In [1]:

```
#Importing librairies

import pandas as pd
import numpy as np

# Scikit-learn library: For SVM
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn import svm

import itertools

# Matplotlib library to plot the charts
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

# Library for the statistic data vizualisation
import seaborn

%matplotlib inline
```

In [4]:

```
data = pd.read_csv('C:/Users/nikita/Downloads/jupyternotebooks/creditcard.csv') # Readin
df = pd.DataFrame(data) # Converting data to Panda DataFrame
```

In [5]:

```
df = pd.DataFrame(data) # Converting data to Panda DataFrame
```

In [6]:

```
df.describe()
```

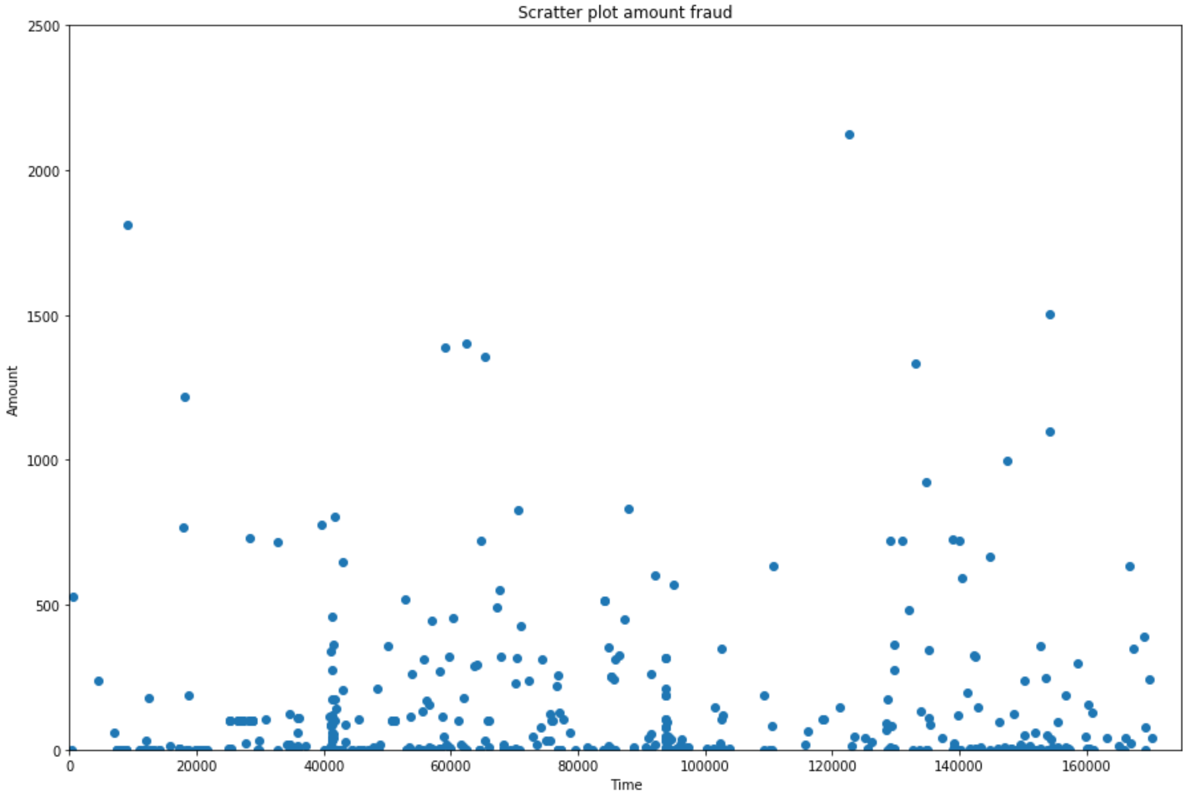
Out[6]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+

8 rows × 31 columns

In [7]:

```
df_fraud = df[df['Class'] == 1] # Recovery of fraud data
plt.figure(figsize=(15,10))
plt.scatter(df_fraud['Time'], df_fraud['Amount']) # Display fraud amounts according to t
plt.title('Scratter plot amount fraud')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.xlim([0,175000])
plt.ylim([0,2500])
plt.show()
```



In [8]:

```
= df_fraud[df_fraud['Amount'] > 1000].shape[0] # Recovery of frauds over 1000
are only '+ str(nb_big_fraud) + ' frauds where the amount was bigger than 1000 over ' +
```

```
There are only 9 frauds where the amount was bigger than 1000 over 492 fra
uds
```

In [9]:

```
number_fraud = len(data[data.Class == 1])
number_no_fraud = len(data[data.Class == 0])
print('There are only '+ str(number_fraud) + ' frauds in the original dataset, even thou
```

```
There are only 492 frauds in the original dataset, even though there are 2
84315 no frauds in the dataset.
```

In [10]:

```
would be : "+ str((284315-492)/284315)+ " which is the number of good classification ove
```

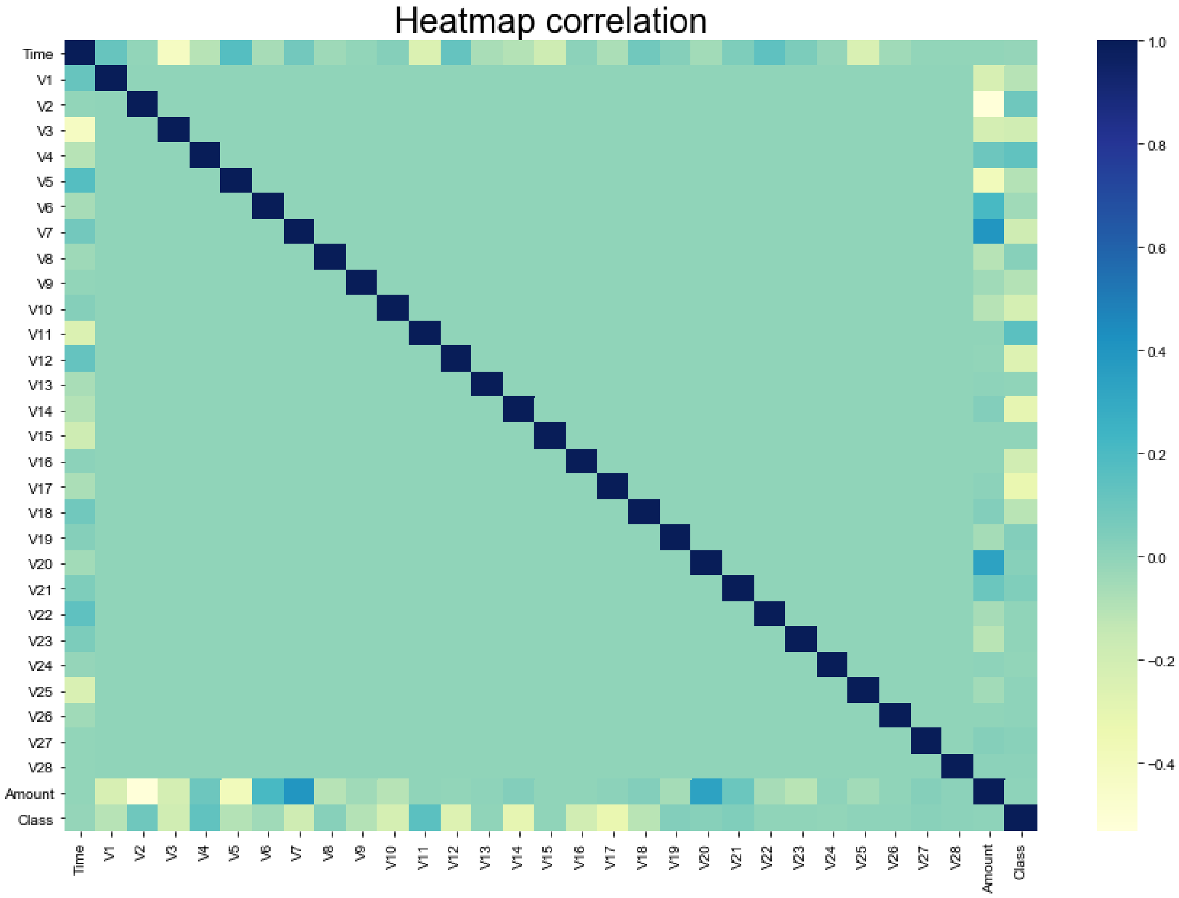
```
The accuracy of the classifier then would be : 0.998269524998681 which is
the number of good classification over the number of tuple to classify
```

In [11]:

```
df_corr = df.corr() # Calculation of the correlation coefficients in pairs, with the de
                    # Pearson, Standard Correlation Coefficient
```

In [12]:

```
plt.figure(figsize=(15,10))
seaborn.heatmap(df_corr, cmap="YlGnBu") # Displaying the Heatmap
seaborn.set(font_scale=2,style='white')

plt.title('Heatmap correlation')
plt.show()
```



In [13]:

```
rank = df_corr['Class'] # Retrieving the correlation coefficients per feature in relati
df_rank = pd.DataFrame(rank)
df_rank = np.abs(df_rank).sort_values(by='Class',ascending=False) # Ranking the absolute
                                                                  # in descending order
df_rank.dropna(inplace=True) # Removing Missing Data (not a number)
```

In [14]:

```
# We seperate ours data in two groups : a train dataset and a test dataset

# First we build our train dataset
df_train_all = df[0:150000] # We cut in two the original dataset
df_train_1 = df_train_all[df_train_all['Class'] == 1] # We seperate the data which are t
df_train_0 = df_train_all[df_train_all['Class'] == 0]
print('In this dataset, we have ' + str(len(df_train_1)) +" frauds so we need to take a

df_sample=df_train_0.sample(300)
df_train = df_train_1.append(df_sample) # We gather the frauds with the no frauds.
df_train = df_train.sample(frac=1) # Then we mix our dataset
```

```
In this dataset, we have 293 frauds so we need to take a similar number of
non-fraud
```

In [15]:

```
X_train = df_train.drop(['Time', 'Class'],axis=1) # We drop the features Time (useless),
y_train = df_train['Class'] # We create our label
X_train = np.asarray(X_train)
y_train = np.asarray(y_train)
```

In [16]:

```
################################ with all the test dataset to see if the model learn corre
df_test_all = df[150000:]

X_test_all = df_test_all.drop(['Time', 'Class'],axis=1)
y_test_all = df_test_all['Class']
X_test_all = np.asarray(X_test_all)
y_test_all = np.asarray(y_test_all)
```

In [17]:

```
X_train_rank = df_train[df_rank.index[1:11]] # We take the first ten ranked features
X_train_rank = np.asarray(X_train_rank)
```

In [18]:

```
################################ with all the test dataset to see if the model learn corre
X_test_all_rank = df_test_all[df_rank.index[1:11]]
X_test_all_rank = np.asarray(X_test_all_rank)
y_test_all = np.asarray(y_test_all)
```

In [19]:

```
class_names=np.array(['0','1']) # Binary label, Class = 1 (fraud) and Class = 0 (no frau
```

In [20]:

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [21]:

```
classifier = svm.SVC(kernel='linear') # We set a SVM classifier, the default SVM Classif
```

In [22]:

```
classifier.fit(X_train, y_train) # Then we train our model, with our balanced data trai
```

Out[22]:

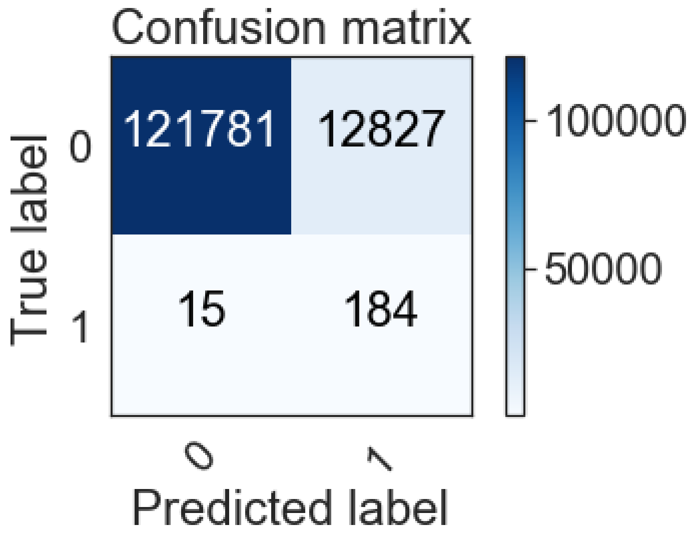
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linea
r',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [23]:

```
prediction_SVM_all = classifier.predict(X_test_all) #And finally, we predict our data te
```

In [24]:

```
cm = confusion_matrix(y_test_all, prediction_SVM_all)
plot_confusion_matrix(cm,class_names)
```



In [25]:

```
print('Our criterion give a result of '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]-
```

```
Our criterion give a result of 0.9206460693685553
```

In [26]:

```
print('We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) + ' to
print('\nSo, the probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("the accuracy is : "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

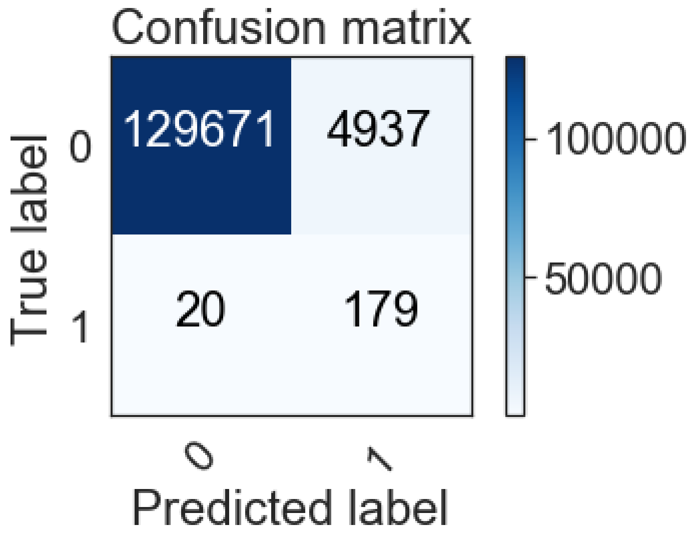
```
We have detected 184 frauds / 199 total frauds.

So, the probability to detect a fraud is 0.9246231155778895
the accuracy is : 0.9047378845312187
```

In [27]:

```
classifier.fit(X_train_rank, y_train) # Then we train our model, with our balanced data
prediction_SVM = classifier.predict(X_test_all_rank) #And finally, we predict our data t
```

In [28]:

```
cm = confusion_matrix(y_test_all, prediction_SVM)
plot_confusion_matrix(cm,class_names)
```



In [29]:

```
print('Our criterion give a result of '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]-
```

```
Our criterion give a result of 0.9122437724387886
```

In [30]:

```
print('We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) + ' to
print('\nSo, the probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("the accuracy is : "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

```
We have detected 179 frauds / 199 total frauds.

So, the probability to detect a fraud is 0.8994974874371859
the accuracy is : 0.9632289124451995
```

In [31]:

```
classifier_b = svm.SVC(kernel='linear',class_weight={0:0.60, 1:0.40})
```

In [32]:

```
classifier_b.fit(X_train, y_train) # Then we train our model, with our balanced data tra
```

Out[32]:

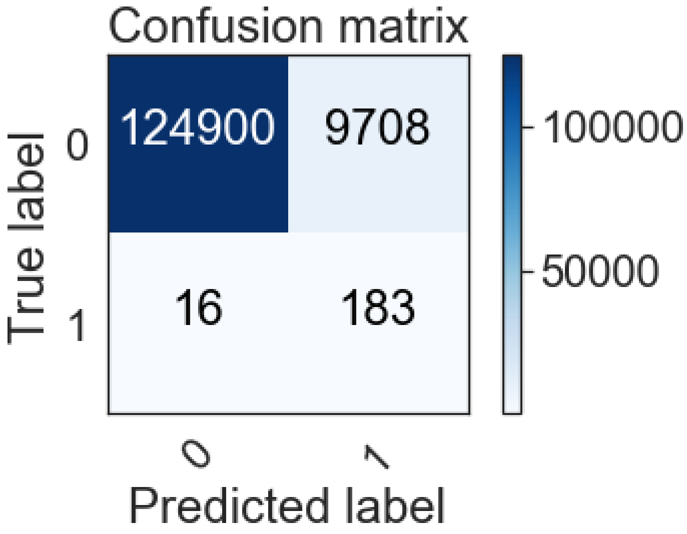
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight={0: 0.6, 1: 0.
4},
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [33]:

```
prediction_SVM_b_all = classifier_b.predict(X_test_all) #We predict all the data set.
```

In [34]:

```
cm = confusion_matrix(y_test_all, prediction_SVM_b_all)
plot_confusion_matrix(cm,class_names)
```



In [35]:

```
print('Our criterion give a result of '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]-
```

```
Our criterion give a result of 0.9212518414097535
```

In [36]:

```
print('We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) + ' to
print('\nSo, the probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("the accuracy is : "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

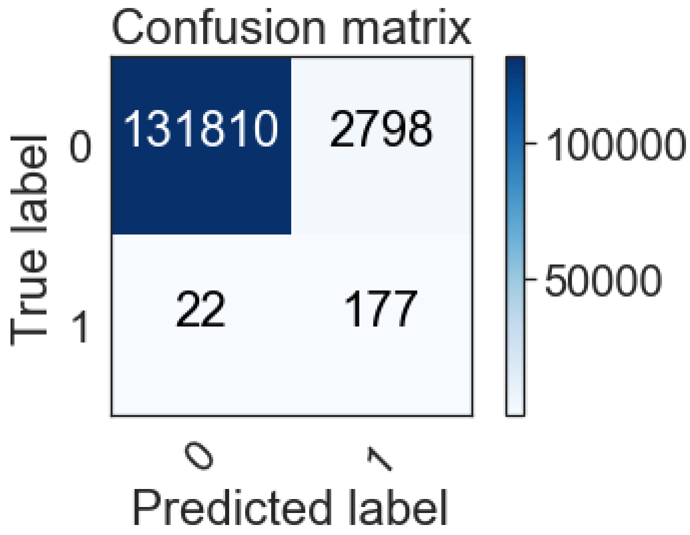
```
We have detected 183 frauds / 199 total frauds.

So, the probability to detect a fraud is 0.9195979899497487
the accuracy is : 0.9278672472497719
```

In [37]:

```
classifier_b.fit(X_train_rank, y_train) # Then we train our model, with our balanced dat
prediction_SVM = classifier_b.predict(X_test_all_rank) #And finally, we predict our data
```

In [38]:

```
cm = confusion_matrix(y_test_all, prediction_SVM)
plot_confusion_matrix(cm,class_names)
```



In [39]:

```
print('Our criterion give a result of '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]
```

```
Our criterion give a result of 0.907374029941111
```

In [40]:

```
print('We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) + ' t
print('\nSo, the probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0]))
print("the accuracy is : "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

```
We have detected 177 frauds / 199 total frauds.

So, the probability to detect a fraud is 0.8894472361809045
the accuracy is : 0.9790812049819372
```

In [ ]:

```
```

In [ ]:

```
```