Autoencoders Ensemble for Anomaly Detection on Data Pipelines

Author: Rafael Leiniö

Neural Networks Postgraduate Class - National Institute of Space Research (INPE)

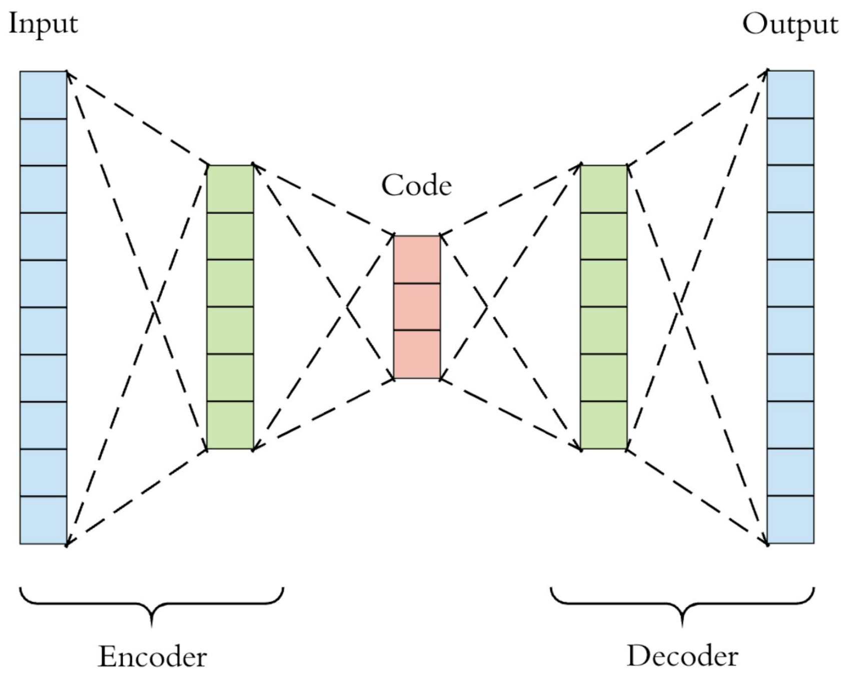
Prof. Dr. Marcos Gonçalves Quiles

Code available at Github

Introduction

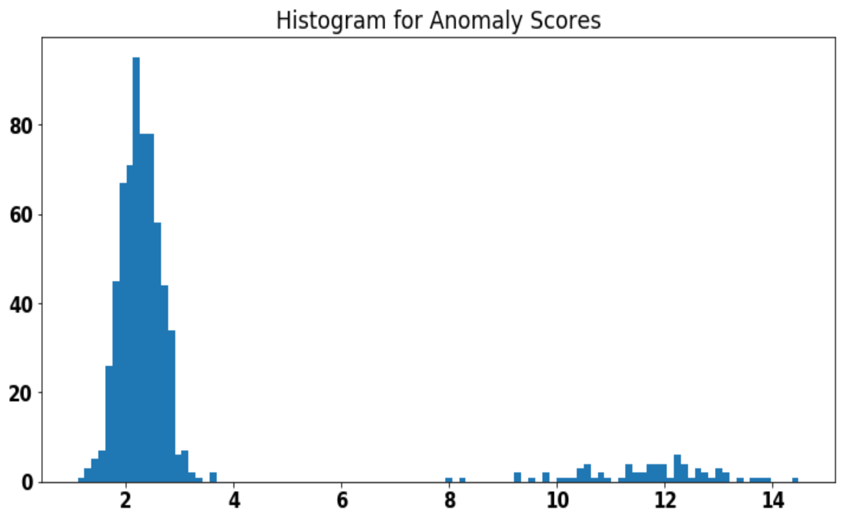
The Autoencoders are an unsupervised learning technique that uses neural networks for the representation learning task. Specifically, it is a neural network architecture that forces a compressed knowledge representation of the original input. If there is any structure in the data (i.e., correlations between input resources), this structure can, therefore, be learned and exploited by forcing entry through the network bottleneck. They work by compressing the entry into a lower dimensionality space of representation and then reconstructing the output of this representation. This type of network consists of two parts:

- **Encoder**: is the part of the network that compresses the entry into a lower dimensionality representation (encoding the input).
- **Decoder**: This part aims to reconstruct the representation of the input of the latent space.



Using autoencoders as an anomaly detector is a simple approach. As explained this learning algorithm work as a dimensionality reduction technique. Encoding means losing some information about the data, but at the same time, the model aims to learn the main pattern of the data. From the encoded layer records more close to the learned pattern can be easily reconstructed, but in opposition, records that are outliers can’t be reconstructed very well. Knowing this, from the output of the network an anomaly score can be attributed by calculating the pairwise distance of the output point to the source point.

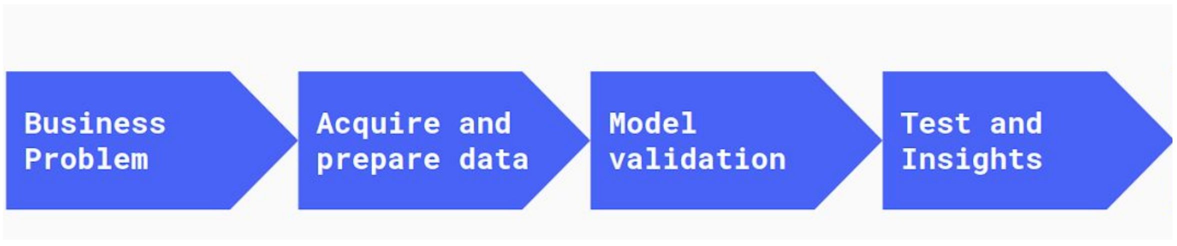
After a network is trained with some data, is possible to plot a histogram of the calculated anomaly scores is the following way:



From this histogram is possible to see a distribution with the majority of the data in the left part of the plot. Around 90% of the data has a score distributed between 0.5 and 4. But is possible to see that this dataset has some points with huge anomaly scores in the left part of the plot. Knowing the data is possible to set a threshold around 4 and define everything more than 4 outliers, for example. It is common in projects of autoencoders for anomaly detection to build an analysis like that.

One problem of unsupervised methods, like the one described, is that they can be easily overfitted to the training dataset and thus perform poorly on other datasets. In the network itself, techniques such as L2 regularization and dropout can be used. But another technique is the ensemble. In this technique, it's not used just one trained model to provide the anomaly score, but a set of models. To calculate the final score it is used some aggregation method (average, maximum or minimum for example) over all the scores provided by the models.

Methodology



Business Problem

Three different artificial datasets were used to train the models. They are simple datasets with just 2 features, but this enables an easy understanding of the process of learning the topology.

Acquire and Prepare Data

For this project there isn’t a ready dataset to be used to train the models. So the dataset had to be created from the raw data daily generated by the data pipelines. One specific data pipeline was chosen to be worked in this project, it is one that extracts data of logs of user behavior. On the Data Lake of the company it was presented all the daily extractions since 2018. The data is partitioned by year, month, and day, so for each day exists an output dataframe generated by this pipeline. Over each of those dataframes the metrics were calculated to generate the final dataset that was used for training the models.

The metrics calculated over the raw dataframe to describe it globally and generate the feature dataset were the following:

- **n_rows**: number of the rows of the dataframe
- **moving_avg_7d_n_rows**: the average of n_rows in the last 7 days
- **null_values_ratio**: the ratio between the total of not-null and the total of null values presented in the dataframe
- **moving_avg_7d_null_values_ratio**: the average of moving_avg_7d_null_values_ratio in the last 7 days
- **max_string_length**: the max string length found in the dataframe
- **moving_avg_7d_max_string_length**: the average of max_string_length in the last 7 days
- **min_string_length**: the minimum string length found in the dataframe

- **moving_avg_7d_min_string_length**: the average of min_string_length in the last 7 days
- **avg_string_length**: the average of the string length over all the string fields in the dataframe
- **moving_avg_7d_avg_string_length**: the average of avg_string_length in the last 7 days
- **max_numeric**: the max numeric value found in the dataframe
- **moving_avg_7d_max_numeric**: the average of max_numeric in the last 7 days
- **min_numeric**: the minimum numeric value found in the dataframe
- **moving_avg_7d_min_numeric**: the average of min_numeric in the last 7 days
- **avg_numeric**: the average numeric value over all the numeric fields in the dataframe
- **moving_avg_7d_avg_numeric**: the average of avg_numeric in the last 7 days
- **date**: the day when the dataframe was created. This column is not used for training the data, it is just metadata.

Model Validation

In the total, 72 models were created to be validated varying the network layers and hyperparameters such: Dropout rate, L2 regularizer, activation functions, and optimization functions.

All the models were submitted to a K-fold validation, with k equal to 5. Calculating for each model the mean RMS of the reconstruction over each fold.

Test and Insights

From the Model Validation phase, it was ranked the best models by the RMS mean score over the 5-folds. The top 5 models were chosen to build the ensemble. These

5 models were re-trained, each one with an overlapped 80% subset of the data, in a kind of k-fold technique again, but this time not with a validation purpose, but to generate the final trained models.

The idea is to generate the final ensemble with models that were trained not seeing all the data, preventing the outliers, possibly contained in the data, to be fitted by all the models. The final ensemble is going to predict anomaly scores based on models that saw different subsets of the data, aiming this way, to enhance the generalism of the model and prevent overfitting.

Results

Top 5 models in Model Validation Stage

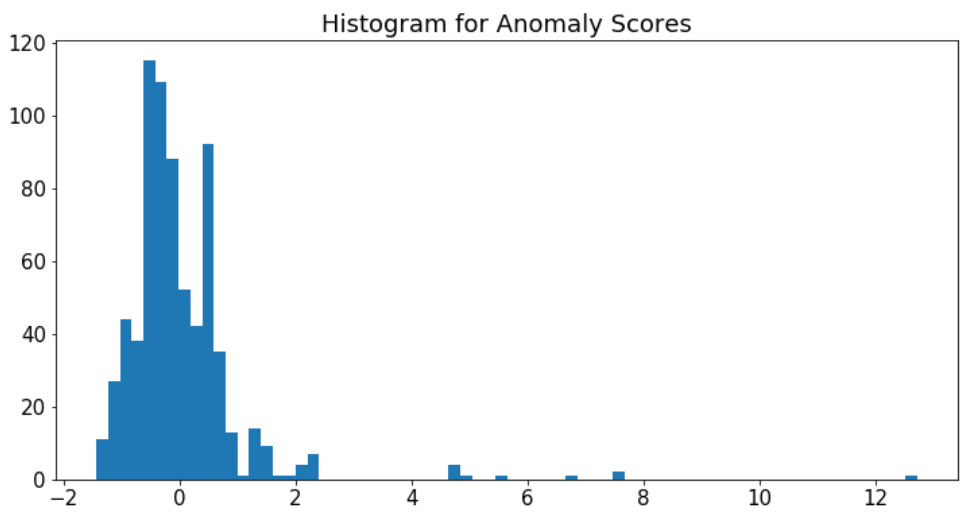
The 5 following models were the ones with the best scores among all the evaluated parameters tested. These 5 models were chosen to compose the final ensemble model.

Layers*	Dropout	L2 Regularizer	Hidden Layers Activation Function	Optimizer	Output Layer Hidden Functions	5-Fold Mean RMS
[16, 8, 4, 2, 4, 8, 16]	0.2	0.01	ReLU	Adam	Sigmoid	1.0590
[16, 8, 4, 8, 16]	0.1	0.01	ReLU	Adam	Sigmoid	1.0690
[16, 14, 10, 6, 10, 14, 16]	0.2	0.01	ReLU	Adam	Sigmoid	1.0735
[16, 12, 8, 4, 8, 12, 16]	0.1	0.01	ReLU	Adam	Sigmoid	1.0746
[16, 16, 12, 8, 12, 16, 16]	0.2	0.01	ReLU	Adam	Sigmoid	1.0799

*The first value is the input layer, equals to the length of the feature array. The last value is the output layer, so obviously the same as the input layer as autoencoders try to reconstruct the input feature array.

Test Results and Insights

Calculating the anomaly scores for all the dataset, the following histogram is generated:



Defining the cut point in 2.5, all the records at the right of this point could be considered outliers. Doing that 10 records were considered anomalies.

Deeply Analysing the data we can see that indeed these 10 records have some of the features very distant from their weekly running average. Analyzing these anomalies from a Data Engineer perspective one of them can be considered a major anomaly. It was in the day 21/02/2018, in the histogram is the point in the extreme right of the plot, the highest anomaly score. In that observation was registered a dataframe with 1 million rows while the weekly average in that time was about 100k. This observation starts a deep investigation of the problem with the Data Engineer team that provided the data for this project.

Besides evaluating the predicted anomaly scores, it was created 10 fake observations representing anomalies and were tested over the model. In this test 6 of 10 got I high anomaly score as desired, but 4 failed to be recognized as outliers. What was observed in this test is that the current model was easily giving high scores for values higher than the normal, but failed in giving high anomaly scores to low values observations. Both extreme high or extreme low observed values should be pointed as outliers, but the model is only working well to spot anomalies with values higher than normal.

The only feature that is not suffering from this issue is **null_values_ratio**. This feature is the only one that does not represent an absolute value, but a ratio. This metric floats between 30 to 40% in all the historical observations. Any value a little bit higher or lowers than these bounds are easily spotted as an outlier by the model, as desired.

This last test gave an interesting insight to be validated in the future. The idea is to engineer all features as ratios between an observation and the moving average calculated over a time window. This way absolute values are not going to be used anymore. It is expected to be an enhance in the performance of the model, solving the described issue.

References

- https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6
- https://saketsathe.net/downloads/autoencode.pdf
- https://arxiv.org/pdf/1811.05269.pdf
- https://github.com/yzhao062/pyod