CSEC 620 | Assignment 3 report

Team 4 - Nick Mangerian, Jacob Ruud, Hugo Tessier Due on 10-28-2021

---

1. **Report the number of benign samples you used (step 5), in addition to the ratio between benign and malware samples.**

There were 120,000 files in total. We considered all the 5,500 malwares of the dataset and a sample of 10,000 benign files. Those files have been preprocessed to identify the features (89108 unique features) and are saved into a table.
For the sack of computation expense, we only consider a subset of the preprocessed table during the classification. The ratio between benign and malware samples can be chosen by the user during the split between training and testing datasets. The samples are selected randomly and training samples are separated to testing samples to avoid the overfitting of our models.

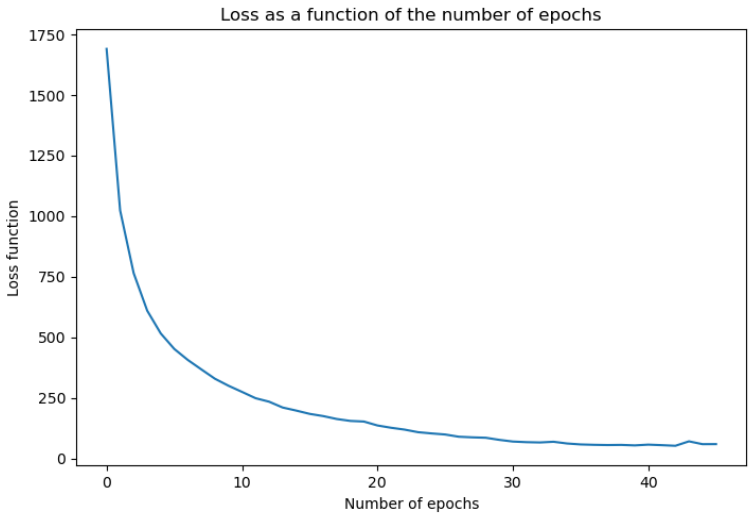
2. **Performance**
   a. ***Report the performance of your classifier based on its accuracy.***

Performance of the Binary classifier:
Hyperparameters:
- C = 0.2
- learning_rate = 0.01
- n_epoch = 100

Confusion matrix:

	Benign	**Malware**
Benign	0.779605	0.009868
Malware	0.032072	0.178454

F1-Score: 0.8948
Accuracy: 95.81%
Execution time: 86.35s



We can observe that the convergence criterion set on the variation of the loss has been reached because it took less than 42 epochs to converge instead of 100 further discussions on these parameters will be made in the hyperparameter tuning section.

Performance of the multiclass classifier:
Hyperparameters:
- C = 0.5
- learning_rate = 0.01
- n_epoch = 10

Concerning the performances of this classifier, one can't consider precision and recall anymore. Therefore, the only performance metrics that we considered are then the confusion matrix, the accuracy and the execution time.

One-vs-All Confusion matrix: (20 biggest malware families)

	Glodream	Jifake	Spyset	BaseBridge	SmForw	Boxer	SendPay	Plankton	Opfake	DroidKungFu
Glodream	0.013986013986014000	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0
Jifake	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Spyset	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.0	0.04895104895104900	0.0	0.0	0.0	0.0	0.006993006993006990	0.0
SmForw	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Boxer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.0	0.013986013986014000	0.0	0.0	0.0
Plankton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.09090909090909090	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.06293706293706290	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.06293706293706290
Nisev	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MobileTx	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.006993006993006990	0.006993006993006990	0.013986013986014000	0.0	0.0	0.0	0.0
Kmin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Adrd	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0	0.006993006993006990	0.0
Imlog	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Iconosys	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0
Geinimi	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
GinMaster	0.0	0.0	0.0	0.0	0.006993006993006990	0.013986013986014000	0.0	0.0	0.0	0.0

temp-1

	Nisev	MobileTx	FakeDoc	FakeInstaller	Kmin	Adrd	Imlog	Iconosys	Geinimi	GinMaster
Glodream	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Jifake	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Spyset	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SmForw	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Boxer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Plankton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Nisev	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MobileTx	0.0	0.013986013986014000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.02097902097902100	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.1958041958041960	0.0	0.0	0.0	0.0	0.0	0.0
Kmin	0.0	0.0	0.0	0.0	0.013986013986014000	0.0	0.0	0.0	0.0	0.0
Adrd	0.006993006993006990	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0
Imlog	0.0	0.0	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0
Iconosys	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.02097902097902100	0.0	0.0
Geinimi	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.013986013986014000	0.0
GinMaster	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.006993006993006990

Accuracy: 68.53 %

Execution Time: 820 seconds

One-vs-One Confusion matrix: (20 biggest malware families)

temp

	GinMaster	Plankton	BaseBridge	Yzhc	Opfake	DroidKungFu	MobileTx	FakeDoc	FakeInstaller	Kmin
GinMaster	0.05970149253731340	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0
Plankton	0.0	0.08208955223880600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.05970149253731340	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
Yzhc	0.0	0.0	0.0	0.0	0.029850746268656700	0.0	0.0	0.0	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.07462686567164180	0.0	0.0	0.0	0.007462686567164180	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.07462686567164180	0.0	0.0	0.0	0.0
MobileTx	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.007462686567164180	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.022388059701492500	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.0	0.014925373134328400	0.0	0.0	0.0	0.23880597014925400	0.0
Kmin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.014925373134328400
Imlog	0.0	0.0	0.0	0.0	0.0	0.014925373134328400	0.0	0.0	0.0	0.0
Adrd	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0
Glodream	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
Geinimi	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
ExploitLinuxLotoor	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FoCobers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SerBG	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Hamob	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.007462686567164180	0.0
Vdloader	0.007462686567164180	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

temp-1

	Imlog	Adrd	Glodream	Geinir	ExploitLinuxLotoor	SendPay	FoCobers	SerBG	Hamob	Vdloader
GinMaster	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Plankton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Yzhc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MobileTx	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Kmin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Imlog	0.007462686567164180	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Adrd	0.0	0.007462686567164180	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
Glodream	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Geinimi	0.0	0.0	0.0	0.007	0.0	0.0	0.0	0.0	0.0	0.0
ExploitLinuxLotoor	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0
FoCobers	0.0	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0
SerBG	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0
Hamob	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Vdloader	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Accuracy: 68.66 %

Execution Time: 178 seconds

<u>Comment on the overall performances:</u>

For malware classification-as for the previous assignment- accuracy isn't the best metric as it doesn't take into account the recall and the precision. The F1-score is better as it penalizes the performance if either the recall or the precision is low. Thus, it will encourage models with few FN and FP, which is good for malware classification.

b. Provide the confusion matrix (only include the top 20 malware families) that you generated in step 5.

Normalized confusion matrix for the malware families:

One-vs-All normalized confusion matrix

	Glodream	Jifake	Spyset	BaseBridge	SmForw	Boxer	SendPay	Plankton	Opfake	DroidKungFu
Glodream	0.013986013986014000	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0
Jifake	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Spyset	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.0	0.04895104895104900	0.0	0.0	0.0	0.0	0.006993006993006990	0.0
SmForw	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Boxer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.0	0.013986013986014000	0.0	0.0	0.0
Plankton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.09090909090909090	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.06293706293706290	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.06293706293706290
Nisev	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MobileTx	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.006993006993006990	0.006993006993006990	0.013986013986014000	0.0	0.0	0.0	0.0
Kmin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Adrd	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0	0.006993006993006990	0.0
Imlog	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Iconosys	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0	0.0
Geinimi	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
GinMaster	0.0	0.0	0.0	0.0	0.006993006993006990	0.013986013986014000	0.0	0.0	0.0	0.0

temp-1

	Nisev	MobileTx	FakeDoc	FakeInstaller	Kmin	Adrd	Imlog	Iconosys	Geinimi	GinMaster
Glodream	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Jifake	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Spyset	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SmForw	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Boxer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Plankton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Nisev	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MobileTx	0.0	0.013986013986014000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.020979020979021000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.1958041958041960	0.0	0.0	0.0	0.0	0.0	0.0
Kmin	0.0	0.0	0.0	0.0	0.013986013986014000	0.0	0.0	0.0	0.0	0.0
Adrd	0.006993006993006990	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0	0.0
Imlog	0.0	0.0	0.0	0.0	0.0	0.0	0.006993006993006990	0.0	0.0	0.0
Iconosys	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.020979020979021000	0.0	0.0
Geinimi	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.013986013986014000	0.0
GinMaster	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.06993006993006990

One-vs-One normalized confusion matrix

temp

	GinMaster	Plankton	BaseBridge	Yzhc	Opfake	DroidKungFu	MobileTx	FakeDoc	FakeInstaller	Kmin
GinMaster	0.05970149253731340	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0
Plankton	0.0	0.08208955223880600	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.05970149253731340	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
Yzhc	0.0	0.0	0.0	0.0	0.029850746268656700	0.0	0.0	0.0	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.07462686567164180	0.0	0.0	0.0	0.007462686567164180	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.07462686567164180	0.0	0.0	0.0	0.0
MobileTx	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.007462686567164180	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.022388059701492500	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.0	0.014925373134328400	0.0	0.0	0.0	0.23880597014925400	0.0
Kmin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.014925373134328400
Imlog	0.0	0.0	0.0	0.0	0.0	0.014925373134328400	0.0	0.0	0.0	0.0
Adrd	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0
Glodream	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
Geinimi	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
ExploitLinuxLotoor	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FoCobers	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SerBG	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Hamob	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.007462686567164180	0.0
Vdloader	0.007462686567164180	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

temp-1

	Imlog	Adrd	Glodream	Geinir	ExploitLinuxLotoor	SendPay	FoCobers	SerBG	Hamob	Vdloader
GinMaster	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Plankton	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BaseBridge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Yzhc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Opfake	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DroidKungFu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MobileTx	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeDoc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FakeInstaller	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Kmin	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Imlog	0.007462686567164180	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Adrd	0.0	0.00746268656716418(	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0	0.0
Glodream	0.0	0.0	0.00746268656716418(	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Geinimi	0.0	0.0	0.0	0.007	0.0	0.0	0.0	0.0	0.0	0.0
ExploitLinuxLotoor	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SendPay	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0	0.0
FoCobers	0.0	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0	0.0
SerBG	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.007462686567164180	0.0	0.0
Hamob	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Vdloader	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

c. Discuss the results in your confusion matrix.

The False Negatives and False Positives in this matrix are less problematic than they are in the binary classifier's matrix as those samples are all malwares. The errors just correspond to reversal between the categories.

Moreover, this confusion matrix can help finding links between the families as some families are more easily confused with others, those may have similar features.

3. Hyperparameter tuning

a. Describe your hyper-parameter tuning process and show figures (with proper labeling) describing interesting results.

There are 3 hyperparameters which need to be tuned in our SVM implementation:

- The regularization weight **C**
- The learning rate **lr**
- The number of epochs of the Stochastic Gradient Descent: **n_epochs**

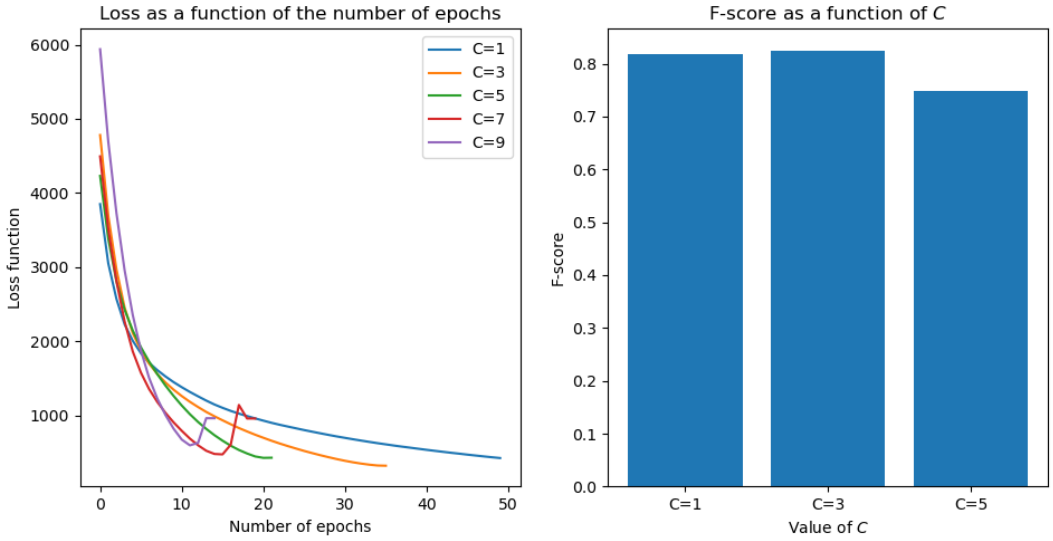
The learning rate (**lr**) and the number of epochs (**n_epochs**) are hyperparameters related to the Stochastic Gradient Descent. Those need to be set such that the Optimization Algorithm (here SGD) converges. In our case, we consider two convergence criteria which are : if the maximum number of epochs is reached (n_epochs) or if the loss function doesn't change much (relative variation is less than a threshold).

The learning rate corresponds to the speed at which our algorithm is converging as it corresponds to the rate at which we update the hyperplane parameters (**w** and **b**). The smaller the learning rate, the more epochs it will need. However, considering a two big learning rate leads to instability of the solution.

The regularization weight **C** corresponds to the importance of large margins for the classifier. The larger the margin the smoother our algorithm will be. Thus we want to consider a C as big as possible. However, the value of **C** can't be found without tuning as it is highly dependent on our data.

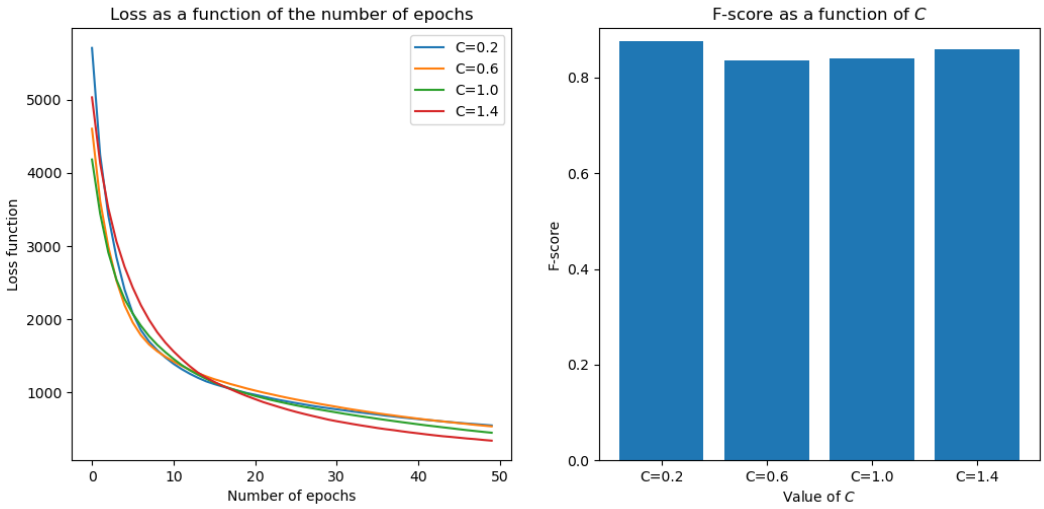
We plotted the F1-score and the evolution of the loss function for some values of **C** in a wide (between 1 and 9) and a narrow range of C (between .2 and 1.4).

Wide range



Note: some values of C can't give a F1-score as their recall and precision are equal to 0.

Narrow range



We can see from these plots that all the algorithms have converged by looking at the loss function graph (on the right). The best value of C appears to be 0.2 in our case (giving the lowest F1-score).

4. Feature importance

a. Feature importance can be ranked by using the value of the w vector for each feature. Show the 10 most and 10 least important features.

The features are sorted by the absolute value of their weights in the **w** vector. Some descriptive statistics are calculated on them:

- *count:* corresponds to the number of values
- *unique:* corresponds to the number of unique values (here {0, 1})
- *top:* value that appears the most
- *freq:* number of occurence of the top value

Most important features: (most to least)

	count	unique	top	freq
activity::com.waluu.android.waluu62.HomeListAct...	2717	2	0	2716
activity::MainActivity	2717	2	0	2693
url::www.metago.net	2717	2	0	2716
url::http://farm1.static.flickr.com/69/16044168...	2717	2	0	2716
activity::.SongsList	2717	2	0	2716
url::http://www.kannadamatrimony.com/Occupation...	2717	2	0	2716
url::http://photo-qh.lifestream.aim.com/	2717	2	0	2716
url::http://xml.apache.org/xerces-2j	2717	2	0	2716
intent::com.lemonlimetime.generic.intent.EXTERN...	2717	2	0	2716
activity::.ui.ViewersActivity	2717	2	0	2716

Least import features: (most to least)

	count	unique	top	freq
url::http://www.apps-builder.com/images/default...	2717	2	0	2713
url::http://m.weather.naver.com/m/crntWetr.nhn?...	2717	2	0	2716
activity::com.subsplash.thechurchapp.BibleChapt...	2717	2	0	2716
activity::.CpuInfoActivity	2717	2	0	2716
service_receiver::.service.Service	2717	2	0	2716
activity::.ProgressBarActivity	2717	2	0	2713
activity::.TypeNGo	2717	2	0	2716
service_receiver::com.banana.spycameralite.Refe...	2717	2	0	2716
url::mtell.thechanner.com	2717	2	0	2716
url::usatvguideservernew.appspot.com	2717	2	0	2716

b. Do you notice any trends or characteristics in the most important features?

One trend that can be gleaned from the more important features is the fact that 50% of these features are URL's, alluding to the fact that malware classification relies heavily on any URL that certain programs might be communicating with. Also, we found it interesting that "SongsList" was included as one of the top classifying features for this dataset and even more surprising that it was found in one of the malware samples. Not surprisingly, it seemed that the "MainActivity" was found in many of the malware samples that were recorded and therefore placed it second on the list of most important features.

c. Would it be good to normalize the features?

In the case of this study, all the features are categorical ({0,1} features). Thus, normalization (i.e min-max mapping) is useless as the data won't change at all. We also thought of using other methods such as PCA to reduce the number of features and increase the speed of the classifier. However, SVM is a good classifier when there are a lot of features and using PCA makes sense when the data set is composed of numerical variables, which is not our case.

5. SVM

a. SVMs can be non-linearized with the use of kernel functions. Speculate on the possible advantages and disadvantages of using non-linear kernel functions, such as the radial basis function or polynomials, on this dataset.

Using kernels makes almost no difference in terms of the execution time due to the kernel trick. However, there are more hyperparameters to tune which lead to higher computation time spent on tuning for the polynomial and radial kernel versions of SVM.
Radial kernel performs as a weighted kNN: which may not be good for very high-dimensional data as it may have the sparsity problem of KNN in high dimensions. Polynomial kernels may perform better than linear SVM but it would increase the computation time because of the tuning.

b. Use the sklearn implementations for LinearSVC and SVC to compare the real performance of a linear SVM and SVM with the 'rbf' kernel. Use a smaller data set.

For this part, the dataset was composed of 4000 training samples (the malware families being splitted respectfully to their initial ratio from the 120000 samples). 3200 benign samples and 800 malware samples. We made a binary classification between benign and malware samples.

Performance of the SVM Linear classifier (sklearn)

Confusion matrix:

	Benign	**Malware**
Benign	0.778783	0.010691
Malware	0.027961	0.182566

F1-Score: 0.9043
Accuracy: 96.13 %
Execution time: 9.65 s

Performance of the SVM Radial classifier (sklearn)

Confusion matrix:

	Benign	**Malware**
Benign	0.783717	0.005757
Malware	0.042763	0.167763

F1-Score: 0.8737
Accuracy: 95.15 %
Execution time: 33.85 s

Comparison:

First, in terms of execution time we can observe that the Linear SVM classifier is 3 times faster than the Radial version (Machine properties: i7-1165G7, RAM 16Go).
Concerning the performances (accuracy and F1-score) Linear SVM is better. It is due to the fact that we could use the best **C** hyperparameter that we found during the tuning of our SVM implementation.

However, Radial SVM is performing well without tuning, so tuning it could lead to a big increase of its accuracy.

In the end, the performances of our SVM implementation with sklearn.SVM are different because the optimization algorithm may not be the same.

c. What does it mean if the magnitude of the y prediction value for a sample is < 1? Hint: there is a particular term used to describe these data points in a SVM.

If the magnitude of the prediction value is less than 1, it means that the sample is between the soft margins of the SVM classifier. These points have a non-zero hinge loss and participate in the loss function. As they participate in the loss, they are called **support vectors**.

d. What feature about the loss function makes these samples particularly important?

The weight of the regularization **C** (which is a hyperparameter of the SVM model) makes these samples particularly important. Those samples participate in the loss function via the Hinge-Loss, the smaller **C** the more these samples influence the classifier.

6. Bonus

a. Implement both One-vs-One and One-vs-All classification.

See 2a.

b. How does the performance between One-vs-One and One-vs-All classification compare based on your implementation?

The number of SVM classification made with the One-vs-One classifier is way bigger than the One-vs-All classifier but the number of samples to compare are smaller. Therefore, in terms of execution time performance, the One-vs-One classifier may be faster as the computation time depends on the number of samples. All other hyperparameters being the same, both classifiers produced roughly 68% accuracy.

c. Speculate as to why one may work better than the other on this problem.

The families of malwares are linked with each other and some families can be hierarchically grouped. Thus, in this specific case, it makes more sense to compare the families one by one (with the one-vs-one classifier). It will help the identification of specific differences between malware families that are really similar whereas the one-vs-all family will just make an aggregation of all the other families and we will then lose the specific information. One-vs-one classification also performs much faster since training is done on a small subset of the train_set instead of the entire set.