Mandatory Exam Assignment 2

Machine Learning F2023, Spring 2023

Table of Contents

[Prerequisite 2](#_Toc135590653)

[Exercise 1 (25 points) 3](#_Toc135590654)

[Subtask 1 3](#_Toc135590655)

[Subtask 2 3](#_Toc135590656)

[Subtask 3 3](#_Toc135590657)

[Subtask 4 3](#_Toc135590658)

[Subtask 5 3](#_Toc135590659)

[Subtask 6 3](#_Toc135590660)

[Subtask 7 3](#_Toc135590661)

[Exercise 2 (25 points) 4](#_Toc135590662)

[Subtask 1 4](#_Toc135590663)

[Subtask 2 4](#_Toc135590664)

[Subtask 3 4](#_Toc135590665)

[Exercise 3A (25 points) 5](#_Toc135590666)

[Subtask 1 5](#_Toc135590667)

[Subtask 2 5](#_Toc135590668)

[Subtask 3 5](#_Toc135590669)

[Subtask 4 5](#_Toc135590670)

[Exercise 3B (25 points) 6](#_Toc135590671)

# Prerequisite

When running the scripts included in the zip file, under each exercise folder, I used **python version 3.10**

**IMPORTANT:** You must install the pip packages yourself. This is done to DRASTICALLY reduce the zip file size from about 500MB to 1MB.

Follow these instructions to install the packages:

1. creating a new project in PyCharm (preferably)

2. Open a terminal at the root of the project

3. activate your virtual environment by following this guide: <https://docs.python.org/3/library/venv.html>

4. Type ‘pip install scikit-learn matplotlib pandas numpy typing\_extensions ssl nltk tensorflow tensorflow\_probability’ (if you are using pip3 then use ‘pip3 install’ instead)

Now you should be ready to run the scripts.

# Exercise 1 (25 points)

## Subtask 1

By changing the criterion to “gini” and using a max depth of 9, I was able to consistently get an accuracy score on the training set of 1.0 or 100%, however this negatively effects in the accuracy score on the test set. This is likely due to overfitting, however since the task specifically asks to improve the accuracy on the training set, this is the best solution I have found.

## Subtask 2

This task is purely code related. Check the python script *‘code/exercise1.py’*

## Subtask 3

I found that using a max depth of 3 for both criterion “gini” and “entropy” resulted in the best accuracy score for the test set. These scores were consistently 0.964 and 0.947 respectively.

## Subtask 4

While trying to improve the accuracy of the Support Vector Machine Classifier, I found that using the “rbf” kernel with a regularization parameter (C) set to “45,000”, I was able to get an accuracy score of train=0.980, test=0.956. These are the best results I was able to achieve.

## Subtask 5

Trying to improve the linear regression classifier was a bit more difficult. Using the solver “lbfgs” I was able to find the best possible regularization term to be “225”. With this I was able to get an accuracy of train=0.980, test=0.982, which is very good and considerably better than the SVC on the test set.

However, trying different solvers didn’t give much better results. I found that using the solver “newton-cg” I was only able to get the same accuracy. It appears there is a limit to how good the classifiers can get on this dataset. It closely resembles hitting a local maximum.

## Subtask 6

For this subtask I had to use a neural net classifier. I decided to use a Multi-Layer Perceptron Classifier (MLPC). This classifier uses a network of nodes that are interconnected. These nodes make predictions on the dataset and send it to the next layer of nodes. Using this is incredibly powerful as you can specify exactly how many layers and nodes per layer you want.  
  
When trying to find the most optimal MLPC, I used a while loop that randomly generated nodes and layers to hopefully find the best solution. Using the node and layer arrangement of [44, 104, 89, 202, 60], I was able get an incredibly good accuracy score on the training set, “0.997”, but that didn’t give a good score on the test set, “0.956”. This is not desirable as it likely means that you are overfitting to your training set as the score is considerably worse on the test set. And this is incredibly slow to run.

I was however able to find a nice middle group with [215, 210, 164] giving a score of train=0.995 and test=0.929. As this is a neural net classifier, each run will have a different result, so in your testing you might get better or worse scores.

## Subtask 7

In my testing using a neural net classifier like an MLPC can yield very good accuracy scores, but this is however a time consuming and slightly randomized solution, as the dataset will likely never go through the node layers in the exact same way. However, using a linear regression classifier can also yield very good results, as high as 98% accuracy for both train and test sets. Using a linear regression can be much faster than using an MLPC, but this depends on the number of nodes and layers in the MLPC.

# Exercise 2 (25 points)

## Subtask 1

When trying to improve the accuracy of the Random Forest Classifier (RFC), I decided to change the number of estimators, jobs, and the random state. The number of estimators is the number of decision trees in the RFC and increasing this number results in a generally better accuracy, but also dramatically increases time efficiency. I found that having 250 trees in the RFC is a good balance between speed and accuracy. Furthermore, the number of jobs is the number of parallel processes that can run and setting it to “-1” means that it will use all the processors possible. This dramatically improves speed. I also decreased the random state from 4 to 2 to get a more streamline result after each iteration. This all resulted in a score that hovers around 0.77 or 77% accuracy but can sometimes dip down to 0.75.

## Subtask 2

I decided to use the MLPC from sklearn python library. While trying out different arrangements of layers and nodes, I found that for one, the MLPC really struggles to get through the shear amount of data that is in the twitter dataset. Simply adding another layer with 2 nodes, makes the run take exponentially longer time. However, I found that using less nodes means better accuracy on the test set. This has its limits, but with the arrangement of [12, 8] I was able to get scores between 0.75 and 0.79. This is using the “lbgfs” solver.

## Subtask 3

I wanted to have two texts, one negative and one positive. However, I wanted to try to make one of them harder to guess for machine learning algorithm. I ended up making the positive text about Marilyn Monroe, but I spoke about her in a questioning manner, but throwing in some positive words. Hopefully the ML classifiers can classify them correctly.

In my testing, both the RFC and MLPC were able to classify the negative text correctly every run. However, they are not as good at classifying the slightly harder positive text. The RFC incorrectly classifies the positive text every time, but the MLPC sometimes classifies it correctly. This happened about 10% of the time.

I therefore decided to make a second positive text, but this time I made it way more positive. The classifiers should have no issue classifying this text correctly. And that is the case for the RFC, getting it correct every time. However, the MLPC only got it correct about 75% of the time. This shows the strengths and weaknesses of both classifiers, namely that the RFC is much more deterministic than the MLPC, while the MLPC has an overall more random output, it is sometimes better than the RFC.

# Exercise 3A (25 points)

## Subtask 1

See from line 33 in file *‘code/exercise3/a.py’*

## Subtask 2

I assume that the robot can see in all directions, including diagonal spaces. I however decided to keep the original assumption that the robot can only see and move one space at a time.

See from line 55 in file ‘*code/exercise3/a.py’*

I made the robot prioritize diagonal spaces as that covers more surface area overall.

## Subtask 3

I am going to be using an identically sized “robot\_map” as the “real\_map”. This means that I can directly set each “space” in the 2-dimensional array to exactly what the robot detects it as. What I will end up with is a 2-dimensional array of visited spaces and spaces that have obstacles. This perfectly suits using a DFS algorithm as I only want to find the shortest route, and I have all the data/information needed to recursively search through the “robot\_map”.

## Subtask 4

See from line 79 in file *‘code/exercise3/a.py’*

# Exercise 3B (25 points)

See code from Input 11 in file *‘code/exercise3/b.ipynb’*