Week 10 — Data Centric AI

In [1]:
```
import pandas as pd
import numpy as np
# import dask.dataframe as dd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from datetime import datetime
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from collections import defaultdict
from sklearn.preprocessing import MinMaxScaler
from joblib import Parallel, delayed
```

In [2]:
```
pd.set_option('display.max_columns', None)

train = pd.read_csv('train_final.csv', low_memory=False)
validation = pd.read_csv('val_set_final.csv')
test = pd.read_csv('test_4_11.csv')
```

In [3]:
```
train.head()
```

Out[3]:

	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	new_cust	sen
0	0	2016-04-28	1334092	N	1	0	0.234694	0	
1	1	2015-07-28	1024586	N	1	0	0.234694	0	
2	2	2016-04-28	856204	N	1	0	0.306122	0	
3	3	2015-08-28	295807	N	1	0	0.489796	0	
4	4	2016-03-28	942624	N	1	1	0.224490	0	

In [4]:
```
validation.head()
```

Out[4]:

	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	first_contract_(
0	0	2016-05-28	1212130	N	1	0	0.204082	2013-1
1	1	2015-07-28	84306	N	1	0	0.500000	1998-0
2	2	2015-07-28	883630	N	1	0	0.418367	2010-0
3	3	2016-05-28	1464700	N	1	1	0.183673	2015-0
4	4	2015-12-28	487783	N	1	1	0.418367	2004-1

In [5]:

```
test.head()
```

Out[5]:

	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	new_cust	sen
0	0	2015-06-28	49335	N	1	0	0.734694	0	
1	1	2016-02-28	1174349	N	1	0	0.214286	0	
2	2	2015-07-28	1393286	N	1	0	0.244898	1	
3	3	2016-03-28	1454346	N	1	0	0.183673	0	
4	4	2016-02-28	1074431	N	1	0	0.234694	0	

Changing columns name and dropping columns so both datasets are the same

In [6]:

```
train = train.rename(columns={'country': 'country_spain'})
```

In [7]:

```
train = train.drop(columns=['Unnamed: 0'])
validation = validation.drop(columns=['Unnamed: 0'])
drop = ['join_channel', 'province_name', 'employee_index', 'segment', 'total_products'
train = train.drop(columns=drop)
validation = validation.drop(columns=drop + ['payroll_acct.1', 'first_contract_date',

test = test.drop(columns=['Unnamed: 0'])
test = test.drop(columns=drop + ['payroll_acct.1'])
```

Reading into the data

Setting products we want to predict

In [8]:
```
products = ['savings_acct', 'guarantees', 'current_acct',
       'derivada_acct', 'payroll_acct', 'junior_acct', 'mas_particular_acct',
       'particular_acct', 'particular_plus_acct', 'short_term_depo',
       'medium_term_depo', 'long_term_depo', 'e_acct', 'funds', 'mortgage',
       'pension', 'loans', 'taxes', 'credit_card', 'securities', 'home_acct',
       'pensions_2', 'direct_debt']
```

Transformation #1

Change #1: Instead of dropping these duplicates on customer column and use only the last instance we will keep those duplicates since it could capture some patterns such as if a client buys product x first, it will likely buy y product next.

We will create copies of the original train and test datasets so we don't change the original one.

In [9]:
```
train_1 = train.copy()
```

In [10]:
```
train = train.drop_duplicates(subset=['customer_code'], keep='last')
validation = validation.drop_duplicates(subset=['customer_code'], keep='last')

# Removing customers from validation set that appear in training set
# validation = validation[~validation['customer_code'].isin(train['customer_code'])]
```

Pre-processing

Defining our Xs and Ys

In [11]:
```
train_2 = train_1.copy()
test_2 = test.copy()
train_3 = train.copy()
```

Transformation #2

For tranformation #2 we will add the date column as one of the features. For that, we will calculate the time since purchase using the month we are trying to predict on June 2016. For this transformation to make sense, we will also keep the first transformation, since the time line of purchase matters now, we will keep the duplicate clients' purchases instead of only keeping the last one

In [12]:
```
train_2['date'] = pd.to_datetime(train_2['date'], format='%Y-%m-%d')

train_2['date'] = train_2['date'].dt.to_period('M').dt.to_timestamp()

# Setting our prediction date, June 28, 2016, as the reference date
reference_date = pd.to_datetime("2016-06-28")
```

```
# Calculate time since purchase
train_2['months_since_purchase'] = (reference_date.year - train_2['date'].dt.year) * 1
                                   (reference_date.month - train_2['date'].dt.month)

print(train_2[['date', 'months_since_purchase']])
```

```
              date  months_since_purchase
0       2016-04-01                      2
1       2015-07-01                     11
2       2016-04-01                      2
3       2015-08-01                     10
4       2016-03-01                      3
...            ...                    ...
5757281 2016-05-01                      1
5757282 2015-08-01                     10
5757283 2015-11-01                      7
5757284 2016-05-01                      1
5757285 2016-01-01                      5

[5757286 rows x 2 columns]
```

In [13]:
```
# Adding feature on test dateased
test_2['date'] = pd.to_datetime(test_2['date'], format='%Y-%m-%d')
test_2['date'] = test_2['date'].dt.to_period('M').dt.to_timestamp()

test_2['months_since_purchase'] = (reference_date.year - test_2['date'].dt.year) * 12
                                  (reference_date.month - test_2['date'].dt.month)

print(test_2[['date', 'months_since_purchase']])
```

```
              date  months_since_purchase
0       2015-06-01                     12
1       2016-02-01                      4
2       2015-07-01                     11
3       2016-03-01                      3
4       2016-02-01                      4
...            ...                    ...
1236739 2016-02-01                      4
1236740 2016-02-01                      4
1236741 2015-08-01                     10
1236742 2016-05-01                      1
1236743 2016-04-01                      2

[1236744 rows x 2 columns]
```

In [14]:
```
X_train = train.drop(['customer_code', 'date'] + products, axis=1)
y_train = train[products]

X_train_1 = train_1.drop(['customer_code', 'date'] + products, axis=1)
y_train_1 = train_1[products]

X_train_3 = train_3.drop(['customer_code', 'date'] + products, axis=1)
y_train_3 = train_3[products]

X_test = test.drop(['customer_code', 'date'] + products, axis=1)
y_test = test[products]

X_train_2 = train_2.drop(['customer_code', 'date'] + products, axis=1)
y_train_2 = train_2[products]
```

```
X_test_2 = test_2.drop(['customer_code', 'date'] + products, axis=1)
y_test_2 = test_2[products]
```

Training

In [15]:
```
# Defining the best training parameter
params = {'C': 10, 'solver': 'liblinear', 'max_iter': 300}
```

Original Database

In [16]:
```
# Initialize dictionary for storing metrics
metrics = defaultdict(lambda: defaultdict(dict))

# Train and evaluate the model on the 'train' dataset
for product in products:
    clf = LogisticRegression(**params)

    # Train data and labels for each product
    y_train_product = y_train[product].values
    y_test_product = y_test[product].values

    # Train the model
    clf.fit(X_train, y_train_product)

    # Predictions
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)
    y_train_pred_proba = clf.predict_proba(X_train)[:, 1]
    y_test_pred_proba = clf.predict_proba(X_test)[:, 1]

    # Calculate metrics
    metrics['train']['train'][product] = {
        'ROC AUC': roc_auc_score(y_train_product, y_train_pred_proba),
        'F1 Score': f1_score(y_train_product, y_train_pred),
        'Confusion Matrix': confusion_matrix(y_train_product, y_train_pred)
    }

    metrics['train']['test'][product] = {
        'ROC AUC': roc_auc_score(y_test_product, y_test_pred_proba),
        'F1 Score': f1_score(y_test_product, y_test_pred),
        'Confusion Matrix': confusion_matrix(y_test_product, y_test_pred)
    }
```

In [17]:
```
# Summarize the average metrics across all products
summary_data = []
for dataset in ['train', 'test']:
    avg_roc_auc = np.mean([metrics['train'][dataset][p]['ROC AUC'] for p in products])
    avg_f1 = np.mean([metrics['train'][dataset][p]['F1 Score'] for p in products])
    summary_data.append(['train', dataset, avg_roc_auc, avg_f1])

# Create summary DataFrame
summary_df = pd.DataFrame(summary_data, columns=['Dataset', 'Type', 'Avg ROC AUC', 'Av
print("Summary Table:")
print(summary_df.to_string(index=False))

# Display which dataset was used
print("\nEvaluated Model on Dataset: train")
```

```
Summary Table:
Dataset  Type  Avg ROC AUC  Avg F1 Score
  train train     0.888038      0.110746
  train  test     0.883099      0.201405

Evaluated Model on Dataset: train
```

Database with first transformation

In [18]:

```python
# Initialize dictionary for storing metrics
metrics = defaultdict(lambda: defaultdict(dict))

# Train and evaluate the model on the 'train_1' dataset
for product in products:
    clf = LogisticRegression(**params)

    # Train data and labels for each product
    y_train_1_product = y_train_1[product].values
    y_test_product = y_test[product].values

    # Train the model
    clf.fit(X_train_1, y_train_1_product)

    # Predictions
    y_train_1_pred = clf.predict(X_train_1)
    y_test_pred = clf.predict(X_test)
    y_train_1_pred_proba = clf.predict_proba(X_train_1)[:, 1]
    y_test_pred_proba = clf.predict_proba(X_test)[:, 1]

    # Calculate metrics
    metrics['train_1']['train'][product] = {
        'ROC AUC': roc_auc_score(y_train_1_product, y_train_1_pred_proba),
        'F1 Score': f1_score(y_train_1_product, y_train_1_pred),
        'Confusion Matrix': confusion_matrix(y_train_1_product, y_train_1_pred)
    }

    metrics['train_1']['test'][product] = {
        'ROC AUC': roc_auc_score(y_test_product, y_test_pred_proba),
        'F1 Score': f1_score(y_test_product, y_test_pred),
        'Confusion Matrix': confusion_matrix(y_test_product, y_test_pred)
    }
```

In [19]:

```python
# Summarize the average metrics across all products
summary_data_1 = []
for dataset in ['train', 'test']:
    avg_roc_auc = np.mean([metrics['train_1'][dataset][p]['ROC AUC'] for p in products
    avg_f1 = np.mean([metrics['train_1'][dataset][p]['F1 Score'] for p in products])
    summary_data_1.append(['train_1', dataset, avg_roc_auc, avg_f1])

# Create summary DataFrame
summary_df_1 = pd.DataFrame(summary_data_1, columns=['Dataset', 'Type', 'Avg ROC AUC',
print("Evaluated Model on Dataset: train")
print(summary_df.to_string(index=False))

print("\nEvaluated Model on Dataset: train_1")
print(summary_df_1.to_string(index=False))
```

```
Evaluated Model on Dataset: train
Dataset  Type  Avg ROC AUC  Avg F1 Score
  train train     0.888038      0.110746
  train  test     0.883099      0.201405

Evaluated Model on Dataset: train_1
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_1 train     0.885656      0.111385
train_1  test     0.883373      0.207875
```

Database with second transformation

In [24]:
```
# Initialize dictionary for storing metrics
metrics = defaultdict(lambda: defaultdict(dict))

# Train and evaluate the model on the 'train_2' dataset
for product in products:
    clf = LogisticRegression(**params)

    # Train data and labels for each product
    y_train_2_product = y_train_2[product].values
    y_test_2_product = y_test_2[product].values

    # Train the model
    clf.fit(X_train_2, y_train_2_product)

    # Predictions
    y_train_2_pred = clf.predict(X_train_2)
    y_test_2_pred = clf.predict(X_test_2)
    y_train_2_pred_proba = clf.predict_proba(X_train_2)[:, 1]
    y_test_2_pred_proba = clf.predict_proba(X_test_2)[:, 1]

    # Calculate metrics
    metrics['train_2']['train'][product] = {
        'ROC AUC': roc_auc_score(y_train_2_product, y_train_2_pred_proba),
        'F1 Score': f1_score(y_train_2_product, y_train_2_pred),
        'Confusion Matrix': confusion_matrix(y_train_2_product, y_train_2_pred)
    }

    metrics['train_2']['test'][product] = {
        'ROC AUC': roc_auc_score(y_test_2_product, y_test_2_pred_proba),
        'F1 Score': f1_score(y_test_2_product, y_test_2_pred),
        'Confusion Matrix': confusion_matrix(y_test_2_product, y_test_2_pred)
    }
```

In [25]:
```
# Summarize the average metrics across all products
summary_data_2 = []
for dataset in ['train', 'test']:
    avg_roc_auc = np.mean([metrics['train_2'][dataset][p]['ROC AUC'] for p in products
    avg_f1 = np.mean([metrics['train_2'][dataset][p]['F1 Score'] for p in products])
    summary_data_2.append(['train_2', dataset, avg_roc_auc, avg_f1])

# Create summary DataFrame
summary_df_2 = pd.DataFrame(summary_data_2, columns=['Dataset', 'Type', 'Avg ROC AUC',
print("Evaluated Model on Dataset: train")
print(summary_df.to_string(index=False))

print("\nEvaluated Model on Dataset: train_1")
print(summary_df_1.to_string(index=False))
```

```
print("\nEvaluated Model on Dataset: train_2")
print(summary_df_2.to_string(index=False))
```

```
Evaluated Model on Dataset: train
Dataset  Type  Avg ROC AUC  Avg F1 Score
  train train     0.888038      0.110746
  train  test     0.883099      0.201405

Evaluated Model on Dataset: train_1
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_1 train     0.885656      0.111385
train_1  test     0.883373      0.207875

Evaluated Model on Dataset: train_2
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_2 train     0.885926      0.111536
train_2  test     0.883623      0.212467
```

Transformation #3

We will scale our features using MinMaxScaler from sklearn.preprocessing before training. This could improve convergence and the stability of the model.

We were between using MinMaxScaler or StandardScaler, however, since our data is not normal distributed, we will use MinMaxScaler

In [26]:

```
scaler = MinMaxScaler()
X_train_3 = scaler.fit_transform(X_train_2)
X_test_3 = scaler.transform(X_test_2)
```

In [27]:

```
# Initialize dictionary for storing metrics
metrics = defaultdict(lambda: defaultdict(dict))

# Train and evaluate the model on the 'train_2' dataset
for product in products:
    clf = LogisticRegression(**params)

    # Train data and labels for each product
    y_train_3_product = y_train_2[product].values
    y_test_3_product = y_test_2[product].values

    # Train the model
    clf.fit(X_train_3, y_train_3_product)

    # Predictions
    y_train_3_pred = clf.predict(X_train_3)
    y_test_3_pred = clf.predict(X_test_3)
    y_train_3_pred_proba = clf.predict_proba(X_train_3)[:, 1]
    y_test_3_pred_proba = clf.predict_proba(X_test_3)[:, 1]

    # Calculate metrics
    metrics['train_3']['train'][product] = {
        'ROC AUC': roc_auc_score(y_train_3_product, y_train_3_pred_proba),
        'F1 Score': f1_score(y_train_3_product, y_train_3_pred),
        'Confusion Matrix': confusion_matrix(y_train_3_product, y_train_3_pred)
```

```
    }

    metrics['train_3']['test'][product] = {
        'ROC AUC': roc_auc_score(y_test_3_product, y_test_3_pred_proba),
        'F1 Score': f1_score(y_test_3_product, y_test_3_pred),
        'Confusion Matrix': confusion_matrix(y_test_3_product, y_test_3_pred)
    }
```

In [28]:

```
# Summarize the average metrics across all products
summary_data_3 = []
for dataset in ['train', 'test']:
    avg_roc_auc = np.mean([metrics['train_3'][dataset][p]['ROC AUC'] for p in products
    avg_f1 = np.mean([metrics['train_3'][dataset][p]['F1 Score'] for p in products])
    summary_data_3.append(['train_3', dataset, avg_roc_auc, avg_f1])

# Create summary DataFrame
summary_df_3 = pd.DataFrame(summary_data_3, columns=['Dataset', 'Type', 'Avg ROC AUC',
print("Summary Table:")
print("Evaluated Model on Dataset: train")
print(summary_df.to_string(index=False))

print("\nEvaluated Model on Dataset: train_1")
print(summary_df_1.to_string(index=False))

print("\nEvaluated Model on Dataset: train_2")
print(summary_df_2.to_string(index=False))

print("\nEvaluated Model on Dataset: train_3")
print(summary_df_3.to_string(index=False))
```

```
Summary Table:
Evaluated Model on Dataset: train
Dataset  Type  Avg ROC AUC  Avg F1 Score
  train train     0.888038      0.110746
  train  test     0.883099      0.201405

Evaluated Model on Dataset: train_1
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_1 train     0.885656      0.111385
train_1  test     0.883373      0.207875

Evaluated Model on Dataset: train_2
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_2 train     0.885926      0.111536
train_2  test     0.883623      0.212467

Evaluated Model on Dataset: train_3
Dataset  Type  Avg ROC AUC  Avg F1 Score
train_3 train     0.885885      0.111541
train_3  test     0.883519      0.205022
```