Review of Problem Statement and Research Question

Anna Nicolais
CMPE 789

Will supervised or unsupervised feature selection produce a more accurate ML model that can classify benign versus malicious traffic on an IDS dataset?

Dataset: UNSW-NB15

- Created by IXIA PerfectStorm tool
- Hybrid between
  - “Real modern normal activity” [8]
  - “Synthetic contemporary attacks” [8]
- 49 Features
  - 36 Numerical
  - 13 Categorical/Boolean
- Total Entries: 2,540,044
- Train Set: 175,341
- Test Set: 82,332

	A	B	C	D	E	F	G	H	I	J
1	id	dur	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload
2	1	0.121478	6	4	258	172	74.08749	252	254	141
3	2	0.649902	14	38	734	42014	78.47337	62	252	839
4	3	1.623129	8	16	364	13186	14.17016	62	252	157
5	4	1.681642	12	12	628	770	13.67711	62	252	274
6	5	0.449454	10	6	534	268	33.37383	254	252	856
7	6	0.380537	10	6	534	268	39.41798	254	252	101
8	7	0.637109	10	8	534	354	26.68303	254	252	603
9	8	0.521584	10	8	534	354	32.59303	254	252	737
10	9	0.542905	10	8	534	354	31.31303	254	252	708
11	10	0.258687	10	6	534	268	57.98514	254	252	148
12	11	0.304853	12	6	4142	268	55.76458	254	252	996
13	12	2.093085	62	28	56329	2212	42.52097	62	252	211
14	13	0.416952	10	6	534	268	35.97536	254	252	92
15	14	0.996221	10	8	564	354	17.06449	254	252	407
16	15	0.576755	10	8	534	354	29.47525	254	252	667
17	16	0.000002	2	0	138	0	500000	254	0	2.76
18	17	0.728252	10	6	534	268	20.59727	254	252	528
19	18	0.393556	10	8	860	1096	43.19589	62	252	157
20	19	0.387852	10	6	534	268	38.67455	254	252	99
21	20	0.53784	10	8	534	354	31.60791	254	252	715

This project used the provided training and testing sets.

Approach: Overview

Experimental Setup

- Google Colab
- Numpy, Pandas, MatPlotLib
- Scikit-Learn (SKLearn)
    - XGBClassifier
        - Supervised
    - PCA
        - Unsupervised
    - Standard Scaler
        - Normalization

[4]

[2]

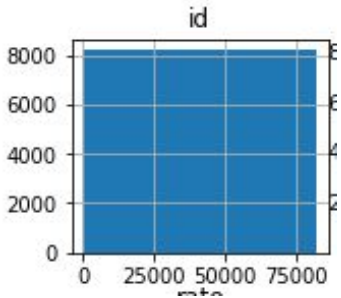
[3]

[1]

[5]

Preprocessing

- Pandas Dataframe to Visualize
  - Needed to delete the ID column...

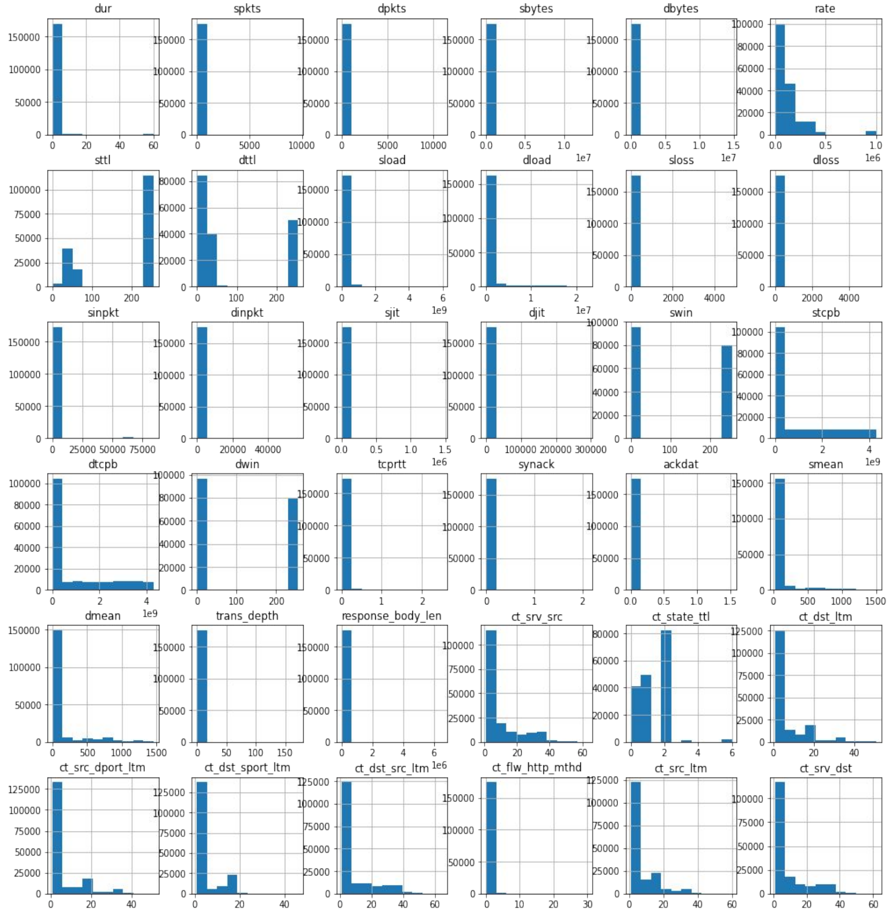


- Deleted categorical data from .csv
  - Reduced to 36 numerical features

Updates:

- Originally worked with 37 features...
  - Had a stray categorical feature in the testing and training set
  - This feature was not the same across either set either
- Used Pandas to bring in data
  - Pandas.to_numpy when needed

Histograms for Both Training and Testing Sets



Train

Test

ML Topics

XGBoost

- What
  - Supervised gradient boosted decision tree
  - Designed for speed and performance
- Why
  - Literature to support [6]
  - Feature importances

[7]

PCA ...

- What
  - Principal Component Analysis
  - Unsupervised Dimension Reduction
- Why
  - Suggested by classmates
  - Supported by literature

....with XGBoost Classification

- Why
  - Compare apples to apples!
  - Use XGBoost to classify using the PCA components

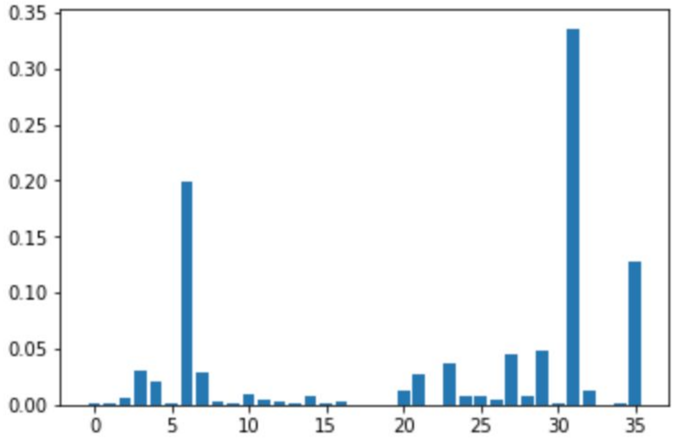
XGBoost ➣ Supervised

- Mask data to certain classes
  - Normal
  - Exploits
  - Generic

```
trainMask = normMaskTrain | expMaskTrain |genMaskTrain
Xtrain = Xtrain[trainMask]
```

- Feature Importances
  - The larger the number, the more important that feature is

```
[0.0000000e+00 0.0000000e+00 1.9479725e-04 3.3108634e-04 4.5649050e-04
 6.2389573e-04 6.8648177e-04 1.1235730e-03 1.2149664e-03 1.2480399e-03
 1.3550825e-03 1.3863370e-03 1.9410467e-03 2.3902759e-03 3.2641746e-03
 3.6658379e-03 3.8683368e-03 4.8685479e-03 6.3580563e-03 7.0599322e-03
 7.2080838e-03 7.2195963e-03 7.9380777e-03 9.5341904e-03 1.2459937e-02
 1.2892747e-02 2.0511359e-02 2.6896603e-02 2.9370632e-02 3.0837227e-02
 3.6561612e-02 4.4406794e-02 4.8723351e-02 1.2823379e-01 1.9927140e-01
 3.3589762e-01]
```



Visualization of Feature Importances

XGBoost Results

All Classes

Accuracy: 75.68%

	precision	recall	f1-score	support
0.0	0.95	0.74	0.83	37000
1.0	0.29	0.58	0.39	6062
2.0	0.00	0.00	0.00	677
3.0	0.53	0.03	0.05	583
4.0	0.42	0.02	0.03	4089
5.0	0.55	0.92	0.69	11132
6.0	1.00	0.96	0.98	18871
7.0	0.86	0.80	0.83	3496
8.0	0.37	0.40	0.38	378
9.0	0.71	0.39	0.50	44
accuracy			0.76	82332
macro avg	0.57	0.48	0.47	82332
weighted avg	0.81	0.76	0.76	82332

Normal = 0 Fuzzers = 1 Analysis = 2 Backdoor =
Generic = 6 Reconnaissance = 7 Shellcode = 8 Wo

Largest 3 Classes

Accuracy: 95.45%

	precision	recall	f1-score	support
0.0	0.99	0.95	0.97	37000
5.0	0.81	0.97	0.88	11132
6.0	1.00	0.96	0.98	18871
accuracy			0.95	67003
macro avg	0.93	0.96	0.94	67003
weighted avg	0.96	0.95	0.96	67003

Normal = 0 Fuzzers = 1 Analysis = 2 Backdoor = 3 DoS = 4 Exploits = 5
Generic = 6 Reconnaissance = 7 Shellcode = 8 Worms = 9

Thresholds - XGBoost

	precision	recall	f1-score	support
0.0	0.99	0.95	0.97	37000
5.0	0.81	0.97	0.88	11132
6.0	1.00	0.96	0.98	18871
accuracy			0.95	67003
macro avg	0.93	0.96	0.94	67003
weighted avg	0.96	0.95	0.96	67003

Thresh=0.001, n=30, Accuracy: 95.50%

(highest accuracy)

- Each feature has an importance value.
  - The higher the value, the more important it is for classification
- Method
  - Loop through the importances
  - Choose only the features that are that important or more
  - Fit the model and classify using only those features
- Purpose
  - Find the smallest number of features while maintaining high accuracy

	precision	recall	f1-score	support
0.0	0.99	0.94	0.96	37000
5.0	0.79	0.96	0.87	11132
6.0	1.00	0.96	0.98	18871
accuracy			0.95	67003
macro avg	0.93	0.95	0.94	67003
weighted avg	0.96	0.95	0.95	67003

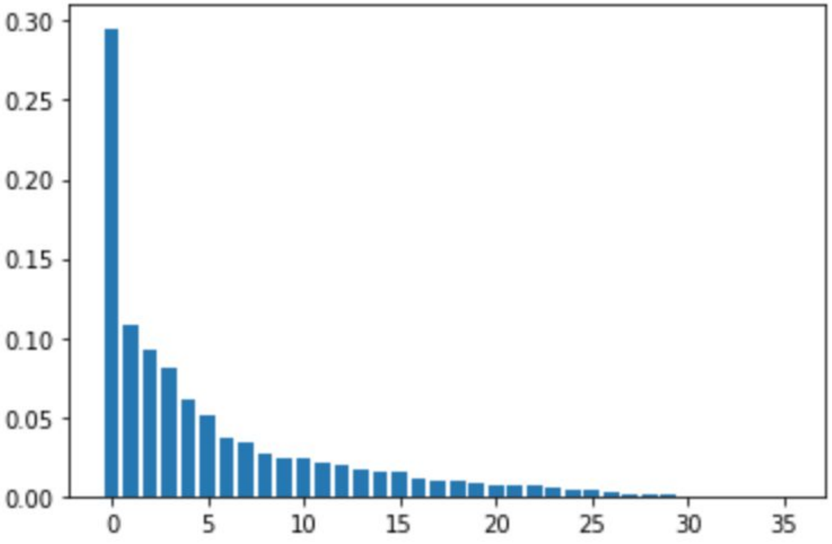
Thresh=0.021, n=10, Accuracy: 95.03%

(fewest features selected
>95% accuracy)

PCA ⤜ Unsupervised

- Started off with all 36 features
- Find importances of each component

Component Importances

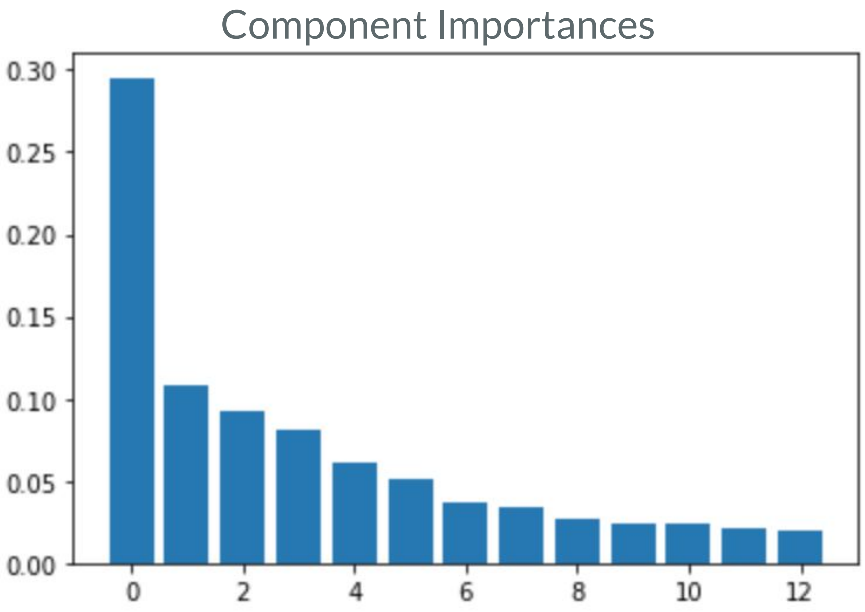


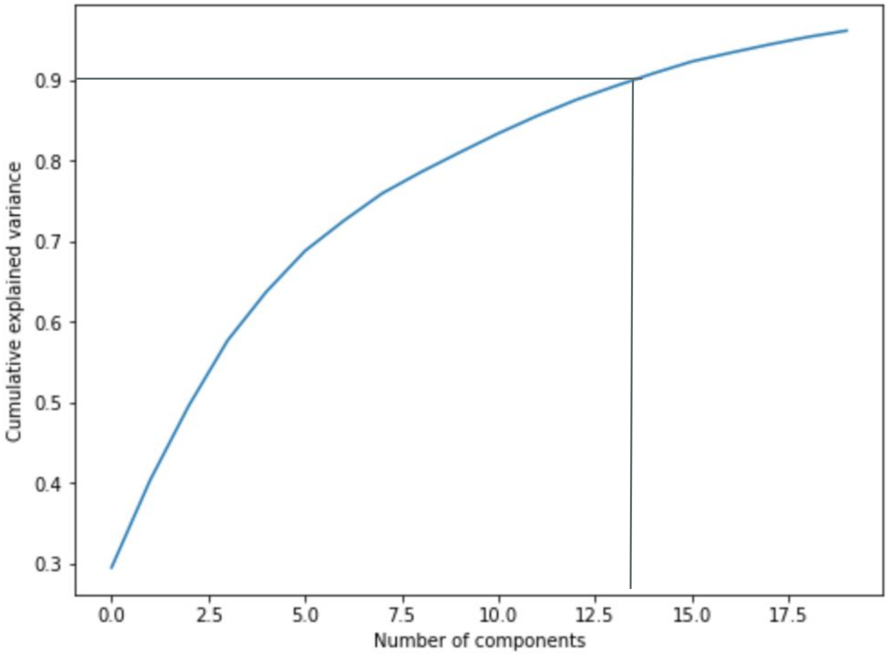
- Normalize input using StandardScaler
  - RobustScaler did not produce expected output, so Standard Scaler was kept.
- Classify with XGBoost
  - Compare apples to apples!

```
[2.95253490e-01 1.08719192e-01 9.24555777e-02 8.06191083e-02
 6.04235568e-02 5.06637723e-02 3.74088076e-02 3.41568978e-02
 2.63659669e-02 2.43812164e-02 2.36063263e-02 2.17295879e-02
 1.97477396e-02 1.68946383e-02 1.60421441e-02 1.47555668e-02
 1.08572772e-02 1.02486208e-02 9.37591006e-03 7.86036909e-03
 7.40635378e-03 6.67352971e-03 6.48078670e-03 5.60916840e-03
 3.94569463e-03 3.78584904e-03 1.99029985e-03 7.53144944e-04
 5.37751598e-04 4.28765411e-04 4.09350191e-04 1.81187233e-04
 1.80993276e-04 3.55177308e-05 1.58420155e-05 6.41783766e-32]
```

How many components to pick?

About 13 components will be enough for 90% cumulative explained variance





XGboost Classification using PCA Components

- Overall Accuracy: 71%
  - Not good!
- F1-Score for Exploits: 0.19
  - Not good!

	precision	recall	f1-score	support
0.0	0.78	0.93	0.85	37000
5.0	0.18	0.19	0.19	11132
6.0	0.96	0.58	0.72	18871
accuracy			0.71	67003
macro avg	0.64	0.57	0.59	67003
weighted avg	0.73	0.71	0.70	67003

Why?

Is it possible that since PCA is more of a data extraction method rather than a feature selection method, it will not produce the same kind of results?

```
Normal = 0     Fuzzers = 1     Analysis = 2     Backdoor = 3     DoS = 4     Exploits = 5
Generic = 6     Reconnaissance = 7     Shellcode = 8     Worms = 9
```

So What?

Why Even Select Features?

Time to fit XGBoost model with all 36 features 95% accuracy

```
CPU times: user 49.1 s, sys: 58.5 ms, total: 49.2 s
Wall time: 51 s
              precision    recall  f1-score   support

         0.0       0.99      0.95      0.97     37000
         5.0       0.81      0.97      0.88     11132
         6.0       1.00      0.96      0.98     18871

    accuracy                           0.95     67003
   macro avg       0.93      0.96      0.94     67003
weighted avg       0.96      0.95      0.96     67003

Thresh=0.000, n=36, Accuracy: 95.45%
```

Time to fit with keeping with a >95% accuracy

```
CPU times: user 16 s, sys: 25.9 ms, total: 16 s
Wall time: 16 s
              precision    recall  f1-score   support

         0.0       0.99      0.94      0.96     37000
         5.0       0.79      0.96      0.87     11132
         6.0       1.00      0.96      0.98     18871

    accuracy                           0.95     67003
   macro avg       0.93      0.95      0.94     67003
weighted avg       0.96      0.95      0.95     67003

Thresh=0.021, n=10, Accuracy: 95.03%
```

What's Better?

- Supervised feature selection produces a better classification model for XGBoost.
- After choosing features, PCA takes twice as long to fit with good number of components

```
CPU times: user 16 s, sys: 25.9 ms, total: 16 s
Wall time: 16 s
              precision    recall  f1-score   support

         0.0       0.99      0.94      0.96     37000
         5.0       0.79      0.96      0.87     11132
         6.0       1.00      0.96      0.98     18871

    accuracy                           0.95     67003
   macro avg       0.93      0.95      0.94     67003
weighted avg       0.96      0.95      0.95     67003

Thresh=0.021, n=10, Accuracy: 95.03%
```

XGBoost with XGBoost Features

```
CPU times: user 37.4 s, sys: 46.7 ms, total: 37.4 s
Wall time: 38.4 s
              precision    recall  f1-score   support

         0.0       0.78      0.93      0.85     37000
         5.0       0.18      0.19      0.19     11132
         6.0       0.96      0.58      0.72     18871

    accuracy                           0.71     67003
   macro avg       0.64      0.57      0.59     67003
weighted avg       0.73      0.71      0.70     67003
```

XGBoost with PCA Components

GitHub

- GitHub Link
  - Contains Jupyter Notebook
  - README
  - and this Presentation

Thank you!

Questions?

References

[1] SciKit

[2] Google Colab

[3] NumPy

[4] Pandas

[5] MatPlotLib

[6] Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset

[7] Wikipedia: XGBoost

[8] UNSW-NB15

Project Google Colab Link