Open Science in Statistics and Machine Learning Report

Master Program of Data Science
Ludwig-Maximilians-Universität München

Xiaohan Sun

Munich, Jan 26th, 2023

Supervised by Andreas Bender, Martje Rave, Giacomo de Nicola, Lisa Wimmer, Felix Henninger

Contents

1 Introduction **1**

2 Book Club Phase **1**
2.1 Introduction . . . 1
2.2 Measuring Procedures . . . 1
2.2.1 Cross-Validation . . . 2
2.2.2 Accuracy Indicators . . . 2
2.3 Research Methods . . . 3
2.3.1 Simulation of the Data . . . 3
2.3.2 Prediction Models and Results . . . 3
2.4 Conclusion . . . 4

3 Project: Code Sharing in R Using rocker/docker **5**
3.1 Mushroom Classification in R . . . 5
3.1.1 Dataset . . . 5
3.1.2 Data Visualization . . . 5
3.1.3 Models and Results . . . 6
3.2 Docker for Reproducibility in R . . . 7
3.2.1 Docker . . . 7
3.2.2 Dockerfile . . . 7
3.3 Create the Docker Image with “dockr” Package . . . 8
3.4 Project Reproduction . . . 9
3.4.1 Reproduction Result . . . 9
3.4.2 Comparison between Two Projects . . . 9

4 Conclusion **10**

5 Acknowledgement **11**

1 Introduction

Open science has gained popularity and has been widely adopted in recent decades. It refers to the practices and principles of making scientific methodologies and output, such as data, publications, and software, accessible, reproducible, and transparent [5]. It has become increasingly important due to many reasons. For example, it enables the sharing of resources among researchers, leading to faster collaboration in the field. Also, open science helps to make high-quality data and resources accessible, and thus the accuracy and validity of results could be improved. However, there are also many obstacles to the widespread adoption of open science, such as concerns about intellectual property and data privacy and the need for adequate recognition for making the work available. We first read some related articles from this course to learn more about open science in statistics and machine learning. Then we implemented the projects to explore topics such as achieving reproducibility using different tools. The following two sections introduce the paper I read and the project I implemented in detail.

2 Book Club Phase

2.1 Introduction

The paper assigned to me is named **"Reliability and Validity in Comparative Studies of Software Prediction Models."** It detected the unreliability of one of the most widely used measuring procedures in software engineering. Although a large number of prediction models, such as CART, Regression Analysis, and Artificial Neural Networks, emerged in the past 20 years, predicting software development efforts is still a big obstacle nowadays. The question of "Which prediction is best?" remains unsolved for empirical studies even with such substantial research activities [7]. Some possible explanations are provided from this paper: First, the quality of execution varied for different experiments, such as research procedure and data quality; Second, the sample size is not large enough to support the software studies; And lastly, which is also the concentration of this paper, stated that the measuring procedures are not able to consistently select the best from two competing models, leading to the problem of invalidity and unreliability [7]. To detect this problem of the lack of convergence, the authors evaluated the research procedures by simulating the data points from the population, comparing linear and nonlinear prediction models, and exploring several commonly used accuracy indicators. More details are illustrated below.

2.2 Measuring Procedures

Two basic but fundamental properties need to be possessed for measuring procedures: reliability and validity. Reliability is achieved when consistent experimental results are obtained when conducting repeated measuring procedures, tests, or other experiments, and validity is aimed at keeping what the measurements intend to do [6]. In comparative studies of prediction models in software engineering, the commonly employed measuring method involves conducting cross-validation on a single sample and evaluating it with

one or multiple accuracy indicators, which is applied for both the basic measure and the summary statistic of the measure [7]. Then, the model with the lowest value of the accuracy indicator is selected as the "best." The following two subsections provide further details on cross-validation and the accuracy indicators for the final model chosen.

2.2.1 Cross-Validation

Compared with v-fold cross-validation, n-fold cross-validation is selected to achieve nearly unbiased estimators from this paper. The algorithm is shown in Algorithm 1. It is chosen because it is a widely used variant, and the authors think this algorithm comes closer to a real-world situation since the model only left out one observation than the v-fold, which used a much smaller subset, leading to the approximation better.

Algorithm 1: N-fold Cross Validation [7]

1. Deleting the observations from the data set one at a time.
2. Calibrating the model to the $n-1$ remaining observations.
3. Measure how well the calibrated model predicts the deleted observation.
4. Averaging these predictions over all n observations.

2.2.2 Accuracy Indicators

Seven accuracy indicators are presented in the paper to evaluate the model performance; three are listed in this section. First is the most widely used evaluation criterion: the *mean magnitude of relative error (MMRE)*. When it takes values less than 0.25, it is considered acceptable for effort prediction models [7]. *MRE* is defined as:

$$MRE = \frac{|y - \hat{y}|}{y} \tag{1}$$

where y=actual and ŷ=prediction.

Another simple measure is the *standard deviation (SD)*, positively correlated with size. It is computed as follows:

$$SD = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n-1}} \tag{2}$$

Also, the *relative standard deviation (RSD)* is extended from *SD* but is used as a relevant measure of the prediction model's performance. The equation is listed below:

$$RSD = \sqrt{\frac{\sum \left(\frac{y_i - \hat{y}_i}{x_i}\right)^2}{n-1}} \tag{3}$$

2.3 Research Methods

2.3.1 Simulation of the Data

The Finnish data set [4] was used as the population to simulate the data. The dataset contains 40 function points (EP), and 38 spans from 460 to 23,000 workhours. Based on Figure 1, the data are heteroscedastic, which means it has an increasing variance. For the simulation, the variables are the same as those from the population, and the sample size remains 40. The model chosen as the population to simulate is:

$$\ln(\mathrm{y}) = 1.70 + 1.05 \cdot \ln(\mathrm{x}) + \mathrm{u}, \sigma = 0.79. \tag{4}$$

where u is the error term, σ is standard deviation.

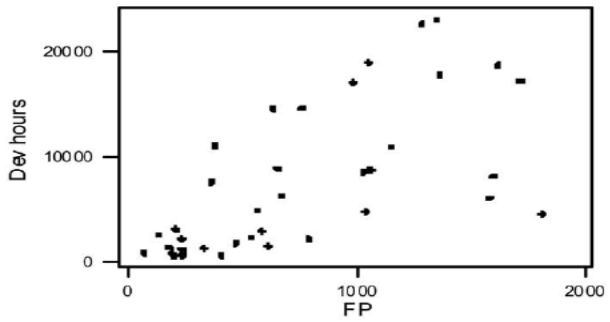


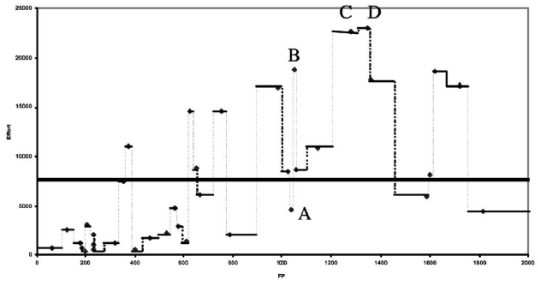
Figure 1: Plot of effort versus FP for Finnish Data Set [7].

2.3.2 Prediction Models and Results

There are two models specified to compare. First is the linear regression model fitted to the data using the *ordinary least squares (OLS)* method. The final log-log, additive, linear *OLS* model is in the form of the following:

$$\ln(y) = \ln(A) + B \cdot \ln(x). \tag{5}$$

The second model implemented in the paper is the *estimation-by-analogy (EBA)* model, which is representative of *arbitrary function approximators (AFAs)*. Unlike Functions such as the linear regression model used above, *AFAs* do not make any assumption about the relationship between the variables and the predictor. The *EBA* model's functional form is depicted using the nearest neighbor. Taking Figure 2a as an example to estimate the effort for a 1,300 FP project, point C is closest, which has 1,282 FP and 22,670 workhours. Then, for k=1, we take 22,670 as the estimated workhours for the 1,300 FP project. If k=2, we use the result calculated by the average of C and D's workhours [7].



(a) EBA for the Finnish Data Set.

Variable	N	AFA(%)	OLS (%)
MAR	1000	15.4	84.6
MMRE	1000	99.4	0.6
MMER	1000	2.8	97.2
MBRE	1000	8.1	91.9
MIBRE	1000	30.6	69.4
RSD	1000	29.2	60.8
LSD	1000	68.4	31.6

(b) Comparison Results.

Figure 2: Models and Results [7].

Figure 2b presents the results for the comparisons. For example, the *AFA* model achieves 99.4 percent high accuracy using *MMRE*, while *MMER* chooses *OLS* with an accuracy of 97.2 percent. It further indicates the inconsistency when measuring the models with different accuracy indicators.

2.4 Conclusion

The lack of convergence of the question "Which prediction model is best?" is of utmost importance as it can lead to conflicting results, reducing the reliability of comparative studies of the software prediction models. The paper demonstrated the unreliability of one of the measuring procedures used for software prediction models by measuring the performance using different accuracy indicators for the linear regression and the EBA models. However, some flaws of this paper need to be addressed here. First, the authors should have addressed the specific reasons for comparing the linear regression and the EBA model to reach the rationality of the comparing results. Different models make different assumptions about the data, and the choice of model will affect the accuracy of the predictions. Second, it seemed reasonable to me when the accuracy indicators did not choose the same model since different indicators may be more or less appropriate for various prediction problems and data types. Third, although the authors used simulation to enhance the dataset, they did not specify the quality of the data, which should be carefully evaluated. Therefore, the results of these studies can be influenced by various factors, leading to the difficulty in determining which prediction model is best. What the authors addressed could be one of the reasons causing the measuring procedure to be unreliable, but it should not be solely considered.

3 Project: Code Sharing in R Using rocker/docker

This project has three major tasks:

1. Analyze the mushroom classification using R.
2. Use Docker to reproduce the analysis result from the first task.
3. Create the Docker image using the "dockr" package.

I completed the first two tasks successfully, but the implementation of the third task illustrates a failure since the package is not supported on CRAN anymore. I will detail each task to demonstrate the success and failure in the following.

3.1 Mushroom Classification in R

3.1.1 Dataset

The dataset applied for this task was created by Dennis Wagner on 05 September 2020. It includes 61069 hypothetical mushrooms with caps based on 173 species drawn from the sourcebook: *Patrick Hardin.Mushrooms & Toadstools* [2]. It contains 16 variables, such as *"cap.diameter"* which is documented in float number in cm; *"habitat"* which uses letters to represent each factor, as g stands for grasses, l stands for leaves, and so on. More details are shown in Figure 3 below. The dataset has the binary class, which means the mushrooms are either edible or poisonous. From Table 1, we can also observe that the dataset contains more poisonous mushrooms than edible ones.

```
'data.frame':   61069 obs. of  16 variables:
 $ class               : Factor w/ 2 levels "edible","poisonous": 1 1 2 2 1
 $ cap.diameter        : num  1.26 10.32 0.92 4.27 3.08 ...
 $ cap.shape           : Factor w/ 7 levels "b","c","f","o",..: 7 3 7 7 3 6
 $ cap.surface         : Factor w/ 11 levels "d","e","g","h",..: 3 2 3 9 8
 $ cap.color           : Factor w/ 12 levels "b","e","g","k",..: 12 1 8 8 1
 $ does.bruise.or.bleed: Factor w/ 2 levels "f","t": 1 1 1 1 1 1 1 2 2 2 ..
 $ gill.attachment     : Factor w/ 7 levels "a","d","e","f",..: 2 1 1 7 2 4
 $ gill.spacing        : Factor w/ 3 levels "c","d","f": 1 1 1 1 2 3 1 2 1
 $ gill.color          : Factor w/ 12 levels "b","e","f","g",..: 11 1 8 11
 $ stem.height         : num  5.04 4.68 4.59 4.55 2.67 3.2 4.87 6.94 8.03 7
 $ stem.width          : num  1.73 19.44 1.15 6.52 5.18 ...
 $ stem.color          : Factor w/ 13 levels "b","e","f","g",..: 13 12 5 12
 $ has.ring            : Factor w/ 2 levels "f","t": 1 2 1 1 1 1 1 1 2 1 ..
 $ ring.type           : Factor w/ 8 levels "e","f","g","l",..: 2 2 2 2 2 2
 $ habitat             : Factor w/ 8 levels "d","g","h","l",..: 1 1 1 1 5 1
 $ season              : Factor w/ 4 levels "a","s","u","w": 1 1 3 1 1 1 1
```

Figure 3: Overview of the Dataset.

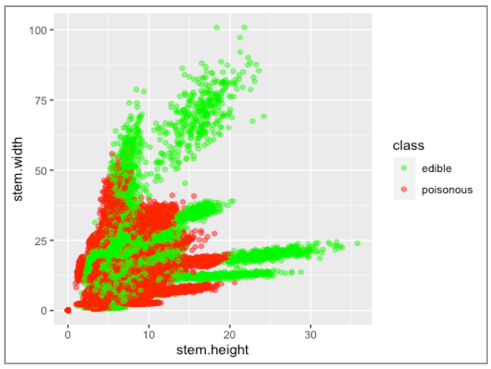
	edible	poisonous
count	27181	33888

Table 1: Count of the Binary Class.

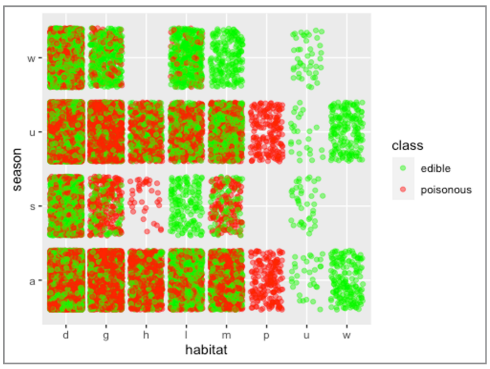
3.1.2 Data Visualization

To better understand the dataset, I generated two plots to detect the relationship between different variables for data visualization. For each plot below, red dots are data points from

the poisonous class, and green dots are from the edible class. Figure 4a is plotted using *"stem.height"* vs. *"stem.width"*, which shows that mushrooms with less stem height and less stem weight tend to be poisonous. Figure 4b used *"habitat"* as the x-axis and *"season"* as the y-axis. We can obtain much information from this plot, such as mushrooms with habitats of urban and waste are always edible no matter what season they appear, and mushrooms with the habitat of paths are more likely to appear in summer and autumn. They are all poisonous.



(a) stem.height vs. stem.width.



(b) habitat vs. season.

Figure 4: Relationships between Different Variables.

3.1.3 Models and Results

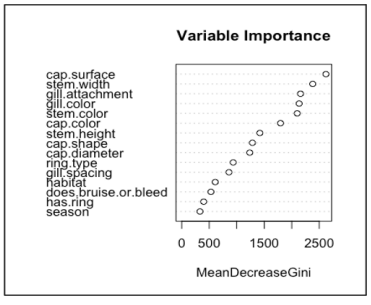
I implemented two models to compare the performance for the final classification task: *Random Forest Classifier* and *Support Vector Machine (SVM)*. I trained both models on 70% of the available data and computed the test performance on the remaining 30%. Figures 5a and 5b are the confusion matrix for each model, and the results showed that *Random Forest Classifier*, with an accuracy that is almost 100% performed much better than *SVM*, which accuracy only reached 79.5%. Therefore, *Random Forest Classifier* was selected and used to generate the variable importance plot. Figure 5c revealed top three variables that are most important to influence the classification are *"cap.surface"*, *"stem.width"*, and *"gill.attachment"*. For the last step, figure 5a is saved to another txt file, which will be recreated again when we run the Docker image successfully later.

```
Confusion Matrix and Statistics

          Reference
Prediction  edible poisonous
  edible      8153         0
  poisonous      1     10166

               Accuracy : 0.9999
                 95% CI : (0.9997, 1)
    No Information Rate : 0.5549
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9999

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9999
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9999
             Prevalence : 0.4451
         Detection Rate : 0.4450
   Detection Prevalence : 0.4450
      Balanced Accuracy : 0.9999

       'Positive' Class : edible
```

(a) Random Forest Classifier.

```
Confusion Matrix and Statistics

          Reference
Prediction  edible poisonous
  edible      6530      2126
  poisonous   1624      8040

               Accuracy : 0.7953
                 95% CI : (0.7894, 0.8011)
    No Information Rate : 0.5549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5881

 Mcnemar's Test P-Value : 2.808e-16

            Sensitivity : 0.8008
            Specificity : 0.7909
         Pos Pred Value : 0.7544
         Neg Pred Value : 0.8320
             Prevalence : 0.4451
         Detection Rate : 0.3564
   Detection Prevalence : 0.4725
      Balanced Accuracy : 0.7959

       'Positive' Class : edible
```

(b) SVM.



(c) Variable Importance.

Figure 5: Model Results.

3.2 Docker for Reproducibility in R

3.2.1 Docker

Initially released on March 20, 2013, *Docker* is now a standout amongst the best innovations. It is an open-source platform that enables developers to run applications with a more effortless and efficient process [8]. The applications are run, deployed, updated, and managed in containers, which are standardized and executable components to package all required operating system (OS) libraries, dependencies, and application source code together. Docker also solves the dependencies issues since it allows to use of older versions of a package while keeping the package on the machine up to date.

Before introducing how to reproduce the mushroom analysis result in R with a Dockerfile, two definitions are needed to make clear. First is the *Docker image*, which is the definition of the Operating System. It is needed to install just once. To build the image, users need to use the *Dockerfile*, a configuration file containing a list of instructions. In the next section, I will illustrate the Dockerfile I wrote to reveal how I use Docker for reproducibility in R. The second definition is the *Docker container*, which I have already given the basic introduction above. The containers must be launched whenever users need this instance, and multiple containers of the same images can be run simultaneously. Compared with R, this is the same principle as installing vs. loading a package. For example, users are building an image like they are using *install.package*() in R, and when they run the image to get the containers, it is like they are calling the *library*() [1].

3.2.2 Dockerfile

Figure 6 is the Dockerfile to generate a txt file with the same result as the *Random Forest Classifier* from the first task. I will now illustrate each part to demonstrate their corresponding goals and how to build the image.

First, every Dockerfile starts with a **FROM**. It specifies the base image on which the *Docker image* will be based. The base image is the starting point for building the Docker image and provides the foundation for running the application and its dependencies. For example, I used *rocker/r-ver* as the base image to ensure the reproducibility and portability of the workflows. The rocker package provides several pre-built images with R and additional software installed, such as Shiny Server and RStudio [3]. Therefore, this makes it easier to use R in a Docker environment without manually building images. Next, the **RUN** command is executed to create a directory to receive the analysis. It could also configure the environment, install software, etc. For the next step, I included the commands to let the Dockerfile install the requested R packages for the mushroom classification task, such as *'string'*, *'purrr'*, and so on. Then, since every command is to be run in the container, documents saved in the local machine must be copied into the container. That's when **COPY** command comes in handy. In the last, the R file copied to the container is sourced, and the final txt file is moved to another folder under the directory of *"/home/results"* for later to be exported from the container.

```
FROM rocker/r-ver:4.2.0

RUN mkdir /home/analysis

RUN R -e "install.packages('stringr',version='1.5.0')"
RUN R -e "install.packages('purrr',version='1.0.0')"
RUN R -e "install.packages('caret',version='6.0.92')"
RUN R -e "install.packages('ggplot2',version='3.3.6')"
RUN R -e "install.packages('randomForest',version='4.7.1.1')"

COPY mushroom_classification.R /home/analysis/mushroom_classification.R
COPY secondary_data_no_miss.csv /home/analysis/secondary_data_no_miss.csv

CMD cd /home/analysis \
  && R -e "source('mushroom_classification.R')" \
  && mv /home/analysis/mushroom_classification_accuracy.txt /home/results/
mushroom_classification_accuracy.txt
```

Figure 6: Dockerfile.

To build a Docker image, we need to create a directory for the Dockerfile and navigate to it. Then, we run the first command marked red in Figure 7. Here, **docker build** is the command to build the image, **-t name** specifies the name for the image, and the dot, in the end, specifies the current directory as the build context. Once this command is executed successfully, the image with the name *"mushroom_analysis"* should appear in the Docker Desktop, as shown below. Next, the second command, starting with **docker run** from Figure 7, allows us to run a container from this image. The container will be automatically closed once it is run successfully, so that's why the container content we need must be exported. The **-v** flag helps us to achieve this goal by specifying the path from the host and the path in the container.

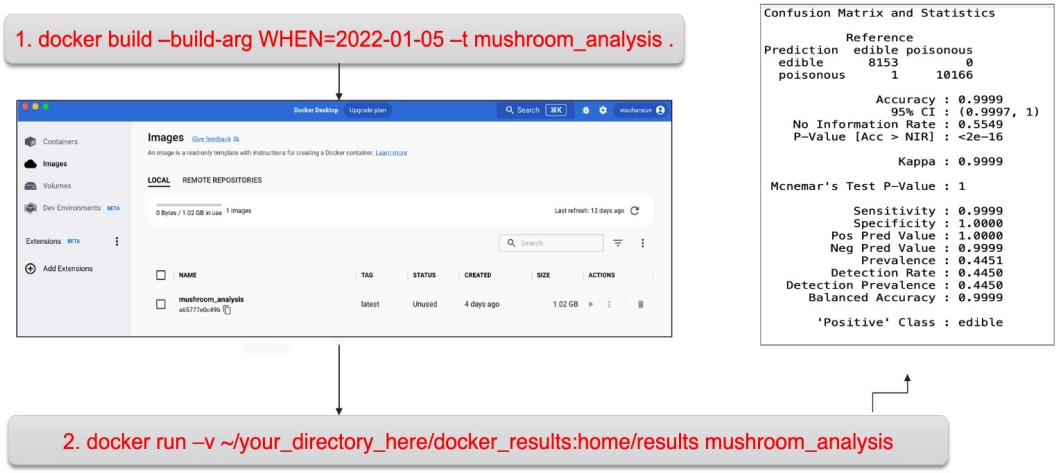


Figure 7: Build and Run the Dockerfile.

3.3 Create the Docker Image with "dockr" Package

For this last project task, I explored how to use the *"dockr"* package to build a lightweight Docker r-base container image without writing the Dockerfile from scratch. Based on my experiment, I indeed could create a Docker image using this package with the function of *"prepare_docker_image"*, with the codes shown in Figure 8a. However, some

other functions that are supposed to provide by this package are not available, such as *"write_lines_to_file."* which is introduced to write lines to the end of the Dockerfile. Figure 8b documented the failure and the related R codes as well. Also, the Dockerfile generated automatically contains too many redundant and useless codes to let users install R packages they do not need. Therefore, using this package is inefficient since it is also no longer supported and actively maintained on CRAN.

```r
library(dockr)

package_dir <- system.file(package = "dockr")

image_dockr <- prepare_docker_image(pkg = package_dir,
                                    dir_image = "/Users/macbook/Desktop/dockr",
                                    dir_install="/Users/macbook/Desktop/dockr")

list.files(image_dockr$paths$dir_image)

list.files(image_dockr$paths$dir_source_packages)

# write lines to the end of the file.
write_lines_to_file(c("# Write lines next."),
                    image_dockr$paths$path_Dockerfile,
                    prepend = FALSE,
                    print_file = FALSE)
```

(a) R Codes to Use "dockr".

```
> # write lines to the end of the file.
> write_lines_to_file(c("# Write lines next."),
+                     image_dockr$paths$path_Dockerfile,
+                     prepend = FALSE,
+                     print_file = FALSE)
Error in write_lines_to_file(c("# Write lines next."), image_dockr$paths$path_Dockerfile,
:
  could not find function "write_lines_to_file"
>
```

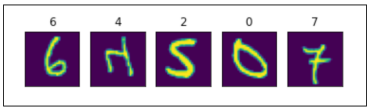
(b) Failure to Run "write_lines_to_file".

Figure 8: The "dockr" Package.

3.4 Project Reproduction

3.4.1 Reproduction Result

The project assigned to me was **"Code Sharing in Python Using pipenv."** The task was to use *pipenv* to create a proper environment to ensure the experiment's reproducibility, which is about the data augmentation of the MINST dataset. Based on my exploration, the project is easy and convenient to run with the help of Google Colab, Jupyter Notebook, and GitHub. Everything is documented with clear step-by-step instructions, and I achieved the same results as the authors as Figure 9 shown below.



(a) MINST Dataset.

```
/usr/local/lib/python3.8/dist-packages/t
  warnings.warn(warn_msg)
Epoch: 1        Training Loss: 1.0679
Epoch: 2        Training Loss: 0.7089
Epoch: 3        Training Loss: 0.5633
Epoch: 4        Training Loss: 0.4837
Epoch: 5        Training Loss: 0.4315
Epoch: 6        Training Loss: 0.3986
Epoch: 7        Training Loss: 0.3711
Epoch: 8        Training Loss: 0.3509
Epoch: 9        Training Loss: 0.3374
Epoch: 10       Training Loss: 0.3266
```

(b) Training Loss.

```
Test Loss: 0.107611

Test Accuracy of     0: 99% (971/980)
Test Accuracy of     1: 99% (1126/1135)
Test Accuracy of     2: 95% (983/1032)
Test Accuracy of     3: 96% (972/1010)
Test Accuracy of     4: 97% (961/982)
Test Accuracy of     5: 94% (839/892)
Test Accuracy of     6: 96% (923/958)
Test Accuracy of     7: 94% (969/1028)
Test Accuracy of     8: 98% (963/974)
Test Accuracy of     9: 94% (951/1009)

Test Accuracy (Overall): 96% (9658/10000)
```

(c) Test Accuracy.

Figure 9: Reproduction Results.

3.4.2 Comparison between Two Projects

Both projects are designed to achieve code sharing for reproducibility, with the ability to manage dependencies and their versions in a single file, integrate with popular tools and ensure improved security where the project is isolated, but each has its benefits

and drawbacks. Compared to Docker, *pipenv* is more easily implemented with a simple and intuitive interface for managing packages and virtual environments [9]. However, Docker can have a steep learning curve for users unfamiliar with Docker images and containers. On the contrary, pipenv is unsuitable for projects requiring multiple GPUs since reproducibility cannot be promised anymore, but Docker containers can be easily deployed and managed, which makes it easier to manage reproducibility.

4 Conclusion

The increasing dataset size and availability of computational resources have brought pros and cons to scientific researchers. On the one hand, larger datasets lead to improved accuracy in machine learning models and improve generalization, meaning that they can make accurate predictions on unseen, new data; on the other hand, processing and analyzing large datasets can be computationally expensive, and it had begun to outgrow older methods for communicating research results. That's why open science has brought attention to statistics and machine learning. It tremendously promotes transparency, reproducibility, and collaboration in the scientific process, leading to a greater understanding of the underlying models, techniques, and algorithms. Open science also helps to reduce the risk of bias by allowing others to inspect and validate the methods and data used in a study.

However, issues of reproducibility are still significant obstacles to open science. Less than 40% of papers provided links to code among the 679 papers presented at NIPs in 2017[10]. Based on my exploration of the reading and the project, not only the code and data should be provided, but also the environment to achieve the highest reproducibility. There are also many other challenges to reproducibility, such as the dependency on specific software or hard configurations. For example, everything went successfully when I ran the mushroom analysis result from Docker on macOS. Still, when my colleagues tried to reproduce my results, the Docker container had problems with his Windows operating system. Also, Docker can have a steep learning period for users who are unfamiliar with containers, which can make it challenging to use for reproducibility. It is also essential to consider the longevity of reproducible examples, such as the links provided in published papers should be stable as long as they can[10]. Last but not least challenge of reproducibility is the consideration of intellectual property. Intellectual property rights may limit access to data, development tools, and other resources needed to reproduce results.

In conclusion, although open science has significantly improved the scientific process's transparency, collaboration, and reproducibility, there are still many corresponding problems researchers have to solve. To overcome these obstacles and promote open science in statistics and machine learning, it is essential to raise awareness of open science's benefits and provide researchers with the tools, resources, and incentives to make their data and methods available openly.

5 Acknowledgement

I would like to express my sincere gratitude to Andreas Bender, Martje Rave, Giacomo de Nicola, Lisa Wimmer, Felix Henninger, who provided invaluable support and guiudance throughout the course. Their insights, expertise, and constructive feedback have been instrumental in shaping the final outcome of my work. I am truly grateful for their unwavering support and dedication to helping me achieve my goals.

References

[1] *An introduction to docker for R users* [2019].
URL: *https://colinfay.me/docker-r-reproducibility/*

[2] Aranay, O. M. and Atrey, P. K. [2022]. Active genetic learning with evidential uncertainty for identifying mushroom toxicity, *2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, IEEE, pp. 395–400.

[3] Boettiger, C. and Eddelbuettel, D. [2017]. An introduction to rocker: Docker containers for r, *arXiv preprint arXiv:1710.03675* .

[4] Briand, L. C., El Emam, K. and Wieczorek, I. [1998]. *A case study in productivity benchmarking: Methods and lessons learned*, Fraunhofer-IESE.

[5] Cumming, G. and Calin-Jageman, R. [2016]. *Introduction to the new statistics: Estimation, open science, and beyond*, Routledge.

[6] Fitzner, K. [2007]. Reliability and validity a quick review, *The Diabetes Educator* **33**(5): 775–780.

[7] Myrtveit, I., Stensrud, E. and Shepperd, M. [2005]. Reliability and validity in comparative studies of software prediction models, *IEEE Transactions on Software Engineering* **31**(5): 380–391.

[8] Rad, B. B., Bhatti, H. J. and Ahmadi, M. [2017]. An introduction to docker and analysis of its performance, *International Journal of Computer Science and Network Security (IJCSNS)* **17**(3): 228.

[9] Samuel Rajiv, C. et al. [n.d.]. A platform to help in generating code for machine learning and data science projects, *Parkavi, A Platform to Help in Generating Code for Machine Learning and Data Science Projects* .

[10] Tatman, R., VanderPlas, J. and Dane, S. [2018]. A practical taxonomy of reproducibility for machine learning research.