Fraud detection using federated learning

Are you ready?

The adventure of fighting frauds

Project General Idea

we're building a fraud detection system we're building a digital bank that uses this system as a way to fight against fraud.

We implement a federated version of our fraud detection model.

Fraud transactions are growing year after anther

There are different against fraud:

- complex authentication
- Get smarter clients
- Machine learning

But why is building a fraud detection system using machine learning is hard?

Most fraud datasets are highly imblaced which cause a lot of problems

Most machine learning models is designed to deal with balanced dataset

Obtaining new fraud samples is extremely hard

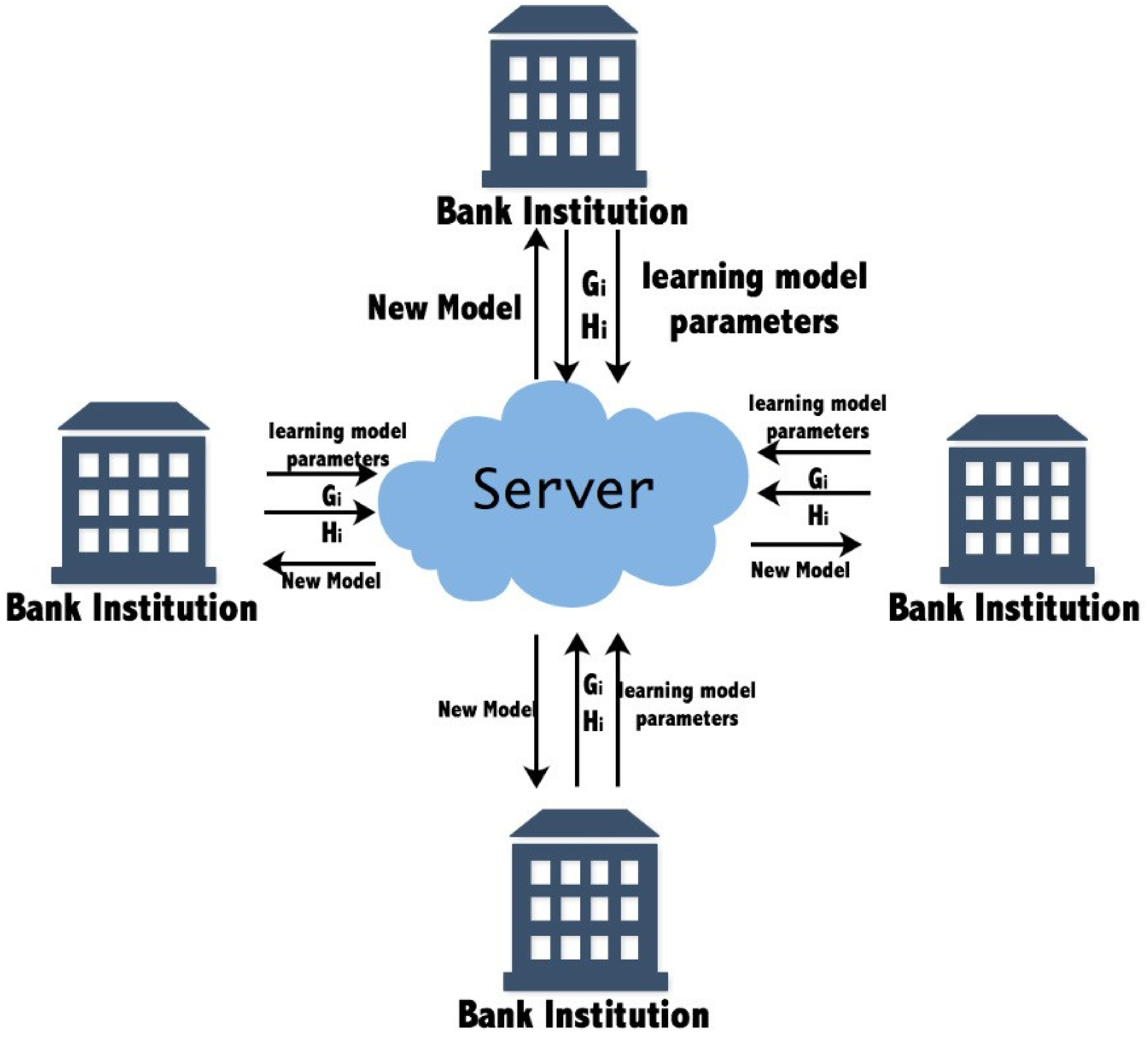
Fraud transactions love to look like non-fraud

Federated Learning

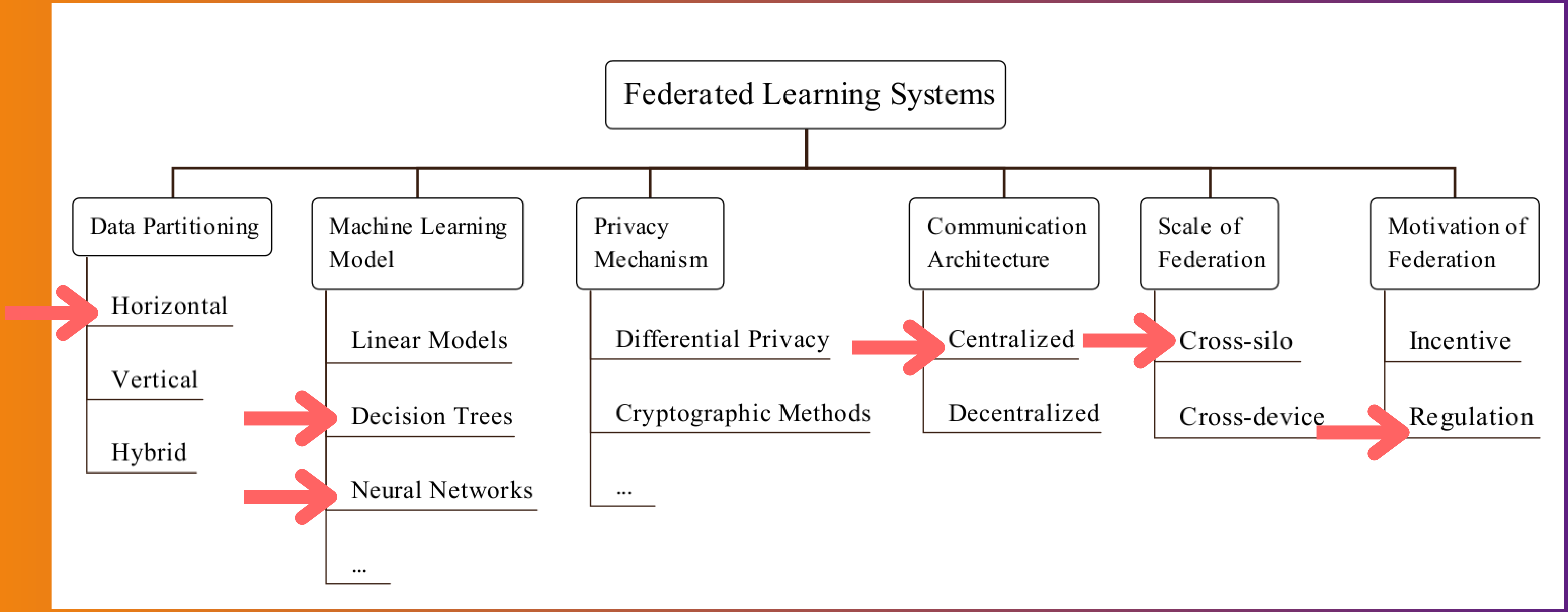
Federated learning is a machine learning setting where clients, collaboratively train a model under the orchestration of a central server.

This training is done without sharing any data. Instead, only the updates are shared with the central server.

Our federated setup is horizontal and Cross-Silo federated learning



Where we are on the federated learning map



Stages of our adaventure

1. Starting fancy: heavy computational power
2. Going for computational-power-friendly solutions.
3. Fed Average statge.
4. Feederating boosting
5. building federated imbalanced Adaboost.

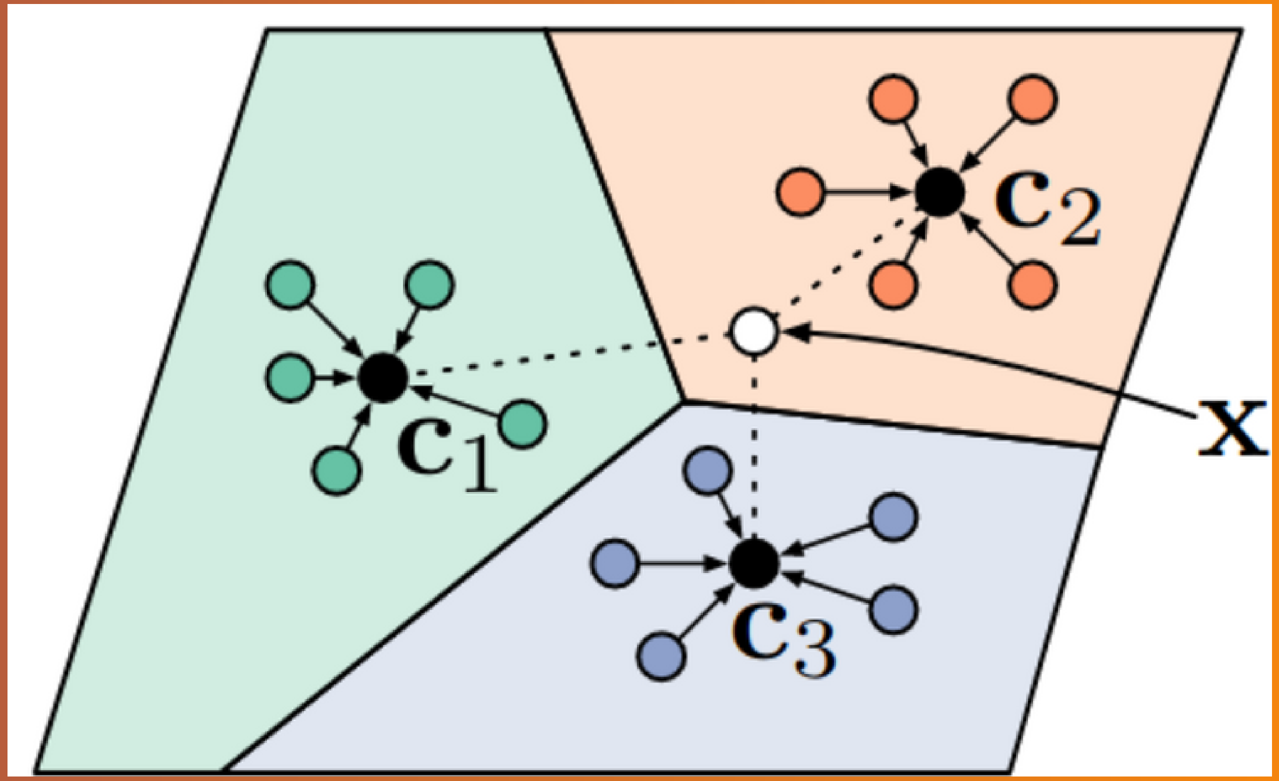
1 Starting fancy: heavy computational power

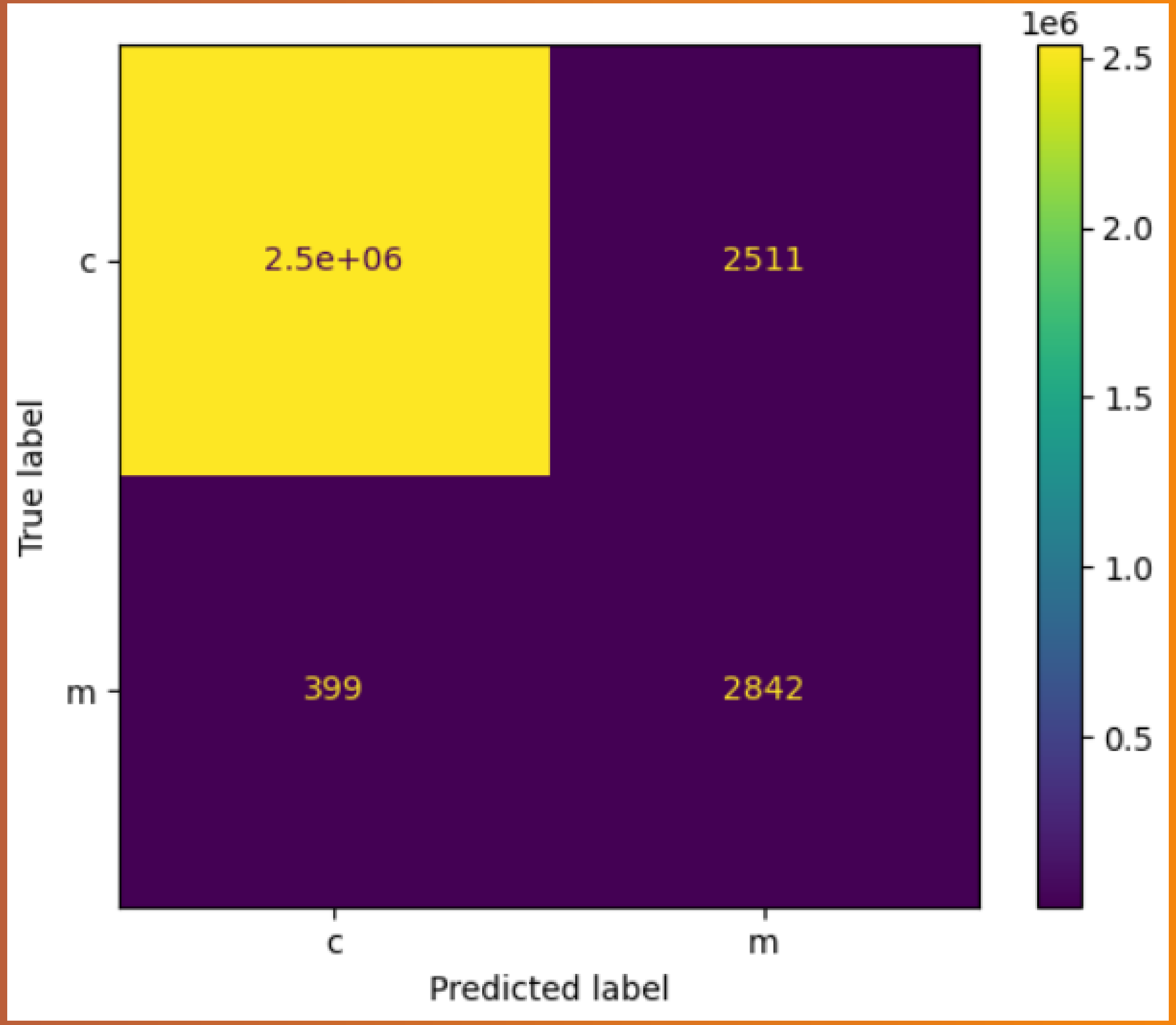
The network architecture: ResNet18 + Prototypical Networks.

It works in 2 steps:

- extract the class prototype, and this step in our case is done using the pre-trained ResNet18, it tries to find a good feature space.
- classifying the query images. to do so, we compute the distance between each unlabelled image and the prototypes using the Euclidean distance.

F1 score = 66%, ROC AUC = 93.7%, average precision = 46.6%, Precision = 53% and Recall = 87.7%

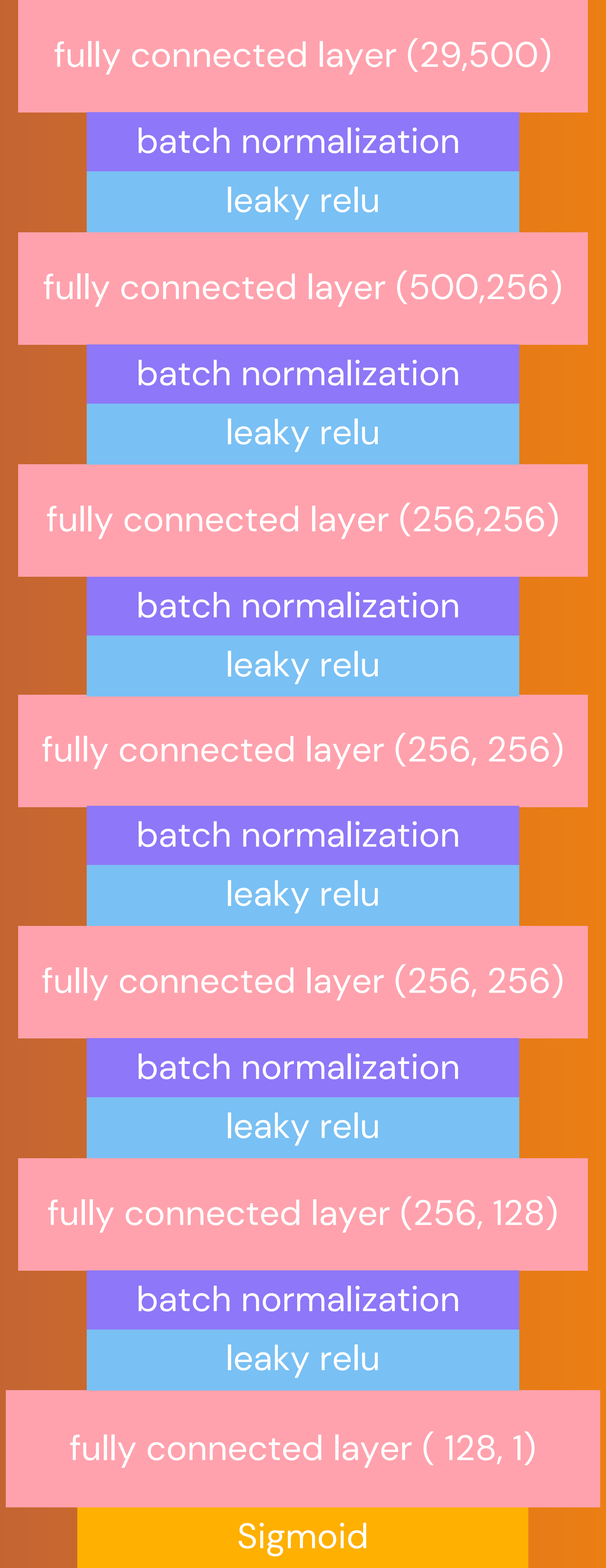




2 Going for computational-power-friendly solutions.

The new approach:

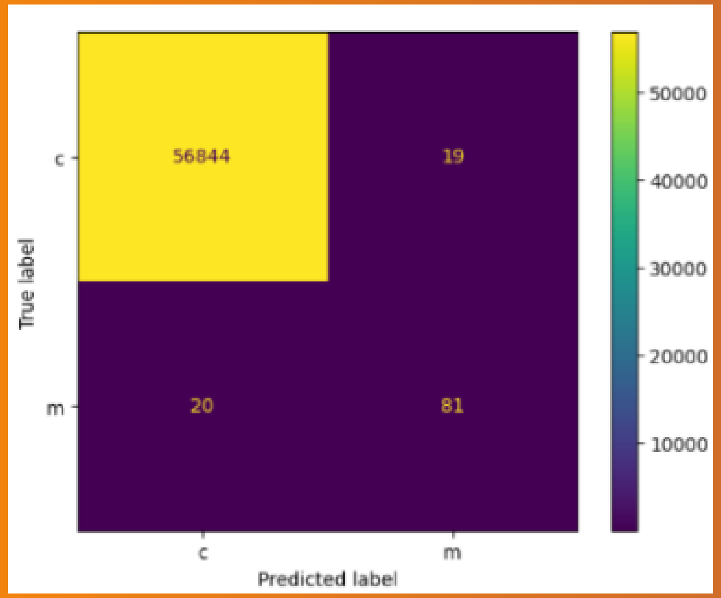
- Feed-forward neural network + different loss function.
- New popular dataset: European credit card, 492 frauds out of 284,807 transactions.
- Different loss functions.


fully connected layer (29,500)
batch normalization
leaky relu
fully connected layer (500,256)
batch normalization
leaky relu
fully connected layer (256,256)
batch normalization
leaky relu
fully connected layer (256, 256)
batch normalization
leaky relu
fully connected layer (256, 256)
batch normalization
leaky relu
fully connected layer (256, 128)
batch normalization
leaky relu
fully connected layer (128, 1)
Sigmoid

Results of stage 2

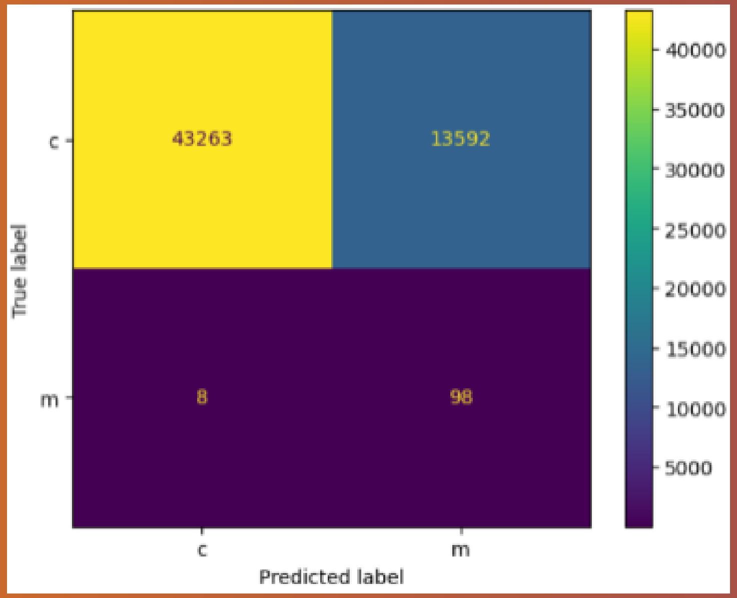
Weighted Binary Cross Entropy with Logits

F1 score = 80%, ROC AUC = 90%, average precision = 66%, Precision = 80% and Recall = 80%



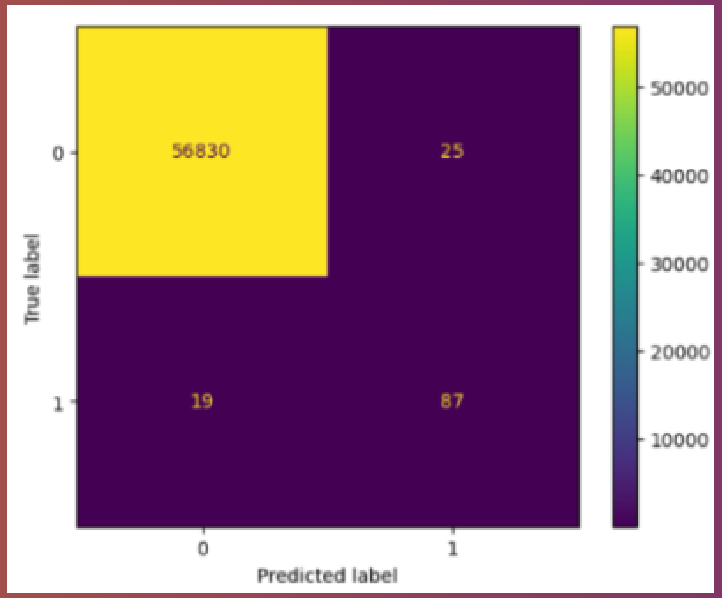
Roc Star: originally old not working in practice loss, but it was modified to be a differentiated loss function

F1 score = 1.4%, ROC AUC = 84%, average precision = .6%, Precision = .71% and Recall = 92%.



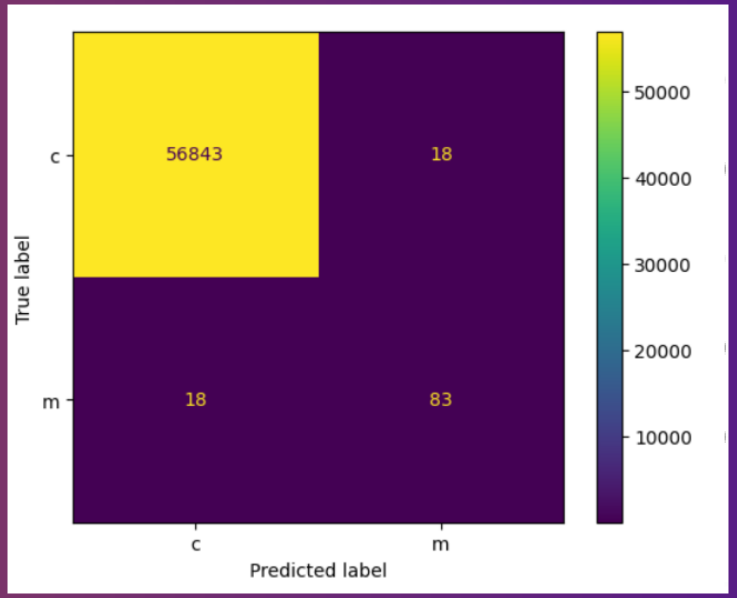
Label-Distribution-Aware Margin Loss

F1 score = 79%, ROC AUC = 91%, average precision = 63%, Precision = 77.7% and Recall = 82%



Focal loss function

F1 score = 82%, ROC AUC = 91%, average precision =67%, Precision = 82% and Recall = 82%



Focal Loss

A better alternative for Cross-Entropy

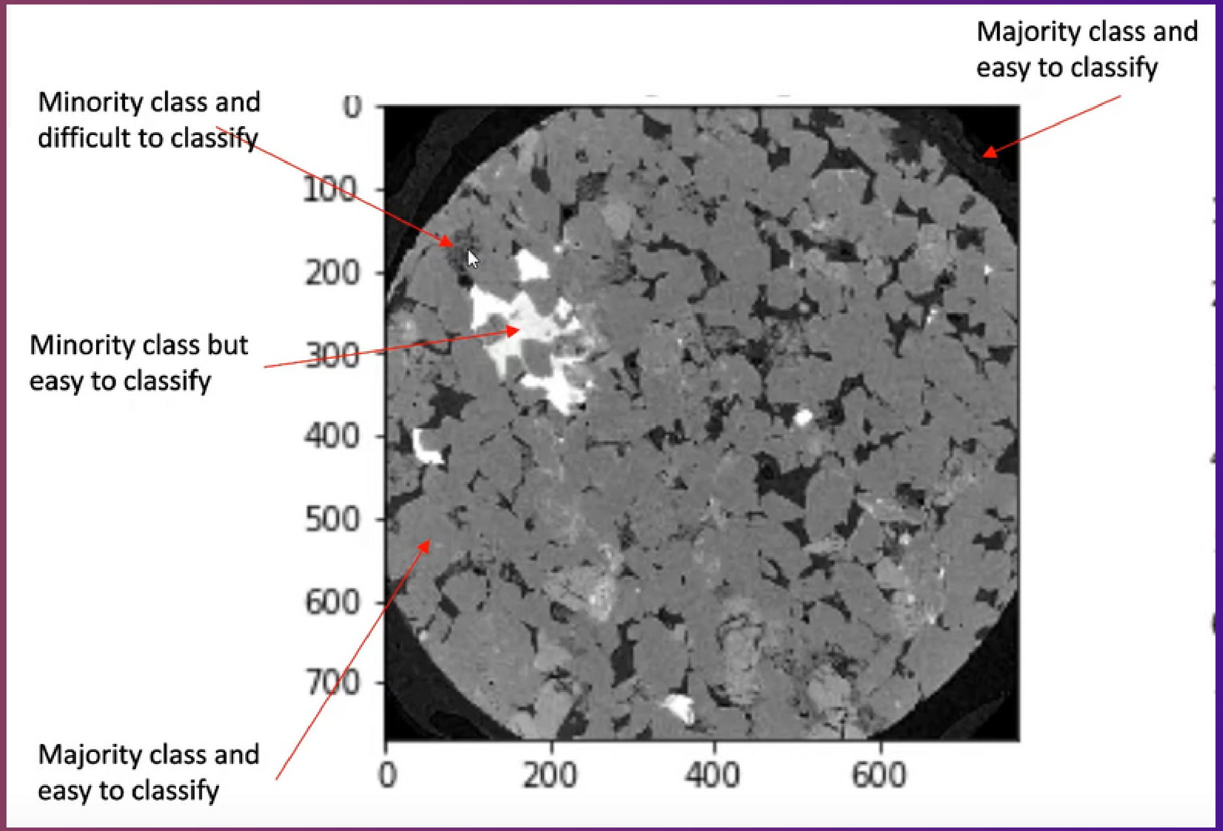
orginally devolped to deal with object detection prblem

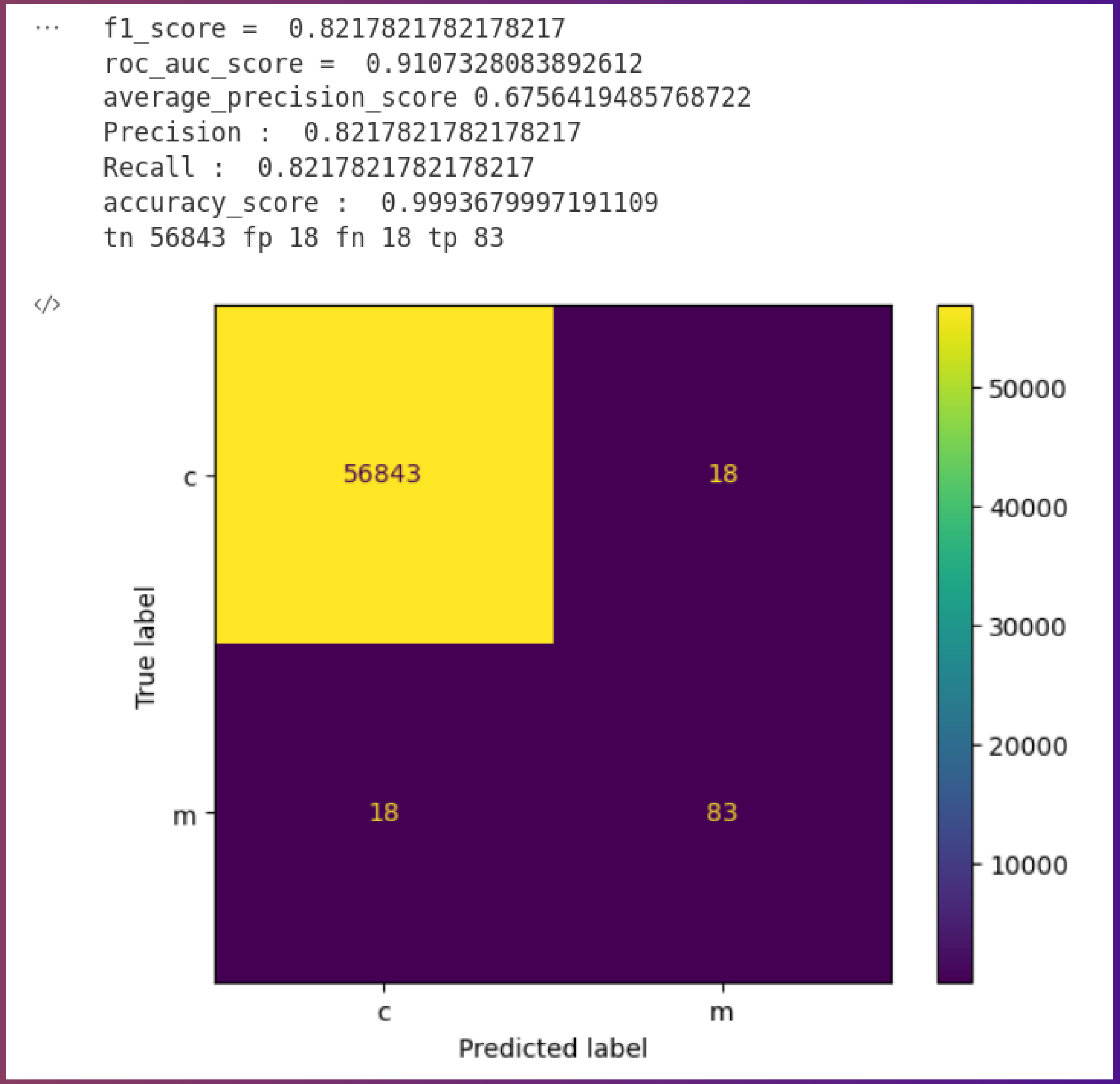
Focal loss focuses on the examples that the model gets wrong rather than the ones that it can confidently predict, ensuring that predictions on hard examples improve over time rather than becoming overly confident with easy ones.

Down Weighting. Down weighting is a technique that reduces the influence of easy examples on the loss function

$$\mathrm{CE}(p_t) = -\log(p_t)$$

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$


Majority class and easy to classify
Minority class and difficult to classify
Minority class but easy to classify
Majority class and easy to classify
0
100
200
300
400
500
600
700
0
200
400
600



```
f1_score =  0.8217821782178217
roc_auc_score =  0.9107328083892612
average_precision_score 0.6756419485768722
Precision :  0.8217821782178217
Recall :  0.8217821782178217
accuracy_score :  0.9993679997191109
tn 56843 fp 18 fn 18 tp 83
```

Compression with related work

Model	F1	Recall	Precision	ROC_AUC	avg_prec
Deep anomaly detection with deviation networks	80%	81%	79%	90%	67%
An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine	56%	40%	97%	90%	-
Enhanced credit card fraud detection based on attention mechanism and LSTM deep model	.48%	99%	.25%	61%	.24%
Federated Learning Used to Detect Credit Card Fraud : MASTER'S THESIS for Madeleine Jansson et all	82%	%77	%89	-	%88
Our model = feed forward neural network+ focal loss	82%	%82	%82	91%	%67

3

Fed Average

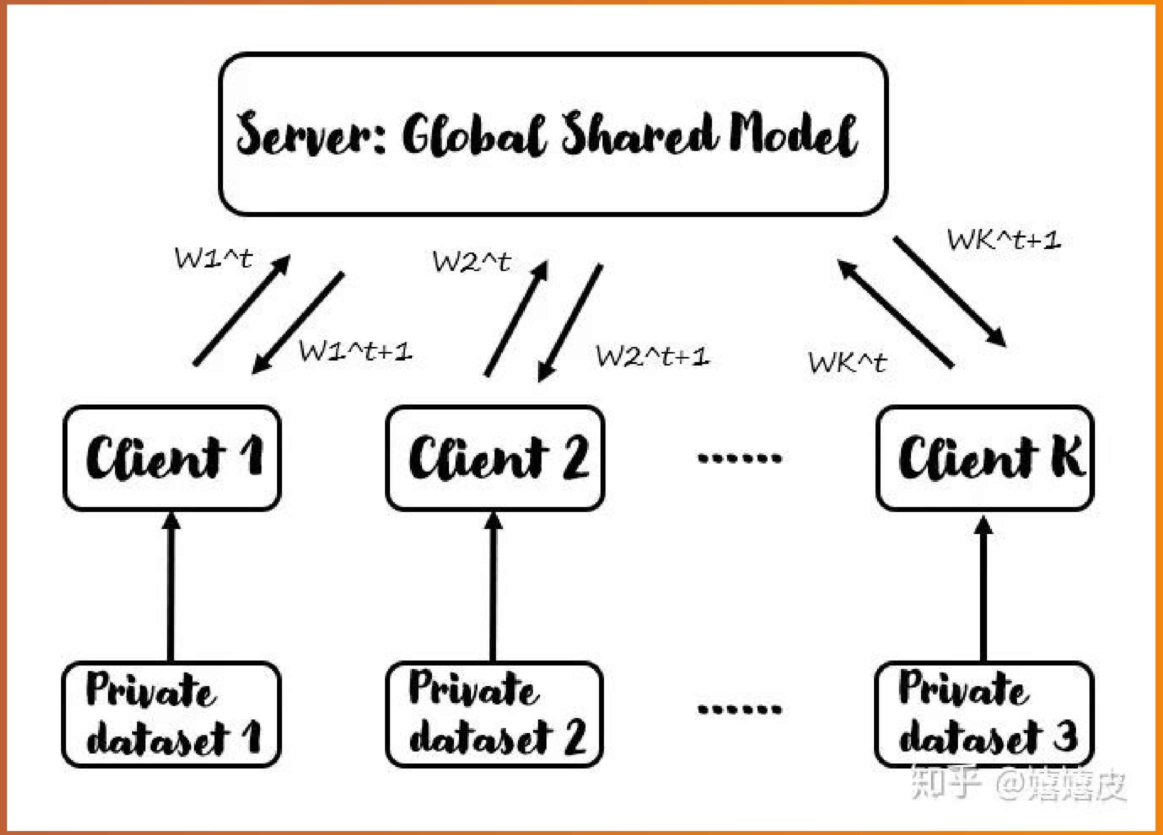
The most famouse aggregator

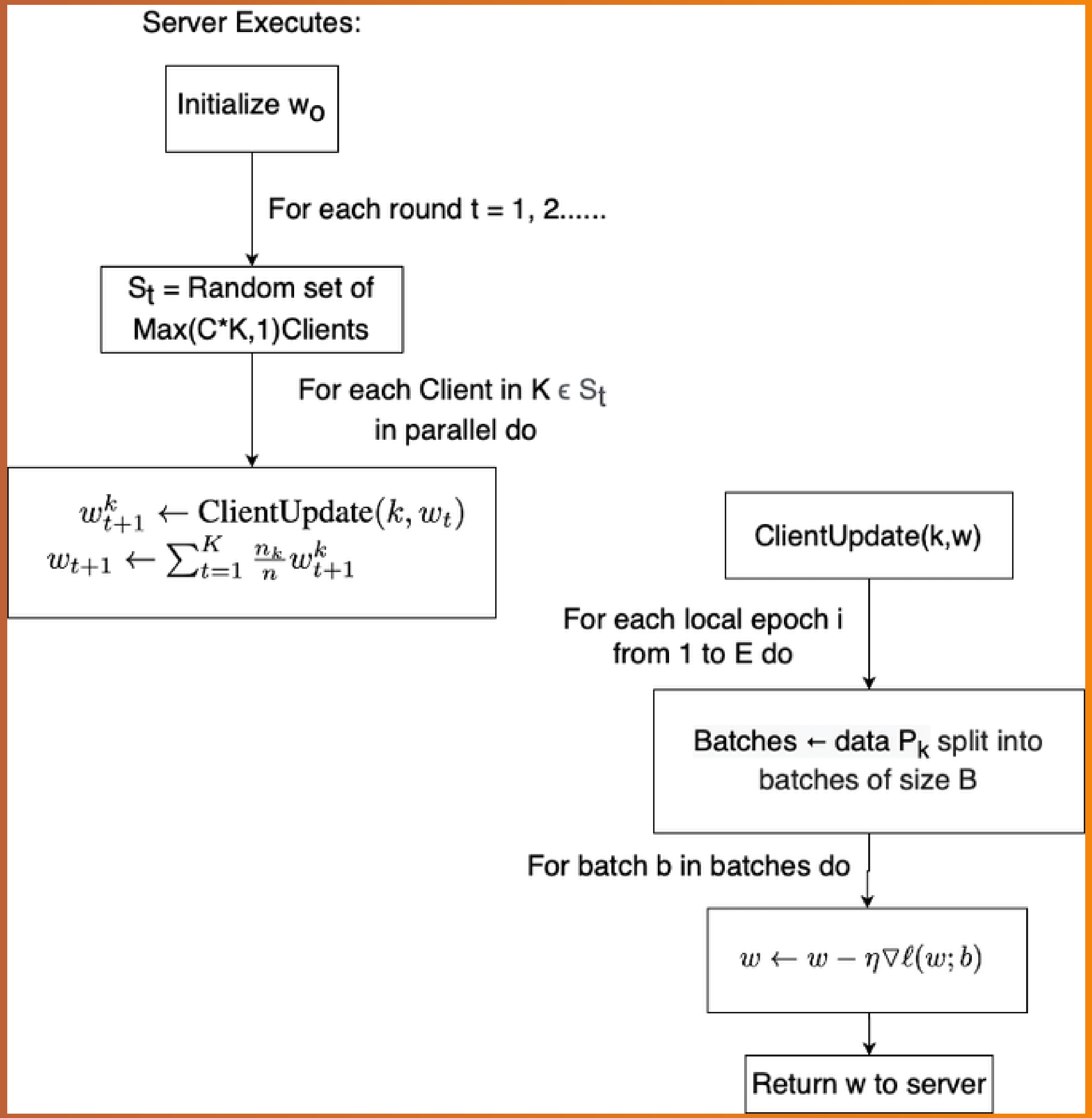
Introduced in 2016, and it considered the first aggregator that was used in federated learning

It has the Simplest algorithm, just take the mean of the weights of the different layers of the network.

It works mainly for deep learning methods: CNN and feed-forward neural networks for example.

Using this method we managed to get a federated model with the following performance: F1 score = 80%, ROC AUC = 89%, average precision =66%, Precision = 79% and Recall = 81%




Server Executes:
Initialize w_0
For each round t = 1, 2......
S_t = Random set of Max(C*K,1)Clients
For each Client in K ϵ S_t in parallel do
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{t=1}^{K} \frac{n_k}{n} w_{t+1}^k$
ClientUpdate(k,w)
For each local epoch i from 1 to E do
Batches ← data P_k split into batches of size B
For batch b in batches do
$w \leftarrow w - \eta \nabla \ell(w; b)$
Return w to server

4

Going into a new adventure

Federated Boosting

One of the most under-explored topics in the federated learning world is the federation of non-gradient descent models.

The paper that we base our final piece of work on is called: "Boosting the Federation: Cross-Silo Federated Learning without Gradient Descent", it was introduced in late 2022.

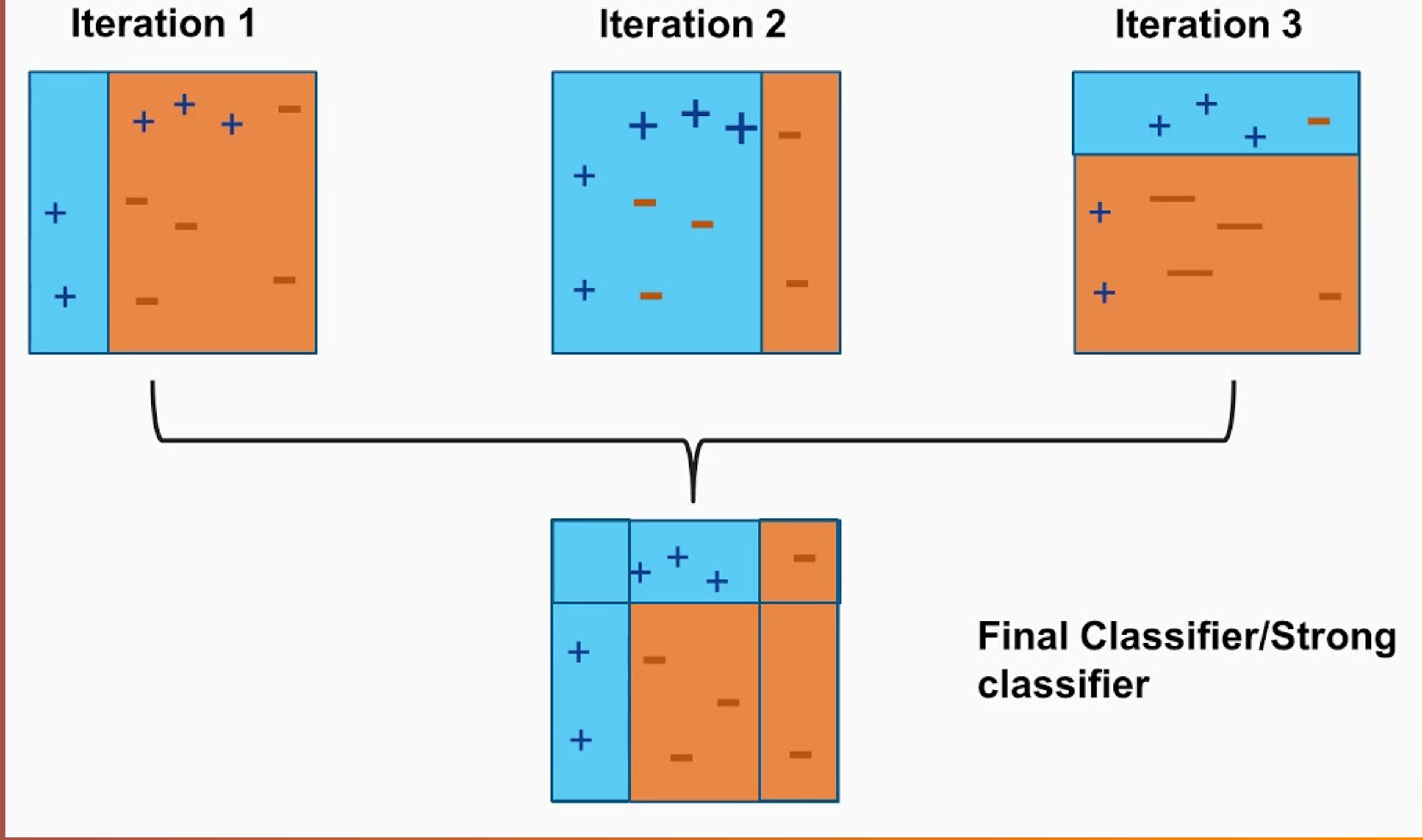
AdaBoost

What and Why ?

Appears particularly interesting as a candidate tool for FL, as it effectively combines classifiers which may be learned independently by the FL clients.

One of the first boosting ensemble methods

Already there's a distributed version of it.



DistBoost.F

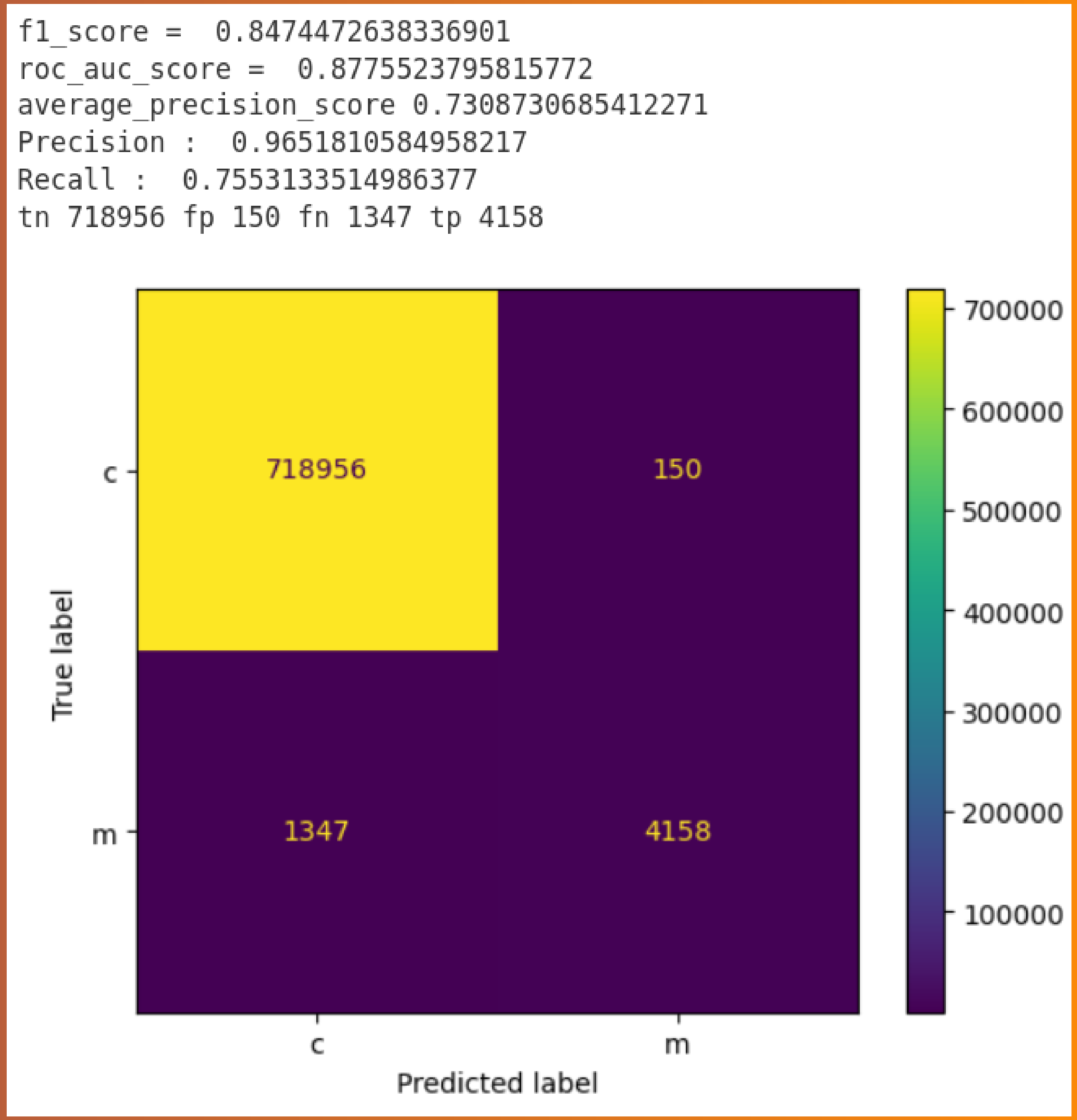
Based on an distributed version of adaboost

The data is partitioned across the clients, and each of them stores a local weight distribution.

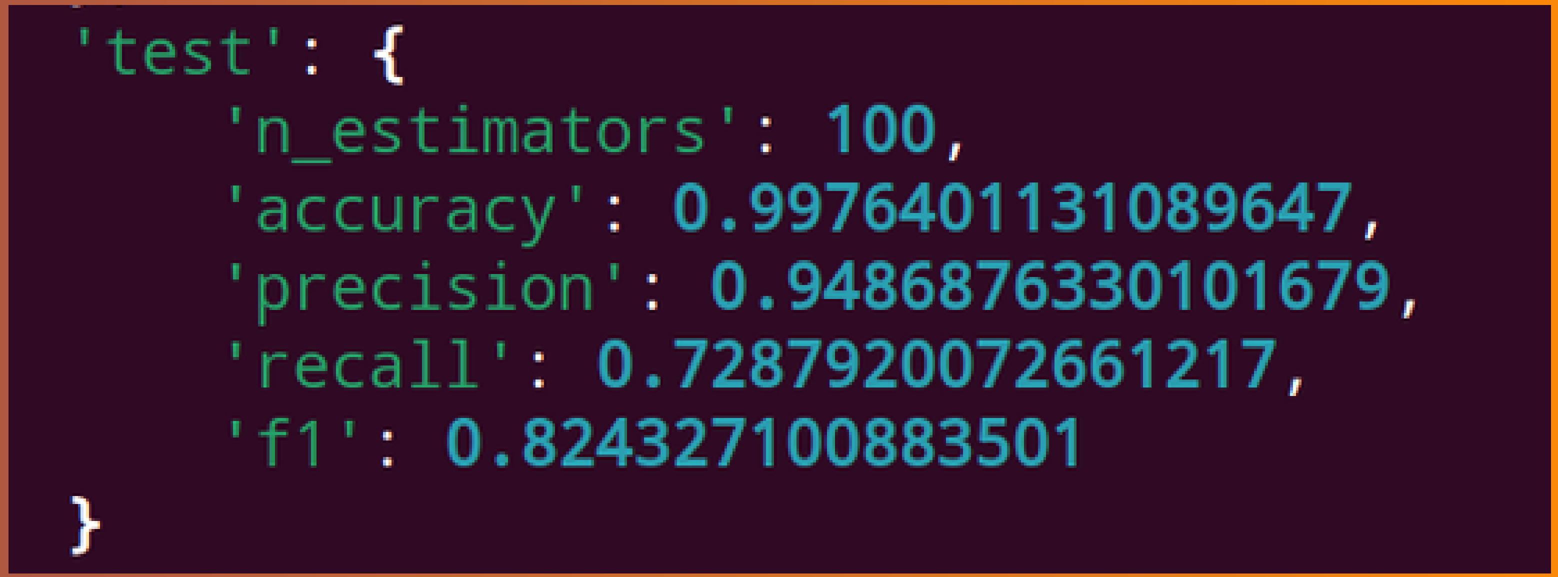
The core idea of DistBoost is that the union of the local distributions represents a good approximation of the centralized AdaBoost's example distribution.

At each iteration, a new weak hypothesis is learned from each client. Unlike AdaBoost, the "global" weak hypothesis is the committee of all the weak classifiers locally trained by the clients, Then, weights are updated and a new iteration begins, To make this last step possible, each client must communicate to the other clients the errors the weak global hypothesis commits to its own local data.

To simplify and mimic the centralized FL setting, we can assume a server acts as an intermediary. The final classifier is the weighted sum of the (global) weak classifiers



```
f1_score =  0.8474472638336901
roc_auc_score =  0.8775523795815772
average_precision_score 0.7308730685412271
Precision :  0.9651810584958217
Recall :  0.7553133514986377
tn 718956 fp 150 fn 1347 tp 4158
```

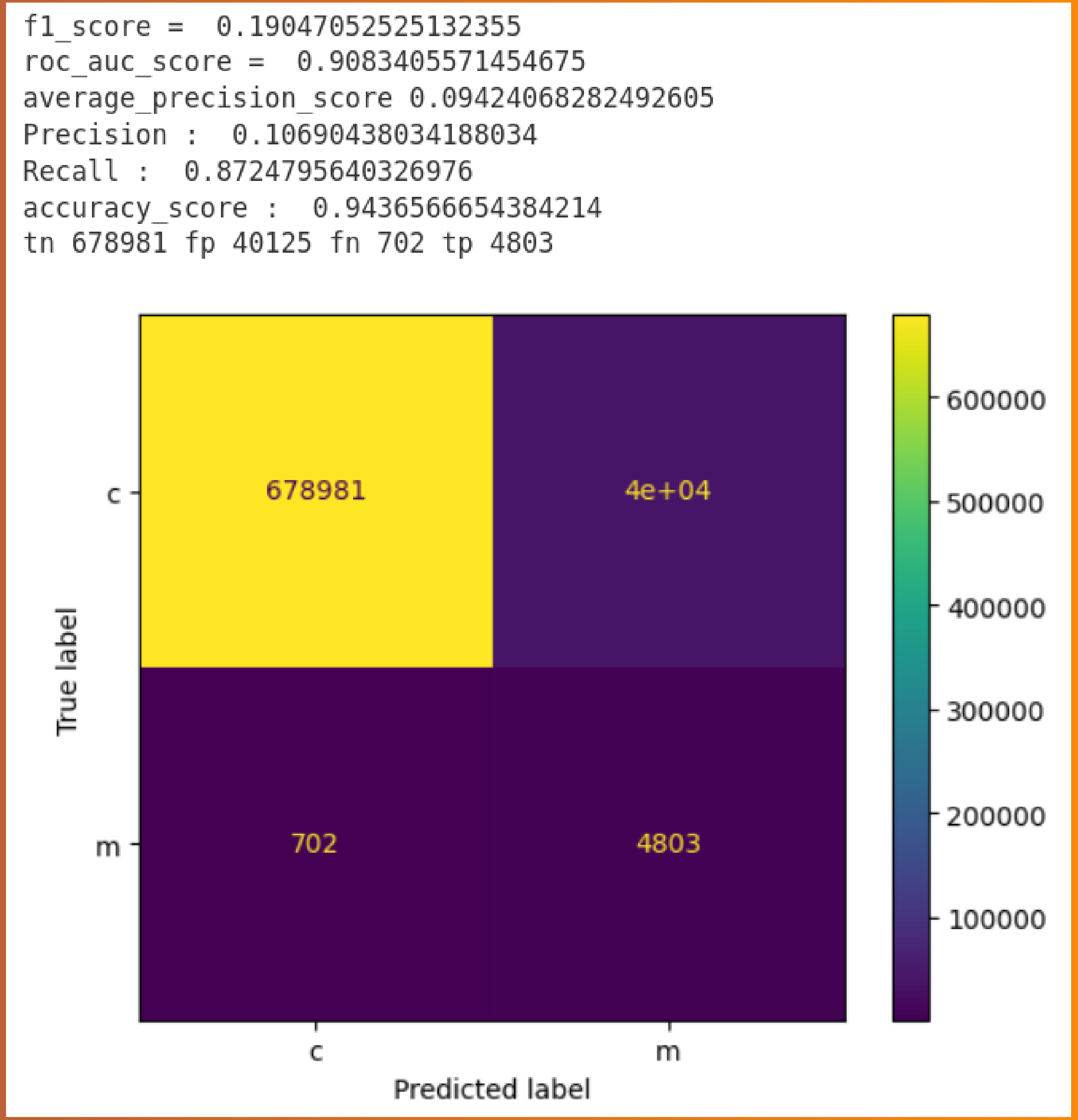


```
'test': {
    'n_estimators': 100,
    'accuracy': 0.9976401131089647,
    'precision': 0.9486876330101679,
    'recall': 0.7287920072661217,
    'f1': 0.824327100883501
}
```

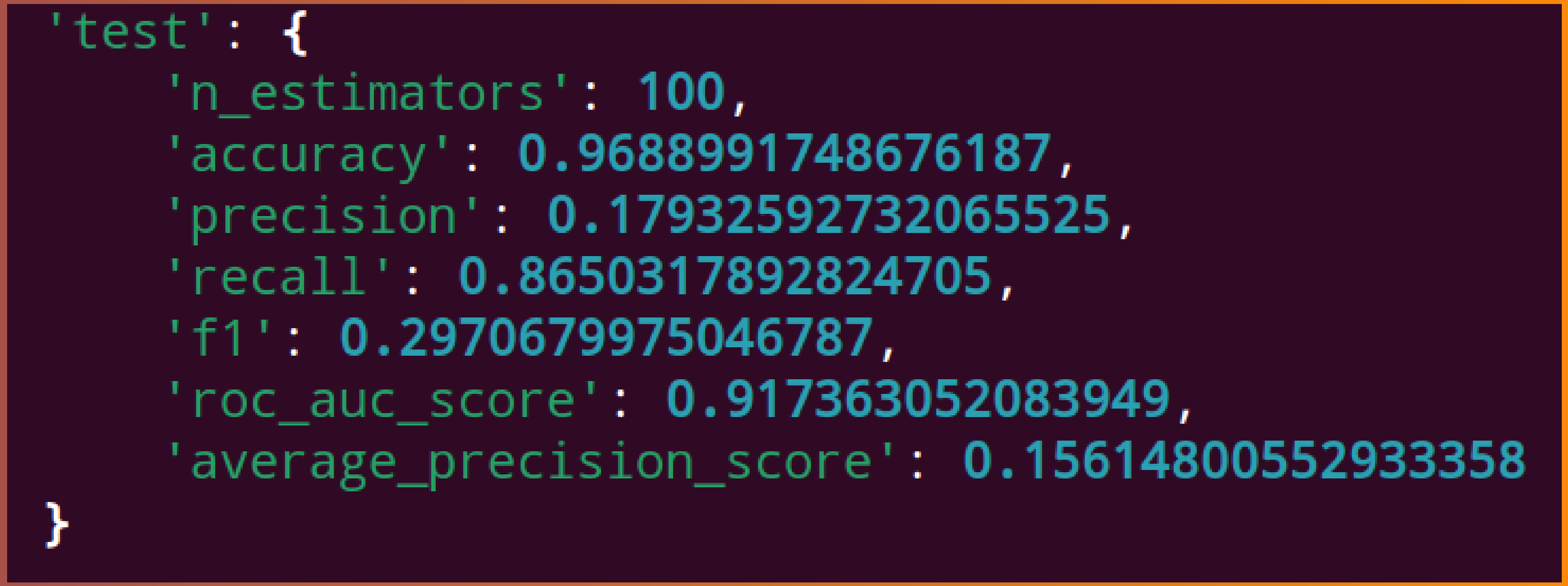
DistBoost.F for Imbalanced dataset

The idea was simple, just re-define the weak-learner of Adaboost.

We used a weighted decision tree as a weak learner



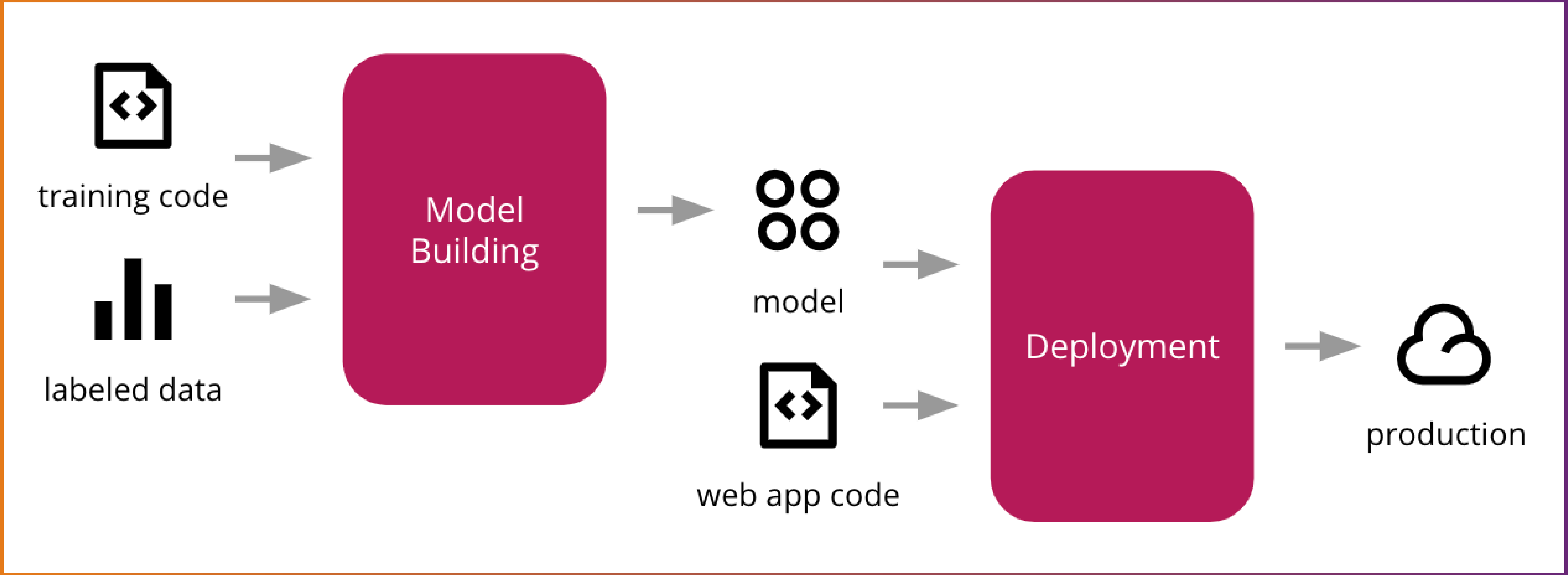
```
f1_score =  0.19047052525132355
roc_auc_score =  0.9083405571454675
average_precision_score 0.09424068282492605
Precision :  0.10690438034188034
Recall :  0.8724795640326976
accuracy_score :  0.9436566654384214
tn 678981 fp 40125 fn 702 tp 4803
```



```
'test': {
    'n_estimators': 100,
    'accuracy': 0.9688991748676187,
    'precision': 0.17932592732065525,
    'recall': 0.8650317892824705,
    'f1': 0.2970679975046787,
    'roc_auc_score': 0.917363052083949,
    'average_precision_score': 0.15614800552933358
}
```

ML production

Models of machine learning are highly advantageous however their potential for generating maximum value is constrained without the knowledge of how to deploy them effectively.



Challenging in Deploying machine learning models:

related to the underlying machine-learning techniques and statistical aspects of the model.

This may involve addressing issues such as **model accuracy**, **overfitting**, **selecting appropriate algorithms**, and **fine-tuning hyperparameters**.

Successful deployment of machine learning models requires addressing these challenges to ensure optimal performance and reliable prediction.

Second challenging in Deployment

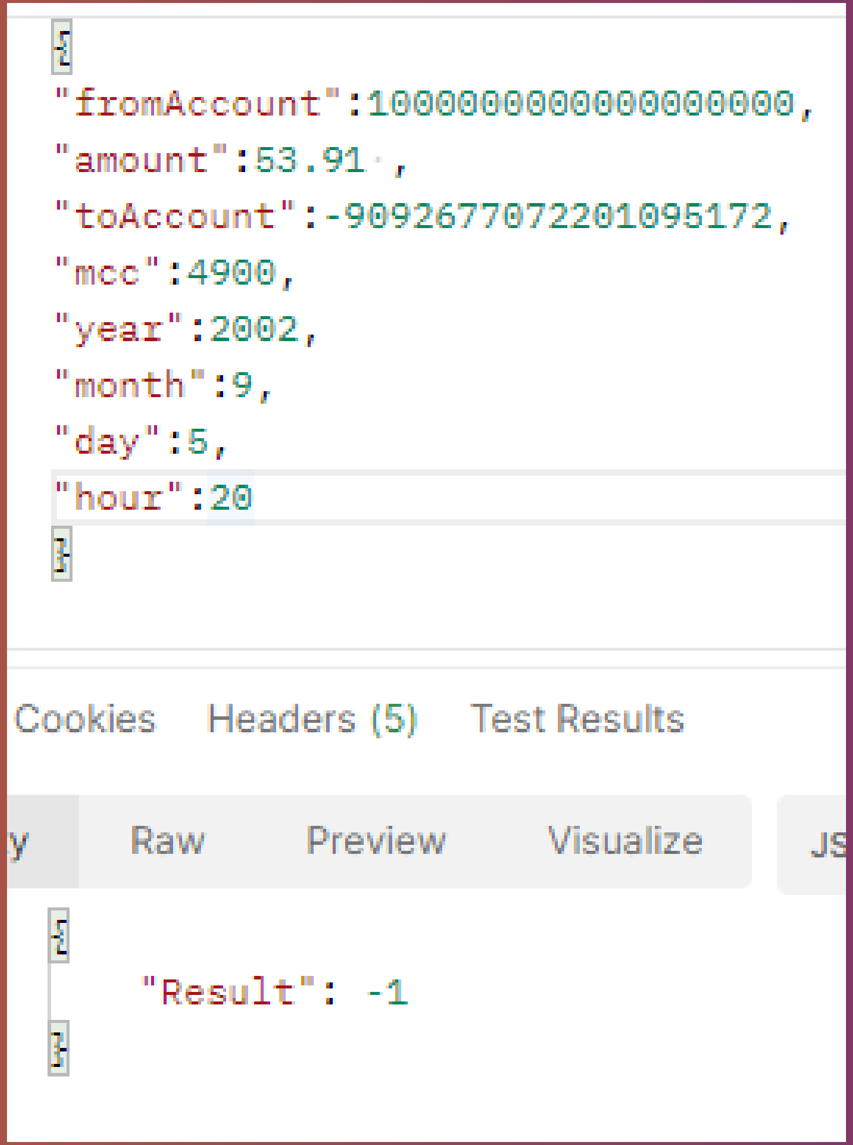
related to software engineering aspects of deployment this involves considerations such as **scalability**, **system integration**, **compatibility with existing software infrastructure** and **ability to handle large volumes of data efficiently**.

Effective deployment requires addressing these software engine issues to ensure a robust and scalable solution.

In Our APP

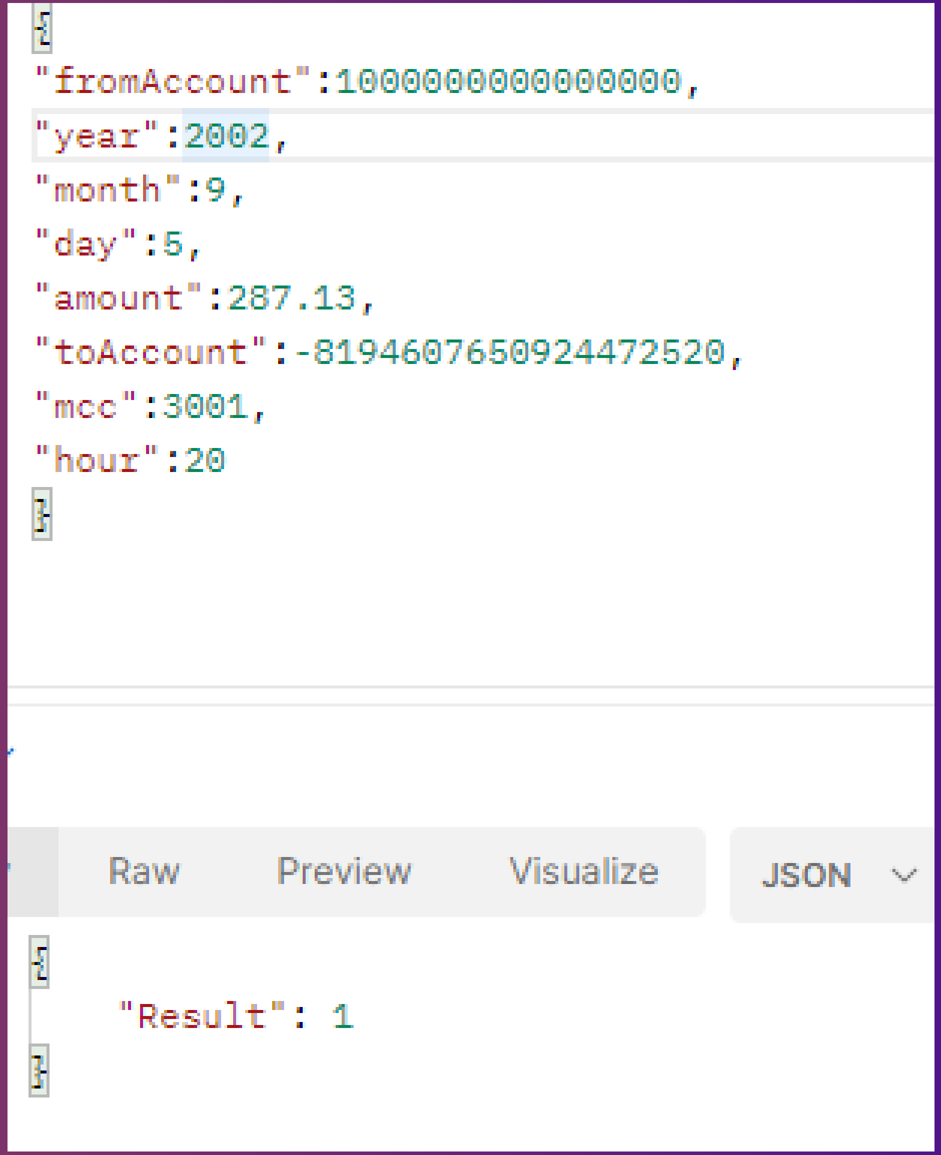
for instance, in our scenario, when user want to done any transaction operation after user entered requested information in transaction page back-end will extract some important information from these informations like(fromAccount - toAccount - year - month - day - hour - mcc).

then send it for model to predict this operation is Fraud (1) or not fraud (0) then resend response for Back-end through API to see if this process will continue if this is non fraud operation or will stop if this fraud operation



```
{
"fromAccount":1000000000000000000,
"amount":53.91 ,
"toAccount":-9092677072201095172,
"mcc":4900,
"year":2002,
"month":9,
"day":5,
"hour":20
}
```

```
{
   "Result": -1
}
```



```
{
"fromAccount":1000000000000000,
"year":2002,
"month":9,
"day":5,
"amount":287.13,
"toAccount":-8194607650924472520,
"mcc":3001,
"hour":20
}
```

```
{
   "Result": 1
}
```

June 18, 20231

Thank you

Have a great day ahead.