Internet Firewall Action Classification

Classification project

- Prahar Ijner

Abstract

This project explores a dataset extracted from the network activity at a university to model the behaviour of a network firewall. The models developed use statistical learning algorithms which were chosen for their non-linear decision boundaries. Due to the unbalanced nature of the dataset, we use a 5 fold cross validation to obtain estimates of the accuracy, sensitivity, specificity, positive predictive value (PPV), negative predictive value (NPV), F1 score, and area under the ROC (AUC), which are used to evaluate the effectiveness of the models.

The project was developed using Python 3.7 and all models were created using the Scikit-learn (0.23.1) library [1]. Other libraries used for the project include Numpy (1.18.5), Pandas (1.0.4), Matplotlib (3.2.2), and Seaborn (0.10.1).

Introduction

Firewalls are essential parts of any network as they monitor the network traffic and block potentially harmful communication requests. The firewall acts as a barrier between the LAN and the rest of the internet. It can be configured with security rules which determine what connections are allowed and/or the ones that should be blocked. These rules are set by a network or system administrator based on the organization's needs and are usually not known by the users of the network. In general, rules can be based on the source of the traffic, destination of the traffic, and type of service the communication is using (protocol).

The goal of this project is to model the behaviour of a firewall based on source and destination port numbers (which correspond to the service being used), port forwarding by NAT, amount of data exchanged between the devices, and the rate of data transfer. This project will go over the different classification models suitable for this dataset, and evaluation of each of the models based on their overall accuracy, confusion matrix, sensitivity, specificity, ROC, and F1 score. The models will be built on the dataset mentioned below and will be trained on identical training and testing splits of the dataset to ensure an un-biased comparison.

Dataset

The dataset used for this project is the Internet Firewall Dataset [1]. The dataset was collected from the internet traffic records on Firat Universit 's firewall. The dataset consists of 65532 records, each having 12 attributes. The attributes are described in Table 1 below. For this project, our target attribute is "Action", which defines the response of the firewall. The dataset describes four classes for action: allow, deny, drop, and reset-both. Allow defines the connections that should be allowed, deny defines the connections that should be blocked and denied in the future, drop silently drops the traffic for application, and reset-both sends a TCP reset to the client and server [2].

As seen in figure 1 and table 2, the dataset is heavily unbalanced with over half of the records belonging to the "allow" class and only 0.08% of the records belonging to the "reset-both" class. Thus, to ensure a training and testing split that is representative of the whole dataset, it is important to stratify the splits and maintain the ratios of classes. Furthermore, using a single train-test split may not give us an accurate diagnosis of the model for the minority classes as there may not be enough diversity within the testing set. So, we use a stratified k fold cross validation to split the dataset [3] [4]. The cross validation is performed by setting a seed for the random number generator to ensure each model gets the exact same splits of train-test splits for cross validation. Throughout this project, we use 5-fold cross-validation. This provides an 80-20 training to testing ratio for every fold. This ensures a large enough training set for a good model and since the results are cross validated, we have a more accurate representation of the errors. Each model evaluated in this project includes an error profile that consists of accuracy and other class specific evaluation metrics which are averaged over the 5 cross validation loops.

The pairwise plots in figure 2 show that there are no dominant attributes that can be used to purely classify the dataset. Moreover, it shows the non-linear nature of most attributes, which is expected from nominal types. Since we do not have a very large number of attributes compared to the number of samples, performing a dimensionality reduction is not deemed necessary.

Attribute	**Description**	**Type**
Source Port	Port number of source (usually randomly assigned)	Nominal
Destination Port	Port number of server (pre-defined integer corresponding to protocol used for communication)	Nominal
NAT Source Port	Network address translation for source port	Nominal
NAT Destination Port	Network address translation for destination port	Nominal
Action	Action to be taken by firewall. This is the response variable. Classes: allow, deny, drop, reset-both	Nominal
Bytes	Number of bytes involved in communication	Ordinal, discrete
Bytes Sent	Number of bytes sent to server	Ordinal, discrete
Bytes Received	Number of bytes received from server	Ordinal, discrete
Packets	Number of packets involved in data transaction	Ordinal, discrete
Elapsed Time (sec)	Time in seconds elapsed since connection was established	Ordinal, discrete
pkts_sent	Number of packets sent to server	Ordinal, discrete
pkts_received	Number of packets received from server	Ordinal, discrete

Table 1 Data description of the Internet Firewall Dataset

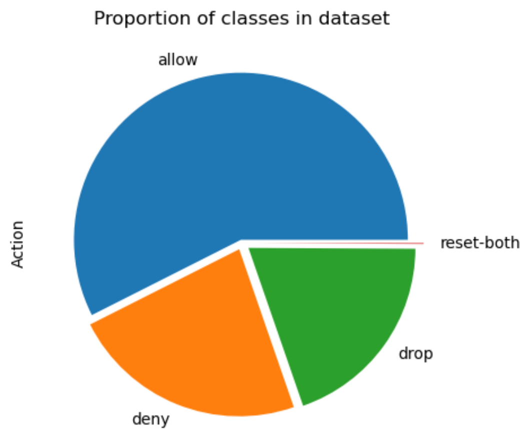


Figure 1 Pie chart indicating the proportions of classes within the dataset

Action	**Count**	**%**
allow	37640	57.44
deny	14987	22.87
drop	12851	19.61
reset-both	54	0.08

Table 2 Counts and percentage of each class within the dataset

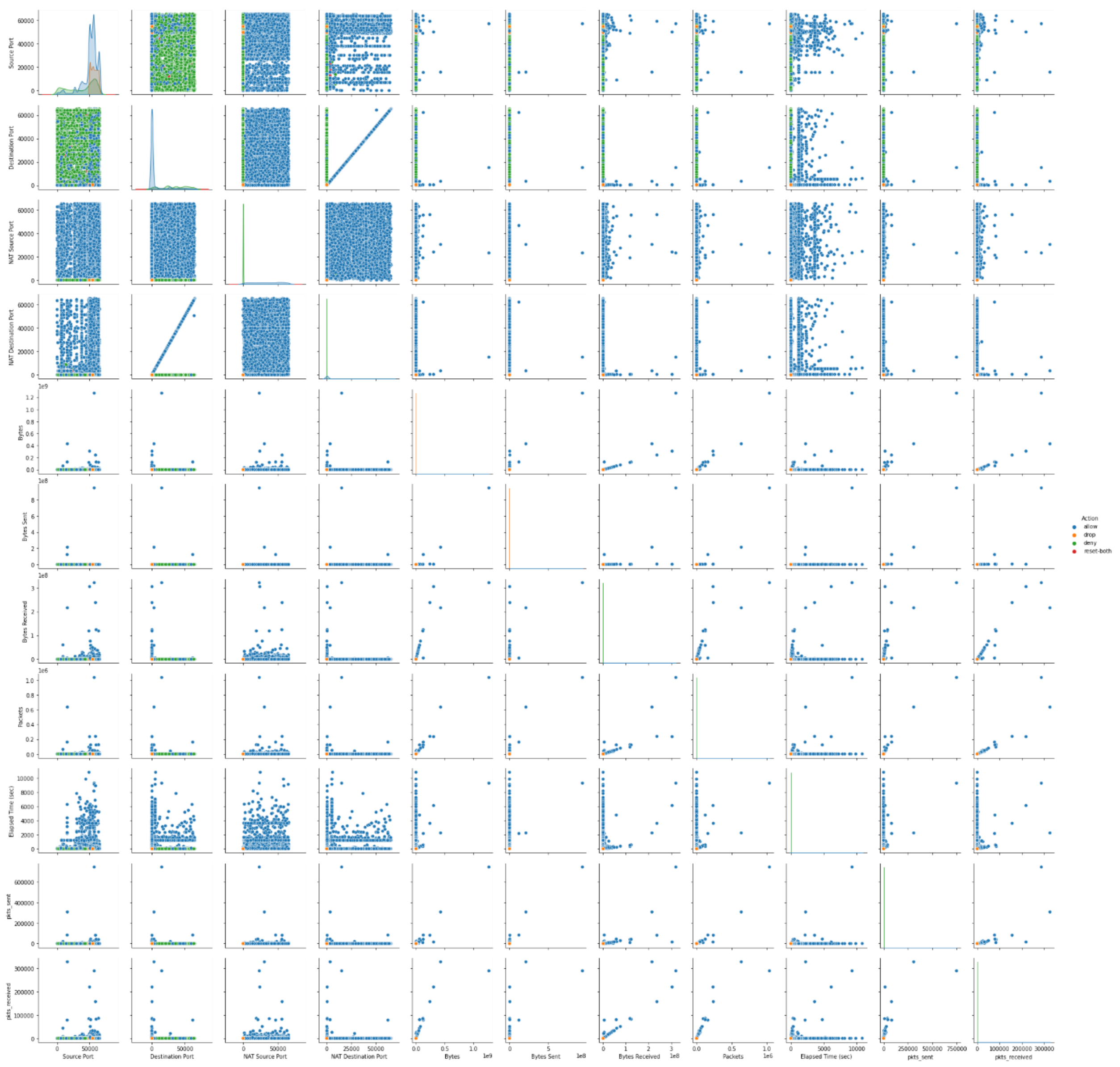


Figure 2 Pairwise plot of all the classification attributes in the Internet Firewall Dataset color coded according to the “Action” class

Models

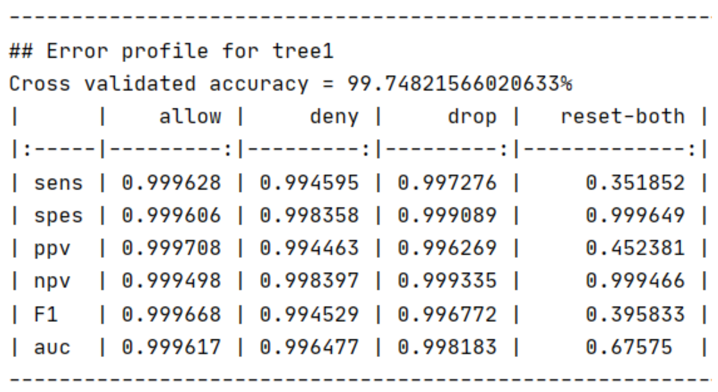
The dataset selected for this project is partially non-linear, in that 4 of the 11 classification attributes are nominal discrete variables. Since they do not have an order, we would expect the classes to form clusters with non-linear decision boundaries rather than be separated by linear decision boundaries. For this reason, all the models evaluated use algorithms that produce non-linear or combination of many linear decision boundaries.

Model 1 (tree1) – Simple decision tree

The first model evaluated was a simple decision, called `tree1`, generated using the code snippet below. The splits are made based on the entropy of the entropy of the nodes.

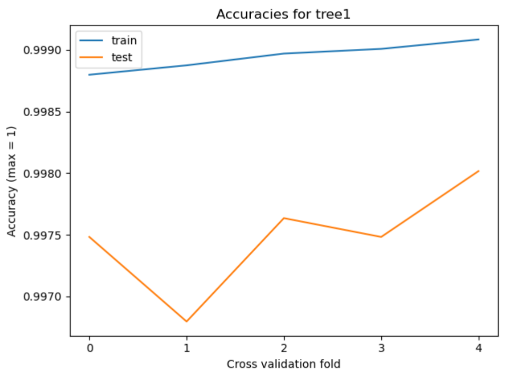
```
from sklearn.ensemble import RandomForestClassifier

tree1 = RandomForestClassifier(n_estimators=1, n_jobs=-1, criterion="entropy")
tree1.fit(X, y)
```

The results of this model are detailed in figure 3 below. The model produced an overall accuracy of 99.75%, with all but one errors being associated with the "deny" class. There is also a consistent difference between the training and testing accuracies, which hint at some level of overfitting. However, this is of the order 10^{-3} and can be ignored.

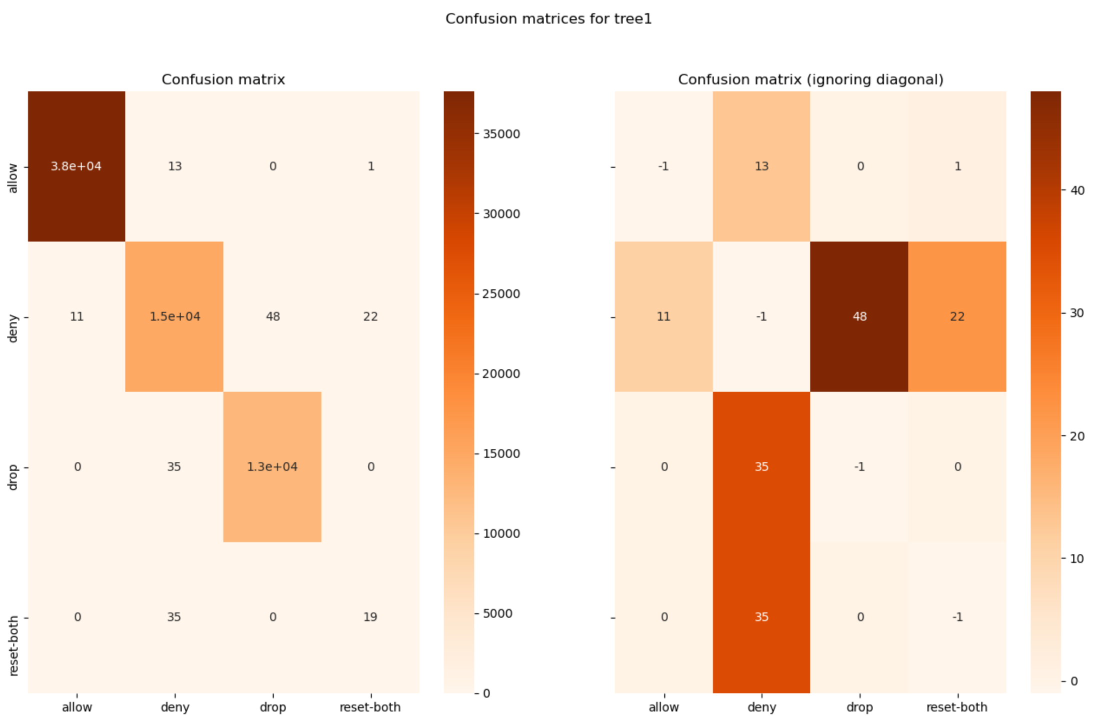


```
-----------------------------------------------------------
## Error profile for tree1
Cross validated accuracy = 99.74821566020633%
|      |    allow |     deny |     drop |   reset-both |
|:-----|---------:|---------:|---------:|-------------:|
| sens | 0.999628 | 0.994595 | 0.997276 |     0.351852 |
| spes | 0.999606 | 0.998358 | 0.999089 |     0.999649 |
| ppv  | 0.999708 | 0.994463 | 0.996269 |     0.452381 |
| npv  | 0.999498 | 0.998397 | 0.999335 |     0.999466 |
| F1   | 0.999668 | 0.994529 | 0.996772 |     0.395833 |
| auc  | 0.999617 | 0.996477 | 0.998183 |     0.67575  |
-----------------------------------------------------------
```

(a)



(b)



(c)

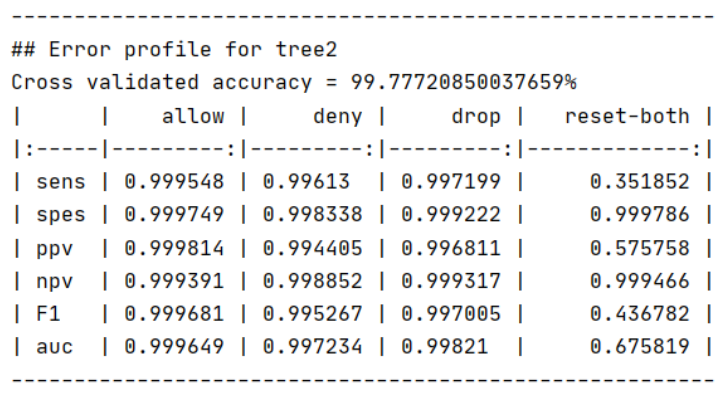
Figure 3 (a): Validation metrics for model tree1, (b): training and testing accuracies during each validation fold for model tree1, (c): Confusion matrix for tree1. The right confusion matrix ignores the values in the diagonals for better visual representation of the misclassifications

Model 2 (tree2) – Decision tree with weight balancing

As mentioned earlier, the dataset we are working with in this project is severely imbalanced. This can cause biases in impurity measures (entropy), leading to sub-optimal splits for nodes. The second model we test uses balanced class weights, assigning a weight to each class that is inversely proportional to its relative frequency [4] [5]. The following code snippet was used to create the model. The splits in this tree were based on the Gini index as it is slightly more efficient without a loss in information.

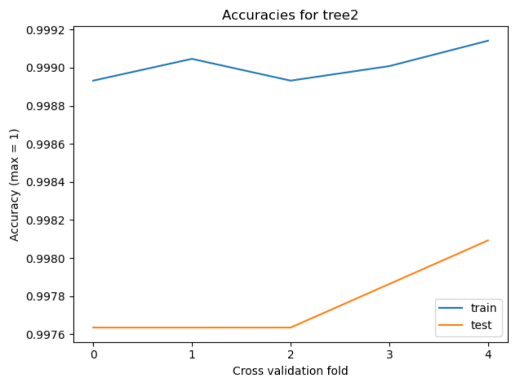
```python
from sklearn.ensemble import RandomForestClassifier

tree2 = RandomForestClassifier(
    n_estimators=1, n_jobs=-1, criterion="gini", class_weight="balanced"
)
tree2.fit(X, y)
```

The error profile for `tree2` was as follows

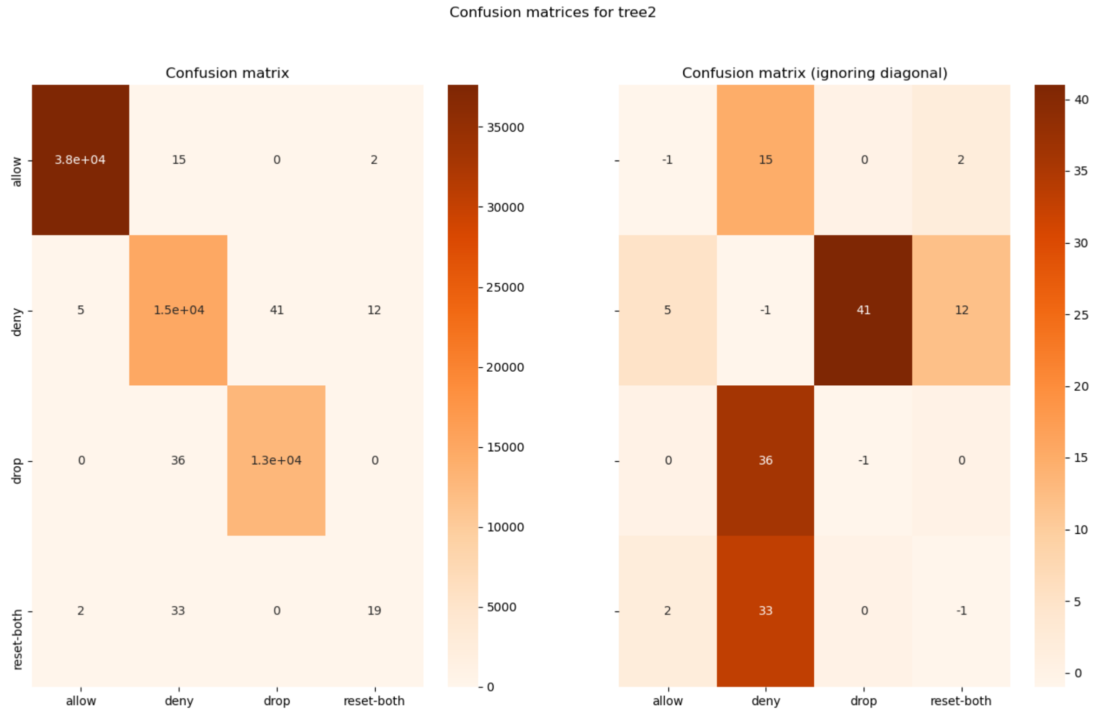


```
-----------------------------------------------------------
## Error profile for tree2
Cross validated accuracy = 99.77720850037659%
|      |    allow |     deny |     drop |   reset-both |
|:-----|---------:|---------:|---------:|-------------:|
| sens | 0.999548 | 0.99613  | 0.997199 |     0.351852 |
| spes | 0.999749 | 0.998338 | 0.999222 |     0.999786 |
| ppv  | 0.999814 | 0.994405 | 0.996811 |     0.575758 |
| npv  | 0.999391 | 0.998852 | 0.999317 |     0.999466 |
| F1   | 0.999681 | 0.995267 | 0.997005 |     0.436782 |
| auc  | 0.999649 | 0.997234 | 0.99821  |     0.675819 |
-----------------------------------------------------------
```

(a)



(b)



(c)

Figure 4 (a): Validation metrics for model tree2, (b): training and testing accuracies during each validation fold for model tree2, (c): Confusion matrix for tree2. The right confusion matrix ignores the values in the diagonals for better visual representation of the misclassifications

Using weight balancing clearly improves the overall accuracy of the tree. However, it does not affect the classification of "reset-both", and very slightly reduces the classification of "drop". However, the gain in overall accuracy seems to be worth this trade-off.

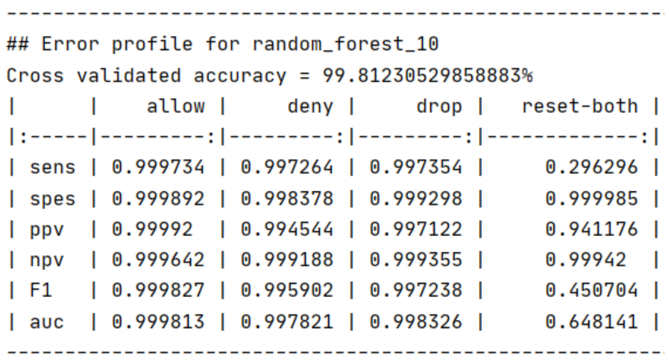
Model 3 (random_forest_10, random_forest_100) – Forest of 10 decision trees

A random forest is a combination of multiple decision trees voting together towards the final prediction. Since we have multiple classifiers in an ensemble, it is less prone to overfitting. As model 2 was better than model 1, we use it as our base classifier for the ensemble. Model 3 was developed using 10 trees using the architecture of model 2, using the code below.

```python
from sklearn.ensemble import RandomForestClassifier

random_forest_10 = RandomForestClassifier(
    n_estimators=10, n_jobs=-1, class_weight="balanced", criterion="gini"
)
random_forest_10.fit(X, y)
```

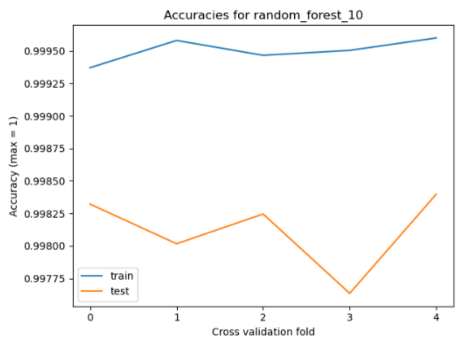
The error profile for `random_forest_10` again shows a clear improvement in the overall accuracy for classification. We can see an improvement in the sensitivity, specificity, positive and negative predictive value (PPV and NPV), F1 score, and area under ROC (AUC) for the classification of “allow”, “deny”, and “drop”. We can also see an improvement in the specificity, PPV, F1, and AUC for “reset-both” class. The classification matrix also looks cleaner for model 3 compared to the single decision trees. The downside of this model is that we have a smaller number of true positives for the class “reset-both”, but given the nature of the dataset, this may not be a major concern. The same architecture was repeated to generate a forest of 100 trees (`random_forest_100`) using the code below. However, there was no improvement compared to `random_forest_10` and thus it was discarded.

```python
from sklearn.ensemble import RandomForestClassifier

random_forest_100 = RandomForestClassifier(
    n_estimators=100, n_jobs=-1, class_weight="balanced", criterion="gini"
)
random_forest_100.fit(X, y)
```

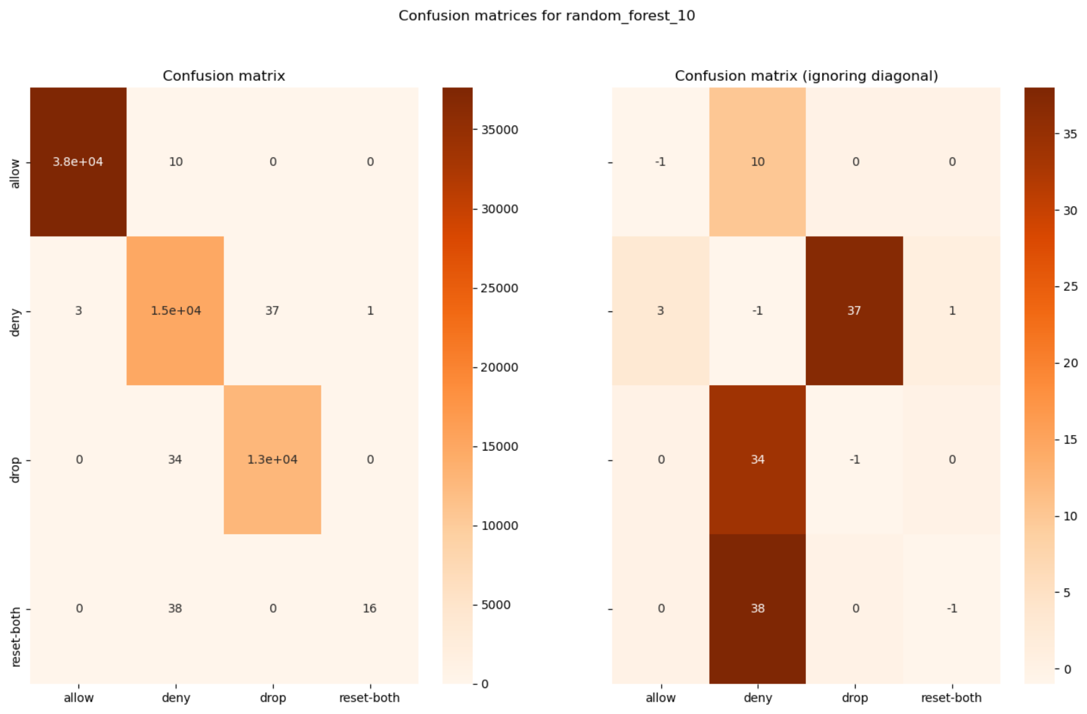


```
------------------------------------------------------------
## Error profile for random_forest_10
Cross validated accuracy = 99.81230529858883%
|      |     allow |      deny |      drop |    reset-both |
|:-----|---------:|---------:|---------:|-------------:|
| sens | 0.999734 | 0.997264 | 0.997354 |      0.296296 |
| spes | 0.999892 | 0.998378 | 0.999298 |      0.999985 |
| ppv  | 0.99992  | 0.994544 | 0.997122 |      0.941176 |
| npv  | 0.999642 | 0.999188 | 0.999355 |      0.99942  |
| F1   | 0.999827 | 0.995902 | 0.997238 |      0.450704 |
| auc  | 0.999813 | 0.997821 | 0.998326 |      0.648141 |
------------------------------------------------------------
```

(a)



(b)



(c)

Figure 5 (a): Validation metrics for random_forest_10, (b): training and testing accuracies during each validation fold for model random_forest_10, (c): Confusion matrix for random_forest_10. The right confusion matrix ignores the values in the diagonals for better visual representation of the misclassifications

Model 4 (knn_1, knn_5, knn_15) – Nearest neighbour classification

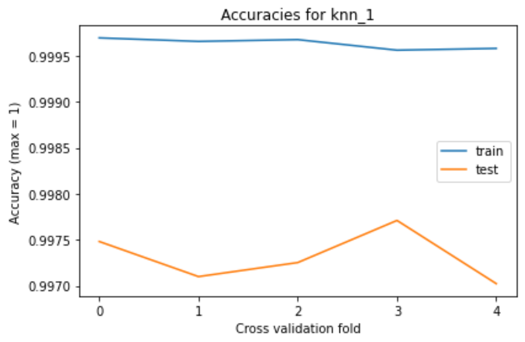
Moving away from decision trees, another approach for non-linear decision boundaries is k-nearest neighbours which classifies points in a multidimensional space based on the classes of "k" nearest neighbours, calculated by a distance metric. The distance metric chosen in this case was the hamming distance as it is better suited for integer valued and categorical attributes. The models were generated using the code snippet below.

```
from sklearn.neighbors import KNeighborsClassifier

knn_1 = KNeighborsClassifier(n_neighbors=1, n_jobs=-1, metric="hamming")
knn_5 = KNeighborsClassifier(n_neighbors=5, n_jobs=-1, metric="hamming")
knn_15 = KNeighborsClassifier(n_neighbors=15, n_jobs=-1, metric="hamming")

knn_1.fit(X, y)
knn_5.fit(X, y)
knn_15.fit(X, y)
```

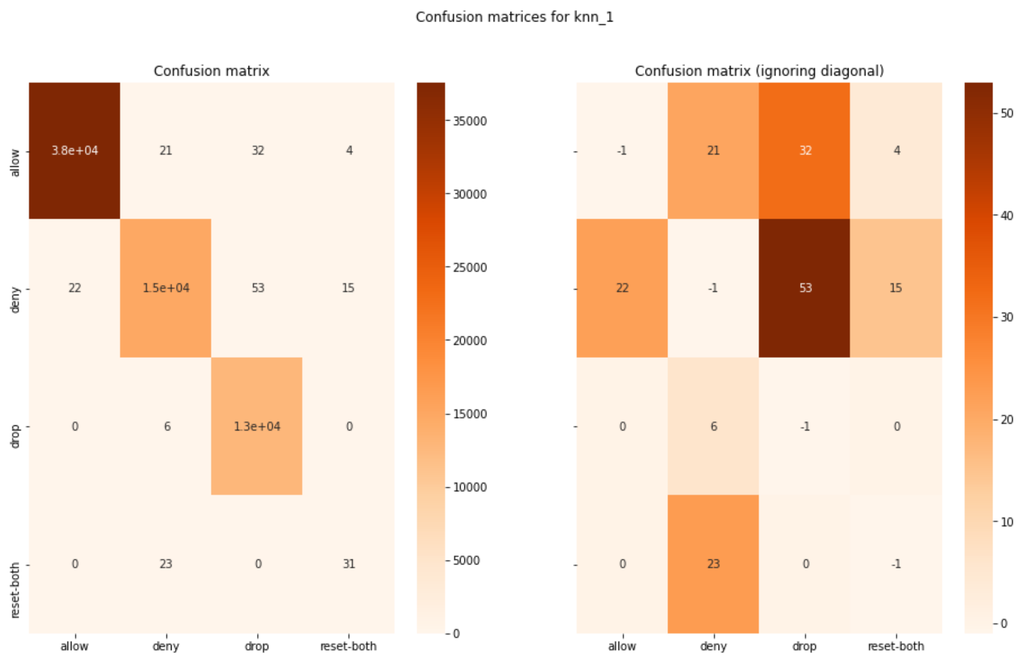
Error profile for `knn_1`

```
------------------------------------------------------------
## Error profile for knn_1
Cross validated accuracy = 99.7314289897036%
|      |     allow |      deny |      drop |   reset-both |
|:-----|---------:|---------:|---------:|-------------:|
| sens | 0.998486 | 0.993995 | 0.999533 |     0.574074 |
| spes | 0.999211 | 0.999011 | 0.998387 |     0.99971  |
| ppv  | 0.999415 | 0.996655 | 0.993426 |     0.62     |
| npv  | 0.997959 | 0.998221 | 0.999886 |     0.999649 |
| F1   | 0.99895  | 0.995323 | 0.99647  |     0.596154 |
| auc  | 0.998848 | 0.996503 | 0.99896  |     0.786892 |
------------------------------------------------------------
```

(a)



(b)



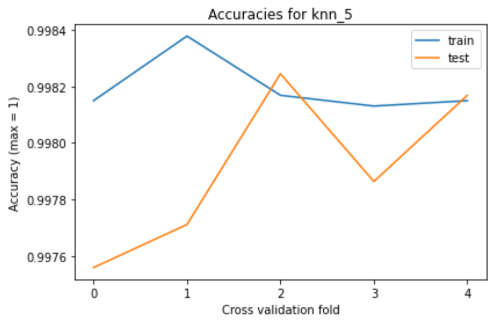
(c)

Figure 6 (a): Validation metrics for knn_1, (b): training and testing accuracies during each validation fold for model knn_1, (c): Confusion matrix for knn_1.

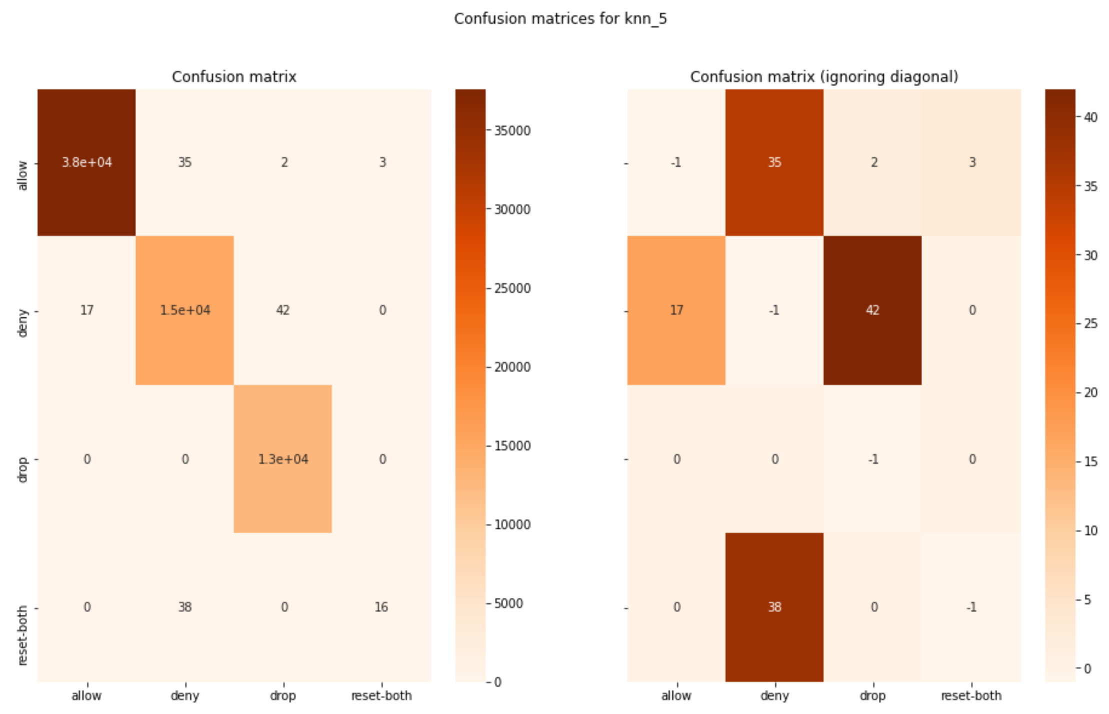
Error profile for `knn_5`

```
------------------------------------------------------------
## Error profile for knn_5
Cross validated accuracy = 99.79094266793345%
|      |     allow |      deny |      drop |   reset-both |
|:-----|---------:|---------:|---------:|-------------:|
| sens | 0.998937 | 0.996063 | 1        |     0.296296 |
| spes | 0.999391 | 0.998556 | 0.999165 |     0.999954 |
| ppv  | 0.999548 | 0.995134 | 0.996588 |     0.842105 |
| npv  | 0.998567 | 0.998832 | 1        |     0.99942  |
| F1   | 0.999243 | 0.995598 | 0.998291 |     0.438356 |
| auc  | 0.999164 | 0.997309 | 0.999582 |     0.648125 |
------------------------------------------------------------
```

(a)



(b)



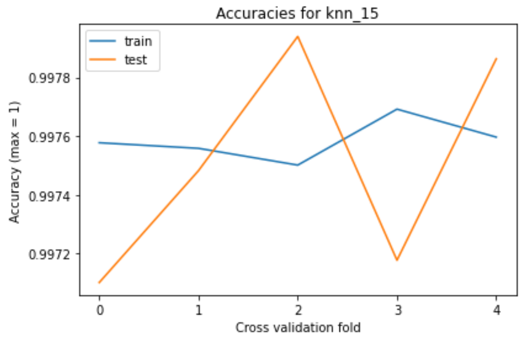
(c)

Figure 7 (a): Validation metrics for knn_5, (b): training and testing accuracies during each validation fold for model knn_5, (c): Confusion matrix for knn_5

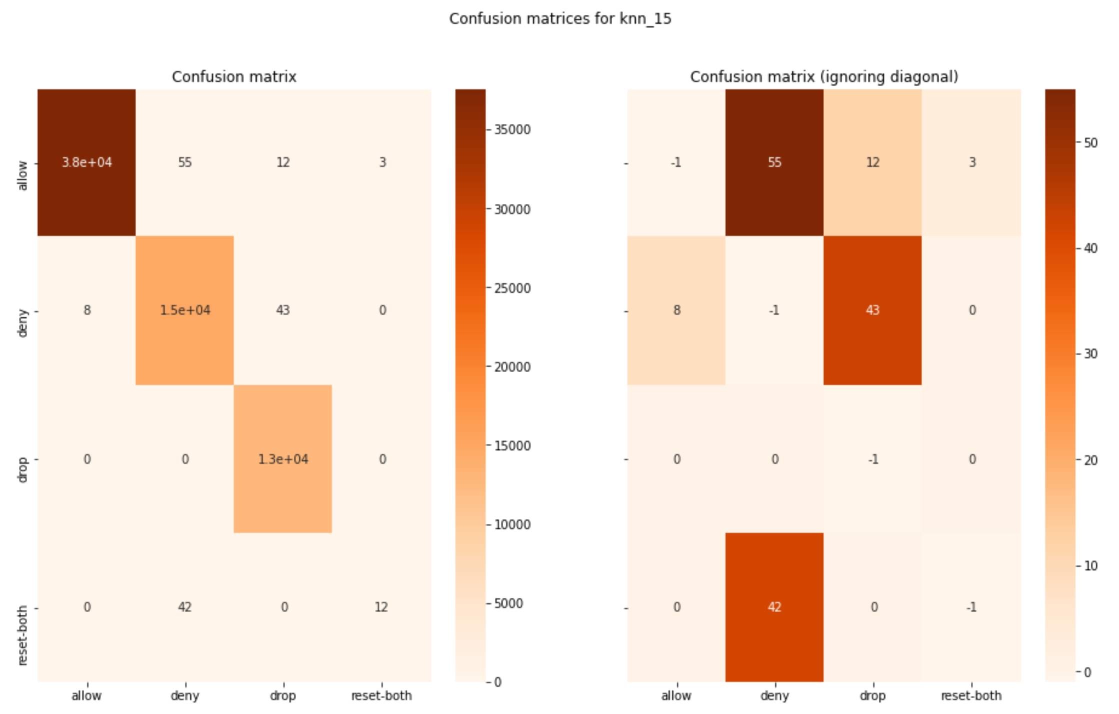
Error profile for `knn_15`

```
----------------------------------------------------------
## Error profile for knn_15
Cross validated accuracy = 99.75126723173015%
|      |     allow |      deny |      drop |   reset-both |
|:-----|----------:|----------:|----------:|-------------:|
| sens | 0.99814   | 0.996597  | 1         |     0.222222 |
| spes | 0.999713  | 0.998081  | 0.998956  |     0.999954 |
| ppv  | 0.999787  | 0.993548  | 0.995738  |     0.8      |
| npv  | 0.997496  | 0.99899   | 1         |     0.999359 |
| F1   | 0.998963  | 0.99507   | 0.997865  |     0.347826 |
| auc  | 0.998927  | 0.997339  | 0.999478  |     0.611088 |
----------------------------------------------------------
```

(a)



(b)



(c)

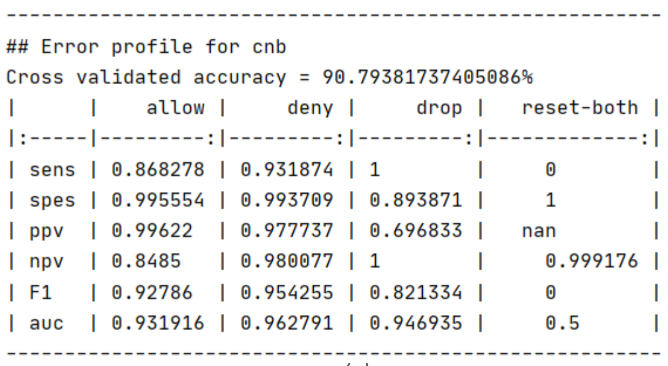
Figure 8 (a): Validation metrics for knn_15, (b): training and testing accuracies during each validation fold for model knn_15, (c): Confusion matrix for knn_15

The results of the `knn_1`, `knn_5`, and `knn_15` models show comparable results to the decision trees and random forest classifiers used in model 1, 2, and 3. Models `knn_5` and `knn_15` are the first models where the testing accuracy is greater than the training accuracy, indicating they are not overfitted. Although the `knn_1` model shows a great performance over the "reset-both" class, it comparatively suffers in performance for the other classes.

Model 6 (cnb) – Complement Naïve Bayes

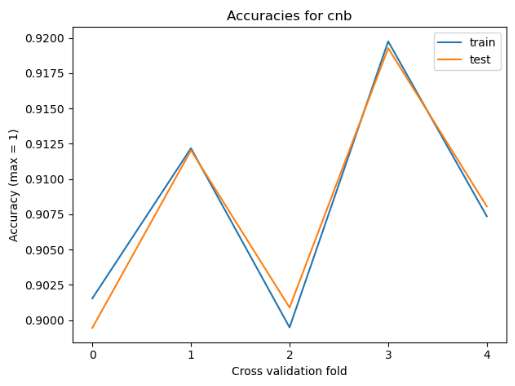
Our final model that we will be evaluating for this dataset is generated using Complement Naïve Bayes algorithm [4] [6]. This is an adaptation of multinomial naïve Bayes that is better suited for imbalanced datasets, like the one we are working with. However, it is not expected that `cnb`, even after adapting for imbalanced datasets, will perform better than our previous models as it works on conditional probabilities and we are working with a dataset where more than 50% of the records belong to a single class. The `cnb` was implemented by the following code.

```
from sklearn.naive_bayes import ComplementNB

cnb = ComplementNB()
cnb.fit(X, y)
```

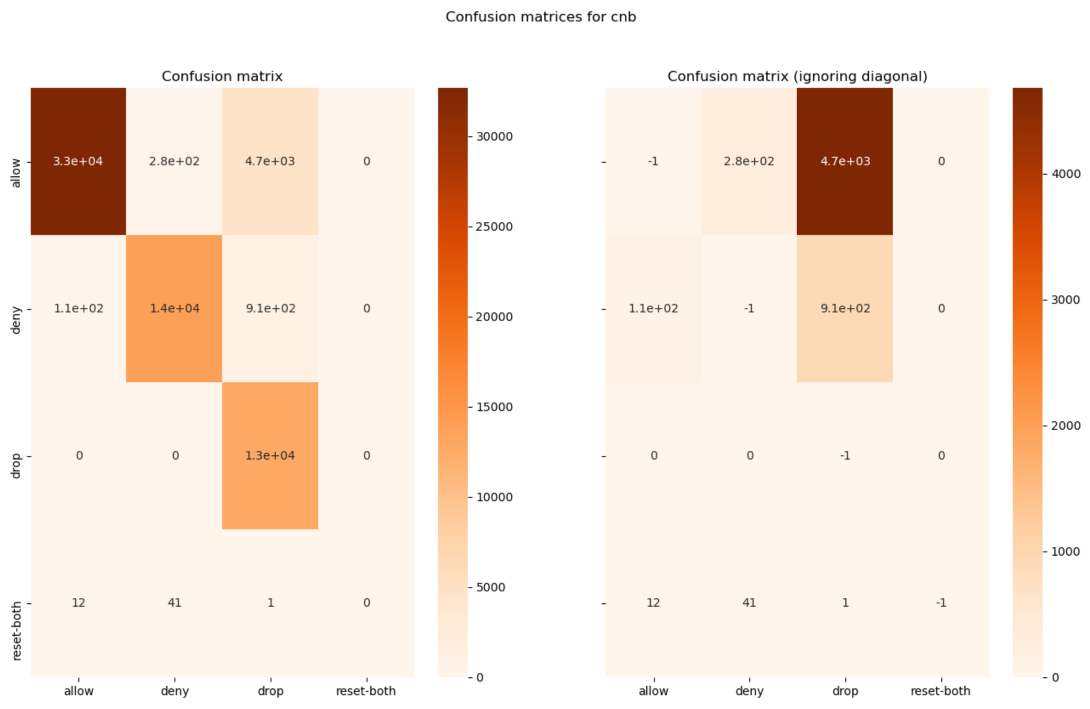


```
--------------------------------------------------------
## Error profile for cnb
Cross validated accuracy = 90.79381737405086%
|      |     allow |      deny |      drop |    reset-both |
|:-----|---------:|---------:|---------:|-------------:|
| sens | 0.868278 | 0.931874 | 1        |      0       |
| spes | 0.995554 | 0.993709 | 0.893871 |      1       |
| ppv  | 0.99622  | 0.977737 | 0.696833 |    nan       |
| npv  | 0.8485   | 0.980077 | 1        |      0.999176 |
| F1   | 0.92786  | 0.954255 | 0.821334 |      0       |
| auc  | 0.931916 | 0.962791 | 0.946935 |      0.5     |
--------------------------------------------------------
```

(a)



(b)



(c)

Figure 9 (a): Validation metrics for cnb, (b): training and testing accuracies during each validation fold for model cnb, (c): Confusion matrix for knn_15

As expected, the `cnb` model does not perform close to the other models we have evaluated. There are over 4000 “allow” cases that are classified as “drop”. Moreover, 110 “deny” cases are classified as “allow”, which is not great for a firewall implementation as it can lead to network attacks.

Results

To compare all the models described above, we look at three main metrics: cross validated accuracy, weighted AUC, and weighted F1 scores. We do not consider the other metrics reported in each of the error profiles as the AUC and F1 scores incorporate their effects. The weighted AUC for a model is computed adding all the individual class AUCs multiplied by the proportion of the dataset represented by that class. The weighted F1 score is calculated in a similar fashion, and these are reported in table 3 and 4. Table 5 summarizes the accuracies, weighted AUC, and weighted F1 scores for all the models. The evaluation metrics are graphically represented in figure 10; note that model `cnb` has been excluded from the visual comparison due to it's significantly lower performance.

Model	**AUC**				
	allow (57.44%)	**deny (22.87%)**	**drop (19.61%)**	**reset-both (0.08%)**	**Weighted AUC**
tree1	0.999617	0.996477	0.998183	0.67575	0.998359
tree2	0.999649	0.997234	0.99821	0.675819	0.998555
random_forest_10	0.999813	0.997821	0.998326	0.648141	0.998784
knn_1	0.998848	0.996503	0.99896	0.786892	0.998164
knn_5	0.999164	0.997309	0.999582	0.648125	0.998541
knn_15	0.998927	0.997339	0.999478	0.611088	0.998362
cnb	0.931916	0.962791	0.946935	0.5	0.941577

Table 3: Individual class and weighted AUC for all models

Model	**F1 Score**				
	allow (57.44%)	**deny (22.87%)**	**drop (19.61%)**	**reset-both (0.08%)**	**Weighted F1**
tree1	0.999668	0.994529	0.996772	0.395833	0.997442
tree2	0.999681	0.995267	0.997005	0.436782	0.997696
random_forest_10	0.999827	0.995902	0.997238	0.450704	0.997982
knn_1	0.99895	0.995323	0.99647	0.596154	0.997312
knn_5	0.999243	0.995598	0.998291	0.438356	0.997774
knn_15	0.998963	0.99507	0.997865	0.347826	0.997336
cnb	0.92786	0.954255	0.821334	0	0.912264

Table 4: Individual class and weighted F1 scores for all models

Model	Accuracy	Weighted AUC	Weighted F1
tree1	0.997482	0.998358581	0.997442
tree2	0.997772	0.998555438	0.997696
random_forest_10	0.998123	0.998784491	0.997982
knn_1	0.997314	0.998164097	0.997312
knn_5	0.997909	0.9985409	0.997774
knn_15	0.997513	0.998361604	0.997336
cnb	0.907938	0.941576806	0.912264

Table 5: Metrics to compare model performances

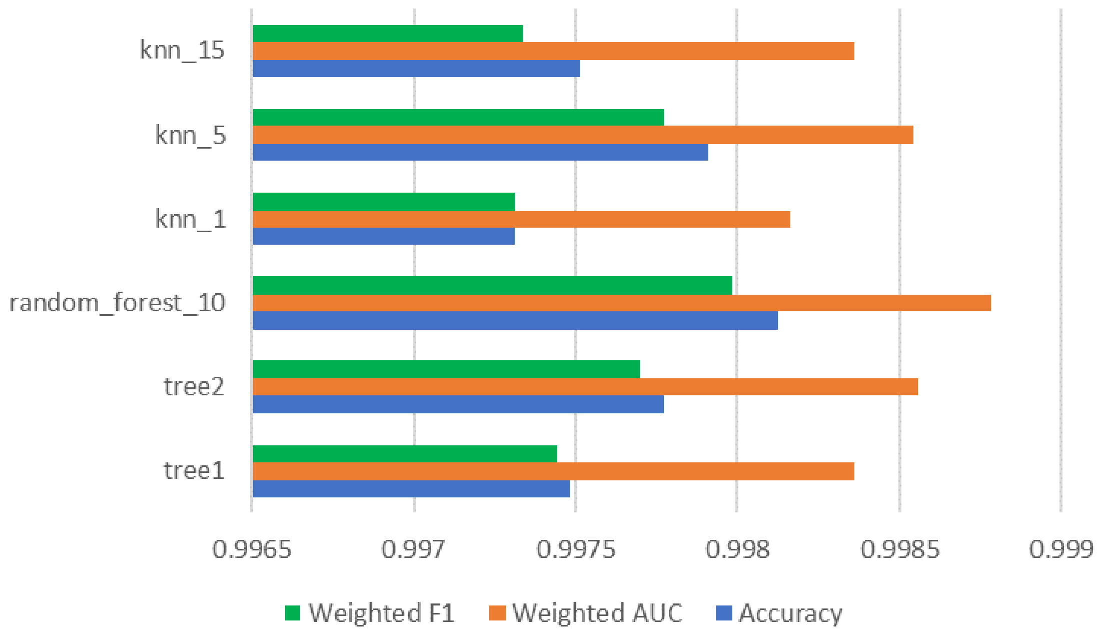


Figure 10: Visual comparison of metrics for all models except cnb

Looking at the results from the models, it is clear that Bayesian models are not ideal for this application as it has an accuracy around 9% lower than the other models. Model 3 (`random_forest_10`), which uses a random forest with 10 decision trees has the best overall accuracy. However, models `knn_5` and `tree2` are very close in terms of weighted AUC. For our dataset, it is more important to have a strong decision boundary between class "allow" and the other classes than among other classes. So, comparing the AUC and F1 values for “allow” in `knn_5` (AUC=0.999164, F1=0.999243), `random_forest_10` (AUC=0.999813, F1=0.999827), and `tree2` (AUC=0.999649, F1=0.999681), we can conclude that `random_forest_10` is a better choice.

Overall, we have a very successful classification with the random forests and the nearest neighbours classifiers. The cross validated accuracies are consistently over 99% and the AUCs for classes “allow”, “deny”, and “drop” are over 0.99 for all models except `cnb`.

The classes “allow” and “drop” are well identified throughout all models. The model 1 through 3, which use decision trees favour the “allow” class, while models 4 and 5 show slightly better grouping for the “drop” class. The class “reset-both” is consistently misclassified throughout the models. The only model that correctly classified at least 50% of the records for “reset-both” is `knn_1`. It is expected for this class to have a poor decision boundary as we have an imbalanced with only 0.08% of the records belonging to this class. If we ignore this class, “deny” and “drop” are commonly confused with each other. However, in a practical situation, this may not be a major concern given that both block unwanted traffic through the network.

References

[1] "Internet Firewall Data Data Set," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data. [Accessed 07 11 2020].

[2] F. Ertam and M. Kaya, "Classification of firewall log files with multiclass support vector machine," in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, 2018.

[3] "sklearn.model_selection.StratifiedKFold — scikit-learn 0.23.2 documentation," Scikit learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html?highlight=stratified%20k%20fold#sklearn.model_selection.StratifiedKFold. [Accessed 07 11 2020].

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel and P. Prettenhofer, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, no. 85, pp. 2825-2830, 2011.

[5] "3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.23.2 documentation," Scikit learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemble.RandomForestClassifier. [Accessed 07 11 2020].

[6] "sklearn.naive_bayes.ComplementNB — scikit-learn 0.23.2 documentation," Scikit learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html?highlight=complement%20naive#sklearn.naive_bayes.ComplementNB. [Accessed 07 11 2020].