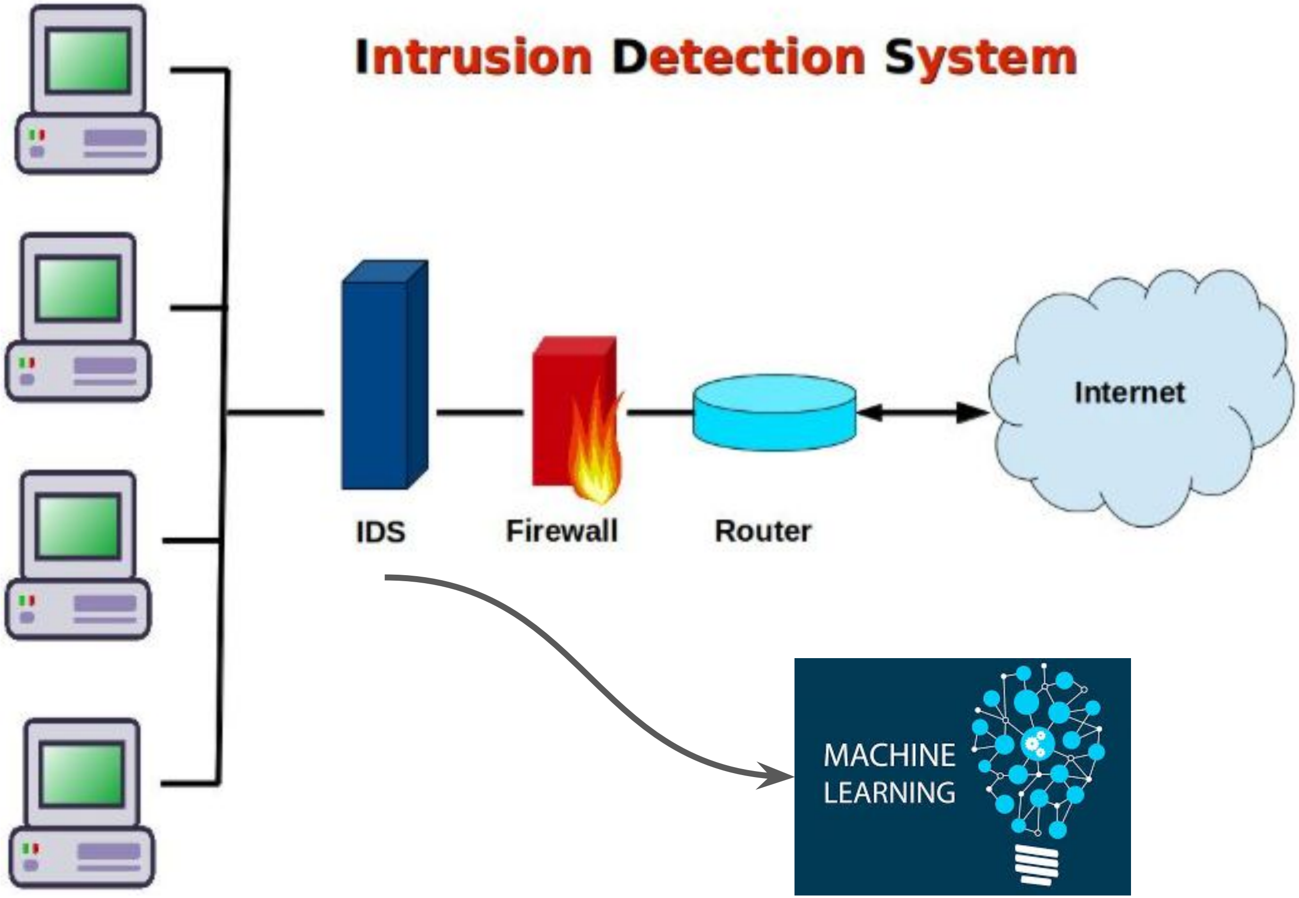
Network Intrusion Detection Evaluation Using Machine Learning Models on Big Data

Manos Chatzimpyrros, csd3137
Dimitris Karathanasis, csd3547
Savvas Kastanakis, csdp1096


Intrusion Detection System
IDS
Firewall
Router
Internet
MACHINE
LEARNING

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

Picking a Dataset

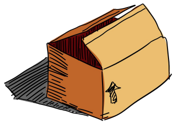
- ❖ **CICIDS2017** dataset contains **benign traffic** and the most up-to-date common **attacks**, which resembles the true real-world data (PCAPs)
  - ➢ The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS.
    - Monday, Normal Activity, 11.0G
    - Tuesday, attacks + Normal Activity, 11G
    - Wednesday, attacks + Normal Activity, 13G
    - Thursday, attacks + Normal Activity, 7.8G
    - Friday, attacks + Normal Activity, 8.3G

UNB
UNIVERSITY OF NEW BRUNSWICK

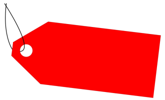
https://www.unb.ca/cic/datasets/ids-2017.html

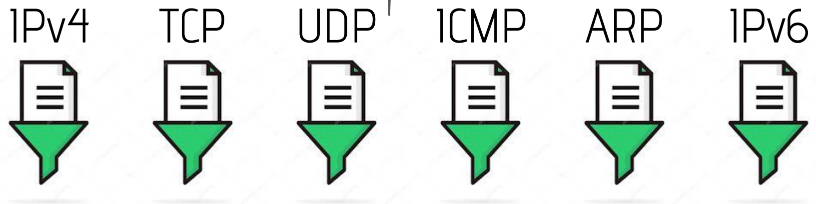
A Data Point

Raw Network Packets (50GB)

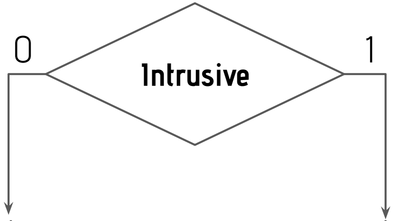


Labels





Default where None



Packet Length	ACK	Checksum	Network Mask	...	Destination Port
655	95	85674478	26	...	80

Label
0 / 1

EC2 Environment Setup

Type	vCPUs	Memory (GiB)	Network Performance	IPv6 Support
t3a.2xlarge	8	32	Up to 5 Gigabit	Yes

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

First Pipeline

Read Dataset

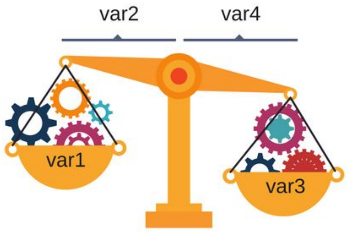
Split to 60/20/20

86.6% — Baseline Accuracy on Validation Set

86.6% — LR Accuracy on Validation Set

Balance the Dataset

- *Accuracy is misleading*
- Many ML algorithms are designed to maximize overall accuracy
- We can confirm this from the previous slide:
  - **LR ignores the minority class in favor of the majority class**



- First thing we can do is to **balance the dataset**
  - **Up-sampling** is the process of randomly duplicating observations from the minority class in order to reinforce its signal
  - There are several heuristics for doing so, but the most common way is to simply resample with replacement

Tuning the Pipeline

Read Dataset

Majority Class Value Counts
1.055.865
Minority Class Value Counts
162.012

Balance Dataset

Class 0 Value Counts
1.055.865
Class 1 Value Counts
1.055.865

Split to 60/20/20

50% Baseline Accuracy on Validation Set

79% LR Accuracy on Validation Set

Worse than before, but more representative result

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

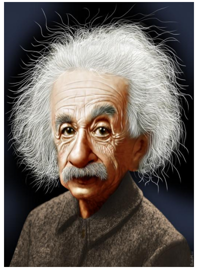
Take-away Message

Switch to ROC AUC

How about measuring the performance of our ML models on the primary imbalanced dataset and on the latter balanced one, using ROC AUC

Train LR on Imbalanced DS	Train LR on Balanced DS
73.3	73.3

If you judge a fish on its ability to climb a tree, it will live its whole life believing it is stupid



ROC AUC behaves the same both for balanced / imbalanced classes and provides a more representative view of the model's performance

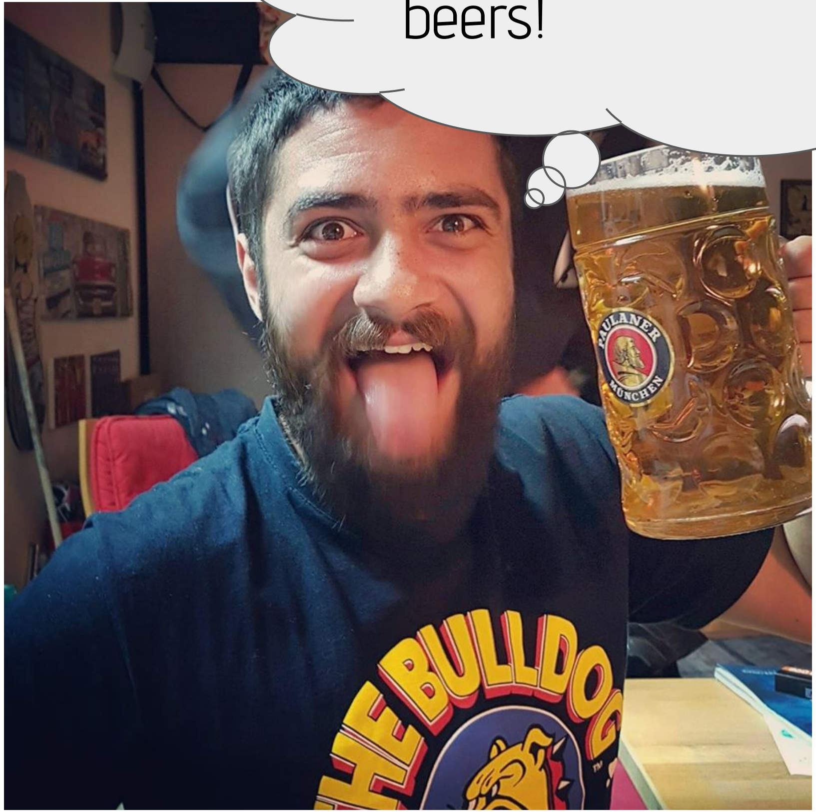
Random Forests For The Win

- One approach we considered is using **tree-based algorithms**
- Decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes
- In modern applied machine learning, tree ensembles (Random Forests, Gradient Boosted Trees, etc.) almost always outperform decision trees, so we jumped right into those

```
clf_4 = RandomForestClassifier().fit(X, y)
prob_y_4 = clf_4.predict_proba(validation_features)
prob_y_4 = [p[1] for p in prob_y_4]
print( roc_auc_score(validation_labels, prob_y_4) )
```

97%

Let's deliver it and go for beers!



I think we are overfitting

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

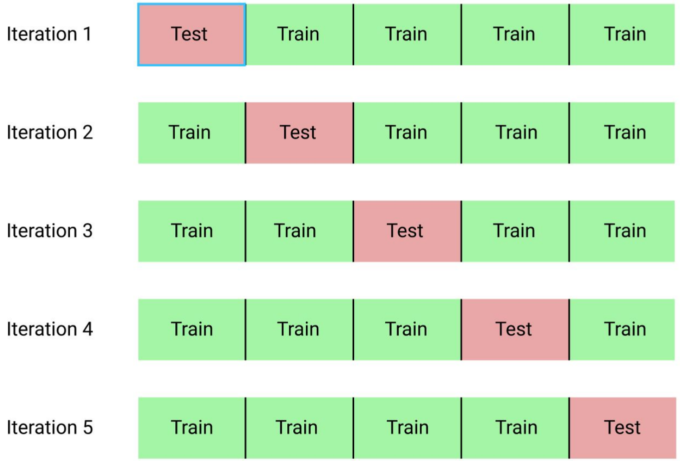
Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

How we prevented overfitting

- **Cross-validation** is a powerful preventative measure against overfitting
- Use your **initial training data** to generate multiple **mini train-test splits**
- Use these splits to tune your model.
- In standard **k-fold cross-validation**, we partition the data into k subsets, called folds
- Then, we iteratively train the algorithm on k-1 folds while using the remaining fold as the test set (called the "**holdout fold**").



How about CV scores on RF and LR

Cross Validated Random Forest on Imbalanced DS	**Cross Validated Random Forest on Balanced DS**	Cross Validated Logistic Regression on Imbalanced DS	Cross Validated Logistic Regression on Balanced DS
64%	**84%**	70%	80%

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

Why Hyperparameter Tuning (HPT)

- During the models' comparison we have applied **Hyperparameter Tuning** on both models (LR, RF)
- We seeked for the **optimal configurations of each model**, such that only the best instances would be compared

- Also combined the two models:
  - **Feature Selection** using RF
  - Drop unimportant Features
  - LR on "important" Dataset
  - ...
  - Didn't work

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

Why using Cost Sensitive Models

We used penalized learning algorithms that **increase the cost of classification mistakes** on the minority class

A popular algorithm for this technique is **Penalized-SVM**

During training, we can use the argument `class_weight='balanced'` to penalize mistakes on the minority class by an amount proportional to how under-represented it is


Who ate my credits?

Running Support Vector Machines (SVM)

Read Dataset

Majority Class Value Counts
1.055.865
Minority Class Value Counts
162.012

Balance Dataset (Downsample)

Class 0 Value Counts
162.012
Class 1 Value Counts
162.012

Split to 60/20/20

Get a 5% Sample → Limitations

50% — ROC AUC on SVM

52% — Cross Validated SVM

Agenda

Data Preprocessing

First Actions

Picking a Performance Metric

How we Handled Overfitting

Hyperparameter Tuning

Cost Sensitive Training Algorithms

Take-away Message

Take Away Messages

- When using Machine Learning Models:
    - **Balancing** a Dataset can reinforce the signal of a minority class
    - Picking a representative **Performance Metric** is crucial
    - Use **Cross Validation** to Prevent Overfitting
    - Best model:
        - **Cross Validated Random Forest on Upsampled Dataset with 100 Decision Trees**
        - 84% Score on Validation Set
        - 84% Score on Test Set

Tools Used

PySpark

scikit learn

Pandas

NumPy

aws

Thank you

Back-Up Slides

LR HPT

Parameters	Values	Action
solver	'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'	Algorithm to use in the optimization problem
penalty	'L1', 'L2'	Used to specify the norm used in the penalization
class_weight	'Balanced', None	Weights associated with classes
random_state	123, None	Pseudo random number generator seed when shuffling the data
multi_class	'ovr', 'multinomial'	If the option chosen is 'ovr', then a binary problem is fit
n_jobs	int, None	Number of CPU cores used

RF HPT

Parameters	Values	Action
n_estimators	int	The number of trees in the forest
criterion	'gini, 'entropy'	The function to measure the quality of a split
bootstrap	boolean	If False, the whole dataset is used to build each tree
class_weight	'Balanced', None	Weights associated with classes
random_state	123, None	Random Seed
max_depth	int, None	Maximum depth of a tree
n_jobs	int, None	Number of CPU cores used

Running Stochastic Gradient Descent (SGD)