Linear Model 3 - Linear SVM

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from auxiliars import *
import pickle
```

Data

In [2]:
```
data = pd.read_csv("./data/stdHTRU_2.csv")
```

In [3]:
```
col = data['class'].map({1:'r', 0:'b'})
pd.plotting.scatter_matrix(data.drop(['class'], axis = 1), c=col, f
igsize=(15,15))
```

Out[3]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1153c4b
38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116875b
70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1168b60
48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1168e75
f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116917b
a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1169571
98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1169857
48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1169b6d
30>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1169b6d
68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116a258
98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116a57e
48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116a924
38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116ac59
e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116af6f
98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116b335
88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116b64b
38>],
```

```
       [<matplotlib.axes._subplots.AxesSubplot object at 0x116ba21
28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116bd16
d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116c05c
88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116c412
78>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116c708
28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116ca2d
d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116ce03
c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116d109
78>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x116d43f
28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116d805
18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116db0a
c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116ded0
b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116e1d6
68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116e4fc
18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116e8d2
08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116ebd7
b8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x116ef1d
68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116f2d3
58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116f5b9
08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116f91e
b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116fcb4
a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x116ffba
58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1170400
48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11706a5
f8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11709bb
a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1192441
98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1192787
48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1192a6c
f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1192e32
```

```
e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1193168
98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119347e
48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1193844
38>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1193b49
e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1193e7f
98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1194245
88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119454b
38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1194931
28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1194c26
d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1194f2c
88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1195302
78>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1195628
28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119591d
d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1195ce3
c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1196009
78>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x119631f
28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1196715
18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1196a1a
c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1196df0
b8>]],
      dtype=object)
```

From the Scatter Matrix we can de deduce that the Linear Kernel should be sufficient for the separation of classes.

Even so, we can obvserve that some features, see for example DM_mean-DM_stdev, have very close data. In order to reduce the impact of this fact, let's train SVM with (standarized) normal data and data with selected features.

We split a separate test set of relative size 20%:

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(data[data.colum
        ns[0:8]],
                                                                data['class'],
                                                                test_size = 0.2
        ,
                                                                random_state =
        1234)
```

We will analyze the performance of the method with no-correlated standarized data:

```
In [5]: noCorrData = pd.read_csv("./data/noCorrStdHTRU_2.csv")
```

```
In [6]: X_train_NC, X_test_NC, y_train_NC, y_test_NC = train_test_split(noC
        orrData[noCorrData.columns[0:6]],
                                                                noCorrData['cla
        ss'],
                                                                test_size = 0.2
        ,
                                                                random_state =
        1234)
```

Model Training

In order to train Linear SVM we are going to use the scikit-learn LinearSVC class, specialized in Linear SVM.

```
In [7]: from sklearn.svm import LinearSVC
```

```
In [8]: SVMClass = LinearSVC(random_state = 1234, max_iter = 5000)
```

LinearSVC allow us to hypertuning the following parameters:

- Regularization parameter C.
- Class weights:
  - Dict: Weights specified by class.
  - Balanced: Uses the values of target (y) to automatically adjust weights inversely proportional to class frequencies in the input data.

In order to hypertuning model parameters and get a better idea on how the model performs on unseen data, we will use GridSearchCV.

```
In [9]: from sklearn.model_selection import GridSearchCV
```

Values of the 10-Fold CV Grid to test:

In [10]:
```
grid = {'C': [10**x for x in range(-3, 4, 1)],
        'class_weight': [{0: 1, 1: 1}, 'balanced']}
```

In [11]:
```
grid
```

Out[11]:
```
{'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
 'class_weight': [{0: 1, 1: 1}, 'balanced']}
```

Grid Search 10-Fold CV:

In [12]:
```
gs10cv = GridSearchCV(SVMClass, param_grid = grid, cv = 10, n_jobs
= -1)
```

Normal Data Training

In [13]:
```
gs10cv.fit(X_train, y_train)
```

Out[13]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=LinearSVC(C=1.0, class_weight=None, dual=Tr
ue,
                                 fit_intercept=True, intercept_sca
ling=1,
                                 loss='squared_hinge', max_iter=50
00,
                                 multi_class='ovr', penalty='l2',
                                 random_state=1234, tol=0.0001, ve
rbose=0),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
,
                         'class_weight': [{0: 1, 1: 1}, 'balanced'
]},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
             scoring=None, verbose=0)
```

In [14]:
```
pd.DataFrame(gs10cv.cv_results_)
```

Out[14]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param
0	0.067043	0.015112	0.005495	0.005022	0.001	{0: 1,
1	0.129222	0.038246	0.005577	0.003026	0.001	balanc

2	0.106115	0.053240	0.020416	0.045160	0.01	{0: 1, 1
3	0.569645	0.281467	0.003367	0.000886	0.01	balanc
4	0.724118	0.322835	0.004992	0.002068	0.1	{0: 1, 1
5	3.026904	0.825948	0.008067	0.015337	0.1	balanc
6	3.008573	0.900897	0.007850	0.007868	1	{0: 1, 1
7	16.433442	2.413215	0.007260	0.003504	1	balanc
8	6.205535	0.231656	0.003481	0.000471	10	{0: 1, 1
9	14.519821	0.568743	0.003143	0.000644	10	balanc
10	12.269337	0.270419	0.004005	0.001751	100	{0: 1, 1
11	14.698447	0.589340	0.003300	0.000281	100	balanc
12	20.436544	4.038092	0.013488	0.012635	1000	{0: 1, 1
13	22.278455	3.952154	0.006060	0.004269	1000	balanc

In [15]:
```
gs10cv.best_params_
```

Out[15]:
```
{'C': 1, 'class_weight': {0: 1, 1: 1}}
```

```
In [16]: pd.DataFrame(gs10cv.cv_results_).iloc[gs10cv.best_index_]
```

```
Out[16]: mean_fit_time                                                 3.00857
         std_fit_time                                                 0.900897
         mean_score_time                                            0.00784965
         std_score_time                                             0.00786795
         param_C                                                             1
         param_class_weight                                       {0: 1, 1: 1}
         params                      {'C': 1, 'class_weight': {0: 1, 1: 1}}
         split0_test_score                                            0.976955
         split1_test_score                                            0.981145
         split2_test_score                                            0.975559
         split3_test_score                                             0.97905
         split4_test_score                                            0.976257
         split5_test_score                                            0.982542
         split6_test_score                                            0.981145
         split7_test_score                                            0.976257
         split8_test_score                                            0.979734
         split9_test_score                                            0.979734
         mean_test_score                                              0.978838
         std_test_score                                             0.00231603
         rank_test_score                                                     1
         Name: 6, dtype: object
```

```
In [17]: # Save model
         SVMClassFile = open('./models/SVMClass_BestCV_STDData_pickle_file',
         'wb')
         pickle.dump(gs10cv, SVMClassFile)
```

No-correlated Data Training

Grid Search 10-Fold CV:

```
In [18]: gs10cv_nc = GridSearchCV(SVMClass, param_grid = grid, cv = 10, n_jo
         bs = -1)
```

Training:

```
In [19]: gs10cv_nc.fit(X_train_NC, y_train_NC)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/svm/_base.py:947: C
onvergenceWarning: Liblinear failed to converge, increase the numb
er of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Out[19]: GridSearchCV(cv=10, error_score=nan,
             estimator=LinearSVC(C=1.0, class_weight=None, dual=Tr
ue,
                                 fit_intercept=True, intercept_sca
ling=1,
                                 loss='squared_hinge', max_iter=50
00,
                                 multi_class='ovr', penalty='l2',
                                 random_state=1234, tol=0.0001, ve
rbose=0),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
,
                         'class_weight': [{0: 1, 1: 1}, 'balanced'
]},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
             scoring=None, verbose=0)
```

```
In [20]: pd.DataFrame(gs10cv_nc.cv_results_)
```

Out[20]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param
0	0.072511	0.016275	0.004814	0.001002	0.001	{0: 1,
1	0.096511	0.018379	0.005374	0.004214	0.001	balanc
2	0.174571	0.054502	0.006091	0.005424	0.01	{0: 1,
3	0.345941	0.051665	0.004582	0.002179	0.01	balanc
4	0.346532	0.057209	0.003759	0.001156	0.1	{0: 1,
5	2.928549	0.289189	0.005061	0.002702	0.1	balanc

6	3.288173	0.410353	0.004000	0.001605	1	{0: 1, 1
7	20.338766	3.271273	0.016011	0.014751	1	balanc
8	7.889672	0.098631	0.003510	0.001085	10	{0: 1, 1
9	17.747058	0.976429	0.004255	0.001413	10	balanc
10	15.145197	0.359181	0.003471	0.001386	100	{0: 1, 1
11	19.591599	0.378313	0.003652	0.001386	100	balanc
12	19.349106	1.263056	0.005189	0.003010	1000	{0: 1, 1
13	17.759900	2.997603	0.003675	0.001294	1000	balanc

In [21]:
```
gs10cv_nc.best_params_
```

Out[21]:
```
{'C': 100, 'class_weight': {0: 1, 1: 1}}
```

In [22]:
```
pd.DataFrame(gs10cv_nc.cv_results_).iloc[gs10cv_nc.best_index_]
```

Out[22]:
```
mean_fit_time                                                15.1452
std_fit_time                                                0.359181
mean_score_time                                           0.00347054
std_score_time                                            0.00138645
param_C                                                          100
param_class_weight                                      {0: 1, 1: 1}
params                    {'C': 100, 'class_weight': {0: 1, 1: 1}}
split0_test_score                                           0.975559
split1_test_score                                           0.980447
split2_test_score                                           0.975559
split3_test_score                                           0.979749
split4_test_score                                           0.975559
split5_test_score                                           0.981844
split6_test_score                                           0.981844
split7_test_score                                           0.974162
split8_test_score                                           0.980433
split9_test_score                                           0.979734
mean_test_score                                             0.978489
std_test_score                                            0.00278821
rank_test_score                                                    1
Name: 10, dtype: object
```

In [23]:
```
# Save model
SVMClassFileNC = open('./models/SVMClass_BestCV_NCorrSTDData_pickle
_file', 'wb')
pickle.dump(gs10cv_nc, SVMClassFileNC)
```

Testing

Normal Data Model Testing

In [24]:
```
y_pred = gs10cv.predict(X_test)
```

In [25]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3249
           1       0.93      0.80      0.86       331

    accuracy                           0.98      3580
   macro avg       0.96      0.90      0.92      3580
weighted avg       0.98      0.98      0.98      3580
```

In [26]:
```
print ("Confusion Matrix:")
confusionMatrix(y_test, y_pred, classes = [0,1])
```

```
Confusion Matrix:
```

Out[26]:

Predicted	0	1
Real		
0	3229	20
1	65	266

In [27]:
```
print("Test Error:")
(1-accuracy_score(y_test, gs10cv.predict(X_test)))*100
```

```
Test Error:
```

Out[27]: 2.3743016759776525

No-correlated Data Model Testing

In [28]:
```
y_pred_NC = gs10cv_nc.predict(X_test_NC)
```

In [29]:
```
print(classification_report(y_test_NC, y_pred_NC))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3249
           1       0.93      0.82      0.87       331

    accuracy                           0.98      3580
   macro avg       0.95      0.90      0.93      3580
weighted avg       0.98      0.98      0.98      3580
```

In [30]:
```
print ("Confusion Matrix:")
confusionMatrix(y_test_NC, y_pred_NC, classes = [0,1])
```

```
Confusion Matrix:
```

Out[30]:

Predicted	0	1
Real		
0	3228	21
1	61	270

In [31]:
```
print("Test Error:")
(1-accuracy_score(y_test_NC, gs10cv_nc.predict(X_test_NC)))*100
```

```
Test Error:
```

Out[31]:
```
2.2905027932960897
```