LAB_01 REPORT FOR Pattern Recognition

Tryfonopoulos Dimitrios – edem0340037

STEP 1

Read the data from the file. The data must be read in a format compatible with scikit-learn in 4 tables X_train, X_test, y_train and y_test. The X_train table contains the training samples, without the labels) and has dimensions (n_samples_train x n_features). Y_train is a one-dimensional array of lengths n_samples and contains the corresponding labels for X_train. Respectively for the test data.

We read the data using np.loadtxt since our file is in .txt format. For X_train & X_test we keep all the data except those of the first column since they contain the label of each digit. The elements (labels) of the first column are read as y_train & y_test since they are the actual predictions for each element.

STEP 2

Draw the 131-digit (located in position 131) of the train data. Tip: use the numpy.reshape function to organize the 256 attributes in a 16x16 table, and the matplotlib.pyplot.imshow function to display the digit.

To display the element in position 131 we select the characteristics of this digit from X_train [131 ,:], we change the shape of the vector to 16x16 to be able to display them.

STEP 3

Choose 1 random sample from each label, a total of 10 samples). Draw them in a figure with subplots. (Hint: fig = plt.figure (); fig.add_subplot (,,,))

Using the random function, we randomly select for each number from 0-9, one of all the samples, after we have first kept each number in a table repeatedly. After changing the shape again from line vector to 16x16 we can display each randomly selected sample for each of the numbers.

STEP 4

Calculate the average value of the pixel characteristics (10, 10) for the digit "0" based on the train data.

After holding only the samples that contain the number ‘0’ we find the index for the number in the pixel (10,10) and we calculate its average value (np.mean) 10 * 16 + 11 = 171.

Another solution to the above problem via the np.reshape function.

STEP 5

Calculate the scatter of pixel characteristics (10, 10) for the digit "0" based on the train data.

Corresponding implementation to the previous step but instead of np.mean we will use np.std.

STEP 6

Calculate the average value and spread of the characteristics of each pixel for the digit "0" based on the train data.

After keeping in a table (A) all the samples representing the digit ‘0’ based on the labels from the y_train data. We get the average value per column (axis = 0) and so we find the average value for each pixel of the number zero for all samples.

STEP 7

Draw the digit "0" using the mean values you calculated in Step 6.

Using the mat_mean table we found above and after rearranging it we draw the digit ‘0’ based on the average value of each pixel (feature).

STEP 8

Draw the digit "0" using the dispersion values calculated in Step 6. Compare the result with the result of Step 7 and explain any differences.

Drawing the digit ‘0’ using the values of the scatter for all its pixels we notice that the part ‘is missing’ from the digit in the middle, expected since the scatter symbolizes the + - deviations from the mean value. For the best display, the report shows the sum and the difference between the mean and the deviation using the subplot command.

STEP 9

(a) Calculate the average value and scatter of attributes for all digits (0-9) based on the train data. (b) Draw all the digits using the mean values calculated in Step 9 (a).

Corresponding to the previous questions we create a table of dimensions (10,256) where 10- different numbers, 256 -values where are the average value for each pixel for each number.

Then after rearranging them in (16x16) we design them.

STEP 10

Sort the number 101 digits of the test data (located at position 101) into one of 10 categories (each of the 10 digits, 0-9, representing one category) based on the Euclidean distance1 (hint: use the values you calculated in Step 9 (a)). Was the classification successful?

After calculating the Euclidean distance between each sample from the mean of each attribute we calculated in the previous examples, we keep the minimum, together with the corresponding index (np.argmin) which represents the number with which that sample was found to have the shorter Euclidean distance, the corresponding prediction that is. However, we finally notice that the classification was not a correct fact that can be explained if we graphically represent the specific sample. We will find that it is the number 6 instead of 0 but due to the way it is written it looks like 0.

STEP 11

(a) Classify all digits of the test data into one of 10 categories based on Euclidean distance. (b) Calculate the success rate for Step 11 (a).

We repeat the above classification process based on the Euclidean distance. Success rate is ~ 81.42%

STEP 12

Implement the Euclidean distance classifier as a scikit-learn estimator. (Hint: the lib.py file you are given)

Following the form of the classes of sklearn classifiers (_init_, fit, predict, score) we create the Euclidean classifier function.

STEP 13

a) Calculate the Euclidean classifier score using 5-fold cross-validation b) Draw the Euclidean classifier decision area. c) Draw the learning curve of the Euclidean classifier.

a) Calling the cross_val_score function from the package sklearn.model_selection, and the K-fold with number of partitions n_splits = 5 we find the score of this classifier.

b) From the tutorial_Skikit-learn of the course we copy the plot_clf function which is used to create the Decision Surface for the Euclidean classifier.

c) We draw the learning curve using the learning_curve function from sklearn.model_selection.

STEP 14

Calculate the a-priori odds for each class (class priors).

I calculate the a priori as the value of the number of samples for each digit in relation to the number of digits.

STEP 15

a) Classify all the digits of the test data into the 10 categories using the mean value and dispersion values calculated in Step 9 (a), using a Naive Bayesian classifier2. Do not use ready-made implementations. Your implementation must be compatible with scikit-learn as shown in Step 12. b) Calculate the score for Step 15 (a). c) Compare your implementation of Naive Bayes with the implementation of scikit-learn (GaussianNB).

a) Different implementations for .pynb and lib.py

b) Using the accuracy score I calculate the score of the previous step 0.82

c) Comparing the odds the custom_NaiveBayes classifier behaves 14% better

STEP 16

Repeat Step 15 (a), (b) assuming that the dispersion for all characteristics, for all categories is equal to 1.

For the same var number for all samples the classifier performance was better than expected. With a different var I have different predictions each time. The lower the var, the better the predictions.

STEP 17

Compare the performance of Naive Bayes, Nearest Neighbors, SVM classifiers (with different kernels). You can use scikit-learn implementations.

In the implementation of the algorithms, we can observe the different performance of the classifiers. For Nearest Neighbors we observe that the higher the number of neighbors the better the classifier performance.

STEP 18

The basic idea of this step is to combine some classifiers with a fairly good performance in order to achieve a performance higher than the individual performance. This technique is known as ensembling. It is important that the classifiers that are combined are characterized by a different type of error, for example, one classifier tends to mistype the digit 3, while another tends to mistype the digit 7. Select some of the classifiers you used in the previous steps. Use the scikit-learn VotingClassifier to combine them in hard or soft voting. This meta-classifier combines the individual classifiers by letting them vote for the result. You must select a single number of classifiers. Why;

We used the selection of the above classifiers as they had the best performance on the previous queries. The only number of classifiers manages to avoid having draws in their scores.