Classification using Naive Bayes

From Distributed Data Mining Winter Semester 2015-2016

Contents

- 1 classification with Naive Bayes
  - 1.1 Getting the data and creating the RDD
  - 1.2 Preparing the training data
  - 1.3 Detecting network attacks using Naive Bayes
    - 1.3.1 Training a classifier
    - 1.3.2 Using hypothesis testing
  - 1.4 Output

classification with Naive Bayes

Naive Bayes is a simple multiclass classification algorithm. It can be trained efficiently, within a single pass of training data, it computes a conditional probability distribution of each feature given label and then apply a bayes theorem to compute the conditional probability of label given an observation and use it for prediction.NaiveBayes implements multinomial naive Bayes. It takes an RDD of LabeledPoint and an optionally smoothing parameter lambda as input, and output a NaiveBayesModel, which can be used for evaluation and prediction.

Again we will use the KDD Cup 1999 complete datasets in order to test Spark capabilities with large datasets.Simple mathematical explanation of naive bayes can be found here (http://spark.apache.org/docs/latest/mllib-naive-bayes.html/) spark.mllib supports multinomial naive Bayes and Bernoulli naive Bayes. These models are typically used for document classification.

Getting the data and creating the RDD

We will use urllib to get the data directly and use it for implementation.

```
import urllib
f = urllib.urlretrieve ("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data.gz", "kddcup.data.gz")
data_file = "./kddcup.data.gz"
raw_data = sc.textFile(data_file)
print "Train data size is {}".format(raw_data.count())
```

We will also use 10% of the sample data for testing and load it in separate RDD.

Preparing the training data

For this data we are interested in detecting the networking attacks, hence we label each network interaction as non attack (i.e. 'normal' tag) or attack (i.e. anything else but 'normal').

```
from pyspark.mllib.regression import LabeledPoint
from numpy import array
def parse_interaction(line):
   line_split = line.split(",")
   # leave_out = [1,2,3,41]
   clean_line_split = line_split[0:1]+line_split[4:41]
   attack = 1.0
   if line_split[41]=='normal.':
       attack = 0.0
   return LabeledPoint(attack, array([float(x) for x in clean_line_split]))
```

```
training_data = raw_data.map(parse_interaction)
```

Similarly we prepare the test data:

```
test_data = test_raw_data.map(parse_interaction)
```

Detecting network attacks using Naive Bayes

We use Naive bayes to predict the binary response. spark.mllib supports multinomial naive Bayes and Bernoulli naive Bayes. These models are typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term (in multinomial naive Bayes) or a zero or one indicating whether the term was found in the document (in Bernoulli naive Bayes). Feature values must be nonnegative.

Training a classifier

```
from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
from time import time
```

```
# Build the model
t0 = time()
model = NaiveBayes.train(training_data)
tt = time() - t0
```

`print "Classifier trained in {} seconds".format(round(tt,3))

Using hypothesis testing

Hypothesis testing is a powerful tool in statistical inference and learning to determine whether a result is statistically significant. We use RDD[LabeledPoint] to enable feature selection via chi-squared independence tests.We want to perform some sort of feature selection.Features need to be categorical. Real-valued features will be treated as categorical in each of its different values.we will consider just features that either take boolean values or just a few different numeric values in our dataset. We could overcome this limitation by defining a more complex parse_interaction function that categorises each feature properly.

```
feature_names = ["land","wrong_fragment",
            "urgent","hot","num_failed_logins","logged_in","num_compromised",
            "root_shell","su_attempted","num_root","num_file_creations",
            "num_shells","num_access_files","num_outbound_cmds",
            "is_hot_login","is_guest_login","count","srv_count","serror_rate",
            "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
            "diff_srv_rate","srv_diff_host_rate","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_serror_rate",
            "dst_host_rerror_rate","dst_host_srv_rerror_rate"]
```

```
def parse_interaction_categorical(line):
   line_split = line.split(",")
   clean_line_split = line_split[6:41]
   attack = 1.0
   if line_split[41]=='normal.':
       attack = 0.0
   return LabeledPoint(attack, array([float(x) for x in clean_line_split]))
```

```
training_data_categorical = raw_data.map(parse_interaction_categorical)
```

```
from pyspark.mllib.stat import Statistics
```

```
chi = Statistics.chiSqTest(training_data_categorical)
```

Now we can check the resulting values after putting them into a Pandas data frame.

```
import pandas as pd
pd.set_option('display.max_colwidth', 30)
records = [(result.statistic, result.pValue) for result in chi]
chi_df = pd.DataFrame(data=records, index= feature_names, columns=["Statistic","p-value"])
```

Output

i have used 1 masters and 2 slaves to perform the test. The 10% of the data is used for the training the model and remaining 90 for testing . The output of the scripts is as follows:

```
Train data size is 494021
```

```
Test data size is 311029
```

```
Classifier trained in 21.965 seconds
```

```
Prediction made in 27.117 seconds. Test accuracy is 0.9182
```

For feature selection :

	Statistic	p-value
land	3.191410	7.402616e-02
wrong_fragment	304.309631	0.000000e+00
urgent	4.814020	1.859331e-01
hot	1939.879629	0.000000e+00
num_failed_logins	24.935016	1.434174e-04
logged_in	312455.306674	0.000000e+00
num_compromised	635.549407	0.000000e+00
root_shell	17.030507	3.678404e-05
su_attempted	39.839347	2.233550e-09
num_root	2277.286802	0.000000e+00
num_file_creations	821.828337	0.000000e+00
num_shells	149.023961	0.000000e+00
num_access_files	1753.117284	0.000000e+00
num_outbound_cmds	0.000000	1.000000e+00
is_hot_login	0.000000	1.000000e+00
is_guest_login	515.383504	0.000000e+00
count	438656.520829	0.000000e+00
srv_count	221348.426646	0.000000e+00
serror_rate	26650.209505	0.000000e+00
srv_serror_rate	30009.256774	0.000000e+00
rerror_rate	747.695493	0.000000e+00
srv_rerror_rate	2227.301678	0.000000e+00
same_srv_rate	39218.869964	0.000000e+00
diff_srv_rate	38481.946673	0.000000e+00
srv_diff_host_rate	135374.306837	0.000000e+00
dst_host_count	245057.453656	0.000000e+00
dst_host_srv_count	136994.054761	0.000000e+00
dst_host_same_srv_rate	120218.197644	0.000000e+00
dst_host_diff_srv_rate	129405.698322	0.000000e+00
dst_host_same_src_port_rate	285015.925542	0.000000e+00
dst_host_srv_diff_host_rate	221692.890239	0.000000e+00
dst_host_serror_rate	37825.124327	0.000000e+00
dst_host_srv_serror_rate	42916.904442	0.000000e+00
dst_host_rerror_rate	8805.320953	0.000000e+00
dst_host_srv_rerror_rate	21137.696659	0.000000e+00