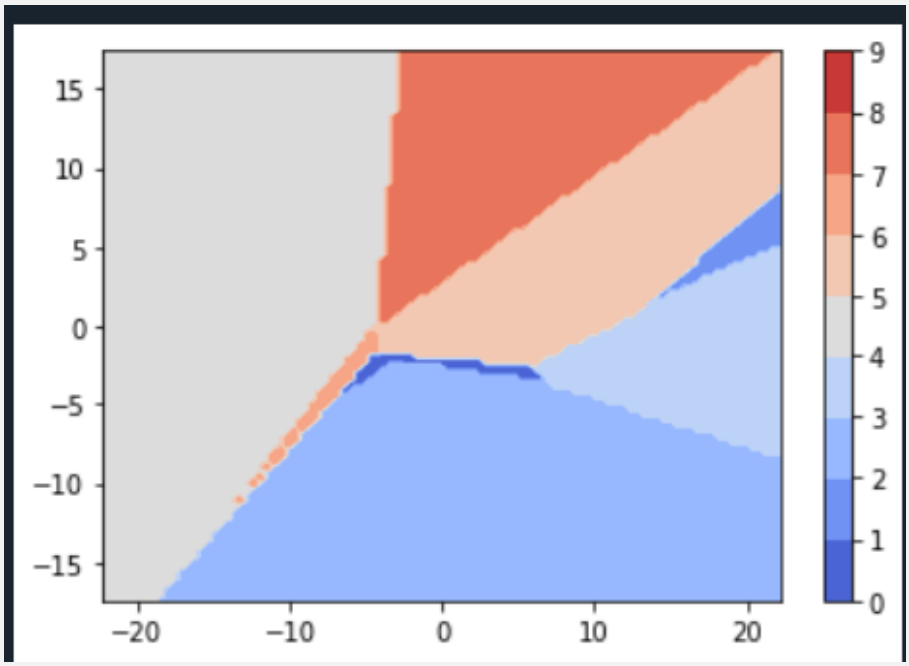
The mean accuracy is 0.846

The confusion matrix is below:

| True \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 845 | 2 | 8 | 51 | 4 | 4 | 72 | 0 | 14 | 0 |
| 1 | 4 | 951 | 7 | 31 | 5 | 0 | 1 | 0 | 1 | 0 |
| 2 | 15 | 2 | 748 | 11 | 137 | 0 | 79 | 0 | 8 | 0 |
| 3 | 32 | 6 | 12 | 881 | 26 | 0 | 40 | 0 | 3 | 0 |
| 4 | 1 | 0 | 98 | 36 | 775 | 0 | 86 | 0 | 4 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 914 | 0 | 57 | 2 | 26 |
| 6 | 185 | 1 | 122 | 39 | 95 | 0 | 533 | 0 | 25 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 34 | 0 | 925 | 0 | 41 |
| 8 | 3 | 1 | 8 | 5 | 2 | 4 | 13 | 4 | 959 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 22 | 0 | 47 | 1 | 930 |

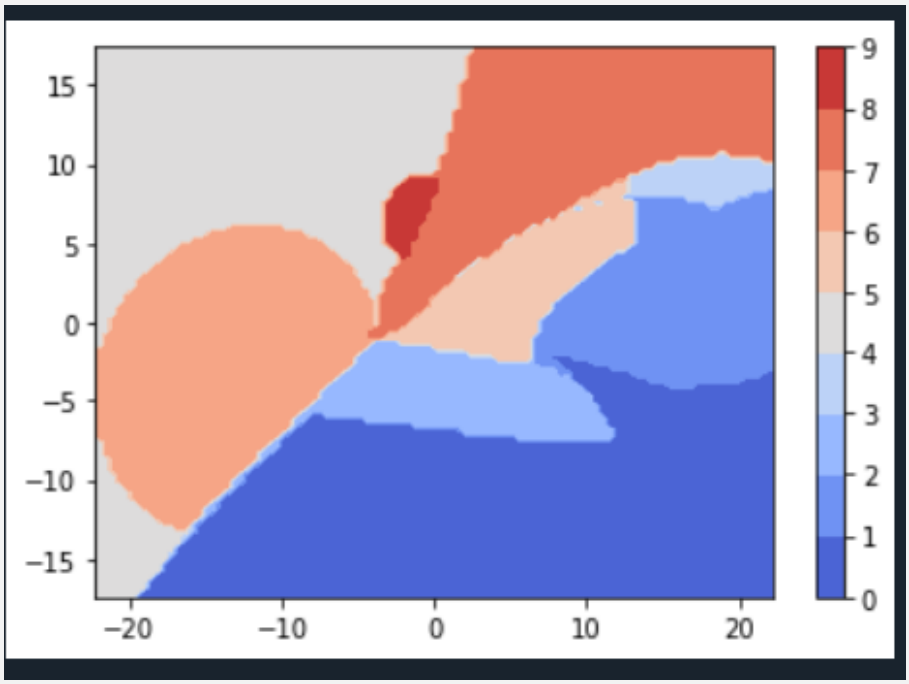
*Question 2 continued on next page...*

**2.3** (6 points) We now want to visualise the decision regions for the logistic regression classifier we trained in Question 2.1.



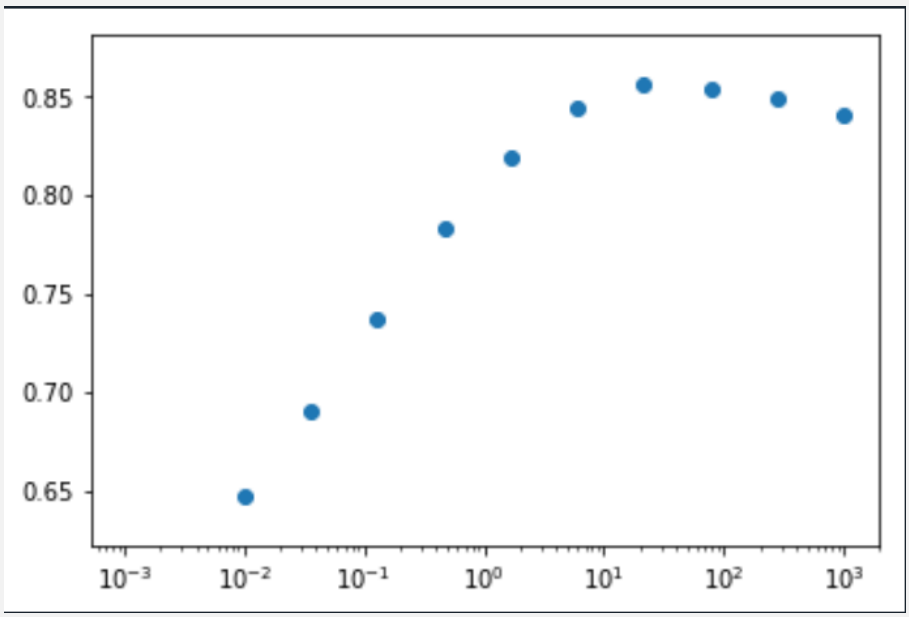

From the image we get, we find that it only has 8 classes, it only predict 8 classes for the data we feed into. And for class 0 and class 7, it predicts little instances, the decision regions are small.

**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.




The biggest difference is the SVM classifier predicts 9 classes but Logistic regression classifier 8. And the decision regions for class 0 and class 7 in 2.4 is larger compared with those in 2.3. The decision regions for every classes in 2.3 and 2.4 are approximated stay in similar specific area, but the region size are different. The decision regions for the SVM classifier is more detailed compared with one for logistic regression.

**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`.




The highest obtained mean accuracy score is 0.857, which corresponds C=21.54

**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.

classification accuracy on the training set is 0.908
on test set is 0.877

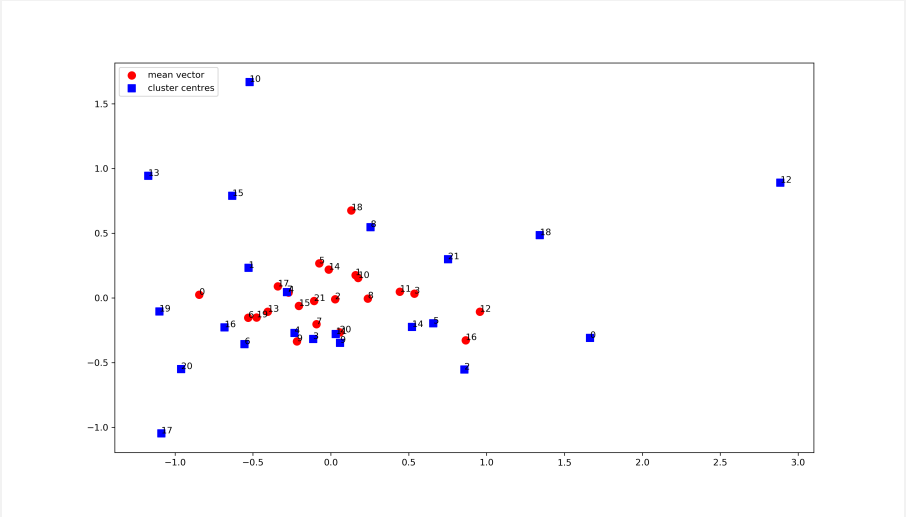
# Question 3 : (32 total points) Clustering and Gaussian Mixture Models

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

**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.

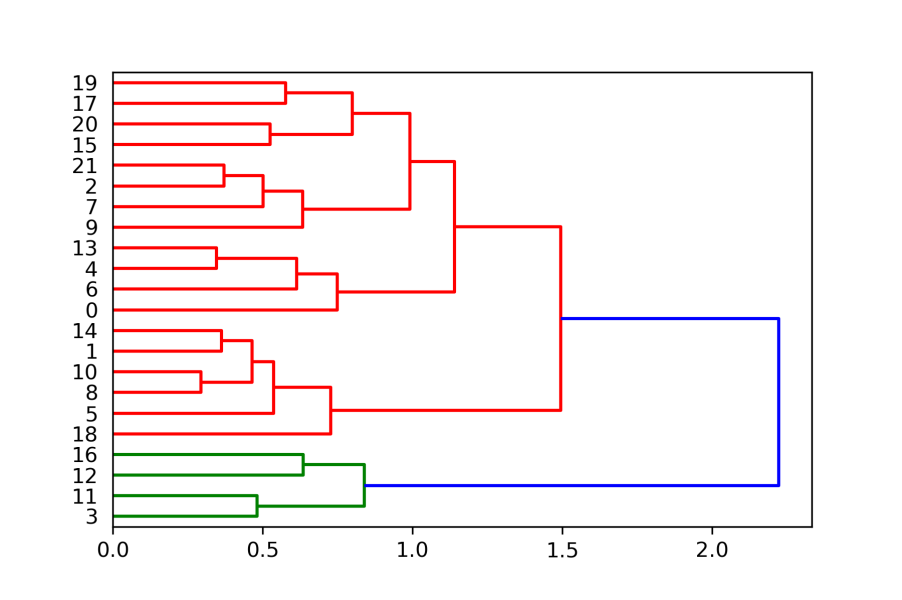
The sum of squared distances of samples to their closest cluster centre is:
38185.817
The number of samples for each cluster is:
0: 1018
1: 1125
2: 1191
3: 890
4: 1162
5: 1332
6: 839
7: 623
8: 1400
9: 838
10: 659
11: 1276
12: 121
13: 152
14: 950
15: 1971
16: 1251
17: 845
18: 896
19: 930
20: 1065
21: 1466

**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.




the mean vectors and cluster centres are not similar, where the mean vectors are more stay centralized in specific small area. However, for the cluster centres, they are more sporadic.

**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.



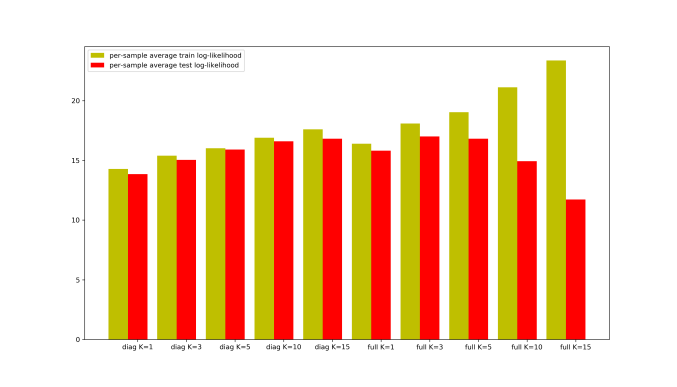

From the dendrogram, we see that nearby languages on vertical axis are clustered together. With two clusters correspond to n_components=2, where the red cluster consists of the most language, and the green one contains 4 languages. The distance threshold is between about 0 to 2.2

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


For single linkage, we can see that it typically characterized by one very large clusters. From the dendrogram, two cluster consist of 5 languages, with the main cluster collecting the left 61 languages.
For complete linkage, it suffers from some of the same problems as single linkage, the dendrogram indicates the unbalanced structure, with two small clusters each consists of two languages, and a very large clusters.
The dendrogram for 'ward', the clusters are farily balanced, unlike the single linkage and complete linkage. With two main clusters and a cluster with only two languages.

*Question 3 continued on next page...*

**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.,




| | diag_train | full_train | diag_test | full_test |
|---|---|---|---|---|
| K=1 | 14.280 | 16.394 | 13.843 | 15.811 |
| K=3 | 15.396 | 18.117 | 15.041 | 17.013 |
| K=5 | 16.011 | 19.080 | 15.907 | 16.791 |
| K=10 | 16.933 | 20.95 | 16.597 | 15.391 |
| K=15 | 17.670 | 23.146 | 16.845 | 12.337 |

From the table and graph above. We can find:
per-sample average log-likelihood increase for diag_train, full_train, diag_test with the increase of K, but for full_test, it will decrease after K=5.
The per-sample average log-likelihood for 'diagonal covariance' most less than 'full covariance' under the same K value.

**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,

```
def SimpleGMM_train(X, n_components):
    # apply Kmeans clustering
    Kmeans=KMeans(n_clusters=n_components, random_state=1).fit(X)
    # mean vectors (ndarray, shape (n_components, n_features))
    Xmean=Kmeans.cluster_centers_

    labels=Kmeans.labels_
    # a vector of diagonal elements of covariance matrix
    diagonal_elements=np.zeros((n_components,X.shape[1]))
    for i in range(n_components):
        diagonal_elements[i,:]=np.diag(np.cov(X[labels==i].T))

    # weights of mixture components
    weights=np.zeros((n_components))
    for j in range(n_components):
        weights[j]=np.count_nonzero(labels==j)/X.shape[0]
    return Xmean,diagonal_elements,weights


def SimpleGMM_eval(X, Ms, Dcovs, Pk):
    n_components=len(Ms)
    Kmeans=KMeans(n_clusters= n_components, random_state=1).fit(X)
    labels=Kmeans.labels_

    loglikelihoods=[]
    for i in range(n_components):
        loglikelihoods.append(Pk[labels[i]]*multivariate_normal.pdf(X, mean=Ms[i],

    p=pd.DataFrame(loglikelihoods)
    q=np.array(p.describe().loc['mean'])
    lh=q*n_components

    Log_L_samples=[]
    for w in range(len(lh)):
        Log_L_samples.append(np.log(lh[w]))

    return Log_L_samples
```

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

the weights of mixture components for each K = 1, 5, 10:
K=1: [ 1 ]
K=5: [ 0.151, 0.287, 0.249, 0.181, 0.133 ]
K=10: [ 0.090 , 0.113 , 0.065 , 0.100 , 0.143 , 0.071 , 0.102 , 0.071 , 0.114 , 0.132 ]

**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 $K = 1, 3, 5, 10, 15$.

| | average log-likelihood on training data | average log-likelihood on test data |
|---|---|---|
| K=1 | 14.280 | 13.843 |
| K=3 | 15.092 | 15.072 |
| K=5 | 15.671 | 15.394 |
| K=10 | 16.594 | 16.229 |
| K=15 | 17.150 | 16.519 |

The average log-likelihood on training data are larger than those on test data when using the same K, the model is trained on training data, which will make it fit the training data better.

Compared with the per-sample average log-likelihood in Question 3.5. We find that for every situation of K and data-type, the log-likelihood in Question 3.5 are always little larger than in 3.8. As the EM algorithm will incrementally optimises model parameters, but the SimpleGMM we used will not.