Henry C Woo

University of Illinois at Urbana-Champaign

Wells Fargo Campus Analytics Challenge 2020

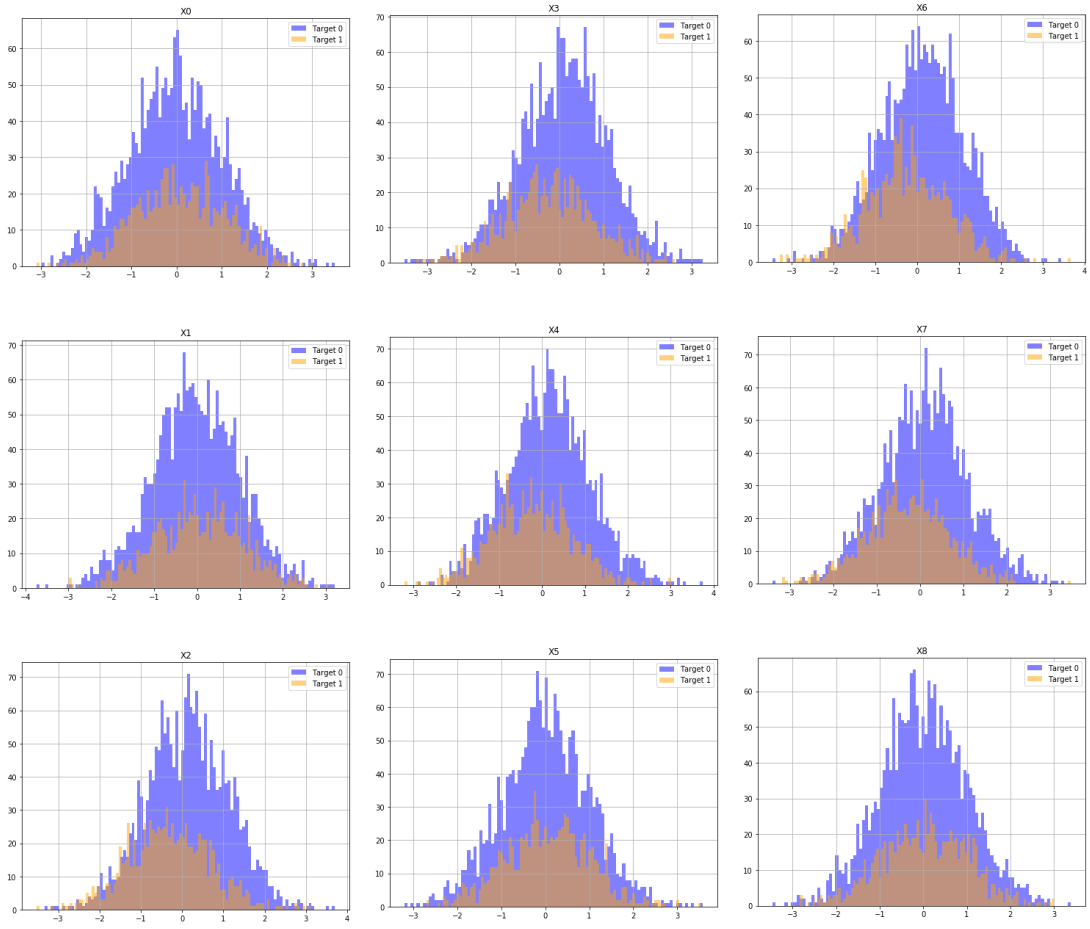
Deliverable 2

Exploratory Analysis

Before I started building models, I needed more information on the data itself. The code and results from this analysis can be found in the `notebooks` folder. In this challenge, there was no context to the type of data I was dealing with. For example, are they time-series? What are the scales in which the data points lie? How separable are the two target classes? Do the features appear to follow a distribution?

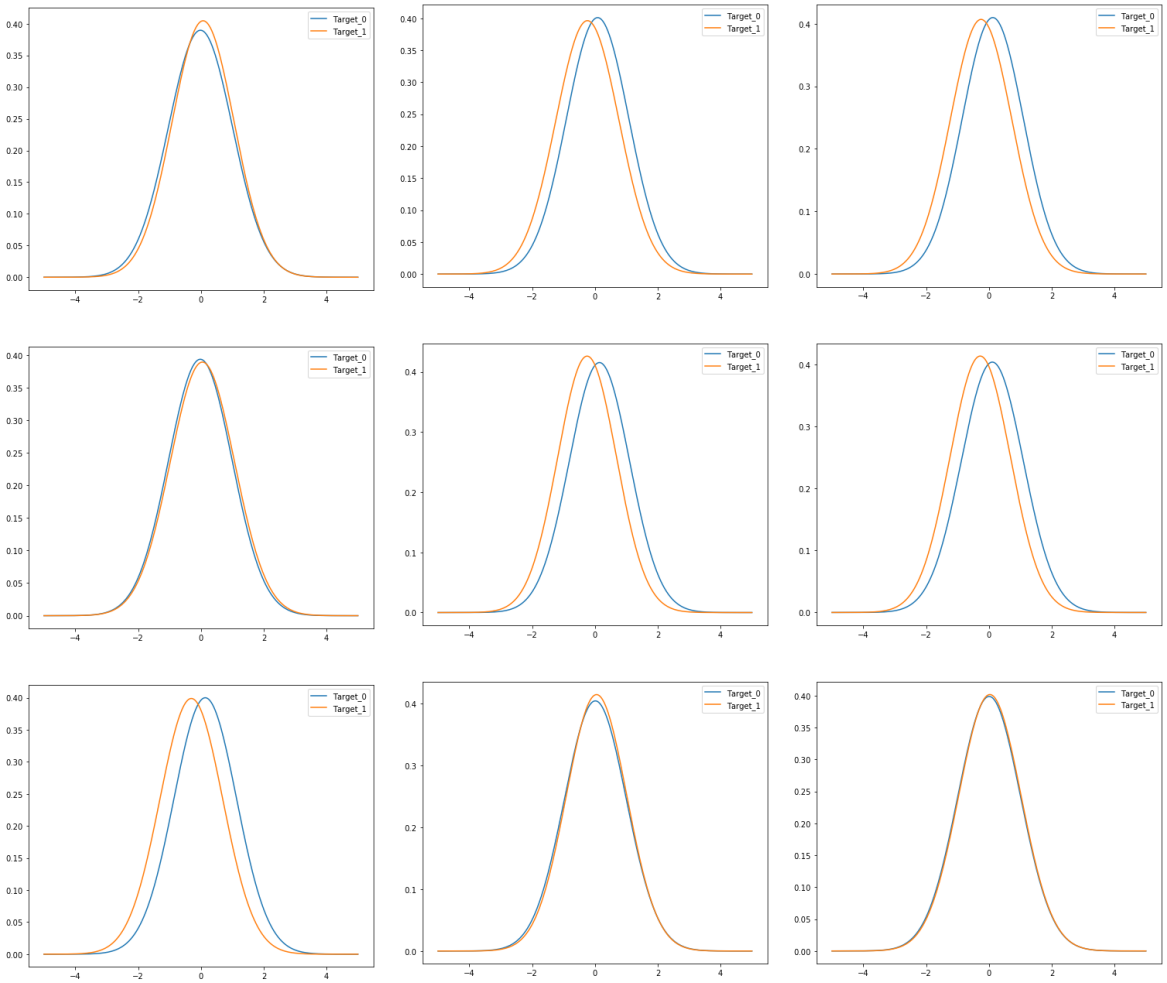
So, the first step was to take a look at each individual feature. There is a total of 30 quantitative features labelled ‘X0’, ‘X1’, ‘X2’, …, ‘X29’ and one categorical feature labelled ‘XC’. I plotted a histogram of the quantitative features to visually see the distributions of the values versus their target labels.

Plotted below are the first 9 quantitative features.



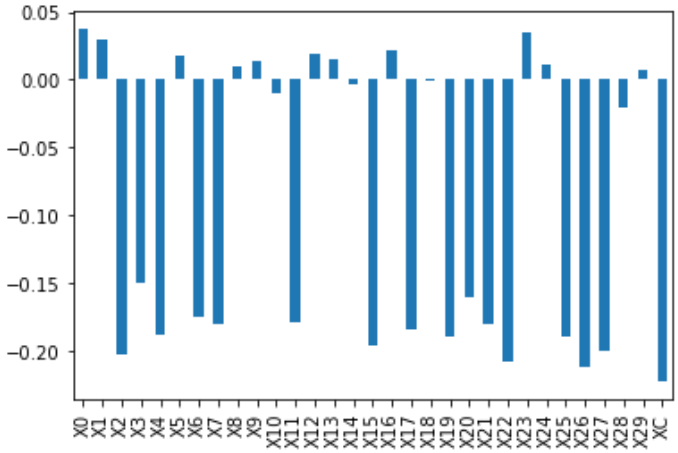
Histograms of First 9 Quantitative Feature

The histograms are plotted with raw data values (no normalization) and they appear to follow unimodal distributions. In the next visualization, the data is assumed, for now, to follow a Gaussian distribution and the probability density functions of each target label are graphed on top of each other. This is to see how different or similar the distributions. The stronger the difference, the more separable the target labels are.



Probability Density Functions (Assumed Gaussian) of First 9 Quantitative Features

Unfortunately, it seems that some of the distributions are nearly identical, so I needed to look more closely into the correlation between each feature and the target label.



Pearson Correlations Between Features and Target Label

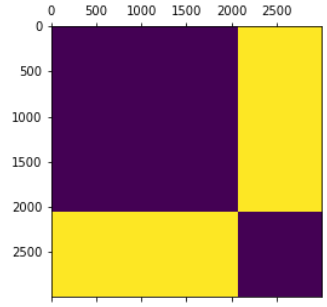
From the graph, there are many features that have negative correlation with the target label. The feature column 'XC' had the greatest correlation magnitude, so it is best not to omit it from training.

Further Analysis

Some further analysis was done to find relationships between datapoints and its target label.

Similarity Matrix

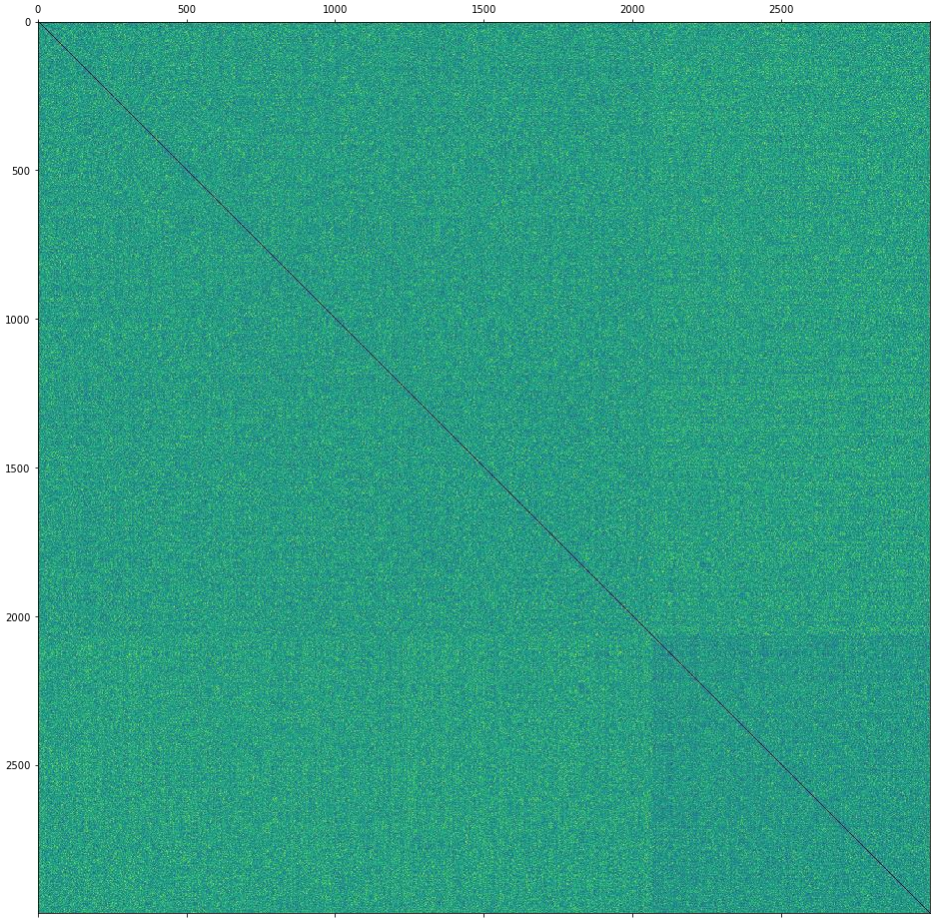
One way to demonstrate the separability of datapoints is to show a similarity matrix. The first 2069 indices of the similarity matrix have target labels 0 and the following 931 indices have target labels 1. The darker the pixel, the more similar the features are. So in an ideal situation, the similarity matrix would look like the following where all datapoints are the same for its target label but dissimilar with the other.



Ideally Distinctive Features Similarity Matrix Example

This matrix is simply used to demonstrate what to expect and does not represent the data whatsoever.

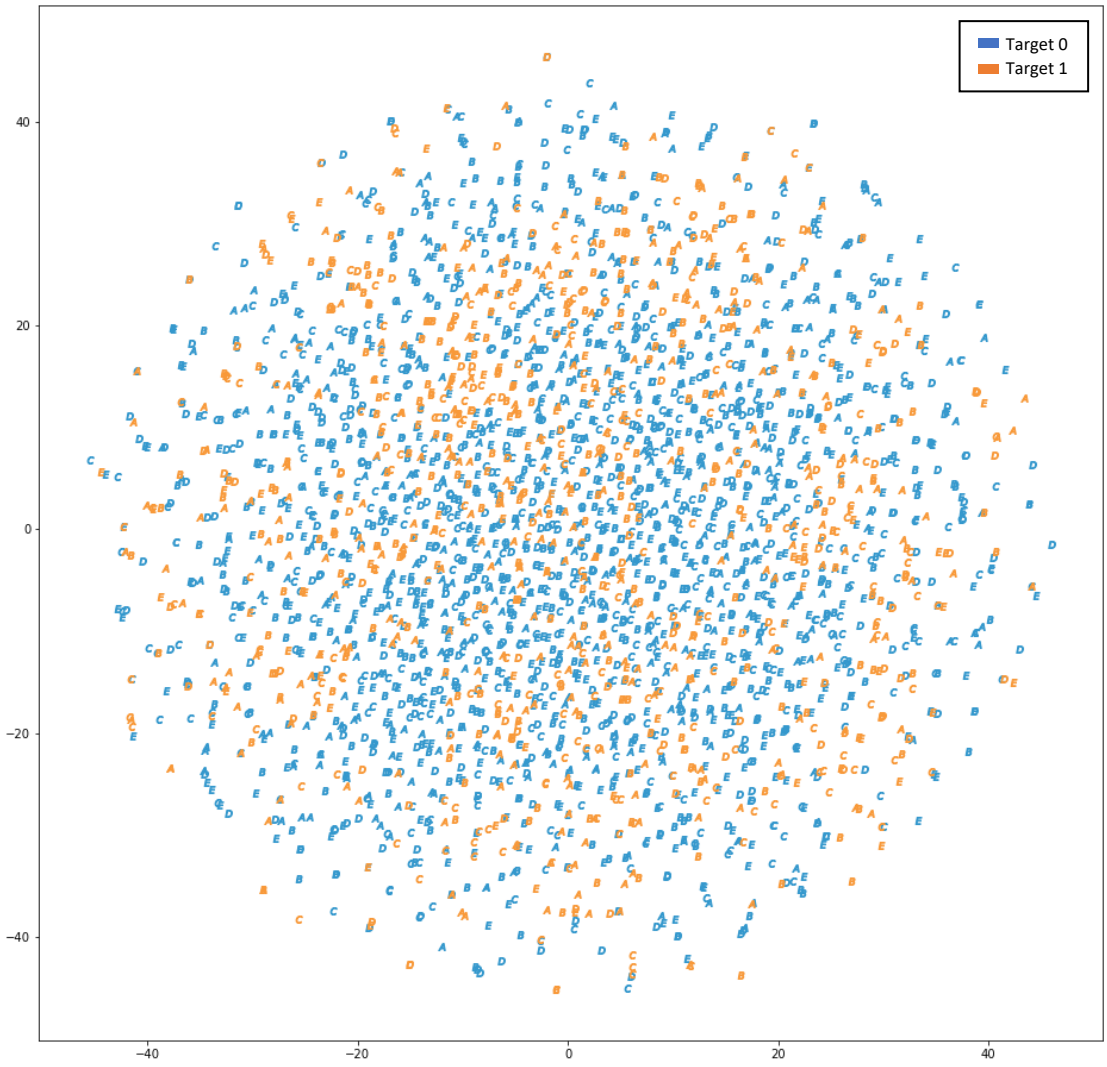
The following similarity matrix uses provided data and instead of prominent divisions inside the matrix, we can observe very faint separations. However, this does demonstrate that datapoints within their respective target label do indeed have similar characteristics. The correlation metric used was Pearson's Correlation.



Pearson's Correlation Similarity Matrix

t-SNE Visualization

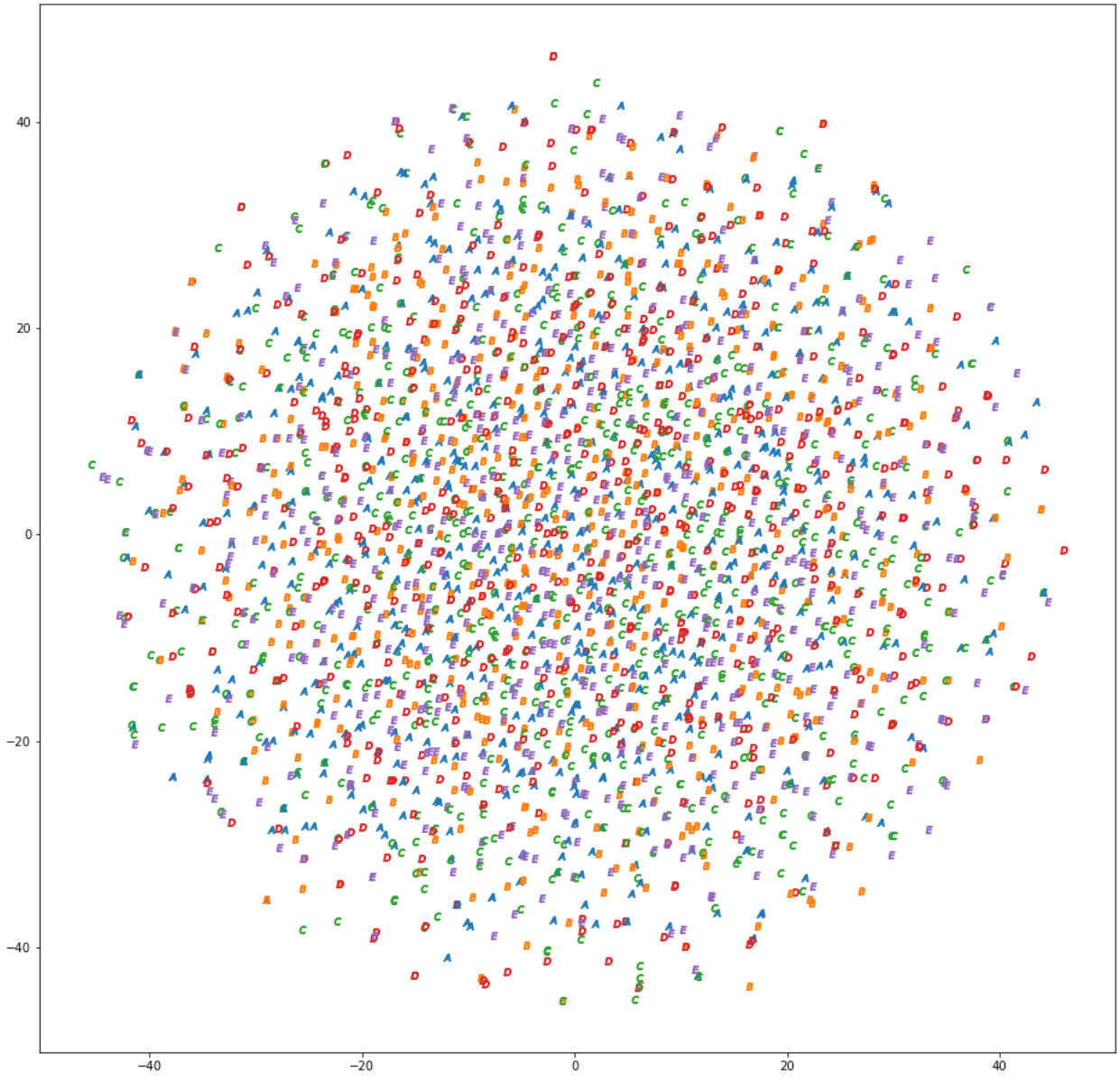
In this demonstration, I attempt to see if there are visually identifiable clustering of datapoints with the same target label. I first needed to reduce the high-dimensional features into 2 feature values in order to visualize the data easier. For this, I use t-SNE dimensionality reduction.



t-SNE Dimension Reduced Datapoints

Each point in the visualization is plotted using the 'XC' char value. Because I am interested in the grouping based on target label, I ignore the char value and observe any clustering based on color. Unfortunately, in this demonstration groupings cannot be distinguished and the spread is wide. It appears that reducing to two dimensions does not represent the data well enough to observe clusters, and the best chances will likely be to implement powerful non-linear methods (Neural Networks) to classify the data.

Due to the slight correlation of the ‘XC’ char feature with the target label, I also tried to observe any clustering of this char feature. Below is that demonstration.



t-SNE Dimension Reduced Datapoints Colorized by XC Char Feature

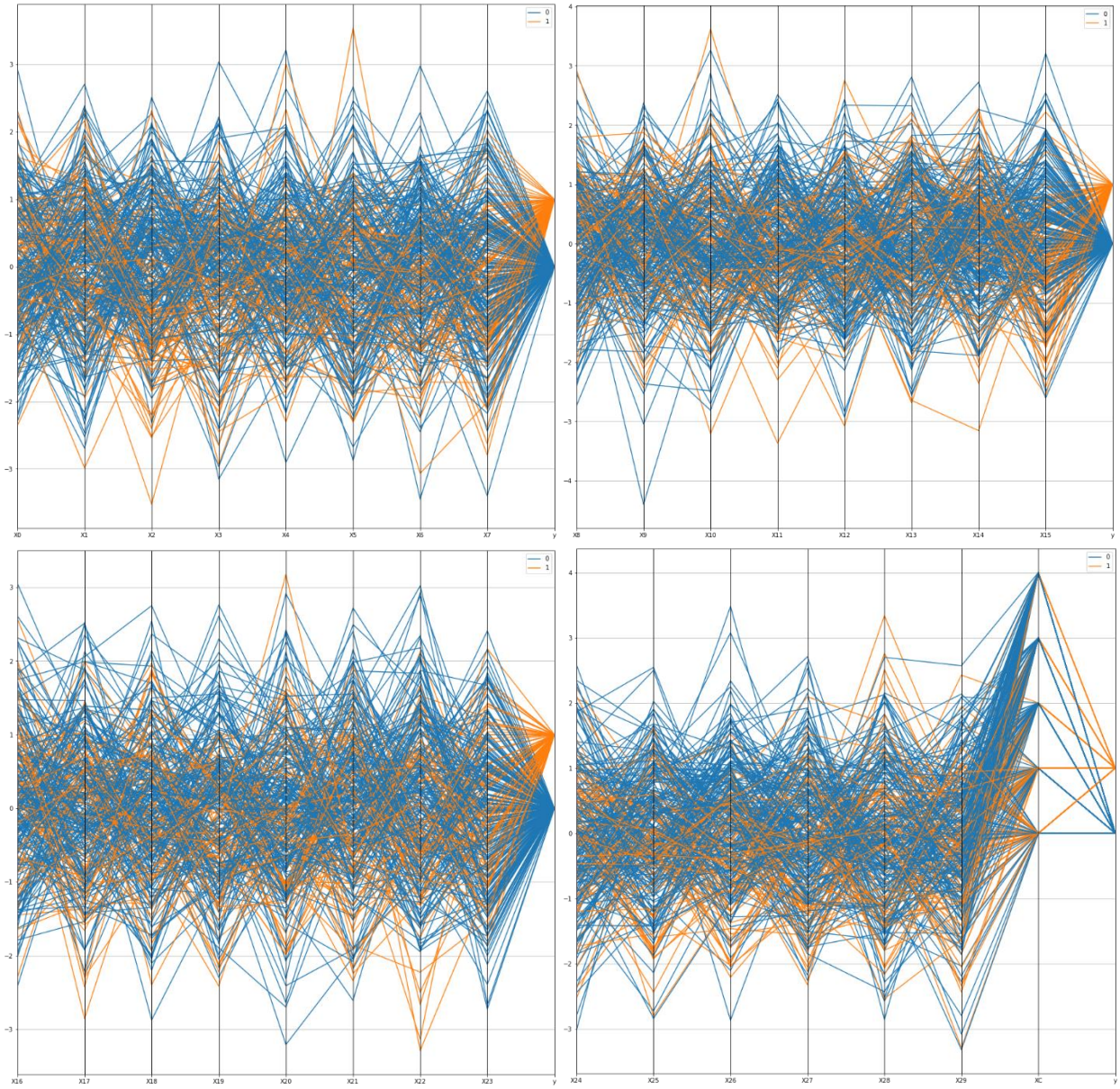
Unfortunately this demonstration also does not allow me to perceive any obvious clustering. This demonstration was done out of curiosity and for more completeness.

Parallel Coordinates

Parallel coordinates is another way to observe trends in features within target labels. To plot the ‘XC’ character values, the chars {A, B, C, D, E} were converted into their numerical indices. The mapping is as follow:

A = 0; B = 1; C = 2; D = 3; E = 4.

Due to the large amount of datapoints to be plotted on the graph, 10% from each target label were randomly selected to represent the entire dataset.



Parallel Coordinates of Features to Target Label

By representing the data in parallel coordinates we also get to view the data in the form of time series as well even if the features are not sequential. After observing the graphs, it is difficult to confidently conclude much. A rough observation would be that target label 0 features tend to have larger values, but overall, there does not appear to be obvious trends in feature values.

Modelling

I trained a total of five different models; four of which are neural networks. Also, to find better hyperparameters I used a grid search method which changed values of the amount of hidden layers, hidden units, convolution blocks, kernel sizes, number of channels, learning rate, include/exclude batchnorm and dropout layers, and the type of optimizer. After the grid searches were complete, I aggregated all the results to compare them all with each other. Here, I present my evaluation method, architecture, and best results for each model architecture.

Evaluation

I present here the method I used for evaluating the quality of a model.

The method I used was a k-fold cross validation with k = 5 which translates to an 80/20 train and evaluation split. After gathering results from all 5 experiments, the results were averaged and I particularly paid attention to the average weighted f1-score, average macro f1-score, and the accuracy.

The **f1-score** is often referred to as the harmonic mean between precision and recall. This means I am trying to reduce both false positives (precision) and false negatives (recall).

The weighted f1-score is useful because of the class imbalance present in the training set. It takes the f1-scores of each class independently and then takes the weighted sum of each depending on the number of true labels (also known as support) of each class.

The macro f1-score does not consider the class imbalance and instead sums the f1-scores of each class without any weights.

Accuracy is the amount of correctly classified samples out of the entire sample set.

Models

The first model used was `XGBoost`. This model took as input all features including the 'XC' feature as a categorical encoding (the same mapping as in the parallel coordinates section). However, I quickly moved onto neural networks when I found significantly stronger results.

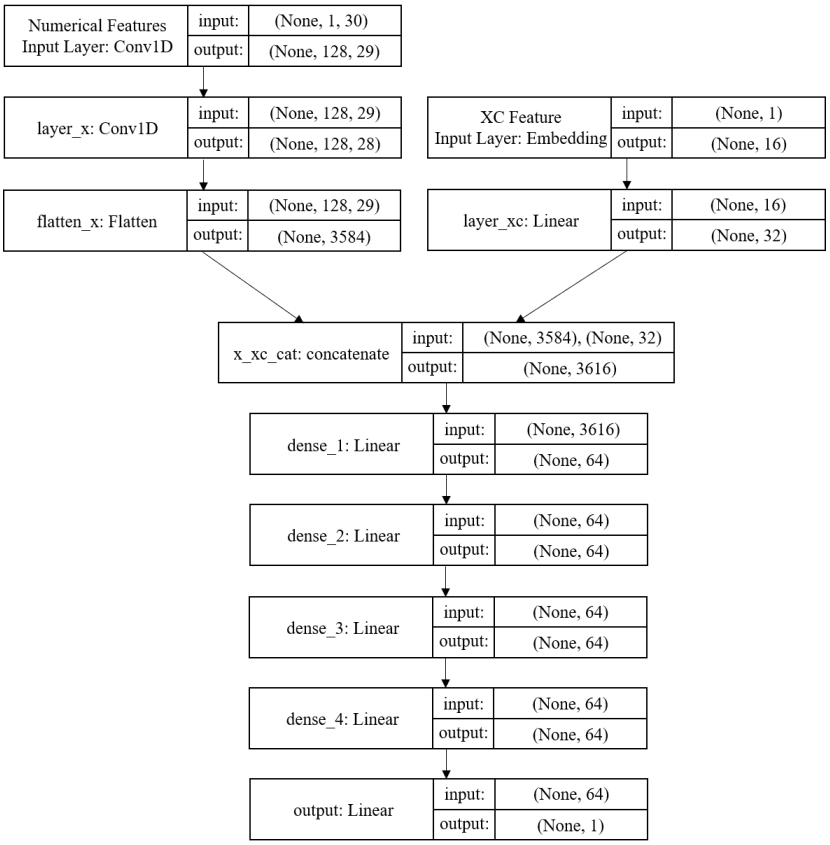
Neural networks are powerful for classification tasks. Weights and biases can be adjusted to give certain features more of an impact in deciding the class label. Also, convolution layers have been used to achieve many state-of-the-art results due to their powerful feature extraction capabilities. A large limitation of these methods, however, is that large datasets are required to train these models and can lead to overfitting due to the large number of tunable parameters.

The first neural network presented here is the `dense_no_xc` model. This model accepted as input only the numerical features, ‘X0’, ‘X1’, ‘X2’, …, ‘X29’, and excluded the ‘XC’ feature. This was to be used as a baseline because we discovered in the exploratory analysis that the ‘XC’ feature had the greatest magnitude correlation with the target label. This simple model was built using a series of fully connected layers.

The next neural network is the `dense_ohe` model. The letters ‘ohe’ stands for one-hot encoding in which the ‘XC’ feature was encoded and accepted as part of the input along with the numerical features. Results showed improvements over the `dense_no_xc` model. This model was built using a concatenation and a series of fully connected layers.

Another neural network is the `dense_embed` model. Like the `dense_ohe` model, an embedding layer was used instead. Using an embedding layer showed even more improvements. This model was also built using a concatenation and a series of fully connected layers.

The last neural network trained was the `conv1d_embed` model. This model also accepted the ‘XC’ feature as input through an embedding layer. Also, this model uses 1D convolutions to extract features from the numerical inputs. After the extraction, the features from both the embedding layer and numerical inputs are concatenated together and further processed through fully connected layers. A diagram of the architecture is provided below. This architecture was able to achieve the best results I could find out of all the models tested.



*Best Performing **conv1d_embed** Model Architecture*

Results

As aforementioned, the best results were acquired from the `conv1d_embed`. Recall that this means this model was the best at reducing both false negatives and false positives.

The best results for each model are shown below.

Model	Weighted F1-score	Macro F1-score	Accuracy
XGBoost	0.899556	0.880976	0.901
dense_no_xc	0.890284	0.871198	0.890
dense_ohe	0.958849	0.951552	0.959
dense_embed	0.969950	0.964726	0.970
conv1d_embed	**0.974340**	**0.969737**	**0.974**

Reproducibility and Reflection

The training sessions and parameter initializations use a preset seed. This means that any users that would like to reproduce these results may do so deterministically. The command below will reproduce the `conv1d_embed` model that reached the best metrics.

```
…WellsFargoChallenge2020/src> python train.py --model=conv1d_embed --lr=1e-4 --epochs=128 --
conv_blocks=1 --filters=128 --hidden_layer=3 --hidden_units=64 --kernel_size=2
```

It can be difficult finding and tuning parameters of a neural network. I encountered many models that failed significantly during my grid searches. However, when the right parameters are set, the neural network can be very powerful. Some real world applications of this kind of model would be tasks such as 1D signal classification for subjects like human movement, earthquake magnitude classification, or voice classification. Convolution layers in general are known to be highly adaptive and work extraordinarily well; it's recent successes motivated me to use them in my experiments.

Aggregated results and hyperparameters from the total of 2346 experiments were recorded and can be found in:

`…WellsFargoChallenge2020/reports/aggregate_experiments.csv`