Schaffhausen Institute of Technology | Assignment 2
2021/22 Term 2 | Dynamic Analysis and Machine Learning
Assigned Date: 05/04/2022
Due Date: 18/04/2022

Due Date

Late assignments will not be accepted and will receive ZERO mark.

Objectives

The objectives of this assignment are as follows.

- **Appling KNN and SVM to solve the problem of malware detection.**
- **Solving the problem of fault localization by using Artificial Neural Networks, employing coverage data from a test suite by means of virtual test cases methodology.**
- **Fine-tuning the hyper-parameters of your models using cross validation and validation set approach.**
- **Demonstrating your understanding of Dimensionality Reduction and PCA on how to obtain best PC's.**

1 Malware Detection [2 Points]

In this task, you will use the Android traffic dataset, which you used in Assignment-1, to solve the problem of Malware detection:

1. **Read data in Python. Then rescale it with a scaler of your choice.**
2. **Split your data into train and test sets (80% and 20% respectively).**
3. **Before you start building the models, analyze the data (see the number of features, etc.) and tell us whether you expect kNN or SVM to perform better on this problem. Justify your answer.**
4. **Implement kNN for malware detection. You must perform hyper-parameter tuning for your model on the following hyper-params: {n_neighbors=range(1,10), weights=['distance', 'uniform'], metric=['euclidean', 'manhattan', 'chebyshev', 'cosine']}. Print the mean score and its standard deviation for each of the built models. Print the best hyper-paramters.**
5. **Implement SVM for malware detection. You must perform hyper-parameter tuning for your model on the following hyper-params: {C: np.linspace(1, 2, num=5), kernel: ['rbf', 'sigmoid'], gamma: np.logspace(-7, -3, num=5)}. Print the mean score and its standard deviation for each of the built models. Print the best hyper-parameters.**
6. **Train both kNN and SVM using the best hyper-parameters on the train set, and print the accuracy on the test set for both models. Do the results support your answer to 1.3?**

2 Fault Localization [3 Points]

This question is based on the work of Zheng et.al [1] on fault localization, which is also included in Week 6's required reading. Therefore, you must read their the paper before solving this task.

In summary, given the coverage data of several test cases for a computer program, in this question, you will build an Deep Artificial Neural Network (DNN) that will predict

the probability-of-being-faulty for each statement of the program.

Have a look at Figure 1, which represents the coverage data of 10 test cases for a computer program that has 12 executable statements. Each row represents a test case while each column (except the last one) represents a statement. The cell $[t_i, s_j] = 1$ means that the i-th test case covers the j-th statement. The last column (r) indicates whether the test t_i failed or not ($1 \longrightarrow fail$, $0 \longrightarrow success$).

More specifically, you are required to perform the following steps on the attached dataset 'coverage.csv':

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	r
t_1	1	1	1	0	1	1	0	0	0	0	0	1	0
t_2	1	1	1	1	0	0	0	0	0	0	0	1	0
t_3	1	1	0	0	0	0	1	1	1	0	0	1	0
t_4	1	1	0	0	0	0	1	1	0	1	0	1	0
t_5	1	1	1	0	1	1	0	0	0	0	0	1	0
t_6	1	1	1	0	1	0	0	0	0	0	0	1	0
t_7	1	1	0	0	0	0	1	1	1	0	0	1	0
t_8	1	1	0	0	0	0	1	1	1	0	0	1	0
t_9	1	1	1	0	1	1	0	0	0	0	0	1	1
t_{10}	1	1	1	0	1	1	0	0	0	0	0	1	1

Figure 1

1. **Construct a DNN model that has one input layer, one output layer, and 3 hidden layers. The number of nodes in the input layer would be the same as number of statements in the coverage data, whereas the number of nodes in each hidden layer should be simply set as 32. The output layer would have just one node (representing the execution result r) with sigmoid activation function. The activation function in the hidden layers should be the relu function.**

2. **Use the coverage data as the training set to fit/train your DNN model to obtain the complex nonlinear mapping relationship between the coverage data of a test case and its execution result. Regarding the hyper-parameters: Optimizer=Adam(lr=1e-5), epochs=30.**

3. **In this step, you will construct the set of virtual test cases. Once again, to get a better understanding of this set, refer to [1]. In summary, this set will have a structure as shown in Figure 2, consisting of the same number of statements as in the coverage data. Note that here each row would represent a virtual test case which covers only one statement of the given computer program. Since each test case only covers one statement, thus the statement would be highly suspicious if the corresponding test case is failed.**

4. **Input the coverage data vector of virtual test set to the trained DNN model and get the output for each test case, which would reflect the probability that its corresponding executable statement has a bug. Finally, rank the statements according to the predicted probability scores.**

5. **Output a csv file with 3 columns:**

   - **statement: shows the statement number.**
   - **suspiciousness: indicates the probability that the statement is a bug.**
   - **rank: the rank of the statement to be debugged by a developer.**

6. **(Bonus task) Tune the following hyper-parameters to achieve the best accuracy on a validation split (15% of the training set).**

   - **Number of layers.**
   - **Number of neurons in each layer.**
   - **adding dropout layers.**
   - **optimizer and learning rate.**
   - **activation functions.**

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
t_1	1	0	0	0	0	0	0	0	0	0	0	0
t_2	0	1	0	0	0	0	0	0	0	0	0	0
t_3	0	0	1	0	0	0	0	0	0	0	0	0
t_4	0	0	0	1	0	0	0	0	0	0	0	0
t_5	0	0	0	0	1	0	0	0	0	0	0	0
t_6	0	0	0	0	0	1	0	0	0	0	0	0
t_7	0	0	0	0	0	0	1	0	0	0	0	0
t_8	0	0	0	0	0	0	0	1	0	0	0	0
t_9	0	0	0	0	0	0	0	0	1	0	0	0
t_{10}	0	0	0	0	0	0	0	0	0	1	0	0
t_{11}	0	0	0	0	0	0	0	0	0	0	1	0
t_{12}	0	0	0	0	0	0	0	0	0	0	0	1

Figure 2

3 PCA [1 point]

You last task will be to figure out the optimal number of Principal Components (PCs) for the Digits dataset (from sklearn.datasets import load_digits), which initially consists of 64 numeric features. You need to apply the so called *elbow method*. Its essence is as follows. You plot the special graph called "scree plot", which shows the proportion of explained variance for each principal component (monotonically decreasing), as in Figure 3. In this plot it is clear that starting from a certain point, resembling human elbow (3-4 here), adding new principal components doesn't add much in terms of explained variance. Or, in other words, diminishing returns are no longer worth the additional cost. Hence, this is the point to stop - we take only 4 first PCs, and ignore the rest of them. This is a simple intuitive heuristic which helps to identify the appropriate number of PCs when there is a trade-off between the number of features and the amount of variance we want to preserve.

Your task will be to apply PCA, produce the scree plot for the given data and decide the optimal number of PCs to keep. Calculate and print their total explained variance. Thoroughly justify your choice - this is the main part of the work for this task.

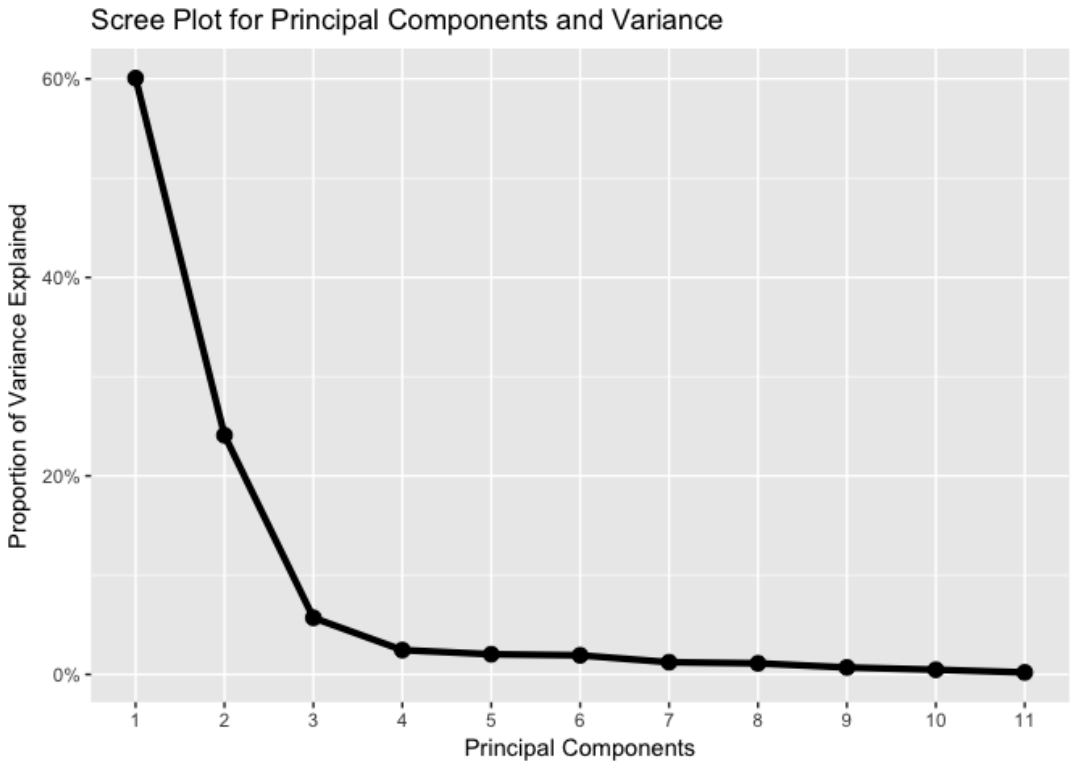

Figure 3: Scree plot.

Notes

- **Cheating is a serious academic offense and will be strictly treated for all parties involved. So delivering nothing is always better than delivering a copy.**
- **Code cleanness and style are assessed. So maybe you want to take a look at our references: Link 1 and Link 2.**
- **Organize your notebook appropriately. Divide it into sections and cells with clear titles for each task and subtask.**

References

[1] **Wei Zheng, Desheng Hu, Jing Wang, "Fault Localization Analysis Based on Deep Neural Network", Mathematical Problems in Engineering, vol. 2016, Article ID 1820454, 11 pages, 2016. https://doi.org/10.1155/2016/1820454**