Credit Card Fraud Detection Scikit-Learn and Snap ML

Project goal: to recognize fraudulent credit card transactions.

- Models are: **Decision Tree and Support Vector Machine**.
- The dataset includes information about transactions made by credit cards in September 2013 by European cardholders.

Importing Required Libraries

In [ ]:
```
# Import the libraries we need to use in this lab
import warnings
warnings.filterwarnings('ignore')

# from __future__ import print_function
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize, StandardScaler
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import roc_auc_score
import time
import gc, sys
```

We have access to a dataset that is highly unbalanced. This is also the case of the current dataset: only 492 transactions out of 284,807 are fraudulent.

In [ ]:
```
# read the input data
raw_data = pd.read_csv('creditcard.csv')
```

```
print("There are " + str(len(raw_data)) + " observations in the credit card frau
print("There are " + str(len(raw_data.columns)) + " variables in the dataset.")

# display the first rows in the dataset
raw_data.head()
```

```
There are 284807 observations in the credit card fraud dataset.
There are 31 variables in the dataset.
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0

5 rows × 31 columns

In practice, a financial institution may have access to a much larger dataset of transactions.

In [ ]:

```
n_replicas = 10

# inflate the original dataset
big_raw_data = pd.DataFrame(np.repeat(raw_data.values, n_replicas, axis=0), colu

print("There are " + str(len(big_raw_data)) + " observations in the inflated cre
print("There are " + str(len(big_raw_data.columns)) + " variables in the dataset

# display first rows in the new dataset
big_raw_data.head()
```

```
There are 2848070 observations in the inflated credit card fraud dataset.
There are 31 variables in the dataset.
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
1	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
2	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
3	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
4	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098

5 rows × 31 columns

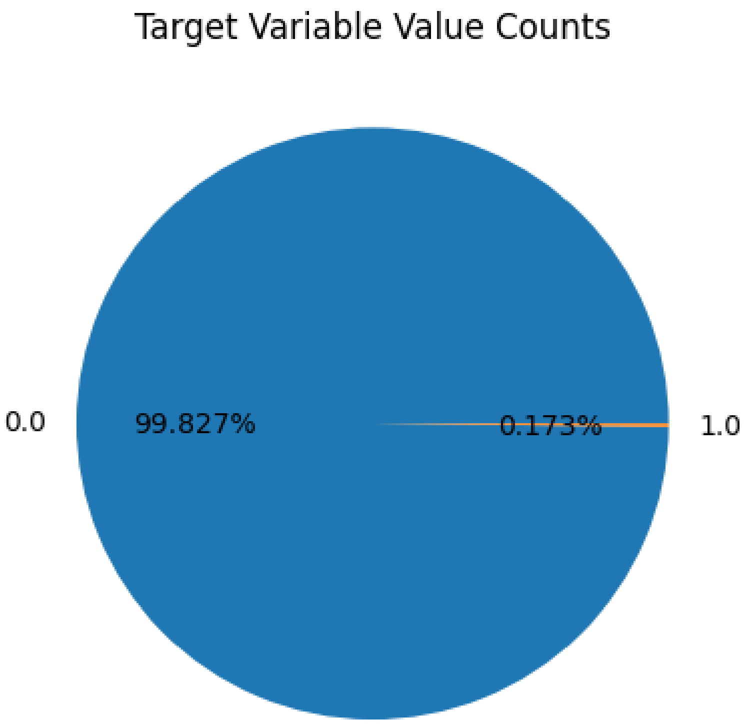
In [ ]:

```
# get the set of distinct classes
labels = big_raw_data.Class.unique()

# get the count of each class
sizes = big_raw_data.Class.value_counts().values

# plot the class value counts
fig, ax = plt.subplots()
```

```
ax.pie(sizes, labels=labels, autopct='%1.3f%%')
ax.set_title('Target Variable Value Counts')
plt.show()
```



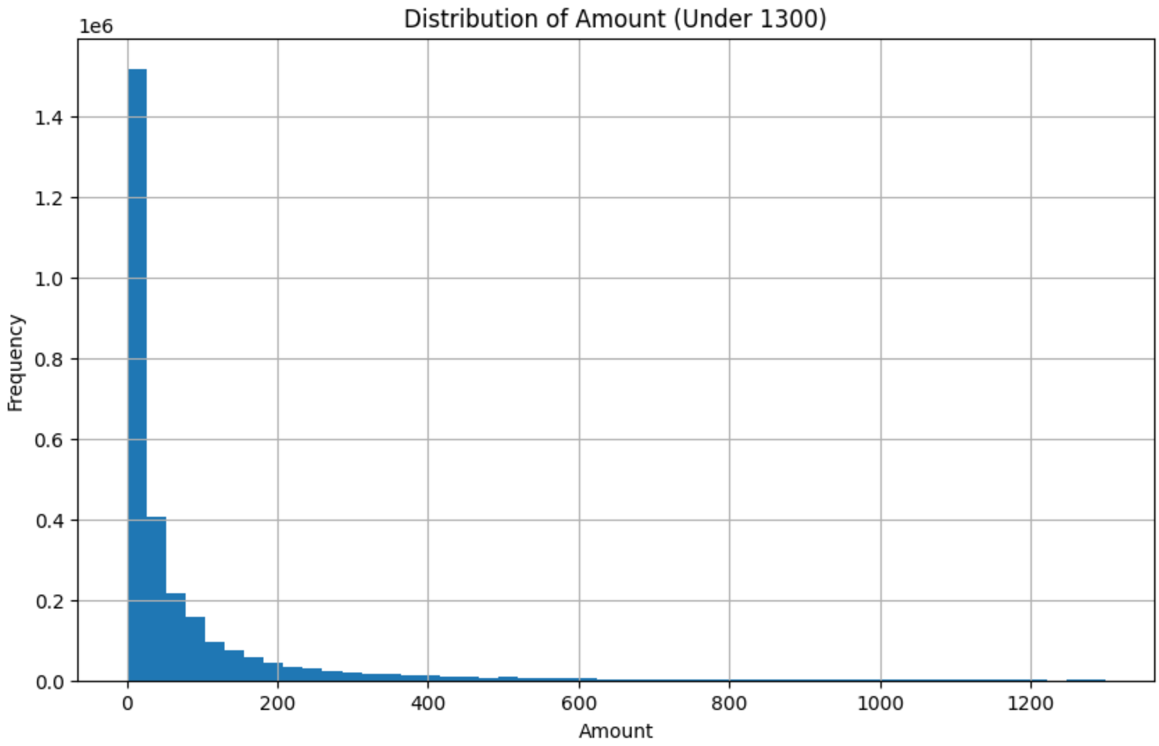
As shown above, the Class variable has two values:

- 0 (the credit card transaction is legitimate)
- 1 (the credit card transaction is fraudulent)

In [ ]:

```
# Filter the data for amounts under 5000
filtered_data = big_raw_data[big_raw_data['Amount'] < 1300]

# Distribution histogram of amount under 5000
plt.figure(figsize=(10, 6))
plt.hist(filtered_data['Amount'], bins=50)
plt.title('Distribution of Amount (Under 1300)')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



In [ ]:
```
# Range of amounts (min/max)
amount_min = big_raw_data['Amount'].min()
amount_max = big_raw_data['Amount'].max()
print('Range of Amounts (min/max):', amount_min, '/', amount_max)
```

```
Range of Amounts (min/max): 0.0 / 25691.16
```

In [ ]:
```
# 90th percentile of amount values
amount_90th_percentile = np.percentile(big_raw_data['Amount'], 90)
print('90th percentile of Amount values:', amount_90th_percentile)
```

```
90th percentile of Amount values: 203.0
```

Data preprocessing such as scaling/normalization is typically useful for linear models to accelerate the training convergence. We standardize features by removing the mean and scaling to unit variance.

In [ ]:
```
big_raw_data.iloc[:, 1:30] = StandardScaler().fit_transform(big_raw_data.iloc[:,
data_matrix = big_raw_data.values

# X: feature matrix (for this analysis, we exclude the Time variable from the da
X = data_matrix[:, 1:30]

# y: labels vector
y = data_matrix[:, 30]

# data normalization
X = normalize(X, norm="l1")

# print the shape of the features matrix and the labels vector
print('X.shape=', X.shape, 'y.shape=', y.shape)
```

```
X.shape= (2848070, 29) y.shape= (2848070,)
```

In [ ]:
```
del raw_data
del big_raw_data
gc.collect()
```

```
Out[ ]: 6059
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
print('X_train.shape=', X_train.shape, 'Y_train.shape=', y_train.shape)
print('X_test.shape=', X_test.shape, 'Y_test.shape=', y_test.shape)
```

```
X_train.shape= (1993649, 29) Y_train.shape= (1993649,)
X_test.shape= (854421, 29) Y_test.shape= (854421,)
```

```python
w_train = compute_sample_weight('balanced', y_train)
```

Import the Decision Tree Classifier Model from scikit-learn:

```python
from sklearn.tree import DecisionTreeClassifier
```

For reproducible output across multiple function calls, set `random_state` to a given integer value:

```python
sklearn_dt = DecisionTreeClassifier(max_depth=4, random_state=35)
```

Train a Decision Tree Classifier using scikit-learn and use the function **time** to record the training time of our Decision Tree model.

```python
t0 = time.time()
sklearn_dt.fit(X_train, y_train, sample_weight=w_train)
sklearn_time = time.time()-t0
print("[Scikit-Learn] Training time (s):  {0:.5f}".format(sklearn_time))
```

```
[Scikit-Learn] Training time (s):  53.69087
```

```python
from snapml import DecisionTreeClassifier
```

```python
snapml_dt = DecisionTreeClassifier(max_depth=4, random_state=45, n_jobs=4)
```

```python
# train a Decision Tree Classifier model using Snap ML
t0 = time.time()
snapml_dt.fit(X_train, y_train, sample_weight=w_train)
snapml_time = time.time()-t0
print("[Snap ML] Training time (s):  {0:.5f}".format(snapml_time))
```

```
[Snap ML] Training time (s):  7.18752
```

```python
# Snap ML vs Scikit-Learn training speedup
training_speedup = sklearn_time/snapml_time
print('[Decision Tree Classifier] Snap ML vs. Scikit-Learn speedup : {0:.2f}x '.
```

```
[Decision Tree Classifier] Snap ML vs. Scikit-Learn speedup : 7.47x
```

```python
sklearn_pred = sklearn_dt.predict_proba(X_test)[:,1]
snapml_pred = snapml_dt.predict_proba(X_test)[:,1]
```

```python
sklearn_roc_auc = roc_auc_score(y_test, sklearn_pred)
print('[Scikit-Learn] ROC-AUC score : {0:.3f}'.format(sklearn_roc_auc))

snapml_roc_auc = roc_auc_score(y_test, snapml_pred)
print('[Snap ML] ROC-AUC score : {0:.3f}'.format(snapml_roc_auc))
```

```
[Scikit-Learn] ROC-AUC score : 0.966
[Snap ML] ROC-AUC score : 0.966
```

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

In [ ]:
```
sklearn_svm = LinearSVC(class_weight='balanced', random_state=31, loss="hinge",
```

Train a linear Support Vector Machine model using Scikit-Learn:

In [ ]:
```
t0 = time.time()
sklearn_svm.fit(X_train, y_train)
sklearn_time = time.time() - t0
print("[Scikit-Learn] Training time (s):  {0:.2f}".format(sklearn_time))
```

```
[Scikit-Learn] Training time (s):  109.16
```

In [ ]:
```
from snapml import SupportVectorMachine
```

In [ ]:
```
snapml_svm = SupportVectorMachine(class_weight='balanced', random_state=25, n_jc
print(snapml_svm.get_params())
```

```
{'class_weight': 'balanced', 'device_ids': [], 'fit_intercept': False, 'gamma':
1.0, 'generate_training_history': None, 'intercept_scaling': 1.0, 'kernel': 'line
ar', 'loss': 'hinge', 'max_iter': 1000, 'n_components': 100, 'n_jobs': 4, 'normal
ize': False, 'random_state': 25, 'regularizer': 1.0, 'tol': 0.001, 'use_gpu': Fal
se, 'verbose': False}
```

Train an SVM model using Snap ML:

In [ ]:
```
t0 = time.time()
model = snapml_svm.fit(X_train, y_train)
snapml_time = time.time() - t0
print("[Snap ML] Training time (s):  {0:.2f}".format(snapml_time))
```

```
[Snap ML] Training time (s):  15.07
```

In [ ]:
```
# compute the Snap ML vs Scikit-Learn training speedup
training_speedup = sklearn_time/snapml_time
print('[Support Vector Machine] Snap ML vs. Scikit-Learn training speedup : {0:.
```

```
[Support Vector Machine] Snap ML vs. Scikit-Learn training speedup : 7.24x
```

In [ ]:
```
sklearn_pred = sklearn_svm.decision_function(X_test)
snapml_pred = snapml_svm.decision_function(X_test)
```

In [ ]:
```
acc_sklearn  = roc_auc_score(y_test, sklearn_pred)
print("[Scikit-Learn] ROC-AUC score:   {0:.3f}".format(acc_sklearn))

acc_snapml  = roc_auc_score(y_test, snapml_pred)
print("[Snap ML] ROC-AUC score:   {0:.3f}".format(acc_snapml))
```

```
[Scikit-Learn] ROC-AUC score:   0.984
[Snap ML] ROC-AUC score:   0.985
```

In [ ]:
```
from sklearn.metrics import hinge_loss

# get the confidence scores for the test samples
sklearn_pred = sklearn_svm.decision_function(X_test)
snapml_pred  = snapml_svm.decision_function(X_test)
```

```
# evaluate the hinge loss metric for Sklearn
loss_sklearn = hinge_loss(y_test, sklearn_pred)
print("[Scikit-Learn] Hinge loss:   {0:.3f}".format(loss_sklearn))

# evaluate the hinge loss for Snap ML
loss_snapml = hinge_loss(y_test, snapml_pred)
print("[Snap ML] Hinge loss:   {0:.3f}".format(loss_snapml))
```

```
[Scikit-Learn] Hinge loss:   0.234
[Snap ML] Hinge loss:   0.228
```