Credit-Card Fraud Detection

A Recall-First ML Approach

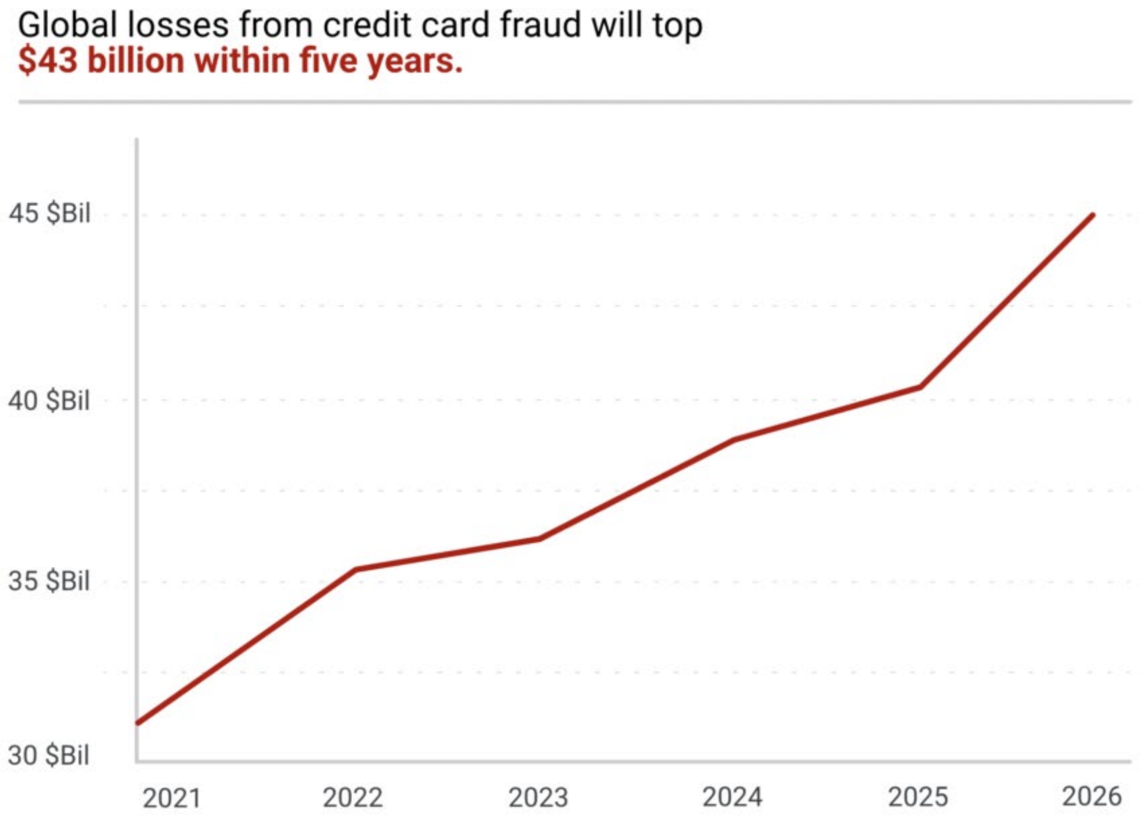
https://github.com/dizzydroid/fraud-detection

Why Fraud Detection?

Fraud in financial transactions is a critical issue with billions in losses annually.

Fraudulent transactions are **rare** but highly damaging.

Manual detection is inefficient—there is a need for **automated, accurate systems**.



Why Fraud Detection?

Fraud in financial transactions is a critical issue with billions in losses annually.
Fraudulent transactions are **rare** but highly damaging.
Manual detection is inefficient—there is a need for **automated, accurate systems**.

Goal

Build a model to **identify fraudulent credit card transactions** with **high recall**.

Why "recall"?

A high recall helps us minimize **False Negatives** ; $\dfrac{\overbrace{TP}^{\text{count of True Positives}}}{TP + \underbrace{FN}_{\text{count of False Negatives}}}$

In the context of Fraud Detection, this ensures we **do not miss** a *fraudulent transaction*.

The Research

Previous Studies

Past work shows traditional rule-based systems are rigid and generate many false alarms.

Machine learning approaches improve accuracy.

- *Dal Pozzolo et al. (2015)*: Focused on **recall/precision** in imbalanced data
- *Carcillo et al. (2018)*: Used ensemble models like **Random Forests**
- *Bahnsen et al. (2016)*: Introduced **cost-sensitive learning**
- **Deep learning** (e.g., autoencoders, LSTM) is emerging but less interpretable and data-hungry.

Most prior research focused on a **single dataset**, often optimizing for **accuracy or AUC**, while **recall**—a critical metric in fraud detection—**was underemphasized**.

Recent methods include:

- **Voting ensembles** and **hybrid models** (e.g., TabNet + XGBoost, autoencoders)
- Emphasis on performance, but limited **generalization** and **recall awareness**

This project addresses these gaps by designing a method that explicitly targets **high recall**, aiming to better capture fraudulent cases even in **highly imbalanced datasets**.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

PaySim simulates mobile money transactions based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country.

This synthetic dataset is scaled down 1/4 of the original dataset and it is created just for Kaggle.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.

type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.
type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount: amount of the transaction in local currency.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.
type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount: amount of the transaction in local currency.
nameOrig: customer who started the transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalance Org	newbalan ceOrig	nameDest	oldbalan ceDest	newbala nceDest	isFraud	isFlagge dFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.
type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount: amount of the transaction in local currency.
oldbalanceOrg: initial balance before the transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C....117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.
type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount: amount of the transaction in local currency.
oldbalanceOrg: initial balance before the transaction.
newbalanceOrig: customer's balance after the transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time.
type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount: amount of the transaction in local currency.
oldbalanceOrg: initial balance before the transaction.
newbalanceOrig: customer's balance after the transaction.
nameDest: recipient ID of the transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C....117	1089.0	28.69	M1...462	0.0	0.0	0	0
...	...	...	...	...	...	...	...	...	...	...

oldbalanceDest: initial recipient balance before the transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

oldbalanceDest: initial recipient balance before the transaction.

newbalanceDest: recipient's balance after the transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

oldbalanceDest: initial recipient balance before the transaction.

newbalanceDest: recipient's balance after the transaction.

isFraud: identifies a fraudulent transaction (1) and non fraudulent (0).

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalance Org	newbalan ceOrig	nameDest	oldbalan ceDest	newbala nceDest	isFraud	isFlagge dFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

oldbalanceDest: initial recipient balance before the transaction.

newbalanceDest: recipient's balance after the transaction.

isFraud: identifies a fraudulent transaction (1) and non fraudulent (0).

isFlaggedFraud: The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200,000 in a single transaction.

Proposed Strategy

The **PaySim** dataset

kaggle/datasets/ealaxi/paysim1

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	1060.31	C….117	1089.0	28.69	M1…462	0.0	0.0	0	0
…	…	…	…	…	…	…	…	…	…	…

Features

Target

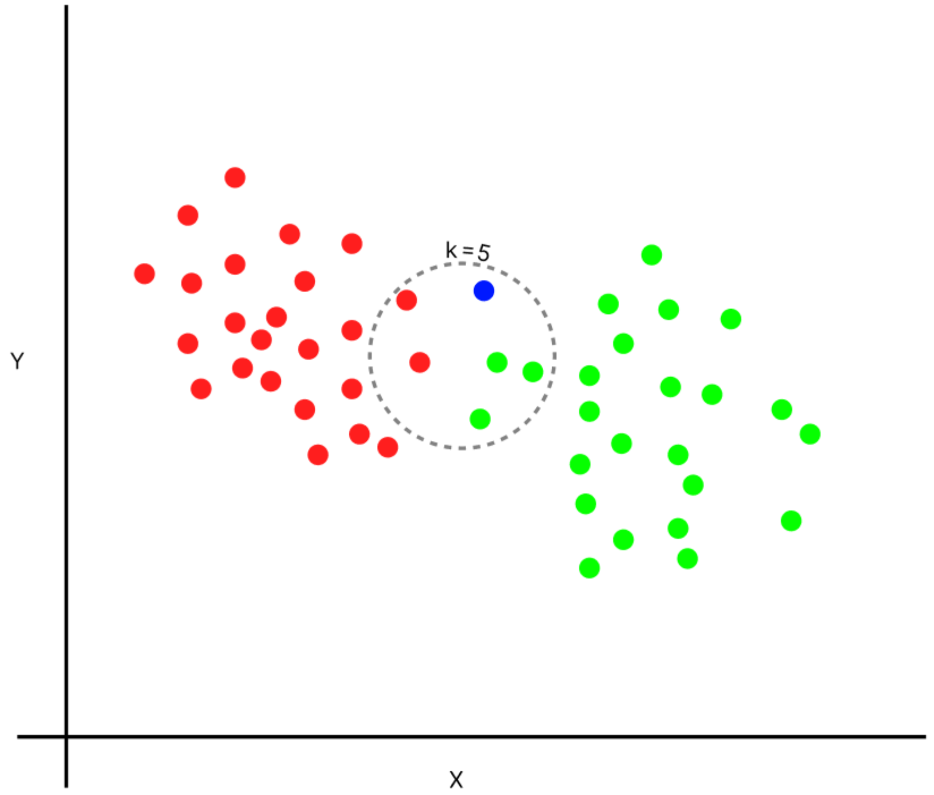
The categorical columns are: **type**, **nameOrig** and **nameDest**
Thus, we encode them into numerical values for a more efficient model training process.

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
0	1	3	9839.64	757869	170136.0	160296.36	1662094
1	1	3	1864.28	2188998	21249.0	19384.72	1733924

Proposed Strategy

Machine Learning Models Used

1. K Nearest Neighbors Classifier (KNN)



KNN classifiers are excellent non-parametric models that work really well on classifying points based on their **proximity** with other points in the dataset which usually may be exactly what you need in many applications, it serves as the quick alternative to other unsupervised methods as K-Means clustering.

They come at a cost, however.
Since KNN is non-parametric, it doesn't really "learn" the data the same way other models do, there is *no parameter tuning*, because there is *no parameters*!

Proposed Strategy

Machine Learning Models Used

1. K Nearest Neighbors Classifier (KNN)

The key difference among KNN classifiers is how exactly they learn from the data, this can differ from one to another depending on several factors, most notably:
Number of Neighbors and **Distance Calculation**

Our implementation follows the paper's setting of 5 neighbors using Euclidean distance

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Euclidean Distance formulation between two points "p", "q"

Proposed Strategy

Machine Learning Models Used

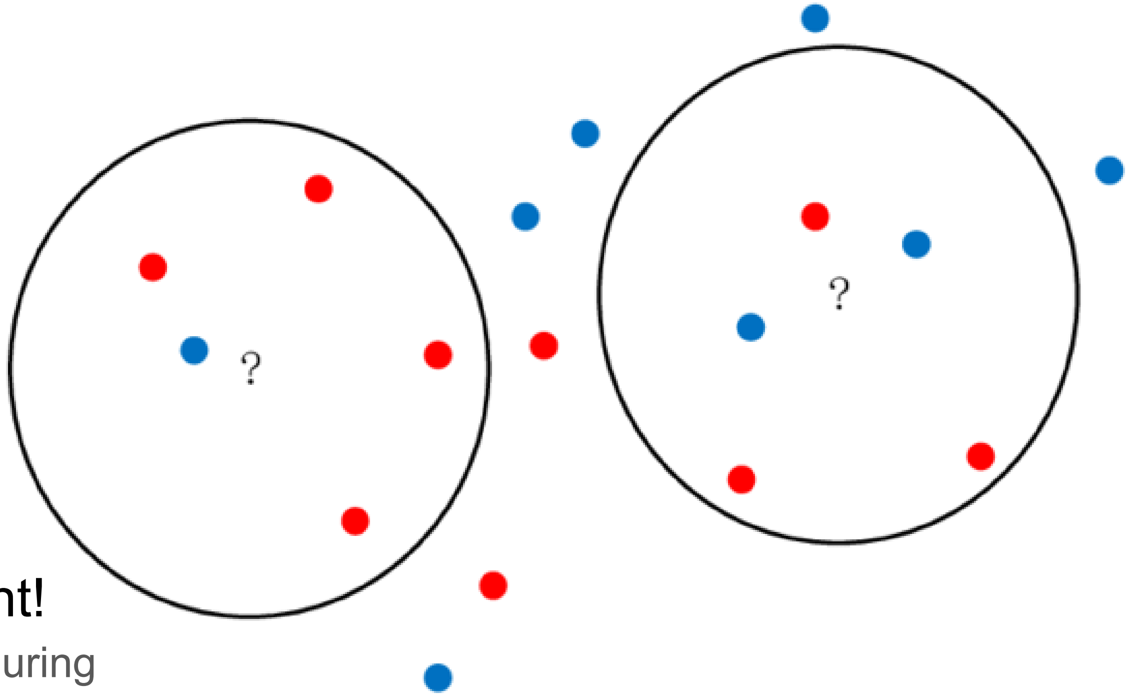
1. K Nearest Neighbors Classifier (KNN)

In our dataset, this effectively means that the point under study (denoted by “?” in the figure) will **store** the coordinates of essentially *every* point (6 million approx.)

This is practically how scikit-learn’s `model.fit()` works in case of `KNNClassifier`

So, the training process takes virtually no time since it’s simply storing coordinates.
(even though the resulting model (pkl) file would be hefty, capping at around 800MB in our training)

The **prediction** process, however, *will* be time consuming, since the model is **computing** proximity (distance) with every other unseen point!
(This process capped at 903 seconds (approx. 15 minutes) during our training on a t3.xlarge instance with 16 GiB of memory)



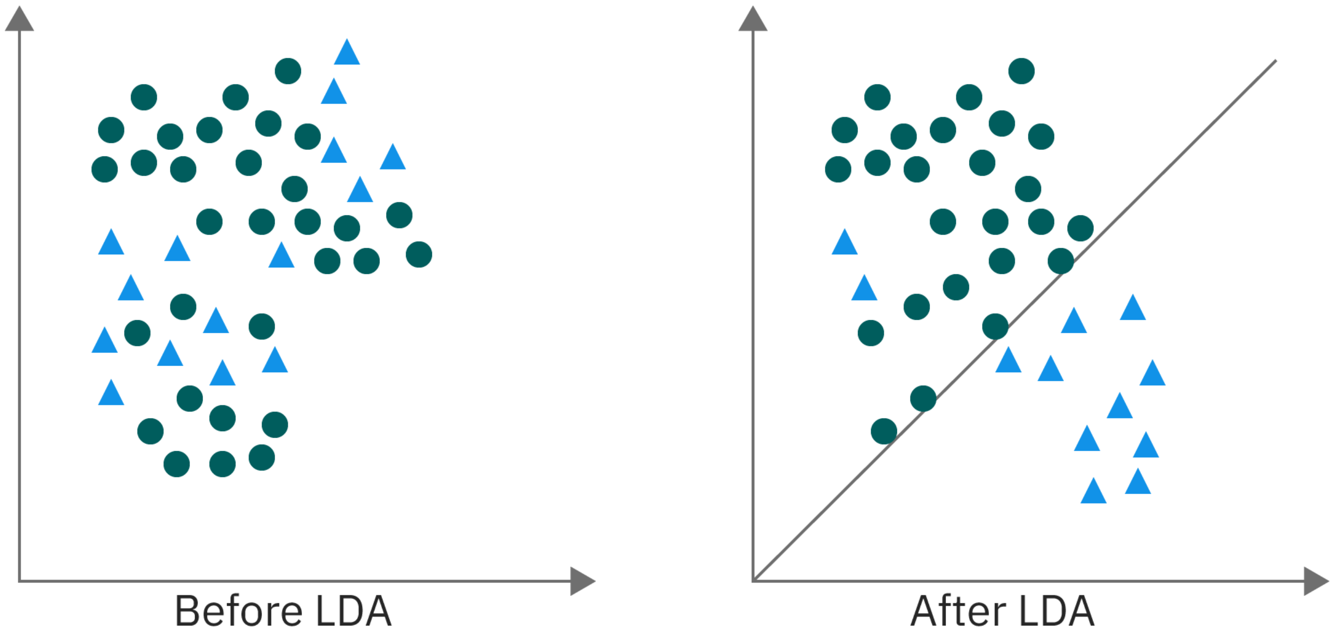
Proposed Strategy

Machine Learning Models Used

2. Linear Discriminant Analysis (LDA)

LDA is another classifier that works by identifying a **linear combination** of features that separates or characterizes two or more classes of objects or events.

LDA does this by projecting data with two or more dimensions into one dimension to be easily classified. The technique is, therefore, sometimes referred to as **dimensionality reduction**.



Proposed Strategy

Machine Learning Models Used

2. Linear Discriminant Analysis (LDA)

In contrast to KNN, LDA is a relatively fast classifier, since it’s parametric and utilizes optimization techniques for its learning process.

3. Linear Regression

A linear regression model is used as the final model, and a threshold during training. We dive into the exact methodology discussed in the following section.

Since Linear Regression is a **regression model**, we convert the model’s predictions into classifications by taking the **mean** of the model’s predictions and thresholding over it into a binary classification.

The Algorithm

Proposed Methodology

		pKNN	pLDA	pLR
0	…	0	0	0.1
1	…	0	1	0.3
2	…	1	0	0.8
3	…	1	1	0.6
4	…	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	...	0	0	0.1
1	...	0	1	0.3
2	...	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

$\underbrace{\hspace{3em}}_{mvLR = 0.4}$

```
FOR i FROM 0 to len(zeros array as dataset) DO

  IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
     IF (pLR[i] < mvLR) THEN
         pOR[i] ← 0
     END IF

   ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
     IF (pLR[i] > mvLR) THEN
         pOR[i] ← 1
       END IF

   ELSE
     pOR [i] ← pKNN[i]
   END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	···	0	0	0.1
1	···	0	1	0.3
2	···	1	0	0.8
3	···	1	1	0.6
4	···	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
      IF (pLR[i] < mvLR) THEN
          pOR[i] ← 0
      END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
      IF (pLR[i] > mvLR) THEN
          pOR[i] ← 1
        END IF

    ELSE
      pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	…	0	1	0.3
2	…	1	0	0.8
3	…	1	1	0.6
4	…	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	···	0	1	0.3
2	···	1	0	0.8
3	···	1	1	0.6
4	···	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

   IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
      IF (pLR[i] < mvLR) THEN
         pOR[i] ← 0
      END IF

   ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
      IF (pLR[i] > mvLR) THEN
         pOR[i] ← 1
      END IF

   ELSE
      pOR [i] ← pKNN[i]
   END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	...	0	1	0.3
2	...	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
      IF (pLR[i] < mvLR) THEN
          pOR[i] ← 0
      END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
      IF (pLR[i] > mvLR) THEN
          pOR[i] ← 1
        END IF

    ELSE
      pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
● 1	0	0	1	0.3
2	...	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
          ●  pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	...	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

  IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
     IF (pLR[i] < mvLR) THEN
         pOR[i] ← 0
     END IF

   ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
     IF (pLR[i] > mvLR) THEN
         pOR[i] ← 1
      END IF

   ELSE
     pOR [i] ← pKNN[i]
   END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	…	1	0	0.8
3	…	1	1	0.6
4	…	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
      IF (pLR[i] < mvLR) THEN
          pOR[i] ← 0
      END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
      IF (pLR[i] > mvLR) THEN
          pOR[i] ← 1
        END IF

    ELSE
      pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	...	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
       IF (pLR[i] < mvLR) THEN
           pOR[i] ← 0
       END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
       IF (pLR[i] > mvLR) THEN
           pOR[i] ← 1
         END IF

    ELSE
       pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

$mvLR = 0.4$

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	...	1	1	0.6
4	...	1	1	0.2

$\underbrace{\qquad}_{mvLR = 0.4}$

```
FOR i FROM 0 to len(zeros array as dataset) DO

  IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
      IF (pLR[i] < mvLR) THEN
          pOR[i] ← 0
      END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
      IF (pLR[i] > mvLR) THEN
          pOR[i] ← 1
        END IF

    ELSE
      pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	…	1	1	0.6
4	…	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
       IF (pLR[i] < mvLR) THEN
           pOR[i] ← 0
       END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
       IF (pLR[i] > mvLR) THEN
           pOR[i] ← 1
         END IF

    ELSE
       pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	…	1	1	0.6
4	…	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	1	1	1	0.6
4	…	1	1	0.2

$\underbrace{\hphantom{0.1}}_{mvLR = 0.4}$

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	1	1	1	0.6
4	...	1	1	0.2

$mvLR = 0.4$

```
FOR i FROM 0 to len(zeros array as dataset) DO

  IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
     IF (pLR[i] < mvLR) THEN
        pOR[i] ← 0
     END IF

  ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
     IF (pLR[i] > mvLR) THEN
        pOR[i] ← 1
     END IF

  ELSE
     pOR [i] ← pKNN[i]
  END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	1	1	1	0.6
4	...	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	1	1	1	0.6
4	...	1	1	0.2

$mvLR = 0.4$

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	1	1	1	0.6
4	...	1	1	0.2

$\text{mvLR} = 0.4$

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Proposed Methodology

		pKNN	pLDA	pLR
0	0	0	0	0.1
1	0	0	1	0.3
2	0	1	0	0.8
3	1	1	1	0.6
4	0	1	1	0.2

mvLR = 0.4

```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Our Implementation

Project Structure

We followed a modular approach in our implementation to ensure maintainability and eliminate any possible single point of failure in the process.

```
fraud-detection/
├── data/
│   ├── raw/
│   └── processed/
├── src/
│   ├── data_ingest.py
│   ├── preprocessing.py
│   ├── train_models.py
│   ├── models/
│   │   ├── knn.py
│   │   ├── lda.py
│   │   └── linreg.py
│   ├── ensemble.py          <-- implements Algorithm 1
│   └── evaluate.py          <-- writes results/metrics.json & results/figures
├── artifacts/               # *.pkl, *.npy files after `make train`
├── results/
│   ├── metrics.json
│   └── figures/
└── README.md
```

```bash
$ bash
make setup          # install dependencies
make preprocess     # preprocess data (encoding, split)
make train          # fit key models and save them
make ensemble       # apply ensemble voting (Algorithm)
make evaluate       # compute metrics, visualize and save results
```

https://github.com/dizzydroid/fraud-detection

Pipeline Visualization

Download raw data (data_ingest.py)

1. GET DATA FROM kaggle
2. STORE IN `data/raw` DIRECTORY

Preprocess data (preprocessing.py)

1. LOAD RAW DATA FROM `data/raw` DIRECTORY
2. APPLY PREPROCESSING AND EXPORT TO `data/preprocessed` DIRECTORY

Model implementation (src/models)

1. IMPLEMENT `fit()`, `predict()` FOR: KNN, LDA AND LINREG.

Model Training (train_models.py)

1. FOR EACH MODEL IN `src/models`, TRAIN ON PREPROCESSED DATA.
2. EXPORT EACH TRAINED MODEL AS PKL
3. EXPORT GROUND TRUTH AS `y_true.npy`

Ensemble Algorithm (ensemble.py)

1. LOAD MODELS, APPLY ENSEMBLE ALGORITHM.
2. EXPORT ENSEMBLE PREDICTIONS AS `y_pred.npy`

Evaluation (evaluate.py)

1. LOAD PREDICTIONS & GROUND TRUTH
2. COMPUTE METRICS AND EXPORT

The Demo Notebook

On our project repository, you can run the code in one of **three** ways:

1) **The Right Way:** using the make commands mentioned earlier, this, however, can be really slow and is highly dependent on hardware
2) **The Slow Way:** running the fraud_detection notebook. This is by far the slowest, since it – not only applies KNN predictions **twice** – but it also trains additional models that are known for being slow like RandomForests.
3) **The Quick Way:** Running the fraud_demo notebook. This is the efficient way when you want to check that the algorithm **just works**. This takes basically no time because we have cached the KNN predictions and pre-loaded them into the notebook to save time and resources.

For a test-run, we recommend you go with **the quick way**, to simply see *in code* the process yourself, play around with the data and check the resulting metrics and visualizations.

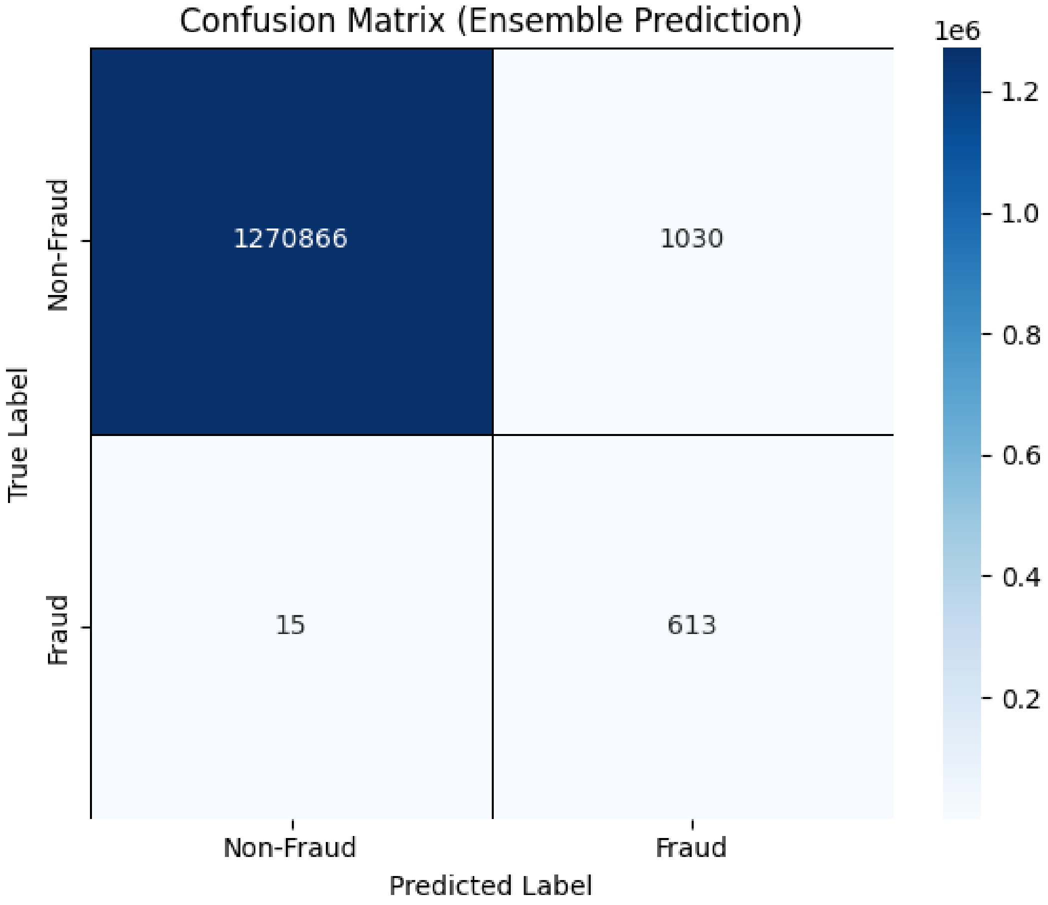
https://github.com/dizzydroid/fraud-detection

Results

A flawless algorithm or wishful thinking?

The Caveat

Achieving approximately 1.0 recall does seem phenomenal, but we have reason to believe it’s **not** the perfect algorithm it claims to be.



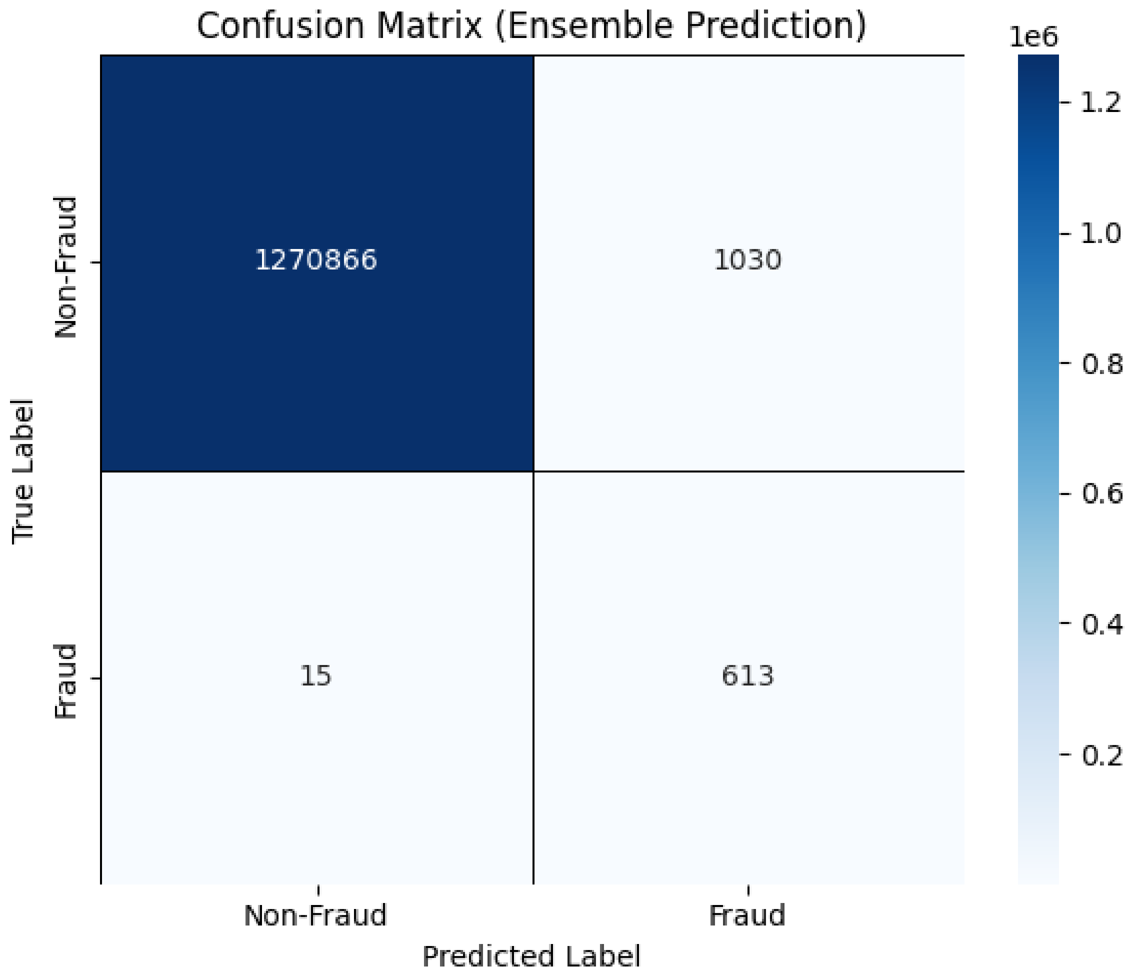
The Caveat

The Up

False Negatives (Frauds detected as Non-Frauds) are basically non-existent which is a good thing especially in real life applicability since we never want any frauds passing as non-frauds.

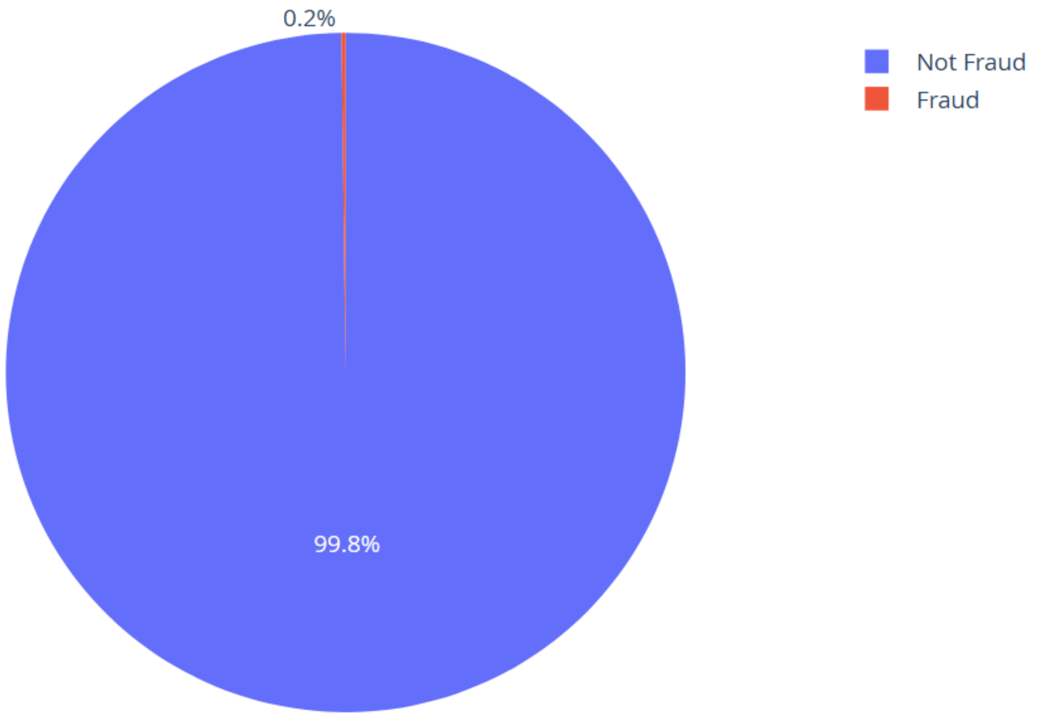
The Down

False Positives are **dominating**, in our Fraudulent example, there's a considerable amount of misclassifying Non-Frauds as Fraud transactions.



Why did it work?

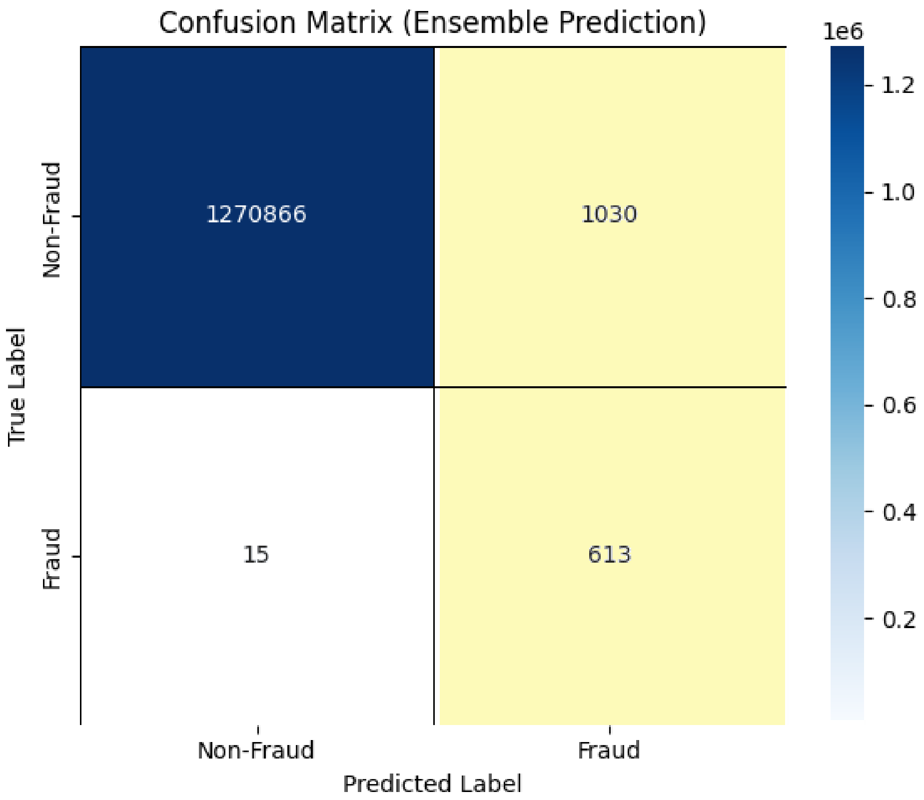
The data imbalance worked in favor of the approach. Since only 0.13% of the ~ 6 Mil dataset were fraudulent. And the approach favors the non-fraud predictions.




```
FOR i FROM 0 to len(zeros array as dataset) DO

    IF (pKNN[i] is 0 OR pLDA[i] is 0) THEN
        IF (pLR[i] < mvLR) THEN
            pOR[i] ← 0
        END IF

    ELSE IF (pKNN[i] is 1 OR pLDA[i] is 1) THEN
        IF (pLR[i] > mvLR) THEN
            pOR[i] ← 1
        END IF

    ELSE
        pOR [i] ← pKNN[i]
    END IF
END FOR
```

Why did it work?

The data imbalance worked in favor of the approach. Since only 0.13% of the ~ 6 Mil dataset were fraudulent. And the approach favors the non-fraud predictions.



1030 examples that were actually non-fraudulent were falsely detected as fraudulent. (*a False Positive*)

While this may not be alarming in the case of a large dataset as this one. In smaller sets, this could highly affect precision which is mainly the reason why the paper's implementation achieved a subpar precision overall:

Index	Dataset #	Top #	Model	Recall	Accuracy	Precision
1	1	1	Our Method	1.0	0.9989	0.0656
2	1	2	DT	0.7910	0.9996	0.8036
3	1	3	RF	0.7855	0.9998	0.9853
4	1	4	ET	0.6400	0.9996	0.9982

Conclusions

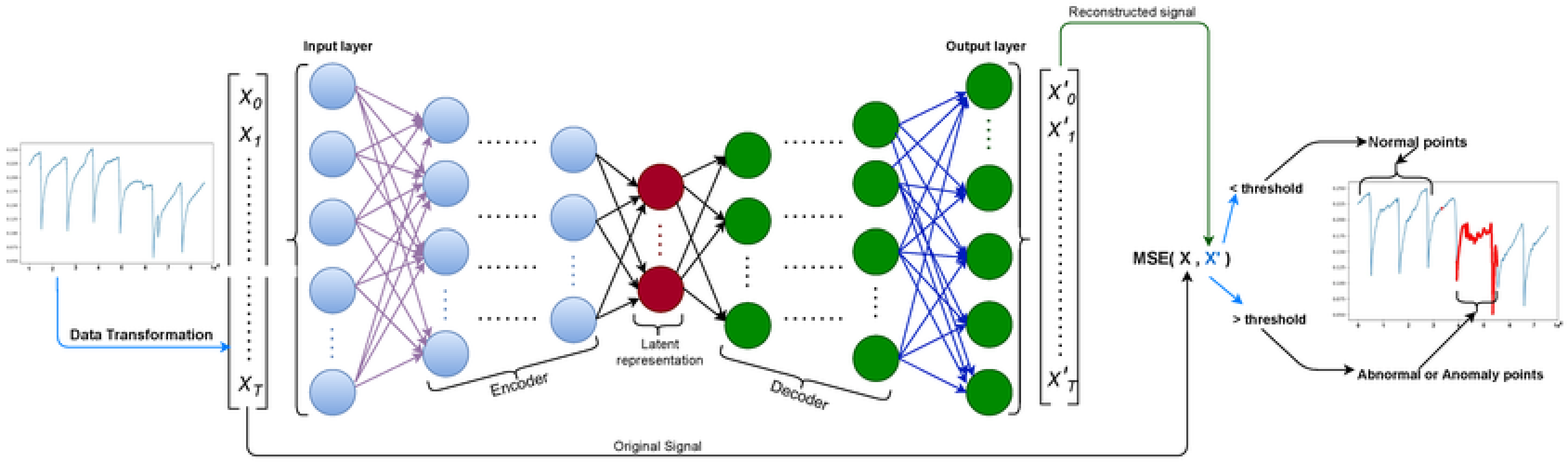
Could we do better?

How can we do better?

While the proposed method is great with **recall**, it performs very poorly with **precision**. A 6% precision is definitely not applicable in real-life solutions.

We believe the following methods can be more efficient in detecting fraudulent transactions while maintaining good recall *and* precision.

1. **Deep Learning + Unsupervised approaches**
   Fraudulent transactions are not common; thus, they can be treated as *anomalies*, the use of **autoencoders** for **anomaly detection** could be rewarding.



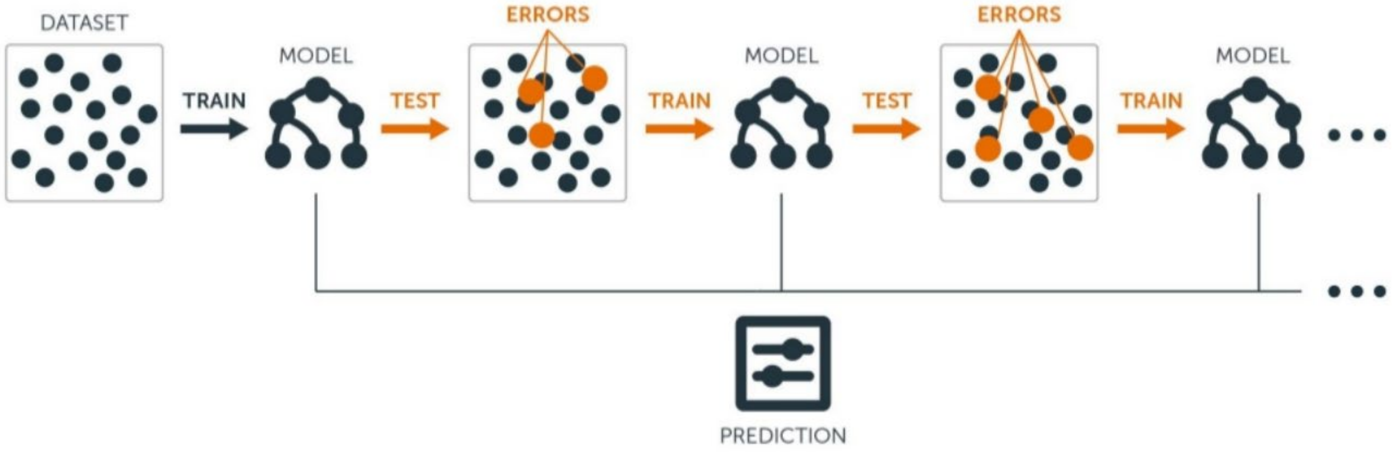
How can we do better?

While the proposed method is great with **recall**, it performs very poorly with **precision**. A 6% precision is definitely not applicable in real-life solutions.

We believe the following methods can be more efficient in detecting fraudulent transactions while maintaining good recall *and* precision.

2. Gradient Boosting

Gradient Boosting is known to handle imbalance really well, in addition to providing high accuracy by nature.



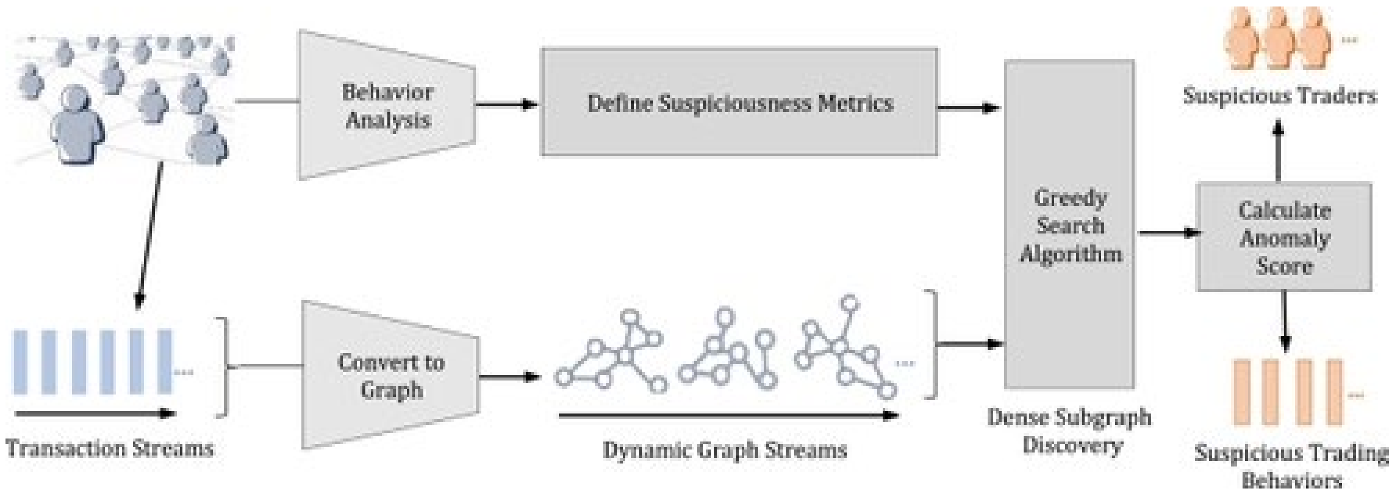
How can we do better?

While the proposed method is great with **recall**, it performs very poorly with **precision**. A 6% precision is definitely not applicable in real-life solutions.

We believe the following methods can be more efficient in detecting fraudulent transactions while maintaining good recall *and* precision.

3. Hybrid Approaches

Combining multiple approaches usually gives good results, **graph-based** methods are a good candidate in the case of fraudulent detections.



How can we do better?

While the proposed method is great with **recall**, it performs very poorly with **precision**. A 6% precision is definitely not applicable in real-life solutions.

We believe the following methods can be more efficient in detecting fraudulent transactions while maintaining good recall *and* precision.

The bottom line

While Chung & Lee's algorithm does a great job at **recall-first** predictions, it struggles with precision, which is likely the major holdback in commercial use.

Granted, more efficient algorithms will surface in the coming years. However, we need to keep it practical. If you need near perfect recall, and won't mind falsely flagging non-frauds, this algorithm will do the job. If you need a good balance, neural networks will save the day. If you need perfect recall *and* precision, you may have to wait a few years.

References

Bahnsen, A. C., Aouada, D., Stojanovic, A., & Ottersten, B. (2016). Feature engineering strategies for credit card fraud detection. Expert Systems with Applications, 51, 134–142. https://doi.org/10.1016/j.eswa.2015.12.030

Carcillo, F., Dal Pozzolo, A., Le Borgne, Y.-A., Caelen, O., Mazzer, Y., & Bontempi, G. (2018). Scarff: A scalable framework for streaming credit card fraud detection with Spark. Information Fusion, 41, 182–194. https://arxiv.org/abs/1709.08920

Chung, J., & Lee, K. (2023). Credit card fraud detection: An improved strategy for high recall using KNN, LDA, and linear regression. Sensors, 23(18), 7788. https://doi.org/10.3390/s23187788

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2015). Credit card fraud detection: A realistic modeling and a novel learning strategy. IEEE Transactions on Neural Networks and Learning Systems, 29(8), 3784–3797. https://doi.org/10.1109/TNNLS.2017.2736643

IBM. (n.d.). What is Linear Discriminant Analysis (LDA)? IBM Think. https://www.ibm.com/think/topics/linear-discriminant-analysis

Kaggle. (n.d.). Synthetic Financial Dataset For Fraud Detection (PaySim). https://www.kaggle.com/datasets/ealaxi/paysim1

Lopez-Rojas, E., & Axelsson, S. (2014). Paysim: A financial mobile money simulator for fraud detection. In Proceedings of the 28th European Conference on Modelling and Simulation. https://doi.org/10.7148/2014-0259

ResearchGate. (n.d.). An example of kNN classification task with k=5 [Figure]. https://www.researchgate.net/figure/An-example-of-kNN-classification-task-with-k-5_fig1_293487460

Scikit-learn developers. (n.d.). scikit-learn: Machine Learning in Python. https://scikit-learn.org/

Fraud detection – A Recall-first ML approach [GitHub repository]. https://github.com/dizzydroid/fraud-detection

Thank you!