# focus_lab_cbm

June 5, 2024

# 1 Import data

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

import trimap

# import jax.random as random
```

We can split the data into input variables `X` and our output variable `y`. I'm taking the first 4 months of the data in case there is significant time drift.

[ ]:
```
base_df = pd.read_csv('Base.csv')

target = 'fraud_bool'
X = base_df.drop(target, axis = 1)
y = base_df[target]

# First 4 months of the data
X_4 = X[X['month'] < 2]
y_4 = y[X['month'] < 2]
# X_4.size
```

# 2 EDA

## 2.1 Variables by type

Categorical input variables:

- `payment_type`: 5 values `['AA', 'AD', 'AB', 'AC', 'AE']`
- `employment_status`: 7 values `['CB', 'CA', 'CC', 'CF', 'CD', 'CE', 'CG']`
- `housing_status`: 7 values `['BC', 'BE', 'BD', 'BA', 'BB', 'BF', 'BG']`
- `source`: 2 values `['INTERNET', 'TELEAPP']`
- `device_os`: 5 values `['linux', 'other', 'windows', 'x11', 'macintosh']`

Binary input variables:

- `email_is_free`
- `phone_home_valid`
- `phone_mobile_valid`
- `has_other_cards`
- `foreign_request`
- `keep_alive_session`

Binned numeric variables:

- `income`: binned to the nearest decile, e.g. 0.1, 0.2, ..., 0.9
- `customer_age`: binned to the nearest decade, e.g. 10, 20, ..., 90
- `proposed_credit_limit`: looks like it might be rounded to the nearest 10

Continuous numeric variables:

- `device_fraud_count` is always 0

[ ]:
```python
# Numerical (continuous/discrete) and categorical features

num_feats = X.select_dtypes(include='number').columns.tolist()

thresh = 10 # lowered threshold compared to notebook to make␣
↪proposed_credit_limit continuous

cont_feats = [feat for feat in num_feats if base_df[feat].nunique() >= thresh]
bool_feats = [feat for feat in num_feats if base_df[feat].nunique() == 2]
disc_feats = [feat for feat in num_feats if base_df[feat].nunique() < thresh␣
↪and feat not in bool_feats]

cat_feats = X.select_dtypes(exclude='number').columns.tolist()

print(f'Features: {X.shape[1]}\n\n\
Continuous: {len(cont_feats)}\n\
{cont_feats}\n\n\
Boolean: {len(bool_feats)}\n\
{bool_feats}\n\n\
Discrete or Binned: {len(disc_feats)}\n\
{disc_feats}\n\n\
Categorical: {len(cat_feats)}\n\
{cat_feats}')
```

```
Features: 31

Continuous: 15
['name_email_similarity', 'prev_address_months_count',
'current_address_months_count', 'days_since_request', 'intended_balcon_amount',
'zip_count_4w', 'velocity_6h', 'velocity_24h', 'velocity_4w',
'bank_branch_count_8w', 'date_of_birth_distinct_emails_4w', 'credit_risk_score',
```

```
'bank_months_count', 'proposed_credit_limit', 'session_length_in_minutes']

Boolean: 6
['email_is_free', 'phone_home_valid', 'phone_mobile_valid', 'has_other_cards',
'foreign_request', 'keep_alive_session']

Discrete or Binned: 5
['income', 'customer_age', 'device_distinct_emails_8w', 'device_fraud_count',
'month']

Categorical: 5
['payment_type', 'employment_status', 'housing_status', 'source', 'device_os']
```

### 2.2 Missingness:

-1 indicates missing value in:

- `prev_address_months_count`
- `current_address_months_count`
- `bank_months_count`
- `session_length_in_minutes`
- `device_distinct_emails_8w`

Any negative represents a missing value in `intended_balcon_amount`.

## 3 PCA

We start by performing PCA on the first 4 months of the data using the continuous numeric variables only.

[ ]:
```python
# Extract continuous numeric variables
# Scale them
X_num_cont = X_4[cont_feats]
scaler = StandardScaler()
X_num_cont = scaler.fit_transform(X_num_cont)

# PCA
pca_num_cont = PCA(n_components = 2)
pca_num_cont.fit(X_num_cont)
print(pca_num_cont.explained_variance_ratio_)
```

```
[0.12142462 0.10307044]
```

[ ]:
```python
# Plot PCA
X_num_cont_pca = pca_num_cont.transform(X_num_cont)
plt.scatter(X_num_cont_pca[y_4 == False, 0],
            X_num_cont_pca[y_4 == False, 1],
            c = "blue",
            s = 5,
```

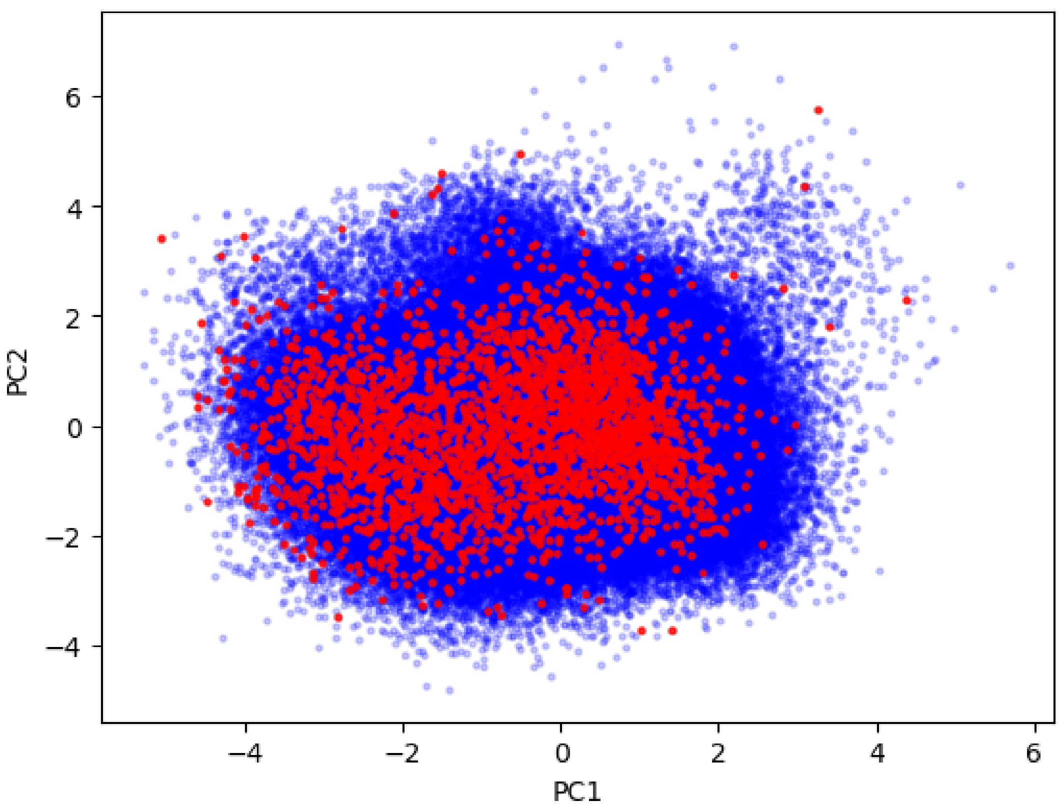
```
            alpha = 0.2)
plt.scatter(X_num_cont_pca[y_4 == True, 0],
            X_num_cont_pca[y_4 == True, 1],
            c = "red",
            s = 5,
            alpha = 0.8)

# Get the loadings
loadings_num_cont = pca_num_cont.components_
# print(loadings_num_cont)

# Plot the loadings
# for i, (comp1, comp2) in enumerate(zip(loadings_num_cont[0],␣
 ↪loadings_num_cont[1])):
#     plt.arrow(0, 0, comp1, comp2, color = 'r', alpha = 0.5)
#     if cont_feats is not None:
#         plt.text(comp1, comp2, cont_feats[i], color = 'g', ha = 'center', va␣
 ↪= 'center')

plt.xlabel('PC1')
plt.ylabel('PC2')
```

[ ]: Text(0, 0.5, 'PC2')



Let's try adding the binned variables 'income', 'customer_age', and 'device_distinct_emails_8w' (treating them as if they are continous since at least they are numeric).

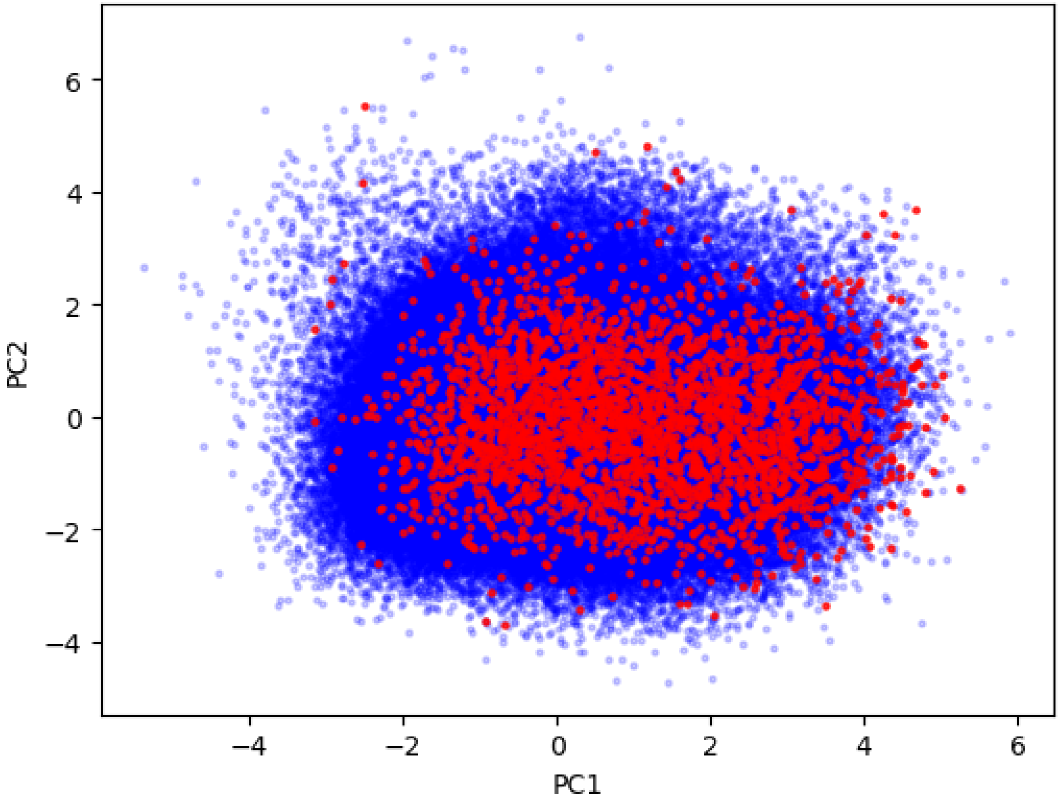
[ ]:
```
# Extract continuous numeric variables
# Scale them
X_num_cont_bin = X_4[cont_feats + ['income', 'customer_age',␣
↪'device_distinct_emails_8w']]
X_num_cont_bin = scaler.fit_transform(X_num_cont_bin)

# PCA
pca_num_cont_bin = PCA(n_components = 2)
pca_num_cont_bin.fit(X_num_cont_bin)
print(pca_num_cont_bin.explained_variance_ratio_)
```

```
[0.11264159 0.08603202]
```

[ ]:
```
# Plot PCA
X_num_cont_bin_pca = pca_num_cont_bin.transform(X_num_cont_bin)
plt.scatter(X_num_cont_bin_pca[y_4 == False, 0],
            X_num_cont_bin_pca[y_4 == False, 1],
            c = "blue",
            s = 5,
            alpha = 0.2)
plt.scatter(X_num_cont_bin_pca[y_4 == True, 0],
            X_num_cont_bin_pca[y_4 == True, 1],
            c = "red",
            s = 5,
            alpha = 0.8)
plt.xlabel('PC1')
plt.ylabel('PC2')
```

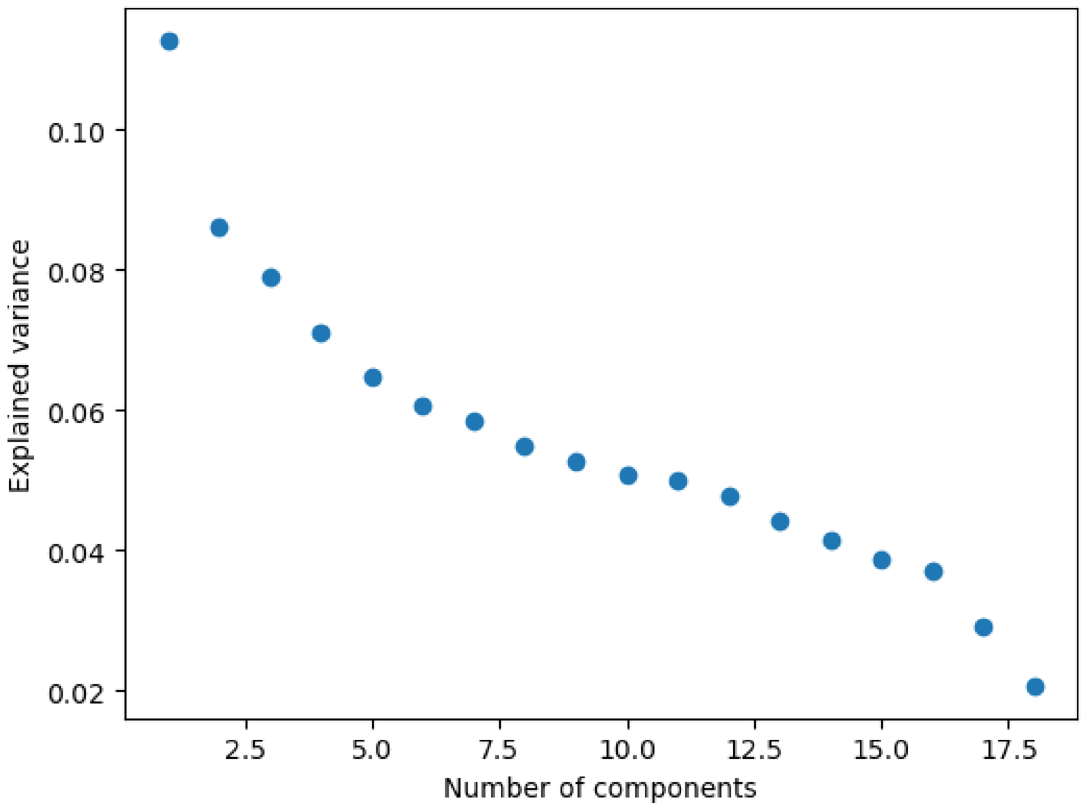
[ ]:
```
Text(0, 0.5, 'PC2')
```




[ ]:
```
# Elbow plot for number of principle components
pca_num_cont_bin = PCA()
pca_num_cont_bin.fit(X_num_cont_bin)

plt.scatter(x = range(1, len(pca_num_cont_bin.explained_variance_ratio_) + 1),
                        y = pca_num_cont_bin.explained_variance_ratio_)
plt.xlabel('Number of components')
plt.ylabel('Explained variance')
```

[ ]: Text(0, 0.5, 'Explained variance')




Fraudulent observations aren't outliers on either PCA :( Some potential issues to address:

- Address missingness: `intended_balcon_amount` and `prev_address_months_count` are missing in 70%+ of the total cases, and `bank_months_count` is missing in 25% of total cases, so probably should either exclude or impute these. Missingness might also be predictive of fraud vs. not fraud, but I don't think we can do PCA on an indicator variable?
- Binary / categorical variables: I don't think PCA on one-hot encoded variables works, maybe we can try FAMD to deal with these? Not sure how to plug FAMD into TriMap
- Number of PCs: Two principle components isn't explaining much of the variance, maybe more would help?

## 4 TriMap

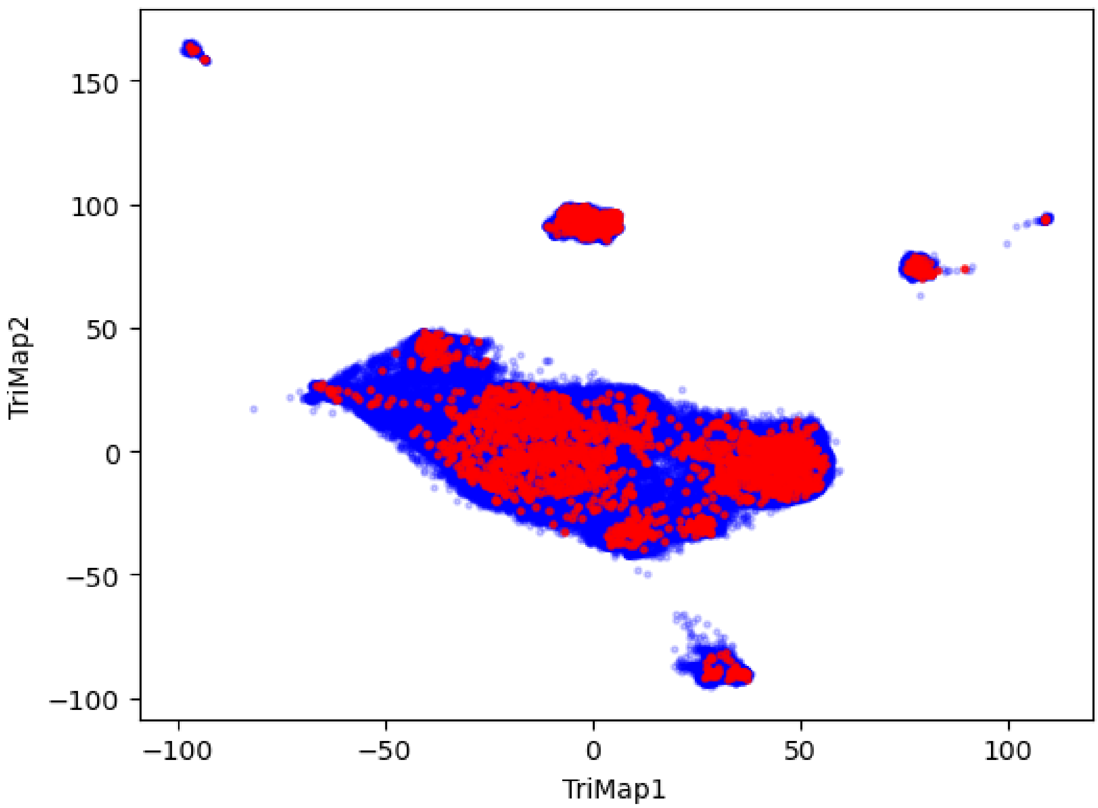
What happens if we try the default implementation of TriMap on our data?

```
[ ]: # TriMap for continous variables + income + customer_age +
       ↪device_distinct_emails_8w
     X_num_cont_bin_trimap = trimap.TRIMAP().fit_transform(X_num_cont_bin)
```

```
[ ]: # Plot TriMap
     plt.scatter(X_num_cont_bin_trimap[y_4 == False, 0],
```

```
            X_num_cont_bin_trimap[y_4 == False, 1],
            c = "blue",
            s = 5,
            alpha = 0.2)
plt.scatter(X_num_cont_bin_trimap[y_4 == True, 0],
            X_num_cont_bin_trimap[y_4 == True, 1],
            c = "red",
            s = 5,
            alpha = 0.8)
plt.xlabel('TriMap1')
plt.ylabel('TriMap2')
```

[ ]: Text(0, 0.5, 'TriMap2')




Like the PCA, the fraudulent cases are actually contained within the larger cloud of non-fraudulent cases. Also, clusters of outliers in the TriMap aren't necessarily fraud. We probably still have the same issues of not dealing with missingness correctly, leaving out the binary / categorical variables, etc.