machine learning

May 16, 2016

In [1]:
```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Make the graphs a bit prettier, and bigger
pd.set_option('display.mpl_style', 'default')
plt.rcParams['figure.figsize'] = (15, 5)
plt.rcParams['font.family'] = 'sans-serif'

# This is necessary to show lots of columns in pandas 0.12.
# Not necessary in pandas 0.13.
pd.set_option('display.width', 5000)
pd.set_option('display.max_columns', 60)
```

Load file sampled from data in auth.txt.gz so that number of fails is similar to success

In [3]:
```
df=pd.read_csv('md/msample1.csv', header=None)
```

In [4]:
```
len(df)
```

Out[4]:
```
400306
```

In [5]:
```
df[8].value_counts()
```

Out[5]:
```
Success    200261
Fail       200045
Name: 8, dtype: int64
```

Creating clsssification label

In [6]:
```
Y=(df[8]=='Success')
```

Creating features for machine learning: columns 5-7 from df for authentication type, logon type and authentication orientation are expanded to include all labels from the columns as new expanded columns holding 1(True) if the label applies and 0 (False) otherwise. columns 2-4 used to create columns that test whether source and destination computer are the same, source user is same as source computer, etc.

In [32]:
```
def map_user(x):
    if x.startswith('C'):
        return 'C'
    elif x.startswith('U'):
        return 'U'
    else:
        return x
```

In [33]:
```
df['source_class']=df['source_user'].map(map_user)
df['destination_class']=df['destination_user'].map(map_user)
df["source_user"], df["source_domain"] = zip(*df[1].str.split('@').tolist())
df["source_user"]=df["source_user"].str.rstrip('$')
df["destination_user"], df["destination_domain"] = zip(*df[2].str.split('@').tolist())
df["destination_user"]=df["destination_user"].str.rstrip('$')
X=pd.DataFrame.from_items([('time', (df[0]%(24*60*60)).astype(int))])
X['same_user']= (df['destination_user']==df['source_user'])
X['same_domain']=(df['destination_domain']==df['source_domain'])
X['source_user_comp_same']=(df[3]==df['source_user'])
X['destination_user_comp_same']=(df['destination_user']==df[4])
X['same_comp']=(df[3]==df[4])
X['source_domain_comp_same']=(df[3]==df['source_domain'])
X['destination_domain_comp_same']=(df['destination_domain']==df[4])

for j in [5,6, 7]:
    for label in sorted(df[j].unique()):
        if label=='?':
            if j==5:
                X['?_authentication type']=(df[j]==label)
            elif j==6:
                X['?_logon type']=(df[j]==label)
        else:
            X[label]=(df[j]==label)
for cl in ['source_class', 'destination_class']:
    for label in df[cl].unique():
        if cl=='source_class':
            X['source_'+label]=(df[cl]==label)
        else:
            X['destination_'+label]=(df[cl]==label)
```

In [34]:
```
X
```

Out[34]:

	time	same_user	same_domain	source_user_comp_same	destination_user_comp_same	same_comp	so
0	2	True	True	False	True	False	
1	3	True	True	False	False	True	
2	11	True	True	False	False	False	
3	140	True	True	True	False	False	
4	176	True	True	False	False	True	
5	185	True	True	False	False	True	
6	224	True	True	True	False	False	
7	250	True	True	False	False	True	
8	252	True	True	False	False	False	
9	333	True	True	True	True	True	
10	348	True	True	False	True	False	
11	416	True	True	False	True	False	
12	459	True	True	True	True	True	
13	470	True	True	True	True	True	
14	485	True	True	False	False	False	
15	490	True	True	True	False	False	
16	510	True	True	False	True	False	
17	542	True	True	False	False	True	
18	551	True	True	True	False	False	
19	570	True	True	False	False	True	
20	588	True	True	False	True	False	

```
21          623     True    True                False                   True    False
22          679     True    True                False                   True    False
23          704     True    True                False                  False     True
24          726     True    True                False                  False     True
25          745     True    True                 True                  False    False
26          750     True    True                False                  False     True
27          859     True    True                False                  False     True
28          936     True    True                False                  False     True
29          975     True    True                False                  False     True
...         ...      ...     ...                  ...                    ...      ...
400276    86055     True    True                False                  False    False
400277    86057     True    True                False                  False     True
400278    86061     True    True                 True                   True     True
400279    86073     True    True                False                  False     True
400280    86089     True    True                False                  False     True
400281    86093     True    True                False                  False    False
400282    86104     True    True                False                  False     True
400283    86127     True    True                False                  False     True
400284    86131     True    True                False                  False     True
400285    86151     True    True                False                  False     True
400286    86179     True    True                 True                  False    False
400287    86251     True    True                False                  False     True
400288    86259     True    True                False                  False     True
400289    86267     True    True                 True                   True     True
400290    86267     True    True                False                  False     True
400291    86275     True    True                False                  False     True
400292    86280     True    True                False                  False     True
400293    86281     True    True                False                  False     True
400294    86281     True    True                False                   True    False
400295    86287     True    True                False                  False     True
400296    86298     True    True                False                  False     True
400297    86341     True    True                False                  False     True
400298    86347     True    True                False                  False     True
400299    86354     True    True                False                  False     True
400300    86356     True    True                False                  False     True
400301    86372     True    True                False                  False     True
400302    86373     True    True                False                  False     True
400303    86374     True    True                False                  False     True
400304    86391     True    True                False                  False     True
400305    86393     True    True                False                  False    False

[400306 rows x 56 columns]
```

Separate current dataset into train and test data

In [35]:
```python
n=int(len(X)*.7)
Xtrain=X[:n]
Ytrain=Y[:n]
Xtest=X[n:]
Ytest=Y[n:]
```

1 Logistic regression

In [36]:
```python
from sklearn import linear_model, datasets
logreg = linear_model.LogisticRegression(C=1e5).fit(Xtrain, Ytrain)
```

```
In [37]: print logreg.score(Xtrain, Ytrain), logreg.score(Xtest, Ytest)
```

```
0.845849957532 0.870082936415
```

```
In [38]: from sklearn.metrics import confusion_matrix
         trainPred=logreg.predict(Xtrain)
         testPred=logreg.predict(Xtest)
         print confusion_matrix(Ytrain, trainPred)
         confusion_matrix(Ytest, testPred)
```

```
[[ 98017  27581]
 [ 15614 139002]]
```

```
Out[38]: array([[63244, 11203],
                [ 4399, 41246]])
```

Coefficients for logistic regression should tell which parameters are important

```
In [39]: logreg.coef_
```

```
Out[39]: array([[ -3.32691813e-06,   1.73652260e-01,   1.02334575e-01,
                 -8.53431039e-02,  -1.71384450e-01,  -1.40186009e-01,
                 -8.31512032e-03,   1.84727423e-01,   2.47089330e-01,
                 -3.18469345e-05,   4.77905527e-01,   1.74150387e-05,
                  1.81851633e-05,   0.00000000e+00,  -1.07659982e-04,
                 -2.99574704e-04,  -9.01649007e-05,  -2.70560804e-04,
                 -7.17526311e-04,   3.03624207e-07,  -2.02790002e-03,
                 -2.52719736e-01,  -5.86351682e-04,  -3.42599686e-01,
                 -5.89664843e-03,  -8.03137146e-05,  -2.19807461e-04,
                 -7.97512954e-01,  -4.73433942e-02,  -2.20369663e-03,
                 -1.03100534e-02,   9.04248405e-01,   4.90831960e-04,
                  3.89683819e-04,  -2.08136298e-03,   1.14808300e-01,
                 -4.11027758e-02,   4.72027787e-02,   1.04460228e+00,
                 -1.27290486e-01,   8.03091834e-04,   3.67455912e-04,
                  2.77403964e-01,  -1.12370610e+00,  -1.47702535e-01,
                  8.83690991e-02,   1.02245873e-02,   8.23869142e-02,
                  8.82213234e-02,  -2.11640471e-03,  -2.49416013e-01,
                  1.72531468e-01,   1.04951942e-02,   8.23869142e-02,
                  8.87410314e-02,   1.46443885e-02]])
```

```
In [40]: X.columns
```

```
Out[40]: Index([u'time', u'same_user', u'same_domain', u'source_user_comp_same', u'destination_user_comp_s
```

2 Try L1 penanlty

```
In [41]: clf_l1_LR = linear_model.LogisticRegression(C=1000, penalty='l1', tol=0.001).fit(Xtrain, Ytrai
         print clf_l1_LR.score(Xtrain, Ytrain), clf_l1_LR.score(Xtest, Ytest)
```

```
0.93957118488 0.94798154748
```

3 Try L2 penalty

```
In [42]: clf_l2_LR = linear_model.LogisticRegression(C=1000, penalty='l2', tol=0.001).fit(Xtrain, Ytrai
         print clf_l2_LR.score(Xtrain, Ytrain), clf_l2_LR.score(Xtest, Ytest)
```

```
0.551781852441 0.380091929521
```

4 Gradient Boosting

```
In [43]: from sklearn.ensemble import GradientBoostingClassifier
         clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.05, max_depth=1, random_sta
         print clf.score(Xtrain, Ytrain), clf.score(Xtest, Ytest)
```

```
0.889084770925 0.880649835126
```

5 Analysis

From the results I just got, I can see that Logistic regression with L1 penalty works better than Gradient Boosting than logistic regression without any normalization than logistic regression with L2 penalty.

Lasso logistic regression (L1 penalty) works really well for correlated features, whereas L2 penalty fails badly when features are correlated. Given how I constructed my features, they can easily turn out to be correlated, but I probably want spent more time on understanding correlations between features as Lasso gives really good accuracy score.

At this point, I do not know if I should trust my result as I tested it on a very small subset of data from auth.txt.gz. I'd like to get more independent randomly sampled subsets to see how well my results hold on.

```
In [44]: clf_l1_LR.coef_
```

```
Out[44]: array([[ -7.50457104e-07,   1.32847712e-01,   3.78732145e-01,
                  -7.13165785e-01,  -2.28256820e+00,   1.48145803e+00,
                  -2.76890051e+00,   2.23519986e-01,   2.58836292e-02,
                  -6.66344962e+00,   9.75513636e-01,   5.81266499e+00,
                   5.79483716e+00,   0.00000000e+00,  -9.10379549e+00,
                  -1.89396464e+00,  -1.38363865e+00,  -3.18072600e+00,
                  -1.09602377e+01,  -4.12067808e-01,  -5.14550659e+00,
                  -8.62836074e+00,  -1.29707104e+01,  -2.08497456e+00,
                  -2.77111489e+00,  -1.10213812e+01,  -1.18760992e+01,
                  -7.80461599e-01,  -1.37218556e+00,  -1.44158909e+00,
                   4.23947918e-01,   7.09622148e-01,   3.18630133e+00,
                  -3.04568171e-01,   4.67773287e-01,   2.96238282e+00,
                   2.73577682e-01,   1.26674201e+01,   8.43775338e+00,
                  -6.14667759e-02,   9.03515422e+00,   8.28870634e+00,
                   2.02242224e+00,  -3.95490524e+00,  -1.88803251e-01,
                   2.08150387e-01,   5.31432751e+00,   2.16673559e+00,
                   9.85399450e-01,  -4.84371857e+00,  -2.56466695e-01,
                   4.58664283e-01,   6.21752178e+00,   8.48711216e+00,
                   1.02330574e+01,   6.41755199e+00]])
```

```
In [46]: pd.DataFrame.from_items([("feature",X.columns), ("LR contribution",clf_l1_LR.coef_[0]*100)])
```

Out[46]:

	feature	LR contribution
0	time	-0.000075
1	same_user	13.284771
2	same_domain	37.873214
3	source_user_comp_same	-71.316578
4	destination_user_comp_same	-228.256820
5	same_comp	148.145803
6	source_domain_comp_same	-276.890051
7	destination_domain_comp_same	22.351999
8	?_authentication type	2.588363
9	ACRONIS_RELOGON_AUTHENTICATION_PACKAGE	-666.344962
10	Kerberos	97.551364

11	MICROSOFT_AUTHENTICATION_PA	581.266499
12	MICROSOFT_AUTHENTICATION_PAC	579.483716
13	MICROSOFT_AUTHENTICATION_PACK	0.000000
14	MICROSOFT_AUTHENTICATION_PACKA	-910.379549
15	MICROSOFT_AUTHENTICATION_PACKAG	-189.396464
16	MICROSOFT_AUTHENTICATION_PACKAGE	-138.363865
17	MICROSOFT_AUTHENTICATION_PACKAGE_	-318.072600
18	MICROSOFT_AUTHENTICATION_PACKAGE_V	-1096.023773
19	MICROSOFT_AUTHENTICATION_PACKAGE_V1	-41.206781
20	MICROSOFT_AUTHENTICATION_PACKAGE_V1_	-514.550659
21	MICROSOFT_AUTHENTICATION_PACKAGE_V1_0	-862.836074
22	NETWARE_AUTHENTICATION_PACKAGE_V1_0	-1297.071042
23	NTLM	-208.497456
24	Negotiate	-277.111489
25	Setuid	-1102.138117
26	Wave	-1187.609923
27	?_logon type	-78.046160
28	Batch	-137.218556
29	CachedInteractive	-144.158909
30	Interactive	42.394792
31	Network	70.962215
32	NetworkCleartext	318.630133
33	NewCredentials	-30.456817
34	RemoteInteractive	46.777329
35	Service	296.238282
36	Unlock	27.357768
37	AuthMap	1266.742014
38	LogOff	843.775338
39	LogOn	-6.146678
40	ScreenLock	903.515422
41	ScreenUnlock	828.870634
42	TGS	202.242224
43	TGT	-395.490524
44	source_U	-18.880325
45	source_C	20.815039
46	source_LOCAL SERVICE	531.432751
47	source_ANONYMOUS LOGON	216.673559
48	source_NETWORK SERVICE	98.539945
49	source_SYSTEM	-484.371857
50	destination_U	-25.646670
51	destination_C	45.866428
52	destination_LOCAL SERVICE	621.752178
53	destination_ANONYMOUS LOGON	848.711216
54	destination_NETWORK SERVICE	1023.305735
55	destination_SYSTEM	641.755199

Some comments: table above shows relative contribution of each feature to the final prediction. Features 11-21 are potentially the same thing. I will not do anything about it for now because I am getting good scores, but this is something one can look into to see if it causes problems.

```
In [ ]:
```

```
In [ ]:
```