Title: Anomaly Detection of Electrical Signals in a Particle Accelerator

By: Colin O’Brien

Tags: anomaly detection, machine learning, deep learning, autoencoders, support vector machines, feature engineering, Fourier transformations

While studying at the University of Virgina, I took part in a capstone course while I was lucky enough to work with JLab and Oak Ridge facilities (get their full real names) on an anomaly detection problem they were facing. They were trying to use electrical signals from some of the machines in a particle accelerator to predict if the system was about to malfunction. On this page, I want to outline one of the proposed solutions to this problem. The main pieces of the pipeline I used are as follows:

1. Fourier transformation as a data transformation step
2. Training an autoencoder
3. Applying a Radial Support Vector Machine to the latent space of the autoencoder.

Something important to note about the methods I used is that I was not only concerned with building a method that could accurately find abnormal observations, I wanted to also achieve a level of interpretability and ability to visualize the difference between observations.

Data Overview:

The data consisted of electrical signals from the HVCM system, one of the key systems in creating the beam of a particle accelerator. The observations fell into one of two categories:

* “Normal”: a signal collected right before the beam was produced in the particle accelerator.
  + I like to think of these as a “normal heartbeat” of the HVCM system.
* “Faults”: Signals collected right before the machine failed to produce the beam, meaning an experiment or other research could not be completed. Once a Fault occurs, the error must be diagnosed, with 14 different types of Faults.
  + I think of these as the “heartbeat right before a heart attack” in the HVCM system.

For each observation, there are 19 different sub-components of the HVCM system, each with a different signal (or heartbeat). To the right is an example for one of the “Normal” observations. The aim is train a method that looks across all 19 parameters and finds differences in these signals in order to make classifications.

Building Pipeline and Training models:

Bag of Symbolic Fourier Approximation Symbols

After splitting the data into training and test data, the first step is applying the Bag of Symbolic Fourier Approximation Symbols (BOSS) transformation. For a given observation, each of the 19 sub-components have a sliding window applied. Each snapshot captured by the sliding window is decomposed by its Fourier coefficient, which is used to assign the snapshot to one of the 8 different words in a 2 letter, 3 index dictionary. As the sliding window moves across the signal, the many snapshots are used to produce a histogram, which can be represented as 1D vector.

In a less technical perspective, the snapshots are grouped together, with similar snapshots being given the same word/location in the histogram. This process is repeated for all 19 parameters, creating a 19 by 8 matrix for each observation. (Add advantages to doing this here?)

Training autoencoder

The second step is training an autoencoder. An autoencoder takes some in some kind of data in the form of matrices or tensors, compresses that data down to a small latent space and then tries to rebuild the original data. The reason for doing this is to get a low dimensional representation of the data, thus finding a few learned features that can be used to accurately represent the differences between observations.

Below is an architecture of the encoder I built. The 19x8 matrices from the BOSS transformation are flattened, gradually compressed down to a 3-dimensional space, then reconstructed. The model is trained using gradient descent based on the loss produced by the difference between the input 19x8 matrices and the output 19x8 matrices (this is repeated until the loss converges). Once the autoencoder is trained, data can be projected into the latent space.

Training Support Vector Machine

Since we have label data for classification, a support vector machine (SVM) make sense to try to build a decision boundary between the different classes of data. Since the data is not likely to be easier to separate linearly, a radial kernel will be used, which allows for complex decision boundaries.

We have two main parameters here that need to be tuned using the training data, C and gamma. C is essentially the tradeoff between misclassifications and a smooth decision boundary. If the decision boundary is too jagged, the SVM will not generalize to unseen data. Gamma is a parameter that controls how much weight training observations have on the decision boundary. This helps control the SVM’s ability to learn the shape of the distribution of each class. Again, if gamma is too high, the SVM will overfit and will not generalize to unseen data. I tuned these parameters using a simple grid search (5-fold cross validation). Since I have unbalanced data (more Normal observations than Faults), I used the parameters that maximized my F-1 score.

Visualizing Training Results

Here we see the scores of the SVM once fit to all the training data, as well as a visual representation of the latent space, with color indicating whether a signal was correctly classified.

The two blue clusters in the bottom right indicate large clusters of correctly classified Normal observations. This was interesting as there seems to be two separate sub-types of normal signals being produced by the HVCM. Green observations are correctly classified Faults. TPS and DV/DT High Faults seemed to be forming some small clusters of their own, suggesting in future analysis, these types of faults may want to be considered a sub-category.

There are a few red dots indicating Faults that were misclassified. These Faults are appearing as identical to a trained autoencoder, suggesting that at least within the HVCM electrical data, there is no signal in these observations that could be used to identify them as abnormal. A takeaway from this would be that more data from other systems beyond the HVCM may be needed in order to classify these observations accurately.

The accuracy score metric is not particularly helpful given unbalanced data, but the higher precision score compared to recall indicates the model is better at avoiding false positives than it is at finding faults.

Visualizing test Results

Here we see the test data passed through the same pipeline (BOSS transformation -> autoencoder -> SVM). Given that this is unseen data, it is expected that there is a drop off in performance. However, many same trends can be seen. Precision remains higher, the two normal clusters are present and TPS faults appear to be forming their own cluster.

Conclusion?

Area for further development

Like any data science project, there are areas to improve with more time and resources. One of the largest is parameter tunning for the BOSS transformation. Given the computation resources I had access to at the time, I was only able to use a smaller possible dictionary of words. Another area that would have been interesting to explore would be perforce of the SVM verse using SoftMax output layers for classification. The latent space layer could have been connected to an additional SoftMax layer. Finally, measuring uncertainty would have been an interesting facet to explore, with various Bayesian approaches to classification providing more measurements for uncertainty.