

Logistic regression

In [96]:

```
import urllib

data_file = "data/kddcup.data.gz"
# f = urllib.urlretrieve ("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.da

raw_data = sc.textFile(data_file)

print "Train data size is %d" % raw_data.count()
```

Train data size is 4898431

In [97]:

```
raw_data = raw_data.sample(False, 0.01)
print "Train data size is %d" % raw_data.count()
```

Train data size is 49257

In [98]:

```
raw_data.take(3)
```

Out[98]:

```
[u'0,tcp,http,SF,236,1228,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0.0
0,0.00,0.00,0.00,1.00,0.00,0.00,2,2,1.00,0.00,0.50,0.00,0.00,0.0
0,0.00,0.00,normal.',
 u'0,tcp,http,SF,222,861,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,5,5,0.0
0,0.00,0.00,0.00,1.00,0.00,0.00,32,123,1.00,0.00,0.03,0.04,0.0
0,0.00,0.00,0.00,normal.',
 u'0,tcp,http,SF,232,1661,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.0
0,0.00,0.00,0.00,1.00,0.00,0.00,65,200,1.00,0.00,0.02,0.03,0.0
0,0.00,0.00,0.00,normal.']
```

In [99]:

```
test_data_file = "data/corrected.gz"

# ft = urllib.urlretrieve("http://kdd.ics.uci.edu/databases/kddcup99/corrected

test_raw_data = sc.textFile(test_data_file)

print "Test data size is %d" % test_raw_data.count()
```

Test data size is 311029

In [100]:

```
test_raw_data = test_raw_data.sample(False, 0.01)
print "Test data size is %d" % test_raw_data.count()
```

Test data size is 3158

In [13]:

```
raw_data.take(3)
```

Out[13]:

```
[u'0,tcp,http,SF,184,124,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0.0
0,0.00,0.00,0.00,1.00,0.00,0.00,10,10,1.00,0.00,0.10,0.00,0.0
0,0.00,0.00,0.00,normal.',
 u'0,tcp,http,SF,165,3600,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,7,7,0.0
0,0.00,0.00,0.00,1.00,0.00,0.00,7,243,1.00,0.00,0.14,0.03,0.0
0,0.00,0.00,0.00,normal.',
 u'0,tcp,http,SF,248,2129,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,23,2
3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,23,255,1.00,0.00,0.04,0.0
3,0.00,0.00,0.00,0.00,normal.']
```

In [101]:

```
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)
```

In [102]:

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

In [103]:

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from time import time

# Build the model
t0 = time()
logit_model = LogisticRegressionWithLBFGS.train(training_data)
tt = time() - t0

print "Classifier trained in %.3f seconds" % round(tt, 3)
```

Classifier trained in 38.384 seconds

In [104]:

```
labels_and_preds = test_data.map(lambda p: (p.label, logit_model.predict(p.fea
```

In [105]:

```
t0 = time()
test_accuracy = labels_and_preds.filter(lambda (v, p): v == p).count() / float
tt = time() - t0

print "Prediction made in %.3f seconds. Test accuracy is %.4f" % (round(tt,3),
Prediction made in 4.810 seconds. Test accuracy is 0.9158
```

In [25]:

```
def parse_interaction_corr(line):
    line_split = line.split(",")
    # leave_out = [1,2,3,25,27,35,38,40,41]
    clean_line_split = line_split[0:1]+line_split[4:25]+line_split[26:27]+line
    attack = 1.0
    if line_split[41]=='normal.':
        attack = 0.0
    return LabeledPoint(attack, array([float(x) for x in clean_line_split]))

corr_reduced_training_data = raw_data.map(parse_interaction_corr)
corr_reduced_test_data = test_raw_data.map(parse_interaction_corr)
```

In [26]:

```
# Build the model
t0 = time()
logit_model_2 = LogisticRegressionWithLBFGS.train(corr_reduced_training_data)
tt = time() - t0

print "Classifier trained in %.3f seconds" % round(tt, 3)

Classifier trained in 38.202 seconds
```

In [27]:

```
labels_and_preds = corr_reduced_test_data.map(lambda p: (p.label, logit_model_
t0 = time()
test_accuracy = labels_and_preds.filter(lambda (v, p): v == p).count() / float
tt = time() - t0

print "Prediction made in %.3f seconds. Test accuracy is %.4f" % (round(tt, 3)
Prediction made in 4.952 seconds. Test accuracy is 0.8630
```

In [28]:

```
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)
```

In [36]:

```
import pandas as pd
pd.set_option('display.max_colwidth', 30)

records = [(round(result.statistic, 3), round(result.pValue, 4)) for result in
chi_df = pd.DataFrame(data=records, index= feature_names, columns=["Statistic"
chi_df
```

Out[36]:

	Statistic	p-value
land	0.000	1.0000
wrong_fragment	2.258	0.3233
urgent	0.000	1.0000
hot	183.810	0.0000
num_failed_logins	0.502	0.4787
logged_in	32608.635	0.0000
num_compromised	42.775	0.0000
root_shell	15.946	0.0001
su_attempted	7.972	0.0047
num_root	263.435	0.0000
num_file_creations	119.655	0.0000
num_shells	3.986	0.0459
num_access_files	199.507	0.0000
num_outbound_cmds	0.000	1.0000
is_hot_login	0.000	1.0000
is_guest_login	141.832	0.0000
count	45473.768	0.0000
srv_count	23466.490	0.0000
serror_rate	2795.171	0.0000
srv_serror_rate	3099.628	0.0000
rerror_rate	127.221	0.0000
srv_rerror_rate	304.608	0.0000
same_srv_rate	4028.799	0.0000
diff_srv_rate	3975.083	0.0000
srv_diff_host_rate	13813.927	0.0000
dst_host_count	25169.413	0.0000
dst_host_srv_count	14501.481	0.0000
dst_host_same_srv_rate	12526.973	0.0000
dst_host_diff_srv_rate	13359.883	0.0000
dst_host_same_src_port_rate	29002.515	0.0000
dst_host_srv_diff_host_rate	22032.865	0.0000
dst_host_serror_rate	4157.533	0.0000
dst_host_srv_serror_rate	4570.070	0.0000

dot best server rate	1500.000	0.0000
----------------------	----------	--------

In [37]:

```
def parse_interaction_chi(line):
    line_split = line.split(",")
    # leave_out = [1,2,3,6,19,41]
    clean_line_split = line_split[0:1] + line_split[4:6] + line_split[7:19] +
    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_chi = raw_data.map(parse_interaction_chi)
test_data_chi = test_raw_data.map(parse_interaction_chi)
```

In [38]:

```
# Build the model
t0 = time()
logit_model_chi = LogisticRegressionWithLBFGS.train(training_data_chi)
tt = time() - t0

print "Classifier trained in %.3f seconds" % round(tt, 3)
```

Classifier trained in 37.947 seconds

In [39]:

```
labels_and_preds = test_data_chi.map(lambda p: (p.label, logit_model_chi.predict(p)))
t0 = time()
test_accuracy = labels_and_preds.filter(lambda (v, p): v == p).count() / float(test_data_chi.count())
tt = time() - t0

print "Prediction made in %.3f seconds. Test accuracy is %.4f" % (round(tt, 3), test_accuracy)
```

Prediction made in 4.814 seconds. Test accuracy is 0.9122

Linear Regression

In [85]:

```
import sys
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from numpy import array

def parse_point(line):
    values = [float(x) for x in line.replace(',', ' ').split(' ')]
    return LabeledPoint(values[0], values[1:])
```

In [86]:

```
data = sc.textFile("data/lpsa.data")
parsed_data = data.map(parse_point)
```

In [89]:

```
parsed_data.count()
```

Out[89]:

67

In [88]:

```
t0 = time()
model = LinearRegressionWithSGD.train(parsed_data)
tt = time() - t0

print "Regressor trained in %.3f seconds" % round(tt, 3)
```

Regressor trained in 1.846 seconds

In [93]:

```
parsed_data.take(1)[0].label
```

Out[93]:

-0.4307829

In [91]:

```
model.predict([-1.63735562648,-2.00621178481,-1.86242597251,-1.02470580167,-0.
```

Out[91]:

-1.8431929735178869

In [94]:

```
values_and_preds = parsed_data.map(lambda p: (p.label, model.predict(p.feature
mse = values_and_preds.map(lambda(v, p): (v - p)**2).reduce(lambda x, y: x+y)
print "Mean squared error is %.3f" % mse
```

Mean squared error is 6.207

In [95]:

```
values_and_preds.take(8)
```

Out[95]:

```
[(-0.4307829, -1.8431929735166417),  
(-0.1625189, -1.5069898510647171),  
(-0.1625189, -1.1282811232397254),  
(-0.1625189, -1.6277430447022609),  
(0.3715636, -0.4362902788834806),  
(0.7654678, -1.8759241492991019),  
(0.8544153, -0.34539227123181521),  
(1.2669476, -0.49560851400156936)]
```

Support vector machines

In [72]:

```
from pyspark.mllib.classification import SVMWithSGD  
from pyspark.mllib.evaluation import BinaryClassificationMetrics  
from pyspark.mllib.regression import LabeledPoint  
from pyspark.mllib.linalg import Vectors  
from pyspark.mllib.util import MLUtils  
  
data = MLUtils.loadLibSVMFile(sc, "data/sample_libsvm_data.txt")
```

In [73]:

```
splits = data.randomSplit((0.6, 0.4))  
training = splits[0].cache()  
test = splits[1]
```

In [74]:

```
num_iterations = 100  
  
t0 = time()  
model = SVMWithSGD.train(training, num_iterations)  
tt = time() - t0  
  
print "SVM trained in %.3f seconds" % round(tt, 3)
```

SVM trained in 1.756 seconds

In [75]:

```
model.clearThreshold()  
score_and_labels = test.map(lambda p: (model.predict(p.features), p.label))
```

In [76]:

```
score_and_labels.map(lambda (p, v): (p - v)**2).reduce(lambda x, y: x+y)
```

Out[76]:

4.0

In []: