1. preprocessing

In [1]:
```
import pandas as pd
```

In [2]:
```
#import data
traindata = pd.read_csv('train.csv')
valdata = pd.read_csv('valid.csv')
testdata = pd.read_csv('test.csv')
valdata
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.80786
2	36.0	-1.169422	1.158314	1.406800	0.860189	-0.103810	0.122035	0.264451	-0.10876
3	49.0	0.921544	-0.067084	0.077461	0.953638	0.067412	0.016152	0.320452	0.038534
4	51.0	1.259873	0.254240	0.514789	0.620924	-0.475930	-0.992286	0.066417	-0.20927
5	52.0	-3.005237	2.600138	1.483691	-2.418473	0.306326	-0.824575	2.065426	-1.82934
6	52.0	1.147369	0.059035	0.263632	1.211023	-0.044096	0.301067	-0.132960	0.227885
7	132.0	-1.571359	1.687508	0.734670	1.293350	-0.217532	-0.002677	0.147364	0.515362
8	172.0	-1.428535	1.793578	0.545758	-0.399309	0.673577	0.745919	-0.358475	-4.04472
9	200.0	-0.304923	-0.521388	1.341181	-0.255797	-1.120916	0.918260	0.174348	0.358936
10	231.0	-0.961507	0.743351	1.173582	-0.222593	0.740528	1.739959	0.313013	0.699419
11	249.0	-0.788406	1.125172	0.934213	1.155815	-0.028518	0.489476	0.067377	0.775357
12	259.0	-1.569485	-1.932133	1.249203	-4.434211	1.244282	0.402688	-0.649554	0.534756
13	268.0	1.146065	0.285853	0.562439	1.459336	-0.225891	-0.346303	0.131988	-0.08517
14	273.0	0.216727	0.710626	1.224876	1.139405	-0.376351	-0.253986	-0.034557	-0.81254
15	290.0	-0.695818	0.581773	2.378180	0.063396	0.329119	-0.449865	1.269104	-0.75836
16	362.0	-0.692699	0.291548	1.575228	-1.219400	-0.302257	0.670790	0.333059	0.153875
17	363.0	-1.028699	0.910515	1.915180	2.469384	-0.008375	0.597584	0.251531	-0.33173
18	370.0	1.354445	-0.815297	0.836498	-0.617140	-1.304124	-0.025274	-1.147177	0.162996
19	394.0	-0.553092	1.667591	-0.047357	0.514249	0.589388	-0.635411	1.126611	-0.31188
20	422.0	-0.792748	-0.554385	1.698544	-3.774039	0.005129	-0.353166	0.362089	0.044961
21	449.0	-0.856525	0.583290	1.389014	-0.344699	0.267594	-0.951375	0.523117	-0.04922
22	484.0	1.060623	0.077500	-0.049338	0.546163	0.004831	-0.732323	0.547012	-0.27500
23	487.0	1.164359	-0.283474	-0.247022	0.192064	-0.327522	-0.822581	0.241389	-0.22846
24	513.0	1.255258	0.075190	0.225733	0.881766	0.154508	0.631960	-0.385968	0.189493
25	517.0	1.314713	-0.328688	0.002645	-0.805044	-0.467260	-0.522747	-0.180850	-0.09347
26	545.0	-1.030747	0.894747	2.375620	0.218372	-0.862126	-0.180483	-0.260299	0.592381
27	572.0	-0.772941	0.815404	1.365536	-0.519298	1.034867	1.658806	0.436415	0.505824
28	626.0	1.162650	-0.059101	0.625173	0.946211	-0.825452	-0.913223	-0.051476	-0.10570
29	636.0	-0.220314	1.371503	0.039843	0.607252	0.524369	-0.520815	1.274831	-0.35519
...	...	...	...	...	...	...	...	...	...
9970	172098.0	-0.347837	0.989816	-0.127569	-0.531810	0.072949	-1.162303	0.635664	0.253214
9971	172112.0	1.859056	0.430990	-0.445339	3.856303	0.251548	0.137637	0.045917	0.021581
9972	172202.0	2.014613	-0.082815	-3.070613	0.256580	2.933003	3.302523	0.011913	0.672023
9973	172205.0	0.060338	0.414768	1.174723	-0.458932	-0.154833	-0.375374	0.245951	0.027719
9974	172212.0	1.867206	-0.588987	-0.576779	0.473870	-0.496392	-0.134690	-0.541734	0.019939

In [3]:
```
#split X / y
X_raw_df, y_df = traindata.drop('Class', axis=1), traindata.Class
X_raw_val, y_val = valdata.drop('Class', axis=1), valdata.Class
```

In [4]:
```
# scale
from sklearn import preprocessing
standardize = ['Time', 'Amount']
scaled = preprocessing.StandardScaler().fit(X_raw_df[standardize])
columns_scaled = scaled.transform(X_raw_df[standardize])
X_df = X_raw_df.copy()
X_df[standardize] = columns_scaled

scaled2 = preprocessing.StandardScaler().fit(X_raw_val[standardize])
columns_scaled2 = scaled2.transform(X_raw_val[standardize])
X_val = X_raw_val.copy()
X_val[standardize] = columns_scaled2
```

2. Models

Logistic Regression

In [5]:
```
from sklearn.metrics import f1_score
multi_results = pd.DataFrame()
scores = pd.DataFrame()
```

In [6]:
```
from sklearn.linear_model import LogisticRegression
logr_model = LogisticRegression(class_weight='balanced')
logr_model.fit(X=X_df, y=y_df)
y_predicted_logr = logr_model.predict(X=X_val)
score_lr = f1_score(y_true = y_val, y_pred=y_predicted_logr)
score_lr
```

Out[6]: 0.40646651270207851

Decision Tree

In [7]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [8]:
```
dt_model = DecisionTreeClassifier(random_state=28, class_weight='balanced')
dt_model.fit(X=X_df, y=y_df)
y_predicted_dt = dt_model.predict(X=X_val)
score_dt = f1_score(y_true=y_val, y_pred=y_predicted_dt)
score_dt
```

Out[8]: 0.80000000000000004

Random Forest

In [9]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [10]:
```
rf_model = RandomForestClassifier(random_state=28, class_weight='balanced')
rf_model.fit(X=X_df, y=y_df)
y_predicted_rf = rf_model.predict(X=X_val)
score_rf = f1_score(y_true = y_val, y_pred=y_predicted_rf)
score_rf
```

Out[10]: 0.84090909090909094

Dimension reduction

logistic regression

In [11]:
```
from sklearn.feature_selection import RFE
```

In [12]:
```
logr = LogisticRegression(class_weight='balanced')
selector = RFE(logr)
selector.fit(X_df, y_df)
selected_cols = selector.support_
X_selected_df = X_df.loc[:, selected_cols]
X_selected_val = X_val.loc[:, selected_cols]
```

In [13]:
```
X_selected_df.columns
```

Out[13]:
```
Index(['V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V12', 'V14', 'V16', 'V20', 'V22',
       'V25', 'V26', 'V27', 'Amount'],
      dtype='object')
```

In [13]:
```
nfs = [1,  5, 10, 15, 25, 29]
logr = LogisticRegression(class_weight='balanced')
for fs in nfs:
    selector = RFE(logr, n_features_to_select=fs)
    selector.fit(X_df, y_df)
    selected_cols = selector.support_
    X_selected_df = X_df.loc[:, selected_cols]
    X_selected_val = X_val.loc[:, selected_cols]
    logr_model2 = LogisticRegression(class_weight='balanced')
    logr_model2.fit(X_selected_df, y_df)
    y_predicted_logr2 = logr_model2.predict(X=X_selected_val)
    score_logr2 = f1_score(y_true = y_val, y_pred=y_predicted_logr2)
    print(fs, score_logr2)
```

```
1 0.407582938389
5 0.422604422604
10 0.412177985948
15 0.387665198238
25 0.405529953917
29 0.406466512702
```

decision tree

In [14]:
```
nfs = [7, 8, 9, 15, 16, 28, 29]
dt = DecisionTreeClassifier(random_state=28, class_weight='balanced')
for fs in nfs:
    selector = RFE(dt, n_features_to_select=fs)
    selector.fit(X_df, y_df)
    selected_cols = selector.support_
    X_selected_df = X_df.loc[:, selected_cols]
    X_selected_val = X_val.loc[:, selected_cols]
    dt_model2 = DecisionTreeClassifier(random_state=28, class_weight='balanced')
    dt_model2.fit(X_selected_df, y_df)
    y_predicted_dt2 = dt_model2.predict(X=X_selected_val)
    score_dt2 = f1_score(y_true=y_val, y_pred=y_predicted_dt2)
    print(fs, score_dt2)
```

```
7 0.829015544041
8 0.833333333333
9 0.820512820513
15 0.8125
16 0.80829015544
28 0.8
29 0.8
```

3. modification

parameter adjustments

logistic regression

In [15]:
```
cs = [0.01, 0.1, 1, 5, 10]
for c in cs:
    lr = LogisticRegression(C=c, class_weight='balanced')
    lr.fit(X_selected_df, y_df)
    y_c_lr = lr.predict(X=X_selected_val)
    score_c_lr = f1_score(y_true=y_val, y_pred=y_c_lr)
    print(score_c_lr)
#choose c=0.01
```

```
0.447300771208
0.433497536946
0.407407407407
0.401826484018
0.400911161731
```

In [18]:
```
randomf = RandomForestClassifier(class_weight='balanced', random_state=28)
randomf.fit(X=X_df, y=y_df)
y_randomf = randomf.predict(X=X_val)
score1 = f1_score(y_true=y_val, y_pred=y_randomf)
score1
```

Out[18]: 0.84090909090909094

random forest

In [19]:
```
#max features
maxf = [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, "auto"]
for f in maxf:
    lst = []
    for i in range(10):
        rf = RandomForestClassifier(class_weight='balanced', max_features=f)
        rf.fit(X=X_df, y=y_df)
        y_f_rf = rf.predict(X=X_val)
        score_f_rf = f1_score(y_true = y_val, y_pred=y_f_rf)
        lst.append(score_f_rf)
    print(f, sum(lst)/len(lst))
```

```
20 0.861899755082
21 0.863942267016
22 0.862497733404
23 0.865379528853
24 0.867800098065
25 0.863279656753
26 0.862320656652
27 0.863316185088
28 0.869717461011
29 0.867549036023
auto 0.853555920894
```

In [20]:
```
# max_depth
max_depth = [7, 8, 9, 10, 11]
for d in max_depth:
    lst = []
    for i in range(10):
        rf = RandomForestClassifier(class_weight='balanced', max_features=25, max_depth=d)
        rf.fit(X=X_df, y=y_df)
        y_f_rf = rf.predict(X=X_val)
        score_f_rf = f1_score(y_true = y_val, y_pred=y_f_rf)
        lst.append(score_f_rf)
    print(d, sum(lst)/len(lst))
```

```
7 0.86703047365
8 0.868164780391
9 0.870651948896
10 0.868658323623
11 0.862075333404
```

In [21]:
```
#n_estimators
ne = [10, 20, 30, 50, 100, 120]
for est in ne:
    lst = []
    for i in range(5):
        rf = RandomForestClassifier(class_weight='balanced', max_depth=9, max_features=25, n_es
timators=est)
        rf.fit(X=X_df, y=y_df)
        y_f_rf = rf.predict(X=X_val)
        score_f_rf = f1_score(y_true = y_val, y_pred=y_f_rf)
        lst.append(score_f_rf)
    print(est, sum(lst)/len(lst))
```

```
10 0.870807857037
20 0.873639819435
30 0.872416250891
50 0.873366595391
100 0.872416250891
120 0.872416250891
```

Chosen model

In [31]:
```
randomfnone1 = RandomForestClassifier(class_weight='balanced', max_features=25, max_depth=9, n_
estimators=20)
randomfnone1.fit(X=X_df, y=y_df)
y_none1 = randomfnone1.predict(X=X_val)
scorenone1 = f1_score(y_true = y_val, y_pred=y_none1)
scorenone1
```

Out[31]: 0.87431693989071035

In [34]:
```
list(y_none1)
```

```
Out[34]:  [0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
```

In [ ]: