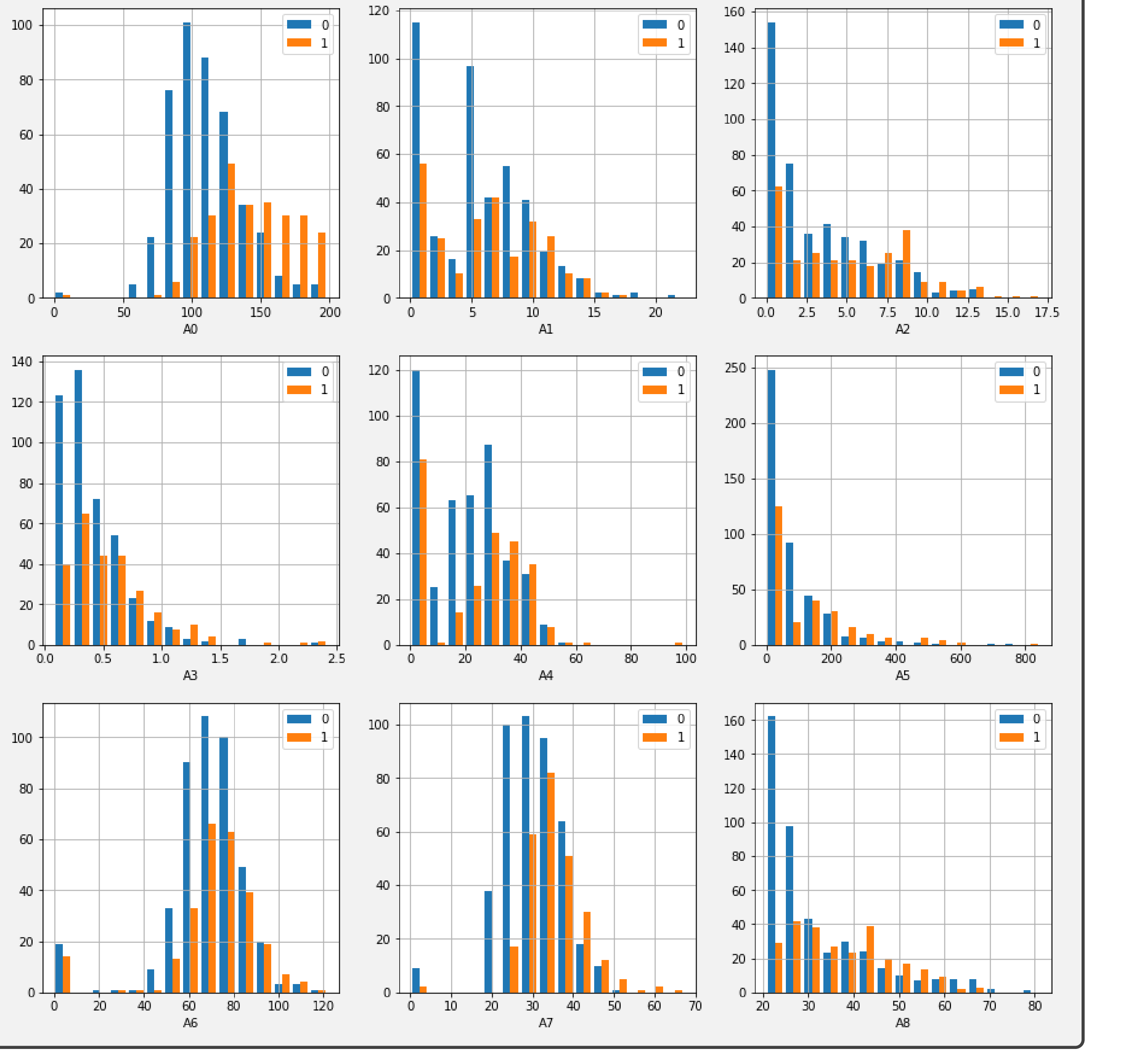
Question 1 : (70 total points) Experiments on a binary-classification data set

1.1 (9 points) We want to see how each feature in `Xtrn` is distributed for each class. Since there are nine attributes, we plot a total of nine figures in a 3-by-3 grid, where the top-left figure shows the histograms for attribute 'A0' and the bottom-right 'A8'. In each figure, you show histograms of instances of class 0 and those of class 1 using `pyplot.hist([Xa, Xb], bins=15)`, where `Xa` corresponds to instances of class 0 and `Xb` to those of class 1, and you set the number of bins to 15. Use grid lines. Based on the results you obtain, discuss and explain your findings.

Overall, features A0, A5 and A6 has relatively larger absolute values, at approximately > 100 range, whereas other features have ranges < 100, especially A3 which is < 3.
There are more instances with label 0 than 1, so to compare the frequencies directly will not be meaningful, but we can still visually compare the trend and shape of the histograms. We can see that for most features, especially A0 and A7, instances with label 0 and 1 are quite visibly separated, with the former having smaller values on average than the latter. That is, we can visually see from the diagram that the blue bars have a trend to lie in the left, whereas the orange bars tend to lie in the right, relative to each other.



1.2 (9 points) Calculate the correlation coefficient between each attribute of `Xtrn` and the label `Ytrn`, so that you calculate nine correlation coefficients. Answer the following questions.

(a) Report the correlation coefficients in a table.
(b) Discuss if it is a good idea to use the attributes that have large correlations with the label for classification tasks.
(c) Discuss if it is a good idea to ignore the attributes that have small correlations with the label for classification tasks.

(a)

A0 and Ytrn	A1 and Ytrn	A2 and Ytrn
0.4912	0.0874	0.2273
A3 and Ytrn	A4 and Ytrn	A5 and Ytrn
0.2074	0.1077	0.1857
A6 and Ytrn	A7 and Ytrn	A8 and Ytrn
0.0763	0.3045	0.2403

(b) If we are hoping for a linear relationship between attributes or using a linear classifier, then it would be a good idea to use attributes with large correlations, since its absolute value measures the strength of linear relationship. However, it might not be of too much use if we expect strong non-linear relationship or using a non-linear classifier, so in this case it's likely not a good idea.

(c) Ignoring attributes that have small correlations can perhaps help when the classifier model is showing signs of overfitting, but when the relationship between an attribute and Y is strongly non-linear, correlation coefficient likely won't give us too much insight as to whether we should ignore it. If keeping the attributes doesn't worsen the performance of the classifier, or instead increases its performance, then it is likely not a good idea to ignore those attributes, otherwise it would be a good idea.

1.3 (4 points) We consider a set of instances of two variables, $\{(u_i, v_i)\}_{i=1}^N$, where N denotes the number of instances. Show (using your own words and mathematical expressions) that the correlation coefficient between the two variables, r_{uv}, is translation invariant and scale invariant, i.e. r_{uv} does not change under linear transformation, $a + bu_i$ and $c + dv_i$ for $i = 1, \dots, N$, where a, b, c, d are constants and $b > 0, d > 0$.

The formula for correlation coefficient is: $\frac{1}{n}\sum_{i=1}^{n}\frac{(X_i-\mu_X)(Y_i-\mu_Y)}{\sigma_X\sigma_Y}$, where $\sigma = \sqrt{\mu_X^2 - (\mu_X)^2}$ and $\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i$ by definition.

It is clear that μ_x will go through the same linear transformation as all X_i, that is $new(\mu_X) = a\mu_X$.

It is then easy to see that under the linear transformation, $new(X_i - \mu_X) = a(X_i - \mu_X)$.

For any instance X_i, a linear transformation in the form $a + bX_i$ where $b > 0$ will result in its new variance becoming $a\sigma_X$, by replacing μ_X with $a + b\mu_X$ in the above formula.

Therefore, for the correlation coefficient, it will become $\frac{1}{n}\sum_{i=1}^{n}\frac{a^2(X_i-\mu_X)(Y_i-\mu_Y)}{a^2\sigma_X\sigma_Y} = \frac{1}{n}\sum_{i=1}^{n}\frac{(X_i-\mu_X)(Y_i-\mu_Y)}{\sigma_X\sigma_Y}$, which is the same as before the linear transformation.

Question 1 continued on next page. . .

1.4 (5 points) Calculate the unbiased sample variance of each attribute of `Xtrn`, and sort the variances in decreasing order. Answer the following questions.

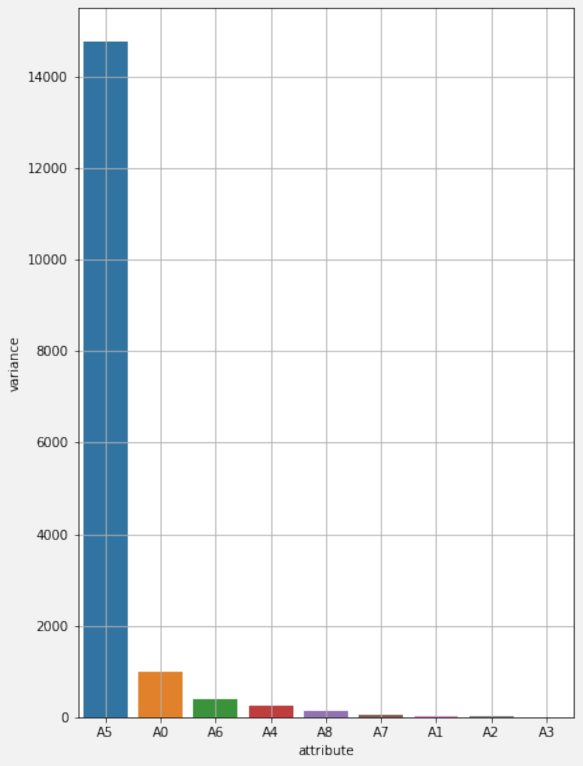
(a) Report the sum of all the variances.

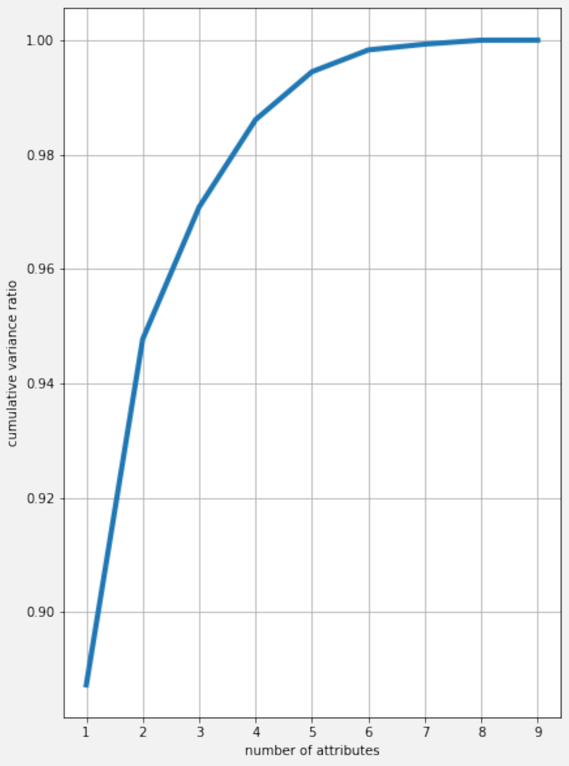
(b) Plot the following two graphs side-by-side. Use grid lines in each plot.

- A graph of the amount of variance explained by each of the (sorted) attributes, where you indicate attribute numbers on the x-axis.
- A graph of the cumulative variance ratio against the number of attributes, where the range of y-axis should be [0, 1].

(a) 16645.6

(b) plot 1 on the left and 2 on the right





Question 1 continued on next page. . .

1.5 (8 points) Apply Principal Component Analysis (PCA) to `Xtrn`, where you should not rescale `Xtrn`. Use Sklearn's PCA with default parameters, i.e. specifying no parameters.

(a) Report the total amount of unbiased sample variance explained by the whole set of principal components.

(b) Plot the following two graphs side-by-side. Use grid lines in each plot.

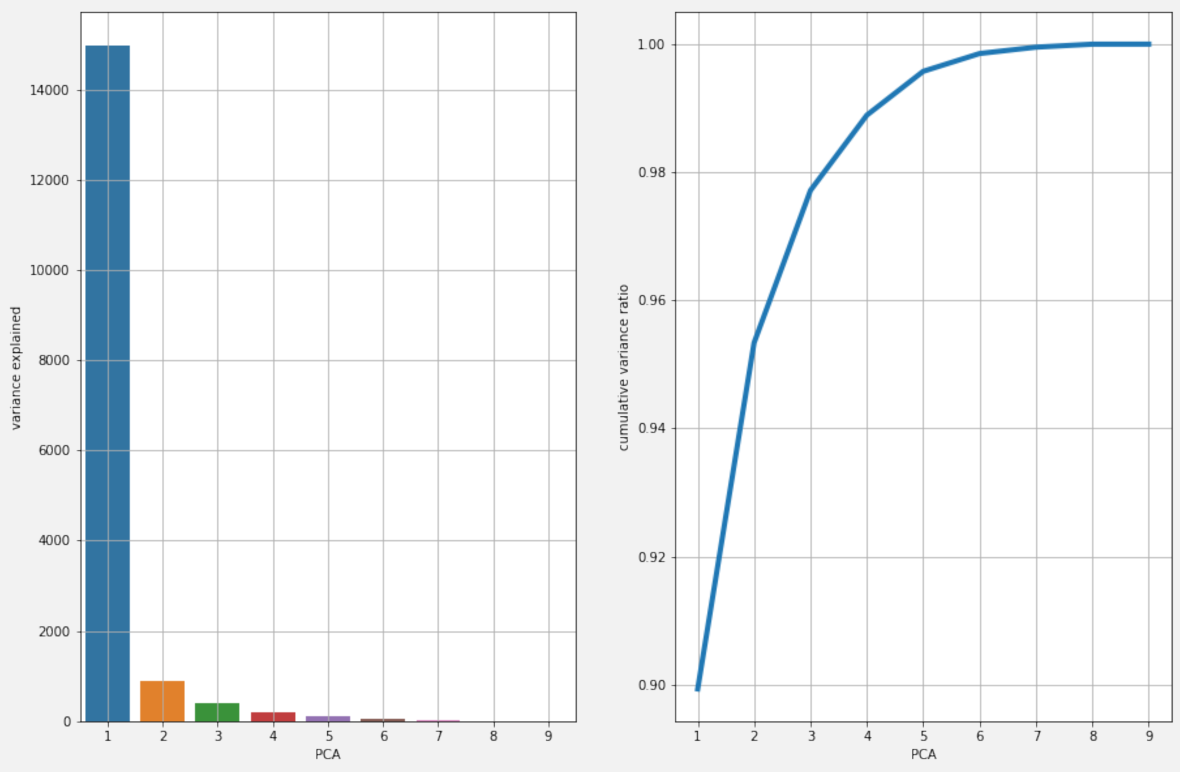
- A graph of the amount of variance explained by each of the principal components.
- A graph of the cumulative variance ratio, where the range of y-axis should be [0, 1].

(c) Mapping all the instances in `Xtrn` on to the 2D space spanned with the first two principal components, and plot a scatter graph of the instances on the space, where instances of class 0 are displayed in blue and those of class 1 in red. Use grid lines. Note that the mapping should be done directly using the eigen vectors obtained in PCA - you should not use Sklearn's functions, e.g. `transform()`.

(d) Calculate the correlation coefficient between each attribute and each of the first and second principal components, report the result in a table.

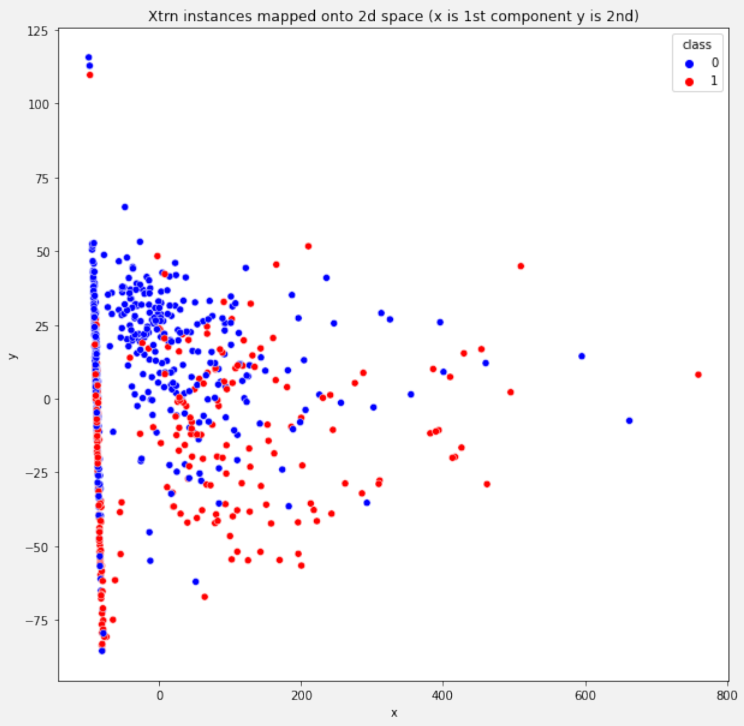
(a) Total amount of unbiased sample variance: 16645.6

(b) plot 1 on the left and 2 on the right



(*continued from the previous page for Q1.5*)

(c) The first component is x (horizontal) and the second is y (vertical)



(d) A stands for attribute (9 attributes) and C stands for principle components (the first 2)

	A0	A1	A2	A3	A4	A5	A6	A7	A8
C1	0.3856	-0.0458	-0.0571	0.1858	0.4592	0.9997	0.1006	0.2323	-0.0016
C2	-0.9143	-0.0908	-0.2255	-0.0799	0.0972	0.0241	-0.2554	-0.1726	-0.3734

1.6 (4 points) We now standardise the data by mean and standard deviation using the method described below, and look into how the standardisation has impacts on PCA.

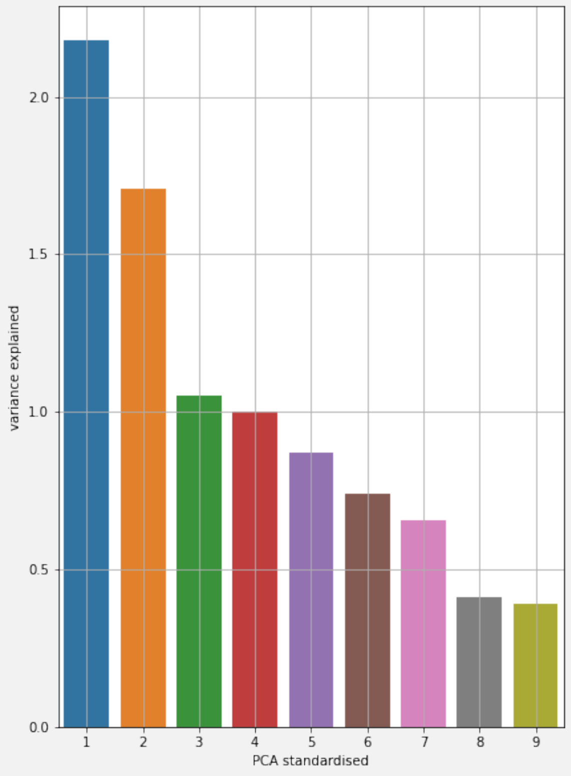
Create the standardised training data `Xtrn_s` and test data `Xtst_s` in your code in the following manner.

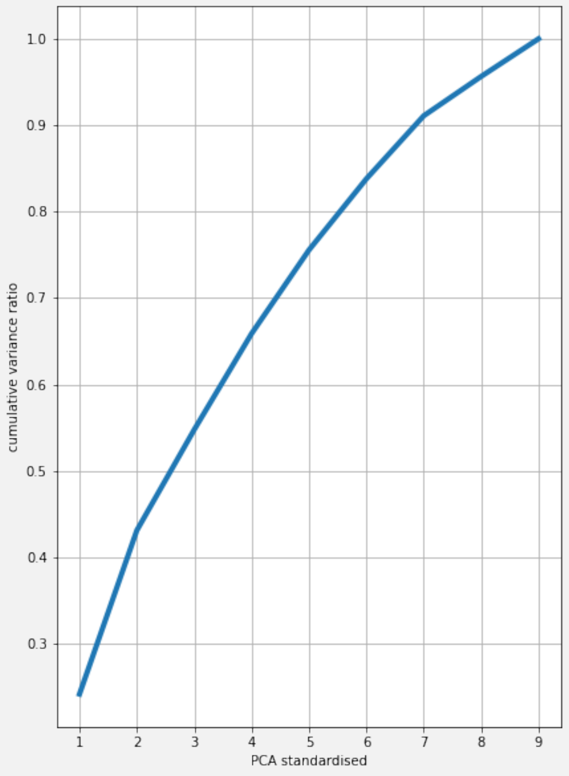
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(Xtrn)
Xtrn_s = scaler.transform(Xtrn)      # standardised training data
Xtst_s = scaler.transform(Xtst)      # standardised test data
```

Using the standardised data `Xtrn_s` instead of `Xtrn`, answer the questions (a), (b), (c), and (d) in 1.5.

(a) Total amount of unbiased sample variance: 9.0129

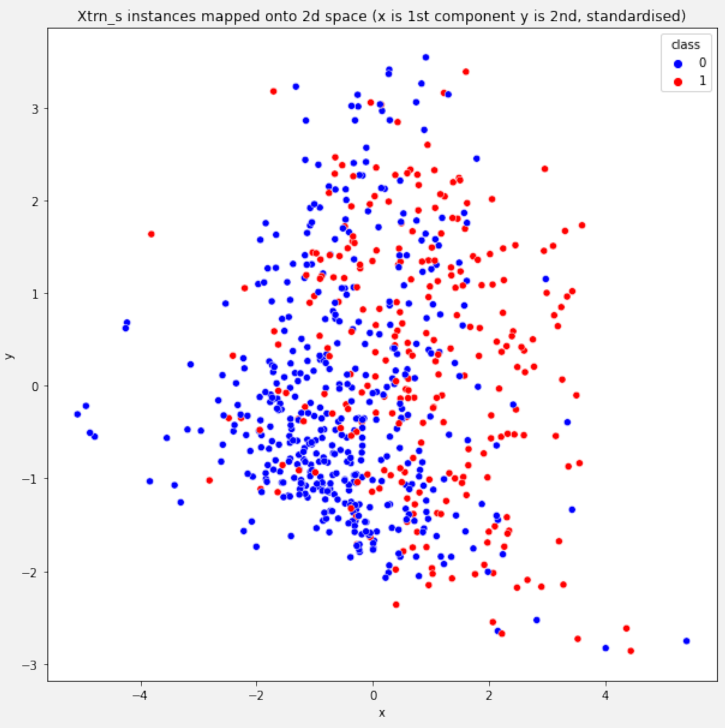
(b) plot 1 on the left and 2 on the right





(*continued from the previous page for Q1.6*)

(c) The first component is x (horizontal) and the second is y (vertical)



(d) A stands for attribute (9 attributes) and C stands for principle components (the first 2)

	A0	A1	A2	A3	A4	A5	A6	A7	A8
C1	0.6007	0.0573	0.2680	0.3657	0.6230	0.6299	0.5229	0.6512	0.3529
C2	0.1774	0.1000	0.7600	-0.2076	-0.4660	-0.3698	0.2242	-0.1684	0.7812

1.7 (7 points) Based on the results you obtained in 1.4, 1.5, and 1.6, answer the following questions.

(a) Comparing the results of 1.4 and 1.5, discuss and explain your findings.
(b) Comparing the results of 1.5 and 1.6, discuss and explain your findings and discuss (*using your own words*) whether you are strongly advised to standardise this particular data set before PCA.

(a) It can be seen, more visibly from the cumulative variance diagram, that PCA increases the ratio of variance explained compared to raw attributes. For example, with PCA, the variance ratio for the first component is around 0.90, while for the raw attribute, the largest one is well below 0.90. The same can be seen from the first two largest (for PCA above 0.95 while for raw attributes below 0.95), etc. That is, although both explains 100% of variance when all attributes/components are considered, with PCA, when only some of the more dominant ones are considered, it increases the variance explained. This is because PCA by nature and definition, picks dimensions to maximise variability while making those principle components have 0 covariance.

(b) 1.6 differs from 1.5 only in that the training instances used are standardised. Noticeably in 1.5, the variance explained ratio is dominated heavily by only one principle component. That is because there is an attribute with very large values relative to all others , which leads to it having a large covariance to others, and in turn the large variance explained by it. However, after normalisation, we can see that all other principle components contribute much more evenly to variance explained. It is therefore strongly advised to standardise this data set, because otherwise we would believe only one or two principle components are important (variance explained $> 90\%$, for example), while in fact it is clearly not.

Question 1 continued on next page…

1.8 (12 points) We now want to run experiments on Support Vector Machines (SVMs) with a RBF kernel, where we try to optimise the penalty parameter C. By using 5-fold CV on the standardised training data `Xtrn_s` described above, estimate the classification accuracy, while you vary the penalty parameter C in the range 0.01 to 100 - use 13 values spaced equally in log space, where the logarithm base is 10. Use Sklearn's `SVC` and `StratifiedKFold` with default parameters unless specified. Do not shuffle the data.

Answer the following questions.

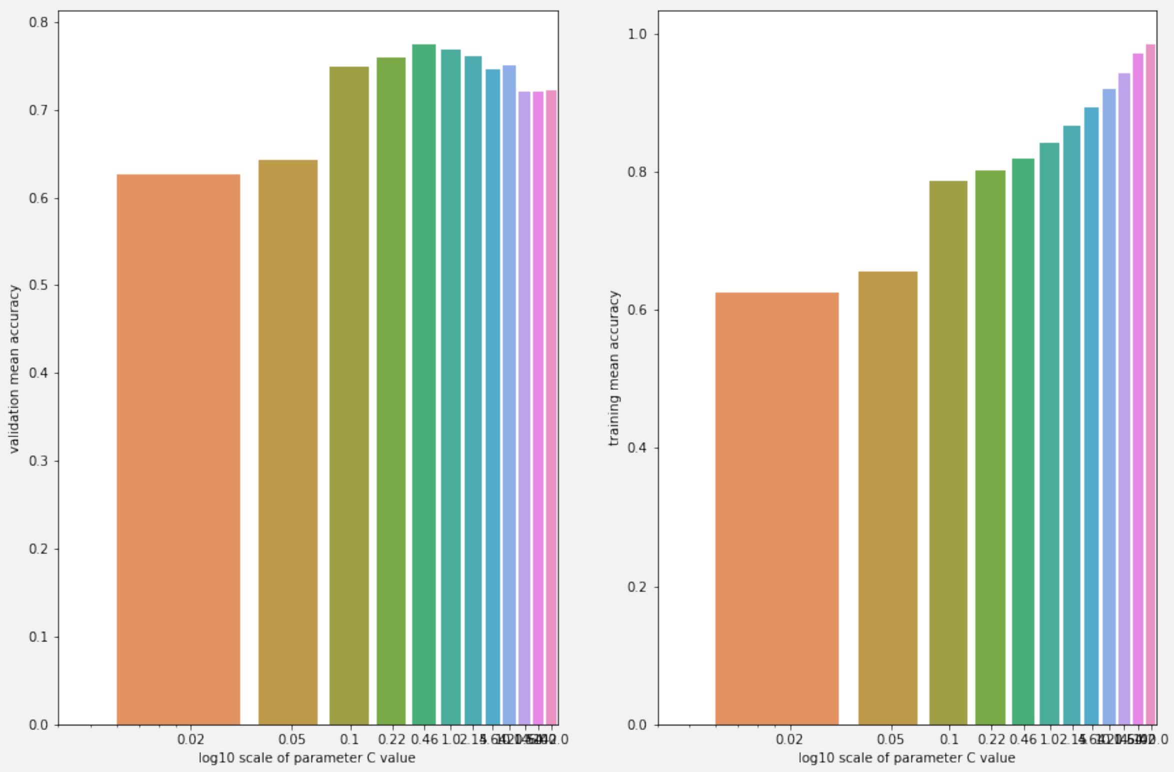
(a) Calculate the mean and standard deviation of cross-validation classification accuracy for each C, and plot them against C by using a log-scale for the x-axis, where standard deviations are shown with error bars. On the same figure, plot the same information (i.e. the mean and standard deviation of classification accuracy) for the training set in the cross validation.

(b) Comment (in brief) on any observations.

(c) Report the highest mean cross-validation accuracy and the value of C which yielded it.

(d) Using the best parameter value you found, evaluate the corresponding best classifier on the test set { `Xtst_s`, `Ytst` }. Report the number of instances correctly classified and classification accuracy.

(a) Testing set on the left and training set on the right.



(b) Mean accuracy on training and validation are close, and accuracy for training increases monotonically with C's value, but that is not the case for validation which sees a drop in accuracy with some higher C values.

(c) Highest Mean cross-validation accuracy is 0.774, yielded by C value of 0.464.

(d) 77 correctly classified out of 100, so the accuracy is 77%.

1.9 (5 points) We here consider a two-dimensional (2D) Gaussian distribution for a set of two-dimensional vectors, which we form by selecting a pair of attributes, A4 and A7, in `Xtrn` (NB: not `Xtrn_s`) whose label is 0. To make the distribution of data simpler, we ignore the instances whose A4 value is less than 1. Save the resultant set of 2D vectors to a Numpy array, `Ztrn`, where the first dimension corresponds to A4 and the second to A7. You will find 318 instances in `Ztrn`.

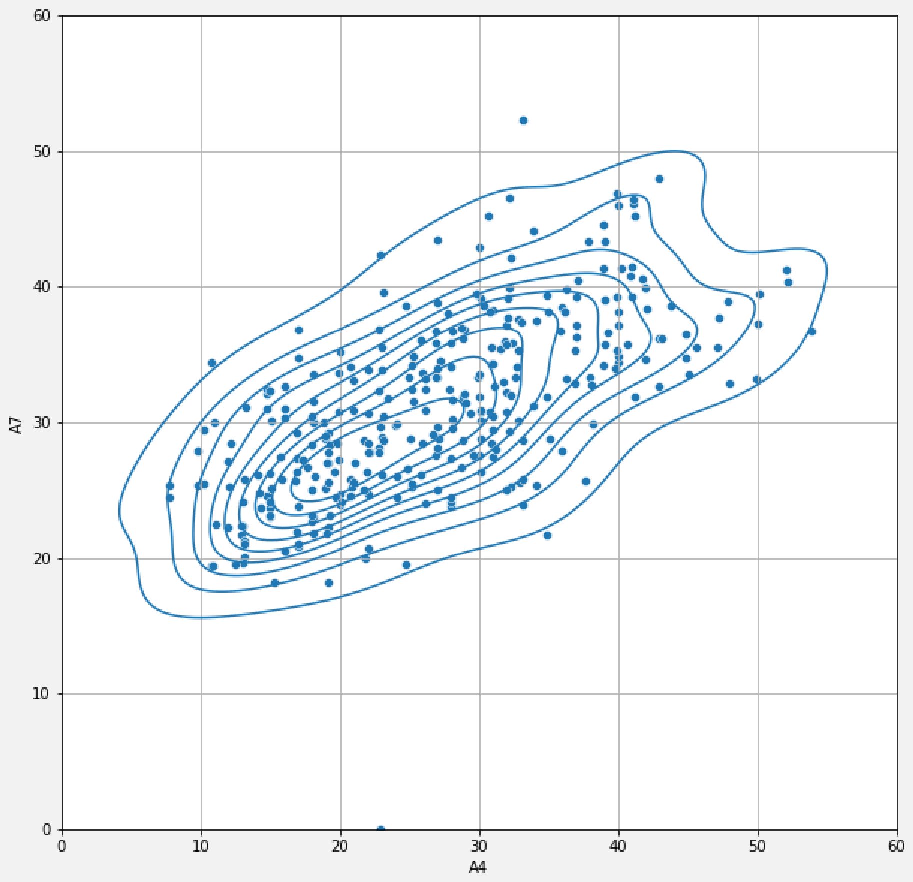
Using Numpy's libraries, estimate the sample mean vector and unbiased sample covariance matrix of a 2D Gaussian distribution for `Ztrn`. Answer the following questions.

(a) Report the mean vector and covariance matrix of the Gaussian distribution.
(b) Make a scatter plot of the instances and display the contours of the estimated distribution on it using Matplotlib's contour. Note that the first dimension of `Ztrn` should correspnd to the x-axis and the second to y-axis. Use the same scaling (i.e. equal aspect) for the x-axis and y-axis, and show grid lines.

(a) Mean vector is $(27.021, 31.093)$. Covariance matrix is

	A4	A7
A4	95.141	41.470
A7	41.470	46.693

(b) Scatter plot and contour.



1.10 (7 points) Assuming naive-Bayes, estimate the model parameters of a 2D Gaussian distribution for the data `Ztrn` you created in 1.9, and answer the following questions.

(a) Report the sample mean vector and unbiased sample covariance matrix of the Gaussian distribution.

(b) Make a new scatter plot of the instances in `Ztrn` and display the contours of the estimated distribution on it. Note that you should always correspond the first dimension of `Ztrn` to x-axis and the second dimension to y-axis. Use the same scaling (i.e. equal aspect) for x-axis and y-axis, and show grid lines.

(c) Comparing the result with the one you obtained in 1.9, discuss and explain your findings, and discuss if it is a good idea to employ the naive Bayes assumption for this data `Ztrn`.

(a) Your Answer for (a) Here

(b) Your Answer for (b) Here

(c) Your Answer for (c) Here

Question 2 : (75 total points) Experiments on an image data set of handwritten letters

2.1 (5 points)

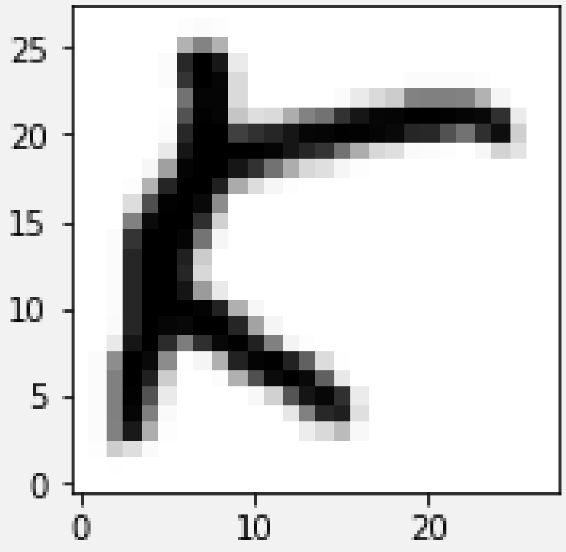
(a) Report (using a table) the minimum, maximum, mean, and standard deviation of pixel values for each `Xtrn` and `Xtst`. (Note that we mean a single value of each of min, max, etc. for each `Xtrn` and `Xtst`.)

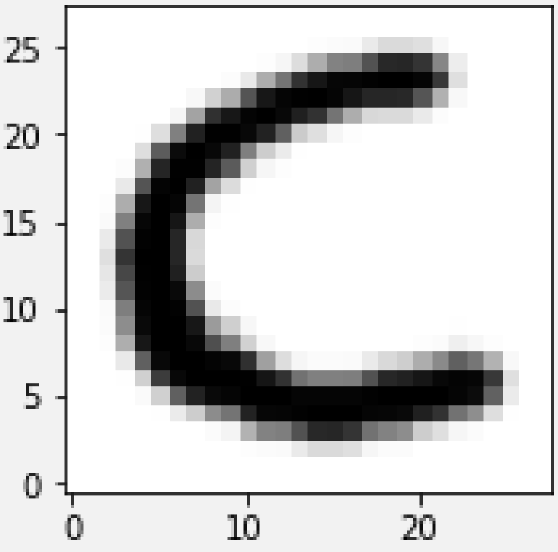
(b) Display the gray-scale images of the first two instances in `Xtrn` properly, clarifying the class number for each image. The background colour should be white and the foreground colour black.

(a)

	min	max	mean	std
Xtrn	0.0	1.0	0.17738	0.33498
Xtst	0.0	1.0	0.17563	0.33346

(b) Class number for the image on the left is 10, for the image on the right is 2.





Question 2 continued on next page. . .

2.2 (4 points)

(a) `Xtrn_m` is a mean-vector subtracted version of `Xtrn`. Discuss if the Euclidean distance between a pair of instances in `Xtrn_m` is the same as that in `Xtrn`.

(b) `Xtst_m` is a mean-vector subtracted version of `Xtst`, where the mean vector of `Xtrn` was employed in the subtraction instead of the one of `Xtst`. Discuss whether we should instead use the mean vector of `Xtst` in the subtraction.

(a) Instances are vectors of shape (728,), so the Euclidean distance between pair of instances in Xtrn_m is the same as that in Xtrn, since subtraction of mean will cancel out when calculating Euclidean distance.

(b) It would make sense to use mean of Xtrn, because there are more training instances, therefore the mean would be closer to population mean, which is a good sign, whereas the mean of Xtst may be further away from population mean, making testing less meaningful.

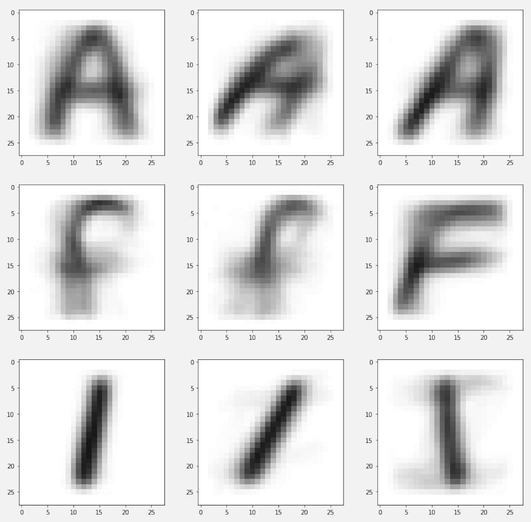
Question 2 continued on next page. . .

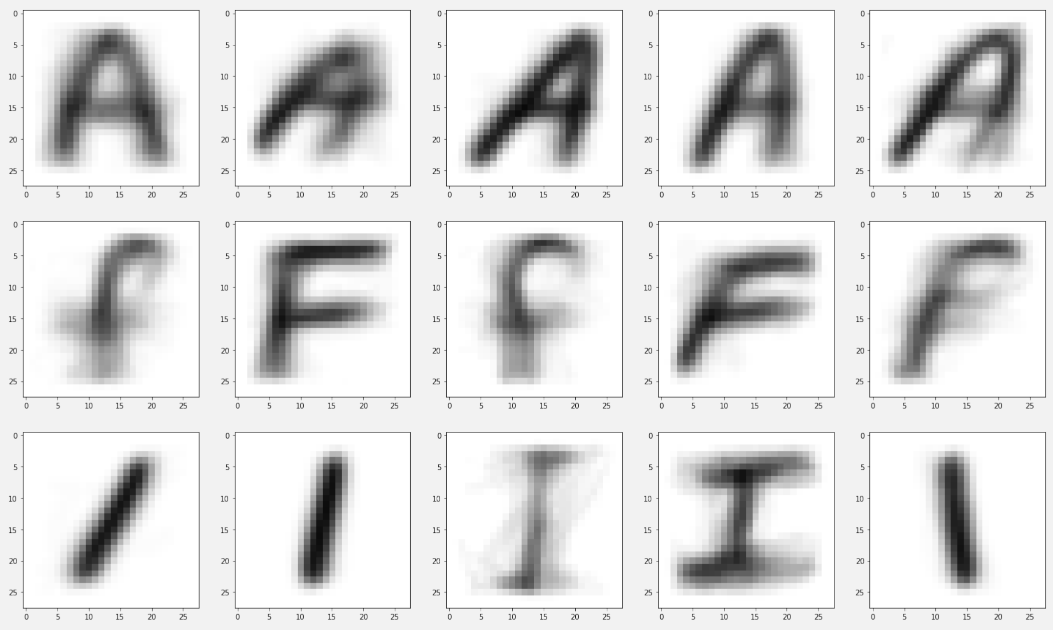
2.3 (7 points) Apply k-means clustering to the instances of each of class $0, 5, 8$ (i.e. 'A', 'F', 'I') in `Xtrn` with $k = 3, 5$, for which use Sklearn's `KMeans` with `n_clusters=`k and `random_state=0` while using default values for the other parameters. Note that you should apply the clustering to each class separately. Make sure you use `Xtrn` rather than `Xtrn_m`. Answer the following questions.

(a) Display the images of cluster centres for each k, so that you show two plots, one for $k = 3$ and the other for $k = 5$. Each plot displays the grayscale images of cluster centres in a 3-by-k grid, where each row corresponds to a class and each column to cluster number, so that the top-left grid item corresponds to class 0 and the first cluster, and the bottom-right one to class 8 and the last cluster.

(b) Discuss and explain your findings, including discussions if there are any concerns of using this data set for classification tasks.

(a) Images of cluster centres for k=3 (above) and k=5 (below).





(b) It can be seen that the cluster centres visually resemble the shape of the corresponding letters, and each cluster centre for a letter can reflect a distinct type of hand writing style. It looks like k=3 is not as good as k=5 in terms of clustering performance. However, there are mislabelled data which will have affected the clustering performance and the cluster centres.

2.4 (5 points) Explain (using your own words) why the sum of square error (SSE) in k-means clustering does not increase for each of the following cases.

(a) Clustering with $k + 1$ clusters compared with clustering with k clusters.
(b) The update step at time $t + 1$ compared with the update step at time t when clustering with k clusters.

(a) Suppose we have k clusters already, and we add in another cluster centre. The worst case in terms of SSE would be the new cluster centre lies infinitely far away from all data points, and that would change nothing, meaning no increase (or decrease) in SSE. Otherwise if that newly added cluster centre actually affects clustering, then due to k-means algorithm's iteration of moving cluster centres to minimise inertia, it will by definition decrease SSE. Therefore it does not increase.

(b) We know that k-means algorithm will move the current cluster centre to the new mean of the current cluster. If the SSE were to increase, it would mean that some new data point is further away from the cluster centre than previous ones (which increases SSE), but that is not possible, since the new cluster centre moves towards the new mean/centroid, which means if any point is actually further away, it will not come into this cluster. Therefore any update step will not increase the SSE.

Question 2 continued on next page. . .

2.5 (11 points) Here we apply multi-class logistic regression classification to the data. You should use Sklearn's `LogisticRegression` with parameters '`max_iter=1000`' and '`random_state=0`' while use default values for the other parameters. Use `Xtrn_m` for training and `Xtst_m` for testing. We do not employ cross validation here. Carry out a classification experiment.

(a) Report the classification accuracy for each of the training set and test set.
(b) Find the top five classes that were misclassified most in the test set. You should provide the class numbers, corresponding alphabet letters (e.g. A,B,...), and the numbers of misclassifications.
(c) For each class that you identified in the above, make a quick investigation and explain possible reasons for the misclassifications.

(a) Accuracy for training set: 0.916. Accuracy for testing set: 0.722.

(b) Top 5 misclassified letters.

Class Number	Letter	#misclassification
11	L	53
17	R	48
8	I	42
10	K	38
13	N	36

(c) These letters are less distinguishable from other letters, for example, many of the misclassified 'I' are mistaken as 'L', 'R' are mistaken as 'K', 'N' are mistaken as 'H' and 'W'. It could also be that they have larger variations in hand writing styles which makes them even harder to separate from each other.

2.6 (20 points) Without changing the learning algorithm (i.e. use logistic regression), your task here is to improve the classification performance of the model in 2.5. Any training and optimisation (e.g. hyper parameter tuning) should be done within the training set only. Answer the following questions.

(a) Discuss (using your own wards) three possible approaches to improve classification accuracy, decide which one(s) to implement, and report your choice.
(b) Briefly describe your implemented approach/algorithm so that other people can understand it without seeing your code. If any optimisation (e.g. parameter searching) is involved, clarify and describe how it was done.
(c) Carry out experiments using the new classification system, and report the results, including results of parameter optimisation (if any) and classification accuracy for the test set. Comments on the results.

(a) 1. Correct the erroneous labels in the dataset.
2. Normalise data before training.
3. Perform hyper parameter tuning. Do cross validation in the process, such as k-fold stratification.
Correcting the mislabeled data takes too long due to the amount of instances, therefore won't be implemented. Normalising data is quick and worth a try, so it will be implemented. Hyper parameter tuning is likely the most effective approach so will definitely be implemented.

(b) The data is normalised by using the StandardScaler from sklearn, which automatically normalises the dataset to unit variance and mean of 0. The hyper parameter C, is chosen from a range (in my case [0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 2, 5, 10], handpicked because there is no need to exhaustively try out the values), and also the "penalty" hyper parameter, which can be ['l2', 'none'] (this selection is restricted by the fact that the hyper parameter C is being used, for example, "l1" can be used only when C is not supplied as a hyper parameter). Then a grid search cross validation is applied using sklearn's GridSearchCV implementation, to find the combination of values for C and penalty on the model, by computing the cross validated testing accuracy. Cross validation is done by the sklearn library, using default 5-fold. The combination of two values that gives the best testing accuracy is regarded as the best hyper parameter.

Question 2 continued on next page. . .

(*continued from the previous page for Q2.6*)

(c) It appears that normalisation has very little effect on the accuracy (still improving by a few percent), which is understandable, since all pixel values are in the range [0, 1], so normalisation did not make a big difference on the dataset. Hyper parameter tuning has a greater effect. With the C value increasing, there is first an increase of both training accuracy and testing accuracy (from (0.647, 0.677) respectively to (0.753, 0.823).). But afterwards, testing accuracy started to decrease while training accuracy kept increasing (testing accuracy dropped steadily to 0.728 while training accuracy increased to 0.908), because the larger the C value, the weaker the regularisation which leads to overfitting on training data. The best parameter found was with C=0.1 and penalty of "l2". This has a classification accuracy of 0.754.

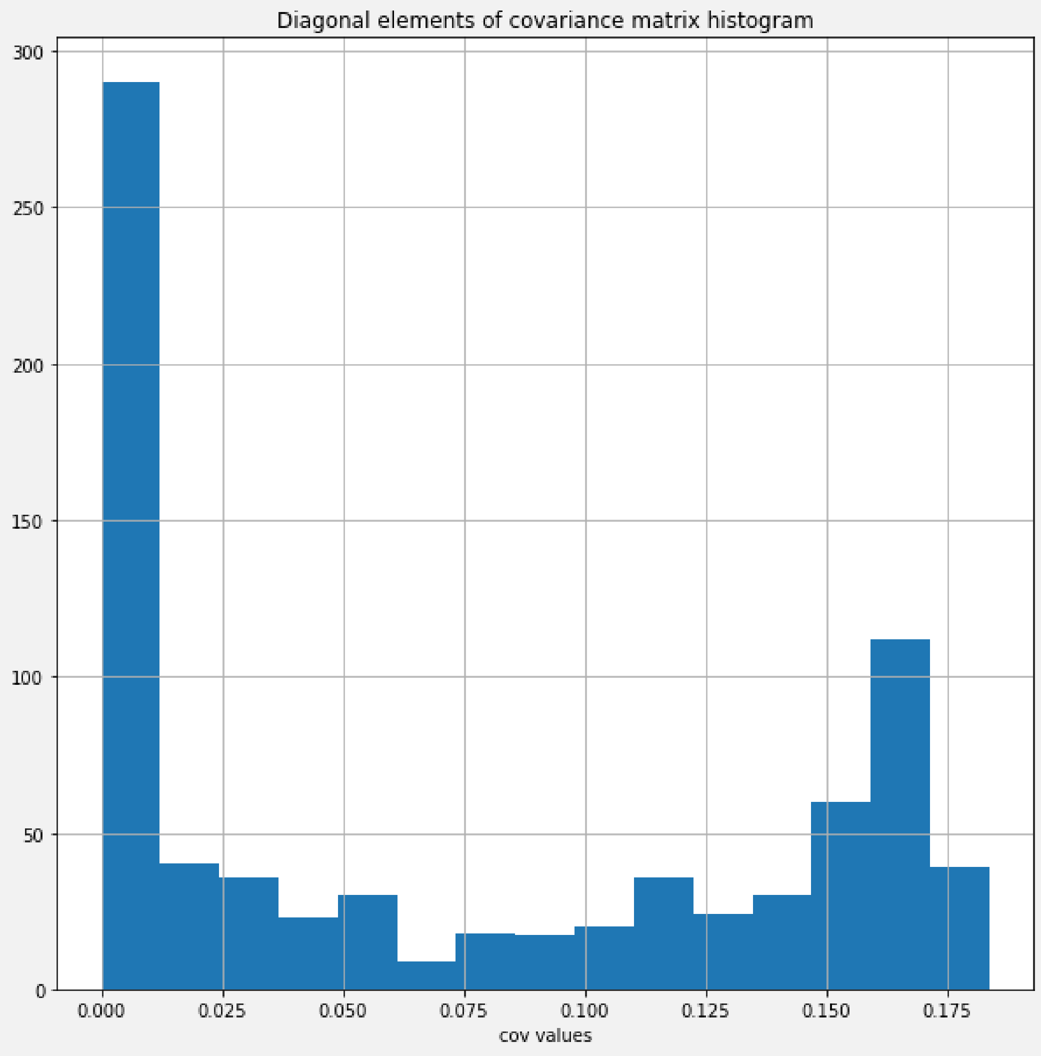
2.7 (9 points) Using the training data of class 0 ('A') from the training set `Xtrn_m`, calculate the sample mean vector, and unbiased sample covariance matrix using Numpy's functions, and answer the following.

(a) Report the minimum, maximum, and mean values of the elements of the covariance matrix.
(b) Report the minimum, maximum, and mean values of the diagonal elements of the covariance matrix.
(c) Show the histogram of the diagonal values of the covariance matrix. Set the number of bins to 15, and use grid lines in your plot.
(d) Using Scipy's `multivariate_normal` with the mean vector and covariance matrix you obtained, try calculating the likelihood of the first element of class 0 in the test set (`Xtst_m`). You will receive an error message. Report the main part of error message, i.e. the last line of the message, and explain why you received the error, clarifying the problem with the data you used.
(e) Discuss (using your own words) three possible options you would employ to avoid the error. Note that your answer should not include using a different data set.

(a) For all elements: $min = -0.09747, max = 0.18379, mean = 0.00171$

(b) For diagonal elements: $min = 0.00000$, $max = 0.18379$, $mean = 0.07231$

(c) Histogram of diagonal values of covariance matrix.



(*continued from the previous page for Q*)

(d) Error message is: "LinAlgError: singular matrix". This means the covariance matrix cannot be inverted, therefore the Gaussian pdf cannot be computed. This means some instances in the dataset, in this case some gray scale images, are almost identical to each other so that they can (almost) be expressed as linear combinations of each other.

(e) Adding some noise to the images may also help to mitigate instances being almost identical to each other. Another option could be to pick a (random) subset of pixels/attributes to avoid whole images being linear combinations of each other. It may be possible to preprocess the dataset to remove near duplicates of instances to lower the chance of having a singular covariance matrix.

Question 2 continued on next page...

2.8 (8 marks) Instead of Scipy's `multivariate_normal` we used in 2.7, we now use Sklearn's `GaussianMixture` with parameters, `n_components=1, covariance_type='full'`, so that there is a single Gaussian distribution fitted to the data. Use { `Xtrn_m`, `Ytrn` } as the training set and { `Xtst_m`, `Ytst` } as the test set.

(a) Train the model using the data of class 0 ('A') in the training set, and report the log-likelihood of the first instance in the test set with the model. Explain why you could calculate the value this time.

(b) We now carry out a classification experiment considering all the 26 classes, for which we assign a separate Gaussian distribution to each class. Train the model for each class on the training set, run a classification experiment using a multivariate Gaussian classifier, and report the number of correctly classified instances and classification accuracy for each training set and test set.

(c) Briefly comment on the result you obtained.

(a) Log likelihood of first test instance for class 0 is -838252.18. Gaussian mixture models do not need the determinant of covariance matrix, unlike with multivariate normal, and it uses the EM algorithm.

(b) For test set: total instances = 2600, correctly classified = 1803, accuracy = 0.693.
For training set: total instances = 7800, correctly classified = 7800, accuracy = 1.0.

(c) Accuracy on training set is 100%, and that is likely because we are training a model for every letter which converges and give good fit on the training data. Accuracy for testing set is quite good, though obviously there is overfitting.

Question 2 continued on next page...

2.9 (6 points) Answer the following question on Gaussian Mixture Models (GMMs).

(a) Explain (using your own words) why Maximum Likelihood Estimation (MLE) cannot be applied to the training of GMMs directly.

(b) The Expectation Maximisation (EM) algorithm is normally used for the training of GMMs, but another training algorithm is possible, in which you employ k-means clustering to split the training data into clusters and apply MLE to estimate model parameters of a Gaussian distribution for each cluster. Explain the difference between the two algorithms in terms of parameter estimation of GMMs.

(a) Using MLE would require the prior probability of the Gaussian model, which cannot be easily computed since we will need to calculate partial derivatives of the log likelihood, and when there are many Gaussians it becomes very difficult.

(b) For EM, we will take an iterative approach, much like with k-means, where we can start out randomly, then iteratively update the labels we currently have for all the data points, then update the parameters with respect to the updated data points to better fit the updated data points, until stop criterion is met. For MLE with k-means, we process of finding parameter mean relies on k-means for the k cluster centres. Then, starting from the cluster centres, MLE will be performed to find parameter variance to maximise the log likelihood, which is not iterative and requires solving for partial derivatives.