KDD Cup 1999

Network Intrusion Detection As a Classification Task

Hengji Liu, z5043667
Igit Pratama Soeriasaputra, z5022437
COMP9417, Assignment 2

Introduction

Network devices collect a lot of data regarding connection characteristics. It can be used to categorise connections to tell if they are legitimate or illegitimate.

The aim of this project is building a model to classify network connections into five major connection types using the KDD’99 dataset. Among them, one is normal type and the rest are network attack types.

In this report, four machine learning algorithms are used: Adaptive Boosting, Random Forest, Extremely Randomized Trees and Logistic Regression. Their hyper-parameters are tuned by the means of grid search. Furthermore, voting and stacking are also investigated to improve the prediction.

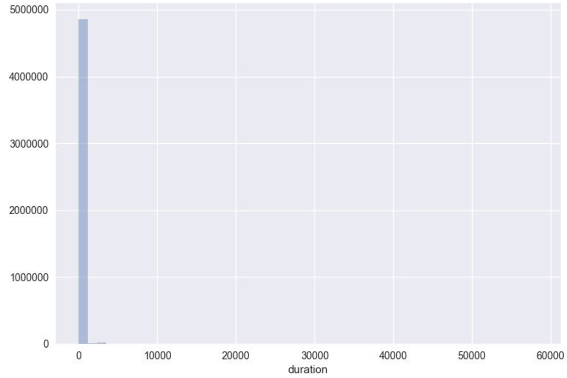
The performance of classification is measured by a cost matrix which gives different penalty to different mis-classification. Precision and recall rates of each category are also listed in a table. Among the five models that we investigated, the best prediction could gain the 9th place in the competition.

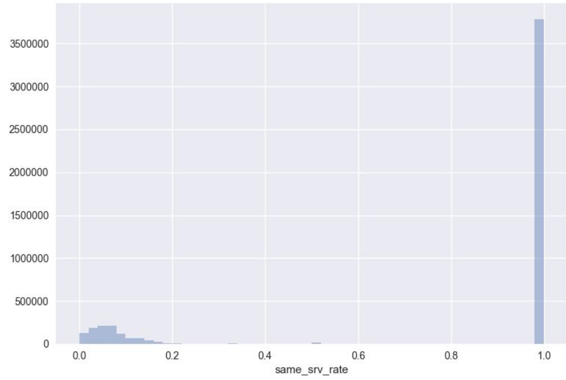
Exploratory Data Analysis

There are 4,898,431 samples in the training data and 311,029 in the testing data. Each sample contains 31 features. As shown in the table 1, the features are either categorical or numerical type. All features are well-collected and meaningful, because they are constructed by domain experts. Their specific meanings can be seen in the appendix. With some experiments, it was confirmed that there are no missing values or obviously irregular values in the data.

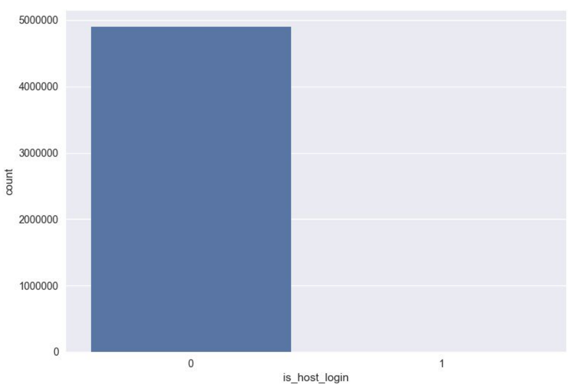
Feature Name		Type
duration src_bytes dst_bytes num_failed_logins hot num_compromised num_root num_file_creations	num_shells num_access_files num_outbound_cmds count srv_count dst_host_count dst_host_srv_count	Numeric (integer)
serror_rate srv_serror_rate rerror_rate srv_rerror_rate same_srv_rate diff_srv_rate srv_diff_host_rate dst_host_same_srv_rate	dst_host_diff_srv_rate dst_host_same_src_port_rate dst_host_srv_diff_host_rate dst_host_serror_rate dst_host_srv_serror_rate dst_host_rerror_rate dst_host_srv_rerror_rate	Numeric (decimal)
protocol_type service	flag attack_type	Categorical (multiple symbolic values)
land logged_in root_shell	su_attempted is_host_login is_guest_login	Categorical (0/1 binary)

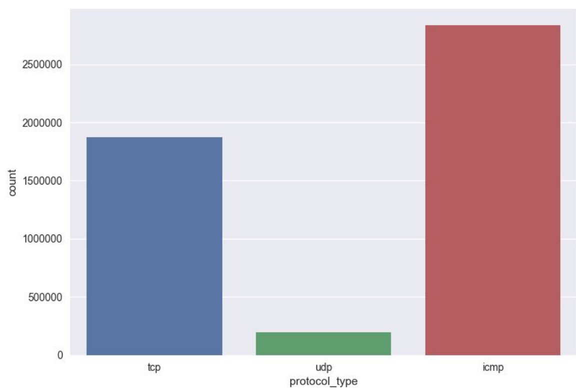
Interestingly, both numeric and categorical features have very skewed distribution. For example, numerical features like *duration* and *same_srv_rate* both have a surge on one value and a long tail.

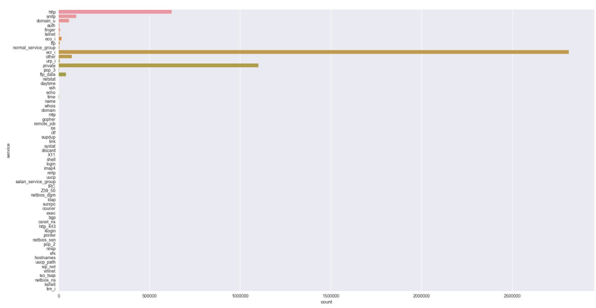


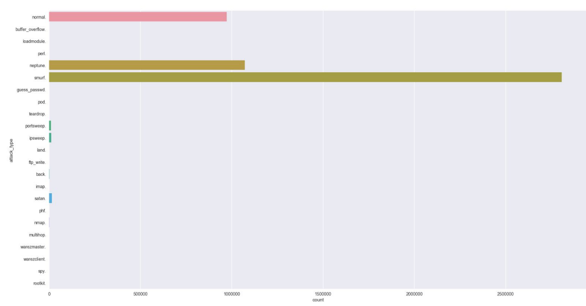


Categorical data show a feast-or-famine pattern like *is_host_login, protocol_type* and *service*.









Though the final results are compared in terms of the major five categories, the labels in the training data give rather specific types as shown in the right down figure above.

Feature Engineering

Due to the statistically ill-formed data, tree-based classifier were mainly used in the project. Hence, no normalisation or standardisation is actually needed. For categorical data, two feature engineering techniques are used: one-hot encoding and sparse features merging. Features are then selected using the feature importance attribute from Random Forest Classifier class to optimise training time and quality.

Merging Sparse Features

Feature *flag* and *service* have many possible values (shown in the appendix). Let *v* be a symbolic value in a feature column and *v*'s corresponding *attack_type* is solely *t*. If there are many *v* whose corresponding *t* is the same, then there is no significance between these *v*. This is called a sparse feature. Considering keeping sparse features and perform one-hot encoding, the resulting matrix will have two columns that are not independent and hence not full rank.

Merging sparse feature could reduce the computation cost so does training time. It also minimises the possibility that a sample is wrongly classified. A check was run on on *flag* and *service* to see if this kind of feature value exists. The result is that on *flag* column, *RSTOS0* corresponds to *portsweep*. No other value in this column has the same corresponding attack type. On the other hand, some values in *service* column correspond to entirely *normal* or *satan*.

Service value		Attack type
ntp_u urh_i	tftp_u red_i	normal
pm_dump http_2784 harvest	aol http_8001	satan

ntp_u, *urh_i*, *tftp_u*, *red_i* is merged to *normal_service_group* and *pm_dump*, *http_2784*, *harvest*, *aol*, *http_8001 is* merged to *satan_service_group*.

One-hot Encoding

If a categorical feature has multiple symbolic values, it is impossible to train on data having this kind of feature. One-hot encoding is a way to solve this.

In this project, *protocol_type*, *service* and *flag* were transformed from symbolic feature to one-hot encoded feature. For example, *protocol_type* has three values: *tcp*, *udp*, *icmp.*

Then, three columns named *tcp*, *udp*, *icmp* was appended to the data to represent the value of *protocol_type*. Finally, *protocol_type* was dropped from the data.

The figure below is an illustration.

row#	*protocol_type*
888	tcp
889	icmp
890	udp
891	udp

==>

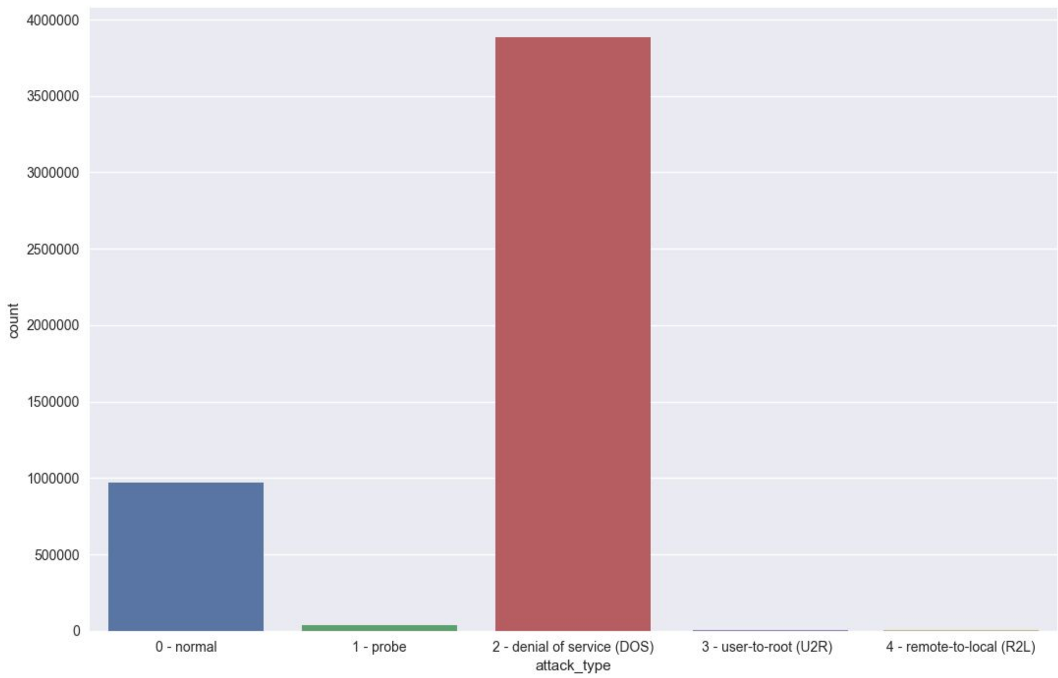
row#	*tcp*	*udp*	*icmp*
888	1	0	0
889	0	0	1
890	0	1	0
891	0	1	0

Apart from *protocol_type*, this transformation was also applied on *service* and *flag*.

Map to the major five categories

As described in the task, there can be different specific attack types in the test data. If the model learns how to classify samples into specific types, it will not suit the test data. Moreover, the final score itself is measured in terms of the major five categories. Thus, attack types column from the dataset should be mapped into the major five categories according to the marking script revealed at http://cseweb.ucsd.edu/~elkan/tabulate.html.

After mapping, the distribution of categories is shown in the figure below.

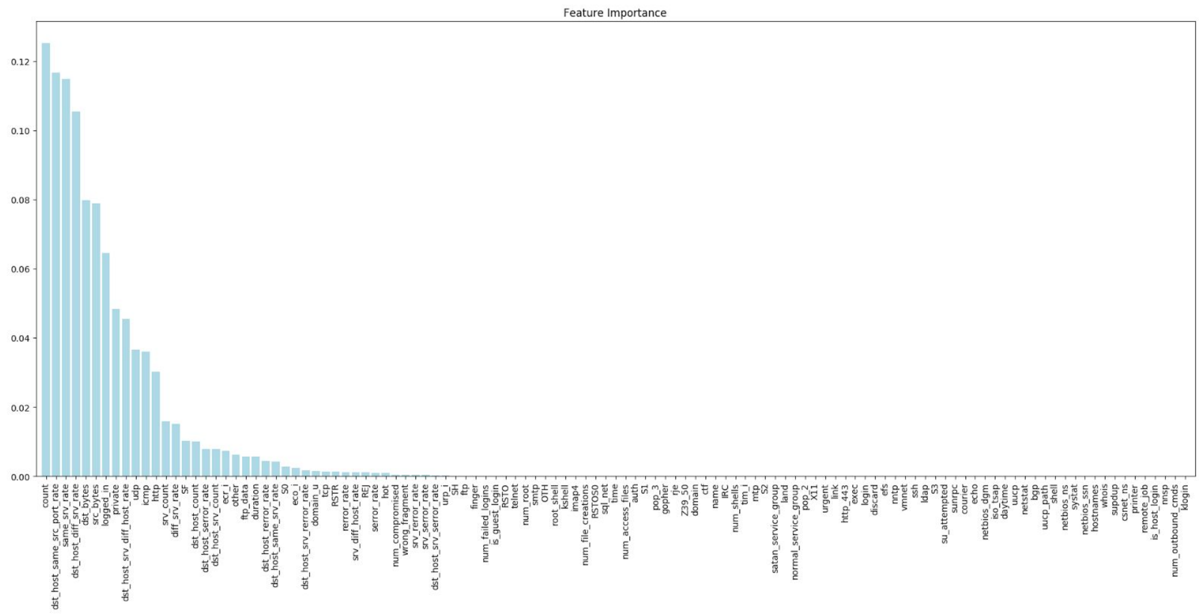


This also means the resulting model will be learning how to classify data directly into the major five categories instead of the specific attack types.

Feature Selection

After one-hot encoding, there are 115 features. Apparently, not all of them is significantly useful for the classification. Because of that, feature selection is implemented. In this way, only features of importance are used for building the classifier. Additionally, this step also drastically reduce training time without any major drop in model performance. Among many ways of feature selection, the feature importance attribute in the Random Forest model was utilised in this project to sieve features and show their relatedness to the classification.

Due to the inherent randomness of Random Forest, the model may give features different importance weights every time. But by training the model several times, it is observed that approximately 40 to 45 features have significant contribution in influences on the classification task. An indicative figure can be seen below.



First, full training data was fed to Random Forest and the top 50 features were kept as a list of selected features. Then, repeat this for nine more times. Finally, those features which showed themselves in all ten lists were chosen to compose the final feature list.

Out of 115 features, 44 features were selected to train the model:

Feature Name		Origin of the feature
duration src_bytes	dst_bytes wrong_fragment	Basic features of individual TCP connection
hot logged_in	num_compromise	Content features within a connection (suggested by domain knowledge)
http domain_u telnet eco_i ftp	ecr_i other urp_i private	Service
SF REJ RSTO	S0 RSTR	Flag
tcp udp	icmp	Protocol type
count srv_count serror_rate srv_serror_rate rerror_rate	srv_rerror_rate same_srv_rate diff_srv_rate srv_diff_host_rate	Time-based Traffic Features (suggested by domain knowledge)
dst_host_count dst_host_srv_count dst_host_same_srv_rate dst_host_diff_srv_rate dst_host_same_src_port_rate dst_host_srv_diff_host_rate dst_host_serror_rate dst_host_srv_rerror_rate dst_host_rerror_rate dst_host_srv_serror_rate		Host-based Traffic Features (suggested by domain knowledge)

Models

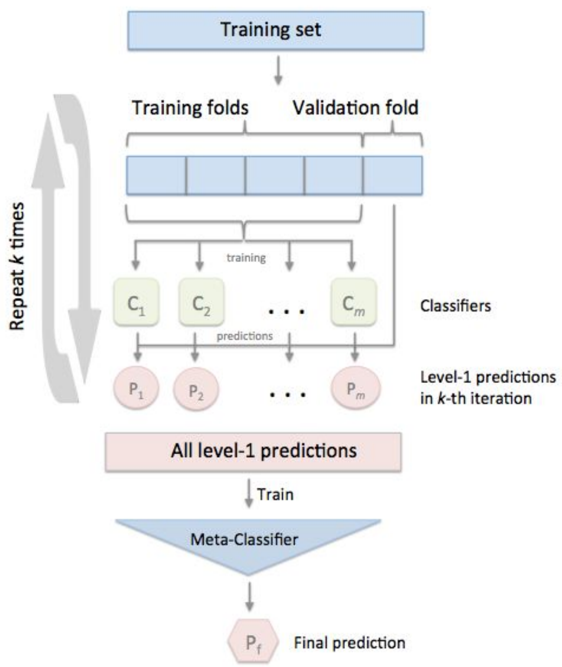
The are five model being trained and tuned in this project:

- Random Forest
- Extremely Randomised Trees
- Adaptive Boosting
- Ensemble Voting (soft-voting)
- Ensemble Stacking (Logistic Regression as meta-classifier)

The first two algorithms are a tree based algorithm that builds to enhance basic decision tree-based classification performance. The idea of improvement is similar: instead of constructing a big tree, how about grow many trees based on a subset of data (with replacement) from the dataset and then use a majority vote to decide the class. In this way, the outcome will still have a minimum bias (characteristics of tree-based classifier) but reduced variance.

The boosting algorithm works almost the same way, but it uses weighted model. In this way, it encourages new model to become an expert for instance that misclassified by the previous one.

When using voting and stacking method, all three models mentioned above were put into the training. Soft-voting was adopted so that model decision is built on probabilities with different weights. As for stacking, logistic regression was used as a meta-classifier. The stacking algorithm is showed in the following figure.



Each algorithm's hyperparameters then were tuned separately using GridSearch from sklearn.

Results

This is a multi-class classification task. In this competition, a cost-based score is calculated according to a cost matrix. Apparently, if a R2L or U2R attack is predicted to be a normal connection, it might cause more danger than a normal connection misclassified as illegitimate.

Cost Matrix		Predicted				
		0 - normal	*1 - probe*	*2 - DOS*	*3 - U2R*	*4 - R2L*
Actual	*0 - normal*	0	1	2	2	2
	1 - probe	1	0	2	2	2
	2 - DOS	2	1	0	2	2
	3 - U2R	3	2	2	0	2
	4 - R2L	4	2	2	2	0

Apart from a score, a prediction table is also helpful when researching on the results. The winning entry in 1999 got the following prediction table. Comparing to category 0, 1 and 2, it did poorly on 3 and 4.

Winning entry in KDD'99		Predicted					%correct (recall)
		0	*1*	*2*	*3*	*4*	
Actual	*0*	60262	243	78	4	6	99.5%
	1	511	3471	184	0	0	83.3%
	2	5299	1328	223226	0	0	97.1%
	3	168	20	0	30	10	13.2%
	4	14527	294	0	8	1360	8.4%
%correct (precision)		74.6%	64.8%	99.9%	71.4%	98.8%	
score		0.2331					

There are 5 models trained and tuned in this project, namely Random Forest, AdaBoost, Extra Trees, Voting and Stacking. Their results are listed below.

Results (Random Forest)		Predicted					%correct (recall)
		0	*1*	*2*	*3*	*4*	
Actual	*0*	60304	219	68	2	0	99.5%
	1	855	3129	182	0	0	75.1%
	2	6390	33	223430	0	0	97.2%
	3	219	2	0	3	4	1.3%
	4	15922	1	0	0	266	1.6%
%correct (precision)		72.1%	92.5%	99.9%	60%	98.5%	
score		0.2532					

Results (Adaptive Boosting)		Predicted					%correct (recall)
		0	*1*	*2*	*3*	*4*	
Actual	*0*	59220	579	224	247	323	97.7%
	1	385	3733	48	0	0	89.6%
	2	6089	41041	182722	0	1	79.5%
	3	119	55	2	44	8	19.3%
	4	15364	18	6	545	256	1.6%
%correct (precision)		73.0%	82.2%	99.8%	5.3%	43.5%	
score		0.3824					

Results (Extra Trees)		Predicted					%correct (recall)
		0	1	2	3	4	
Actual	0	60290	236	62	3	2	99.5%
	1	593	3300	273	0	0	79.2%
	2	6377	564	222912	0	0	97.0%
	3	217	4	0	3	4	1.3%
	4	15622	3	0	2	562	3.5%
%correct (precision)		72.6%	80.4%	99.9%	37.5%	98.9%	
score		0.2508					

Results (Voting)		Predicted					%correct (recall)
		0	1	2	3	4	
Actual	0	60286	230	73	3	1	99.5%
	1	474	3505	181	0	6	84.1%
	2	6306	288	223259	0	0	97.1%
	3	204	16	0	4	4	1.8%
	4	15806	3	0	3	377	2.3%
%correct (precision)		72.6%	86.7%	99.9%	40.0%	97.2%	
score		0.2508					

Results (Stacking)		Predicted					%correct (recall)
		0	1	2	3	4	
Actual	0	60291	233	69	0	0	99.5%
	1	657	3317	192	0	0	79.6%
	2	6179	252	223422	0	0	97.2%
	3	222	0	0	1	5	0.4%
	4	15984	2	0	1	202	1.2%
%correct (precision)		72.3%	87.2%	99.9%	50.0%	97.6%	
score		0.2528					

Discussions

As shown in the recall and precision table below, values worse than the winning entry are greyed out.

Recall	*0*	*1*	*2*	*3*	*4*
Winning in yr 99	0.995	0.833	0.971	0.132	0.084
Random Forest	0.995	0.751	0.972	0.013	0.016
Extra Tree	0.995	0.792	0.970	0.013	0.035
AdaBoost	0.977	0.896	0.795	0.193	0.016
Voting	0.995	0.841	0.971	0.018	0.023
Stacking	0.995	0.796	0.972	0.004	0.012

Precision	*0*	*1*	*2*	*3*	*4*
Winning in yr 99	0.746	0.647	0.999	0.714	0.988
Random Forest	0.721	0.925	0.999	0.600	0.985
Extra Tree	0.726	0.804	0.999	0.375	0.989
AdaBoost	0.730	0.822	0.998	0.053	0.435
Voting	0.726	0.867	0.999	0.400	0.972
Stacking	0.723	0.872	0.999	0.500	0.976

Model

- Extra trees and Voting are the best two models, but still a little worse than the winning entry.
- Random Forest and Extra Trees have similar results, though Extra Trees is slightly better. This also confirms their similarity in the algorithm.
- As a boosting algorithm, AdaBoost did well on minority class 1 at the sacrifice of other major classes. This may because the learning rate was set too large. If a more detailed hyper-parameter tuning was applied, the results might be better.
- Voting and stacking did take some good aspects of the other 3 models in class 1 precision, though not to the best extent. Meanwhile, they also avoid being too bad on class 1 recall.

Class

- Class 0 and 2, the two major classes in the training and test data, are predicted well throughout all models in this project. Especially class 2, it has both good precision and recall. But compare to the winning entry, our models are a little worse on class 0 precision.
- Out models are much better than the winning entry in class 1 precision. AdaBoost and Voting ares slightly better in recall as well. Other models sacrificed a bit on recall to achieve this good precision.
- Class 3 and 4 are not well predicted in the winning entry. Ours have even worse performance.

Related work

The dataset used in this project is a real dataset used for data mining competition in 1999. The winner of the contest used boosting algorithm with a twist in data subset generation (Pfahringer, 2000). Instead of a randomly generated subset of data, it used a particular part of training data that expensive if it misclassified. The approach seems like counterintuitive because it will increase the bias. However, the training data itself is distributed differently with testing data. It made the result of training from full dataset, without the modification in subset generation, will be biased too much to the training data and failed to classify correctly testing data.

This project does not follow this approach. That is why even though more advanced algorithm and method was applied, the result was not getting significantly better than the winner.

References

Pfahringer, B., 2000. Winning the KDD99 classification cup: bagged boosting. ACM SIGKDD Explorations Newsletter, 1(2), pp.65-66.

Appendix

Feature meanings

Stolfo et al. defined higher-level features that help in distinguishing normal connections from attacks. There are several categories of derived features.
The ``same host'' features examine only the connections in the past two seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.
The similar ``same service'' features examine only the connections in the past two seconds that have the same service as the current connection.
"Same host" and "same service" features are together called time-based traffic features of the connection records.
Some probing attacks scan the hosts (or ports) using a much larger time interval than two seconds, for example once per minute. Therefore, connection records were also sorted by destination host, and features were constructed using a window of 100 connections to the same host instead of a time window. This yields a set of so-called host-based traffic features.
Unlike most of the DOS and probing attacks, there appear to be no sequential patterns that are frequent in records of R2L and U2R attacks. This is because the DOS and probing attacks involve many connections to some host(s) in a very short period of time, but the R2L and U2R attacks are embedded in the data portions of packets, and normally involve only a single connection.
Useful algorithms for mining the unstructured data portions of packets automatically are an open research question. Stolfo et al. used domain knowledge to add features that look for suspicious behavior in the data portions, such as the number of failed login attempts. These features are called ``content'' features.

Basic features of individual TCP connections:

feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous

Content features within a connection suggested by domain knowledge:

feature name	*description*	*type*
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete

Traffic features computed using a two-second time window:

feature name	*description*	*type*
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Flag values

flags = ['SF' 'S2' 'S1' 'S3' 'OTH' 'REJ' 'RSTO' 'S0' 'RSTR' 'RSTOS0' 'SH']

Service values

```
# services = ['http' 'smtp' 'domain_u' 'auth' 'finger' 'telnet' 'eco_i' 'ftp' 'ntp_u'
# 'ecr_i' 'other' 'urp_i' 'private' 'pop_3' 'ftp_data' 'netstat' 'daytime'
# 'ssh' 'echo' 'time' 'name' 'whois' 'domain' 'mtp' 'gopher' 'remote_job'
# 'rje' 'ctf' 'supdup' 'link' 'systat' 'discard' 'X11' 'shell' 'login'
# 'imap4' 'nntp' 'uucp' 'pm_dump' 'IRC' 'Z39_50' 'netbios_dgm' 'ldap'
# 'sunrpc' 'courier' 'exec' 'bgp' 'csnet_ns' 'http_443' 'klogin' 'printer'
# 'netbios_ssn' 'pop_2' 'nnsp' 'efs' 'hostnames' 'uucp_path' 'sql_net'
# 'vmnet' 'iso_tsap' 'netbios_ns' 'kshell' 'urh_i' 'http_2784' 'harvest'
# 'aol' 'tftp_u' 'http_8001' 'tim_i' 'red_i']
```