Assignment NO.2 Solutions

Data Mining | Fall 1401 | Dr.Rahmani

Student name : **Amin Fathi**

Student id : **400722102**

ابتدا پکیج‌های لازم را import کرده و dataset داده شده را نیز بارگذاری می‌کنیم.

```
import pandas as pd # data processing
import numpy as np # working with arrays
import matplotlib.pyplot as plt # visualization
from termcolor import colored as cl # text customization
import itertools # advanced tools

from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.model_selection import train_test_split # data split
from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
from sklearn.linear_model import LogisticRegression # Logistic regression algorithm
from sklearn.svm import SVC # SVM algorithm
from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm
from xgboost import XGBClassifier # XGBoost algorithm

from sklearn.metrics import confusion_matrix # evaluation metric
from sklearn.metrics import accuracy_score # evaluation metric
from sklearn.metrics import f1_score # evaluation metric
```

In [2]:
```
df = pd.read_csv(r'C:\Users\User\Desktop\creditcard.csv')
df
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.06692
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.33984
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.68928
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.17557
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.14126
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.50934
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.01622
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.64013
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.12320
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.00879

284807 rows × 31 columns

همان‌طور که مشاهده می‌شود، این dataset شامل ۳۱ ستون و ۲۸۴۸۰۷ سطر یا نمونه است. به عبارتی ۳۰ ستون شامل time، V1 تا V28 و Amount بیانگر ویژگی‌ها و ستون Class بیانگر برچسب‌های موجود در مجموعه داده است.

در ادامه از ما خواسته شده است تا درصورت وجود فیلدهای خالی، آن‌ها را به نحو مناسبی پر کنیم. اما همان‌طور که مشاهده می‌شود، در این مجموعه داده فیلد خالی وجود ندارد.

In [3]:
```
df.isnull().values.any()
```

Out[3]: False

یک راه دیگر برای چک کردن وجود فیلدهای خالی در هر ستون وجود دارد که در زیر نشان داده شده است. از این طریق هم متوجه می‌شویم که این مجموعه داده فیلد خالی در هیچ یک از ستون‌هایش ندارد.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

با بررسی مقادیر موجود در ستون Amount، متوجه تفاوت زیاد در میان داده‌های این ستون می‌شویم. بنابراین نتیجه می‌گیریم که این ستون نیاز به نرمال‌سازی دارد. برای نرمال‌سازی از تابع StandardScaler استفاده شده است.

```
sc = StandardScaler()
amount = df['Amount'].values
df['Amount'] = sc.fit_transform(amount.reshape(-1, 1))
```

در فرایند مدل‌سازی متغیر time را حذف می‌کنیم.

```
df.drop(['Time'], axis=1, inplace=True)
```

در ادامه می‌خواهیم مجموعه داده را به دو مجموعه آموزشی (Training set) و آزمایشی (Test set) تقسیم کنیم. این نکته را هم می‌دانیم که نباید از داده‌های آموزشی در داده‌های آزمایشی نمونه‌ای وجود داشته باشد. بنابراین ابتدا سعی می‌کنیم که سطرهای (نمونه‌های) تکراری را درصورت وجود حذف کنیم تا از وجود نمونه‌ای با شباهت کامل در همزمان در مجموعه آموزشی و آزمایشی جلوگیری شود.

```
print(df.shape)
df.drop_duplicates(inplace=True)
print(df.shape)
```

```
(284807, 30)
(275663, 30)
```

همان‌طور که مشاهده می‌شود، تقریبا نزدیک به ۹۰۰۰ نمونه تکراری در مجموعه داده وجود داشته که حذف شدند.

```
X = df.drop('Class', axis = 1).values
y = df['Class'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

در ادامه مدل‌های مختلف را پیاده‌سازی می‌کنیم. برای پیاده‌سازی مدل‌ها از پکیج ارائه شده توسط scikit-learn استفاده می‌کنیم. سپس، مقدار Accuracy، F1-Score و confusion matrix را نیز برای هر مدل به دست آورده و نمایش می‌دهیم.

- **Decision Tree**

```
#Decision Tree

DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
DT.fit(X_train, y_train)
tree_yhat = DT.predict(X_test)

print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, tree_yhat)))
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))
confusion_matrix(y_test, tree_yhat, labels = [0, 1])
```

```
Accuracy score of the Decision Tree model is 0.9991583957281328
F1 score of the Decision Tree model is 0.7521367521367521

array([[68770,    18],
       [   40,    88]], dtype=int64)
```

```
#Decision Tree

DT = DecisionTreeClassifier(max_depth = 8, criterion = 'entropy')
DT.fit(X_train, y_train)
tree_yhat = DT.predict(X_test)

print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, tree_yhat)))
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))
confusion_matrix(y_test, tree_yhat, labels = [0, 1])
```

```
Accuracy score of the Decision Tree model is 0.999288989494457
F1 score of the Decision Tree model is 0.7896995708154506

array([[68775,    13],
       [   36,    92]], dtype=int64)
```

- **K-Nearest Neighbors**

```
#K-Nearest Neighbors

n = 3
KNN = KNeighborsClassifier(n_neighbors = n)
KNN.fit(X_train, y_train)
knn_yhat = KNN.predict(X_test)

print('Accuracy score of the K-Nearest Neighbors model is {}'.format(accuracy_score(y_test, knn_yhat)))
print('F1 score of the K-Nearest Neighbors model is {}'.format(f1_score(y_test, knn_yhat)))
confusion_matrix(y_test, knn_yhat, labels = [0, 1])
```

```
Accuracy score of the K-Nearest Neighbors model is 0.9994486040977422
F1 score of the K-Nearest Neighbors model is 0.831858407079646

array([[68784,     4],
       [   34,    94]], dtype=int64)
```

```
#K-Nearest Neighbors

n = 7
KNN = KNeighborsClassifier(n_neighbors = n)
KNN.fit(X_train, y_train)
knn_yhat = KNN.predict(X_test)

print('Accuracy score of the K-Nearest Neighbors model is {}'.format(accuracy_score(y_test, knn_yhat)))
print('F1 score of the K-Nearest Neighbors model is {}'.format(f1_score(y_test, knn_yhat)))
confusion_matrix(y_test, knn_yhat, labels = [0, 1])
```

```
Accuracy score of the K-Nearest Neighbors model is 0.999288989494457
F1 score of the K-Nearest Neighbors model is 0.7949790794979079

array([[68772,    16],
       [   33,    95]], dtype=int64)
```

- **Logistic Regression**

```
#Logistic Regression

lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)

print('Accuracy score of the Logistic Regression model is {}'.format(accuracy_score(y_test, lr_yhat)))
print('F1 score of the Logistic Regression model is {}'.format(f1_score(y_test, lr_yhat)))
confusion_matrix(y_test, lr_yhat, labels = [0, 1])
```

```
Accuracy score of the Logistic Regression model is 0.9989552498694062
F1 score of the Logistic Regression model is 0.6666666666666666

array([[68772,    16],
       [   56,    72]], dtype=int64)
```

- **SVM**

```
#Support Vector Machines

svm = SVC()
svm.fit(X_train, y_train)
svm_yhat = svm.predict(X_test)

print('Accuracy score of the Support Vector Machines model is {}'.format(accuracy_score(y_test, svm_yhat)))
print('F1 score of the Support Vector Machines model is {}'.format(f1_score(y_test, svm_yhat)))
confusion_matrix(y_test, svm_yhat, labels = [0, 1])
```

```
Accuracy score of the Support Vector Machines model is 0.999318010331418
F1 score of the Support Vector Machines model is 0.7813953488372093

array([[68785,     3],
       [   44,    84]], dtype=int64)
```

- **Random Forest**

```
#Random Forest

rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)

print('Accuracy score of the Random Forest model is {}'.format(accuracy_score(y_test, rf_yhat)))
print('F1 score of the Random Forest model is {}'.format(f1_score(y_test, rf_yhat)))
confusion_matrix(y_test, rf_yhat, labels = [0, 1])
```

```
Accuracy score of the Random Forest model is 0.9991583957281328
F1 score of the Random Forest model is 0.7314814814814815

array([[68779,     9],
       [   49,    79]], dtype=int64)
```

- **XGBoost**

```
#XGBoost

xgb = XGBClassifier(max_depth = 4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)

print('Accuracy score of the XGBoost model is {}'.format(accuracy_score(y_test, xgb_yhat)))
print('F1 score of the XGBoost model is {}'.format(f1_score(y_test, xgb_yhat)))
confusion_matrix(y_test, xgb_yhat, labels = [0, 1])
```

```
Accuracy score of the XGBoost model is 0.999506645771664
F1 score of the XGBoost model is 0.8495575221238937

array([[68786,     2],
       [   32,    96]], dtype=int64)
```

نتایج به دست آمده از اجرای هر الگوریتم را در جدول زیر مقایسه می‌کنیم.

Algorithms	Accuracy	F1-score
Decision Tree (max_depth = 4)	0.999158	0.75214
Decision Tree (max_depth =8)	0.999289	0.78969
K-Nearest Neighbors (K = 3)	0.999449	0.83186
K-Nearest Neighbors (K = 5)	0.999333	0.80342
K-Nearest Neighbors (K = 7)	0.999289	0.79498
Logistic Regression	0.99895525	0.66667
SVM	0.99931801	0.7814
Random Forest Tree	0.999158396	0.73148
XGBoost	0.999506646	0.84956

همان‌طور که مشاهده می‌شود الگوریتم XGBoost و KNN با K=3، بهترین عملکرد را نشان داده اند. پایین‌ترین عملکرد نیز برای الگوریتم Logistic Regression است. درحالت کلی مشاهده می‌شود که Accuracy برای تمام الگوریتم‌ها تقریبا 99% است.

همچنین در انتها به پیاده‌سازی شبکه عصبی می‌پردازیم و نتایج حاصل نیز نشان داده شده است. شبکه عصبی را در دو مدل با دو معماری متفاوت پیاده‌سازی کرده‌ایم. به این صورت که در هر مدل تعداد نورون‌های هر لایه متفاوت است.

مدل اول:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
model = Sequential([
    Dense(units=20, input_dim = X_train.shape[1], activation='relu'),
    Dense(units=24,activation='relu'),
    Dropout(0.5),
    Dense(units=20,activation='relu'),
    Dense(units=24,activation='relu'),
    Dense(1, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                600

 dense_1 (Dense)             (None, 24)                504

 dropout (Dropout)           (None, 24)                0

 dense_2 (Dense)             (None, 20)                500

 dense_3 (Dense)             (None, 24)                504

 dense_4 (Dense)             (None, 1)                 25

=================================================================
Total params: 2,133
Trainable params: 2,133
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=30, epochs=5)
```

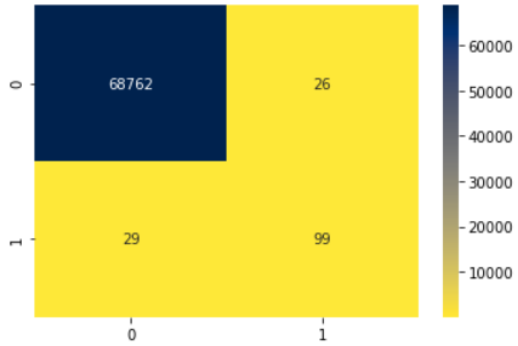
```
Epoch 1/5
6892/6892 [==============================] - 15s 2ms/step - loss: 0.0100 - accuracy: 0.9987
Epoch 2/5
6892/6892 [==============================] - 13s 2ms/step - loss: 0.0035 - accuracy: 0.9993
Epoch 3/5
6892/6892 [==============================] - 14s 2ms/step - loss: 0.0033 - accuracy: 0.9994
Epoch 4/5
6892/6892 [==============================] - 14s 2ms/step - loss: 0.0030 - accuracy: 0.9994
Epoch 5/5
6892/6892 [==============================] - 13s 2ms/step - loss: 0.0029 - accuracy: 0.9994

<keras.callbacks.History at 0x2d3162bd3d0>
```

```
score = model.evaluate(X_test, y_test)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(score[1]*100,score[0]))
```

```
2154/2154 [==============================] - 3s 1ms/step - loss: 0.0041 - accuracy: 0.9992
Test Accuracy: 99.92%
Test Loss: 0.00409175269305706
```

```
import seaborn as sns
y_pred = model.predict(X_test)
y_test = pd.DataFrame(y_test)
cm = confusion_matrix(y_test, y_pred.round())
sns.heatmap(cm, annot=True, fmt='.0f', cmap='cividis_r')
plt.show()
```

```
2154/2154 [==============================] - 3s 1ms/step
```



مدل دوم:

```
model_ = Sequential([
    Dense(units=10, input_dim = X_train.shape[1], activation='relu'),
    Dense(units=20,activation='relu'),
    Dropout(0.5),
    Dense(units=27,activation='relu'),
    Dense(units=30,activation='relu'),
    Dense(1, activation='sigmoid')
])
model_.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 10)                300

 dense_6 (Dense)             (None, 20)                220

 dropout_1 (Dropout)         (None, 20)                0

 dense_7 (Dense)             (None, 27)                567

 dense_8 (Dense)             (None, 30)                840

 dense_9 (Dense)             (None, 1)                 31

=================================================================
Total params: 1,958
Trainable params: 1,958
Non-trainable params: 0
```

```
model_.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_.fit(X_train, y_train, batch_size=30, epochs=5)
```

```
Epoch 1/5
6892/6892 [==============================] - 12s 2ms/step - loss: 0.0112 - accuracy: 0.9986
Epoch 2/5
6892/6892 [==============================] - 13s 2ms/step - loss: 0.0039 - accuracy: 0.9993
Epoch 3/5
6892/6892 [==============================] - 13s 2ms/step - loss: 0.0034 - accuracy: 0.9994
Epoch 4/5
6892/6892 [==============================] - 14s 2ms/step - loss: 0.0033 - accuracy: 0.9994
Epoch 5/5
6892/6892 [==============================] - 12s 2ms/step - loss: 0.0032 - accuracy: 0.9994

<keras.callbacks.History at 0x2d3194c2940>
```

```
score = model_.evaluate(X_test, y_test)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(score[1]*100,score[0]))
```

```
2154/2154 [==============================] - 3s 1ms/step - loss: 0.0040 - accuracy: 0.9992
Test Accuracy: 99.92%
Test Loss: 0.004038030747324228
```

```
y_pred = model_.predict(X_test)
y_test = pd.DataFrame(y_test)
cm = confusion_matrix(y_test, y_pred.round())
sns.heatmap(cm, annot=True, fmt='.0f', cmap='cividis_r')
plt.show()
```

```
2154/2154 [==============================] - 3s 1ms/step
```