Milestone 3 Report

Chengming Xie (cx2234), Hongshan Li (hl3353)

*Topic : **Medical Insurance Fraud Detection** (Group 20)*

Date: 4/16/2020

1. Abstract of our project

Machine learning for data-driven diagnosis has been actively studied in medicine to provide better healthcare. Supporting analysis of a patient cohort similar to a patient under treatment will help clinicians to make better decisions and also help the insurance providers to detect the possible fraud insurance claim. However, such analysis is difficult due to the characteristics of medical records: high dimensionality, irregularity in time,and sparsity. To address this challenge, we deal with these medical records as event sequences in natural language applications. By milestone 2, we have calculated the similarity between each patient's historical medical records. In the milestone 3, our goal is to detect anomaly among patients based on two approaches for calculating the similarity of medical records with different lengths: **Dynamic time warping** (***DTW***) and **LSTM-based Variational AutoEncoder** (***VAE***), which are then followed by Local Outlier Factor and other anomaly detection analysis separately.

2. Overview of Milestone 3 Progress

In this project we used detailed and comprehensive patients' data from MIMIC-III dataset which contains 26 tables and medical records for 61000+ patients. After milestone 2, we have generated distance matrices for patients who have respiratory system related and injury & poisoning-related diseases. In the Milestone 3 implementation, we have implemented two methods to detect the possible anomalies among the patients based on their historical medical records.

The first approach we have implemented is to leverage the *Seq2Seq* architecture with two recurrent neural networks (*LSTM*) and a module that performs the variational approximation.

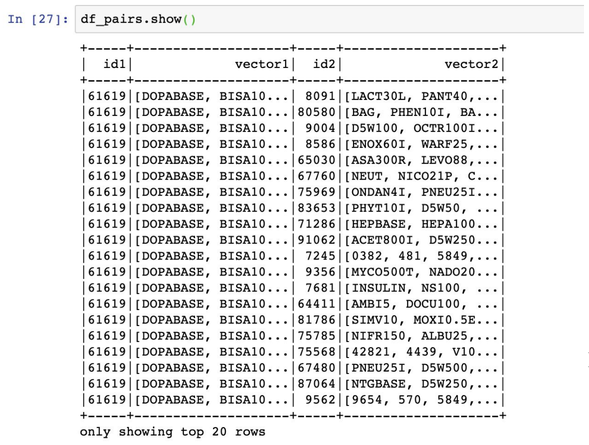
The Variational Autoencoder (*VAE*) is a generative model and can be thought of as a normal autoencoder combined with the variational inference. It encodes data to latent variables and then decodes latent variables to reconstruct the data. Intuitively speaking, we are trying to project the event sequence data of each patient into their latent representation such that we are able to leverage the *LOF* technique to detect the outliers in the latent space.

Secondly, based on the results of Milestone 2, we use a Dynamic Time Warping algorithm to calculate the accumulated distance (cosine or Euclidean, i.e. we find their results are similar) along the optimal warping path between patients' event sequence. The similar patient, the closer the absolute DTW distance. DTW allows us to align event sequence data with variable length and compute similarity between patients. We are then able to calculate the "outlier" score for each patient based on the anomaly detection algorithms (ex. Local Outlier Factor, Isolation Forest.)

3. Challenges and state-of-art solution

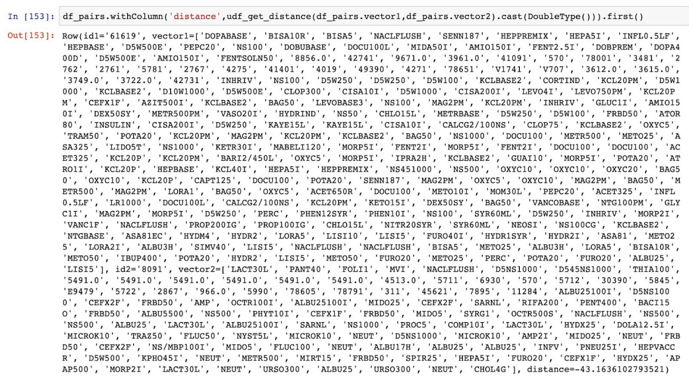
3.1 Challenge of DTW-distance based approach

The first challenge is the time-consuming DTW pairwise distance matrix computation in our second approach. We tried to employ our second approach on the injury & poisoning disease related patients. Since we have 2757 patients in this sub data set, we need to generate a 2757*2757 dimensions distance matrix. In other words, we have to calculate the DTW distance for each pair of patients, a $2757 * 2757$ times of DTW algorithm computation. To tackle the time complexity of our algorithm implementation, we used Spark DataFrame UDF function to calculate the pairwise distance matrix between patients while optimizing the running time efficiency using RDD technique.



```
In [27]: df_pairs.show()

+-----+--------------------+-----+--------------------+
|  id1|             vector1|  id2|             vector2|
+-----+--------------------+-----+--------------------+
|61619|[DOPABASE, BISA10...| 8091|[LACT30L, PANT40,...|
|61619|[DOPABASE, BISA10...|80580|[BAG, PHEN10I, BA...|
|61619|[DOPABASE, BISA10...| 9004|[D5W100, OCTR100I...|
|61619|[DOPABASE, BISA10...| 8586|[ENOX60I, WARF25,...|
|61619|[DOPABASE, BISA10...|65030|[ASA300R, LEVO88,...|
|61619|[DOPABASE, BISA10...|67760|[NEUT, NICO21P, C...|
|61619|[DOPABASE, BISA10...|75969|[ONDAN4I, PNEU25I...|
|61619|[DOPABASE, BISA10...|83653|[PHYT10I, D5W50, ...|
|61619|[DOPABASE, BISA10...|71286|[HEPBASE, HEPA100...|
|61619|[DOPABASE, BISA10...|91062|[ACET800I, D5W250...|
|61619|[DOPABASE, BISA10...| 7245|[0382, 481, 5849,...|
|61619|[DOPABASE, BISA10...| 9356|[MYCO500T, NADO20...|
|61619|[DOPABASE, BISA10...| 7681|[INSULIN, NS100, ...|
|61619|[DOPABASE, BISA10...|64411|[AMBI5, DOCU100, ...|
|61619|[DOPABASE, BISA10...|81786|[SIMV10, MOXI0.5E...|
|61619|[DOPABASE, BISA10...|75785|[NIFR150, ALBU25,...|
|61619|[DOPABASE, BISA10...|75568|[42821, 4439, V10...|
|61619|[DOPABASE, BISA10...|67480|[PNEU25I, D5W500,...|
|61619|[DOPABASE, BISA10...|87064|[NTGBASE, D5W250,...|
|61619|[DOPABASE, BISA10...| 9562|[9654, 570, 5849,...|
+-----+--------------------+-----+--------------------+
only showing top 20 rows
```

Figure I. Spark Dataframe version of distance matrix

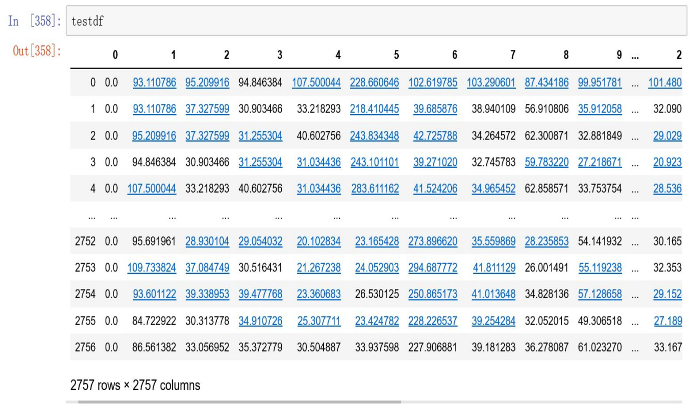


```
In [153]: df_pairs.withColumn('distance',udf_get_distance(df_pairs.vector1,df_pairs.vector2).cast(DoubleType())).first()

Out[153]: Row(id1='61619', vector1=['DOPABASE', 'BISA10R', 'BISA5', 'NACLFLUSH', 'SENN187', 'HEPPREMIX', 'HEPA5I', 'INFL0.5LF',
          'HEPBASE', 'D5W500E', 'PEPC20', 'NS100', 'DOBUBASE', 'DOCU100L', 'MIDA50I', 'AMIO150I', 'FENT2.5I', 'DOBPREM', 'DOPA4
          00D', 'D5W500E', 'AMIO150I', 'FENTSOLN50', '8856.0', '42741', '9671.0', '3961.0', '41091', '570', '78001', '3481', '2
          762', '2761', '5781', '2767', '4275', '41401', '4019', '49390', '4271', '78651', 'V1741', 'V707', '3612.0', '3615.0',
          '3749.0', '3722.0', '42731', 'INHRIV', 'NS100', 'D5W250', 'D5W250', 'D5W100', 'KCLBASE2', 'CORTIND', 'KCL20PM', 'D5W1
          000', 'KCLBASE2', 'D10W1000', 'D5W500E', 'CLOP300', 'CISA10I', 'D5W1000', 'CISA200I', 'LEVO4I', 'LEVO750PM', 'KCL20P
          M', 'CEFX1F', 'AZIT500I', 'KCLBASE2', 'BAG50', 'LEVOBASE3', 'NS100', 'MAG2PM', 'KCL20PM', 'INHRIV', 'GLUC1I', 'AMIO15
          0I', 'DEX50SY', 'METR500PM', 'VASO20I', 'HYDRIND', 'NS50', 'CHLO15L', 'METRBASE', 'D5W250', 'D5W100', 'FRBD50', 'ATOR
          80', 'INSULIN', 'CISA200I', 'D5W250', 'KAYE15L', 'KAYE15L', 'CISA10I', 'CALCG2/100NS', 'CLOP75', 'KCLBASE2', 'OXYC5',
          'TRAM50', 'POTA20', 'KCL20PM', 'MAG2PM', 'KCL20PM', 'KCLBASE2', 'BAG50', 'NS1000', 'DOCU100', 'METR500', 'METO25', 'A
          SA325', 'LIDO5T', 'NS1000', 'KETR30I', 'MABELI120', 'MORP5I', 'FENT2I', 'MORP5I', 'FENT2I', 'DOCU100', 'DOCU100', 'AC
          ET325', 'KCL20P', 'KCL20PM', 'BARI2/450L', 'OXYC5', 'MORP5I', 'IPRA2H', 'KCLBASE2', 'GUAI10', 'MORP5I', 'POTA20', 'AT
          RO1I', 'KCL20P', 'HEPBASE', 'KCL40I', 'HEPA5I', 'HEPPREMIX', 'NS451000', 'NS500', 'OXYC10', 'OXYC10', 'OXYC20', 'BAG5
          0', 'OXYC10', 'KCL20P', 'CAPT125', 'DOCU100', 'POTA20', 'SENN187', 'MAG2PM', 'OXYC5', 'OXYC10', 'MAG2PM', 'BAG50', 'M
          ETR500', 'MAG2PM', 'LORA1', 'BAG50', 'OXYC5', 'ACET650R', 'DOCU100', 'METO10I', 'MOM30L', 'PEPC20', 'ACET325', 'INFL
          0.5LF', 'LR1000', 'DOCU100L', 'CALCG2/100NS', 'KCL20PM', 'KETO15I', 'DEX50SY', 'BAG50', 'VANCOBASE', 'NTG100PM', 'GLY
          C1I', 'MAG2PM', 'MORP5I', 'D5W250', 'PERC', 'PHEN12SYR', 'PHEN10I', 'NS100', 'SYR60ML', 'D5W250', 'INHRIV', 'MORP2I',
          'VANC1F', 'NACLFLUSH', 'PROP200IG', 'PROP100IG', 'CHLO15L', 'NITR20SYR', 'SYR60ML', 'NEOSI', 'NS100CG', 'KCLBASE2',
          'NTGBASE', 'ASA81EC', 'HYDM4', 'HYDR2', 'LORA5', 'LISI10', 'LISI5', 'FURO40I', 'HYDR1SYR', 'HYDR2I', 'ASA81', 'METO2
          5', 'LORA2I', 'ALBU3H', 'SIMV40', 'LISI5', 'NACLFLUSH', 'NACLFLUSH', 'BISA5', 'METO25', 'ALBU3H', 'LORA5', 'BISA10R',
          'METO50', 'IBUP400', 'POTA20', 'HYDR2', 'LISI5', 'METO50', 'FURO20', 'METO25', 'PERC', 'POTA20', 'FURO20', 'ALBU25',
          'LISI5'], id2='8091', vector2=['LACT30L', 'PANT40', 'FOLI1', 'MVI', 'NACLFLUSH', 'D5NS1000', 'D545NS1000', 'THIA100',
          '5491.0', '5491.0', '5491.0', '5491.0', '5491.0', '5491.0', '4513.0', '5711', '6930', '570', '5712', '30390', '5845',
          'E9479', '5722', '2867', '966.0', '5990', '78605', '78791', '311', '45621', '7895', '11284', 'ALBU25100I', 'D5NS100
          0', 'CEFX2F', 'FRBD50', 'AMP', 'OCTR100I', 'ALBU25100I', 'MIDO25', 'CEFX2F', 'SARNL', 'RIFA200', 'PENT400', 'BACI15
          O', 'FRBD50', 'ALBU5500', 'NS500', 'PHYT10I', 'CEFX1F', 'FRBD50', 'MIDO5', 'SYRG1', 'OCTR500S', 'NACLFLUSH', 'NS500',
          'NS500', 'ALBU25', 'LACT30L', 'ALBU25100I', 'SARNL', 'NS1000', 'PROC5', 'COMP10I', 'LACT30L', 'HYDX25', 'DOLA12.5I',
          'MICROK10', 'TRAZ50', 'FLUC50', 'NYST5L', 'MICROK10', 'NEUT', 'D5NS1000', 'MICROK10', 'AMP2I', 'MIDO25', 'NEUT', 'FRB
          D50', 'CEFX2F', 'NS/MBP100I', 'MIDO5', 'FLUC100', 'NEUT', 'ALBU17H', 'ALBU25', 'ALBU25', 'INFV', 'PNEU25I', 'HEPVACC
          R', 'D5W500', 'KPHO45I', 'NEUT', 'METR500', 'MIRT15', 'FRBD50', 'SPIR25', 'HEPA5I', 'FURO20', 'CEFX1F', 'HYDX25', 'AP
          AP500', 'MORP2I', 'LACT30L', 'NEUT', 'URSO300', 'ALBU25', 'URSO300', 'NEUT', 'CHOL4G'], distance=-43.1636102793521)
```

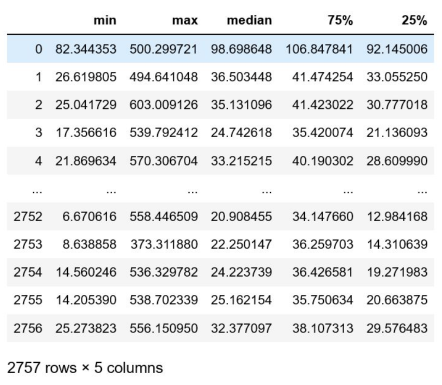
The second challenge is high dimensionality of our dtw distance matrix, as you can see above, the shape of this matrix is 2754*2754 (num of patients * num of patients). Methods like LOF perform poorly in high dimensional data. To address this problem, we found not all the pairwise distances are significant and necessary in model training, so we use statistical summary as our features.

For instance, we use aggregation functions to obtain the quantile distance of each patient and successfully reduce the number dimension from 2754 to 5 while retaining most of the information. By comparing the result, we find the results are almost identical but we significantly reduce the running time. However, by looking at the graph, we find that there are several red points among green clusters and we call them local outliers. We realized that we can not simply use dtw distance as lof feature because when k is larger it possible to miss local outliers (Here K=100).

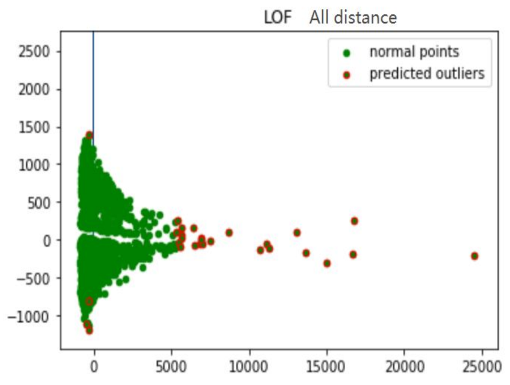
Figure II. Output before (left) and after processing (right)

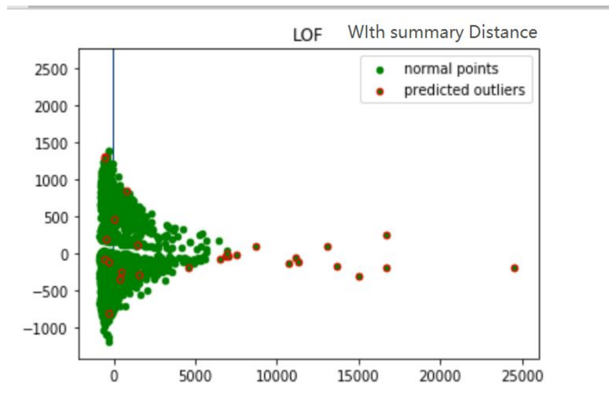


```
In [358]: testdf

Out[358]:
         0           1          2          3           4           5           6           7          8          9  ...       2
   0   0.0   93.110786  95.209916  94.846384  107.500044  228.660646  102.619785  103.290601  87.434186  99.951781  ...  101.480
   1   0.0   93.110786  37.327599  30.903466   33.218293  218.410445   39.685876   38.940109  56.910806  35.912058  ...   32.090
   2   0.0   95.209916  37.327599  31.255304   40.602756  243.834348   42.725788   34.264572  62.300871  32.881849  ...   29.029
   3   0.0   94.846384  30.903466  31.255304   31.034436  243.101101   39.271020   32.745783  59.783220  27.218671  ...   20.923
   4   0.0  107.500044  33.218293  40.602756   31.034436  283.611162   41.524206   34.965452  62.858571  33.753754  ...   28.536
 ...   ...         ...        ...        ...         ...         ...         ...         ...        ...        ...  ...
2752   0.0   95.691961  28.930104  29.054032   20.102834   23.165428  273.896620   35.559869  28.235853  54.141932  ...   30.165
2753   0.0  109.733824  37.084749  30.516431   21.267238   24.052903  294.687772   41.811129  26.001491  55.119238  ...   32.353
2754   0.0   93.601122  39.338953  39.477768   23.360683   26.530125  250.865173   41.013648  34.828136  57.128658  ...   29.152
2755   0.0   84.722922  30.313778  34.910726   25.307711   23.424782  228.226537   39.254284  32.052015  49.306518  ...   27.189
2756   0.0   86.561382  33.056952  35.372779   30.504887   33.937598  227.906881   39.181283  36.278087  61.023270  ...   33.167

2757 rows × 2757 columns
```



```
            min         max      median         75%        25%
   0  82.344353  500.299721   98.698648  106.847841  92.145006
   1  26.619805  494.641048   36.503448   41.474254  33.055250
   2  25.041729  603.009126   35.131096   41.423022  30.777018
   3  17.356616  539.792412   24.742618   35.420074  21.136093
   4  21.869634  570.306704   33.215215   40.190302  28.609990
 ...        ...         ...         ...         ...        ...
2752   6.670616  558.446509   20.908455   34.147660  12.984168
2753   8.638858  373.311880   22.250147   36.259703  14.310639
2754  14.560246  536.329782   24.223739   36.426581  19.271983
2755  14.205390  538.702339   25.162154   35.750634  20.663875
2756  25.273823  556.150950   32.377097   38.107313  29.576483

2757 rows × 5 columns
```





3.2 Challenges and Motivations behind LSTM-based VAE

One biggest challenge of this project is the development of an unsupervised anomaly detection that can efficiently handle the complex temporal structure of event sequences. The methods we use in our second approach (i.e., computing DTW pairwise distances of patients and using traditional anomaly detection models such as LOF and Isolation Forest) make more sense intuitively but fail to capture complex temporal structures in the data.

Under such circumstances, there are several recent papers regarding time series anomaly detection using LSTM-based VAE (eg. *Sequential VAE-LSTM for Anomaly Detection on Time Series, Run-Qing Chen,2019; Visual Anomaly Detection in Event Sequence Data, Shunan Guo,2019* , etc.)

As Shunan Guo stated in their paper, "VAE uses a probabilistic encoder for modeling the distribution of the latent variables...such probabilities give more principled criteria for identifying anomalies and do note require model-specific thresholds. As a result, VAE can better facilitate objective judgements for deciding the boundary of anomalous sequences compared to other unsupervised algorithms."

Thus, based on the papers, we are trying to implement LSTM-based VAE on our patients' event sequence data as our first approach to implement anomaly detection.

During the literature research, we found someone using sequence labeling and pretraining embedding layers instead of multi-hot encoding. Then we can pass as labels the sequences of multi-hot encoding to calculate the weighted categorical cross entropy loss. We will implement this in the rest of the project implementation.

4. Implementation and progress of project

The first approach is to use LSTM-based Variational AutoEncoder to extract sequential patterns from patients' event sequence data and project them into a latent space. Then we implemented Local Outlier Factor analysis to detect the anomalies using the latent representations of each patient's event sequence.

The second approach is to use the results from our milestone 2 implementation. Specifically speaking, we have computed a DTW distance matrix among all patients and then we can use the principles of some anomaly detection algorithms to compute the "anomaly" score for each patient.

4.1. First Approach: LSTM-based Variational AutoEncoder:

The VAE encoder and the VAE decoder are using LSTM-based Recurrent neural networks to better extract sequential patterns from event sequence data. Our LSTM-based VAE model is trying to replicate the one in the paper, *Visual Anomaly Detection in Event Sequence Data, Shunan Guo, 2019*

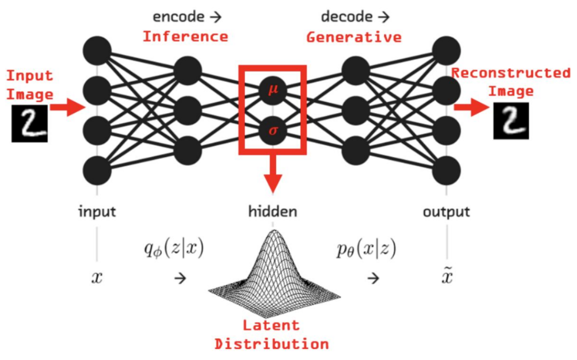


Figure III.: Architecture of VAE (*How to generate images using autoencoders,* Sergios Karagiannakos)

The VAE encoder is trained to project the input sequence data, multi-hot encoding vectors of size (training_size, time_steps, vocab_size), into a latent feature vector that describes a sequential distribution of events occuring in the sequence. Each coordinate represents an event type and is marked 1 if such event type happens in a certain time step and 0 otherwise.

Suppose $h_{enc} = encoder(X)$ is the hidden state vector of the last layer of LSTM RNN. The vector is then projected into two vectors μ and σ to parameterize a normal distribution.

When encoding, we want our latent space representation to be close to the true input (i.e., $q_\phi(z|x)$ and $P(z)$) where we can use Kullback- Leibler Divergence to measure the closeness of two distributions.

The *KL* loss we define is:

$$L_{KL} = -1/2\ \Sigma^{M_z}{}_{i=m}(1 + \sigma^2 - exp(\sigma^2) - \mu^2)$$

Then we can draw a low-dimensional latent vector z by randomly sampling from the distribution. Z is sampled using a reparameterization trick:

$$z = \mu_\phi(x) + \varepsilon * \Sigma_\phi{}^{1/2}(x) \text{ where } \varepsilon \sim N(0, 1)$$

In the decoder, we can reconstruct the input sequence from the latent feature vector z (i.e., z is fed to each layer of the RNN to estimate the probability distribution of events for each time slot).

Thus we use a softmax activation function which is then followed by a weighted cross entropy loss to calculate reconstruction loss. The reconstruction loss is defined as .

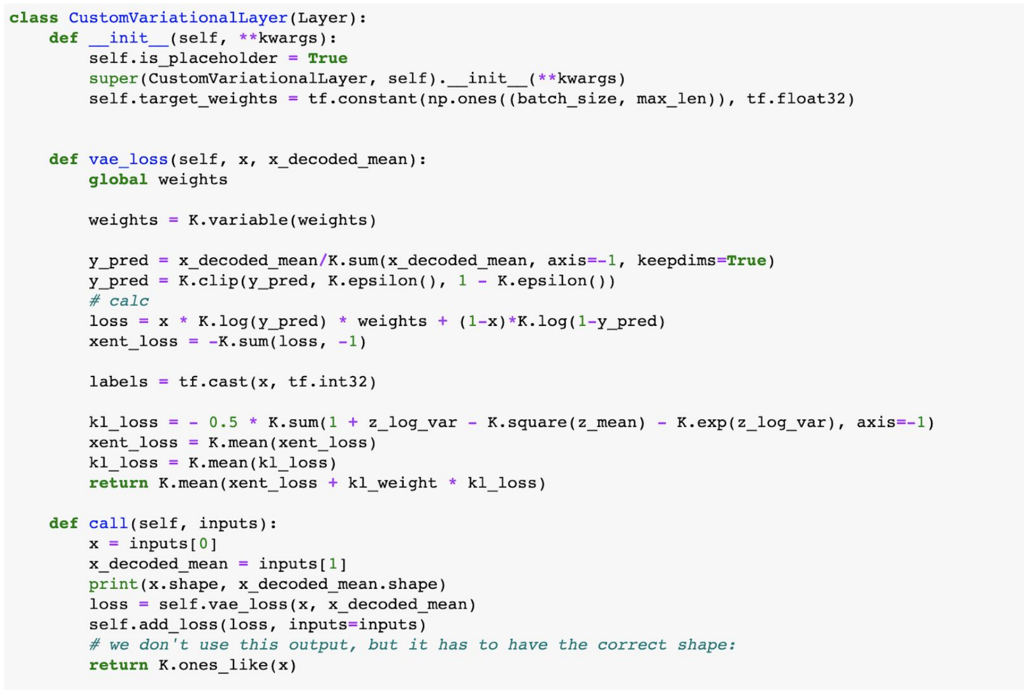
$$L_{reconstruction} \;=\; -1/n\, \Sigma^{n}_{i=1}\Sigma^{|E|}_{j=1}(w_{e_j}X_{i,j}log(x'_{i,j}) + (1 - x_{i,j})(log(1 - x'_{i,j})))$$

where $w_{e_j} \;=\; 1/log(n_j)$ is the weight to balance the prediction for unbalanced class of events in the training data and reduce the marginal importance of events with high occurrence.

The total loss is

$$L \;=\; L_{reconstruction} + w_{kl} * L_{kl}$$

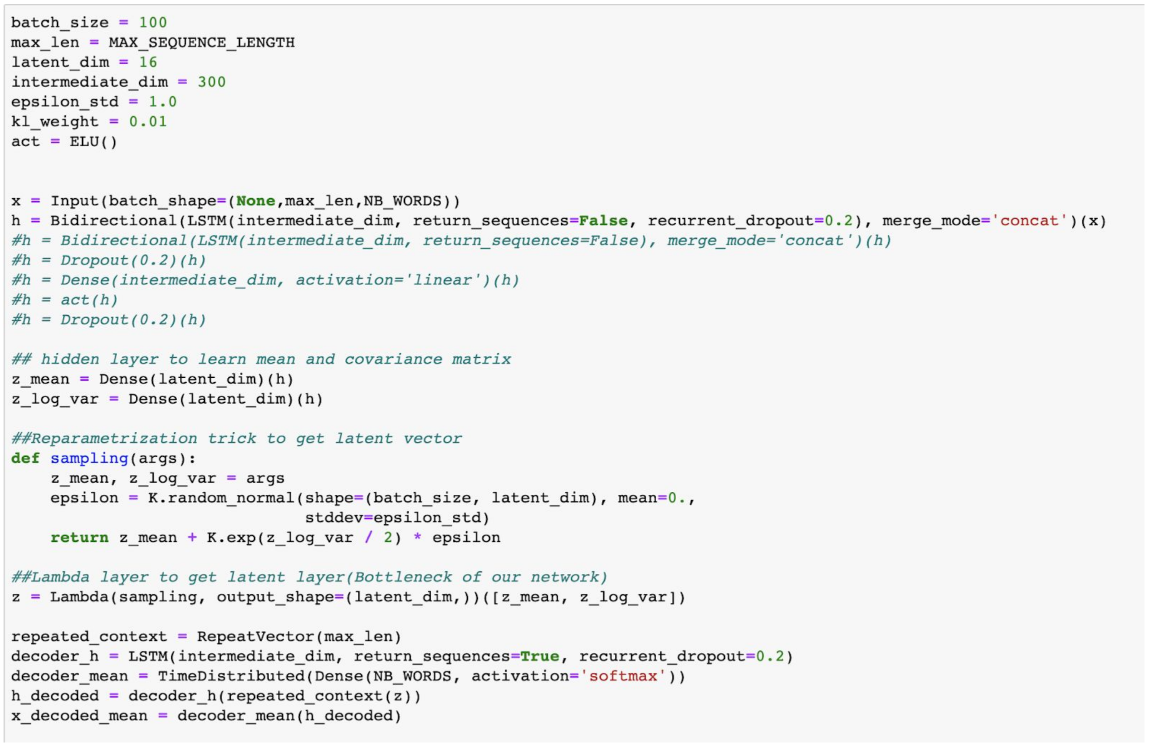
where w_{kl} is the KL weight in the loss term which we set to 1 in our first training for now.



```python
class CustomVariationalLayer(Layer):
    def __init__(self, **kwargs):
        self.is_placeholder = True
        super(CustomVariationalLayer, self).__init__(**kwargs)
        self.target_weights = tf.constant(np.ones((batch_size, max_len)), tf.float32)


    def vae_loss(self, x, x_decoded_mean):
        global weights

        weights = K.variable(weights)

        y_pred = x_decoded_mean/K.sum(x_decoded_mean, axis=-1, keepdims=True)
        y_pred = K.clip(y_pred, K.epsilon(), 1 - K.epsilon())
        # calc
        loss = x * K.log(y_pred) * weights + (1-x)*K.log(1-y_pred)
        xent_loss = -K.sum(loss, -1)

        labels = tf.cast(x, tf.int32)

        kl_loss = - 0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
        xent_loss = K.mean(xent_loss)
        kl_loss = K.mean(kl_loss)
        return K.mean(xent_loss + kl_weight * kl_loss)

    def call(self, inputs):
        x = inputs[0]
        x_decoded_mean = inputs[1]
        print(x.shape, x_decoded_mean.shape)
        loss = self.vae_loss(x, x_decoded_mean)
        self.add_loss(loss, inputs=inputs)
        # we don't use this output, but it has to have the correct shape:
        return K.ones_like(x)
```

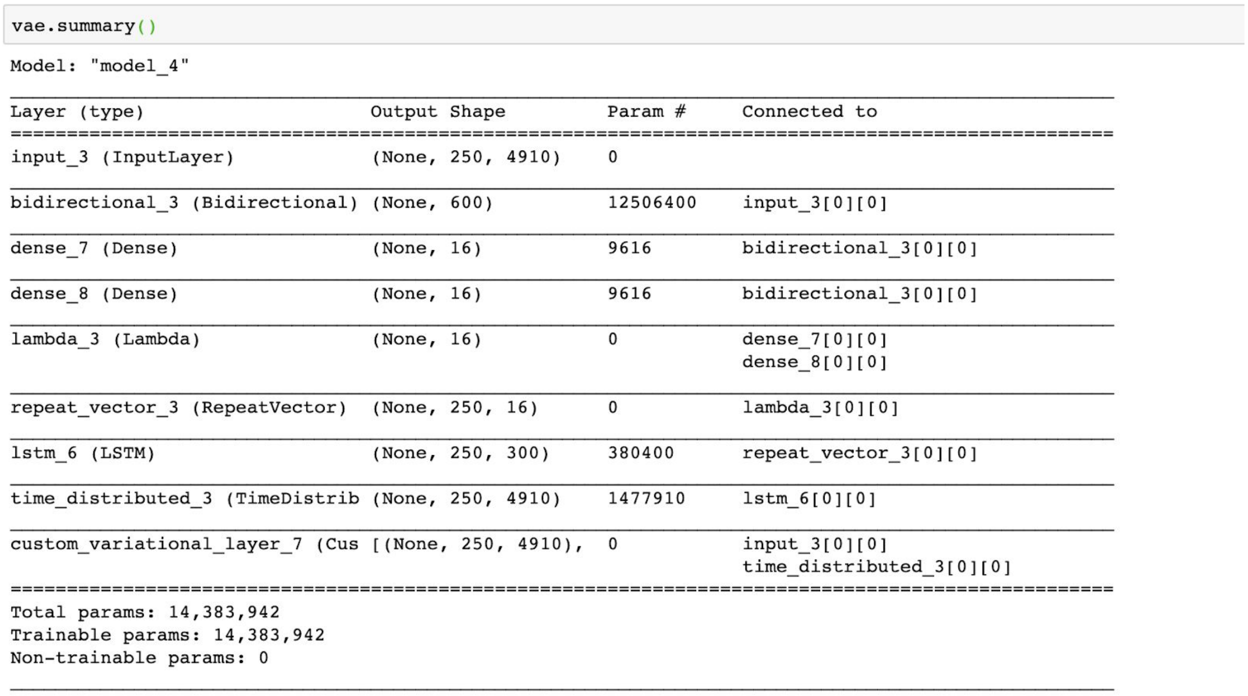
Figure IV: Custom VAE Loss (Keras)

The parameter we set in our first training is roughly the same in the paper mentioned in the beginning of this section. We employed bidirectional LSTM with 300 hidden nodes in the encoding layer and LSTM with 300 hidden nodes as a decoding layer. We set the dimensions of the latent representation layer as 16 in the paper and optimize the loss function with the Adam optimizer with training data batch size of 100 for each training step. We achieved a loss around 1.9 after 100 epochs of training.



```python
batch_size = 100
max_len = MAX_SEQUENCE_LENGTH
latent_dim = 16
intermediate_dim = 300
epsilon_std = 1.0
kl_weight = 0.01
act = ELU()


x = Input(batch_shape=(None,max_len,NB_WORDS))
h = Bidirectional(LSTM(intermediate_dim, return_sequences=False, recurrent_dropout=0.2), merge_mode='concat')(x)
#h = Bidirectional(LSTM(intermediate_dim, return_sequences=False), merge_mode='concat')(h)
#h = Dropout(0.2)(h)
#h = Dense(intermediate_dim, activation='linear')(h)
#h = act(h)
#h = Dropout(0.2)(h)

## hidden layer to learn mean and covariance matrix
z_mean = Dense(latent_dim)(h)
z_log_var = Dense(latent_dim)(h)

##Reparametrization trick to get latent vector
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(batch_size, latent_dim), mean=0.,
                              stddev=epsilon_std)
    return z_mean + K.exp(z_log_var / 2) * epsilon

##Lambda layer to get latent layer(Bottleneck of our network)
z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])

repeated_context = RepeatVector(max_len)
decoder_h = LSTM(intermediate_dim, return_sequences=True, recurrent_dropout=0.2)
decoder_mean = TimeDistributed(Dense(NB_WORDS, activation='softmax'))
h_decoded = decoder_h(repeated_context(z))
x_decoded_mean = decoder_mean(h_decoded)
```

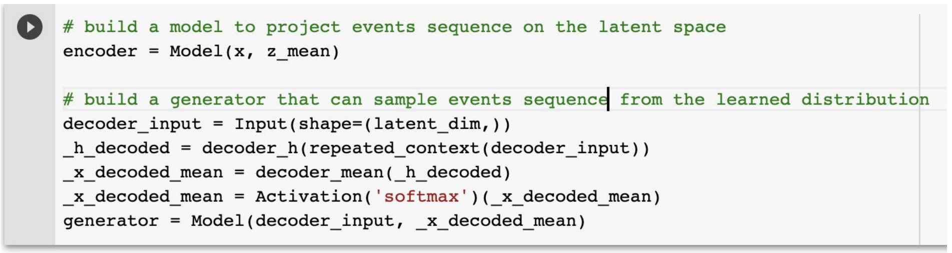
Figure V.: Parameter setting of our model & Model summary



```
vae.summary()
```

```
Model: "model_4"
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_3 (InputLayer)            (None, 250, 4910)    0
____________________________________________________________________________________________________
bidirectional_3 (Bidirectional) (None, 600)          12506400    input_3[0][0]
____________________________________________________________________________________________________
dense_7 (Dense)                 (None, 16)           9616        bidirectional_3[0][0]
____________________________________________________________________________________________________
dense_8 (Dense)                 (None, 16)           9616        bidirectional_3[0][0]
____________________________________________________________________________________________________
lambda_3 (Lambda)               (None, 16)           0           dense_7[0][0]
                                                                 dense_8[0][0]
____________________________________________________________________________________________________
repeat_vector_3 (RepeatVector)  (None, 250, 16)      0           lambda_3[0][0]
____________________________________________________________________________________________________
lstm_6 (LSTM)                   (None, 250, 300)     380400      repeat_vector_3[0][0]
____________________________________________________________________________________________________
time_distributed_3 (TimeDistrib (None, 250, 4910)    1477910     lstm_6[0][0]
____________________________________________________________________________________________________
custom_variational_layer_7 (Cus [(None, 250, 4910),  0           input_3[0][0]
                                                                 time_distributed_3[0][0]
====================================================================================================
Total params: 14,383,942
Trainable params: 14,383,942
Non-trainable params: 0
____________________________________________________________________________________________________
```

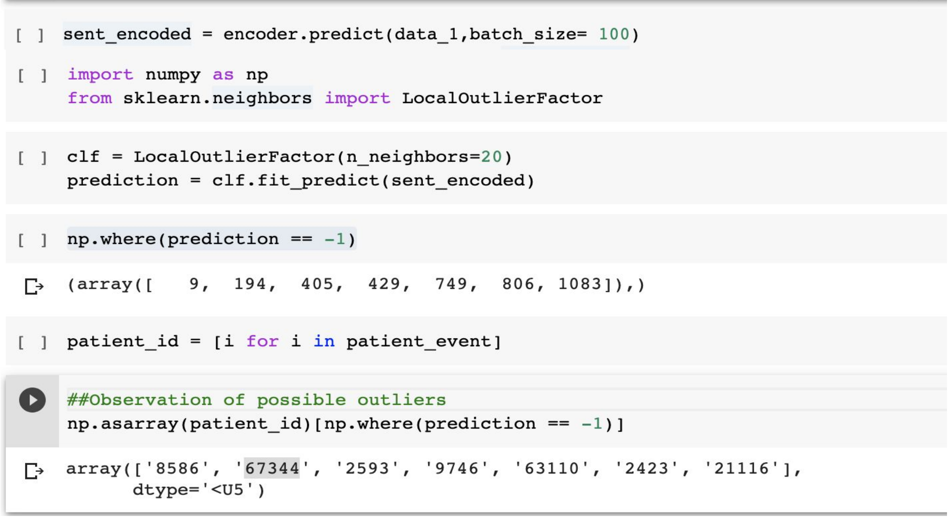
Then we can use our encoding layer to project our training event sequence data into latent representation into a same dimensional space. We are then able to use the LOF Scikit-Learn function to calculate the "outlier" score for each patient. If the outputs of localOutlierFactor.fit_predict() function equals to -1, we can question the patient as an anomaly.



```
# build a model to project events sequence on the latent space
encoder = Model(x, z_mean)

# build a generator that can sample events sequence from the learned distribution
decoder_input = Input(shape=(latent_dim,))
_h_decoded = decoder_h(repeated_context(decoder_input))
_x_decoded_mean = decoder_mean(_h_decoded)
_x_decoded_mean = Activation('softmax')(_x_decoded_mean)
generator = Model(decoder_input, _x_decoded_mean)
```

Figure VI.

Sequence Projection



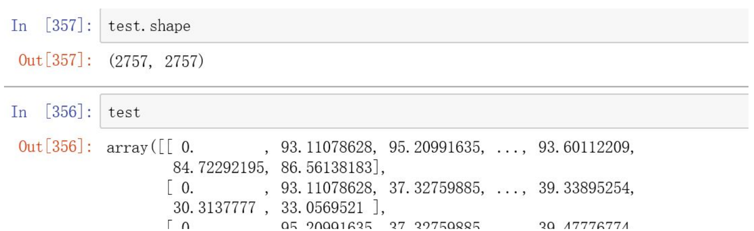
```
[ ] sent_encoded = encoder.predict(data_1,batch_size= 100)

[ ] import numpy as np
    from sklearn.neighbors import LocalOutlierFactor

[ ] clf = LocalOutlierFactor(n_neighbors=20)
    prediction = clf.fit_predict(sent_encoded)

[ ] np.where(prediction == -1)

    (array([   9,  194,  405,  429,  749,  806, 1083]),)

[ ] patient_id = [i for i in patient_event]

    ##Observation of possible outliers
    np.asarray(patient_id)[np.where(prediction == -1)]

    array(['8586', '67344', '2593', '9746', '63110', '2423', '21116'],
          dtype='<U5')
```

Figure VII. Output of LOF function

4.2. Second approach: DTW Distance based approach

The second approach is to implement anomaly detection algorithms based on DTW Distance we computed in Milestone 2. In milestone 2, we have found the optimal alignment between two different sequences and calculated the similarity between sequences by computing a accumulated euclidean distance between the aligned event pairs. DTW aligns two temporal sequences with the optimal matching and uses the sum-of pairs distance for the aligned series to denote sequence similarity. Here we take the pairwise DTW distance of patients with diagnosis in Injure & Fracture category (with ICD9 starts with 800) for example.

Figure VII. Numpy Output of DTW distance matrix



```
In [357]: test.shape

Out[357]: (2757, 2757)

In [356]: test

Out[356]: array([[ 0.         , 93.11078628, 95.20991635, ..., 93.60112209,
                   84.72292195, 86.56138183],
                  [ 0.         , 93.11078628, 37.32759885, ..., 39.33895254,
                   30.3137777 , 33.0569521 ],
                  [ 0.         , 95.20991635, 37.32759885, ..., 39.47776774
```

4.2.1 *Unsupervised Outlier Detection using Local Outlier Factor (LOF)*

The Local Outlier Factor (LOF) algorithm computes the local density deviation of a given data point with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood. More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers.

4.2.1.1

Implemented the LOF algorithm from scratch by computing reachability distance from each patient to his or her k-nearest neighbors.

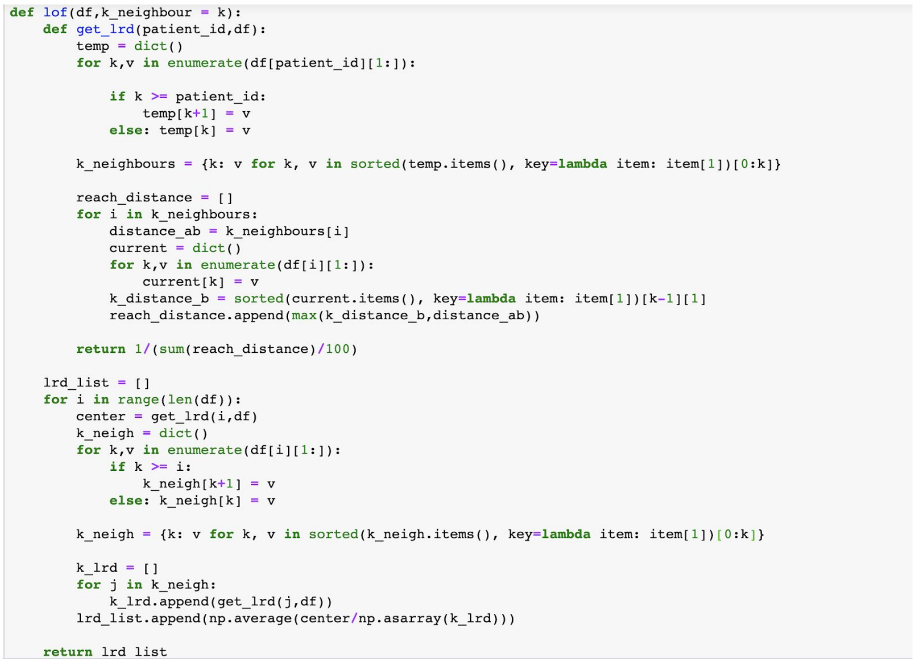
$$reachability\ distance\,(a,b)\ = max\ \{k\ distance(b),\ dist(a,b)\}$$

where $k\ distance(b)$ is the distance of b to its k-th closest point

Then we computed local reachability density (lrd) for each patient a:

$$lrd(a)\ = 1/(\Sigma_{n\in N_k(a)}\ reachability\ distance(a,n)\ /\ k)$$

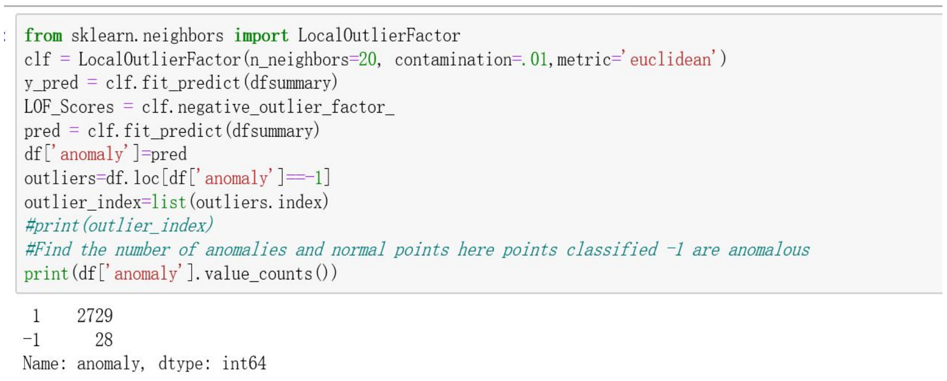
Then the LOF is basically the average ratio of the lrd of a's k nearest neighbours to the lrd of a.



```python
def lof(df,k_neighbour = k):
    def get_lrd(patient_id,df):
        temp = dict()
        for k,v in enumerate(df[patient_id][1:]):

            if k >= patient_id:
                temp[k+1] = v
            else: temp[k] = v

        k_neighbours = {k: v for k, v in sorted(temp.items(), key=lambda item: item[1])[0:k]}

        reach_distance = []
        for i in k_neighbours:
            distance_ab = k_neighbours[i]
            current = dict()
            for k,v in enumerate(df[i][1:]):
                current[k] = v
            k_distance_b = sorted(current.items(), key=lambda item: item[1])[k-1][1]
            reach_distance.append(max(k_distance_b,distance_ab))

        return 1/(sum(reach_distance)/100)

    lrd_list = []
    for i in range(len(df)):
        center = get_lrd(i,df)
        k_neigh = dict()
        for k,v in enumerate(df[i][1:]):
            if k >= i:
                k_neigh[k+1] = v
            else: k_neigh[k] = v

        k_neigh = {k: v for k, v in sorted(k_neigh.items(), key=lambda item: item[1])[0:k]}

        k_lrd = []
        for j in k_neigh:
            k_lrd.append(get_lrd(j,df))
        lrd_list.append(np.average(center/np.asarray(k_lrd)))

    return lrd list
```

Figure VIII. LOF algorithm from scratch: outlier score for each patient

4.2.1.2 Secondly, we use dtw statistics summary as features for each patient. After we set up our feature properly, we are able to implement anomaly detection models (ex, Local Outlier Factor (LOF), Isolation Forest).



```
from sklearn.neighbors import LocalOutlierFactor
clf = LocalOutlierFactor(n_neighbors=20, contamination=.01,metric='euclidean')
y_pred = clf.fit_predict(dfsummary)
LOF_Scores = clf.negative_outlier_factor_
pred = clf.fit_predict(dfsummary)
df['anomaly']=pred
outliers=df.loc[df['anomaly']==-1]
outlier_index=list(outliers.index)
#print(outlier_index)
#Find the number of anomalies and normal points here points classified -1 are anomalous
print(df['anomaly'].value_counts())
```

```
 1    2729
-1      28
Name: anomaly, dtype: int64
```

Figure IX Set up of LOF Model Here we set The number of neighbors considered to be 20

After model training, we visualize the results and check if the classification makes sense.We Normalize and fit the metrics to a PCA to reduce the number of dimensions and then plot them in 2D highlighting the anomalies. We find both global outliers and local outliers in our graph.

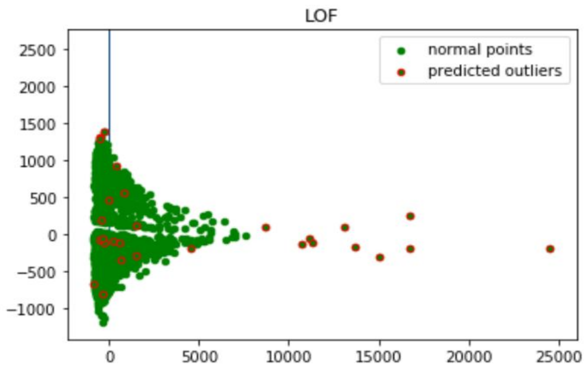


Figure X. Setup of LOF Model. The red points in the left amont green cluster are considered local outlier

4.2.2 Outlier Detection using Isolation Forest

The algorithm isolates each point in the data and splits them into outliers or inliers. Isolating an outlier means fewer loops than an inlier.

The Isolation Forest algorithm isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. The logic argument goes: isolating anomaly observations is easier because only a few conditions are needed to separate those cases from the normal observations.

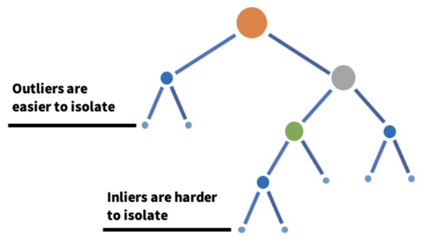
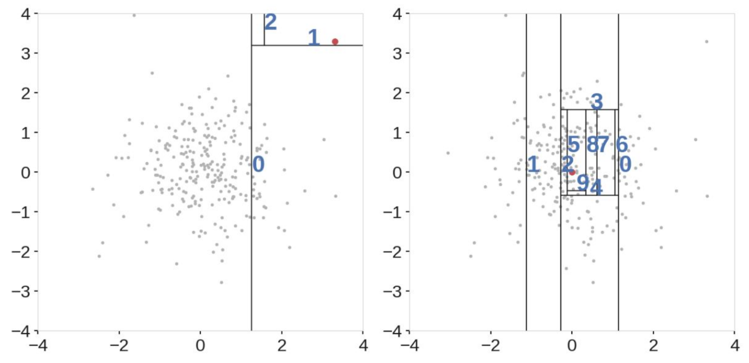
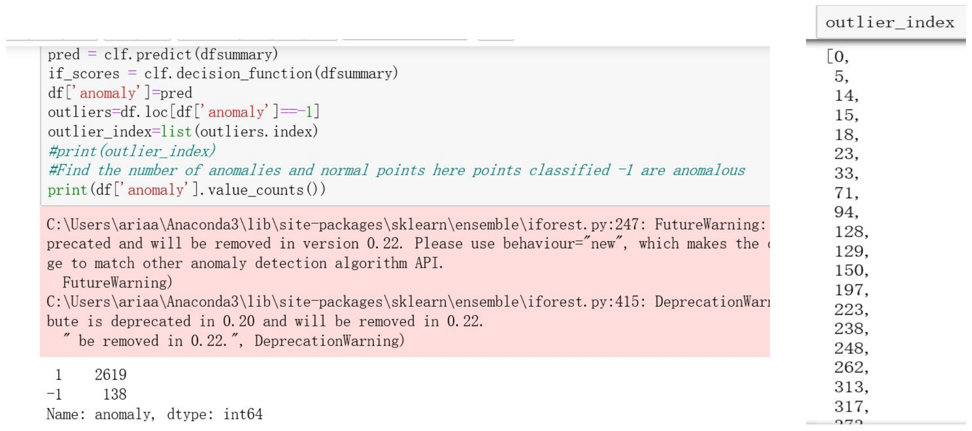


Figure XI &XII Concept of Isolation Forest (Inliers are harder to isolate than outliers.)



(a) Anomaly point (b) Nominal point

On the other hand, isolating normal observations requires more conditions. Therefore, an anomaly score can be calculated as the number of conditions required to separate a given observation.

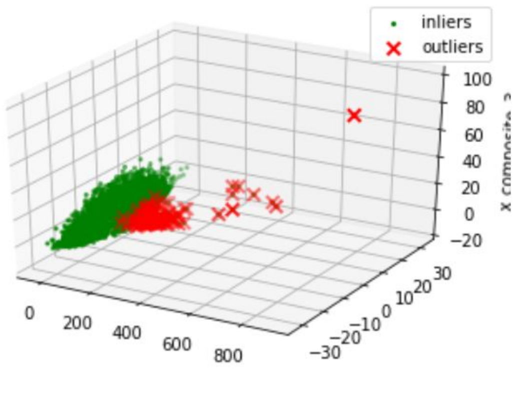


```
pred = clf.predict(dfsummary)
if_scores = clf.decision_function(dfsummary)
df['anomaly']=pred
outliers=df.loc[df['anomaly']==-1]
outlier_index=list(outliers.index)
#print(outlier_index)
#Find the number of anomalies and normal points here points classified -1 are anomalous
print(df['anomaly'].value_counts())
```

```
C:\Users\ariaa\Anaconda3\lib\site-packages\sklearn\ensemble\iforest.py:247: FutureWarning:
precated and will be removed in version 0.22. Please use behaviour="new", which makes the
ge to match other anomaly detection algorithm API.
  FutureWarning)
C:\Users\ariaa\Anaconda3\lib\site-packages\sklearn\ensemble\iforest.py:415: DeprecationWar
bute is deprecated in 0.20 and will be removed in 0.22.
  " be removed in 0.22.", DeprecationWarning)

 1    2619
-1     138
Name: anomaly, dtype: int64
```

```
outlier_index
[0,
 5,
 14,
 15,
 18,
 23,
 33,
 71,
 94,
 128,
 129,
 150,
 197,
 223,
 238,
 248,
 262,
 313,
 317,
```

Figure XIII Setup of Isolation Forest Model and partial output.



5. Discussion of Milestone 3 and Exploration of Next Step

Until milestone 3, we tried several ***new ideas*** here. For example, since traditional unsupervised anomaly detection algorithms have a limited capacity to handle temporal sequence data, we employ LSTM-based VAE to handle the irregularity of event sequence and identify diverse types of anomalies.

However, VAE model implementation faces a challenge that the training of inference model needs a large amount of training data to learn the distribution of input and generate the latent variable (Encoder of LSTM-based VAE). But our two sub datasets corresponding to two categories of ICD-9 codes do not have that much training data size. Thus, we need to consider whether to use the whole dataset of 61000+ patients as our input layer to learn distributions more accurately. For these two training dataset, we may consider the results of DTW distance based outlier detection would seem more justified.

In the rest of the project implementation, we are going to further refine the model we used in milestone 3.

For our LSTM-based VAE model, we will further tune the hyperparameters (Eg. RNN layer dimensions, latent space layer Z dimensions, etc.) of our model. Also, we will include more training data into our input layer and adaptly increases the w_{kl} to make sure that the reconstruction loss is optimized with high priority.

Also, to enhance both model performances, we will try to go back to the data preprocessing and embedding steps and improve them. For instance, we use Skip-gram for word embedding currently. We also want to use Term Frequency-Inverse Document Frequency (TF-IDF) scores measuring the importance of each event, remove noisy events and exclude extremely short (& long) sequences (i.e., sequence length < 2) to prepare a high-quality event sequence data for subsequent training of the anomaly detection model.

Reference:

S. Guo, Z. Jin, Q. Chen, D. Gotz, H. Zha and N. Cao, *Visual Anomaly Detection in Event Sequence Data*, 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 1125-1130.

Guo, Rongchen & Fujiwara, Takanori & Li, Yiran & Lima, Kelly & Sen, Soman & Tran, Nam & Ma, Kwan-Liu. (2020). *Comparative visual analytics for assessing medical records with sequence embedding*. Visual Informatics. 10.1016/j.visinf.2020.04.001.

Anne Fischer, *Generating Text From latent Space: creating novel news headlines from latent space using a Variational AutoEncoder*, 2019

Rasim Alguliyev, Ramiz Aliguliyev, Lyudmila Sukhostat, *Anomaly detection in Big data based on clustering,* Stat., Optim. Inf. Comput., Vol. 5, December 2017