COMP 309/AIML421 — *Machine Learning Tools and Techniques*

Assignment 4: Performance Metrics, and Optimisation

15% of Final Mark — Due: 11:59pm Friday 27th September (week 10) 2024

- Parts 1 and 2 of the assignment involve using metrics to compare different algorithms on the same data. In an real-world scenario, it would be necessary to optimise the settings of each of these algorithms, by testing multiple options for the hyperparameters and choosing the best. Obviously it can be a substantial amount of work to do this aspect well, since each method has its own settings. However, in this assignment our particular focus is on the metrics themselves.

  So: to keep the workload down, you should *use the default hyperparameter settings for learning algorithms* wherever possible: you won't be penalised for adopting these, even though in a more realistic scenario you would surely be comparing not just algorithms, but settings for those algorithms as well.

- The assignment has three parts, and these are worth 30/30/40 percent, respectively. The third part depends on the Week 9 lectures/tutorial.

1 Objectives

The goals of this assignment are to use a popular machine learning tool, i.e. scikit-learn, to investigate two important factors for the success of machine learning applications, which are performance metrics and optimisation. The specific objectives of this assignment are:

- Be able to use different classification/regression methods implemented in scikit-learn, such as kNN, support vector machines, decision tree, random forest, AdaBoost, gradient boosting, linear discriminant analysis, logistic regression, linear regression, k-neighbors regression, Ridge regression, decision tree regression, random forest regression, gradient Boosting regression.
- Compare the performance of different classification/regression methods using a number of popular performance metrics, e.g., accuracy, precision, recall, confusion matrix, AUC under ROC, ... and analyse the results.
- Compare and analyse advantages and disadvantages based on the results of different performance metrics for a given regression task.
- Gain some familiarity with a modern machine learning framework applicable to deep learning (`PyTorch`).

The topics for Parts 1 and 2 were covered in the weeks up to 8, while Part 3 material is in week 9. Research into online resources is encouraged. Complete the following questions. For each part, make sure you finish reading all the questions before you start working on it, and your report for the whole assignment should *not exceed **8** pages* with font size no smaller than 10.

2 Questions

Part 1: Performance Metrics in Regression [30 marks]

This part focuses on performance metrics in regression. The task is to use different regression methods, understand different performance metrics and choose the most appropriate performance metric.

The given *Diamonds* data set, diamonds.csv, is to predict the price of round cut diamonds. This is a regression task with 10 features (the first 10 columns of diamonds.csv) as the input variables and the feature price (the last column of diamonds.csv) as the output variable. The task here is to learn a regression model to discover the relationship between the output variable and the 10 features/input variables. As we discussed in the lectures/tutorials, to use scikit-learn for regression, you may need the following seven steps:

- Step 1. Load Data
- Step 2. Initial Data Analysis
- Step 3. Preprocess Data
- Step 4. Exploratory Data Analysis
- Step 5. Build classification (or regression) models using the training data
- Step 6. Evaluate models by using cross validation (Optional)
- Step 7. Assess model on the test data.

Requirements

Use "309" as the random seed to split the data into a training set and a test set, with 70% as the training data and 30% as the test data.

You should use the following 10 regression algorithms implemented in scikit-learn to perform regression. These 10 algorithms are very popular regression methods: (1) linear regression, (2) k-neighbors regression, (3) Ridge regression, (4) decision tree regression, (5) random forest regression, (6) gradient Boosting regression, (7) SGD regression, (8) support vector regression (SVR), (9) linear SVR, and (10) multi-layer perceptron regression. You are encouraged to read the documentation (and provided references if you would like to know more details) about these methods from scikit-learn, e.g. linear regression is implemented in `sklearn.linear_model.LinearRegression`.

As noted earlier, we are *not* asking you to tune the parameters for these regression methods to make them achieve better performance, because that would be time consuming and is not the real goal of this assignment. The same goes for the classification methods in the next section.

Submit the code of your program (e.g. it could be a plain python file or a jupyter notebook). Call it something obvious (like `regression.py` or `regression.ipynb`). If the program has anything non-trivial about how it should be run (e.g. it takes arguments), then add a very brief `readme.txt` file that describes how to run the code, for a user on the ECS machines.
Part 1 of the overall report should answer the following questions:

- Based on exploratory data analysis, discuss what preprocessing that you need to do before regression, and provide evidence and justifications.
- Report the results (keep 2 decimals) of all the 10 regression algorithms on the *test* data in terms of mean squared error (MSE), root mean squared error (RMSE), relative squared error (RSE), mean absolute error (MAE), and execution time. You should report them in a table.
- Compare the performance of different regression algorithms in terms of MSE, RMSE, RSE, and MAE, then analyse and discuss their differences and provide conclusions.

Part 2: Performance Metrics in Classification [30 marks]

The given *Adult* dataset is a popular classification data set from the UCI machine learning repository, and the task is to determine whether a person earns a salary of over $50K a year. Separate training and test sets are provided, as adult.train and adult.test, respectively. You are recommended to follow the steps that we discussed in the lectures/tutorials and listed in Part 1.

Requirements

Use 10 classification algorithms implemented in scikit-learn to perform classification. These 10 algorithms are very popular classification methods from different paradigms of machine learning: (1) kNN, (2) naive Bayes, (3) SVM, (4) decision tree, (5) random forest, (6) AdaBoost, (7) gradient Boosting, (8) linear discriminant analysis, (9) multi-layer perceptron, and (10) logistic regression. You are encouraged to read the documentation (and provided references if you would like to know more details) about these methods from scikit-learn, e.g. kNN is implemented in *sklearn.neighbors.KNeighborsClassifier*. We assume that class $> 50K$ is the positive class.

As for regression, submit your code and call it something obvious (like `classification.py`).
The report should answer the following questions:

- Based on exploratory data analysis, discuss what preprocessing that you need to do before classification, and provide evidence and justifications.
- Report the results (keep 2 decimals) of all the 10 classification algorithms on the given *test* data in terms of classification accuracy, precision, recall, F1-score, and AUC. You should report them in a table.
- Find the two best algorithms according to each of the performance metrics. Are they the same? Explain why.

Part 3: Optimisation [40 marks]

For this part, you should code in python with TORCH, using a `Jupyter Notebook` or `Colab`. Submit a file named `MyAutoencoder.ipynb`[1]. There's nothing to add to the main report for this part – it's all in the notebook.
This part of the assignment is to get you introduced to using gradient descent within torch.
You're building a simple linear AUTOENCODER (see figure) and training it using `torch.optim` optimisers. An autoencoder is an unsupervised learner that carries out *dimensionality reduction* by mapping the data into a lower dimensional space and back 'out' again to form a 'reconstruction'. The loss is some measure of the difference between the original data items and their reconstructions.
See the schematic in Figure 2. Notation for that figure:

- x_0 and x_1 are the inputs. Their reconstructions are denoted x_i^{recon}.
- w_0^{enc}, w_1^{enc} and b^{enc} are the encoder's weights, and bias.
- z is the activation of the sole "hidden unit".
- The weights and biases on the decoder side are $w_i^{\text{dec}}, b_i^{\text{dec}}$
- All the "activation functions" are simply linear, and so:

$$\begin{aligned} z &= w_0^{\text{enc}} x_0 + w_1^{\text{enc}} x_1 + b^{\text{enc}} \\ x_0^{\text{recon}} &= w_0^{\text{dec}} z + b_0^{\text{dec}} \\ x_1^{\text{recon}} &= w_1^{\text{dec}} z + b_1^{\text{dec}} \end{aligned}$$

You will be using input "data" that you generate yourself as follows:

```
import matplotlib.pyplot as plt
import numpy as np
import torch

# make up some data for x
D = 2
```

[1]Note you can download from colab in that format

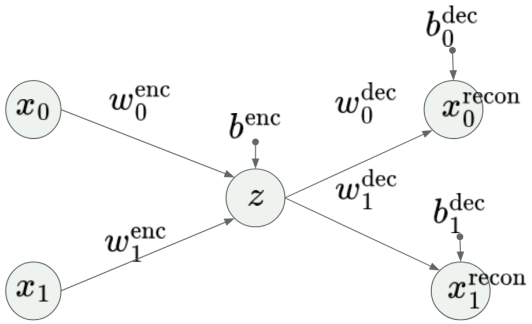

Figure 1: An autoencoder that takes a 2-d vector, and forces it through a 1-d bottleneck with linear functions.

```
x= torch.rand(100,D)
x[:,0] = x[:,0] + x[:,1]
x[:,1] = 0.5*x[:,0] + x[:,1]

plt.scatter(x[:,0],x[:,1])
plt.axis('equal')
```

You will be defining three functions:

- **an encoder**, which takes the 2-d inputs and produces a 1-dimensional z via a weighted sum plus a bias. The encoder weights will be a 1-by-2 tensor and the encoder bias will be a 1-by-1 tensor.
- **a decoder**, which just does the reverse: it takes that 1-dim z and produces a 2-dim output via a weighting of z plus a bias (see the equation in the figure). (*Note: decoding is not to be confused with the torch's backward() operation that propagates gradients! This is a still "forwards" transformation*).
- **a loss function**, which should be the mean squared error (MSE) between x and its reconstruction, averaged over the data set.

By using autograd (`.backward()`) and optimising the MSE loss, you will learn the values of the weights and biases in the encoder, and the decoder (i.e. 3 parameters in the encoder and 4 in the decoder). For each run of learning, start from random weights and biases, such as:

```
wEncoder = torch.randn(D,1, requires_grad=True)
wDecoder = torch.randn(1,D, requires_grad=True)
bEncoder = torch.randn(1, requires_grad=True)
bDecoder = torch.randn(1,D, requires_grad=True)
```

Requirements

1. Use SGD, learning rate 0.01, no momentum, and 1000 steps.
   - Plot the loss versus epochs (steps).
   - All on the same axes of one scatterplot, show (i) the original data in some light colour like 'cyan', (ii) the reconstructed data in a different colour, and (iii) a line from the origin to vector formed by the two learned encoder weights. Perhaps something like...

```
plt.scatter(x[:,0],x[:,1],color='cyan');
plt.scatter(x_reconstruction.detach()[:,0],x_reconstruction.detach()[:,1]);
plt.plot([0,wEncoder[0,0]], [0,wEncoder[1,0]],'-r');
plt.axis('equal')
```

- print out the ratio of the weight in the encoder versus the weight in the decoder, for each of the two dimensions.

2. Now add momentum (say 0.9) and do the same.
3. Switch to RMSprop (with momentum 0.9) and do the same.
4. In a `text` cell of the notebook, write a couple of concise paragraphs, interpreting what you found.

NOTE: additionally, for Students doing AIML421 only: (extra 20 marks)

1. In a `text` cell: explain the relationship between the above autoencoder and PCA.
2. In a `text` cell: in the notebook with the Week 9 tutorial, you may have noticed how the `RMSprop` and `Adam` optimisers seem to prefer travelling along diagonal lines. Why does this happen?

3 Relevant Data Files and Program Files

This assignment and the relevant data and program files are available from the course homepage at `http://ecs.victoria.ac.nz/Courses/COMP309_2024T2/Assignments`

4 Assessment

We will endeavour to mark your work and return it to you as soon as possible, hopefully in 2 weeks. The tutor(s) will run a number of helpdesks to provide assistance to answer any questions regarding what is required.

5 Submission Guidelines

5.1 Submission Requirements

1. For Parts 1 and 2:
   - Two programs (regression, classification), with obvious names. Ensure that any code you provide is easily run on ECS machines. If there is any doubt about how code should be compiled or run, provide a `readme file` that specifies exactly how to do so. If your programs cannot run properly, you should provide a `buglist` file.
   - One PDF document. This document should mark each part clearly.
2. for Part 3:
   - a jupyter notebook.

5.2 Submission Method

Submit through the web submission system from the COMP309 course web site **by the due time**.

KEEP a backup and receipt of submission.

Problems with personal PCs, internet connections and lost files, which although eliciting sympathies, will not result in extensions for missed deadlines. There will be ***no*** extension for minor/routine situations — this is what your late days are for.

5.3 Late Penalties

The assignment must be submitted on time unless you have made a prior arrangement with the course coordinator or have a valid medical excuse (for minor illnesses it is sufficient to discuss this with the course co-ordinator). The penalty for assignments that are handed in late without prior arrangement is one grade reduction per day. Assignments that are more than one week late will not be marked.

5.4 Plagiarism

Plagiarism in programming (copying someone else's code) is just as serious as written plagiarism, and is treated accordingly. Make sure you explicitly write down where you got code from (and how much of it) if you use any other resources asides from the course material. Using excessive amounts of others' code may result in the loss of marks, but plagiarism could result in zero marks!