Inf 2B Coursework-2
Task-3 Report
Bayes classification with Gaussian distributions

UUN: s1616497

April 13, 2018

Abstract

The aim of task three is to use Bayes classification with Gaussian distributions in order to classify a dataset of images into one of 26 different classes: representing the letters of the English alphabet.

1 Code Description Normal Gaussian

1.1 File 1: $my\_gaussian\_classify.py$

Parameter list:
$Xtrn$, $Ctrn$, $Xtst$, $epsilon$

Return value:
$Cpreds$: Nx1 matrix of predicted classes of $Xtst$ matrix feature vectors
Ms : DxK matrix of mean vectors where Ms[:, k] is the sample mean of class k
$Covs$: DxDxK 3D array of covariace matrices, where Covs[:, :, k] is the covariance matrix (after regularization) for class k.

In order to proceed with Bayes classification with Gaussian distributions process, the first step *vectorization* was done through the use of *np.compress*() for extracting the feature vectors belonging to a particular class. Also, *np.sum*() was used to calculate the *mean vectors*.
np.dot() was utilized to calculate the Covariance matrices corresponding to each of the class.

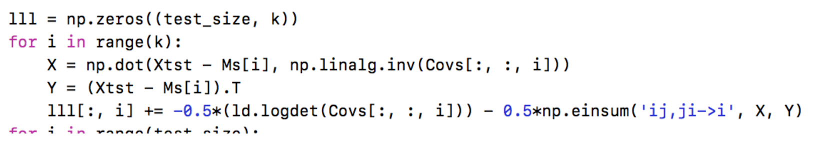
At first, the code took about 75 seconds to run. But, after a bit of matrix manipulation and the use of *np.einsum*(), the run-time was brought down drastically.:

Formulae Used:

$log\ posterior\ probability$ = -0.5$log|\Sigma_i|$ - 0.5$(x - \mu_i)\Sigma^{-1}(x - \mu_i)$

Where:

- $\Sigma_i = Covariance$ matrix for class i
- μ_i = mean vector for class i
- x = feature vector from $Xtst$



```
lll = np.zeros((test_size, k))
for i in range(k):
    X = np.dot(Xtst - Ms[i], np.linalg.inv(Covs[:, :, i]))
    Y = (Xtst - Ms[i]).T
    lll[:, i] += -0.5*(ld.logdet(Covs[:, :, i])) - 0.5*np.einsum('ij,ji->i', X, Y)
for i in range(test_size):
```

Figure 1: Snippet of code for log posterior probability calculation.

The datatypes of the elements in the matrices were changed to float 32 which also helped speed up the calculation procedure.
np.einsum() was used to do only the computations required to obtain the diagonal of the equation shown in Figure 1, all while neglecting the rest of the computations in the matrix product of X and Y.
Where:
X = -0.5$(Xtst - \mu_i)\Sigma_i^{-1}$
Y = $(Xtst - \mu_i).T$

1.2 File 2: $my\_confusion.py$

Parameter list:

- *Ctrues* : The correct classes of the test vectors(Shape: Nx1)
- *Cpreds* : The predicted classes of the test vectors(Shape Nx1)

Return value:

- CM : K-by-K confusion matrix, where $CM[i][j]$ is the number of samples whose target is the $i'th$ class that was classified as j. K is the number of classes, which is 26 for the data set.
- *acc* : The ratio of right predictions to total predications

There was no need of vectorization in this case. Just a for-loop iterating over all the predications was used.

1.3 File 3: $my\_gaussian\_system.py$

The function *time.clock*() was used which is a part of the package *time*.
Used *scipy.io.savemat*() to store the $Confusion\ Matrix$, the 26th class's *mean vector* and the 26th class's $Covariance\ Matrix$ in the respective files.

2 Values Obtained

Prediction Size	Number of Errors	Accuracy
7800	1211	0.844743589744

The time taken for the $my\_gaussian\_classify.py$ file to execute is 16.11 seconds.

3 Code Description Improved Gaussian

3.1 File 4: $my\_improved\_gaussian\_classify.py$

Parameter list:
$Xtrn$, $Ctrn$, $Xtst$

Return value:
Cpreds : Nx1 matrix of predicted classes of $Xtst$ matrix feature vectors

The mean vector for the Xtrn was calculate using *np.sum*()

In order to speed up calculation and improve accuracy, PCA was used.
The $Covariance\ Matrix$ was calculated using the method similar to the one mentioned in section 1.1 .
Next, the eigenvalues and the corresponding eigenvectors were derived using the *np.linalg.eig*() function.
To speed up sorting, quicksort was used on the tuples of eig values and eig vectors using the eigen values as the keys.

The dot product of the matrix containing the eig vectors corresponding to the highest K eig values and Xtrn was taken to reduce the dimensions from D to K.
The same was done to Xtst to reduce the dimensions of the test feature vectors from D to K. *my_gaussian_classify*() was then called on the dimensionally reduced Xtrn and Xtst and the predictions were obtained.

4 Values Obtained

Prediction Size	Number of Errors	Accuracy
7800	927	0.881153846154

The time taken for the *my_gaussian_classify.py* file to execute is 2.68 seconds.