# Detecting Economic Crime using Deep Autoencoder Neural Network

##### **Amir Yunus**

Data Science Immersive, General Assembly, Singapore  
amirullah@hotmail.sg

***Abstract* –** Learning to detect economic crime in large-scale accounting data is one of the long-standing challenges in financial statement audits or fraud investigations. Currently, the majority of applied techniques refer to hand-crafted rules derived from known fraud scenarios. Whilst fairly successful, these rules exhibit the drawback that they often fail to generalise beyond known fraud scenarios and fraudsters gradually find ways to circumvent them.

To overcome this disadvantage, we propose the application of a deep autoencoder neural network to detect anomalous journal entries. We demonstrate that the trained network’s reconstruction error obtainable for a journal entry can be interpreted as a highly adaptive anomaly assessment.

Experiments on three datasets of journal entries, show the effectiveness of the approach resulting in high F1-Scores of 0.971 (train dataset) and 1.000 (test dataset). Our experiment also resulted in less false positive alerts compared to baseline methods. Initial feedback received by peers underpinned the quality of the approach in capturing highly relevant accounting anomalies.

***Index Terms*** - Accounting Information Systems · Enterprise Resource Planning (ERP) · Computer Assisted Audit Techniques (CAATs) · Journal Entry Testing · Forensic Accounting · Fraud Detection · Forensic Data Analytics · Deep Learning

1. **Motivation**

The Association of Certified Fraud Examiners (ACFE) estimates that a typical organisation loses 5% of its annual revenues due to fraud [1]. Economic crime, or commonly known as fraud, refers to “the abuse of one’s occupation for personal enrichment through the deliberate misuse of an organisation’s resources or assets” [47].

**“The median loss of a single financial statement fraud case is USD 150 thousand. The duration from the fraud perpetration till its detection was 18 months.”**

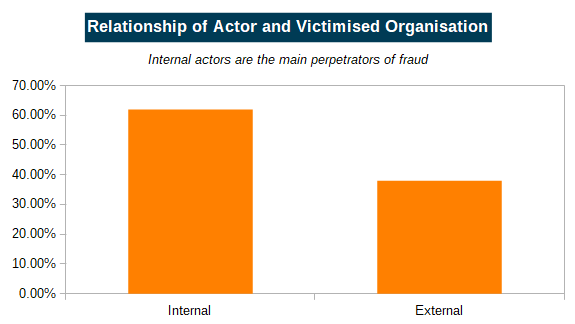
*Source: Association of Certified Fraud Examiners*

A similar study, conducted by PricewaterhouseCoopers (PwC) revealed that nearly 25% of respondents experienced losses between USD 100 thousand and USD 1 million due to fraud [39]. The study also showed that financial statement fraud caused by far the highest median loss surveyed fraud schemes. At the same time, organisations accelerate the digitisation and reconfiguration of business processes affecting in particular the Accounting Information Systems (AIS) or commonly known as Enterprise Resource Planning (ERP) systems [32]. SAP, one of the most common ERP providers, estimates that approximately 77% of the world’s transaction touches one of their systems [41].

**“49 percent of respondents said that their organisation has been a victim of fraud or economic crime in   
the past 24 months.”**

*Source: PricewaterhouseCoopers*

Generally, international audit standards require the direct assessment of journal entries to detect potentially fraudulent activities [2], [22]. These techniques, usually based on some known fraudulent scenarios, are often referred to as “red-flag” tests or statistical analyses such as Benford’s Law [9]. However, these tests fail to generalise beyond historical fraud cases and therefore, unable to detect contemporary fraud methods.



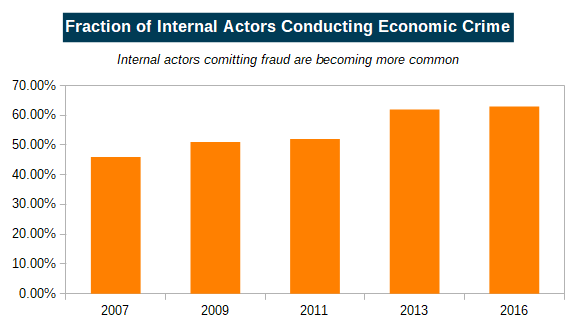
*Source: PricewaterhouseCoopers*

Recent developments in deep learning enables data scientists to extract complex, non-linear features, from raw sensory data, leading to advancements across many domains such as computer vision and speech recognition [29], [31], [34]. This method can supplement the accountants and forensic examiners toolbox [38].

**“Our ERP applications touch 77% of global transaction revenue.”**

*Source: SAP*

In order to conduct fraud, perpetrators need to deviate from regular system usage or posting pattern. Such deviations are recorded by a very limited number of “anomalous” journal entries. Our anomaly assessment is highly adaptive and it allows to flag entries as “anomalous” if they exceed a predefined scoring threshold. We evaluate the proposed method based on three anonymised datasets of journal entries extracted from large-scale SAP ERP systems.



*Source: PricewaterhouseCoopers*

In section 2, we provide an overview of related works in the field of fraud detection. Section 3 follows with a description of the autoencoder network architecture and presents the proposed methodology to detect accounting anomalies. The experimental setup and results are outlined in section 4 and section 5.

1. **Related Work**

The task of detecting fraud and accounting anomalies has been studied by both practitioners and academia [3], [47]. Several references describe different fraud schemes and ways to detect unusual and “creative” accounting practices [44].

**Fraud Detection in Enterprise Resource Planning Data**

Bay, et al. used Naïve Bayes methods to identify suspicious general ledger accounts, by evaluating attributes derived from journal entries measuring any unusual general ledger account activity [8]. Their approach was enhanced by McGlohon, et al. by applying link analysis to identify groups of high-risk general ledger accounts [33].

Kahn, et al. created transaction profiles of SAP ERP users [26], [27]. Similarly, Islam, et al. used SAP R/3 system audit logs to detect known fraud scenarios and collusion fraud via a “red-flag” based matching of fraud scenarios [23].

Debreceny and Gray analysed dollar amounts of journal entries obtained from 29 US organisation [17]. In their work, they searched for violations of Benford’s Law, anomalous digit combinations as well as unusual temporal pattern such as end-of-year postings [9]. More recently, Poh-Sun, et al. demonstrated the generalisation of the approach it to journal entries obtained from 12 non-US organisations [43].

Jans, et al. used latent class clustering to conduct a univariate and multivariate clustering of SAP ERP purchase order transactions [24]. The approach was enhanced by a means of process mining to detect deviating process flows in an organisation procure to pay process [25]. Transactions significantly deviating from the cluster centroids are flagged as “anomalous” and are proposed for a detailed review by auditors.

Argyrou, et al. evaluated self-organising maps to identify “suspicious” journal entries of a shipping company [6]. In their work, they calculated the Euclidean distance of a journal entry and the code-vector of a self-organising map’s best matching unit. In subsequent work, they estimated optimal sampling thresholds of journal entry attributes derived from extreme value theory [7].

Concluding from the reviewed literature, the majority of references draw either on:

1. Historical accounting and forensic knowledge about various “red-flags” and fraud schemes; or
2. Traditional non-deep learning techniques

As a result, we see a demand for unsupervised and novel approaches capable to detect so far unknown scenarios of fraudulent journal entries [46].

**Anomaly Detection using Autoencoder Neural Networks**

Currently, autoencoder networks have been widely used in image classification, machine translation and speech processing [21], [30], [45]. Hawkins, et al. and Williams, et al. were probably the first who proposed autoencoder networks for anomaly detection [20], [48].

Since then, the ability of autoencoder networks to detect anomalous records was demonstrated in different domains such as X-ray images of freight containers, the KDD99, MNIST, CIFAR-10 as well as several other datasets from the UCI Machine Learning Repository [4], [5], [15], [50]. Zhou and Paffenroth enhanced the standard autoencoder architecture by an additional filter layer and regularisation penalty to detect anomalies [51].

Cozzolino and Verdoliva used the autoencoder reconstruction error to detect pixel manipulations of images [13]. The method was enhanced by recurrent neural networks to detect forged video sequences [16]. Paula, et al. used autoencoder networks in export controls to detect traces of money laundering and fraud by analysing volumes of exported goods [35].

1. **Detection of Accounting Anomalies**

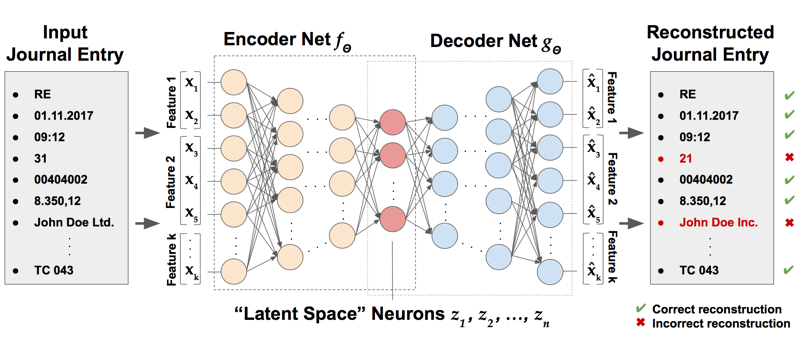
In this section, we introduce the main elements of autoencoder neural networks. We furthermore describe how the reconstruction error of such networks can be used to detect anomalous journal entries in large-scale accounting data.

**Deep Autoencoder Neural Networks**

An autoencoder or replicator neural network defines a special type of feed-forward multilayer neural network that can be trained to reconstruct its input. The difference between the original input and its reconstruction is referred to as reconstruction error.

In general, autoencoder networks are comprised of two non-linear mappings referred to as an encoder and decoder network [40]. Most commonly, the encoder and the decoder are of symmetrical architecture consisting of several layers of neurons each followed by a non-linear function and shared parameters. The encoder maps an input to a compressed representation in the latent space. This latent representation is then mapped back by the decoder to a reconstructed vector of the original input space.

The autoencoder is then trained to learn a set of optimal encoder-decoder model parameters that minimises the dissimilarity of a given journal entry and its reconstruction as faithfully as possible. For binary encoded attribute values, as used in this work, the model measures the deviation between two independent multivariate Bernoulli distribution [10].

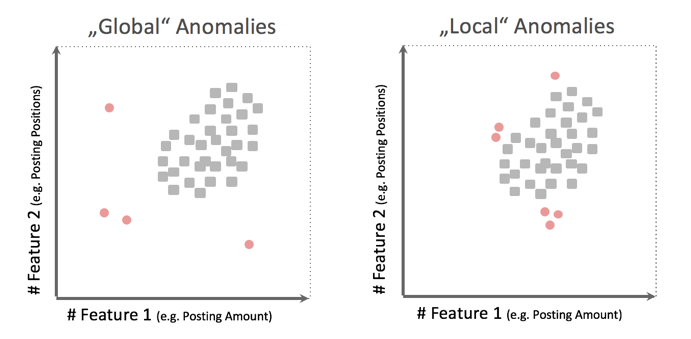


*Source: Marco Schreyer and Timur Sattarov*

To prevent the autoencoder from learning the identity function, the number of neurons of the networks hidden layers are reduced using a “bottleneck” architecture. Imposing such a constraint onto the network’s hidden layer forces the autoencoder to learn an optimal set of parameters that result in a “compressed” model of the most prevalent journal entry attribute value distributions and their dependencies.

**Classification of Accounting Anomalies**

We assume that the majority of journal entries recorded relates to regular business activities. In order to conduct fraud, perpetrators need to deviate from the “normal”. Such behaviour will be recorded by a very limited number of journal entries. Breunig, et al. distinguished two classes of anomalous journal entries, namely global and local anomalies [11].



*Source: Marco Schreyer and Timur Sattarov*

Generally, “red-flag” tests performed by auditors are designed to capture such anomalies. However, such tests often result in a high volume of false positive alerts due to events such as reverse postings, provisions and year-end adjustments usually associated with a low fraud risk. This type of anomaly is significantly more difficult to detect since perpetrators intend to disguise their activities by imitating a regular activity pattern. As a result, such anomalies usually pose a high fraud risk since they correspond to processes and activities that might not be conducted in compliance with organisational standards.

In this work, we detect any unusual value or unusual combination of values observed as anomalies. This was proposed by Das and Schneider on the detection of anomalous records in categorical datasets [14].

**Scoring of Accounting Anomalies**

Our score accounts for any “unusual” attribute value occurring in the journal entry. We do this by taking the reconstructed vectors from the decoder, and perform a matrix subtraction on the original input vectors. Next, we take the mean values of each journal entry and this gives us the reconstruction error.

Given that anomalous entries tend to be 0.02% of all entries recorded for the year, the model will sort the reconstruction error in descending order and focuses on the top entries. Next, a threshold score of 0.019 is applied and entries with reconstruction errors above the threshold will be flagged as anomalous.

1. **Experimental Setup and Network Training**

In this section, we describe the experimental setup and model training. We evaluated the anomaly detection performance of deep autoencoder architecture based on three datasets, namely the train dataset (533,009 entries), the test dataset (33,314 entries) and the clean dataset (33,307 entries with no anomaly).

**Datasets and Data Preparation**

In compliance with strict data privacy regulations, all journal entry attributes have been anonymised using an irreversible one-way hash function during the data extraction process.

Let’s start loading the dataset and investigate its structure and attributes:

# import pandas library

**import** pandas **as** pd

# load the dataset into the notebook

df\_train = pd.read\_csv('../data/train.csv')

# inspect the shape

print(f'Transactional dataset of {df\_train.shape[0]} rows and {df\_train.shape[1]} columns loaded')

Transactional dataset of 533009 rows and 10   
columns loaded

*Exploratory Data Analysis*

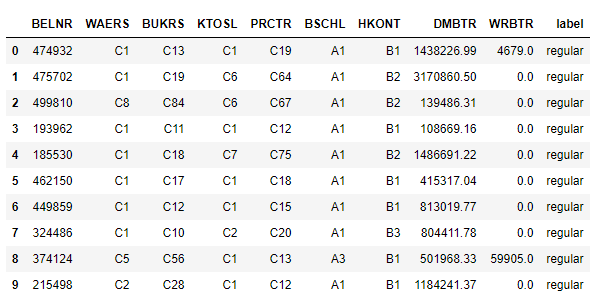
The dataset contains a subset of in total 7 categorical and 2 numerical features. Below is a list of the individual features as well as a brief description of their respective semantics:

* BELNR: The accounting number
* BUKRS: The company code
* BSCHL: The posting key
* HKONT: The posted general ledger account
* PRCTR: The posted profit centre
* WAERS: The currency key
* KTOSL: The general ledger account key
* DMBTR: The amount in local currency
* WRBTR: The amount in document currency
* Label: The “ground-truth” whether the entry is regular, or anomalous “global” or anomalous “local”

Let’s also have a closer look into the top 10 rows of the dataset:

# inspect top rows of dataset

df\_train.head(10)

**

There are only regular labels at the top. Let us have a closer look into the distribution of the regular versus anomalous transactions in the dataset:

# import matplotib and seaborn library

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

# display all graphs using high-quality images

**%**matplotlib inline

ip **=** get\_ipython()

ibe **=** ip.configurables[**-**1]

ibe.figure\_formats **=** {

'pdf',

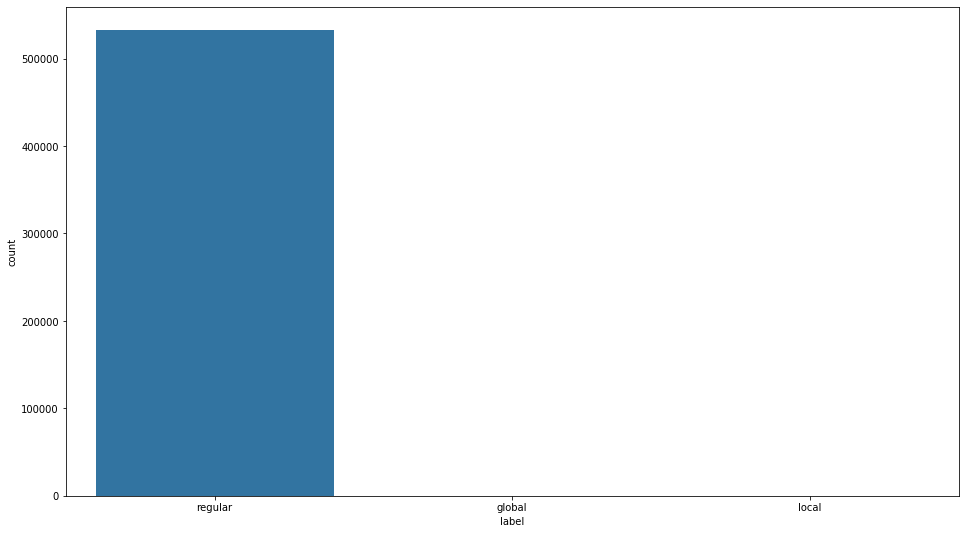
'png'

}

# plot distribution of feature

plt.figure(figsize**=**(16,9))

sns.countplot(df\_train.label);

**

The local and global anomalies cannot be seen by the graph above. Let us look at the value counts:

# number of anomalies vs. regular transactions

df\_train.label.value\_counts()

regular 532909

global 70

local 30

Name: label, dtype: int64

# number of anomalies vs. regular transactions

df\_train.label.value\_counts(normalize **=** **True**)

regular 0.999812

global 0.000131

local 0.000056

Name: label, dtype: float64

Similar to real world scenarios, we are facing a highly “unbalanced” dataset. Overall, the dataset contains only a fraction of 100 (0.018%) anomalous transactions. Of which, 70 (0.013%) are anomalous and 30 (0.005%) are “local” anomalies.

We will remove the label so that our model will predict without knowing the ground-truth. Similar to reality, auditors will not have entries pre-labelled whether it is anomalous or not.

# remove the "ground-truth" label information

label **=** df\_train.pop('label')

*Categorical Features*

From the initial assessment above, we can observe that the majority of attributes correspond to categorical (discrete) attribute values, e.g. the posting date, the general-ledger account, the posting type, the currency. Let us look into the distribution.

*BELNR*

# plot distribution of feature

plt.figure(figsize**=**(16,9))

sns.distplot(df\_train.BELNR)

plt.title('Distribution of BELNR observations', fontsize **=** 20);



The distribution is even. We can conclude that the values are unique and we will not be using this feature for our model.

*WAERS*

# prepare the plot

fig, ax **=** plt.subplots()

fig.set\_figwidth(16)

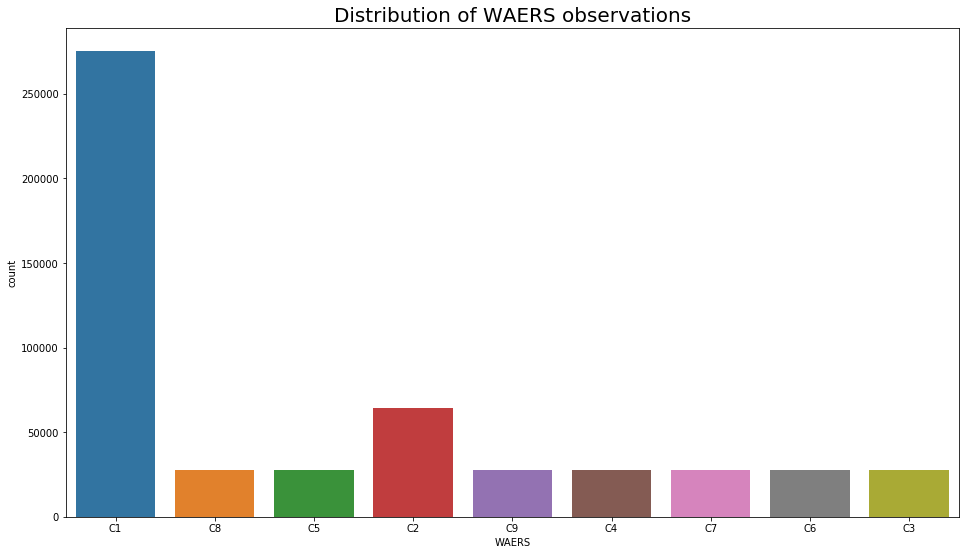
fig.set\_figheight(9)

# plot the distribution of the WAERS feature

g **=** sns.countplot(x**=**df\_train.loc[label**==**'regular', 'WAERS'])

g.set\_xticklabels(g.get\_xticklabels(), rotation**=**0)

g.set\_title('Distribution of WAERS observations', fontsize **=** 20);



C1 appears to be the most common attribute.

*BUKRS*

# prepare the plot

fig, ax **=** plt.subplots()

fig.set\_figwidth(16)

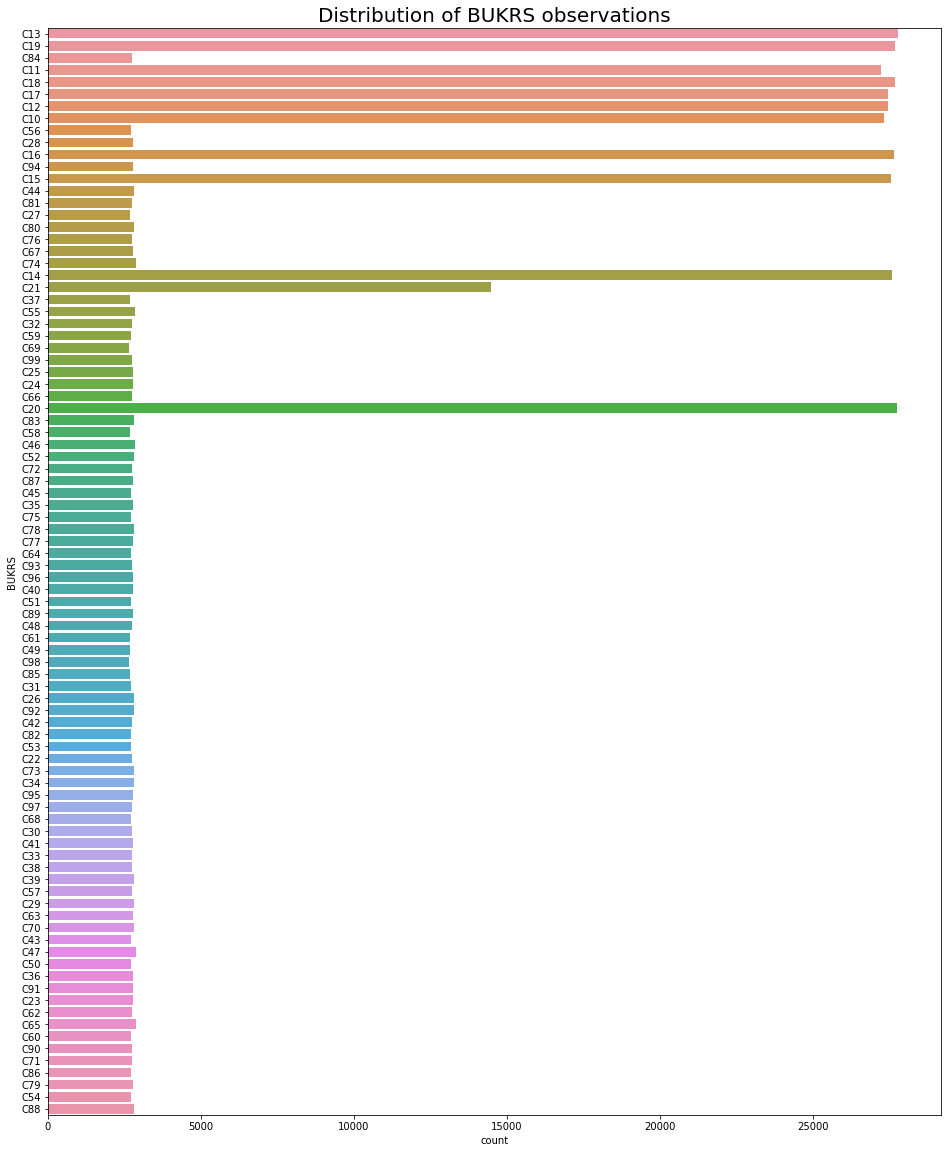
fig.set\_figheight(20)

# plot the distribution of BUKRS feature

g **=** sns.countplot(y**=**df\_train.loc[label**==**'regular', 'BUKRS'])

g.set\_yticklabels(g.get\_yticklabels(), rotation**=**0)

g.set\_title('Distribution of BUKRS observations', fontsize **=** 20);



Most attributes are below 5,000. The most outstanding attributes are C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20 and C21.

*KTOSL*

# prepare the plot

fig, ax **=** plt.subplots()

fig.set\_figwidth(16)

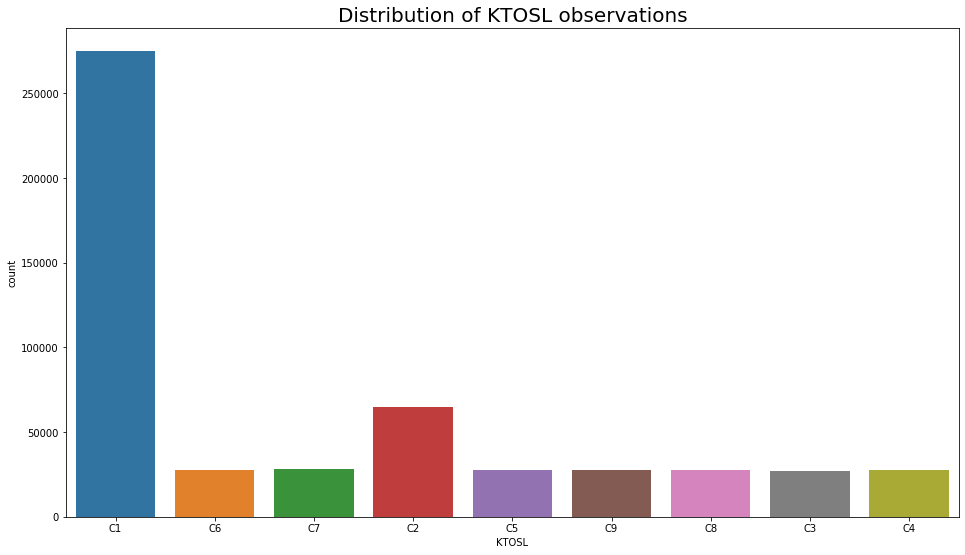
fig.set\_figheight(9)

# plot the distribution of KTOSL feature

g **=** sns.countplot(x**=**df\_train.loc[label**==**'regular', 'KTOSL'])

g.set\_xticklabels(g.get\_xticklabels(), rotation**=**0)

g.set\_title('Distribution of KTOSL observations', fontsize **=** 20);



C1 appears to be the most common attribute.

*BSCHL*

# prepare the plot

fig, ax **=** plt.subplots()

fig.set\_figwidth(16)

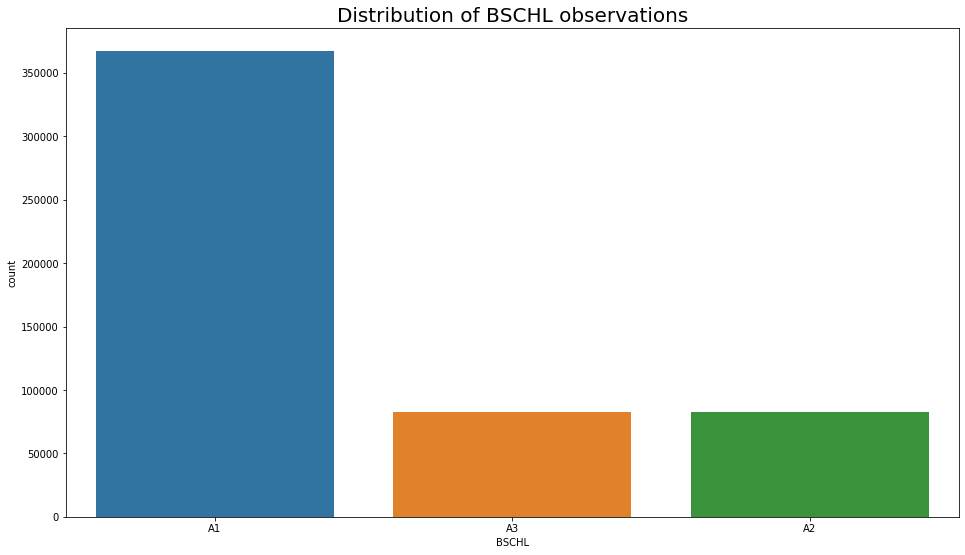
fig.set\_figheight(9)

# plot the distribution of BSCHL feature

g **=** sns.countplot(x**=**df\_train.loc[label**==**'regular', 'BSCHL'])

g.set\_xticklabels(g.get\_xticklabels(), rotation**=**0)

g.set\_title('Distribution of BSCHL observations', fontsize **=** 20);



A1 appears to be the most common.

*PRCTR*

# prepare the plot

fig, ax **=** plt.subplots()

fig.set\_figwidth(16)

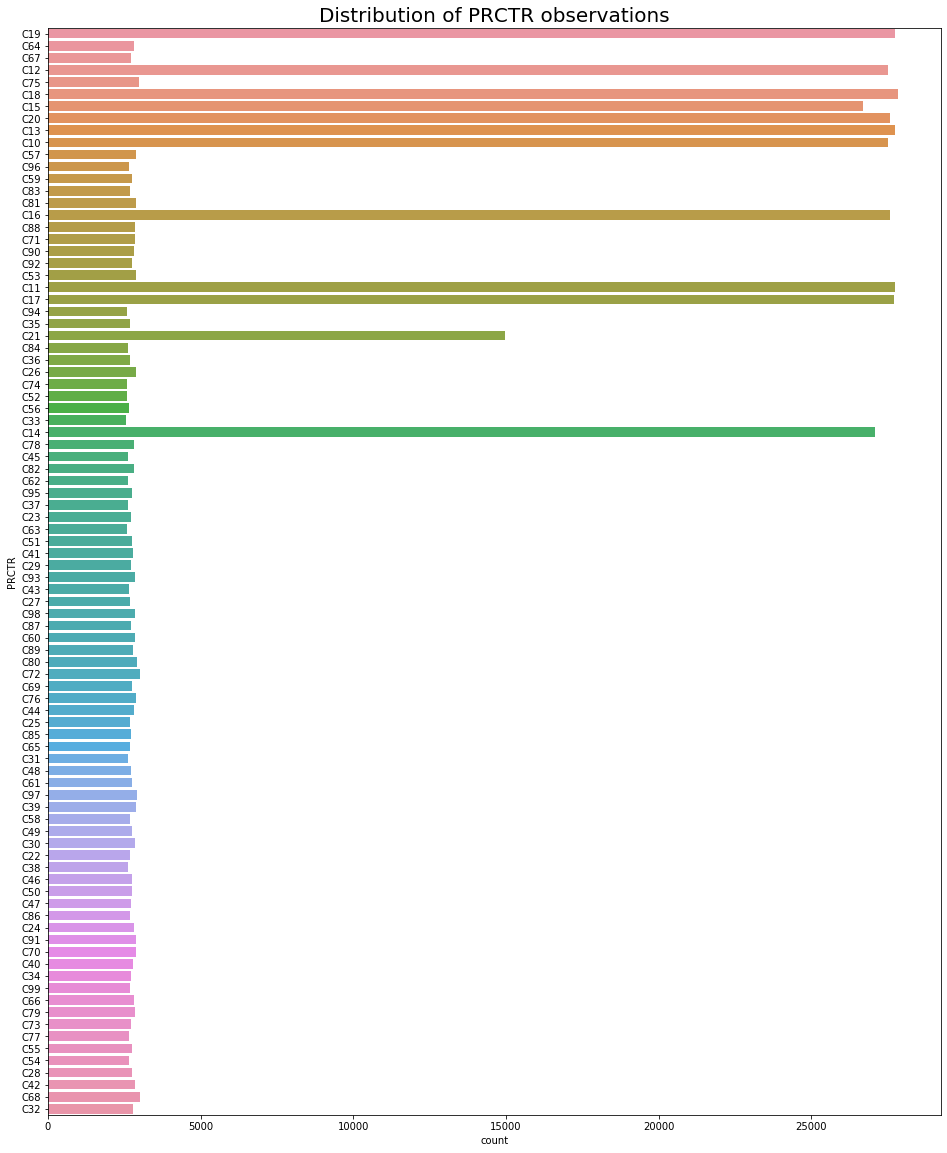
fig.set\_figheight(20)

# plot the distribution of PRCTR feature

g **=** sns.countplot(y**=**df\_train.loc[label**==**'regular', 'PRCTR'])

g.set\_yticklabels(g.get\_yticklabels(), rotation**=**0)

g.set\_title('Distribution of PRCTR observations', fontsize **=** 20);



Most attributes are below 5,000. The most outstanding attributes are C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20 and C21.

*HKONT*

# prepare the plot

fig, ax **=** plt.subplots()

fig.set\_figwidth(16)

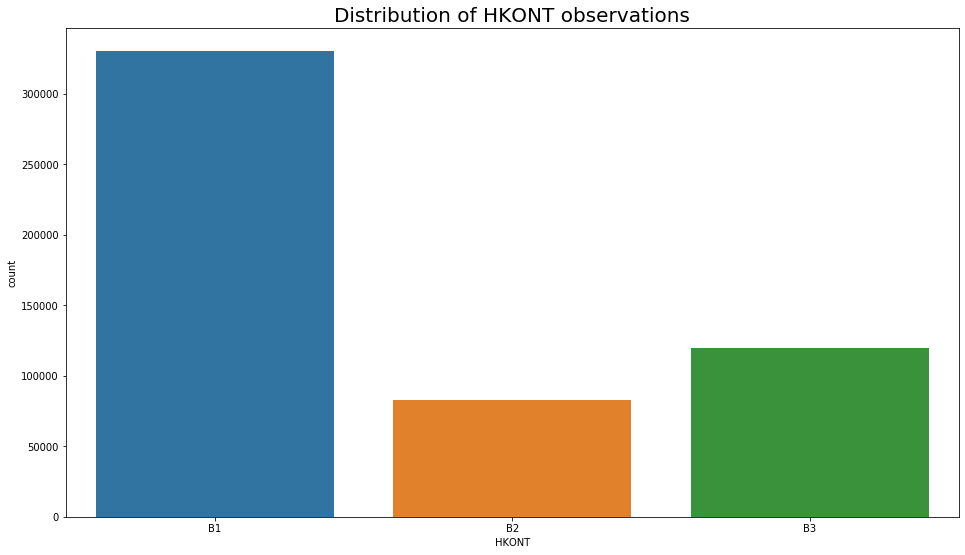
fig.set\_figheight(9)

# plot the distribution of HKONT feature

g **=** sns.countplot(x**=**df\_train.loc[label**==**'regular', 'HKONT'])

g.set\_xticklabels(g.get\_xticklabels(), rotation**=**0)

g.set\_title('Distribution of HKONT observations', fontsize **=** 20);



B1 appears to be the most common.

*Numerical Features*

Let us now inspect the distributions of the two numerical attributes contained in the transactional dataset, namely, the

1. Local currency amount DMBTR; and the
2. Document currency amount WRBTR

*DMBTR*

# plot distribution of DMBTR feature

plt.figure(figsize**=**(16,9))

sns.distplot(df\_train.DMBTR)

plt.title('Distribution of DMBTR observations', fontsize **=** 20);



Looks like it is heavily skewed.

*WRBTR*

# plot distribution of WRBTR feature

plt.figure(figsize**=**(16,9))

sns.distplot(df\_train.WRBTR)

plt.title('Distribution of WRBTR observations', fontsize **=** 20);



As expected, the distribution for both attributes are heavily tailed.

*Feature Engineering*

*One-Hot Encoding*

Unfortunately, neural networks are in general not designed to be designed to be trained directly on categorical data and require the attributes to be trained on to be numeric. One simple way to meet this requirement is by applying a technique referred to as “one-hot” encoding. Using this encoding technique, we will derive a numerical representation of each of the categorical attribute values. One-hot encoding creates new binary columns for each categorical attribute value present in the original data.

Using this technique, we will “one-hot” encode the 6 categorical attributes in the original transactional dataset. This can be achieved using the get\_dummies() function.

# select categorical attributes to be encoded

categ\_cols **=** ['WAERS', 'BUKRS', 'KTOSL', 'PRCTR', 'BSCHL', 'HKONT']

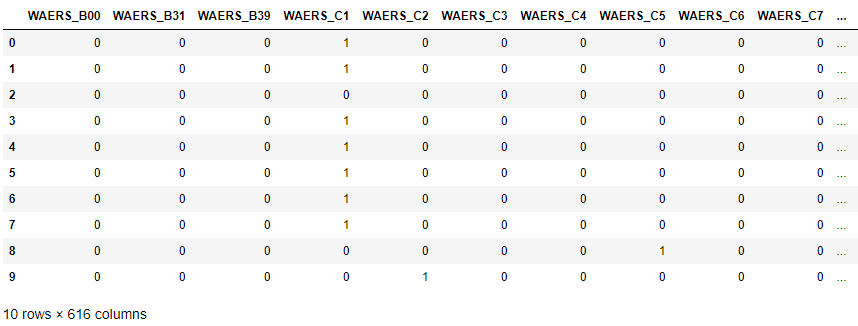
# encode categorical attributes into a binary one-hot encoded representation

df\_train\_categ\_transformed **=** pd.get\_dummies(df\_train[categ\_cols])

Let us inspect the top 10 sample transactions to see if we have been successful.

# inspect top rows

df\_train\_categ\_transformed.head(10)



*Log Transformation*

Recall that the numeric features are heavily tailed. In order to process faster, we first log-scale both variables and then min-max normalise the scaled amounts to the interval [0, 1].

# import numpy library

**import** numpy **as** np

# select "DMBTR" and "WRBTR" attribute

numeric\_cols **=** ['DMBTR', 'WRBTR']

# add a small epsilon to eliminate zero values from data for log scaling

numeric\_attr **=** df\_train[numeric\_cols] **+** 1e-7

numeric\_attr **=** numeric\_attr.apply(np.log)

# normalise all numeric attributes to the range [0,1]

df\_train\_numeric\_attr **=** (numeric\_attr **-** numeric\_attr.min()) **/** (numeric\_attr.max() **-** numeric\_attr.min())

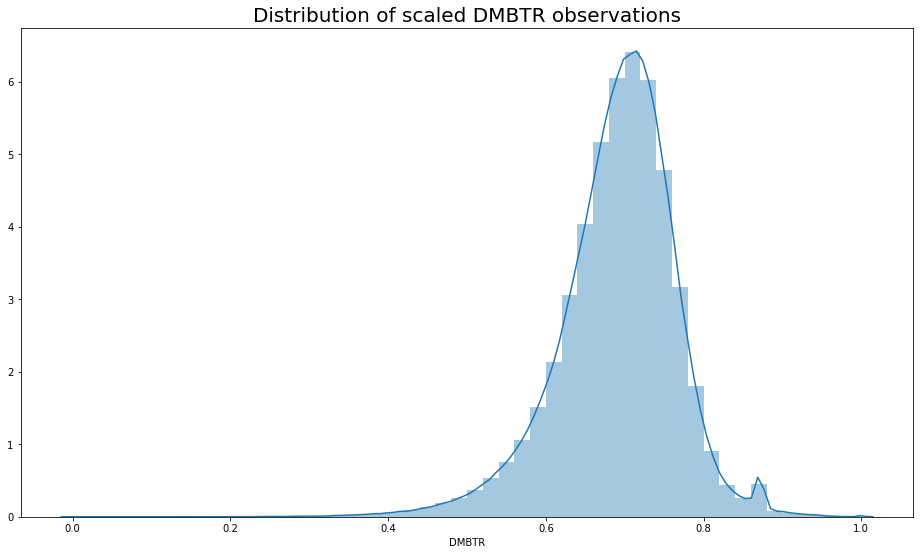
Let us visualise the log-scaled and min-max normalised distribution of both features.

# plot distribution of DMBTR feature

plt.figure(figsize**=**(16,9))

sns.distplot(df\_train\_numeric\_attr.DMBTR)

plt.title('Distribution of scaled DMBTR observations', fontsize **=** 20);

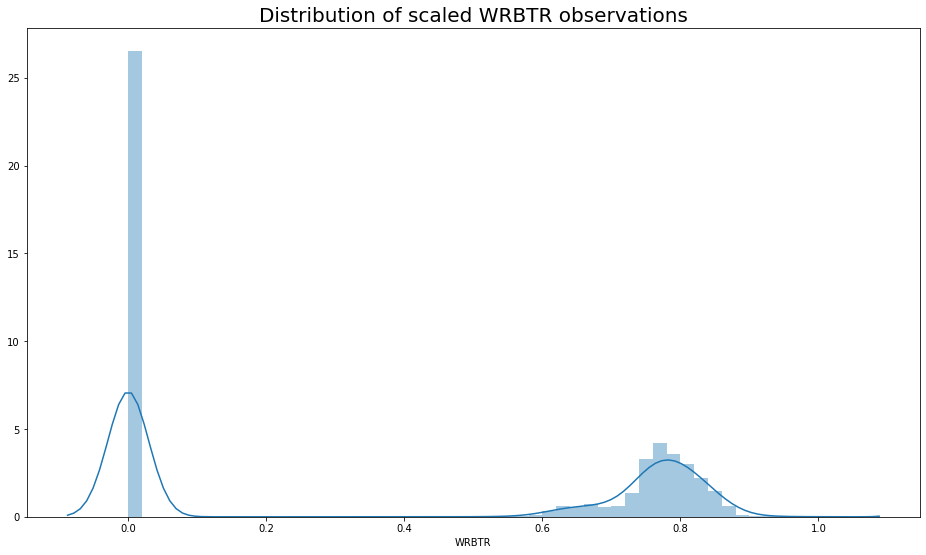


# plot distribution of WRBTR feature

plt.figure(figsize**=**(16,9))

sns.distplot(df\_train\_numeric\_attr.WRBTR)

plt.title('Distribution of scaled WRBTR observations', fontsize **=** 20);



*Merge Features*

Finally, we merge both pre-processed numerical and categorical attributes into a single dataset that we will use for training our deep autoencoder neural network.

# merge categorical and numeric subsets

df\_train\_transformed **=** pd.concat([df\_train\_categ\_transformed, df\_train\_numeric\_attr], axis **=** 1)

Review the shape of the dataset after we applied the distinct pre-processing steps to the attributes.

# inspect final dimensions of pre-processed transactional data

df\_train\_transformed.shape

(533009, 618)

Upon completion of all the pre-processing steps, we should end up with a dataset consisting of a total number of 533,009 entries (observations) and 618 features. The number of features will define the dimensionality of the input-layer and output-layer of our deep autoencoder neural network.

*Exploring using Benford’s Law*

**“Benford’s law is an observation about the frequency distribution of leading digits in many real-life sets of numerical data. In sets that obey the law, the number 1 appears as the most significant digit about 30 percent   
of the time, while 9 appears as the most significant   
digit less than 5% of the time.”**

*Source: Wikipedia*

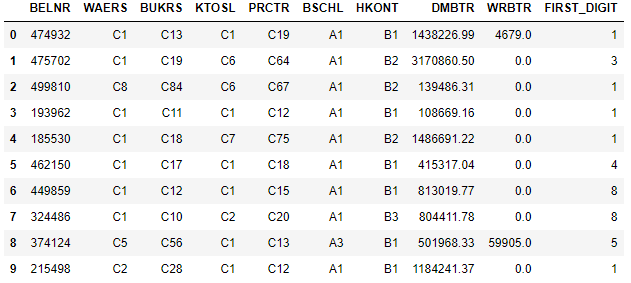
# make a copy of original dataset

df\_train\_bf **=** df\_train.copy()

# map out the first digit and display top 10 rows

df\_train\_bf['FIRST\_DIGIT'] **=** df\_train\_bf.DMBTR.map(**lambda** a: str(a)[0]).astype(int)

df\_train\_bf.head(10)



# display the actual percentage distribution of dataset

actuals **=** df\_train\_bf.FIRST\_DIGIT.value\_counts(normalize**=True**).sort\_index()

actuals

1 0.296899

2 0.160166

3 0.120465

4 0.098717

5 0.083355

6 0.072573

7 0.063357

8 0.055250

9 0.049217

Name: FIRST\_DIGIT, dtype: float64

# calculate the expected distribution based on Benford's Law

digits **=** list(range(1,10))

benford **=** [np.log10(1 **+** 1**/**d) **for** d **in** digits]

plt.figure(figsize **=** (16,9))

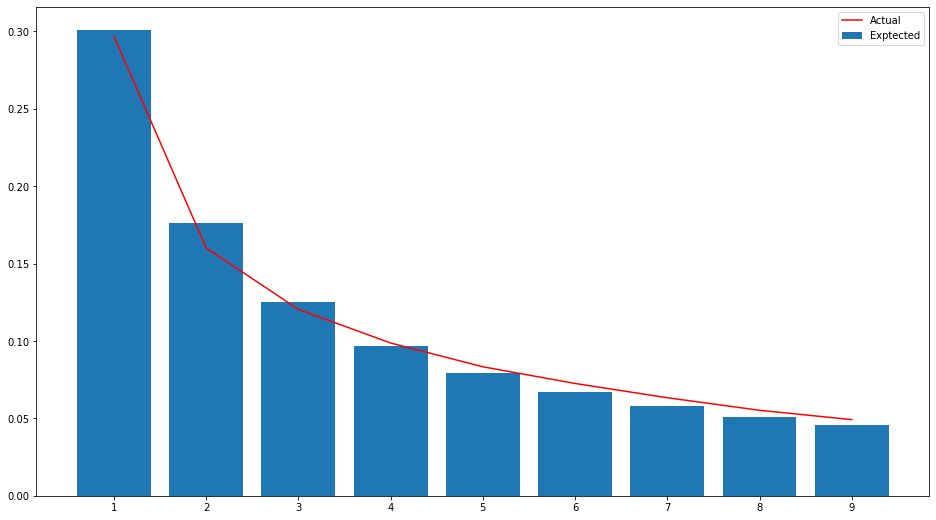
# plot graph to visualise distribution

plt.bar(digits, benford, label**=**'Exptected')

plt.plot(actuals, color**=**'r', label**=**'Actual')

plt.xticks(digits)

plt.legend();



The traditional method of Benford’s Law is unable to detect the anomalies in our dataset. However, we know they exist as we have the labels.

Let us proceed with the Principal Component Analysis.

*Exploring using Principal Component Analysis*

**“Principal Component Analysis (PCA) is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. In SciKit-Learn, PCA is implemented as a transformer object that learns components in its fit method, and can be used on new data to project it on these components”**

*Source: SciKit-Learn*

# create a copy for label

label\_int **=** label.copy()

# map label as integers

label\_int **=** label\_int.map({'regular': 0, 'local': 1, 'global': 1})

# create a copy for dataset

df **=** df\_train\_transformed.copy()

# import minmaxscaler module

**from** sklearn.preprocessing **import** MinMaxScaler

# perform min-max scaling

ss **=** MinMaxScaler()

df\_ss **=** ss.fit\_transform(df)

# import pca module

**from** sklearn.decomposition **import** PCA

# fitting the PCA algorithm with our Data

pca **=** PCA().fit(df\_ss)

# plotting the Cumulative Summation of the Explained Variance

plt.figure(figsize**=**(16,9))

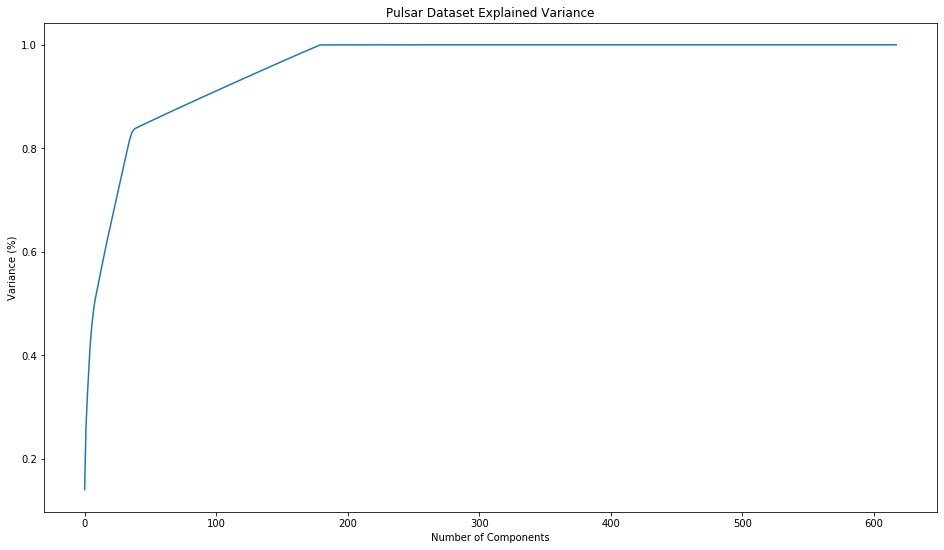
plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.xlabel('Number of Components')

plt.ylabel('Variance (%)') #for each component

plt.title('Pulsar Dataset Explained Variance')

plt.show()



# find the smallest number of components that has an explained variance ratio of at least 0.999

n\_components **=** np.where(np.cumsum(pca.explained\_variance\_ratio\_) **>=** 0.999)[0][0]

n\_components

179

# perform PCA transformation using 179 components

pca **=** PCA(n\_components**=**n\_components)

df\_pca **=** pca.fit\_transform(df\_ss)

# visualise the PCA

plt.figure(figsize**=**(16,9))

plt.scatter(df\_pca[:, 0], df\_pca[:, 1],

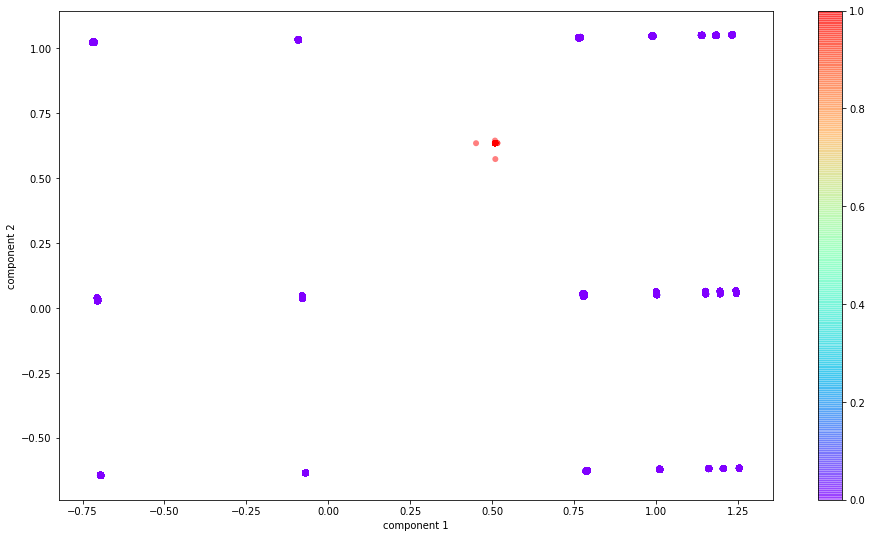
c**=**label\_int, edgecolor**=**'none', alpha**=**0.5,

cmap**=**plt.cm.get\_cmap('rainbow'))

plt.xlabel('component 1')

plt.ylabel('component 2')

plt.colorbar();



Using PCA, we can visualise that there is an anomaly cluster in the middle. However, we are not able to determine which entries are anomalous.

Let us proceed with building the autoencoder.

**Autoencoder Neural Network Training**

For our architecture, we run the experiments five times using distinct parameter initialisation seeds to guarantee a deterministic range of result [49]. Furthermore, we used adaptive moment estimation and initialised the weights of each network layer [19], [28].

Increasing the number of hidden units results in faster error convergence and decreases the number of detected anomalies. Once the training has converged, the trained models are used to obtain the reconstruction errors (RE) of each journal entry.

We set the anomaly threshold as 0.019, implying that a journal entry is labelled “anomalous” if one of its attributes was not reconstructed correctly or occurs very rarely.

*Preparing the Network*

# Generate random seed

np.random.seed(1234)

# import regularizers, Input, Dense and Model modules

**from** keras **import** regularizers

**from** keras.layers **import** Input, Dense

**from** keras.models **import** Model

# latent space dimension

encoding\_dim **=** 2

# input placeholder

input\_data **=** Input(shape **=** (df\_ss.shape[1],))

# encoded input

encoded **=** Dense(512, activation **=** 'relu', activity\_regularizer **=** regularizers.l1(10e-5) ) (input\_data)

encoded **=** Dense(256, activation**=**'relu')(encoded)

encoded **=** Dense(128, activation**=**'relu')(encoded)

encoded **=** Dense(64, activation**=**'relu')(encoded)

encoded **=** Dense(32, activation**=**'relu')(encoded)

encoded **=** Dense(16, activation**=**'relu')(encoded)

encoded **=** Dense(4, activation**=**'relu')(encoded)

encoded **=** Dense(encoding\_dim, activation**=**'relu')(encoded)

# decoded input

decoded **=** Dense(4, activation**=**'relu')(encoded)

decoded **=** Dense(16, activation**=**'relu')(decoded)

decoded **=** Dense(32, activation**=**'relu')(decoded)

decoded **=** Dense(64, activation**=**'relu')(decoded)

decoded **=** Dense(128, activation**=**'relu')(decoded)

decoded **=** Dense(256, activation**=**'relu')(decoded)

decoded **=** Dense(512, activation**=**'relu')(decoded)

decoded **=** Dense(df\_ss.shape[1], activation**=**'sigmoid')(decoded)

# build autoencoder model

autoencoder **=** Model (input\_data, decoded)

# build encoder for autoencoder model

encoder **=** Model (input\_data, encoded)

autoencoder.compile (optimizer **=** 'adam', loss **=** 'binary\_crossentropy')

# import EarlyStopping module

**from** keras.callbacks **import** EarlyStopping

# determine the earlystopping parameters

early\_stopping **=** EarlyStopping(monitor**=**'val\_loss', patience**=**5, verbose**=**1, mode**=**'auto')

# import ModelCheckpoint module

**from** keras.callbacks **import** ModelCheckpoint

# create checkpoint path for each epoch

filepath**=**"../models/{epoch:04}.hdf5"

checkpoint **=** ModelCheckpoint(filepath, monitor**=**'val\_loss', verbose**=**1, save\_best\_only**=False**, mode**=**'max')

# import CSVLogger module

**from** keras.callbacks **import** CSVLogger

# create csv to save loss

csv\_logger **=** CSVLogger("../models/history.csv", append**=True**)

# print dataframe shape

print(f'Training data shape {df\_ss.shape}')

Training data shape (533009, 618)

# display autoencoder architecture

autoencoder.summary()

Model: "model\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

==================================================

input\_1 (InputLayer) (None, 618) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 512) 316928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 256) 131328

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 128) 32896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 64) 8256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_5 (Dense) (None, 32) 2080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_6 (Dense) (None, 16) 528

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_7 (Dense) (None, 4) 68

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_8 (Dense) (None, 2) 10

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_9 (Dense) (None, 4) 12

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_10 (Dense) (None, 16) 80

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_11 (Dense) (None, 32) 544

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_12 (Dense) (None, 64) 2112

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_13 (Dense) (None, 128) 8320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_14 (Dense) (None, 256) 33024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_15 (Dense) (None, 512) 131584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_16 (Dense) (None, 618) 317034

==================================================

Total params: 984,804

Trainable params: 984,804

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Training*

epochs **=** 100

batch\_size **=** 64

autoencoder\_history **=** autoencoder.fit(

df\_ss,

df\_ss,

epochs **=** epochs,

batch\_size **=** batch\_size,

shuffle **=** **False**,

verbose **=** 2,

validation\_split **=** (1 **/** 3),

callbacks **=** [

early\_stopping,

checkpoint,

csv\_logger

]

).history

# plot graph for train and validation loss

fig, ax **=** plt.subplots(figsize **=** (16,9), dpi **=** 72)

ax.plot(autoencoder\_history['loss'], 'b', label**=**'Train', linewidth **=** 2)

ax.plot(autoencoder\_history['val\_loss'], 'r', label **=** 'Validation', linewidth **=** 2)

ax.set\_title('Model Loss', fontsize **=** 20)

ax.set\_ylabel('Loss (MAE)')

ax.set\_xlabel('Epoch')

ax.legend(loc**=**'best');



# create dataframe from reconstructed vectors

X\_pred\_train **=** autoencoder.predict(df\_ss)

X\_pred\_train **=** pd.DataFrame(X\_pred\_train, columns **=** df.columns)

X\_pred\_train.index **=** df.index

# create dataframe and calculate reconstruction error

autoencoder\_scored **=** pd.DataFrame(index **=** df.index)

autoencoder\_scored['loss\_mse'] **=** np.mean(np.abs(X\_pred\_train-df\_ss), axis **=** 1)

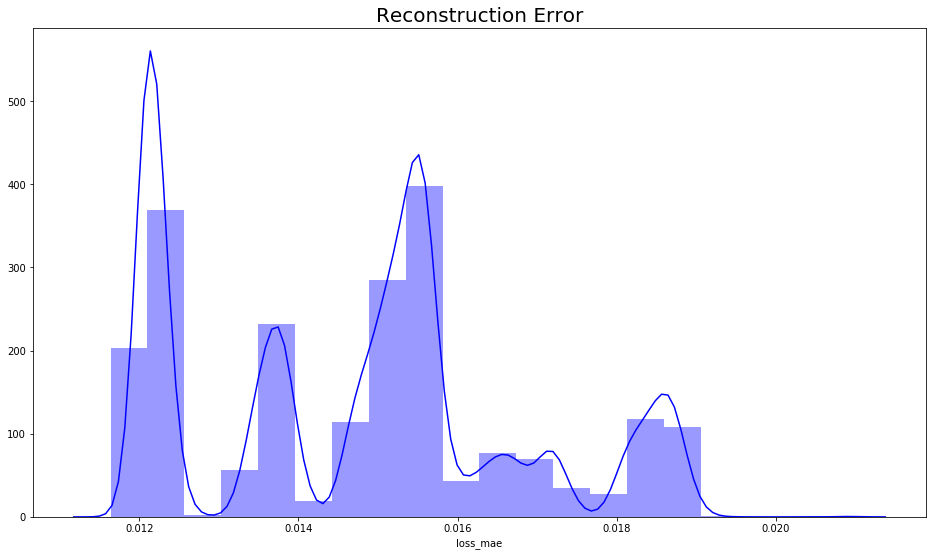
# Plot Reconstruction Error

plt.figure(figsize**=**(16,9), dpi**=**72)

plt.title('Reconstruction Error', fontsize**=**20)

sns.distplot(autoencoder\_scored['loss\_mse'], bins**=**20, kde**=True**, color**=**'blue')

plt.xticks(np.arange(0,0.05,0.005));



*Predict Anomaly*

# determine the anomaly ratio

anomaly\_ratio **=** 0.0002

# determine the number of rows to keep

head **=** int(anomaly\_ratio **\*** df\_ss.shape[0])

# determine the threshold

threshold **=** 0.019

# create dataframe from reconstructed vector

X\_pred **=** autoencoder.predict(df\_ss)

X\_pred **=** pd.DataFrame(X\_pred, columns **=** df.columns)

X\_pred.index **=** df.index

# create dataframe of reconstruction error with labels

autoencoder\_scored **=** pd.DataFrame(index **=** df.index)

autoencoder\_scored['anomaly\_score'] **=** np.mean(np.abs(X\_pred**-**df\_ss), axis**=**1)

autoencoder\_scored **=** autoencoder\_scored.sort\_values('anomaly\_score', ascending**=**False).head(head)

autoencoder\_scored['label'] **=** label

# predict anomaly

autoencoder\_scored['threshold'] **=** threshold

autoencoder\_scored['pred\_anomaly'] **=** (autoencoder\_scored.anomaly\_score **>=** autoencoder\_scored.threshold)

*True Positive*

# display true positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**) **&** (autoencoder\_scored.label **!=** 'regular')].label.value\_counts()

global 70

local 30

Name: label, dtype: int64

*False Positive (Type I Error)*

# display false positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**) **&** (autoencoder\_scored.label **==** 'regular')].label.value\_counts()

regular 6

Name: label, dtype: int64

*False Negative (Type II Error)*

# display false positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **False**) **&** (autoencoder\_scored.label **!=** 'regular')].label.value\_counts()

Series([], Name: label, dtype: int64)

# create dataframe for anomalous entries

df\_results **=** autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**)]['pred\_anomaly']

df\_results **=** pd.concat([df\_train.iloc[df\_results.index], df\_results], axis **=** 1)

**Testing the Autoencoder Neural Network**

# load the dataset into the notebook kernel

df\_test **=** pd.read\_csv('../data/test\_set\_a.csv')

# remove the "ground-truth" label

**if** 'label' **in** df\_test:

label **=** df\_test.pop('label')

# select categorical features

categ\_cols **=** df\_test.select\_dtypes([np.object]).columns

# encode categorical attributes into a binary one-hot encoded representation

df\_test\_categ\_transformed **=** pd.get\_dummies(df\_test[categ\_cols])

# select numerical features

numeric\_cols **=** df\_test.select\_dtypes([np.int64, np.float64, np.uint64]).columns

# add a small epsilon to eliminate zero values from data for log scaling

numeric\_attr **=** df\_test[numeric\_cols] **+** 1e-7

numeric\_attr **=** numeric\_attr.apply(np.log)

# normalize all numeric attributes to the range [0,1]

df\_test\_numeric\_attr **=** (numeric\_attr **-** numeric\_attr.min()) **/** (numeric\_attr.max() **-** numeric\_attr.min())

# merge categorical and numeric subsets

df\_test\_transformed **=** pd.concat([df\_test\_categ\_transformed, df\_test\_numeric\_attr], axis **=** 1)

# create a copy

df **=** df\_test\_transformed.copy()

ss **=** MinMaxScaler()

df\_ss **=** ss.fit\_transform(df)

# import regularizers, Input, Dense and Model modules

**from** keras **import** regularizers

**from** keras.layers **import** Input, Dense

**from** keras.models **import** Model

# latent space dimension

encoding\_dim **=** 2

# input placeholder

input\_data **=** Input(shape **=** (df\_ss.shape[1],))

# encoded input

encoded **=** Dense(512, activation **=** 'relu', activity\_regularizer **=** regularizers.l1(10e-5) ) (input\_data)

encoded **=** Dense(256, activation**=**'relu')(encoded)

encoded **=** Dense(128, activation**=**'relu')(encoded)

encoded **=** Dense(64, activation**=**'relu')(encoded)

encoded **=** Dense(32, activation**=**'relu')(encoded)

encoded **=** Dense(16, activation**=**'relu')(encoded)

encoded **=** Dense(4, activation**=**'relu')(encoded)

encoded **=** Dense(encoding\_dim, activation**=**'relu')(encoded)

# decoded input

decoded **=** Dense(4, activation**=**'relu')(encoded)

decoded **=** Dense(16, activation**=**'relu')(decoded)

decoded **=** Dense(32, activation**=**'relu')(decoded)

decoded **=** Dense(64, activation**=**'relu')(decoded)

decoded **=** Dense(128, activation**=**'relu')(decoded)

decoded **=** Dense(256, activation**=**'relu')(decoded)

decoded **=** Dense(512, activation**=**'relu')(decoded)

decoded **=** Dense(df\_ss.shape[1], activation**=**'sigmoid')(decoded)

# create checkpoint path for each epoch

filepath**=**"../models/test\_{epoch:04}.hdf5"

checkpoint **=** ModelCheckpoint(filepath, monitor**=**'val\_loss', verbose**=**1, save\_best\_only**=False**, mode**=**'max')

# create csv to save loss

csv\_logger **=** CSVLogger("../models/test\_history.csv", append**=True**)

epochs **=** 100

batch\_size **=** 64

autoencoder\_history **=** autoencoder.fit(

df\_ss,

df\_ss,

epochs **=** epochs,

batch\_size **=** batch\_size,

shuffle **=** **False**,

verbose **=** 2,

validation\_split **=** (1 **/** 3),

callbacks **=** [

checkpoint,

csv\_logger

]

).history

# create dataframe from reconstructed vector

X\_pred **=** autoencoder.predict(df\_ss)

X\_pred **=** pd.DataFrame(X\_pred, columns **=** df.columns)

X\_pred.index **=** df.index

# determine the number of rows to keep

head **=** int(anomaly\_ratio **\*** df\_ss.shape[0])

# create dataframe of reconstruction error with labels

autoencoder\_scored **=** pd.DataFrame(index **=** df.index)

autoencoder\_scored['anomaly\_score'] **=** np.mean(np.abs(X\_pred**-**df\_ss), axis**=**1)

autoencoder\_scored **=** autoencoder\_scored.sort\_values('anomaly\_score', ascending**=**False).head(head)

autoencoder\_scored['label'] **=** label

# predict anomaly

autoencoder\_scored['threshold'] **=** threshold

autoencoder\_scored['pred\_anomaly'] **=** (autoencoder\_scored.anomaly\_score **>=** autoencoder\_scored.threshold)

*True Positive*

# display true positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**) **&** (autoencoder\_scored.label **!=** 'regular')].label.value\_counts()

global 4

local 2

Name: label, dtype: int64

*False Positive (Type I Error)*

# display false positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**) **&** (autoencoder\_scored.label **==** 'regular')].label.value\_counts()

Series([], Name: label, dtype: int64)

*False Negative (Type II Error)*

# display false positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **False**) **&** (autoencoder\_scored.label **!=** 'regular')].label.value\_counts()

Series([], Name: label, dtype: int64)

# create dataframe for anomalous entries

df\_test\_results **=** autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**)]['pred\_anomaly']

df\_test\_results **=** pd.concat([df\_train.iloc[df\_test\_results.index], df\_test\_results], axis **=** 1)

**Testing with Clean Dataset**

# load the dataset into the notebook kernel

df\_test **=** pd.read\_csv('../data/test\_set\_c.csv')

# remove the "ground-truth" label

**if** 'label' **in** df\_test:

label **=** df\_test.pop('label')

# select categorical features

categ\_cols **=** df\_test.select\_dtypes([np.object]).columns

# encode categorical attributes into a binary one-hot encoded representation

df\_test\_categ\_transformed **=** pd.get\_dummies(df\_test[categ\_cols])

# select numerical features

numeric\_cols **=** df\_test.select\_dtypes([np.int64, np.float64, np.uint64]).columns

# add a small epsilon to eliminate zero values from data for log scaling

numeric\_attr **=** df\_test[numeric\_cols] **+** 1e-7

numeric\_attr **=** numeric\_attr.apply(np.log)

# normalize all numeric attributes to the range [0,1]

df\_test\_numeric\_attr **=** (numeric\_attr **-** numeric\_attr.min()) **/** (numeric\_attr.max() **-** numeric\_attr.min())

# merge categorical and numeric subsets

df\_test\_transformed **=** pd.concat([df\_test\_categ\_transformed, df\_test\_numeric\_attr], axis **=** 1)

# create a copy

df **=** df\_test\_transformed.copy()

ss **=** MinMaxScaler()

df\_ss **=** ss.fit\_transform(df)

# import regularizers, Input, Dense and Model modules

**from** keras **import** regularizers

**from** keras.layers **import** Input, Dense

**from** keras.models **import** Model

# latent space dimension

encoding\_dim **=** 2

# input placeholder

input\_data **=** Input(shape **=** (df\_ss.shape[1],))

# encoded input

encoded **=** Dense(512, activation **=** 'relu', activity\_regularizer **=** regularizers.l1(10e-5) ) (input\_data)

encoded **=** Dense(256, activation**=**'relu')(encoded)

encoded **=** Dense(128, activation**=**'relu')(encoded)

encoded **=** Dense(64, activation**=**'relu')(encoded)

encoded **=** Dense(32, activation**=**'relu')(encoded)

encoded **=** Dense(16, activation**=**'relu')(encoded)

encoded **=** Dense(4, activation**=**'relu')(encoded)

encoded **=** Dense(encoding\_dim, activation**=**'relu')(encoded)

# decoded input

decoded **=** Dense(4, activation**=**'relu')(encoded)

decoded **=** Dense(16, activation**=**'relu')(decoded)

decoded **=** Dense(32, activation**=**'relu')(decoded)

decoded **=** Dense(64, activation**=**'relu')(decoded)

decoded **=** Dense(128, activation**=**'relu')(decoded)

decoded **=** Dense(256, activation**=**'relu')(decoded)

decoded **=** Dense(512, activation**=**'relu')(decoded)

decoded **=** Dense(df\_ss.shape[1], activation**=**'sigmoid')(decoded)

# create checkpoint path for each epoch

filepath**=**"../models/clean\_{epoch:04}.hdf5"

checkpoint **=** ModelCheckpoint(filepath, monitor**=**'val\_loss', verbose**=**1, save\_best\_only**=False**, mode**=**'max')

# create csv to save loss

csv\_logger **=** CSVLogger("../models/clean\_history.csv", append**=True**)

epochs **=** 100

batch\_size **=** 64

autoencoder\_history **=** autoencoder.fit(

df\_ss,

df\_ss,

epochs **=** epochs,

batch\_size **=** batch\_size,

shuffle **=** **False**,

verbose **=** 2,

validation\_split **=** (1 **/** 3),

callbacks **=** [

checkpoint,

csv\_logger

]

).history

# create dataframe from reconstructed vector

X\_pred **=** autoencoder.predict(df\_ss)

X\_pred **=** pd.DataFrame(X\_pred, columns **=** df.columns)

X\_pred.index **=** df.index

# determine the number of rows to keep

head **=** int(anomaly\_ratio **\*** df\_ss.shape[0])

# create dataframe of reconstruction error with labels

autoencoder\_scored **=** pd.DataFrame(index **=** df.index)

autoencoder\_scored['anomaly\_score'] **=** np.mean(np.abs(X\_pred**-**df\_ss), axis**=**1)

autoencoder\_scored **=** autoencoder\_scored.sort\_values('anomaly\_score', ascending**=**False).head(head)

autoencoder\_scored['label'] **=** label

# predict anomaly

autoencoder\_scored['threshold'] **=** threshold

autoencoder\_scored['pred\_anomaly'] **=** (autoencoder\_scored.anomaly\_score **>=** autoencoder\_scored.threshold)

*True Positive*

# display true positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**) **&** (autoencoder\_scored.label **!=** 'regular')].label.value\_counts()

Series([], Name: label, dtype: int64)

*False Positive (Type I Error)*

# display false positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**) **&** (autoencoder\_scored.label **==** 'regular')].label.value\_counts()

regular 6

Name: label, dtype: int64

*False Negative (Type II Error)*

# display false positives

autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **False**) **&** (autoencoder\_scored.label **!=** 'regular')].label.value\_counts()

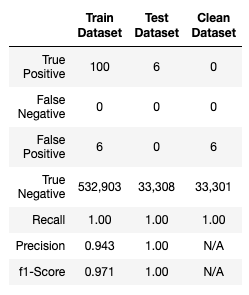
Series([], Name: label, dtype: int64)

# create dataframe for anomalous entries

df\_clean\_results **=** autoencoder\_scored[(autoencoder\_scored.pred\_anomaly **==** **True**)]['pred\_anomaly']

df\_clean\_results **=** pd.concat([df\_train.iloc[df\_clean\_results.index], df\_clean\_results], axis **=** 1)

**Overview of Autoencoder Results**

****

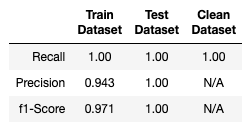
1. **Experimental Results**

This section describes the results of our evaluation. Upon successful training we evaluated the proposed scoring according to two criteria:

* Are the trained autoencoder architectures capable of learning a model of the regular journal entries and thereby detect the anomalies?
* Are the detected anomalies “suspicious” enough to be followed up by accountants or forensic examiners?

**Quantitative Evaluation**

To quantitatively evaluate the effectiveness of the proposed approach, a range of evaluation metrics including recall, precision and F1-Score. The choice of F1-Score is to account for the highly unbalanced anomalous versus non-anomalous class distribution of the datasets.



As seen from the table above, the metrics are high for all datasets. Note that the clean dataset has no anomalies and therefore, unable to provide a precision and F1-Score.

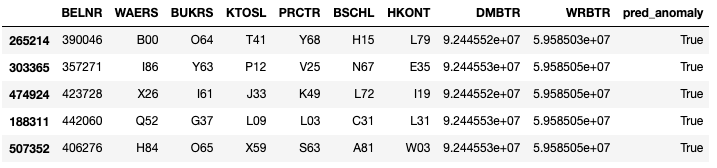
Overall, the deep autoencoder neural network model satisfies our quantitative evaluation.

**Qualitative Evaluation**

To qualitatively evaluate the character of the detected anomalies we need to review all journal entries detected by the architecture.

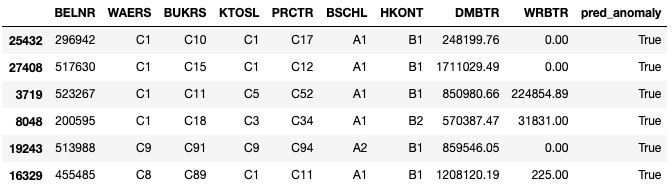
*Train Dataset*

df\_results.head()



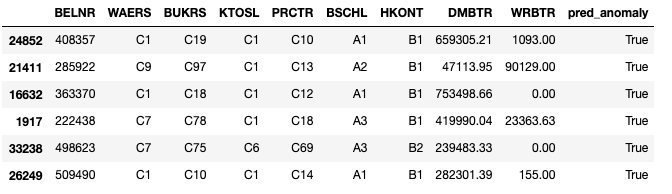
*Test Dataset*

df\_test\_results.head()



*Clean Dataset*

df\_clean\_results.head()



The majority of anomalies probably correspond to journal entries that exhibit one or two rare attribute values. This could be:

* posting errors due to wrongly used general ledger accounts;
* journal entries of unusual document types containing extremely infrequent tax codes;
* incomplete journal entries exhibiting missing currency information;
* shipments to customers that are invoiced in different than the usual currency;
* products send to a regular client but were booked to another company code;
* postings that exhibit an unusual large time lag between document date and posting date;
* irregular rental payments that slightly deviate from ordinary payments.

All of the above indicated a weak control environment around certain business processes of the investigated organization. As the dataset is anonymised as received we are unable to ascertain for a fact whether these entries were truly anomalous, other than the label provided. It requires domain knowledge and the actual dataset, both of which the author does not have.

**Baseline Evaluation**

There are various baseline models for us to consider, such as PCA, kMeans, One-Class Support Vector Machine, Local-Outlier Factor and DBSCAN [11], [12], [18], [36], [37], [42]. However, for simplicity of this report, we will only consider kMeans and Local-Outlier Factor. For purposes of evaluation, our metrics will be on recall, accuracy and F1-score.

*kMeans*

**“The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimising a criterion known as the inertia or within-cluster sum-of-squares.”**

*Source: SciKit-Learn*

# import kMeans module

**from** sklearn.cluster **import** KMeans

# initialise and fit kMeans

pca\_kmeans **=** KMeans(n\_clusters**=**2)

pca\_kmeans.fit(df\_pca)

KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300, n\_clusters=2, n\_init=10, n\_jobs=None, precompute\_distances='auto', random\_state=None, tol=0.0001, verbose=0)

# predict labels

pred\_pca\_kmeans **=** pca\_kmeans.predict(df\_pca)

# import confusionmatrix module

**from** sklearn.metrics **import** confusion\_matrix

# display the confusion matrix

cmat **=** confusion\_matrix(label\_int.astype(int), pred\_pca\_kmeans)

print(f'TN - True Negative {cmat[0,0]}')

print(f'FP - False Positive {cmat[0,1]}')

print(f'FN - False Negative {cmat[1,0]}')

print(f'TP - True Positive {cmat[1,1]}')

print(f'Accuracy Rate: {np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat))}')

print(f'Misclassification Rate: {np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))}')

TN - True Negative 202666

FP - False Positive 330243

FN - False Negative 100

TP - True Positive 0

Accuracy Rate: 0.380229977354979

Misclassification Rate: 0.619770022645021

# import classification module

**from** sklearn.metrics **import** classification\_report

# display the classification report

print(classification\_report(label\_int, pred\_pca\_kmeans, digits**=**5))

precision recall f1-score support

0 0.99951 0.38030 0.55097 532909

1 0.00000 0.00000 0.00000 100

accuracy 0.38023 533009

macro avg 0.49975 0.19015 0.27548 533009

weighted avg 0.99932 0.38023 0.55086 533009

*Local Outlier Factor*

**“The Local Outlier Factor (LOF) algorithm computes a score reflecting the degree of abnormality of the observations. It measures the local density deviation of a given data point with respect to its neighbours. The idea is to detect the samples that have a substantially lower density  
 than their neighbours.”**

*Source: SciKit-Learn*

# import localoutlierfactor module

**from** sklearn.neighbors **import** LocalOutlierFactor

# initialise and fit localoutlierfactor

pca\_lof **=** LocalOutlierFactor(n\_neighbors**=**30, contamination**=**0.0002)

pca\_lof.fit(df\_pca)

LocalOutlierFactor(algorithm='auto', contamination=0.0002, leaf\_size=30, metric='minkowski', metric\_params=None, n\_jobs=None, n\_neighbors=30, novelty=False, p=2)

# predict labels

pred\_pca\_lof **=** pca\_lof.fit\_predict(df\_pca)

pred\_pca\_lof **=** pd.Series(pred\_pca\_lof).replace([**-**1,1],[1,0])

# display the confusion matrix

cmat **=** confusion\_matrix(label\_int.astype(int), pred\_pca\_kmeans)

print(f'TN - True Negative {cmat[0,0]}')

print(f'FP - False Positive {cmat[0,1]}')

print(f'FN - False Negative {cmat[1,0]}')

print(f'TP - True Positive {cmat[1,1]}')

print(f'Accuracy Rate: {np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat))}')

print(f'Misclassification Rate: {np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))}')

TN - True Negative 532817

FP - False Positive 92

FN - False Negative 85

TP - True Positive 15

Accuracy Rate: 0.9996679230557083

Misclassification Rate: 0.00033207694429174744

# display the classification report

print(classification\_report(label\_int, pred\_pca\_kmeans, digits**=**5))

precision recall f1-score support

0 0.99984 0.99983 0.99983 532909

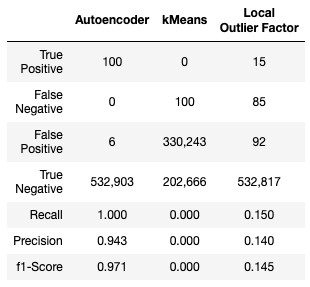
1 0.14019 0.15000 0.14493 100

accuracy 0.99967 533009

macro avg 0.57001 0.57491 0.57238 533009

weighted avg 0.99968 0.99967 0.99967 533009

*Comparison between Autoencoder and Baseline Models*



As seen above, our autoencoder architecture is performing well above the baseline models.

1. **Conclusion and Future Work**

In this work we presented a deep learning-based approach for the detection of anomalous journal entries in large scaled accounting data. Our empirical evaluation demonstrates that the reconstruction error of deep autoencoder networks can be used as a highly adaptive anomaly assessment of journal entries. In our experiments we achieved a superior F1-Score of 0.971 in the train dataset compared to regular machine learning methods.

In this architecture, we looked only at the reconstruction errors. We could also perform clustering at the latent space and detect anomalous entries. This would be an area for future development. A python script is also made available to run the autoencoder architecture on your own dataset. A graphical user interface (GUI) would be something for the author to consider in the near future.

Given the tremendous amount of journal entries recorded by organisations annually, an automated and high precision detection of accounting anomalies can save auditors considerable time and decrease the risk of fraudulent financial statements.

Appendix

The code we used to train and evaluate our models is available at: https://github.com/AmirYunus/GA\_DSI\_Capstone

Acknowledgment

This report is heavily influenced by Marco Schreyer and Timur Sattarov in their paper "Detection of Anomalies in Large-Scale Accounting Data using Deep Autoencoder Networks". They can be reached at marco.schreyer@dfki.de and sattarov.timur@pwc.com.

The author of this report is indebted to their work in advancements in detecting accounting anomalies. The author gives his thanks and any credit should be due to them.

References

1. ACFE: Report to the Nations on Occupational Fraud and Abuse, The 2016 Global Fraud Study. Association of Certified Fraud Examiners (ACFE) (2016), https://s3-us-west-2.amazonaws.com/acfepublic/2016-report-to-the-nations.pdf
2. AICPA: Consideration of Fraud in a Financial Statement Audit. American Institute of Certified Public Accountants (AICPA) (2002), https://www.aicpa.org/Research/Standards/AuditAttest/DownloadableDocuments/AU-00316.pdf
3. Amani, F.A., Fadlalla, A.M.: Data mining applications in accounting: A review of the literature and organizing framework. International Journal of Accounting Information Systems 24, 32-58 (2017)
4. An, J.: Variational Autoencoder based Anomaly Detection using Reconstruction Probability. Tech. rep. (2015)
5. Andrews, J.T.A., Morton, E.J., Griffin, L.D.: Detecting Anomalous Data Using Auto-Encoders. International Journal of Machine Learning and Computing 6(1), 21-26 (2016)
6. Argyrou, A.: Auditing Journal Entries Using Self-Organizing Map. In: Proceedings of the Eighteenth Americas Conference on Information Systems (AMCIS). pp. 1-10. No. 16, Seattle, Washington (2012)
7. Argyrou, A.: Auditing Journal Entries Using Extreme Vale Theory. Proceedings of the 21st European Conference on Information Systems 1(2013) (2013)
8. Bay,S.,Kumaraswamy,K.,Anderle,M.G.,Kumar,R.,Steier,D.M.,Blvd,A.,Jose, S.: Large Scale Detection of Irregularities in Accounting Data. In: Data Mining, 2006. ICDM'06. Sixth International Conference on. pp. 75-86. IEEE (2006)
9. Benford, F.: The Law of Anomalous Numbers. Proceedings of the American Philosophical Society 78(4), 551-572 (1938)
10. Bengio, Y., Yao, L., Alain, G., Vincent, P.: Generalized denoising auto-encoders as generative models. In: Advances in Neural Information Processing Systems, pp. 899-907 (2013)
11. Breunig,M.M.,Kriegel,H.P.,Ng,R.T.,Sander,J.:LOF:IdentifyingDensity-Based Local Outliers. In: Proceedings of the 2000 Acm Sigmod International Conference on Management of Data. pp. 1-12 (2000)
12. Campello, R.J.G.B., Moulavi, D., Sander, J.: Density-Based Clustering Based on Hierarchical Density Estimates. Tech. rep. (2013)
13. Cozzolino, D., Verdoliva, L.: Single-image splicing localization through autoencoder-based anomaly detection. In: 8th IEEE International Workshop on Information Forensics and Security, WIFS 2016. pp. 1-6 (2017)
14. Das, K., Schneider, J.: Detecting anomalous records in categorical datasets. ACM SIGKDD International conference on Knowledge discovery and data mining pp. 220-229 (2007)
15. Dau, H.A., Ciesielski, V., Song, A.: Anomaly Detection Using Replicator Neural Networks Trained on Examples of One Class. In: Asia-Pacific Conference on Simulated Evolution and Learning. pp. 311-322 (2014)
16. D'Avino, D., Cozzolino, D., Poggi, G., Verdoliva, L.: Autoencoder with recurrent neural networks for video forgery detection. arXiv preprint (March), 1-8 (2017), https://arxiv.org/abs/1708.08754
17. Debreceny, R.S., Gray, G.L.: Data mining journal entries for fraud detection: An exploratory study. International Journal of Accounting Information Systems 11(3), 157-181 (2010)
18. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining. pp. 226-231 (1996)
19. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 9, 249-256 (2010)
20. Hawkins, S., He, H., Williams, G., Baxter, R.: Outlier Detection Using Replicator Neural Networks. In: International Conference on Data Warehousing and Knowledge Discovery. pp. 170-180. No. September, Springer Berlin Heidelberg (2002)
21. Hinton, G.E., Salakhutdinov, R.R.: Reducing the Dimensionality of Data with Neural Networks. Science 313(5786), 504-507 (2006)
22. IFAC: International Standards on Auditing 240, The Auditor's Responsibilities Relating to Fraud in an Audit of Financial Statements. International Federation of Accountants (IFAC) (2009), http://www.ifac.org/system/files/downloads/a012-2010-iaasb-handbook-isa-240.pdf
23. Islam, A.K., Corney, M., Mohay, G., Clark, A., Bracher, S., Raub, T., Flegel, U.: Fraud detection in ERP systems using Scenario matching. IFIP Advances in Information and Communication Technology 330, 112-123 (2010)
24. Jans, M., Lybaert, N., Vanhoof, K.: Internal fraud risk reduction: Results of a data mining case study. International Journal of Accounting Information Systems 11(1), 17-41 (2010)
25. Jans, M., Van Der Werf, J.M., Lybaert, N., Vanhoof, K.: A business process mining application for internal transaction fraud mitigation. Expert Systems with Applications 38(10), 13351-13359 (2011)
26. Khan, R., Corney, M., Clark, A., Mohay, G.: Transaction Mining for Fraud Detection in ERP Systems. Industrial Engineering and Management Systems 9(2), 141-156 (2010)
27. Khan, R., Corney, M.: A role mining inspired approach to representing user behaviour in ERP systems. In: Proceedings of The 10th Asia Pacific Industrial Engineering and Management Systems Conference. pp. 2541-2552. No. December (2009)
28. Kingma,D.P.,Ba,J.:Adam:AMethodforStochasticOptimization.arXivpreprint pp. 1-15 (2014), http://arxiv.org/abs/1412.6980
29. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. Advances In Neural Information Processing Systems pp. 1-9 (2012)
30. Lauly, S., Larochelle, H., Khapra, M.: An Autoencoder Approach to Learning Bilingual Word Representations. In: Advances in Neural Information Processing Systems. pp. 1853-1861 (2014)
31. LeCun, Y., Bengio, Y., Hinton, G., Y., L., Y., B., G., H.: Deep learning. Nature 521(7553), 436-444 (2015)
32. Markovitch, S., Willmott, P.: Accelerating the digitization of business processes. McKinsey & Company pp. 1-5 (2014)
33. McGlohon, M., Bay, S., Anderle, M.G.M., Steier, D.M., Faloutsos, C.: SNARE: A Link Analytic System for Graph Labeling and Risk Detection. Kdd-09: 15Th Acm Sigkdd Conference on Knowledge Discovery and Data Mining pp. 1265-1273 (2009)
34. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 pp. 1-12 (2013)
35. Paula, E.L., Ladeira, M., Carvalho, R.N., Marzaga ÃÉo, T.: Deep learning anomaly detection as suppor fraud investigation in Brazilian exports and anti-money laundering. In: Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016. pp. 954-960 (2017)
36. Pearson, K.: LIII. On lines and planes of closest fit to systems of points in space. Philosophical Magazine Series 6 2(11), 559-572 (1901). https://doi.org/10.1080/14786440109462720
37. Pedregosa, F., Varoquaux, G.: Scikit-learn: Machine learning in Python, vol. 12 (2011). https://doi.org/10.1007/s13398-014-0173-7.2, http://dl.acm.org/citation.cfm?id=2078195
38. Pedrosa, I., Costa, C.J.: New trends on CAATTs: what are the Chartered Accountants' new challenges? ISDOC '14 Proceedings of the International Conference on Information Systems and Design of Communication, May 1617, 2014, Lisbon, Portugal pp. 138-142 (2014)
39. PwC: Adjusting the Lens on Economic Crime, The Global Economic Crime Survey 2016. PricewaterhouseCoopers LLP (2016), https://www.pwc.com/gx/en/economic-crime-survey/pdf/GlobalEconomicCrimeSurvey2016.pdf
40. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representation by error propagation. In: Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, pp. 318-362 (1986)
41. SAP: SAP Global Corporate Affairs, Corporate Factsheet 2017 (2017), https://www.sap.com/corporate/de/documents/2017/04/4666ecdd-b67c-0010-82c7-eda71af511fa.html
42. Scholkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J.: Support Vector Method for Novelty Detection. In: Advances in neural information processing systems. vol. 12, pp. 582-588 (2000)
43. Seow, Poh-Sun; Sun, Gary; Themin, S.: Data Mining Journal Entries for Fraud Detection : a Replication of Debreceny and Gray ' S ( 2010 ). Journal of Forensic Investigative Accounting 3(8), 501-514 (2016)
44. Singleton, T., Singleton, A.J.: Fraud auditing and forensic accounting. John Wiley & Sons (2006)
45. T ÃÅoth, L., Gosztolya, G.: Replicator Neural Networks for Outlier Modeling in Segmental Speech Recognition. Advances in Neural NetworksISNN 2004 pp. 996-1001 (2004)
46. Wang,S.:Acomprehensivesurveyofdatamining-basedaccounting-frauddetection research. In: 2010 International Conference on Intelligent Computation Technology and Automation, ICICTA 2010. vol. 1, pp. 50-53 (2010)
47. Wells, J.T.: Corporate Fraud Handbook: Prevention and Detection. John Wiley & Sons (2017)
48. Williams, G., Baxter, R.: A comparative study of RNN for outlier detection in data mining. IEEE International Conference on Data Mining (December 2002), 1-16 (2002)
49. Xu, B., Wang, N., Chen, T., Li, M.: Empirical Evaluation of Rectified Activations in Convolution Network. ICML Deep Learning Workshop pp. 1-5 (2015)
50. Zhai, S., Cheng, Y., Lu, W., Zhang, Z.: Deep Structured Energy Based Models for Anomaly Detection. In: International Conference on Machine Learning. vol. 48, pp. 1100-1109 (2016)
51. Zhou, C.: Anomaly Detection with Robust Deep Auto-encoders. In: KDD '17 Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 665-674 (2017)