

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

EXERCISE 4

Full names: Andreas Kalavas, Panagiota Chita

Emails: andreas.kalavas.stud@pw.edu.pl , panagiota.chita.stud@pw.edu.pl

Introduction:

In this report we present our work on packet classification, using Support Vector Machines and Decision Trees. Our programs should predict whether a packet is a DDoS attack packet, or a normal one. Firstly, we discuss the two methods, and then we compare them and conclude to which is the best one for this classification problem.

1. Support Vector Machines (SVMs):

A SVM is a training algorithm that builds a model that assigns new examples to the proper category, making it a non-probabilistic linear classifier. Below we present our program:

```
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import tree
import time
# from sklearn.metrics import confusion_matrix

with open('kddcup.data_10_percent_corrected', 'rb') as f:
    inputs = f.readlines()

packets = []
for item in inputs:
    packets.append(str(item) [2:(len(item)+1)])

hX = []
hy = []
for packet in packets:
    strlistpac = packet.split(",")
    take = np.random.binomial(1, 0.01)
    if take == 0:
        continue
    if "normal" in packet:
        hy.append(0)
    else:
        hy.append(1)
    hX.append([float(x) for x in strlistpac[4:41]])

# hyperparameter optimization
hX = list(map(list, zip(*hX)))
bestparam = []
bestcla = 0
method = int(input("Enter method : 1 for SVM and 2 for Decision Tree:\n"))
start = time.time()
for i in range(100):
```

```

param = random.sample(range(0,37),10)
htX = []
for z in param:
    htX.append(hX[z])
htX = list(map(list, zip(*htX)))
xtrain, xtest, ytrain, ytest = train_test_split(htX, hy, test_size=0.3,
random_state=32)
if method == 1:
    model = SVC()
elif method == 2:
    model = tree.DecisionTreeClassifier()
model.fit(xtrain, ytrain)
cla = model.score(xtest, ytest)
if bestcla < cla:
    bestcla = cla
    bestparam = param.copy()

htX = []
for z in bestparam:
    htX.append(hX[z])
htX = list(map(list, zip(*htX)))
if method == 1:
    model = SVC()
elif method == 2:
    model = tree.DecisionTreeClassifier()
model.fit(htX, hy)

with open('corrected', 'rb') as f:
    inputs = f.readlines()

packets = []
for item in inputs:
    packets.append(str(item)[2:(len(item)+1)])

X = []
y = []
for packet in packets:
    strlistpac = packet.split(",")

    if "normal" in packet:
        y.append(0)
    else:
        y.append(1)
    X.append([float(x) for x in strlistpac[4:41]])

X = list(map(list, zip(*X)))
htX = []
for z in bestparam:
    htX.append(X[z])
X = list(map(list, zip(*htX)))

print("The packets are correctly classified with probability:\n",
model.score(X,y))
end = time.time()
print("Computational time : \n", end-start)

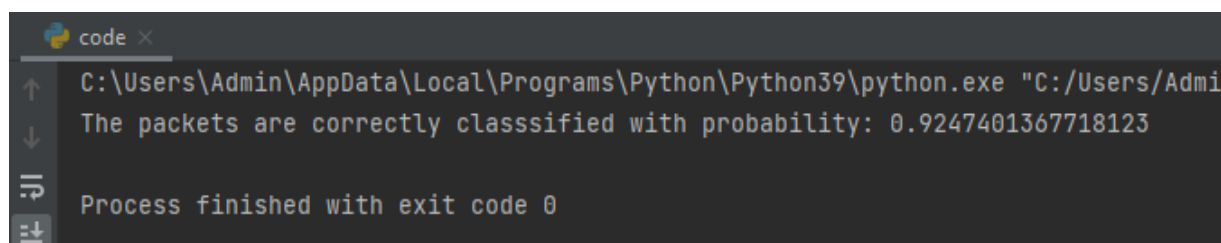
```

Our program opens the file 'kddcup.data_10_percent_corrected' (and not 'kddcup.data.corrected' because it is VERY large ~ 5 000 000 packets), and samples it, because even if it the one tenth of the original input, it is still very big for hyperparameter optimization and the model training. Finally, we have around 5000 packets for the training.

The hyperparameter optimization process is done by choosing randomly 10 of the features, training the model to 70% of the data, and testing it on the rest 30%. Then we get the correct classification probability. We repeat this process a hundred times, and we save the best probability, along with the features that we used to obtain it. After that, we train our model again with the features that gave the best result. This way, we can obtain up to 99% probability of correct classification of the test data.

Then we open and read the file 'corrected', and we test it on the already trained model. This file contains around 300 000 packets, and some of them are even not the same category of attacks as the ones in the training data, but our algorithm manages to classify them correctly with probability of around 93%. This happens, as it is believed that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants.

Below we show the output of one run of the program. Note that as the train data, and the hyperparameter optimization, and thus the model training is done



```
code x
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/Admin...
The packets are correctly classssified with probability: 0.9247401367718123
Process finished with exit code 0
```

probabilistically, each run will result in a different output.

```
Enter method : 1 for SVM and 2 for Decision Tree:
1
The packets are correctly classified with probability:
0.8701600172331198
Computational time :
23.99616837501526

Process finished with exit code 0
```

However, we have run the script for the whole data package and the results were the following:

(Computational time in sec)

```
Enter method : 1 for SVM and 2 for Decision Tree:
1
The packets are correctly classified with probability:
0.9193322809127124
Computational time :
1171.524447441101

Process finished with exit code 0
```

2. Decision Trees:

A decision Tree Classifier is usually used when our data are not easily separable from each other thus the linear approximation leads to large deviations and errors hence low performance. Each node in a decision tree represents a test on an attribute and each branch while we are descending the tree corresponds to a possible answer/value for that attribute. We follow a up-down greedy approach to create the tree.

When we run the program for the whole data package the results are :

```
Enter method : 1 for SVM and 2 for Decision Tree:
2
The packets are correctly classified with probability:
0.905912310427645
Computational time :
28.667895555496216

Process finished with exit code 0
```

```
Enter method : 1 for SVM and 2 for Decision Tree:
2
The packets are correctly classified with probability:
0.9053818132714313
Computational time :
32.21821880340576
Training Set Mean Accuracy = 1.0
Test Set Mean Accuracy = 0.99
```

While when we train the smaller data package we get as results:

```
Enter method : 1 for SVM and 2 for Decision Tree:
2
The packets are correctly classified with probability:
0.9244507746866047
Computational time :
7.1829833984375

Process finished with exit code 0
```

We can observe that the executing time is less by far from the first method. Specifically, the accurate difference between the computational times is:

$$Dt = \text{time_method1} - \text{time_method2} = 1171.524 - 28.667 = 1142.857 \text{ sec} = 19 \text{ min}$$

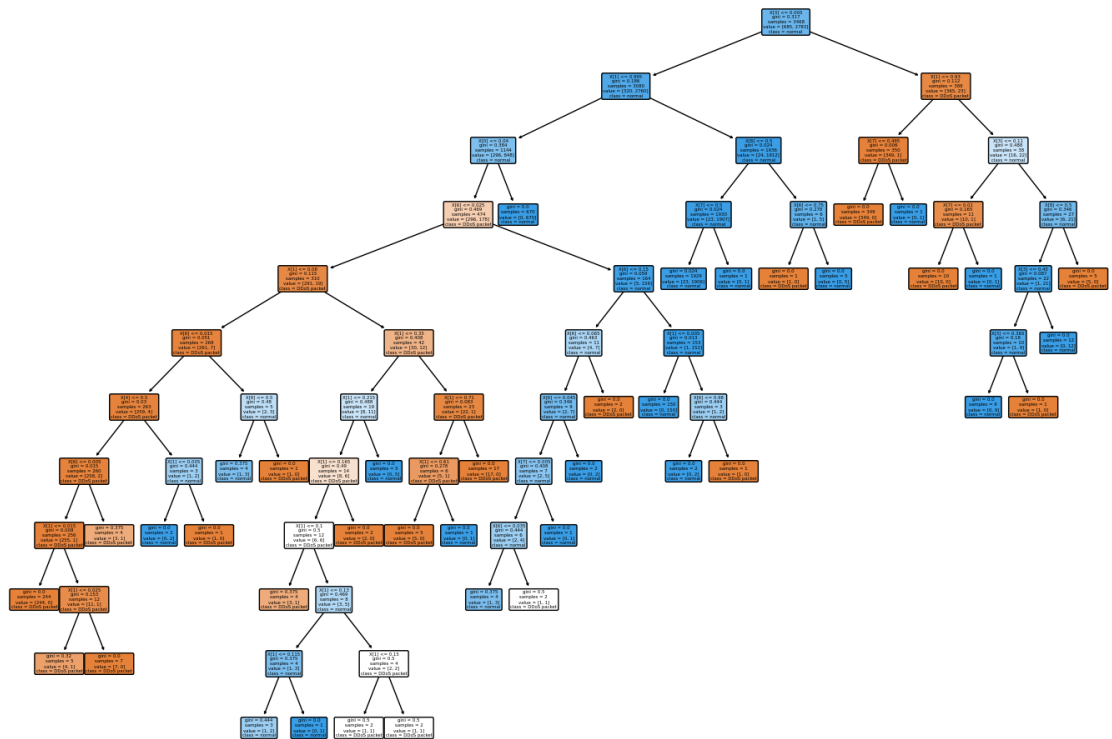
While the probability of each method is in the first case : 0.919 and in the latter: 0.905

$$Dp = \text{prob_method1} - \text{prob_method2} = 0.014$$

It is clear that the tree decision classifier is the best choice in these case, as it needs less computational time and the probabilities are close to each other (the difference is almost negligible in contrast with the execution time). However, to reduce memory consumption, the complexity and the size of the tree we could reduce the size of the data and the results would be similar.

```
Enter method : 1 for SVM and 2 for Decision Tree:
2
The packets are correctly classified with probability:
0.905642239148118
Computational time :
8.299367189407349
Training Set Mean Accuracy = 0.99
Test Set Mean Accuracy = 0.99
```

We can also see the representation of the decision tree:



(it is saved also as a png when you run the program so it has better resolution there for studying it)

Note: When we run the Decision Tree this method is able to handle different data types so the categorical ones must not be a problem but due to the lack of ability to support that data by ScikitLearn we create a function that encodes them and we include them in the algorithm as well.

Comparison:

Now we discuss the advantages and disadvantages of the two methods:

Support Vector Machine	Decision Tree
Can work efficiently with small amount of training data	Can work efficiently in both small and big amount of training data

Can only work with (continuous) numeric features as parameters (in our problem, features like the protocol, cannot be used)	Can work with both categorical and numerical data (continuous)
On its own, it can only perform linear classifications (with the kernel trick it can overcome this limitation)	They derive hyper-rectangles in input space to solve the problem
The final probability is around 93%	The final probability for the small data package is around 92% while for the whole data is approximately 90%
High time complexity	Low time complexity

Note: Decision Trees works better if our data are categorical and it can deal with colinearity more efficiently than SVM, as the former can work with both categorical and numerical data.

However, we will chose the algorithm based on what we want to achieve(depending on our needs if we wish for high accuracy or low time complexity) and furthermore we should bare in mind the type of our data.