Credit Card Fraud - Prevention through Data Science

Brianne Bell, Michael Nguyen, and Dave Friesen
ADS-505-02-FA22

In [1]:
```
__author__ = 'Brianne Bell, Michael Nguyen, Dave Friesen'
__email__ = 'bbell@sandiego.edu, michaelnguyen@sandiego.edu, dfriesen@sandiego.edu'
__version__ = '1.0'
__date__ = 'October 2022'
__license__ = 'MIT'
```

Setup

In [2]:
```
# Set working directory for 'custom' (profiler, model_process) .py find and data access
import os, sys
src_dir = '/Users/davidfriesen/Desktop/OneDrive/projects/cc-fraud-protection/src'
data_dir = '/Users/davidfriesen/Desktop/OneDrive/projects/cc-fraud-protection/data/'
sys.path.append(src_dir)
os.chdir(data_dir)
```

In [3]:
```
# Import basic libraries
import numpy as np
import pandas as pd

# Import visualization libraries
import matplotlib.pyplot as plt
%matplotlib inline

# Import custom EDA library
from profiler import profile, profile_cat

# Import model and performance evaluation libraries
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import LinearSVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier

from sklearn.neural_network import MLPClassifier

# Import custom model library
from model_process import ModelProcess

# Import utility libraries
import copy
```

In [4]:
```
# Set basic np, pd, and plt output defaults (keeping this code 'clean')
%run -i '{src_dir}/defaults.py'
```

Set simple notebook style and auto-avoid scrollable output windows

In [5]:
```
%%html
<style>
    h1, h2 {font-size: 0.9em; color: #0070c0;}
    h1, h2, h3 {line-height: 1.2em !important; margin-top: 2em !important;}
    h4, p {font-size: 1.2em !important; line-height: 1.2em !important;}
</style>
```

In [6]:
```
%%javascript
    IPython.OutputArea.prototype._should_scroll = function(lines) {
        return false;
    }
```

Data Load and Validation

In [7]:
```
fraud_fname = data_dir + 'c-Fraud_Data.csv'
ip_fname = data_dir + 'c-IpAddress_to_Country.csv'

# Get a file row count; this is defining a pattern for a simple control
#   which confirms data load
def ctrl_count(fname):
    f = open(fname)
    count = sum(1 for line in f)
    f.close()
    return count
fraud_ctrl = ctrl_count(fraud_fname)
ip_ctrl = ctrl_count(ip_fname)

# Create and confirm dataframe(s)
fraud_df = pd.read_csv(fraud_fname, on_bad_lines = 'skip', low_memory = False)
ip_df = pd.read_csv(ip_fname, on_bad_lines = 'skip', low_memory = False)

# Now confirm control count (delta of 1 related to expected header row)
print('\nFraud: file=%0d, import=%0d, delta=%0d' %
      (fraud_ctrl, len(fraud_df), fraud_ctrl - len(fraud_df)))
print('\nIP: file=%0d, import=%0d, delta=%0d' %
      (ip_ctrl, len(ip_df), ip_ctrl - len(ip_df)))

#fraud_df.info()
#fraud_df.head(5)
#ip_df.info()
#ip_df.head(5)

# Dataframe comparision e.g., if have sep training/test sets - n/a here
#print('\nSame? ', np.array_equal(fundr_df.columns, ffundr_df.columns))
```

```
Fraud: file=151113, import=151112, delta=1

IP: file=138847, import=138846, delta=1
```

Data Profiling

In [8]:
```
# Profile base dataframe(s)
profile(fraud_df)
profile(ip_df)
```

	Dtype	count	unique	na	na%	mean	std	min	max	skew(>=3)	<v0.01	VIF(>=10)
user_id	int64	151112	151112			200171.0	115369.3	2.0	400000.0			
signup_time	object	151112	151112									
purchase_time	object	151112	150679									
purchase_value	int64	151112	122			36.9	18.3	9.0	154.0			
device_id	object	151112	137956									
source	object	151112	3									
browser	object	151112	5									
sex	object	151112	2									
age	int64	151112	58			33.1	8.6	18.0	76.0			
ip_address	float64	151112	143512			2152145331.0	1248497030.1	52093.5	4294850499.7			
class	int64	151112	2			0.1	0.3		1.0			

	Dtype	count	unique	na	na%	mean	std	min	max	skew(>=3)	<v0.0
lower_bound_ip_address	float64	138846	138846			2724531562.5	897521519.7	16777216.0	3758096128.0		
upper_bound_ip_address	int64	138846	138846			2724557062.2	897497915.5	16777471.0	3758096383.0		
country	object	138846	235								

Univariate Analysis and Data Preparation

Attribute Names (confirm/update)

In [9]:
```
# Rename columsn to simplify and/or avoid usage issues later - tbd

# examples:
#cols = {'homeowner dummy': 'homeowner', 'gender dummy': 'gender'}
#fundr_df.rename(columns=cols, inplace=True)
```

Target and Preliminary Feature Identification *(preliminary)*

In [10]:
```
# Preliminarily identify attributes and types for best further processing
id_nominal_cols = ['user_id', 'device_id', 'ip_address']

cat_nominal_cols = ['source', 'browser', 'sex']
cat_ordinal_cols = []
cat_binary_cols = []

date_interval_cols = ['signup_time', 'purchase_time']

num_interval_cols = ['age']
num_ratio_cols = ['purchase_value']

# Set classification target
target_cls_col = ['class']

# Set regression target (n/a for this excercise)
target_reg_col = []

# Set preliminary feature reduction (removed below)
reduce_X_cols = id_nominal_cols + date_interval_cols + target_cls_col + target_reg_col
```

Data Types (confirm/update)

In [11]:
```
# Update column types where needed (helpful) - tbd

# examples:
#df[cat_nominal_cols] = df[cat_nominal_cols].astype('Int64')
#df[date_interval_cols] = df[date_interval_cols].astype('datetime64')
```

Data Enrichment

In [12]:
```
# tbd:
# signup_time? purchase_time?
# ip_address?
```

Categorical Data Profiling

In [13]:
```
# Better understand 'categorical' data, including balance
profile_cat(fraud_df, cat_nominal_cols + cat_ordinal_cols + cat_binary_cols + target_cls_col)
```

```
source -
   SEO 40.11
   Ads 39.63
Direct 20.26

browser -
 Chrome 40.65
     IE 24.30
 Safari 16.32
FireFox 16.29
  Opera  2.43

sex -
M 58.43
F 41.57

class -
0 90.64
1  9.36
```

Data Partitioning

In [14]:
```
# Create predictor and target dataframes
fraud_X = fraud_df.loc[:, ~fraud_df.columns.isin(reduce_X_cols)].copy()
fraud_y = fraud_df[target_cls_col]
```

In [15]:
```
# Split data and confirm proportions
train_ratio = 0.7; val_ratio = 0.20; test_ratio = 0.10

X_train, X_test, y_train, y_test = train_test_split(
    fraud_X, fraud_y, test_size=1-train_ratio,
    random_state=42, stratify=fraud_y)
X_val, X_test, y_val, y_test = train_test_split(
    X_test, y_test, test_size=test_ratio/(test_ratio+val_ratio),
    random_state=42, stratify=y_test)

trows = fraud_X.shape[0]
print('\nTrain/validation/test: ', X_train.shape[0], '/', X_val.shape[0], '/', X_test.shape[0])

profile_cat(y_train, target_cls_col)
```

```
Train/validation/test:  105778 / 30222 / 15112

class -
0 90.64
1  9.36
```

In [16]:
```
# Potentially create test set from independent source data - n/a here
#X_test = ffundr_df.loc[:, ~ffundr_df.columns.isin(reduce_X_cols)].copy()
#y_test = ffundr_df[target_cls_col]
```

Additional Uni/Multivariate EDA and Feature Engineering/Selection

> **Missing/null values (find/impute/drop) - not needed but handled in classification pipeline below**
> **Categorical features (encoding) - handled in classification pipeline**
> **Outliers (convert/drop) - code available below but not needed**
> **Centering/scaling (standardizing/normalizing) - handled in classification pipeline**
> **"Bad"/duplicate data (find/convert/drop) - not needed**
> **Other multivariate (e.g., correlation, etc.) - ref. below**

Outliers (convert/drop)

In [17]:
```
# Simple function to remove column-level outliers (note this needs expanding
#   to row-level and/or conversion, for best results)
def remove_outliers(df_col):
    q1 = df_col.quantile(0.1)
    q3 = df_col.quantile(0.9)
    iqr = q3 - q1
    lbound = q1 - (1.5 * iqr)
    ubound = q3 + (1.5 * iqr)
    df_out = df_col[(df_col >= lbound) & (df_col <= ubound)]
    removed = len(df_col) - len(df_out)
    if removed > 0:
        print(df_col.name, 'outliers removed: ', removed)
    return df_out

# Per note above, holding this out here but example:
#for c in num_ratio_cols:
#    fraud_df[c] = remove_outliers(fraud_df[c])
```

Correlation Analysis

In [18]:
```
# Show ~relatively strong correlations to consider, using approach similar to
#   what is used in profile() function to highlight >= threshhold
CORR_TH = 0.5
corr_df = pd.DataFrame(
    X_train[num_interval_cols + num_ratio_cols].corr()\
        .apply(lambda x: abs(x*(x>=CORR_TH)))
corr_df = round(corr_df, 1).astype(str)
corr_df.replace(['0', '0.0', 'nan', 'False'], '', inplace=True)
corr_df
```

Out[18]:

	age	purchase_value
age	1.0	
purchase_value		1.0

In [19]:
```
# To consider for categorical value correlation?

def cramers_corrected_stat(confusion_matrix):
    """ calculate Cramers V statistic for categorical-categorical association.
        uses correction from Bergsma and Wicher,
        Journal of the Korean Statistical Society 42 (2013): 323-328
    """
    chi2 = ss.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min( (kcorr-1), (rcorr-1)))

#cols = bin_cols
#corrM = np.zeros((len(cols),len(cols)))
# there's probably a nice pandas way to do this
#for col1, col2 in itertools.combinations(cols, 2):
#    idx1, idx2 = cols.index(col1), cols.index(col2)
#    corrM[idx1, idx2] = cramers_corrected_stat(pd.crosstab(building_df[col1], building_df[col2]))
#    corrM[idx2, idx1] = corrM[idx1, idx2]

#corr = pd.DataFrame(corrM, index=cols, columns=cols)
#fig, ax = plt.subplots(figsize=(7, 6))
#ax = sns.heatmap(corr, annot=True, ax=ax); ax.set_title("Cramer V Correlation between Variables");
```

Data Mining (Unsupervised)

> **Including a placeholder in case we see an opportunity to apply e.g., clustering from coursework. . .**

Modeling

Model Setup (selection)

In [20]:
```
# Set feature cols for appropriate pipeline preprocessing
cat_cols = cat_nominal_cols + cat_ordinal_cols + cat_binary_cols  # one-hot encoding, imputing (if necc)
num_cols = num_interval_cols + num_ratio_cols  # scaling, imputing (if necc)

# Set model list
mp_queue = (
    (LogisticRegression(), {'random_state': 42}),
    (Perceptron(), {'class_weight': 'balanced'}),
    (LinearDiscriminantAnalysis(), None),
    (LinearSVC(), {'max_iter': 500}),

    (KNeighborsClassifier(), {'n_neighbors': 3}),
    (KNeighborsClassifier(), {'n_neighbors': 5}),
    (KNeighborsClassifier(), {'n_neighbors': 7}),

    (DecisionTreeClassifier(), {'max_depth': 4, 'random_state': 42}),
    (DecisionTreeClassifier(), {'max_depth': 5, 'random_state': 42}),

    (RandomForestClassifier(), {'max_depth': 4, 'random_state': 42}),
    (RandomForestClassifier(), {'max_depth': 5, 'random_state': 42}),

    (AdaBoostClassifier(), {'n_estimators': 10, 'random_state': 42}),

    (MLPClassifier(), {'random_state': 42}),
)
```

Model Run and Evaluation (Iteration n)

In [21]:
```
ModelProcess.show_progress = True

# Iterate models (note use of 'copy' is to preserve mutable elements
#   of model_queue tuple for possible later use)
mp_df = pd.DataFrame(mp_queue, columns=['algorithm', 'params'])
mp_df['mp'] = mp_df.apply(
    lambda mp: ModelProcess(copy.deepcopy(mp['algorithm']), None,
                            copy.copy(mp['params']),
                            X_train, y_train,
                            X_val, y_val,
                            X_test, y_test,
                            cat_cols, num_cols).train_validate_test(), axis=1)

# Compile, sort, and display results
mp_df[['train_acc', 'train_f1', 'val_acc', 'val_f1', 'test_acc', 'test_f1']] =\
    mp_df['mp'].apply(
        lambda mp: sum(list(map(
            lambda dataset: mp.score[dataset], ['train', 'val', 'test'])), [])).tolist()
mp_df.sort_values(by=['train_acc', 'val_acc', 'test_acc'],
                  ascending=[False, False, False], inplace=True)
mp_df.loc[:, mp_df.columns != 'mp']
```

Out[21]:

	algorithm	params	train_acc	train_f1	val_acc	val_f1	test_acc	test_f1
4	KNeighborsClassifier()	{'n_neighbors': 3}	0.93	0.77	0.92	0.73	0.92	0.73
5	KNeighborsClassifier()	{'n_neighbors': 5}	0.93	0.76	0.92	0.74	0.92	0.73
6	KNeighborsClassifier()	{'n_neighbors': 7}	0.93	0.76	0.92	0.74	0.92	0.74
8	DecisionTreeClassifier()	{'max_depth': 5, 'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
12	MLPClassifier()	{'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
7	DecisionTreeClassifier()	{'max_depth': 4, 'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
0	LogisticRegression()	{'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
2	LinearDiscriminantAnalysis()	None	0.91	0.48	0.91	0.48	0.91	0.48
3	LinearSVC()	{'max_iter': 500}	0.91	0.48	0.91	0.48	0.91	0.48
9	RandomForestClassifier()	{'max_depth': 4, 'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
10	RandomForestClassifier()	{'max_depth': 5, 'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
11	AdaBoostClassifier()	{'n_estimators': 10, 'random_state': 42}	0.91	0.48	0.91	0.48	0.91	0.48
1	Perceptron()	{'class_weight': 'balanced'}	0.77	0.49	0.77	0.49	0.77	0.50

In [22]:
```
#mp_df.loc[3]['mp'].confusion_matrix('val')
#mp_df.loc[3]['mp'].summary('val')
```