Розпізнавання шахрайства із використанням гіпотези компактності

Постановка задачі:

Використовуючи алгоритм гіпотези компактності здійснити класифікацію записів на 'шахрайство' та 'не шахрайство'.

Посилання на датасет: https://www.kaggle.com/competitions/ieee-fraud-detection/overview

1.1) Очистка параметрів та реалізацій, непридатних для аналізу;
1.2) Кодування категоріальних змінних;
1.3) Формування рівномірної тренувальної та тестової вибірок;
1.4) Оцінка інформативності входів улюбленими методами;
1.5) Тренування одношарового лінійного перцептрона на тренувальній вибірці;
1.6) Оцінка ваги входів за допомогою ідентифікаційної матриці, ітеративно позбуватися найменш інформативних входів;
1.7) На тестовій вибірці переконатися у тому, що вдалося побудувати ілеаотну просту розділяючи поверхню у просторі із мінімальною вимірністю.

Критерії валідації:

Точність моделі на тестових даних 100%.

Етап 1: Data Cleaning

Датасет, який використовуватиметься для тренування і тестування моделі, складається з двох файлів: 'train_transaction.csv' та 'train_identity.csv' (рис. 1-2).

```
train_transaction_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 590540 entries, 0 to 590539
Columns: 394 entries, TransactionID to V339
dtypes: float64(376), int64(4), object(14)
memory usage: 1.7+ GB
```

Рис. 1. Інформація про датасет train_transaction

```
train_identity_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144233 entries, 0 to 144232
Data columns (total 41 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   TransactionID  144233 non-null  int64
 1   id_01          144233 non-null  float64
 2   id_02          140872 non-null  float64
 3   id_03          66324 non-null   float64
 4   id_04          66324 non-null   float64
 5   id_05          136865 non-null  float64
 6   id_06          136865 non-null  float64
 7   id_07          5155 non-null    float64
 8   id_08          5155 non-null    float64
 9   id_09          74926 non-null   float64
 10  id_10          74926 non-null   float64
 11  id_11          140978 non-null  float64
 12  id_12          144233 non-null  object
 13  id_13          127320 non-null  float64
 14  id_14          80044 non-null   float64
 15  id_15          140985 non-null  object
 16  id_16          129340 non-null  object
 17  id_17          139369 non-null  float64
 18  id_18          45113 non-null   float64
 19  id_19          139318 non-null  float64
 20  id_20          139261 non-null  float64
 21  id_21          5159 non-null    float64
 22  id_22          5169 non-null    float64
 23  id_23          5169 non-null    object
 24  id_24          4747 non-null    float64
 25  id_25          5132 non-null    float64
 26  id_26          5163 non-null    float64
 27  id_27          5169 non-null    object
 28  id_28          140978 non-null  object
 29  id_29          140978 non-null  object
 30  id_30          77565 non-null   object
 31  id_31          140282 non-null  object
 32  id_32          77586 non-null   float64
 33  id_33          73289 non-null   object
 34  id_34          77805 non-null   object
 35  id_35          140985 non-null  object
 36  id_36          140985 non-null  object
 37  id_37          140985 non-null  object
 38  id_38          140985 non-null  object
 39  DeviceType     140810 non-null  object
 40  DeviceInfo     118666 non-null  object
dtypes: float64(23), int64(1), object(17)
memory usage: 45.1+ MB
```

Рис. 2. Інформація про датасет train_identity

Датасет train_transaction містить 590540 записів та 394 ознак (рис. 1), а train_identity – 144233 записи і 41 ознаку (рис. 2). Варто відзначити, що train_identity містить значно менше записів, ніж train_transaction.

Було здійснено об’єднання (left merging) датасетів train_transaction і train_identity (рис. 3), внаслідок чого отримано датасет train_df із 590539 записів та 434 ознаками.

```
train_df = pd.merge(train_transaction_df, train_identity_df, on='TransactionID', how='left')
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 0 to 590539
Columns: 434 entries, TransactionID to DeviceInfo
dtypes: float64(399), int64(4), object(31)
memory usage: 1.9+ GB
```

Рис. 3. Результат об’єднання датасетів train_transaction і train_identity

Наступним кроком було переглянуто розподіл класів цільової мітки (рис. 4).

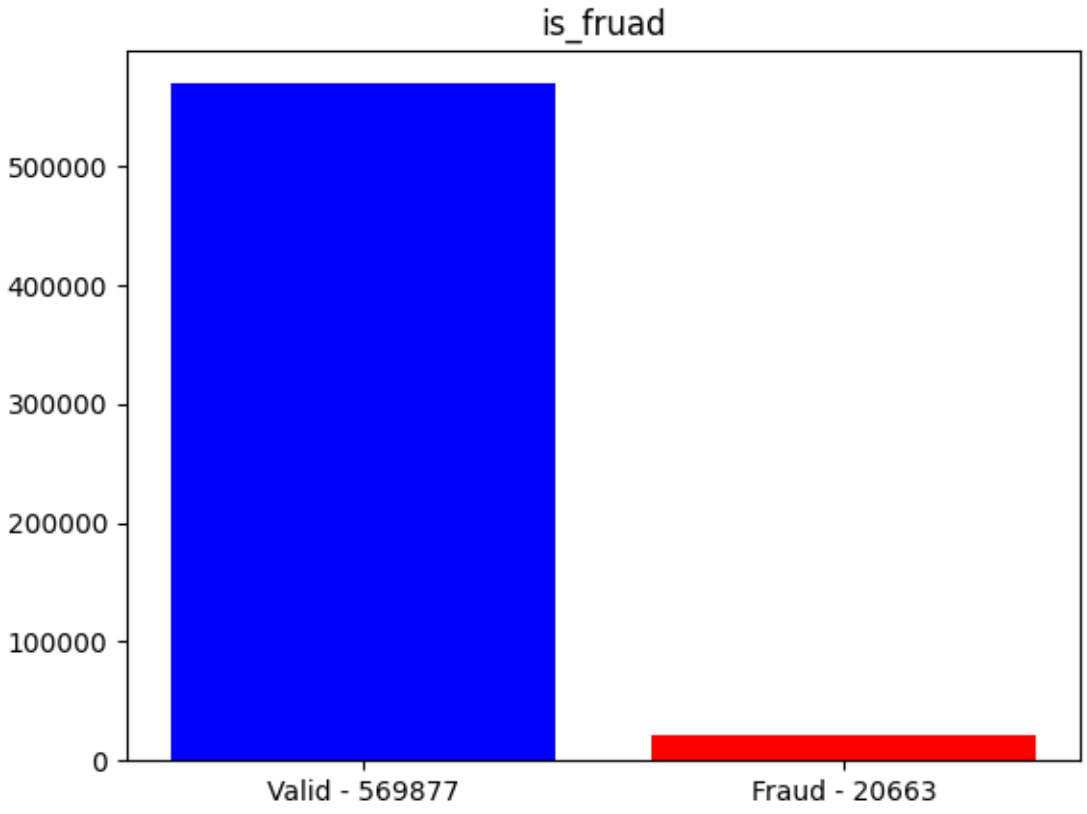

Рис. 4. Розподіл класів цільової мітки 'isFraud'

Із рис. 4 видно, що досліджуваний датасет є незбалансованим.

Далі було здійснено процес Data Cleaning, але таким чином, щоб кількість записів, що відповідають шахрайству (isFraud=True) не зменшилася.

В першу чергу вилучено стовпці із великою кількістю відсутніх значень (missing values). Для цього побудовано окремий датафрейм missing_info, який містить назву стовпця і кількість пропущених значень (рис. 5).

	column_name	missing_count
0	id_24	585793
1	id_25	585408
2	id_07	585385
3	id_08	585385
4	id_21	585381

Рис. 5. Датафрейм, який містить інформацію про кількість відсутніх значень стовпців досліджуваного датасету

Було проглянуто отриманий датафрейм missing_info та знайдено різке падіння кількості втрачених значень від 65706 до 8933 (рис. 6).

	column_name	missing_count
315	V17	76073
316	V31	76073
317	V15	76073
318	V20	76073
319	D10	76022
320	addr1	65706
321	addr2	65706
322	card2	8933
323	card5	4259
324	card4	1577

Рис. 6. Інформація про кількість втрачених значень стовпців досліджуваного датасету

Як видно із рис. 6, для стовпця 'card' кількість втрачених значень дорівнює 8933, що становить приблизно 1.5% від загальної кількості даних. А для стовпця 'addr2' кількість втрачених значень – 65706, що становить понад 10% (приблизно 11%) від загальної кількості даних. Тому всі стовпці, кількість

втрачених значень для яких перевищує 10% (у нашому випадку понад 9000 записів), було вилучено із оригінального датасету. В результаті отримано датасет із 112 стовпцями (рис. 7).

Training dataset form where features missing more than 9000 records have been removed: (590540, 112)

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	...	V312	V313	V314	V315	V316	V317	V318
0	2987000	0	86400	68.5	W	13926	NaN	150.0	discover	142.0	...	0.0	0.0	0.0	0.0	0.0	117.0	0.0
1	2987001	0	86401	29.0	W	2755	404.0	150.0	mastercard	102.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2987002	0	86469	59.0	W	4663	490.0	150.0	visa	166.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	2987003	0	86499	50.0	W	18132	567.0	150.0	mastercard	117.0	...	135.0	0.0	0.0	0.0	50.0	1404.0	790.0
4	2987004	0	86506	50.0	H	4497	514.0	150.0	mastercard	102.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 112 columns

Рис. 7. Досліджуваний датасет із стовпцями, кількість втрачених значень яких не перевищує 10%

Наступним кроком здійснено вилучення записів, які мають відсутнє значення в одному із стовпців (NaN) та не представляють шахрайство (щоб не зменшувати кількість записів із isFraud=True, які становлять меншість).

Для оновленого датафрейму знову створено missing_info, який містить інформацію про кількість втрачених записів для кожного стовпця (рис. 8) і визначено кількість стовпців із втраченими значеннями (рис. 9).

	0	1	2	3	4	5	6	7	8	9	...	102	103	104	105	106	107	108	109	110	111
column_name	card2	card5	card4	card6	card3	V281	V315	D1	V301	V300	...	C6	C7	C8	C9	C11	C12	C13	C14	isFraud	TransactionID
missing_count	8510	4049	1536	1532	1526	1223	1223	1223	1223	1223	...	0	0	0	0	0	0	0	0	0	0

2 rows × 112 columns

Рис. 8. Датафрейм, який містить інформацію про кількість відсутніх значень стовпців оновленого датасету

```
columns_with_NaN_values = missing_info[missing_info['missing_count'] > 0]['column_name'].values
len(columns_with_NaN_values)
```

```
92
```

Рис. 9. Кількість стовпців із відсутніми записами в оновленому датасеті

Із рис. 9 видно, що оновлений датафрейм містить 92 стовпці із відсутніми записами (NaN).

Внаслідок вилучення записів, які мають відсутнє значення в одному із стовпців (NaN) та не представляють шахрайство, кількість записів, які не представляють шахрайство, зменшилася від 569877 (рис. 4) до 558065 (рис. 10).

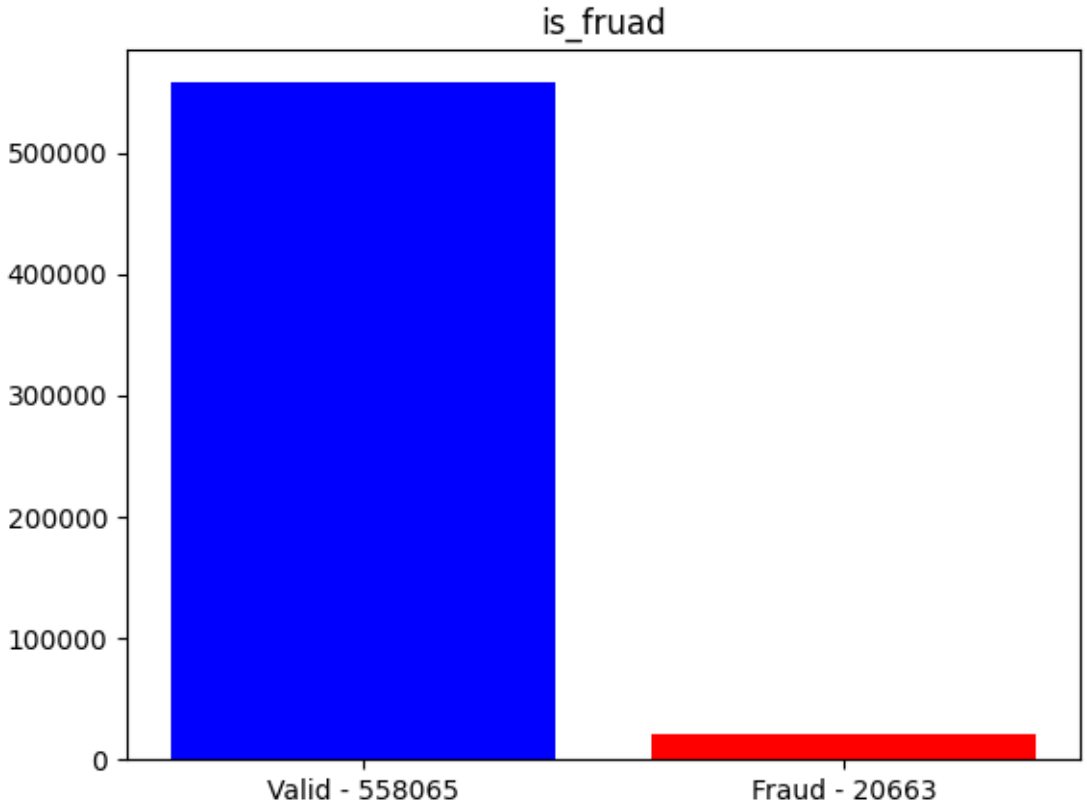

Рис. 10. Розподіл класів цільової мітки 'isFraud' після вилучення записів із відсутніми значеннями

Однак записи, які представляють шахрайство ('isFraud' = True), теж можуть містити відсутні значення (NaN) для певних стовпців. Для перевірки наявності таких записів також було побудовано датафрейм missing_info (рис. 11).

	0	1	2	3	4	5	6	7	8	9	...	102	103	104	105	106	107	108	109	110	111
column_name	card2	card5	V281	D1	V301	V300	V313	V314	V315	V282	...	C6	C7	C8	C9	C11	C12	C13	C14	isFraud	TransactionID
missing_count	423	210	46	46	46	46	46	46	46	46	...	0	0	0	0	0	0	0	0	0	0

2 rows × 112 columns

Рис. 9. Датафрейм, який містить інформацію про кількість відсутніх значень стовпців оновленого датасету

Для подальшого проведення кодування категоріальних змінних і тренування одношарового лінійного перцептрона відсутні значення були замінені на модальні значення відповідного стовпця.

Етап 2: Coding of categorical variables

Для кодування категоріальних змінних було переглянуто інформацію про датасет і його категоріальні змінні (рис. 10).

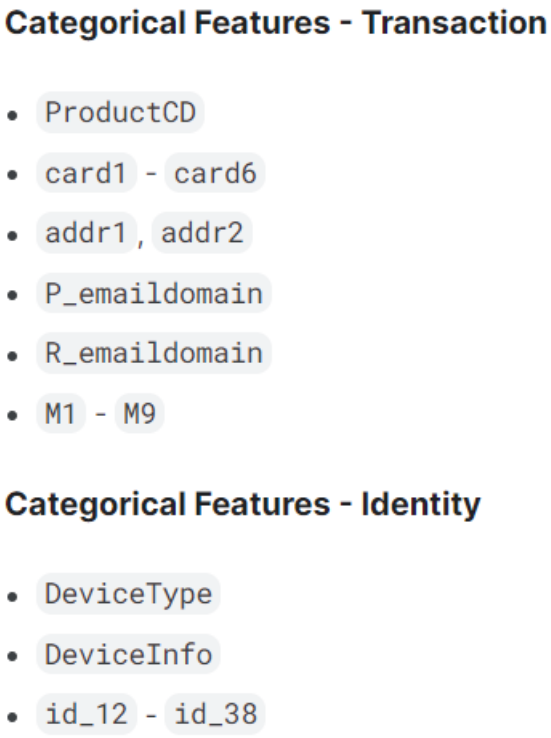

Рис. 10. Категоріальні змінні досліджуваного датасету

Оскільки на етапі Data Cleaning із оригінального датасету було вилучено деякі стовпці, здійснено перевірку наявності категоріальних стовпців в очищеному датасеті (рис. 11).

```
ProductCD is present in the train_df
card1 is present in the train_df
card2 is present in the train_df
card3 is present in the train_df
card4 is present in the train_df
card5 is present in the train_df
card6 is present in the train_df
The number of categorical columns in the training dataset: 7
```

Рис. 11. Категоріальні стовпці в очищеному датасету

Варто зазначити, що із 7 категоріальних стовпців в очищеному датасеті лише три стовці містять назви категорій: 'ProductCD', 'card4', 'card6', тому їх було замінено числовими мітками (рис. 12 - 14). Наступні 4 стовпці, які визначені як категоріальні в описі датасету (рис. 10), вже містять числові значення (рис. 15), тому їхній вміст не було змінено.

```
train_df['ProductCD'].unique()
```

```
array(['W', 'C', 'R', 'S', 'H'], dtype=object)
```

```
train_df['ProductCD'].unique()
```

```
array([0, 1, 2, 3, 4], dtype=int64)
```

Рис. 12. Кодування категоріальних міток стовпця 'ProductCD'

```
train_df['card4'].unique()
```

```
array(['visa', 'mastercard', 'discover', 'american express'], dtype=object)
```

```
train_df['card4'].unique()
```

```
array([0, 1, 2, 3], dtype=int64)
```

Рис. 13. Кодування категоріальних міток стовпця 'card4'

```
train_df['card6'].unique()
```

```
array(['credit', 'debit', 'debit or credit', 'charge card'], dtype=object)
```

```
train_df['card6'].unique()
```

```
array([0, 1, 2, 3], dtype=int64)
```

Рис. 14. Кодування категоріальних міток стовпця 'card6'

```
train_df['card1'].unique()
```

```
array([18268, 13413, 16578, ..., 17972, 13166,  8767], dtype=int64)
```

```
type(train_df['card2'].iloc[0])
```

```
numpy.float64
```

```
type(train_df['card3'].iloc[0])
```

```
numpy.float64
```

```
type(train_df['card5'].iloc[0])
```

```
numpy.float64
```

Рис. 15. Інформація про стовпці 'card1', 'card2', 'card3' та 'card5'

Етап 3: Formation of uniform training and test samples

Як було згадано раніше, досліджуваний датасет є незбалансованим (рис. 10). Для балансування виконано undersampling (рис. 16), внаслідок чого отримано збалансований датасет розміром 41326 записів.

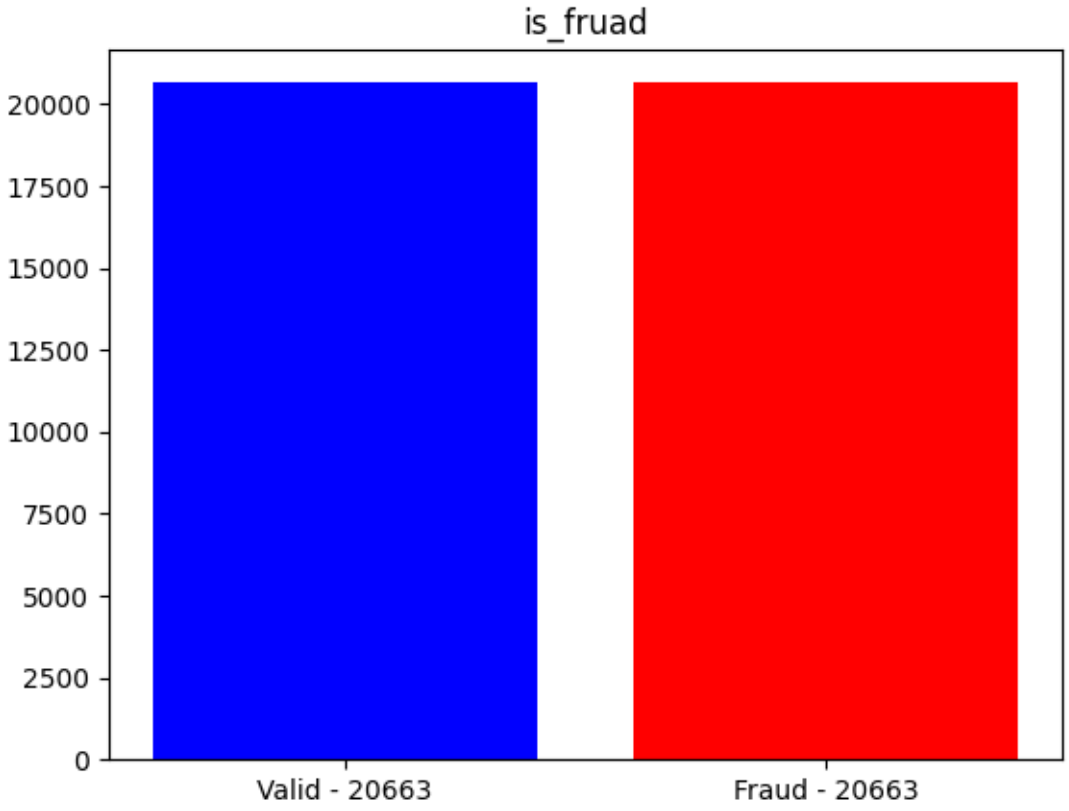

Рис. 16. Розподіл класів цільової мітки 'isFraud' після балансування

Збалансований датафрейм було поділено на тренувальний (train_df) та тестовий (test_df) у співвідношенні 80/20 (33060/8266 записів відповідно) так, щоб вони теж були збалансованими (рис. 17, 18).

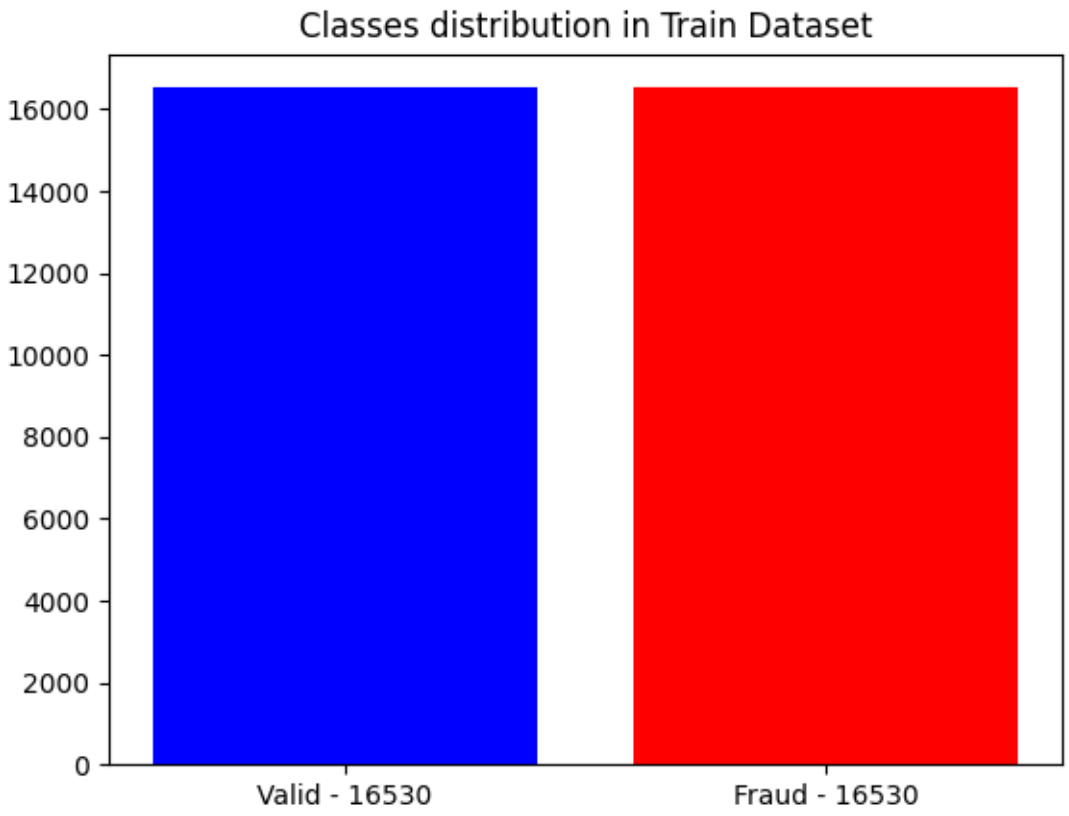

Рис. 17. Розподіл класів цільової мітки 'isFraud' тренувального датасету

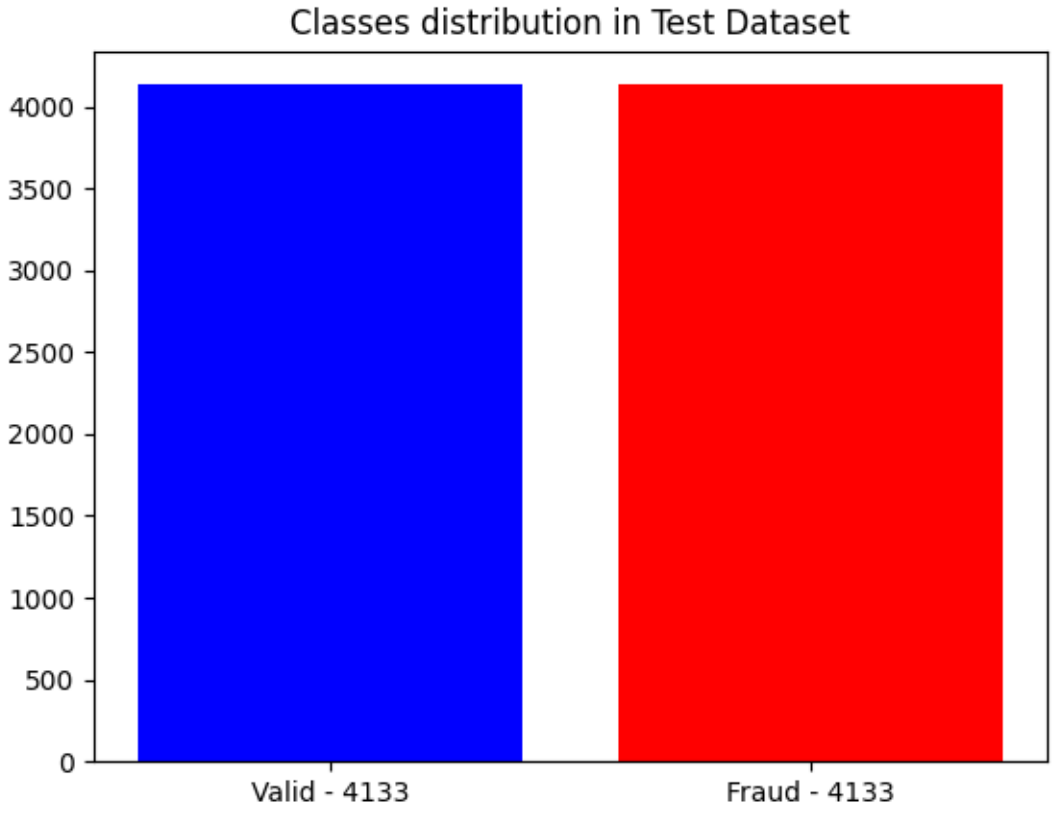

Рис. 18. Розподіл класів цільової мітки 'isFraud' тестового датасету

Етап 4: Оцінка інформативності входів методом RFE

Для оцінки інформативності входів обрано метод RFE (Recursive Feature Elimination) із використанням моделі RandomForestClassifier бібліотеки Scikit-Learn, оскільки він показав найкращі результати для Домашнього завдання №6. Результати використання алгоритму на тренувальних даних наведено на рис. 19.

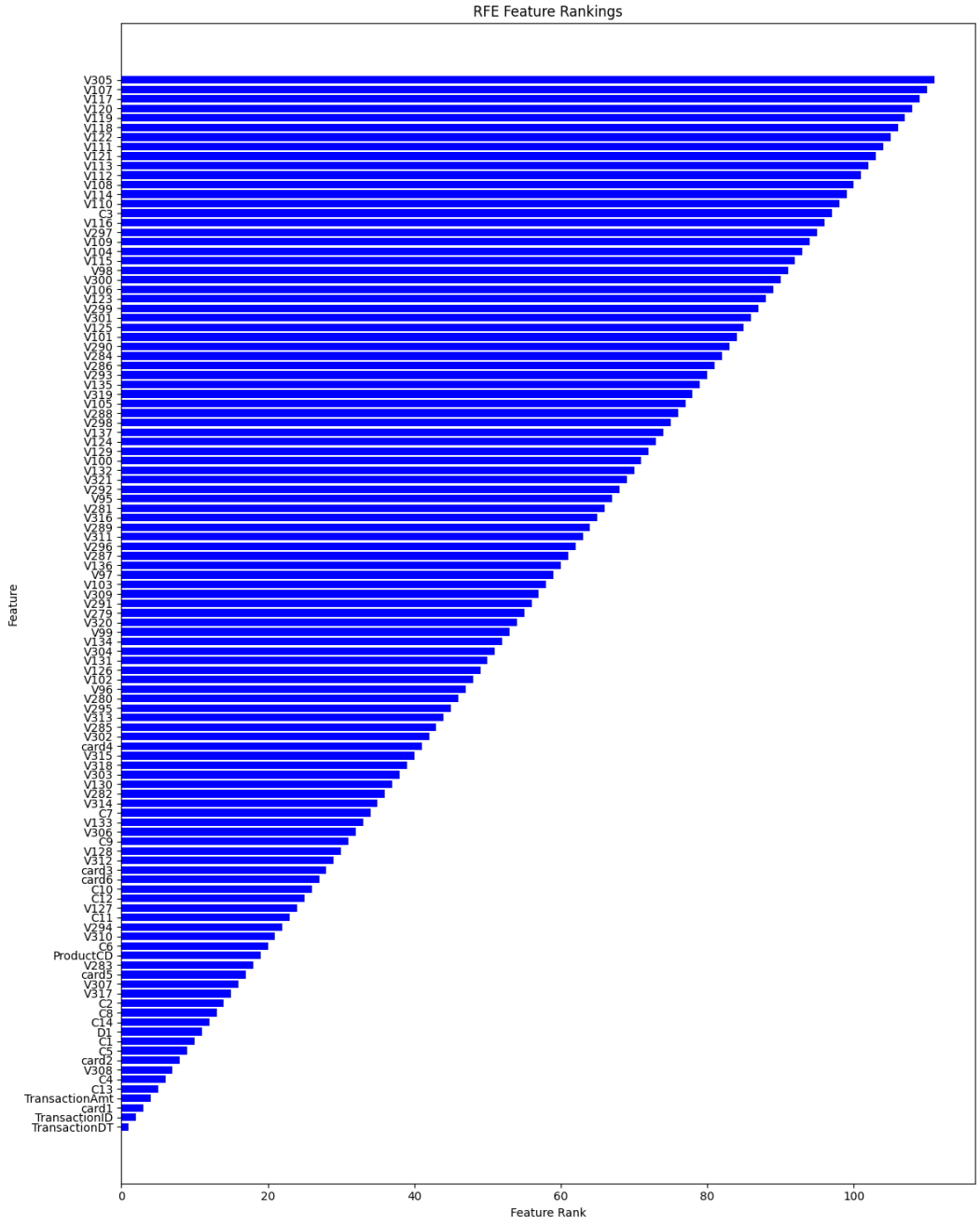

Рис. 19. Результат виконання алгоритму RFE (Recursive Feature Elimination)

Оскільки ознаки з нижчими значеннями рейтингу для алгоритму RFE вважаються більш важливими (є ознаками, які найбільше сприяють продуктивності моделі), із рис. 19 можна виділити топ-10 найважливіших ознак:

1) 'TransactionDT';
2) 'TransactionID';
3) 'card1';
4) 'TransactionAmt';
5) 'C13';
6) 'C4';
7) 'V308';
8) 'card2';
9) 'C5';
10) 'C1'.

Етап 5: Алгоритмічна реалізація гіпотези компактності

Перед безпосереднім використанням алгоритму гіпотези компактності було виконано декілька експериментів з метою визначення оптимальної кількості нейронів на вихідному шарі перцептрона (табл. 1).

Таблиця 1.

Тестування кількості нейронів на вихідному шарі перцептрона

Кількість нейронів	1	2	3	4	5
Середня точність для 5-ти експериментів, %	56.429	48.367	34.875	27.338	24.539
Максимальна точність, %	56.944	61.323	43.189	36.184	41.991

Із табл. 1 видно, що найкраща середня точність характерна для 1 нейрона на вихідному шарі одношарового лінійного перцептрона.

Наступним кроком реалізовано алгоритмічний варіант алгоритму гіпотези компактності (рис. 20).

```python
# Get feature names
current_accuracy = 0

while current_accuracy < 0.9 and X_train_scaled.shape[1] >= 10:
    print('\n')
    # Define and compile the model
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(1, input_dim=X_train_scaled.shape[1], activation='sigmoid')
    ])
    model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.RMSprop(), metrics=['accuracy'])

    # Train the model on a sample containing samples of both classes
    perceptron_history = model.fit(X_train_scaled, y_train, verbose=0, epochs=10, batch_size=64)

    # Estimate the accuracy of the model for the current number of features
    _, current_accuracy = model.evaluate(X_test_scaled, y_test)

    # Apply the identity (square) matrix with a zero row
    identity_matrix = np.identity(X_train_scaled.shape[1])
    offset = np.zeros((1, X_train_scaled.shape[1]))
    input_matrix = np.vstack([identity_matrix, offset])

    # Calculate the feedback for each row minus the offset
    feedback = model.predict(input_matrix)
    feedback_minus_offset = feedback - model.predict(offset)  # Subtract the offset term

    # Evaluate the weight of inputs in absolute terms
    weights = np.abs(feedback_minus_offset)

    # Find the least important feature
    least_important_feature_index = np.argmin(weights[:-1]) #weights[:-1]

    # Extract the least important feature from the training, validation and test data
    X_train_scaled = np.delete(X_train_scaled, least_important_feature_index, axis=1)
    X_test_scaled = np.delete(X_test_scaled, least_important_feature_index, axis=1)
    print(f"len(weights) = {len(weights)}")
    print(f"Removed feature index: {least_important_feature_index}; Number of features: {X_train_scaled.shape[1]}; Current accuracy: {current_accuracy:.5f}")
```

Рис. 20. Алгоритмічна реалізація алгоритму гіпотези компактності

Алгоритм, зображений на рис. 20, працює за наступним принципом:

1) Тренування одношарового лінійного перцептрона на тренувальних даних;
2) Оцінка точності перцептрона на тестових даних;
3) Формування ідентифікаційної (квадратної) матриці, яка містить і нульовий рядок, який використовується для визначення зміщення;
4) Отримання ваги вхідних параметрів, обчислених як абсолютне значення відгуку для кожного рядка мінус зміщення;
5) Ітеративне видалення найменш важливого на даній ітерації стовпця.

Умовою виходу із циклу є досягнення точності 90% і більше на тестових даних або зменшення кількості ознак у тренувальному/тестовому датасетах до 10.

На перших ітераціях виконання алгоритму точність на тестових даних становила приблизно 66-67% (рис. 21). Після вилучення 81-ої ознаки (коли кількість ознак становила 19) спостерігається різке збільшення точності від 66.949% до 71.631%, тобто на понад 4.5% (рис. 22). В результаті виконання алгоритму кількість ознак було зведено до 9 (виконалася умова виходу із циклу: кількість ознак>=10), а отримана точність становила 70.373% (рис. 23).

```
259/259 [==============================] - 1s 2ms/step - loss: 0.2145 - accuracy: 0.6731
4/4 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 31ms/step
len(weights) = 112
Removed feature index: 50; Number of features: 110; Current accuracy: 0.67312


259/259 [==============================] - 1s 2ms/step - loss: 0.2205 - accuracy: 0.6615
4/4 [==============================] - 0s 6ms/step
1/1 [==============================] - 0s 31ms/step
len(weights) = 111
Removed feature index: 63; Number of features: 109; Current accuracy: 0.66150


259/259 [==============================] - 1s 2ms/step - loss: 0.2189 - accuracy: 0.6628
4/4 [==============================] - 0s 6ms/step
1/1 [==============================] - 0s 31ms/step
len(weights) = 110
Removed feature index: 65; Number of features: 108; Current accuracy: 0.66284


259/259 [==============================] - 1s 2ms/step - loss: 0.2191 - accuracy: 0.6651
4/4 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 31ms/step
len(weights) = 109
Removed feature index: 49; Number of features: 107; Current accuracy: 0.66513
```

Рис. 21. Точність одношарового лінійного перцептрона на перших ітераціях

```
259/259 [==============================] - 1s 2ms/step - loss: 0.2231 - accuracy: 0.6695
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 15ms/step
len(weights) = 22
Removed feature index: 0; Number of features: 20; Current accuracy: 0.66949


259/259 [==============================] - 1s 2ms/step - loss: 0.1904 - accuracy: 0.7163
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 24ms/step
len(weights) = 21
Removed feature index: 0; Number of features: 19; Current accuracy: 0.71631
```

Рис. 22. Значне збільшення точності внаслідок видалення 81-ої ознаки

```
259/259 [==============================] - 1s 2ms/step - loss: 0.2012 - accuracy: 0.7037
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 32ms/step
len(weights) = 11
Removed feature index: 7; Number of features: 9; Current accuracy: 0.70373
```

Рис. 23. Результат виконання алгоритму гіпотези компактності

Як бачимо із рис. 23, хорошої точності алгоритмічна реалізація алгоритму гіпотези компактності не дала.

Етап 6: Використання алгоритму гіпотези компактності вручну

Оскільки алгоритмічна реалізація не показала хороших результатів, було прийнято рішення відкидати ознаки ітераційно і вручну, використовуючи алгоритм гіпотези компактності (рис. 24). Результат тренування одношарового лінійного перцептрона та визначення ваг вхідних параметрів наведено на рис. 25 і рис. 26.

```
# Get feature names
feature_names = X_train.columns

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
X_test_scaled = scaler.transform(X_test)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_dim=X_train_scaled.shape[1], activation='sigmoid')
])
model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.RMSprop(), metrics=['accuracy'])

# Train the model on a sample containing samples of both classes
perceptron_history = model.fit(X_train_scaled, y_train, epochs=10, batch_size=64)

# Estimate the accuracy of the model for the current number of features
_, current_accuracy = model.evaluate(X_test_scaled, y_test)

weights = np.abs(model.layers[0].get_weights()[0].flatten())

# Sort feature indices based on their weights
sorted_feature_indices = np.argsort(weights.squeeze())

# Sort feature names based on their weights
sorted_feature_names = feature_names[sorted_feature_indices]

# Create a bar plot for feature weights
plt.figure(figsize=(12, 15))
plt.barh(range(len(sorted_feature_names)), weights.squeeze()[sorted_feature_indices], align='center', color='blue')
plt.yticks(range(len(sorted_feature_names)), sorted_feature_names)
plt.xlabel('Feature Weight')
plt.ylabel('Feature')
plt.title('Feature Weights or Importance')
plt.tight_layout()
plt.show()
```

Рис. 24. Код для реалізації однієї ітерації алгоритму гіпотези компактності

```
Epoch 1/10
517/517 [==============================] - 1s 1ms/step - loss: 0.2008 - accuracy: 0.6961
Epoch 2/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1824 - accuracy: 0.7289
Epoch 3/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1803 - accuracy: 0.7312
Epoch 4/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1794 - accuracy: 0.7336
Epoch 5/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1787 - accuracy: 0.7339
Epoch 6/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1781 - accuracy: 0.7364
Epoch 7/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1777 - accuracy: 0.7375
Epoch 8/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1772 - accuracy: 0.7387
Epoch 9/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1768 - accuracy: 0.7393
Epoch 10/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1766 - accuracy: 0.7409
259/259 [==============================] - 0s 1ms/step - loss: 0.2161 - accuracy: 0.6723
```

Рис. 25. Результат тренування перцептрона (1-ша ітерація – 111 ознак)

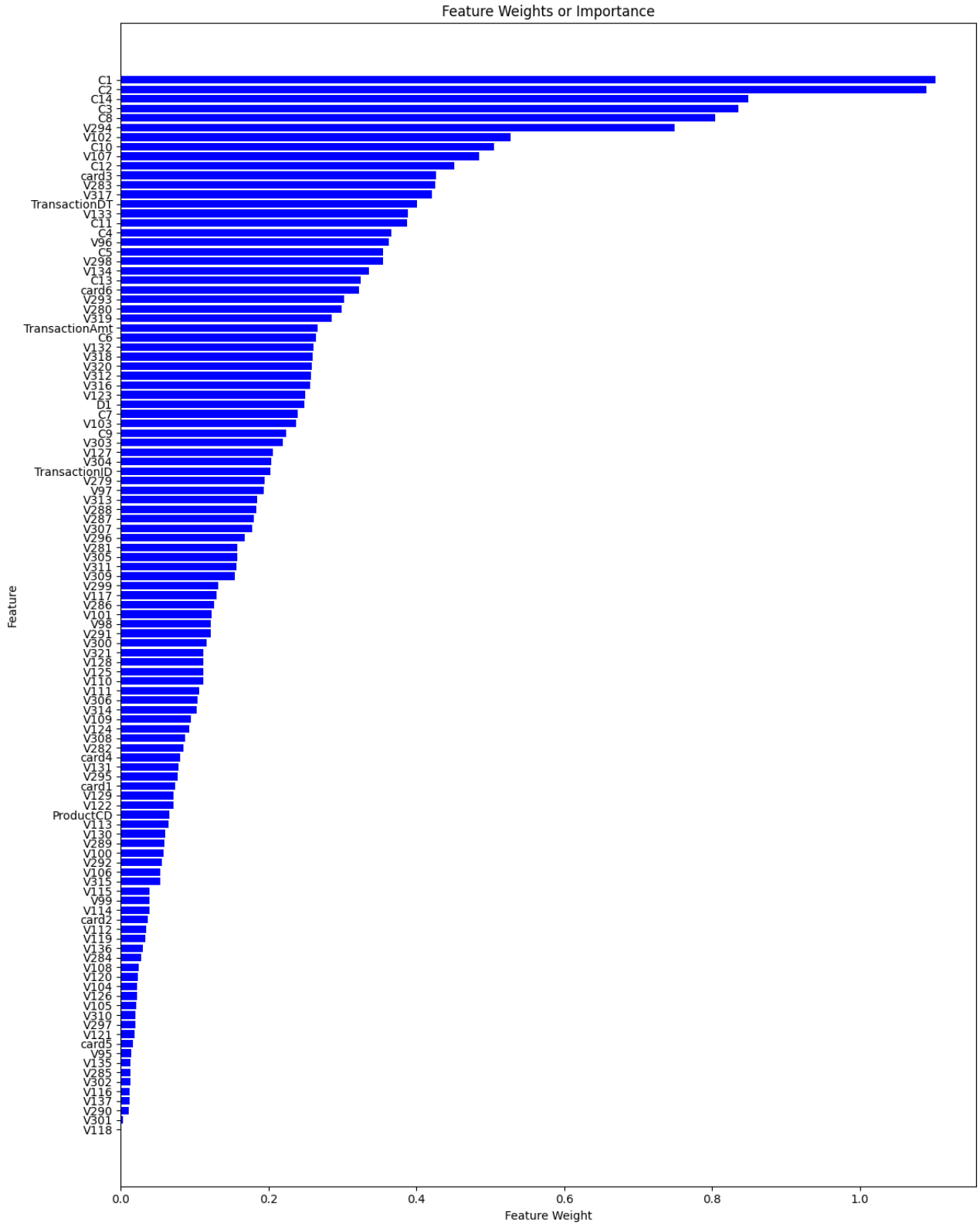

Рис. 26. Ваги ознак, отримані безпосередньо від нейрону натренованого перцептрона

Під час перших декількох ітерацій видалялися ознаки із найменшими вагами. Наприклад, внаслідок першої ітерації було вилучено 2 ознаки: 'V118' та 'V301', оскільки вони мали найменші ваги.

Також після вилучення ознак здійснювалася перевірка втрат точності моделі на тренувальних даних: якщо точність різко падала на 2-5%, то ознаки поверталися назад в датафрейм і відкидалися по одній з метою виявлення важливої ознаки.

Починаючи із 6-ої ітерації видалення ознак здійснювалося із аналізом результатів для двох попередніх ітерацій: якщо в двох попередніх ітераціях ознака була маловажливою, її було вилучено; інакше – ознака залишалася. Для кращого розуміння наведено приклад для вилучення ознак на 10-ій ітерації (рис. 27).

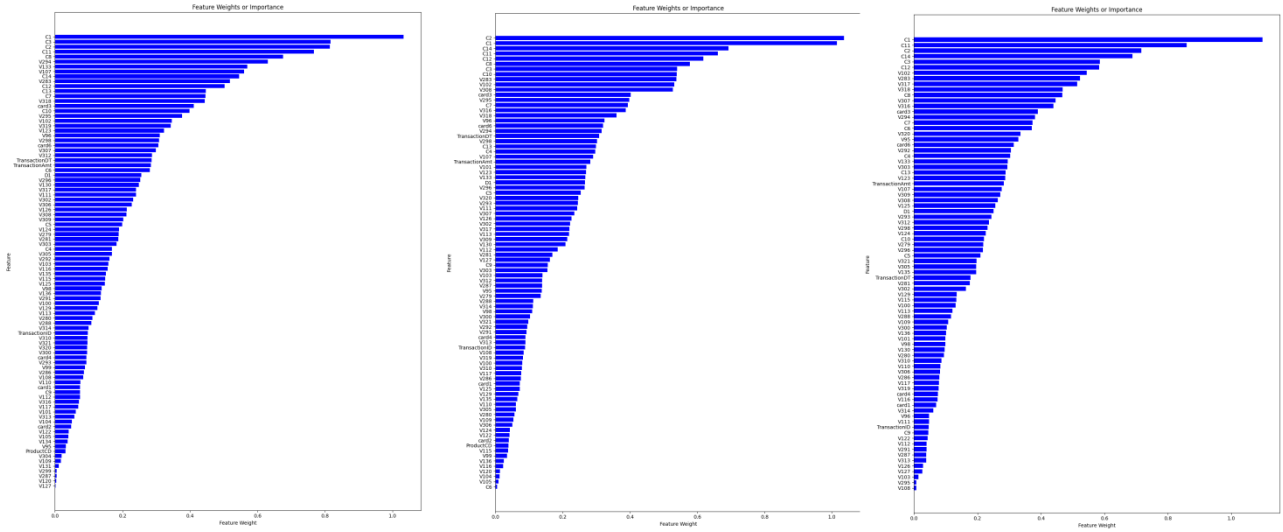

Рис. 27. Ваги ознак, отримані на 8-ій, 9-ій і 10-ій ітерації

Наприклад, на 10-ій ітерації найменш важливими виявилися ознаки: 'V108' та 'V295'. Ознака 'V108' була також не дуже важливою і на 8-ій та 9-ій ітераціях, тому її було вилучено. З іншої сторони, ознака 'V295' виявилася досить важливою на 8-ій і 9-ій ітераціях, тому її було залишено як стовпець датафрейму. За аналогічною схемою вилучалися/залишалися топ-10 найменш важливих ознак на кожній ітерації.

В результаті виконання 15-ти ітерацій вдалося звести кількість ознак до 48 (рис. 28), при цьому точність перцептрона майже не змінилася (рис. 29).

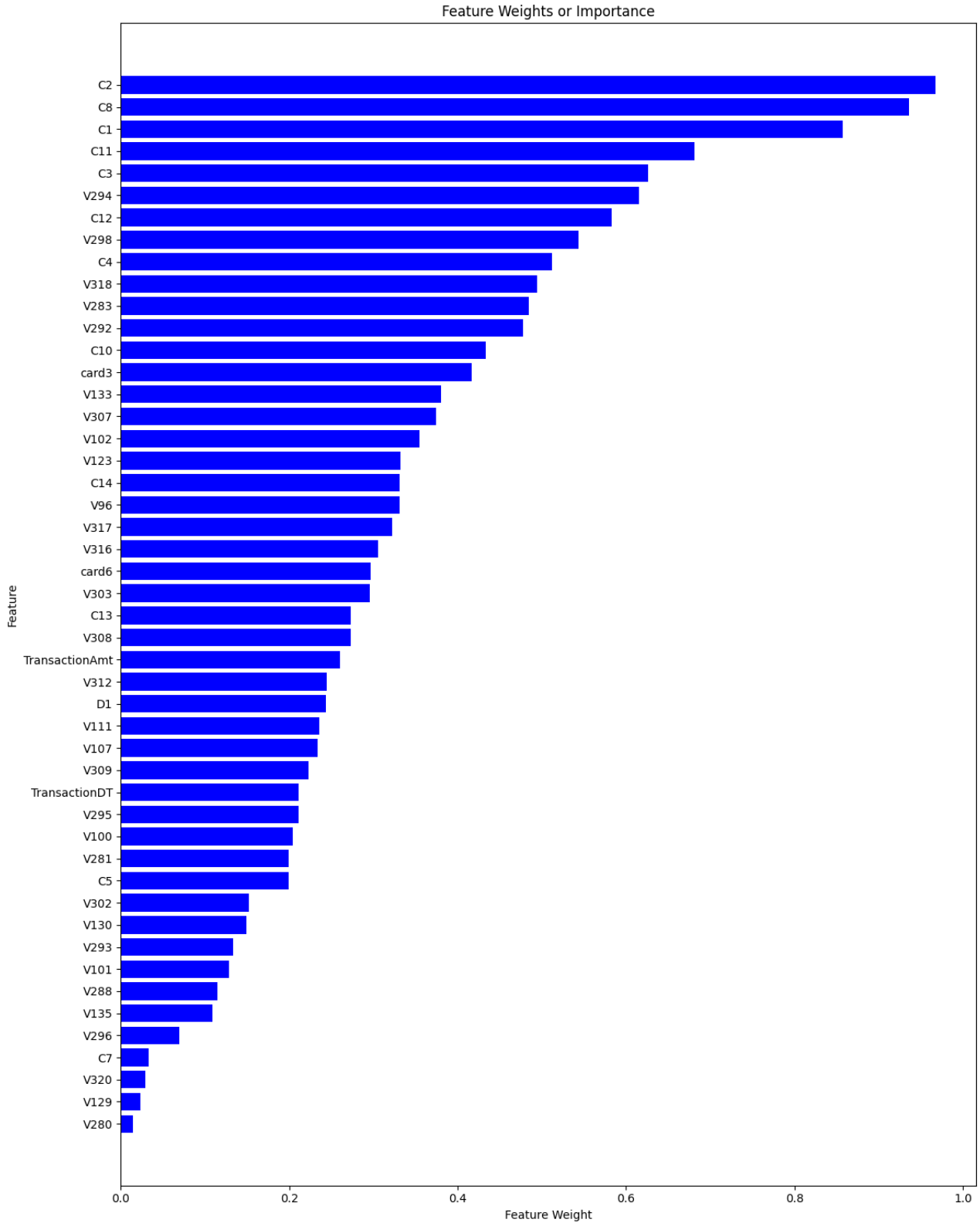

Рис. 28. Ваги ознак, отримані на 15-ій ітерації

```
Epoch 1/10
517/517 [==============================] - 1s 1ms/step - loss: 0.2048 - accuracy: 0.6873
Epoch 2/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1834 - accuracy: 0.7305
Epoch 3/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1811 - accuracy: 0.7339
Epoch 4/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1801 - accuracy: 0.7364
Epoch 5/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1795 - accuracy: 0.7372
Epoch 6/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1791 - accuracy: 0.7381
Epoch 7/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1788 - accuracy: 0.7386
Epoch 8/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1785 - accuracy: 0.7391
Epoch 9/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1783 - accuracy: 0.7404
Epoch 10/10
517/517 [==============================] - 1s 1ms/step - loss: 0.1781 - accuracy: 0.7410
259/259 [==============================] - 0s 1ms/step - loss: 0.2200 - accuracy: 0.6689
```

Рис. 29. Результат тренування перцептрона (15-та ітерація – 48 ознак)

Виходячи із результатів тренування лінійного одношарового перцептрона на кожній ітерації було також визначено топ-10 найвпливовіших ознак та отримано топ-10 найвпливовіших ознак для всіх 15-ти ітерацій:

1) 'C1';
2) 'C2';
3) 'C11';
4) 'V107';
5) 'C3';
6) 'C8';
7) 'V102';
8) 'V283';
9) 'C4';
10) 'C12'.

Порівнявши найважливіші ознаки, виділені з допомогою алгоритму гіпотези компактності із результатами алгоритму RFE (рис. 19), можна помітити значні відмінності. Однак, для обидвох алгоритмів ознаки 'C1' та 'C4' ввійшли в топ-10 найважливіших ознак.

Висновки

Для датасету, який містив дані про шахрайство було здійснено спочатку процес Data Cleaning: вилучено стовпці із занадто великою кількістю втрачених значень (понад 10%), вилучено записи, які мають відсутнє значення в одному із стовпців (NaN) та не представляють шахрайство (щоб не зменшувати кількість записів із isFraud=True, які становлять меншість). Відсутні значення записів, які представляють шахрайство, були замінені на модальні значення відповідного стовпця.

Три стовці, які містять назви категорій ('ProductCD', 'card4', 'card6') було замінено числовими мітками, інші категоричні стовпці ('card1', 'card2', 'card3' та 'card5') уже містили числові дані, тому їх не було змінено.

Далі було отримано збалансовані тренувальний і тестовий датасети та виділено топ-10 найважливіших ознак за допомогою алгоритму RFE.

Оптимальною кількістю нейронів для вихідного шару одношарового лінійного перцептрона виявилася 1 нейрон. Побудована алгоритмічна реалізація гіпотези компактності дала найкращий результат - 71.631% для 19 ознак.

Під час ітеративного використання алгоритму гіпотези компактності вручну, вдалося зменшити кількість ознак від 111 до 48, при цьому тренувальна і тестова точності майже не змінилися і становили 73-74% і 66-67% відповідно.

Порівнявши топ-10 найвпливовіших ознак для алгоритму RFE та ітеративного алгоритму гіпотези компактності, було зроблено висновки, що вони доволі сильно відрізняються. Водночас для обидвох алгоритмів ознаки 'C1' та 'C4' ввійшли в топ-10 найважливіших ознак.