Development project, Machine Learning

RAVELOMANANA Nampoina
SABIR Ilias
YE Xianzhi
XU Yao-hua Franck

December 2022

Contents

1 Introduction

Development project in Machine Learning is aimed at developing good programming practices, using standard development tools and getting used to collaborative work. The objective of the project is to apply Machine Learning binary classification models onto two different data-sets:

- Banknote Authentication Dataset
- Chronic Kidney Disease

2 Data-sets

2.1 Banknote Authentication Dataset

The Banknote Authentication Dataset is about distinguishing genuine and forged banknote. Data were extracted from images that were taken from banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. A Wavelet Transform tool was used to extract features from these images. The data-sets contains 1372 instances of 4 attributes plus a nominal attribute that gives the class of each instances :

	Meaning	**Type**
1	Variance of Wavelet Transformed image	Numerical
2	Skewness of Wavelet Transformed image	Numerical
3	Curtosis of Wavelet Transformed image	Numerical
4	Entropy of image	Numerical
5	Class	Nominal

The data-set does not contain missing or tainted values and can be used out of the box.

2.2 Chronic Kidney Disease

The Chronic Kidney Disease contains data to predict either a patient does or does not suffer from chronic kidney disease based on 25 physiological features :

	Name	Meaning	Type	Unit
1	age	Age in years	Numerical	Year
2	bp	Blood pressure	Numerical	mm/Hg
3	sg	Specific Gravity	Nominal	(1.005,1.010,1.015,1.020,1.025)
4	al	Albumin	Nominal	(0,1,2,3,4,5)
5	su	Sugar	Nominal	(0,1,2,3,4,5)
6	rbc	Red Blood Cells	Nominal	(normal,abnormal)
7	pc	Pus Cell	Nominal	(normal,abnormal)
8	pcc	Pus Cell clumps	Nominal	(present,notpresent)
9	ba	Bacteria	Nominal	(present,notpresent)
10	bgr	Blood Glucose Random	Numerical	mgs/dl
11	bu	Blood Urea	Numerical	mgs/dl
12	sc	Serum Creatinine	Numerical	mgs/dl
13	sod	Sodium	Numerical	mEq/L
14	pot	Potassium	Numerical	mEq/L
15	hemo	Hemoglobin	Numerical	gms
16	pcv	Packed Cell Volume	Numerical	
17	wc	White Blood Cell Count	Numerical	cells/cumm
18	rc	Red Blood Cell Count	Numerical	millions/cmm
19	htn	Hypertension	Nominal	(yes,no)
20	dm	Diabetes Mellitus	Nominal	(yes,no)
21	cad	Coronary Artery Disease	Nominal	(yes,no)
22	appet	Appetite	Nominal	(good,poor)
23	pe	Pedal Edema	Nominal	(yes,no)
24	ane	Anemia	Nominal	(yes,no)
25	class	Class	Nominal	(ckd,notckd)

There are 400 instances in the data-set, 250 CKD and 150 not CKD which means that the data-set is biased.

Moreover, the data-set contains a lot of missing values and tainted attribute's values that need to be corrected :

	Original Value	Corrected Value
1	\t43	43
2	\t6200	6200
3	\t8400	8400
4	\tno	no
5	\tyes	yes
6	rbc	rbc
7	ckd\t	ckd
8	\t?	

3 Machine Learning Workflow

The goal of this project is to apply the following workflow for both data-sets :

1. Import the dataset
2. Clean the data, perform pre-processing
   - Replace missing values by average or median values
   - Center and normalize the data
3. Split the dataset
   - Split between training set and test set
   - Split the training set for cross-validation
4. Train the model (including feature selection)
5. Validate the model

4 Machine Learning Models

In this section, we are going to describe the models we use. The models used are directly imported from the library Sklearn except the MLP which is implemented using Pytorch. This choice of using two different libraries adds a new constraint as these libraries are really different and it will be discussed further in the next section.

4.1 Support-Vector Machine

Support vector machine (SVM) are supervised learning models and associated learning algorithms for analyzing data in classification and regression analysis. Given a set of training instances, each labeled as belonging to one or the other of two classes, the SVM training algorithm computes an hyperplan that maximize a hard or soft margin (allow a small error) depending on if the data is linearly divisible.

When the training data is linearly inseparable, a nonlinear SVM can be learned by using the kernel trick and soft interval maximization.

In the project, SVM models are imported from **Sklearn** with default parameters so no kernels have been used.

4.2 Multi-Layer Perceptron

Multi-layer perceptron (MLP) is a forward-structured artificial neural network that contains an input layer, an output layer, and multiple hidden layers.

The basic unit of computation in a neural network is the neuron, generally called a "node" or "unit". Each node can receive inputs from other nodes, or from external sources, and then compute the output. Each input has its own weight and bias, with the weight used to regulate the magnitude of the effect of that input on the output and the bias providing a trainable constant value for each node.

Before the result is output by the neuron, it is processed by an activation function. The activation function generally uses a nonlinear function, whose role is to introduce non-linearity into the output of the neuron. The most popular activation functions are the **Sigmoid** function, the **tanh** function and the **ReLU** function.

The MLP model named TorchMLP is implemented with Pytorch. It is a fully connected Neural Network with 3 hidden layers with ReLU activation function on hidden layers and Sigmoid activation function for the output layer. We also added some **dropout** to avoid overfitting. More details can be found in the **README** file.

4.3 Logistic regression

Logistic regression solves dichotomous problems and is used to express the probability of something happening. The principle of logistic regression is to map the result of a regression $(-\infty, \infty)$ (usually a linear regression) to $(0, 1)$ using a logistic function by minimizing the Binary Cross Entropy. The logic function here is the sigmoid function.

The Logistic Regression used here is imported from Sklearn with default parameters. Its implement a simple classifier with regulation strength of 1.0 and linear regression as based transformation.

4.4 Decision Tree

The decision tree algorithm is a supervised learning algorithm based on if-then-else rules, it uses a tree-like structure and uses layers of inference to achieve the final classification. For prediction, a judgment is made at the node of the tree with a certain attribute value, and based on the judgment result, it decides which branch node to enter until it reaches the leaf node to get the classification result.

The learning process of a decision tree consists of 3 steps:

1. Feature selection : determines which features are used to make judgments. The commonly used criterion in feature selection is: information gain.

2. Decision tree generation : once the features are selected, the nodes are triggered from the root node, and the information gain of all features is calculated for the nodes.

3. Decision tree pruning : removing some branches to reduce the risk of overfitting

Decision tree are easy to understand and interpret, can be analyzed visually, and rules can be easily extracted. The Decision Tree is imported from Sklearn. Different tests have been made to tune the depth of the tree : a depth of 3 is enough for both datasets.

4.5 Random Forest

Random Forest is a supervised algorithm, which is an integrated learning method consisting of many decision trees, with no correlation between the different decision trees.

When we perform the classification task, new input samples enter and let each decision tree in the forest judge and classify them separately. Each decision tree will get a classification result of its own, and whichever classification result of the decision tree has the most classifications, then the random forest will take that result as the final result.

There are 4 steps to constructing a random forest :

1. A sample with a sample size of N is randomly selected N times, one at a time, to obtain N samples, which are used to train a decision tree.

2. When each sample has M attributes, m attributes are randomly selected from these M attributes when each node of the decision tree needs to be split, satisfying the condition $m << M$. Then some strategy (e.g. information gain) is used to select 1 attribute from these m attributes as the splitting attribute for that node.

3. Each node in the decision tree is split according to step 2 (if the next attribute selected for the node is the same attribute that was used in the splitting of its parent node, then the node has already reached the leaf node and does not need to be split), until it is no longer possible to split. Note that there is no pruning during the whole decision tree formation process.

4. Follow steps 1 to 3 to build a large number of decision trees, which will form a random forest.

Random Forest is also imported from Sklearn with default configuration which means it is tuning 100 trees. Computing less tree should not reduce the performances on both datasets but it may reduce the overall training duration.

4.6 K-Nearest Neighbor

The KNN (K-Nearest Neighbor) algorithm is a basic classification and regression algorithm which belongs to the category of classification methods in supervised learning. The general idea is formulated as follows.

1. Given a training set M and a test object n, where the object is a vector consisting of an attribute value and an unknown category label.

2. Calculate the distance or similarity between object m and each object in the training set, and determine the list of nearest neighbors.

3. The category with the largest number of occupants in the nearest neighbor list is awarded to the test object z.

4. In general, we select only the top K most similar data in the training sample, which is where "k" comes from in the k-nearest neighbor algorithm.

Here we took $k = 5$ and used the KNN model from Sklearn.

4.7 Naive Bayes

Naive Bayes is a classification algorithm based on Bayes' theorem. According to the Bayesian equation, we have:

$$P(\text{class}|\text{features}) = \frac{P(\text{features}|\text{class})P(\text{class})}{P(\text{features})}$$

The main advantages of the Naive Bayes are:

- This model originates from classical mathematical theory and has stable classification efficiency.
- It performs well on small-scale data, can handle multiple classification tasks, and is suitable for incremental training, especially when the amount of data exceeds the memory, we can go to incremental training in batches.
- Less sensitive to missing data and simpler algorithm, often used for text classification.

Naive Bayes is also imported from Sklearn with default settings.

5 Code structure

The project is split in two main parts : the file **main.py** and the folder **/Utils** :

- **main.py** is the main script and contains the workflow definition. To run it, we use the following command in a shell :

  ```
  python3 main.py dataset/path model_name
  ```

  The script will parse the arguments, import the given data-set, create the associated model and run the workflow by using the functions defined in the **/Utils** folder

- The **/Utils** folder is composed of four files :
  - **dataProcessing.py** contains all the functions related to dataset : get_dataset_name(), import_data(), clean_data().
  - **models.py** contains all the classes and function related to model. As we are using two really different libraries (**sklearn** has models that can be used out of the box while **PyTorch** gives tools to create and train Neural Network model), we decided to use a design pattern called **Adapter** so that we can manipulate PyTorch models as we manipulate Sklearn models. It results on simpler implementation of the workflow in the **main.py** file and a better code in terms of readability and re-usability. If we want to apply this workflow with other model from other libraries, we just need to define a new **ModelAdapter** for this model which results on minimal and local changes in the code-base.
  - **params.py** contains some predefined attributes for sklearn models and the different corrections needed for each data-sets. For instance, corrections and data type for Kidney dataset are stored here, the **main.py** script just need to fetch these static attributes whenever we want to perform on Kidney dataset and passes them to the **clean_data()** function. Adding new data-set should be easier that way, as we just need to register the name of the data-set into this file alongside with the different corrections that we want to make and the list of the numerical data.
  - **render.py** contains only one function used to render workflow's results.

6 Results

To measure models performances, we will use these two metrics:

- precision: is the ratio of true positives to the total of the true positives and false positives
- recall: is the ratio of true positives to the total of the true positives and false negatives

We will compare the different models, while applying PCA to the data. We will also take a look at the computational time.

6.1 Banknote dataset

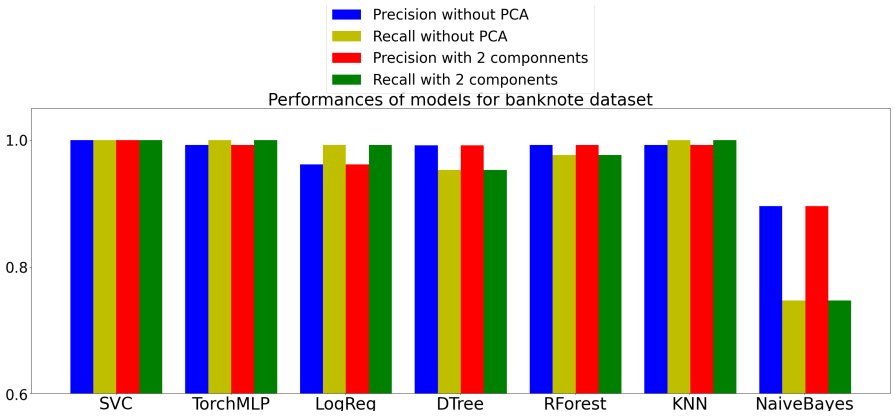

Figure 1: Precision and Recall

One important thing to notice is that we have the worst performance when we apply PCA and keep only two components. Applying PCA is very useful when we have high dimensional data, in order to reduce dimension, prevent overfitting, or denoise data. Here, we can see that applying PCA on few dimensional data is not really helping; it may increase or (more likely) decrease the model performance.

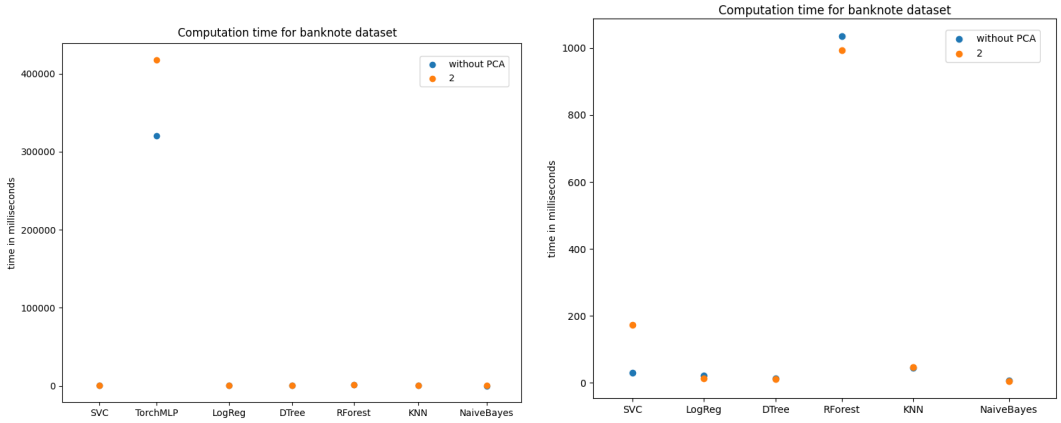

Figure 2: Computational Time including and excluding MLP

We can notice that the neural network takes a lot more time than the other models. But the training time of MLP depends on the number of parameters of the network, taking a simpler neural network will reduce the computation time.

We can notice that the Random Forest model is the one who takes the most time. We used the default scikit-learn Random Forest model which takes as default parameter 100 trees; so it is pretty understandable that it took more time than the other models. Here, we can notice that all models perform pretty well but TorchMLP and Random Forest are taking more time to train. Decision tree or Logistic regression are probably the best models. They both give good result with short computational time.

6.2 Kidney dataset

For the Chronic Kidney Disease Dataset, all models are performing very well with and without PCA. Only KNN performance decreases with PCA but we still get **preicision, recall** > 0.95 From computational time point of view, TorchMLP and Random Forest are the slower. Simple models like

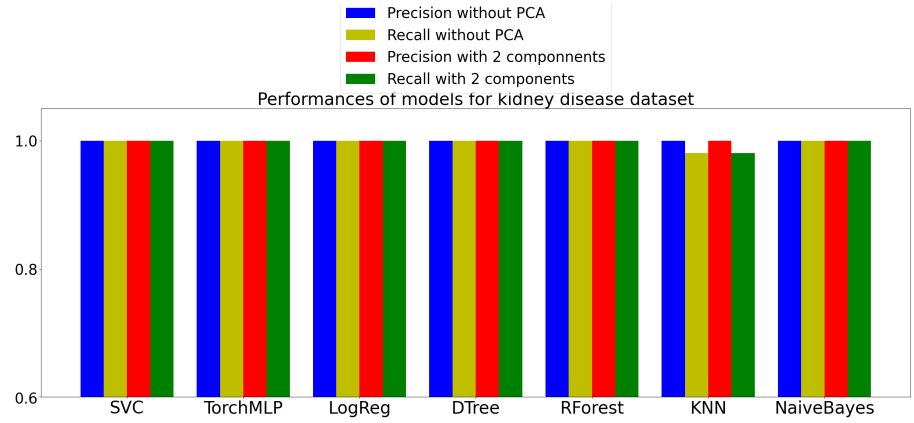

Figure 3: Precision and Recall

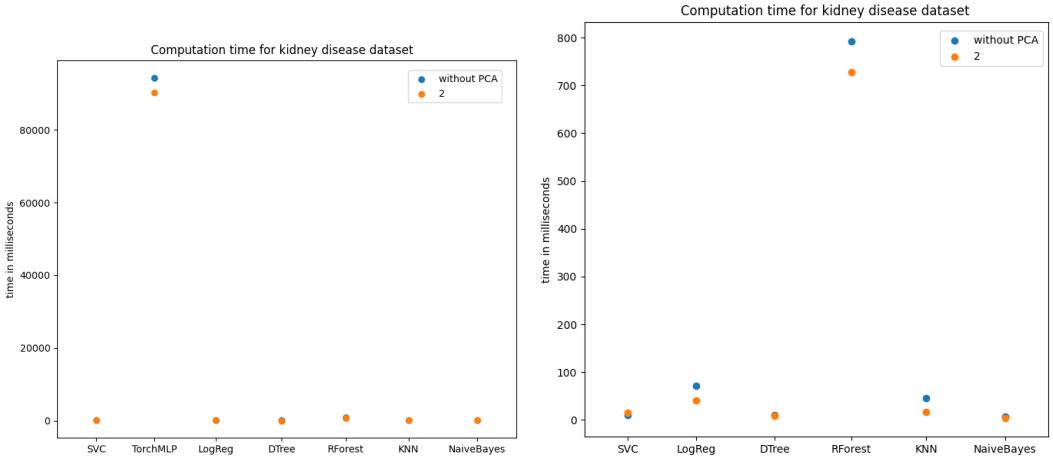

Figure 4: Computational Time including and excluding MLP

Decision Tree and Naives Bayes are performing well with really short training duration and might be the best suited models here in term of performance-computational cost ratio.

7 Good programming practices

7.1 Work in a team

Team management

Working in a team requires a lot of effort, especially when many developers are working simultaneously on the same project. For this, it is necessary to Agile work. This approach allows to quickly release features and to adapt to changes in a flexible and proactive way.

The agile method goal is to divide the software development into several stages and to fix short-term objectives. This approach is different from the traditional project management methods where we usually wait until we have completed all the steps of a project before delivering the final product to the client. It is necessary to plan daily or weekly meetings to synchronize and split the tasks between the developers.

In our case, we planned meetings at the end of each cycle (one cycle duration is one week) to make sure that the project was going according to plan. It is also a perfect moment to split the tasks for the next cycle. We didn't use any agile management software, but we were able to share documents on google drive and discuss on Messenger.

Code management

In addition to the project management, it is also necessary to manage the source code repository and the management of the different versions of the code. To manage that, the Git version control system offers many features for this job such as:

- Easy to share code and data
- Synchronize changes
- Manage programming conflicts
- Everything is backed up by default
- Can add automated tests/ automated deployment (Continuous integration/ Continuous deployment)
- Project management
- Documentation
- Version management

To work efficiently with git, it is necessary to follow the following practices:

- Make small changes
- Develop using branches (PS: Only tested code in the main branch)
- Add a description message for each commit
- Code review before each merge

7.2 Provide a reproducible code

Any process to develop code should be based on the foundation of reproducibility. Results must be able to achieve repeatability by other developers or users for verification. The purpose of providing reproducible code is not necessarily to save time but to share knowledge. For that, it is necessary to respect the following rules to provide reproducible code:

- **Use standard formats** that can be readable by a wide range of users.
- **Documentation of the code** (the main purpose of the code, how to run it and the dependencies that are required)
- **Create a portable and platform independent code:** use standard libraries, specify the absolute path, check for system-specific dependencies, and use a specific file for configurations
- **Never hard-code values**

7.3 Follow good programming practices

To deliver a high-quality code, it is necessary to follow a set of programming practices. For example, in the case of programming in Python, it is best to respect all the PEP 8/PEP 20 practices which allow to standardize the code and to apply the good programming practices. Among these rules, we find in particular:

- Naming rules: use meaningful names for variables and respect the naming conventions (Snake Case)
- Importing modules: import first the internal modules (listed in alphabetical order), i.e., the basic Python modules, then the external modules (those you have installed in addition).
- Line length: for readability reasons, a line of code should not exceed 79 characters.

- Comments: they must give clear explanations on the usefulness of the code, add context to the code and must be updated in case of code modifications. Moreover, it is necessary to comment its code in English.
- Share your environment
- Encoding
- Indentation