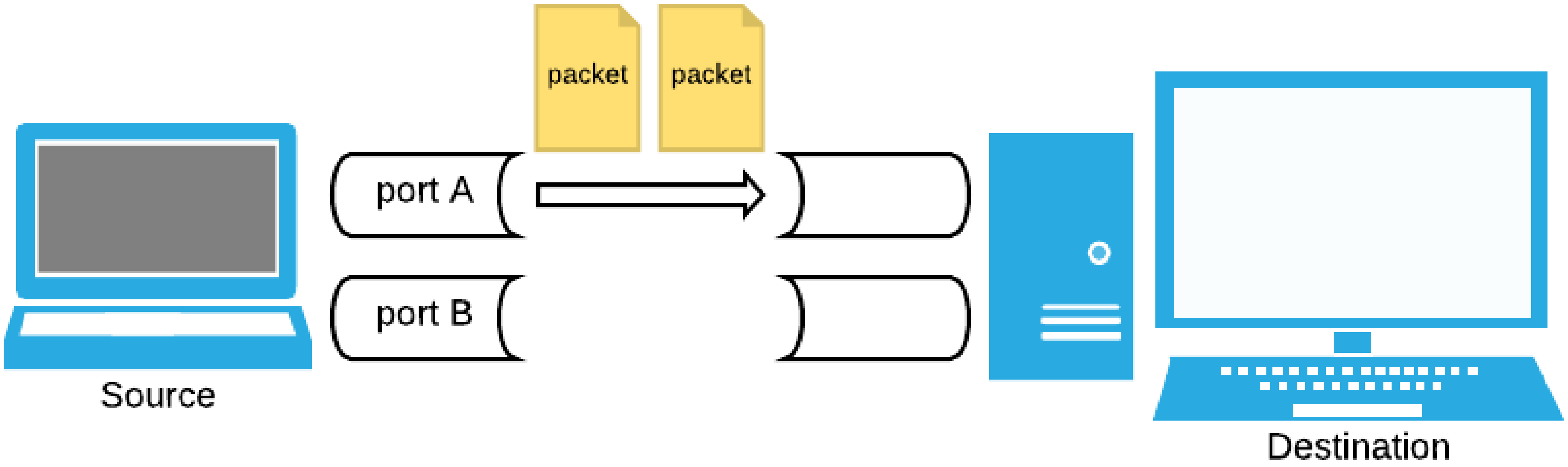
Data fusion

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

Computers, ports, and protocols



The LANL cyber dataset

> `flows` : Flows are sessions of continuous data transfer between a port on a source computer and a port on a destination computer, following a certain protocol.

```
flows.iloc[1]
```

```
time                     471692
duration                      0
source_computer           C5808
source_port               N2414
destination_computer     C26871
destination_port         N19148
protocol                      6
packet_count                  1
byte_count                   60
```

[1] https://csr.lanl.gov/data/cyber1/

The LANL cyber dataset

> `attack` : information about certain attacks performed by the security team itself during a test.

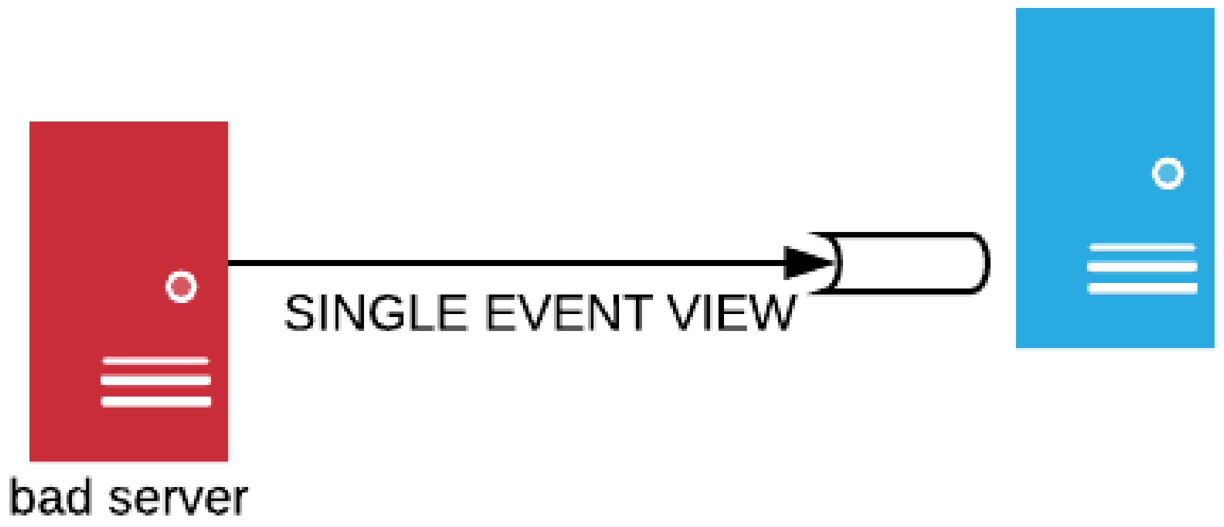
```
attacks.head()
```

```
     time user@domain source_computer destination_computer
0  151036   U748@DOM1          C17693                 C305
1  151648   U748@DOM1          C17693                 C728
2  151993  U6115@DOM1          C17693                C1173
3  153792   U636@DOM1          C17693                 C294
4  155219   U748@DOM1          C17693                C5693
```

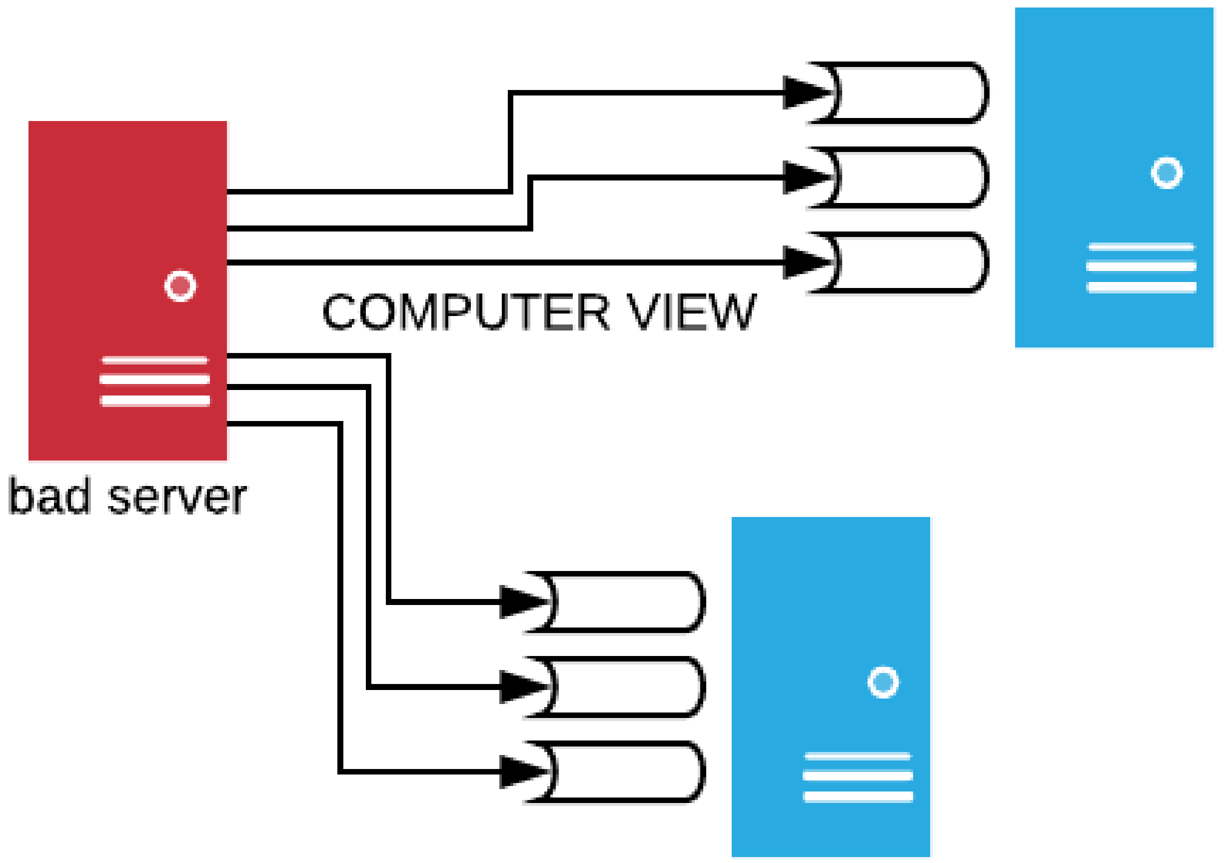
[1] https://csr.lanl.gov/data/cyber1/

Labeling events versus labeling computers

A single event cannot be easily labeled.



But an entire computer is either infected or not.



Group and featurize

Unit of analysis = `destination_computer`

```
flows_grouped = flows.groupby('destination_computer')

list(flows_grouped)[0]
```

```
('C10047',
        time  duration    ...     packet_count byte_count
2791  471694         0    ...               12       6988
2792  471694         0    ...                1        193
...
2846  471694        38    ...              157      84120
```

Group and featurize

From one DataFrame per computer, to one feature vector per computer.

```
def featurize(df):
    return {
        'unique_ports': len(set(df['destination_port'])),
        'average_packet': np.mean(df['packet_count']),
        'average_duration': np.mean(df['duration'])
    }
```

Group and featurize

```
out = flows.groupby('destination_computer').apply(featurize)
```

```
X = pd.DataFrame(list(out), index=out.index)
X.head()
```

```
                      average_duration      ...       unique_ports
destination_computer                        ...
C10047                        7.538462      ...                 13
C10054                        0.000000      ...                  1
C10131                       55.000000      ...                  1
...
[5 rows x 3 columns]
```

Labeled dataset

```
bads = set(attacks['source_computer'].append(attacks['destination_computer']))
y = [x in bads for x in X.index]
```

The pair `(X, y)` is now a standard labeled classification dataset.

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
clf = AdaBoostClassifier()
accuracy_score(y_test, clf.fit(X_train, y_train).predict(X_test))
```

```
0.92
```

Ready to catch a hacker?

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Labels, weak labels and truth

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

Labels are not always perfect

Degrees of truth:

- **Ground truth**
  - the computer crashes and a message asks for ransom money
- **Human expert labeling**
  - the analyst inspects the computer logs and identifies unauthorized behaviors
- **Heuristic labeling**
  - too many ports received traffic in a very small period of time

Labels are not always perfect

Noiseless or *strong* labels:

- Ground truth
- Human expert labeling

Noisy or *weak* labels:

- Heuristic labeling

Feature engineering:

- Features used in heuristics

Features and heuristics

Average of unique ports visited by each infected host:

```
np.mean(X[y]['unique_ports'])
```

```
15.11
```

Average of unique ports visited per host disregarding labels:

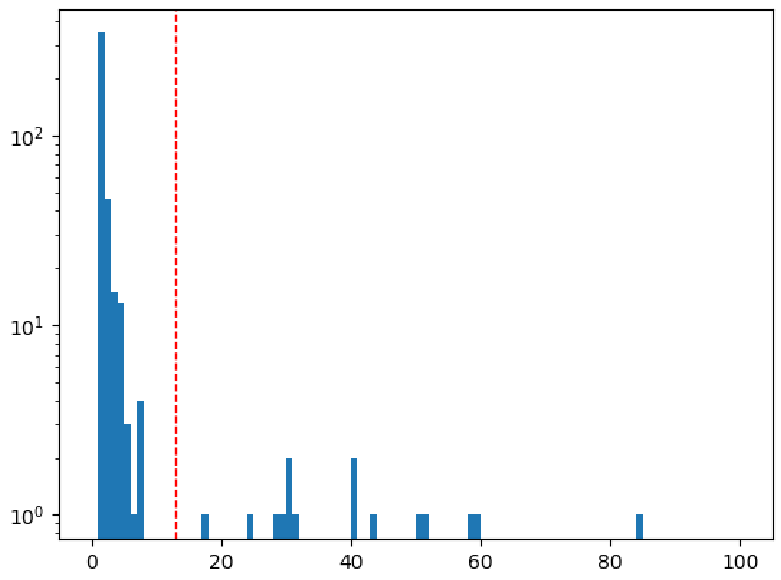
```
np.mean(X['unique_ports'])
```

```
11.23
```

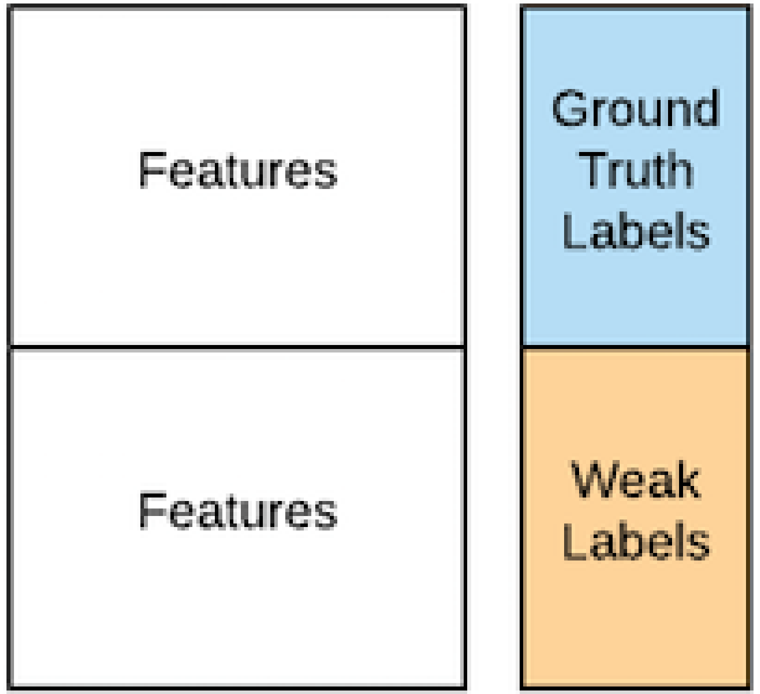
From features to labels

Convert a feature into a labeling heuristic:

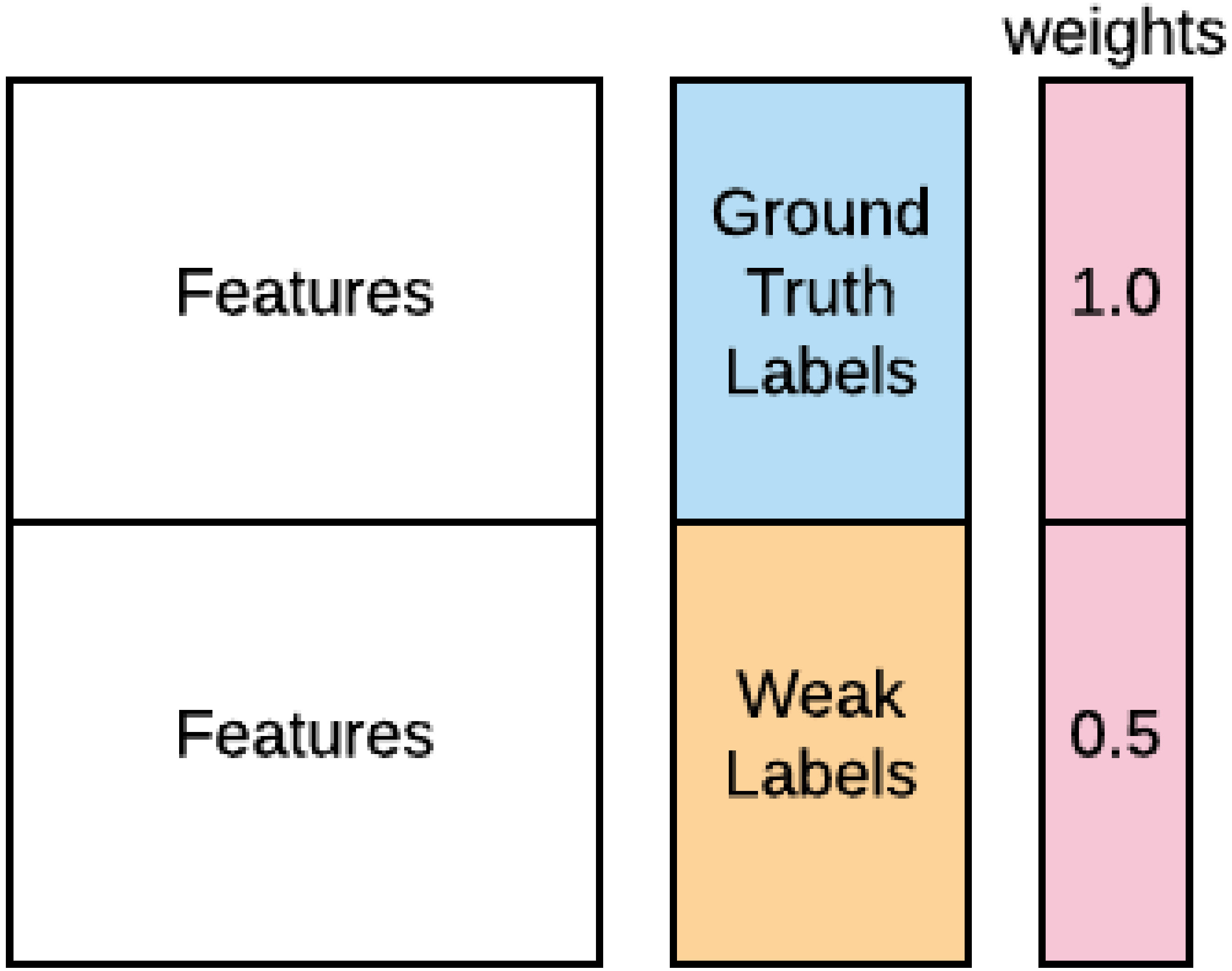
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
y_weak_train = X_train['unique_ports'] > 15
```



From features to labels




```
X_train_aug = pd.concat([X_train, X_train])
y_train_aug = pd.concat([pd.Series(y_train), pd.Series(y_weak_train)])
```




```
weights = [1.0]*len(y_train) + [0.1]*len(y_weak_train)
```

Accuracy using ground truth only:

```
0.91
```

Ground truth and weak labels without weights:

```
accuracy_score(y_test, clf.fit(X_train_aug, y_train_aug).predict(X_test))
```

```
0.93
```

Add weights:

```
accuracy_score(y_test, clf.fit(X_train_aug, y_train_aug, sample_weight=weights).predict(X_test))
```

```
0.95
```

Labels do not need to be perfect!

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Loss functions Part I

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

The KDD '99 cup dataset

```
kdd.iloc[0]
```

```
kdd.iloc[0]
duration                        51
protocol_type                  tcp
service                       smtp
flag                            SF
src_bytes                     1169
dst_bytes                      332
land                             0
...
dst_host_rerror_rate             0
dst_host_srv_rerror_rate         0
label                         good
```

False positives vs false negatives

Binarize label:

```
kdd['label'] = kdd['label'] == 'bad'
```

Fit a Gaussian Naive Bayes classifier:

```
clf = GaussianNB().fit(X_train, y_train)
predictions = clf.predict(X_test)
results = pd.DataFrame({
    'actual': y_test,
    'predicted': predictions
})
```

```
     actual  predicted
0      True       True
1     False      False
2      True      False
3     False       True
```

False positives vs false negatives

Binarize label:

```
kdd['label'] = kdd['label'] == 'bad'
```

Fit a Gaussian Naive Bayes classifier:

```
clf = GaussianNB().fit(X_train, y_train)
predictions = clf.predict(X_test)
results = pd.DataFrame({
    'actual': y_test,
    'predicted': predictions
})
```

```
   actual  predicted
0    True       True
1   False      False
2    True      False
3   False       True
```

False positives vs false negatives

Binarize label:

```
kdd['label'] = kdd['label'] == 'bad'
```

Fit a Gaussian Naive Bayes classifier:

```
clf = GaussianNB().fit(X_train, y_train)
predictions = clf.predict(X_test)
results = pd.DataFrame({
    'actual': y_test,
    'predicted': predictions
})
```

```
   actual  predicted
0    True       True
1   False      False
2    True      False
3   False       True
```

False positives vs false negatives

Binarize label:

```
kdd['label'] = kdd['label'] == 'bad'
```

Fit a Gaussian Naive Bayes classifier:

```
clf = GaussianNB().fit(X_train, y_train)
predictions = clf.predict(X_test)
results = pd.DataFrame({
    'actual': y_test,
    'predicted': predictions
})
```

```
    actual  predicted
0     True       True
1    False      False
2     True      False
3    False       True
```

The confusion matrix

```
conf_mat = confusion_matrix(
    ground_truth, predictions)
```

```
array([[9477,   19],
       [ 397, 2458]])
```

```
tn, fp, fn, tp = conf_mat.ravel()
(fp, fn)
```

```
(19, 397)
```

Actual / Predicted	**Bad traffic**	**Normal traffic**
Labelled bad	True Positives 9477	False Positives 19
Labelled normal	False Negatives 397	True Negatives 2458

Scalar performance metrics

```
accuracy = 1-(fp + fn)/len(ground_truth)

recall = tp/(tp+fn)

fpr = fp/(tn+fp)

precision = tp/(tp+fp)

f1 = 2*(precision*recall)/(precision+recall)
```

```
accuracy_score(ground_truth, predictions)
recall_score(ground_truth, predictions)
precision_score(ground_truth, predictions)
f1_score(ground_truth, predictions)
```

False positives vs false negatives

Classifier A:

```
tn, fp, fn, tp = confusion_matrix(
    ground_truth, predictions_A).ravel()
(fp,fn)
```

```
(3, 3)
```

```
cost = 10*fp + fn
```

```
33
```

Classifier B:

```
tn, fp, fn, tp = confusion_matrix(
    ground_truth, predictions_B).ravel()
(fp,fn)
```

```
(0, 26)
```

```
cost = 10*fp + fn
```

```
26
```

Which classifier is better?

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Loss functions Part II

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

Probability scores

```
clf = GaussianNB().fit(X_train, y_train)
```

```
scores = clf.predict_proba(X_test)
```

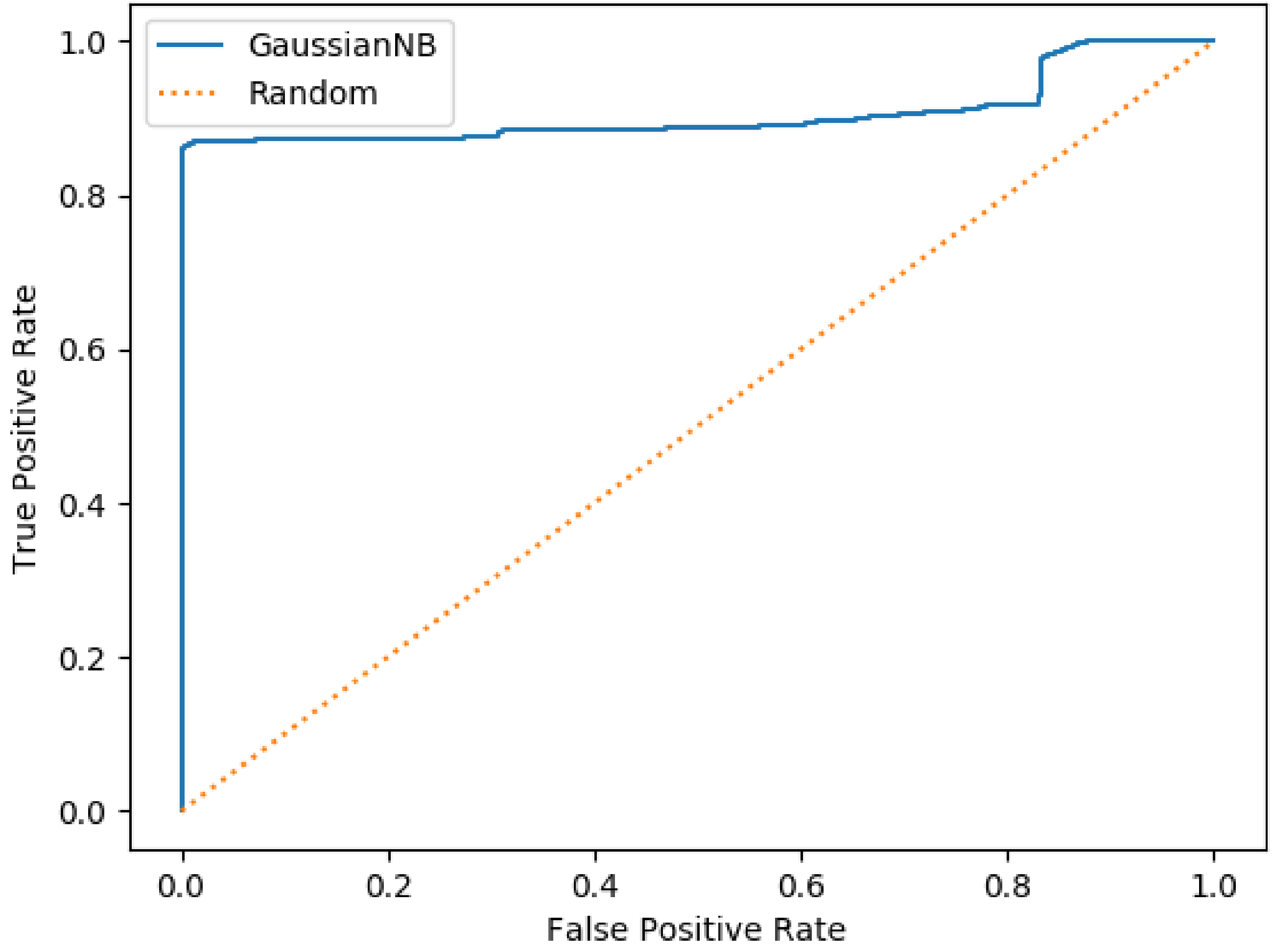
```
array([[3.74717371e-07, 9.99999625e-01],
       [9.99943716e-01, 5.62841678e-05],
       ...,
       [9.99937502e-01, 6.24977552e-05]])
```

```
[s[1] > 0.5 for s in scores] == clf.predict(X_test)
```

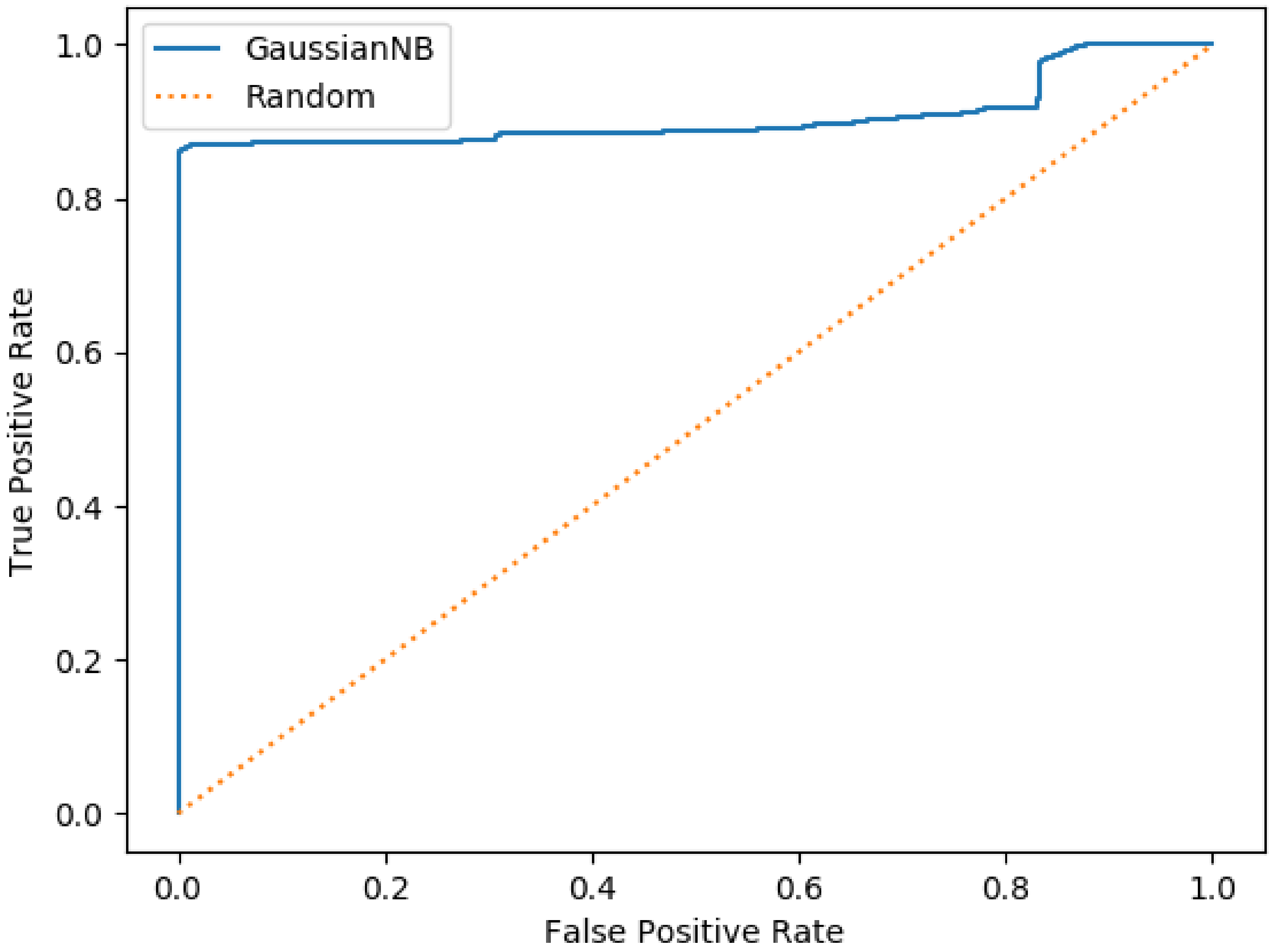
Probability scores

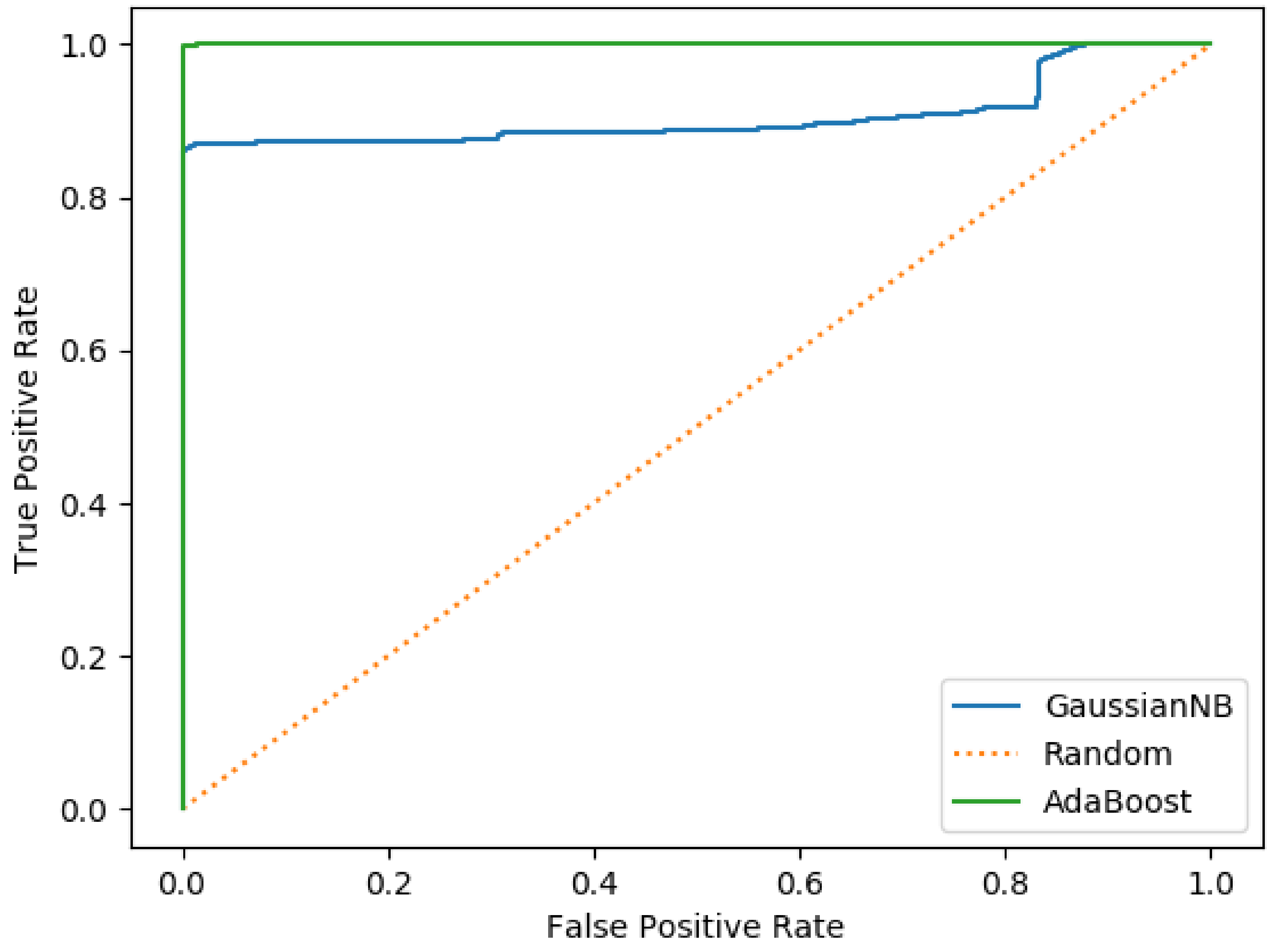
Threshold	false positive	false negative
0.0	178	0
0.25	66	17
0.5	35	37
0.75	13	57
1.0	0	72

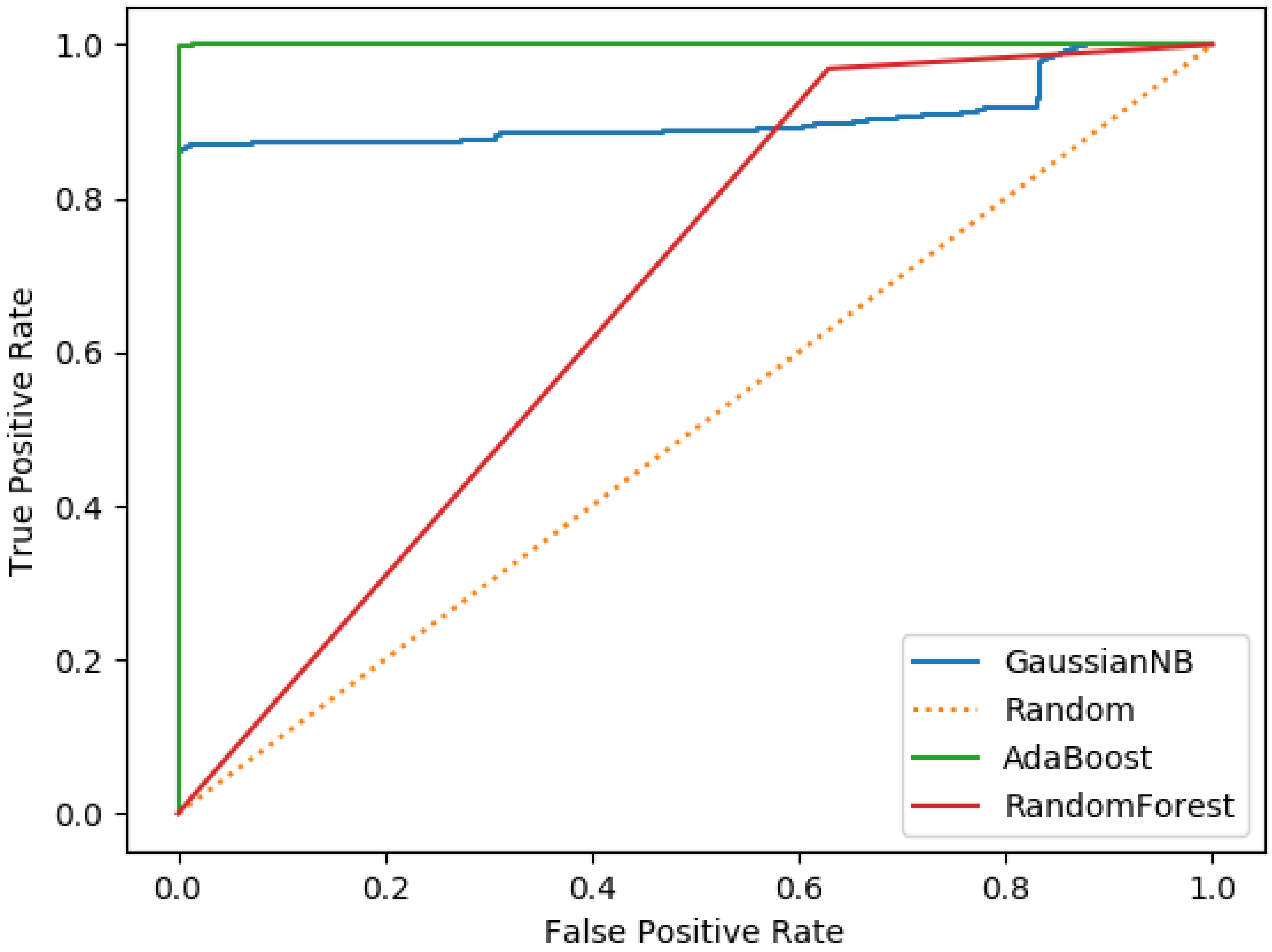
ROC curves




```
fpr, tpr, thres = roc_curve(
    ground_truth,
    [s[1] for s in scores])
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```


GaussianNB
Random
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
GaussianNB
Random
AdaBoost


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
GaussianNB
Random
AdaBoost
RandomForest

AUC

```
clf = AdaBoostClassifier().fit(X_train, y_train)
scores_ab = clf.predict_proba(X_test)
roc_auc_score(ground_truth, [s[1] for s in scores_ab])
```

```
0.9999
```

Cost minimisation

```
def my_scorer(y_test, y_est, cost_fp=10.0, cost_fn=1.0):
    tn, fp, fn, tp = confusion_matrix(y_test, y_est).ravel()
    return cost_fp*fp + cost_fn*fn
```

```
t_range = [0.0, 0.25, 0.5, 0.75, 1.0]
costs = [
   my_scorer(y_test, [s[1] > thres for s in scores]) for thres in t_range
]
```

```
[94740.0, 626.0, 587.0, 507.0, 2855.0]
```

Each use case is different!

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON