July 10 2017

Machine learning case study with KNN

Introduction

These days, machine learning applications in the cyber security domain are very popular. These days, machine learning applications in the cyber security domain are very popular. Since i started to take Applied Machine Learning in Python course (https://www.coursera.org/learn/python-machine-learning/), I am trying to apply machine learning algorithms to infosec cases. In this article, I tried to detect dos/ddos attacks with K-Nearest-Neighbor algorithm which is inredibly accurate and simple.

I coded this project with python and the following commonly used libraries.

- scikit-learn
- pandas

I used KDDCUP99 dataset for training. Normally, this dataset shouldn't be used for training the real systems.[1] Since this study is for learning purposes I used this dataset freely.

KDDCUP99 Dataset

Kddcup99 dataset created by DARPA in 1998. And it used at KDD-CUP competition in 1999. That's why it's named Kddcup99. Dataset contains 41 features. Some of these features extracted from network packets. But 10 of these features are host-based information and only gained from compromised hosts. For example:

- su_attempted
- num_shells
- num_access_file

I deleted these host-based features. Because I inspect only network packets for detection.

Part of the dataset is shown below.

duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num
0	tcp	http	SF	181	5450	0	0	0	0	
0	tcp	http	SF	239	486	0	0	0	0	
0	tcp	http	SF	235	1337	0	0	0	0	
0	tcp	http	SF	219	1337	0	0	0	0	

Also, dataset contains 22 different attack types as label. Attack types are shown below.

```
- ftp_write
- guess_passwd
- imap
- ipsweep
- land
- loadmodule
- multihop
- neptune
- nmap
- perl
- phf
- pod
- portsweep
- rootkit
- satan
- smurf
- spy
- teardrop
- warezclient
- warezmaster
```

I aggregated and eliminated some of these labels because we will only detect DoS/DDoS traffic. Nmap, buffer overflow and other similiar attack types are out of our scope.

As i said before, Kddcup99 dataset shouldn't used with real network ids system. Because, dataset contains some obsolote attacks and host-based features. These features can't be extracted from network packets by network intrusion detection system.

Machine Learning Phases

General machine learning process has 4 stage:

- Preprocessing data
- Building model
- Model training
- Testing an optimization

Preprocessing Data

In this phase, dataset must be analyzed and visualized carefully. If there are errors or missing values in dataset

1. If these errors are too many in one feature, this feature can be deleted.
2. These errors can be replaced with mean of this attribute.

After data clenaing, values must be normalized. A lot of techinuques used for data normalization. These techniques must be examined very well and appropirate method for dataset should selected. For example:

```
#for mass attribute
sample = mass[i]
min_value = min(mass)
max_value = max(mass)
normalized_sample = (sample - min_value) / (max_value - min_value)
```

Kddcup99 dataset is a preprocessed, normalized and well cleaned data. Therefore, dataset doesn't contain any error. So, i passed these phases.

We may need to change feature types for our algorithm. For instance, kddcup99 dataset contains categorical(protocol_type) variables. But categorical values can't be used with KNN algoritm. So, we need to delete or transform these data into numerical values. Categorical features in dataset are as follows.

- protocol_type
- service
- flag
- land

Only **land** feature expressed with numerical values. So I used this feature as it is. But I transformed other 3 features into numerical values. For this purpose, I found all different values with command below. And I numerated this values from 0 to 3. I did this transformation for other two features.

```
cat kddcup_data | cut -d"," -f2 | sort | uniq
icmp
tcp
udp
```

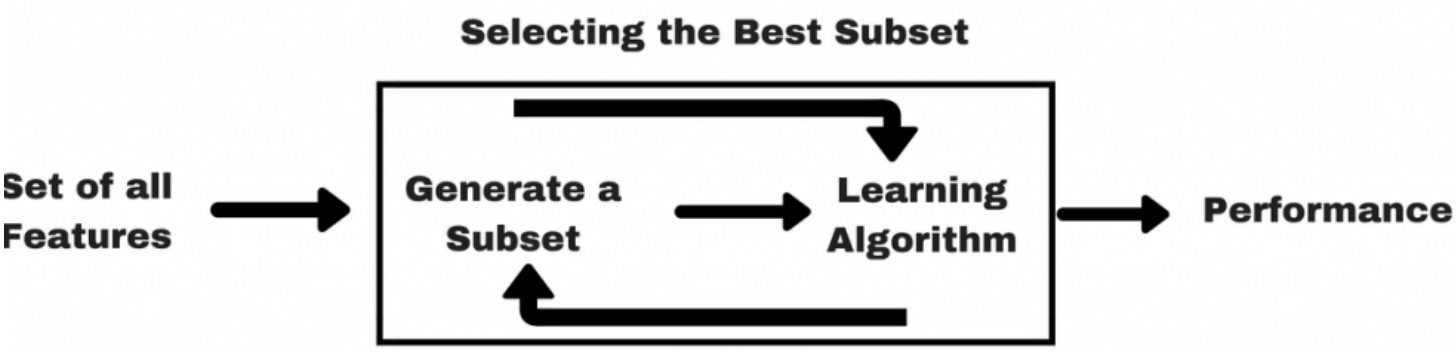
Protocol Type	Value
ICMP	0
TCP	1
UDP	2

I automatized all work up to this stage with this () script.

Afterwards, we may select valuable features for dataset. Feature selection process should made by domain expert. Good selection provides more accurate model and fast prediction.

Firstly, we should analyze features in terms of usability. Kddcup99 dataset has host-based features and these features aren't usable for our case. Therefore, I omitted host-based features.(10-22)

Secondly, we might use different feature selection algorithm or use PCA(Principal Component Analysis) for combining and reducing features. I used Select-K-Best method for feature selection in this study. The Select-K-Best algorithm tests the model with different feature subsets to find the desired K number of features, and measures the success of the model according to the desired metric.(Eg: chi2, f_classif, mutual_info_classif [4])



I tested model with full features and selected 5 features. So model accuracy is almost the same but evaluation time is differ. Code is here ().

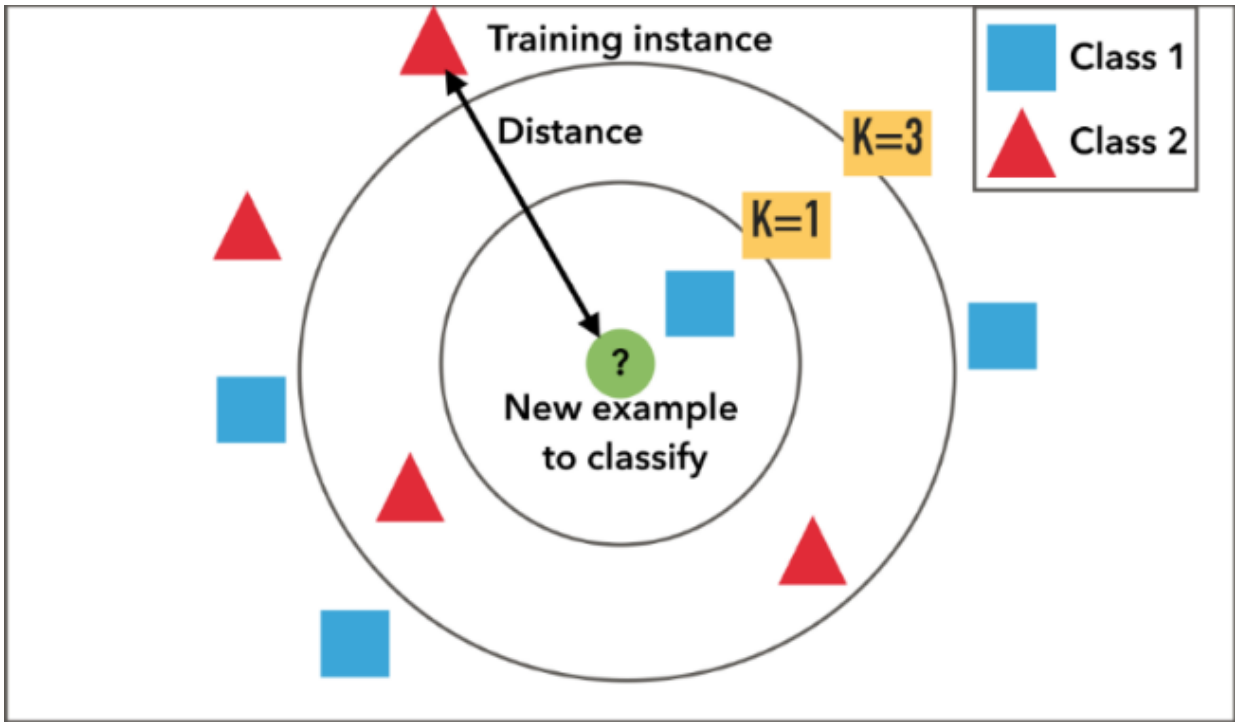
```
frkn@frkn:~/Desktop/applied_ml$ python knn.py
Testing with full data
[+] Classifier trained in 3.27163791656
[+] Model Evaluated in 4.10666203499
[!] Test score is 0.999297772534
------------------------------------------------
Testing with selected features
[+] Selected features
[-->] ['duration', 'src_bytes', 'dst_bytes', 'count', 'dst_host_srv_count']
[+] Classifier trained in 1.65853691101
[+] Model Evaluated in 0.355732917786
[!] Test score is 0.999044970647
------------------------------------------------
```

Building Model

The second stage of machine learning process is to train model with preprocessed data. With this model we will classify previously unseen data. As I said before, I create model with K-Nearest-Neighbor algorithm.

K-Nearest-Neighbor Algorithm

KNN is supervised machine learning algorithm. So, it needs labeled data for creating model. KNN classifies the sample according to the class of nearest K point to sample point. It is called majority voting. Obviously, it's so simple but incredibly powerfull.



As it seen figure above. Class of new sample is **class one** when k equal one. If k equal three then sample's class will be **class 2**.

Normally time complexity for training KNN algorithm is O(1). So, it copies all data to generic array. But with this way, prediction complexity will be O(kdn) approximately. k for neighbor count, d for feature dimension and n for training sample size. In prediction phase, algorithm compares new sample with all data and select k nearest point. This takes a lot of time with large datasets. But in scikit-learn, KNN uses kdtree or balltree data structures default instead of array. These data structures decrease prediction time significantly but increase training time little bit. Time complexity for train kdtree is O(nlogn), predicton time complexity is O(klogn) in average.[5]

I implemented KNN algorithm with normal way but it is so slow with large datasets in testing phase. Code is here ().

Implementation

Implementation is also simple with scikit-learn. First, we need install and import libraries.

```
import pandas as pd
from sklearn.model_selection import train_test_split #for creating train and test d
ataset from all data
from sklearn.neighbors import KNeighborsClassifier #scikit-learn KNN class
from sklearn.feature_selection import SelectKBest #for selecting features
from sklearn.feature_selection import chi2 #success metric for select-k-best
```

After, we need read and split data to training data and class label. Dataset file must contains feature names as header. Pandas read function, reads and stores these feature names as keys of dictionary.

```
def get_features(data):
  features = []
  for key in data.keys():
         features.append(key)
  features.remove("label") # i remove class labels from features
  return features

data = pd.read_csv(filename)
features = get_features(data)
X = data[features] # train data
y = data["label"]  # class labels
```

We need to select best features now.

```
selector = SelectKBest(score_func=chi2,k=5) # selector instance with chi2 metric
selector.fit(X,y) # calculating best five features
indexes_selected = selector.get_support(indices=True) # extracting features indexes
selected_features = [] # it will contain features names
for i in indexes_selected:
  selected_features.append(features[i])

X = data[selected_features] # new training data with selected features
```

If we will train and test model with the same data, we need to split dataset into two. After this partition we can create classifier instance and train it. And we can evaluate score with testset.

```
X_train, X_test, y_train, y_test = train_test_split(X,y) #split data into two poart
knn = KNeighborsClassifier(n_neighbors = 5) # creating classifier instance with 5 n
eighbors
knn.fit(X_train,y_train) # training the model
score = knn.score(X_test,y_test) #evaluating the model
```

Complete script is here ().

Testing with Real Data

Model score is about **0.99** , but this value seems very unreal. And I don't trust kddcup99 dataset so much. So, I tried to test model with some real data. I can't same features from pcaps very well. And i found KDDCUP99 Extractor () script as a result of long search. I compiled this code with Jetbrains Clion. It made building c++ code pretty easy.

First, I started syn flood with Hping and recorded with Wireshark.

```
hping3 -S IP --flood
```

After, I recorded some normal traffic like visiting facebook, telnet, file download etc. And I used kddcup99 extracor to extracting data from pcaps. I preprocessed extracted data with this () script. After these steps, I tested model with this data. Surprisingly, it works like a charm.

There was 65537 samples in dos-attack data. Some of them syn packet and some of them rst-ack answer packets. Classifier classified 65089 packets as a dos attack and 448 packets as a normal.

```
> print len(packets)
65537
> print result
{'dos': 65089, 'normal': 448}
```

And I tested model with normal pcap. Extracted data has 127 sample. Classifier classified all packets as normal.

```
> print len(packets)
127
> print result
{'normal': 127}
```

Lastly, these test results are very good. But may be unreliable. Please, create your model and test it different pcaps. If there is a mistake, please contact and we fix it together.

Note: I used KNN again in this kaggle contest () because i didn't trust theese result very much. But surprisingly, success rate about 0.98 again 😀

References

- http://www.alglib.net/other/nearestneighbors.php

Bye

Tweet

0 Yorum synack.blog frknozr

Öner Paylaş En İyiye Göre Sırala

Tartışma başlat…

İlk yorum yapan siz olun.

Abone Olun Sitenize Disqus ekleyinDisqus EkleEkle Gizlilik