Textual Anomaly Detection in Financial Reports

DSCI 6004: Natural Language Processing

1. Prasad Thamada
2. Roshini Bandi

Project Objectives

The objective of this project is to develop an automated system for detecting textual anomalies in financial reports through advanced NLP techniques.

The system aims to enhance the accuracy and efficiency of anomaly detection in financial documents, thereby assisting financial analysts and auditors in identifying potential fraud or errors

DATASET

```
data = pd.read_csv('creditcard_data.csv')
data.head()
```

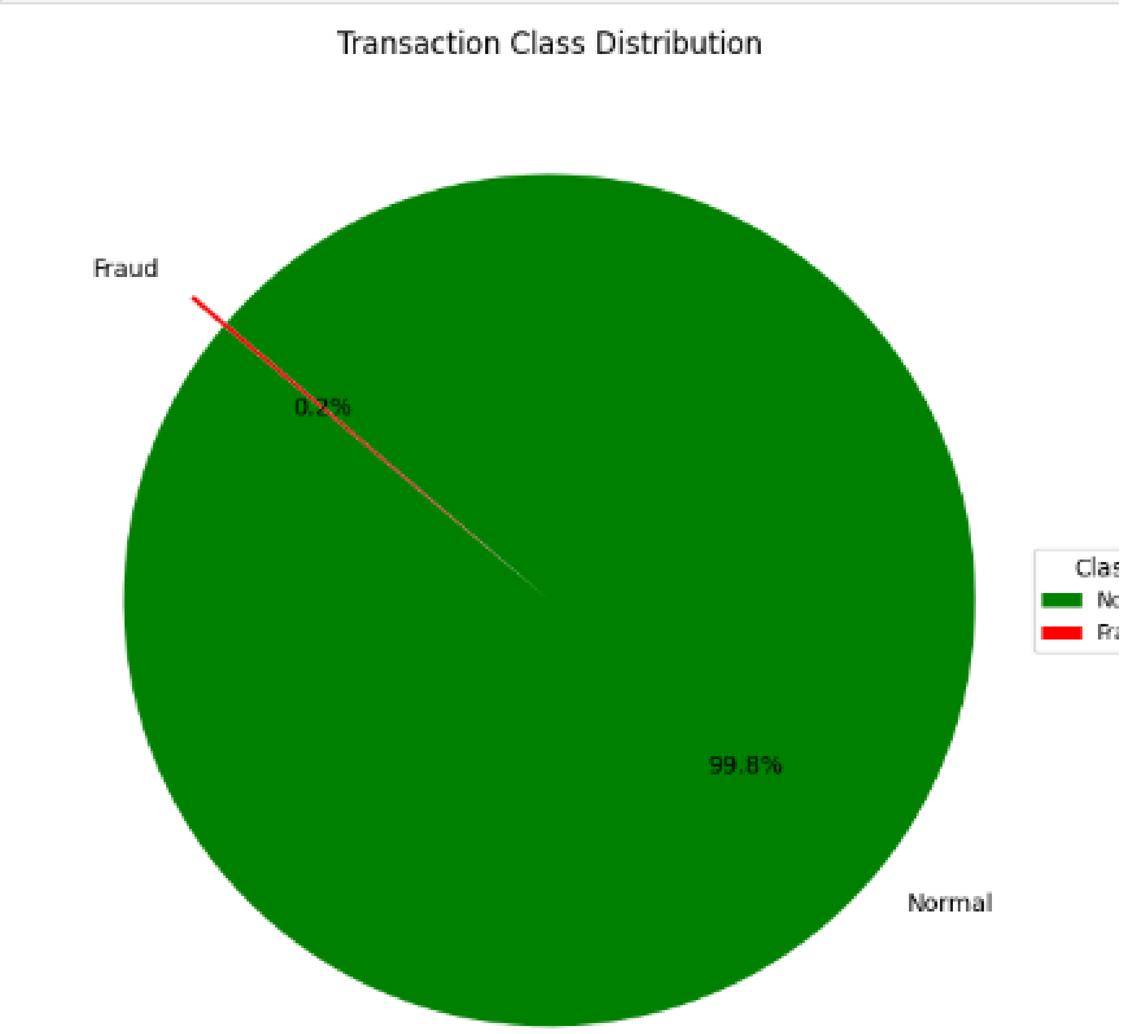
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.1
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.1
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.5

5 rows × 31 columns

Anomalies Trend

The pie chart shows the distribution of transaction classes according to fraud.

It reveals that the vast majority, 99.8%, of transactions are classified as normal. Fraudulent transactions account for a very small portion, just 0.2%.



Model

- The Autoencoder class defines a simple autoencoder model with a linear encoder-decoder architecture. It learns to compress input data into a lower-dimensional latent space representation (encoding_dim) and then reconstructs the original input from this encoded representation.
- The choice of activation functions (ReLU for encoding and sigmoid for decoding) is common in autoencoder architectures for non-linear mapping and bounded output values, respec

```python
#  Autoencoder model for anomaly detection
class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Linear(input_dim, encoding_dim)
        self.decoder = nn.Linear(encoding_dim, input_dim)

    def forward(self, x):
        x = torch.relu(self.encoder(x))
        x = torch.sigmoid(self.decoder(x))
        return x
```

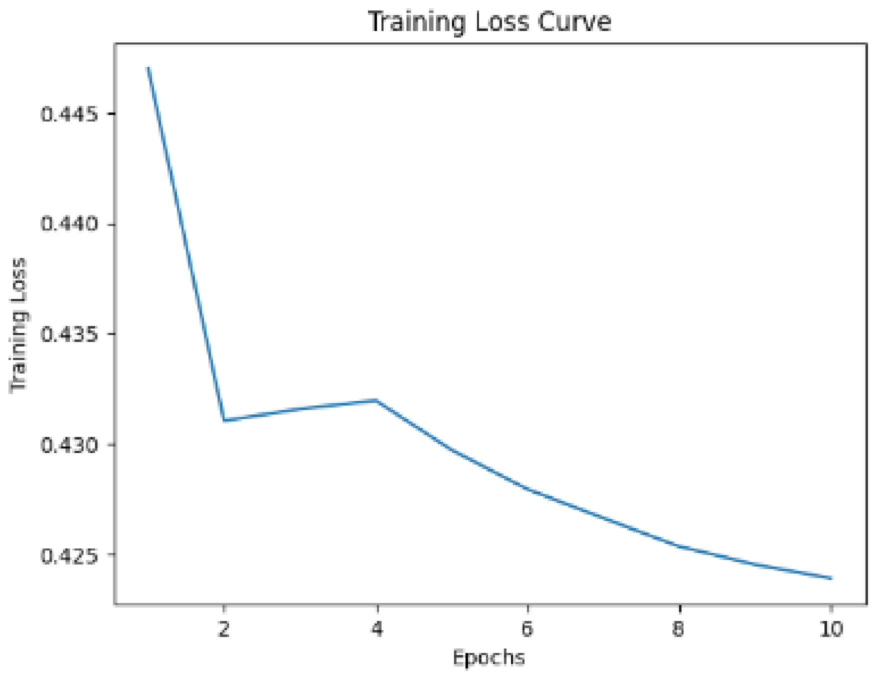
Hyperparameters and Finetuning

- The MSE Loss function calculates the mean squared difference between the model's predictions and the actual target values during training.
- The Adam optimizer updates the model's parameters (weights and biases) based on the calculated gradients during backpropagation, with a learning rate (lr) of 0.001.
- The training loop will iterate over the dataset for 50 epochs, with each batch containing 64 samples.

```
# Initialize the model, loss function, and optimizer
model = Autoencoder(input_dim, encoding_dim)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
epochs = 10
batch_size = 64
```

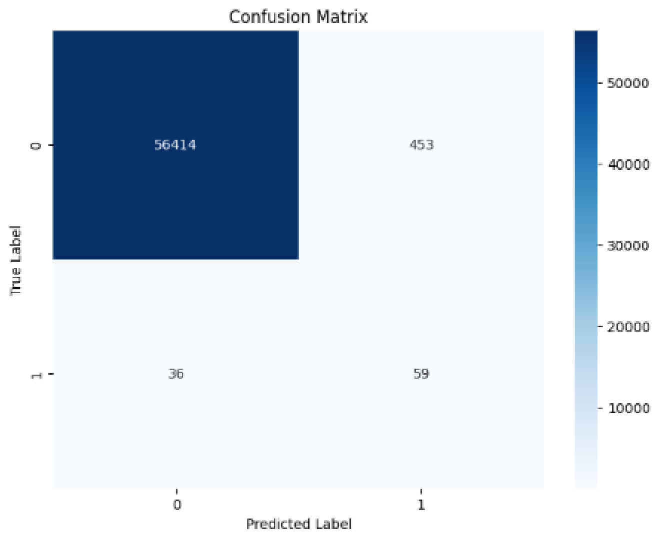
Model Training

- The loss trend during training shows a consistent decrease over the epochs, starting from an initial value of 0.447 and progressively reducing to 0.424 by the final epoch. This decline in loss reflects the model's ability to learn and optimize its parameters based on the training data, indicating improved performance and reduced errors.
- While the rate of decrease slows down in later epochs, the overall trend suggests that the model is converging towards a more optimal solution.
- Monitoring loss trends during training is crucial as it provides insights into the training progress, the effectiveness of the chosen optimizer and learning rate, and helps in assessing the model's capacity to generalize well to unseen data.



Validation Results

- The validation results for the model in Textual Anomaly Detection in Financial Reports showcase a mixed performance. The high accuracy of 99.14% indicates that the model is proficient in classifying data points correctly.
- However, the precision of 11.52% suggests that the model has a high rate of false positives, meaning that it incorrectly identifies normal data points as anomalies quite frequently.
- The recall of 62.11% indicates that the model successfully detects a significant portion of actual anomalies but may miss some anomalies, leading to false negatives. The F1 score of 19.44% balances precision and recall, showcasing the model's effectiveness in anomaly detection but also highlighting the need for improvements, especially in reducing false positives and enhancing recall for a more comprehensive anomaly detection system in financial reports.




```
Validation Results:
Accuracy: 0.9914153295179242
Precision: 0.115234375
Recall: 0.6210526315789474
F1 Score: 0.1943986820428361
```

Deployment

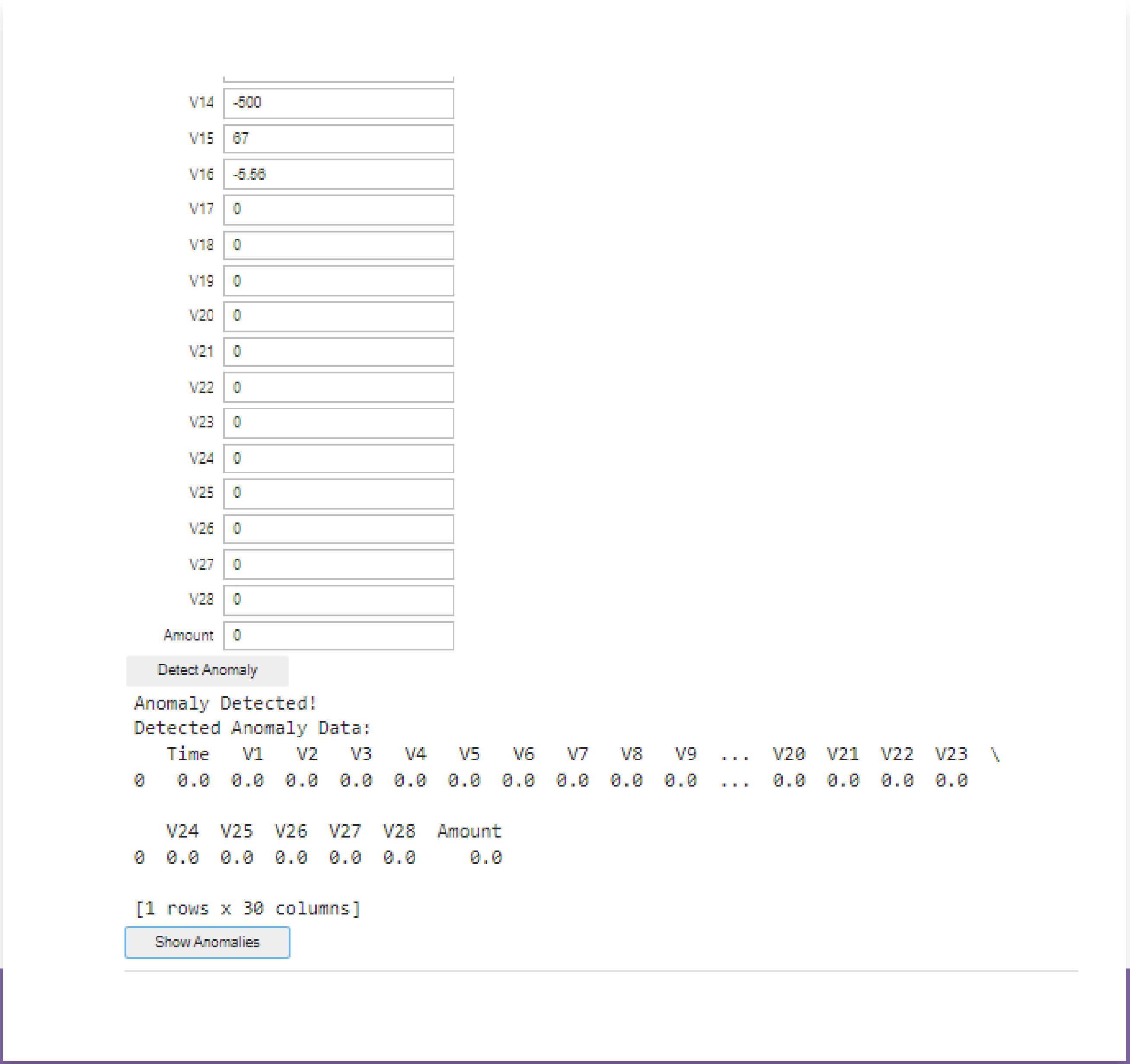
- The model detected a total of 512 anomalies based on the provided data.
- The dataset, we see instances where the model flagged transactions with notable reconstruction errors as anomalies.
- For instance, transaction at index 0 has a reconstruction error of 62.61, transaction at index 300 has a reconstruction error of 7.88, transaction at index 389 has a reconstruction error of 23.85, transaction at index 459 has a reconstruction error of 11.77, and transaction at index 463 has a reconstruction error of 8.52.

```python
# nomalies in the test data based on reconstruction errors
anomalies = merged_data[merged_data['Reconstruction_Error'] > threshold]
print(f"Number of anomalies detected: {len(anomalies)}")
print(anomalies.head())
```

```
Number of anomalies detected: 512
      Time        V1        V2        V3        V4        V5        V6  \
0      0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
300  217.0 -2.421230 -1.369602  2.261281  2.011034  1.878525 -1.275607
389  284.0  1.141436  0.081893  0.503625  1.487212 -0.473170 -0.411384
459  336.0 -0.895224  0.562106  2.817524 -0.718734  0.223222  0.796156
463  340.0  1.195494  0.194929  0.617510  0.649717 -0.474718 -0.716084

           V7        V8        V9  ...       V22       V23       V24  \
0    0.239599  0.098698  0.363787  ...  0.277838 -0.110474  0.066928
300 -1.251029  0.212619  0.142608  ...  0.157737 -0.624865  0.487156
389 -0.053193  0.071036  0.553486  ... -0.150234  0.001322  0.369459
459  0.464887 -0.002081  0.387537  ...  0.221249 -0.380422 -0.245721
463 -0.027078 -0.073385  0.057251  ... -0.590119  0.210111  0.388014

          V25       V26       V27       V28  Amount  Class  \
0    0.128539 -0.189115  0.133558 -0.021053  149.62      0
300  0.270894 -0.093370  0.330056 -0.056340   24.00      0
389  0.556295 -0.326645  0.028101  0.015215    7.89      0
459  0.202958  0.320802 -0.174340 -0.331954    7.72      0
463  0.092789  0.104973 -0.013644  0.018238    0.99      0

     Reconstruction_Error
0               62.606667
300              7.875507
389             23.847078
459             11.766143
463              8.519963
```

Deployment

- Deploying textual anomaly detection models with widgets empowers users to interactively explore and refine anomaly detection strategies, leading to more accurate and actionable insights from textual data.



Cited Work

1. Barrett, L., Fletcher, S., Ortan, A., & Kingan, R. (2019, October). Textual Outlier Detection and Anomalies in Financial Reporting. In Proceedings of the 2nd KDD Workshop on Anomaly Detection in Finance (pp. 1-10). Anchorage, Alaska, USA: Bloomberg BNA.
2. Bertero, C., Roy, M., Sauvanaud, C., & Tredan, G. (2017, October). Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection. In Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE) (pp. 1-8). doi:10.1109/ISSRE.2017.43

Thankyou