REPORT

1. General

I used the R language for Part 1 and Python for Part 2 as slow performance of R was expected. The implementation of each subtask is provided as a separate file with code. The filenames are self-explanatory and include subtask IDs. All code ran successfully on my machine, and the enclosed log.txt file contains the R output generated automatically using the *sink* function and the Python output copied from the console. Please comment the *sink* function in R code files to enable the output directly to the console. Each file has an initial comment block showing all the possible sources of information used to create the code.

2. Problem 1

Task: use the Pima Indians diabetes dataset for classification [http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes]. This data has a set of attributes of patients, and a categorical variable telling whether the patient is diabetic or not (1 and 0, respectively). Also, a value of 0 may indicate a missing value for an attribute.

Part A

Build a simple naive Bayes classifier from scratch to use 20% of the data for evaluation and the other 80% for training (768 data-points altogether). Use a normal distribution to model each of the class-conditional distributions.

The accuracy (number of correct predictions as a fraction of total predictions) of the classifier on the 20% evaluation data:
<u>0.74118</u>

In several consecutive runs, this number varies and is usually lower than the training accuracy (see the log file or run the program yourself) which can be explained by the fact that the classifier does not train on the test dataset. On rare occasions, this number can be almost equal or even exceed the training accuracy which is probably due to the random nature of the train-test data split.

Modifications that I made to the sample code:
- added prior to each log likelihood equation;
- modified variable names to be more meaningful.

Part B

Adjust the code so that, for attributes 3, 4, 6, and 8, a value of 0 was regarded as a missing value when estimating the class-conditional distributions, and the posterior.

The accuracy of the classifier on the 20% evaluation data:
<u>0.73529</u>

In several consecutive runs, it can be clearly seen that both the training and testing accuracies in this case are slightly lower than in Part A which is due to the introduction of NA values and the way they are handled by R functions (e.g. reducing the number of elements in a row). The same as for Part A, the testing accuracy is usually slightly lower than the training one, but sometimes both values can be comparable (randomness of train-test split).

The same modifications were made here as in Part 1A.

Part C

Use the caret and klaR packages to build a naive bayes classifier, assuming that no attribute has a missing value (with 10-fold cross-validation).

The accuracy of the classifier on the 20% evaluation data:
<u>0.7778</u>

On an average, the accuracy of this classifier is usually higher than in Parts A and B, but the range of its values is much broader: from approx. 70 to approx. 82 while the accuracy from Parts A and B fluctuates within a much smaller range (on the order or 2-3%)

Part D

Train and evaluate an SVM (using SVMLight) to classify this data (do not substitute NAs for zeros for attributes 3, 4, 6, and 8).

The accuracy of the classifier on the 20% evaluation data:
<u>0.76471</u>

Part D (Elaborate)

The same as Part D, but a more elaborate version of SVM was used to include squared features and 3 error penalties (trade-offs between maximum margin and classification error during training).

SVM accuracy on the 20% evaluation data:
<u>0.65359, 0.65359, 0.77124 (for error penalties of 0.005, 0.01, 0.1, respectively)</u>.

These results are not always consistent, e.g. the first accuracy (for the error penalty of 0.005) can be even around 35, the second one can be around 65, and the third one in the seventies. I even saw instances when the third accuracy (for the error penalty of 0.1) was in the thirties. However, in most cases the three accuracies have an increasing trend and grow with the value of the error penalty. Some randomness of results can probably be explained by the way the original dataset is split for training/testing.

3. Problem 2

Use the MNIST dataset of 60,000 training and 10,000 test examples of handwritten digits which has 10 classes ("0" to "9") [http://yann.lecun.com/exdb/mnist/]; the dataset is stored in an unusual format. Use your or a third-party reader.

Do this for two cases:

- *Untouched images: as is, 28 × 28.*

- *Bounding box: find the horizontal and vertical ink range, cut it out of the original image, resize the resulting image to 20 × 20.*

I used an R-based reader which converts the original dataset into two csv files: train.csv and test.csv (you can also download the two data files from this post in the interest of time). Then I used pandas in Python to read from the csv files, set the training and testing features and labels and train/test the three required classifiers: Gaussian and Bernoulli Naive Bayes and Random Forest (with the required depth and number of trees). I used the OpenCV module in Python to conduct manipulations with images.

Part A

Accuracy	**Gaussian**	**Bernoulli**
untouched images	0.5559	0.8415
stretched bounding box	0.841	0.834

It is quite evident that the bounding and stretching of images had a very positive effect on the Gaussian NB classifier, and its accuracy has significantly improved. The Bernoulli NB classifier shows a similar, relatively high accuracy for both groups of images.

Part B

Untouched raw pixels:

	depth = 4	**depth = 8**	**depth = 16**
#trees = 10	0.759	0.9047	0.9496
#trees = 20	0.7935	0.9215	0.9583
#trees = 30	0.8059	0.9207	0.9618

Stretched bounding box:

	depth = 4	**depth = 8**	**depth = 16**
#trees = 10	0.7444	0.9094	0.9519
#trees = 20	0.795	0.9203	0.9639
#trees = 30	0.8003	0.9235	0.9645

One can see from these numbers that the accuracy somewhat increases with the number of trees, but the biggest increase in the accuracy is with depth. This is probably due to the fact that we get more features by increasing the depth because we have a somewhat large dataset with 10 different classes, and the increased depth does not lead to overfitting at the level of 16. Although, overfitting is possible if we further increase the depth.

If one compares the performance of the classifier on the untouched images vs. bounded and stretched images, there is hardly any difference for the depths of 4 and 8 for any number of trees, but there is a very slight improvement for the depth of 16 when Random Forest is applied to the modified images.