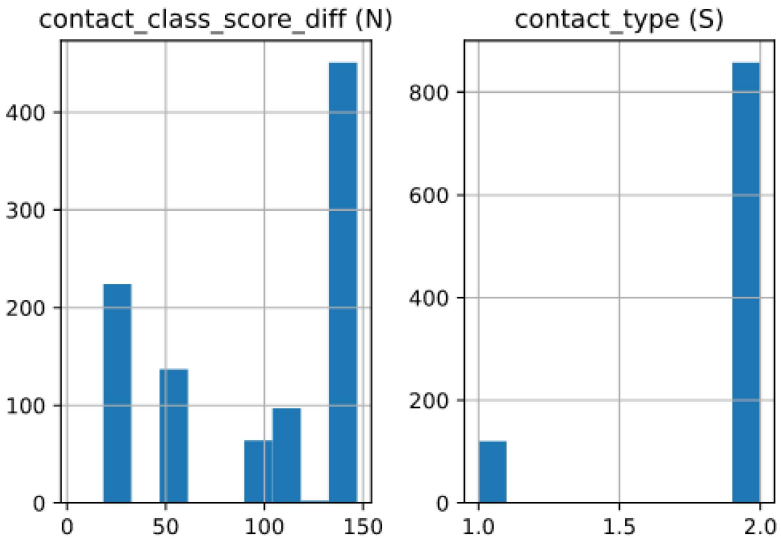
CS 6375 Final Project Report

Rutvij Shah and Hallie Nell

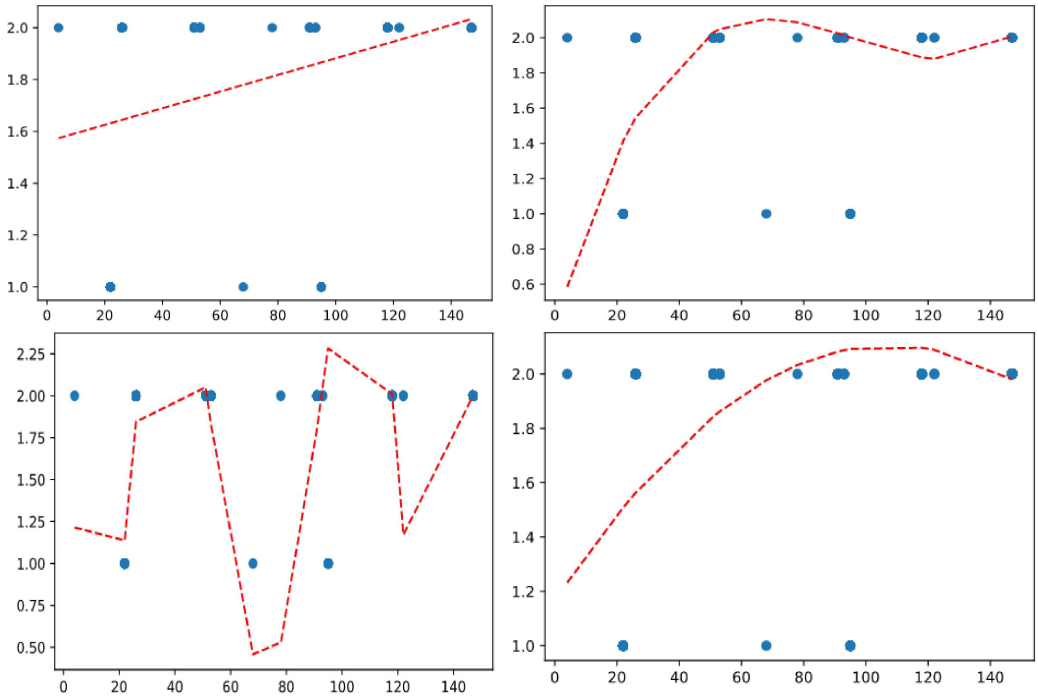
Main Goals: We wish to explore how a small feature set and high class imbalance affects various models, explore different techniques to handle class imbalance, and investigate potential discrepancies in the data (such as whether or not a device is faulty).

Data Set Description: For this project, our dataset comprises of contact tracing data from CareBand's wearable device for patient care and was collected over a period of two hours from a single device. Our dataset is composed of 978 data points. This data is gathered by using two devices, a "reporting device" and a "contacted device", both of which have a Device ID. The type of contact falls into one of two categories: close (under 6ft) and proximate (more than 6ft). The contact type will serve as our classification label with close contact considered to be our positive class. A timestamp of when the contact was reported, the on device ML score of the contact classification, and counter are included as features. Finally, we had a feature called delay. To proceed with the classification, the device IDs were converted into integers and the features counter and delay were discretized to lessen the amount of points in the feature space and because our custom code is optimized to handle discrete features.

Preliminary Analysis of the Data: This problem is a Binary Classification where close contact, referred to as type 1, is our positive class. Two things are immediately obvious when looking at our data set; some features may be irrelevant, resulting in a small usable feature set, and there is a high class imbalance: 120 positive examples vs 858 negative examples. For the former, we will evaluate whether or not some of the attributes affect the classification. Next, we will address the class imbalance, which is visualized to the left. Our approach to handling class imbalance is threefold. One, we will use the re-sampling techniques of random over-sampling, random under-sampling, tomek links, SMOTE, and near miss. The results of these techniques will be compared to each other and to the results with no re-sampling. Second, we will be leveraging various ensembles' ability to handle class imbalance. Last, it involves penalizing false negatives by weighing the positive-minority class higher. Penalizing the algorithms' mistakes on the minority class is a useful tactic because in this scenario, the cost of mislabeling a close contact is much higher than the cost of mislabeling a proximate contact, and we want our model to reflect this fact.



Next, some simple curve fitting was done using a linear function and a couple of polynomial functions with different degrees.

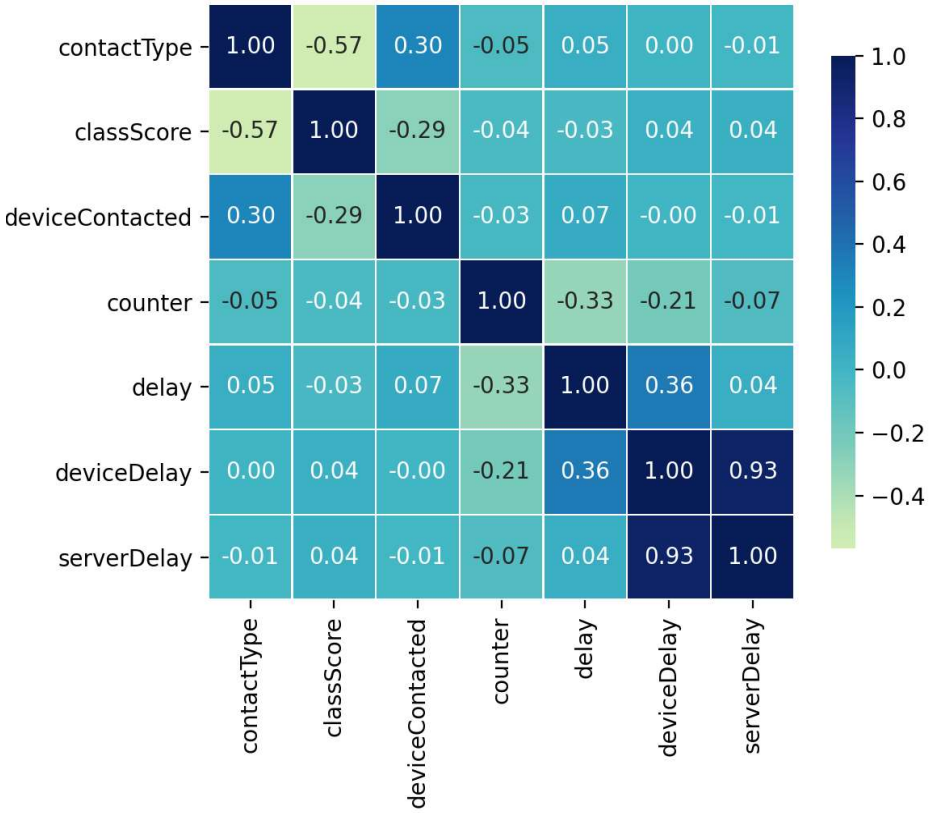


Clearly, the data is not linearly separable. Thus, we need to choose models that can handle complex decision boundaries

Next, we used a correlation matrix to examine feature importance.

We can see that some of the features have little to no relationship to the contact type.

It seems like only 2 features are correlated to our classification label



Potential Discrepancies in the Data:

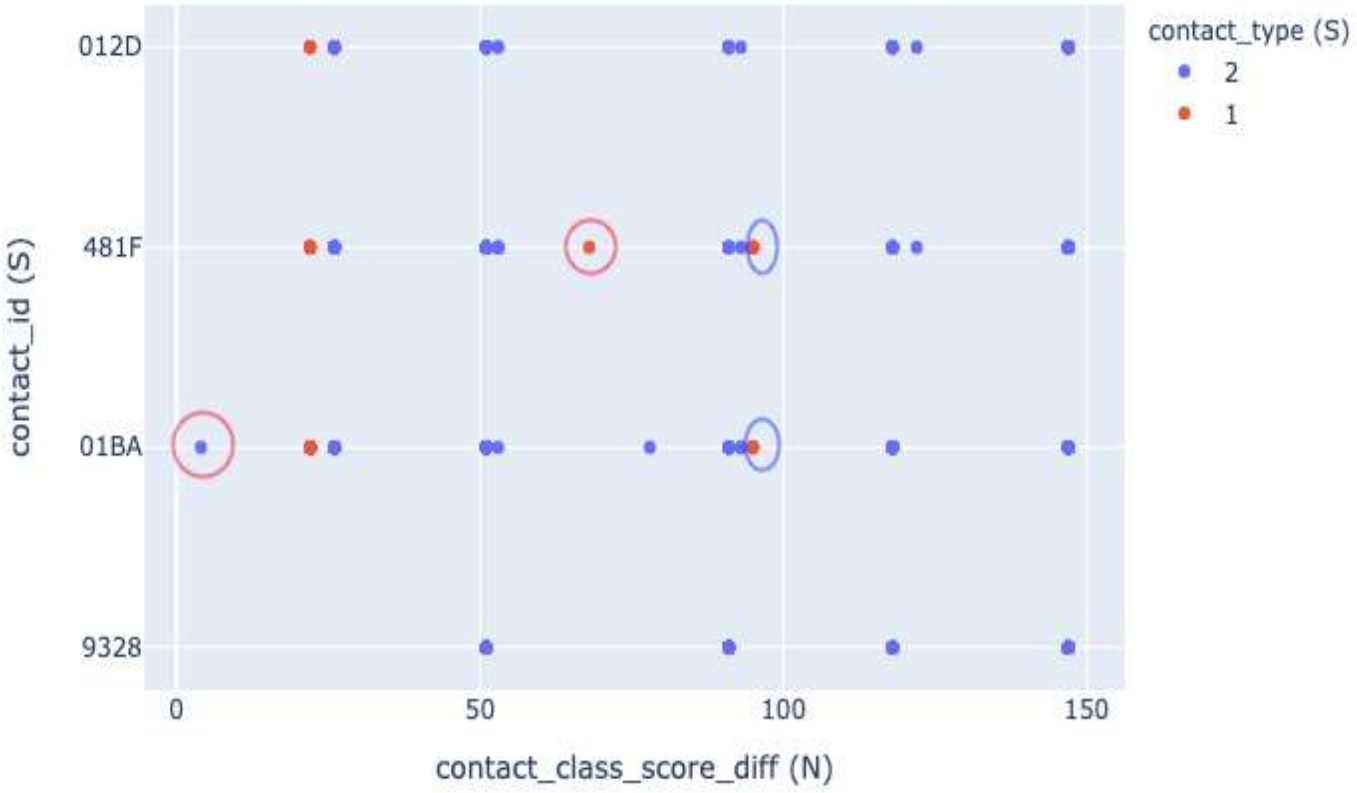
Here are the contact classification scores present in each class and their frequencies. Notice how the values of 78, 68, and 4 only occur once. This makes us suspicious that these outliers may be noise.

```
Set of Diff values for Type 1 contacts {68, 22, 95}
Set of Diff values for Type 2 contacts {4, 122, 78, 51, 147, 53, 118, 26, 91, 93}
Set intersection: Empty Set
Class distribution 1: 120 2: 858
```

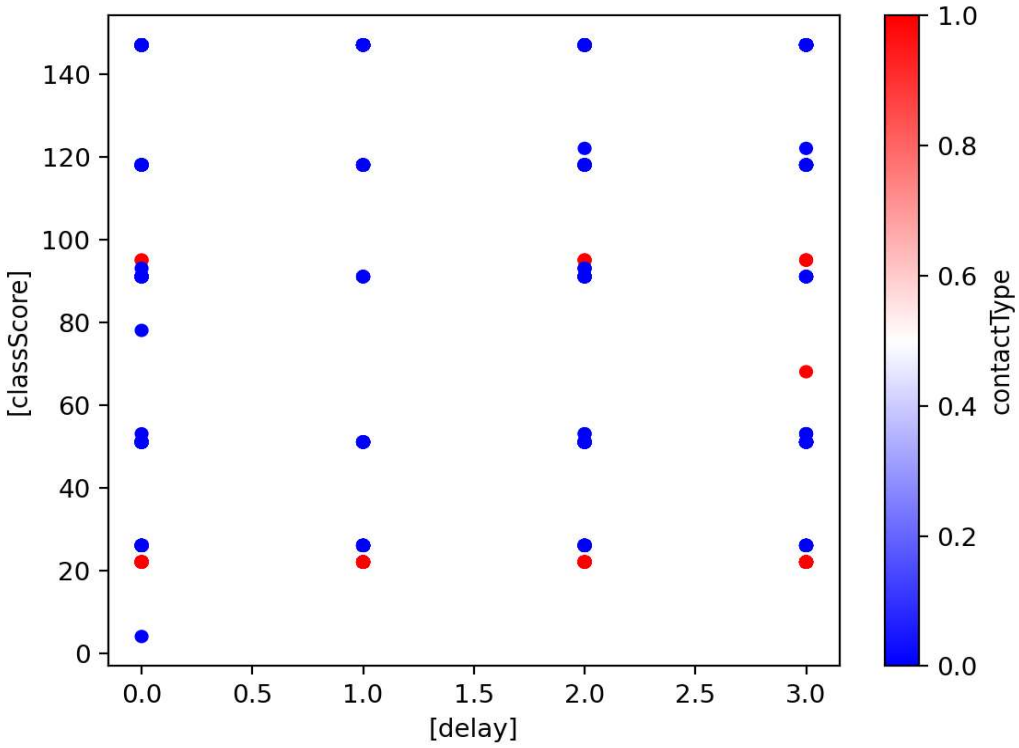
```
Number of times each difference value is observed {91: 52, 26: 113, 147: 451, 51: 130, 118: 97, 22: 111, 78: 1, 53: 7, 95: 8, 93: 4, 122:
2, 68: 1, 4: 1}
```

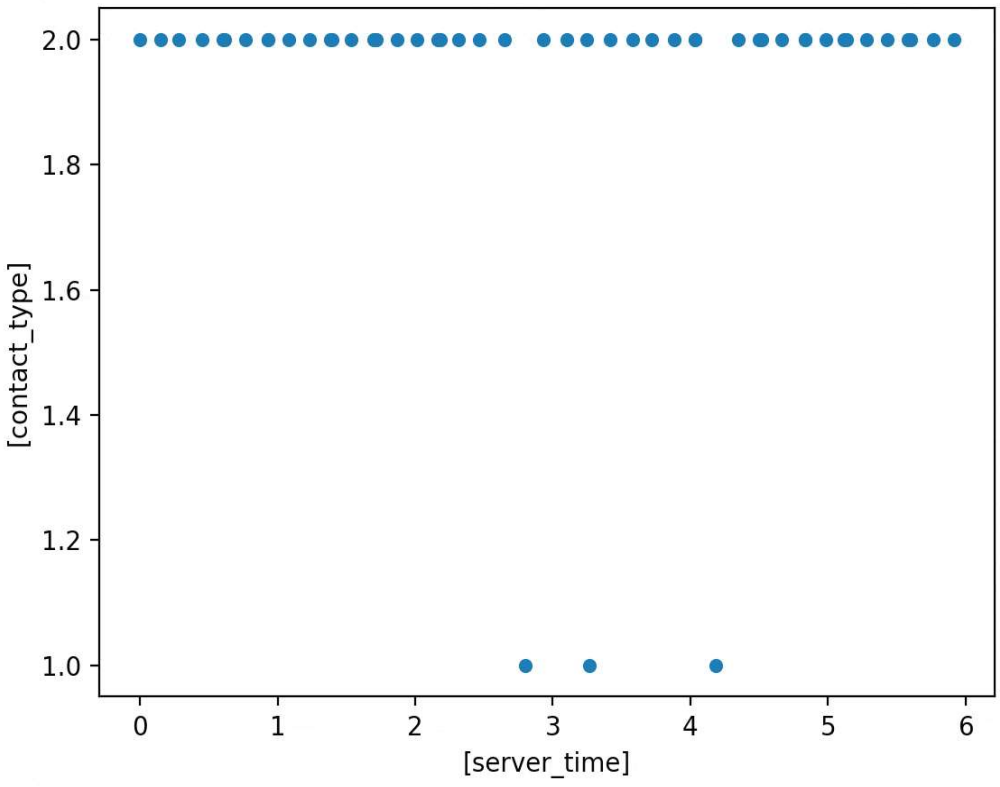
On the plot below of the most relevant feature vs the device with which the contact was detected, we mark close in red and proximate in blue. As we can see, the data seems to follow a pattern which makes us aware of potential outliers; the points circled in red are low frequency (1 obs each) and thus have a higher likelihood of being outliers than those circled in blue, which have a relatively higher frequency. The image below allows us to visualize the outliers with contact class scores of 4 and 68.

This graph illustrates that the device IDs don't seem to dramatically affect contact type.



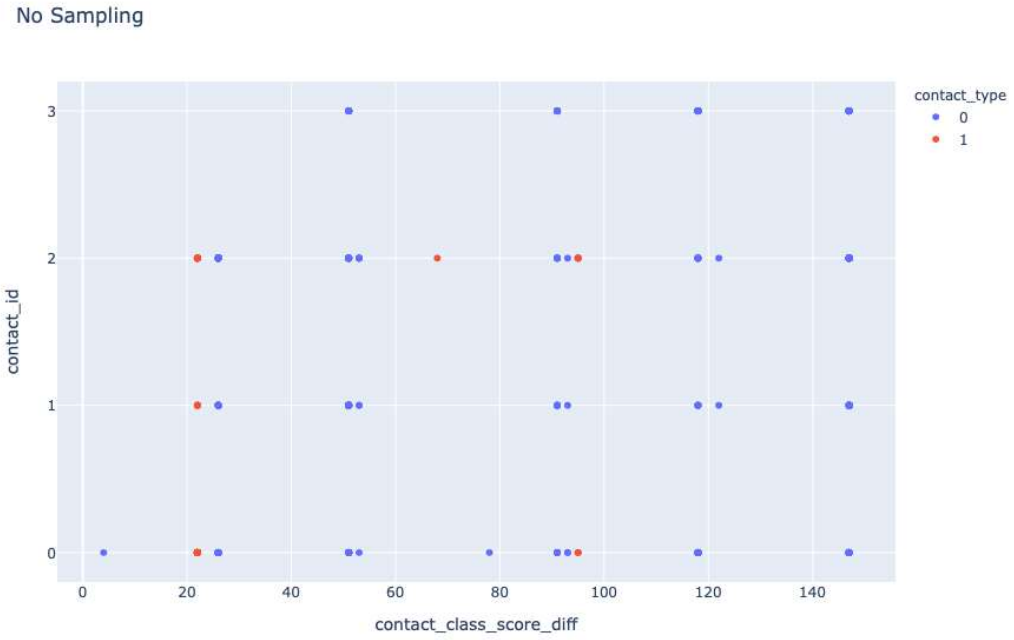
We hypothesized that a high delay value might be related to the potential outliers; thus, we produced the following graph. One of the outliers does have a high delay of category 3. However, we believe this is merely a coincidence because other data points that we would not identify as outliers also have the same delay. Furthermore, the other outlier has one of the smallest delays. In short, we do not have enough evidence to substantiate that hypothesis. From this image we can see that contact type is not affected by the delay.





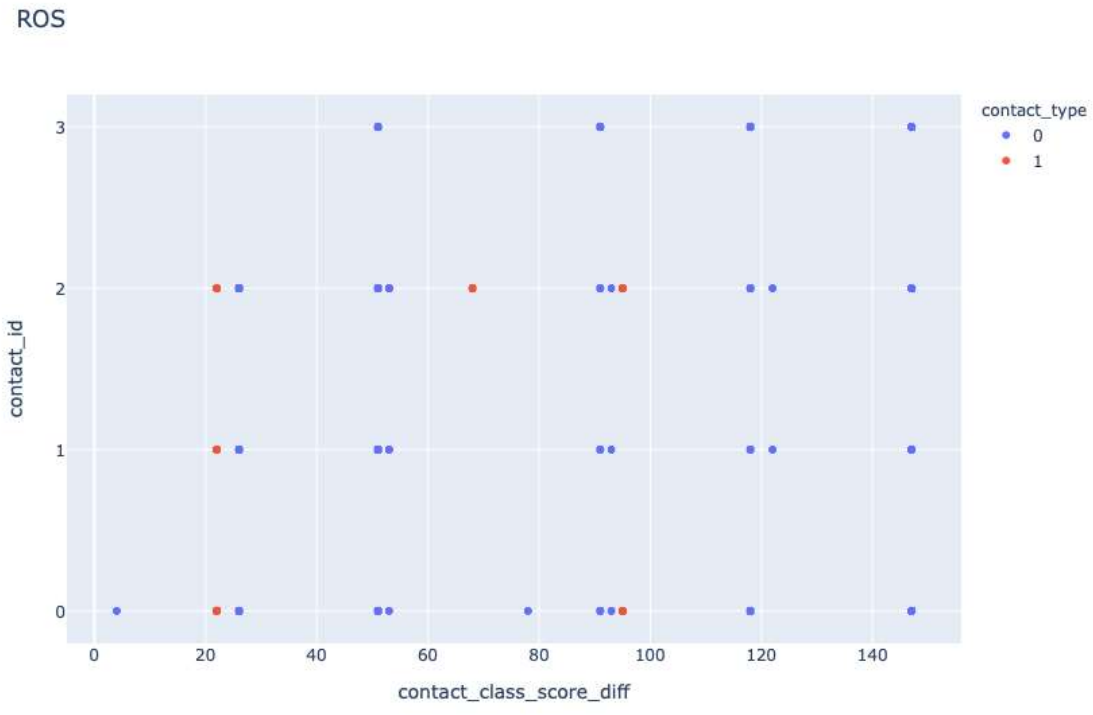
Simple Time-Series Analysis: To the left is a GIF displaying the contact-type vs server time. Here we can watch for when close contact occurred.

Data Processing: The sampled data was split into training and test sets at a 70:30 ratio. Additionally, the raw data was also split into 8 validation sets using the stratified cross validation technique. Stratification was employed to preserve the class distribution in each of the folds. We chose to have 8 folds as opposed to the usual 10 because our dataset is only of size 978. Using a 10-fold cross-validation method would result in a test set with only about 98 samples, which felt like an inadequate size. The folds were further resampled using random under and over sampling methods. The purpose of the validation sets is to demonstrate the skill of the classifiers employed and to counter the relatively small dataset.

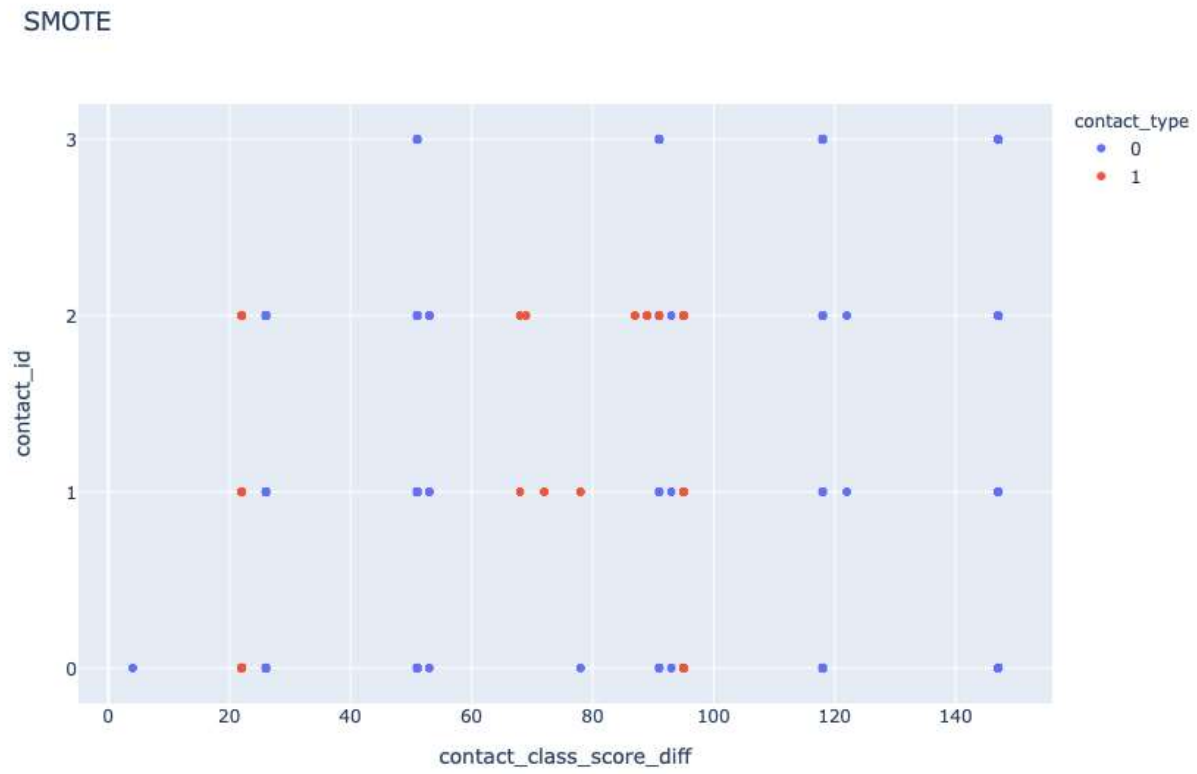


The sampling techniques used were as follows:

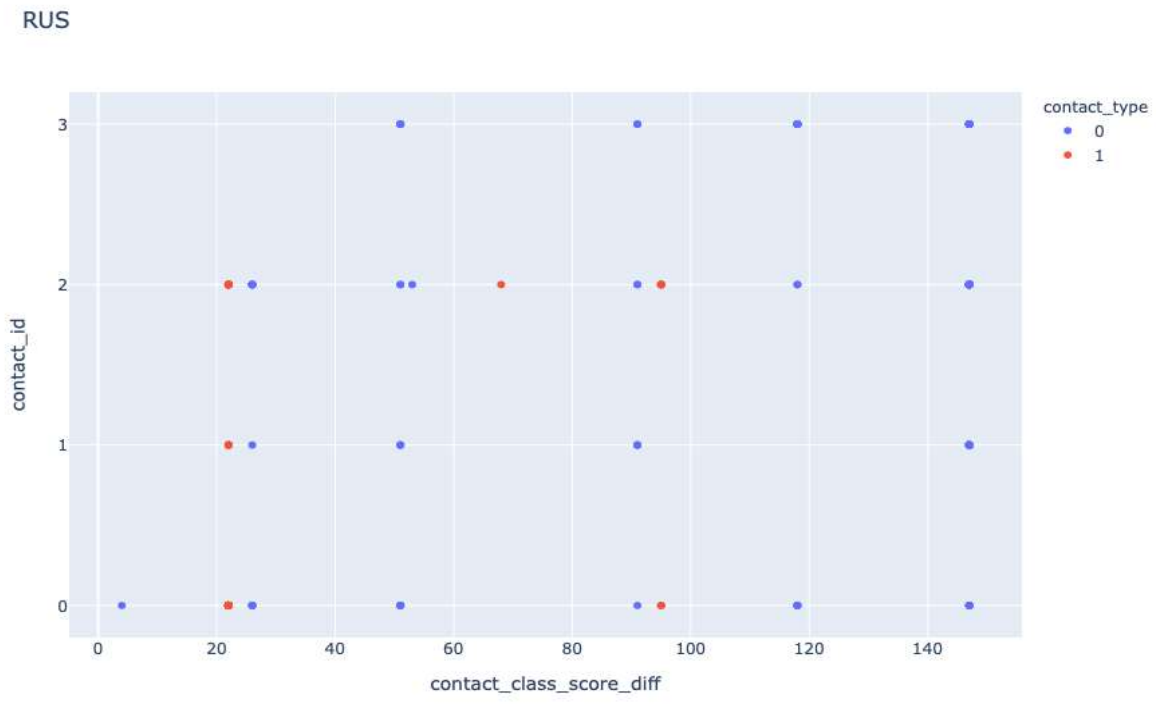
- Random Over-Sampling - This involves duplicating instances of the minority class to even out the class distribution.



- SMOTE (Synthetic Minority Oversampling Technique)- This technique synthesizes new data points for the minority class. The general approach to doing so involves selecting k-nearest neighbors of the minority class and interpolating points between a chosen point and its neighbors.

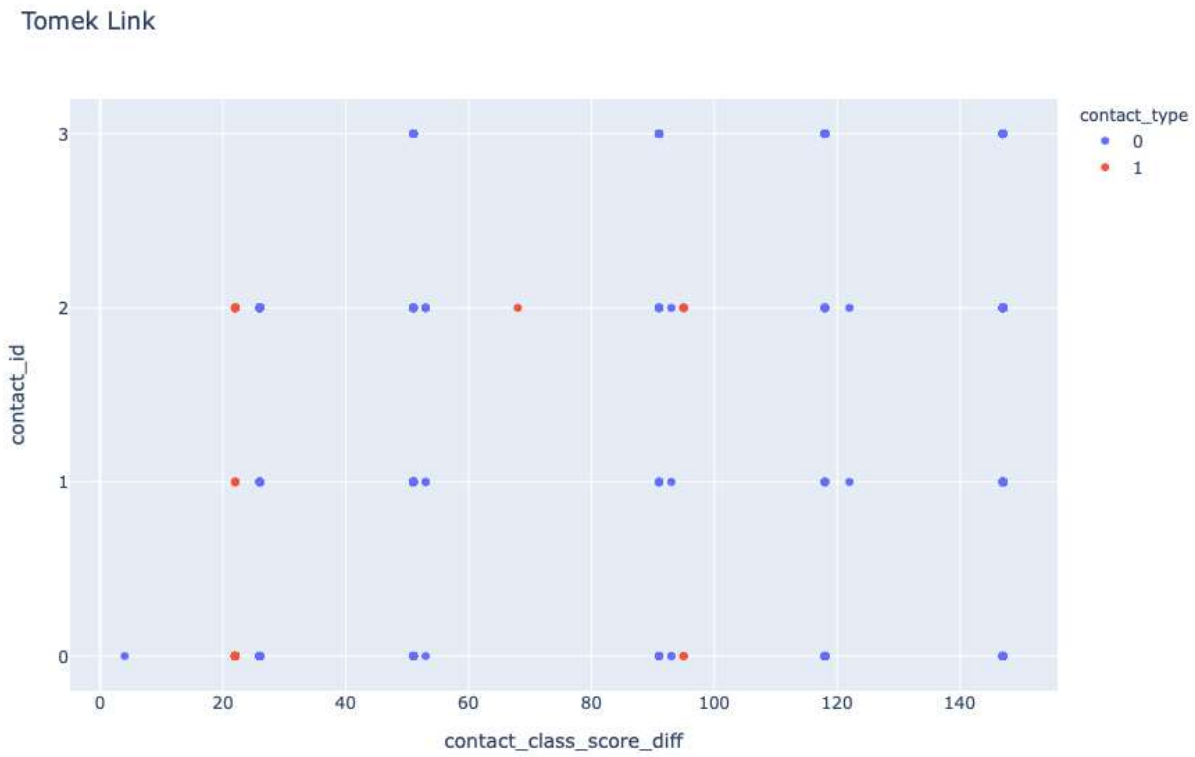


- Random Under-Sampling - This technique discards data points in the majority class to have the same number of instances as the minority class. This technique leads to data loss.

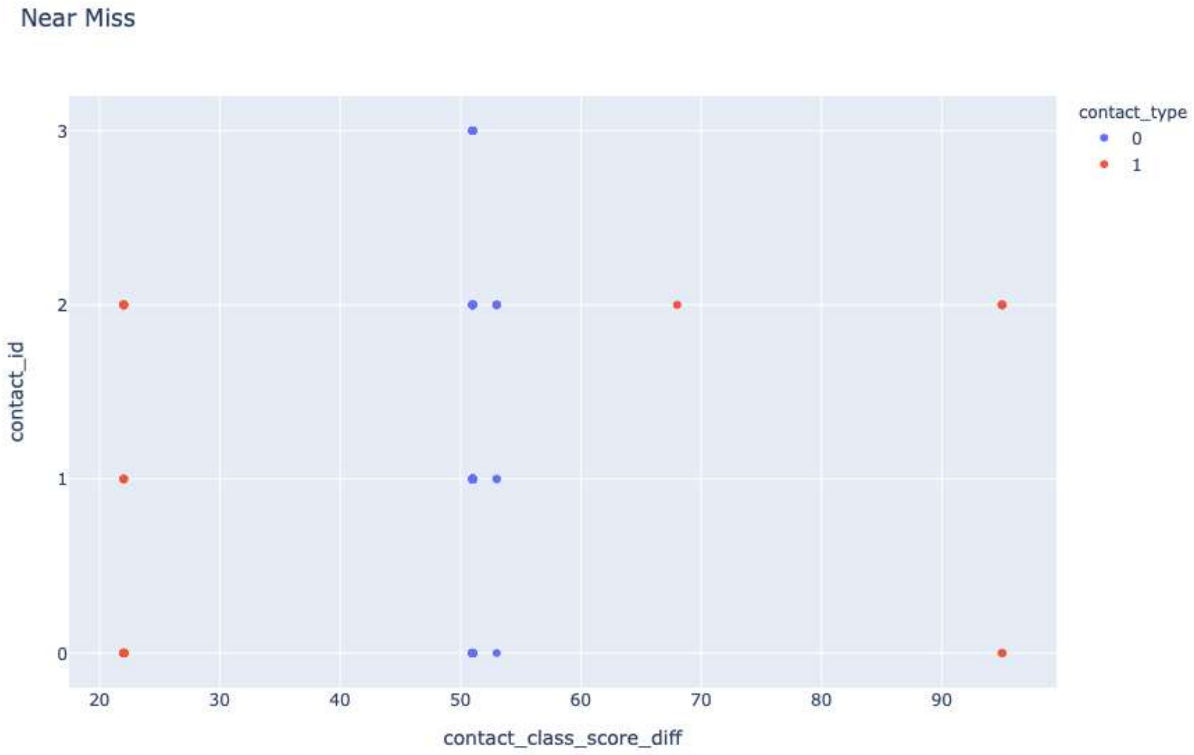


.

- Tomek-Links - A Tomek-Link is defined as an instance of the majority class and the minority class such that the two are nearest neighbors of each other. For such instances, the majority class instance is deleted to facilitate a smoother class boundary. In our dataset, only one Tomek-Link was identified.



- Near Miss - There are three heuristics utilized by this sampling technique. Our initial intention was to use NM-3, which involves two steps. Step 1 selects kNN from the majority class for each minority class example. Step 2 selects that majority class example to retain from the k neighbors which has the smallest average distance to the minority class point. Further research showed that this will retain points closer to the decision boundary and improve precision at the cost of recall. Since our objective was to improve recall, we chose the version which demonstrated best recall in the original paper for Near Miss. This was NM-2, which chooses majority class examples closest to all the minority class examples, by taking those majority class points with the smallest average distance to three of the furthest minority class examples. The intention being to get a cluster of the majority class. While the precision might drop, recall is improved by having a lower likelihood of a positive-minority point being classified as a negative-majority point.

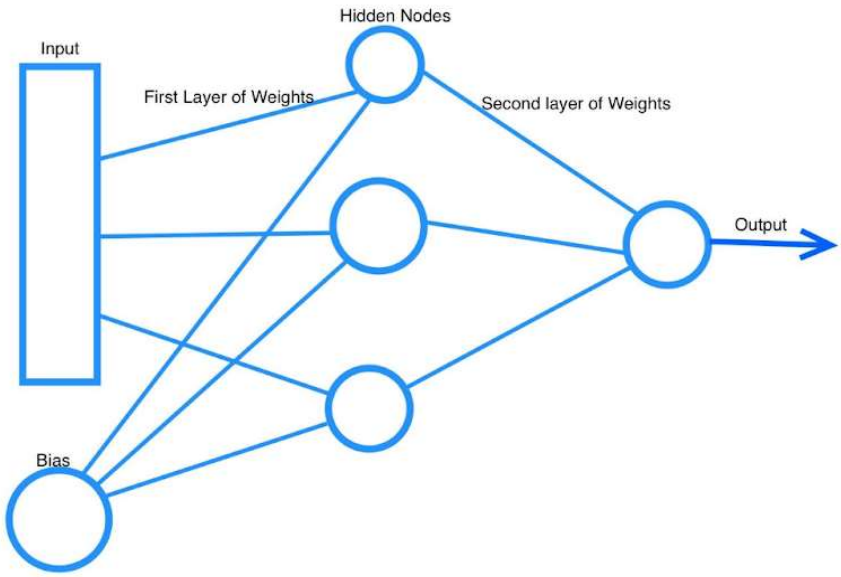


Descriptions of the Models Used: Given that the dataset isn't linearly separable and doesn't seem to be modeled by a lower order polynomial, we chose to implement models that are nonlinear and thus afford more flexible decision boundaries. Our models of choice are Neural Nets and Decision Trees. Neural Nets can approximate any function to arbitrary precision, and Decision Trees employ axis parallel rectangles to encapsulate a feature space that is non-linear and complex. One of the drawbacks of regular decision trees is their tendency to overfit with increasing depth, i.e. low bias with high variance. With respect to our data, there is an additional drawback. Since the number of feature value pairs are substantial for the most relevant feature, decision trees will tend to capture the most common points for that feature (the contact class score) and disregard other features' minor influences and the lower frequency points. To combat this, we used bagging and boosting. Bagging helps reduce variance with a minimal cost to bias. While boosting will use decision stumps (high bias, low variance), for which it will initially reduce bias and eventually affect variance reduction as well. Though our data has outliers, there is a relatively small percentage of them, and thus, we expect bagging and especially boosting to capture them without any adverse influence on the predictions.

We further expect ID3 binary decision trees, and the ensembles based on them, to help combat feature irrelevance since information gain will prioritize features which are more indicative. Since ID3s split on feature-value pairs, they'll increase the potential splitting points, as compared to splits on feature thresholds helping us tackle feature paucity. Another possible advantage of using boosted decision stumps could be that since the data would be susceptible to underfitting given the lower number of features, stumps could provide the low bias learners that can be effectively boosted.

Neural Net:

Neural networks are universal function approximators, A neural network can approximate any function to a high degree. Thus, we expected the model to have high performance on our complex dataset. We created a simple, fully connected neural network with one hidden layer composed of three nodes and an output layer of one node. Note, this is not a single-layer network because the input passes through two sets of weights and is not connected directly to the outputs through a single layer of weights.



The algorithm used Batch Gradient Descent with the addition of RMSProp, which uses an exponential moving average to create an adaptive learning rate. The loss function used during the gradient descent was mean squared error. The sigmoid, tanh, and ReLu activation functions were explored. The sigmoid activation function produces the estimated probability that the input belongs to the positive class. The larger the probability, the more likely the data point is to be in the positive class. In our implementation, the sigmoid activation function was assigned a classification threshold of 0.5. This threshold describes what the minimum probability of a data point being positive has to be before the classifier will predict said point to be positive. Since we use re-sampling techniques to balance the dataset, we decided to make the threshold 0.5. For a similar reasoning, the tanh activation function was given a classification threshold of 0 since the range of tanh is [-1,1]. We were getting unsatisfactory results with our implementation of the neural network, which will be discussed below, so we decided to attempt to implement the Binary Cross-Entropy(BCE) Loss function because this function reflects the notion that the neural network should pay a high penalty when the estimated probability is completely wrong.

Results of the Models: When analyzing results, it is important to note that accuracy rate alone may be misleading when evaluating our models. This is because the classifier could predict the majority class every time without using any of the features, and it will still have an accuracy rate in the 80s. Based on the recall values, we can see how accuracy is a spurious metric in this instance.

Our Implementations

Comparing the Different Models to Each Other

	Unsampled	NearMiss	ROS	RUS	SMOTE	Tomek Links
Bagging #Bags = 5 Max depth = 3	039 | 001 000 | 254 Precision: 1.0 Recall: 0.975 F1: 0.987	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	274 | 000 000 | 241 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	274 | 000 000 | 241 Precision: 1.0 Recall: 1.0 F1: 1.0	030 | 000 000 | 264 Precision: 1.0 Recall: 1.0 F1 Score: 1.0
AdaBoost #Stumps = 10 Max depth = 1	039 | 001 000 | 254 Precision: 1.0 Recall: 0.975 F1: 0.987	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1: 1.0	274 | 000 000 | 241 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	274 | 000 000 | 241 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	030 | 000 000 | 264 Precision: 1.0 Recall: 1.0 F1 Score: 1.0
Decision Tree with Entropy Max Depth 10	000 | 040 000 | 254 Precision: 0 Recall: 0.0 F1: 0	000 | 034 000 | 038 Precision: 0 Recall: 0.0 F1: 0	073 | 201 050 | 191 Precision: 0.593 Recall: 0.266 F1: 0.368	000 | 034 000 | 038 Precision: 0 Recall: 0.0 F1 Score: 0	191 | 083 151 | 090 Precision: 0.558 Recall: 0.697 F1: 0.620	000 | 030 000 | 264 Precision: 0 Recall: 0.0 F1 Score: 0
Neural Network with Sigmoid Activation and step size = 0.01	040 | 000 254 | 000 Precision: 0.14 Recall: 1.0 F1 Score: 0.239	034 | 000 038 | 000 Precision: 0.47 Recall: 1.0 F1 Score: 0.641	274 | 000 241 | 000 Precision: 0.53 Recall: 1.0 F1 Score: 0.694	000 | 034 000 | 038 Precision: 0 Recall: 0.0 F1 Score: 0	000 | 274 000 | 241 Precision: 0 Recall: 0.0 F1 Score: 0	000 | 030 000 | 264 Precision: 0 Recall: 0.0 F1 Score: 0
Neural Network with tanh	000 | 040 000 | 254 Precision: 0 Recall: 0.0	000 | 034 000 | 038 Precision: 0 Recall: 0.0	274 | 000 241 | 000 Precision: 0.53	000 | 034 000 | 038 Precision: 0 Recall: 0.0	000 | 274 000 | 241 Precision: 0 Recall: 0.0	030 | 000 264 | 000 Precision: 0.10

Activation and step size = 0.01	F1 Score: 0	F1 Score: 0	Recall: 1.0 F1 Score: 0.694	F1 Score: 0	F1 Score: 0	Recall: 1.0 F1 Score: 0.185
Neural Network with Relu Activation and step size = 0.01	000 | 040 000 | 254 Precision: 0 Recall: 0.0 F1 Score: 0	034 | 000 038 | 000 Precision: 0.47 Recall: 1.0 F1 Score: 0.641	000 | 274 000 | 241 Precision: 0 Recall: 0.0 F1 Score: 0	000 | 034 000 | 038 Precision: 0 Recall: 0.0 F1 Score: 0	274 | 000 241 | 000 Precision: 0.53 Recall: 1.0 F1 Score: 0.694	030 | 000 264 | 000 Precision: 0.10 Recall: 1.0 F1 Score: 0.185

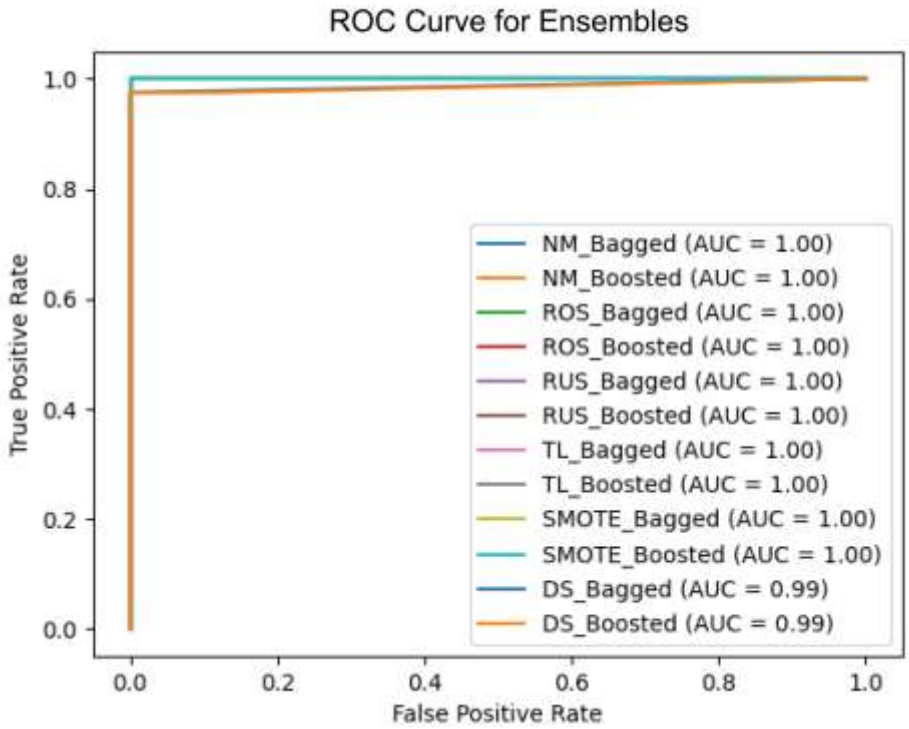
Simplified Overview

Test Set Results (Un-sampled Data)

Note: 86.39% of the data is in the negative class!

Models	Accuracy	Recall
adaBoost	99.66%	97.5%
Decision Trees	86.39%	0%
Neural Networks	86.39% or 0%	0%
xgBoost w/ weighted classes	99.66%	97.5%
Bagging	99.66%	97.5%

Recall since False Negatives for close contacts are costly.



As is evident from the confusion matrices and the ROC curves, both the ensemble methods performed exceedingly well with the sampled dataset and have an AUC of 1. In case of the unsampled data, both are unable to classify a single point, which drops the AUC to 0.99. Upon further analysis, we found that the point with the contact class score 68 is the one being misclassified since it is absent from the unsampled training set but is present in the test set. Either the sampling techniques used include the point via oversampling and is thus in both the test set and the training set. Or, in the case of undersampling, it is either in both or is only in the training set. Since the outlier is a member of the minority class, it is unlikely to ever be discarded via sampling and can either be overrepresented and thus become part of the data or is excluded from the smaller subset of the test sample. This is consistent with our observations. If there were multiple such points present, the performance of the ensembles could degrade depending upon other circumstances.

The expectation with Tomek Links was that it would identify the majority class point with the really low contact score difference of 4 as the link to be discarded. Instead, it identified the link between the minority class point 68 and the majority class point 78, with the latter to be discarded for undersampling. This is probably because of the fact that the distance between the majority point at 4 and it's nearest minority neighbor at 22 is higher than the distance from the minority point at 22 to its nearest majority neighbor at 24. That said, there are multiple points at both contact score 22 and 24, which would mean their true nearest neighbors are in fact points with the same class label at the same score value. Thus, only the singular points at 68 and 78 ever fit the Tomek link criterion.

With SMOTE, the synthetic minority points are generated in the region of contact class score values ranging from 58 to 95. This has the effect of making the potential outlier at 68 a part of the data and

making the negative datapoint at 78 a candidate for false positive classification with the models training on the dataset oversampled with SMOTE.

Near Miss sampling using its second version performs as expected and retains only those majority class points that cluster and are closest to the overall set of minority points. This is evident from the fact that the sampled points are a 'zoomed-in' portion of the entire feature space, and while all the minority points are retained, majority points with high class score difference (>52) and the one with the low class score 4 are disregarded entirely. This sampling technique leads to the highest amount of information loss and might not be appropriate whilst scaling this problem.

The decision trees performed abysmally on both the sampled and unsampled datasets. For the undersampled and unsampled data sets, decision trees predicted all the examples to be of negative class. In the case of using Tomek Links, only one point was reduced from the original set, thus the class imbalance was propagated. Thus, the DT for TL and unsampled data were expected to be the same, and they were.

Both had the form:
{(0, 147, False): 0, (0, 147, True): 0}

This shows that following the split on contact class score = 147, all samples are marked to be in the same class. The highest recorded contact class score was 147. Which is to be expected as the majority of the examples are negative and mutual information will see that for score values not equal to 147. Points correspond well to a point being negative just as they do for points with the score value of 147 (recall the 87.7% of the data is the negative class).

In the case of undersampling, our hypothesis is that the lack of data points combined with the small feature space doesn't provide adequate information for the decision trees to train on the data causing them to massively underfit.

For the sake of comparison, we used the decision tree we implemented for our first assignment, which afforded a node to split on multiple values of a single feature. That decision tree performed well on the dataset since there is one feature of importance, and the tree was able to capture all the score to label mappings. There are questions whether this approach would scale, and if it did, whether it would be efficient.

The Confusion Matrix on the Test Set for Depth of 1 :
034 | 000
001 | 037

Following is the decision tree of depth one which produced the results above. It has been condensed.

Decision Tree of Depth 1
Node value = None
Feature f1{68, 51, 147, 53, 22, 118, 26, 91, 95}

[{Node value = 68, Label value = 1}, {Node value = 51, Label value = 0}, {Node value = 147, Label value = 0}, {Node value = 53, Label value = 0}, {Node value = 22, Label value = 1}, {Node value = 118, Label value = 0}, {Node value = 26, Label value = 0}, {Node value = 91, Label value = 0}, {Node value = 95, Label value = 1}]

For the oversampled datasets, the binary decision trees see a minor improvement across all metrics, but they're still worse than random choice.

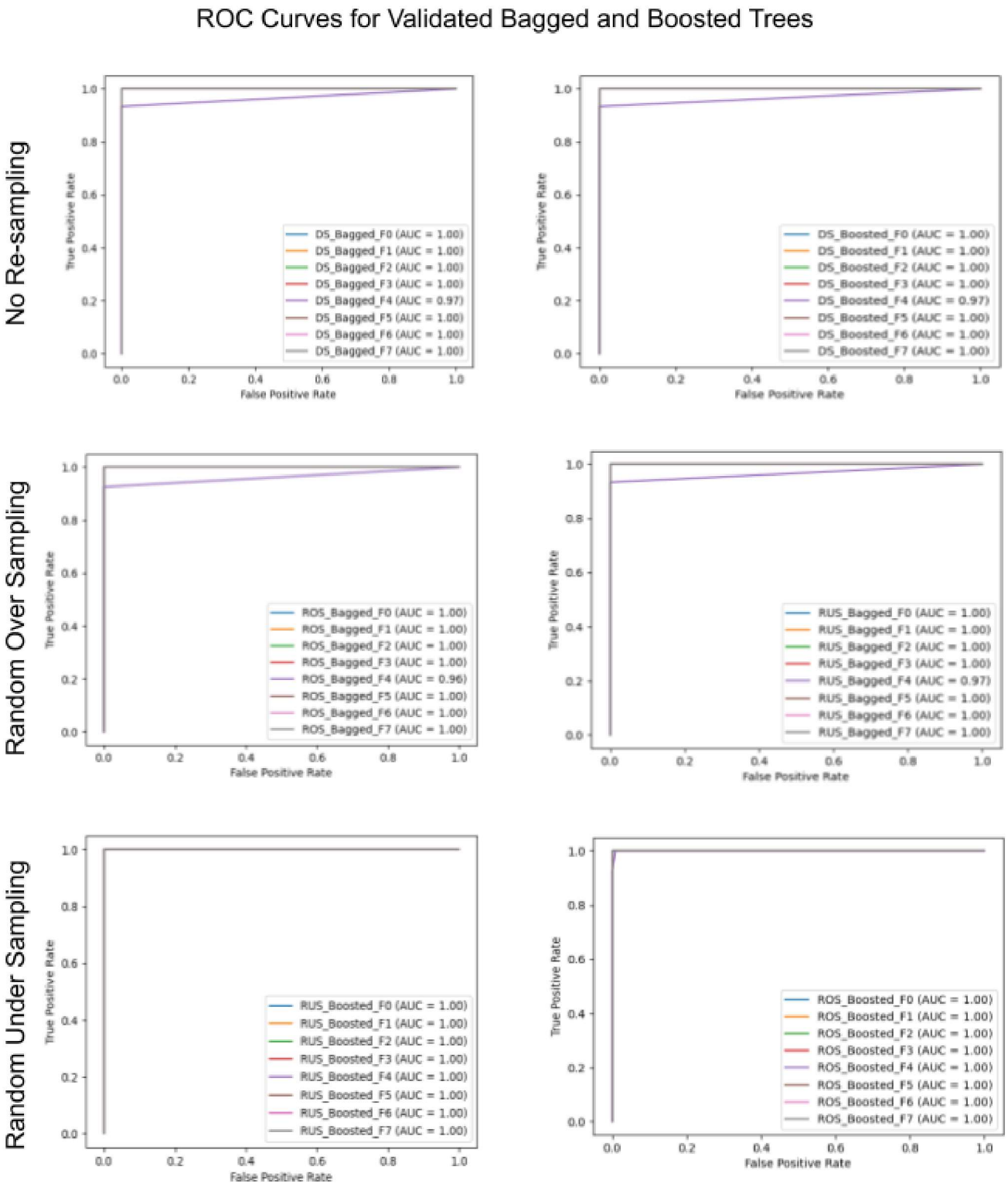
The tree generated by training on ROS is the only learner that has seen a greater correlation between 'counter' value and the class label. It predicts that for counter values in the third quartile, labels are true and the rest are false.
{(2, 2, False): 0, (2, 2, True): 1}

The tree generated by training on SMOTE is the only learner that sees a correlation between the delay classes and the class label. At level one it predicts data points with score value 26 to be negative, and at level two for data points not having score value 26. If their delay isn't in class 2 they're of the positive class. Once again, the performance is only slightly better than random choice.
{(0, 26, False): {(3, 2, False): 1, (3, 2, True): 0}, (0, 26, True): 0}

We suspect bagging does better since it has multiple learners each of which end up focusing on a different subset of the sampled data, allowing for the bagged ensemble to capture the various contact class score points separately. While in the case of boosting, the focus on data points that were labelled incorrectly meant the stumps are coaxed to learn the more difficult to classify points. In other words, the decision trees which were stump-like and underfitting had their bias reduced and variance contained to give accurate predictions.

While modifying the bagging and boosting algorithms from assignment 2 to output prediction probabilities, we noticed something peculiar. Since the probabilities were to be based on the votes of each of the learners, negative weights lead to negative probability values. Upon further investigation we realized that the learners in later iterations of adaBoost were being assigned increasingly negative values. Based on our understanding, that meant the weighted vote of that learner had a negative effect on the overall vote, and thus deteriorated the prediction capacity of the ensemble. Further reading on the topic revealed that our line of thinking is possibly true, and certain suggestions were made to address it. These included (a) stop learning when negative weights are encountered, (b) disregard the learner with negative weights and continue learning to see if the next learner's weight is also negative, and (c) retain the negative weight learners and complete learning. The last approach worked for us initially since the results were the same as the ones we obtained later, but to plot ROC-AUC curves, we needed prediction probabilities. Thus, we decided to drop the

learners which had negative weights. There was no change in the performance since the negative weights were fractions of the positive ones.



All of the ensemble methods performed consistently. The validations above demonstrate the skill of our ensemble methods. The area under the ROC curve is 1 or close to 1. Fold 4 is the only one not producing

AUC equal to 1. We found that the outlier with the contact score of 68 was in the test set of fold 4. Hence, when this point is in the training set, its effects are minimized by the ensemble techniques.

Discussion of Our Neural Network:

Obviously, there is an issue with our neural network implementation. Markedly, a recall of 0% shows that it only predicted the negative class correctly, which is detrimental to our model's predictive capabilities. Either every data point is classified as positive or every data point is classified as negative. Hence, the model is not effectively learning from the data. At first, we tried manipulating the initialization of the weights. We shifted them to be more or less negative. We thought that if all the weights were positive, this may be negatively affecting the tanh and ReLu activation functions since their classification thresholds were set to 0. This, unfortunately, did not mend the issue. Next, we speculated that maybe there were not enough hidden layers or enough nodes in those layers. Knowing the proper amount of both takes systematic experimentation. We tried increasing the number of hidden layer nodes to 5. After this produced no change, we explored what was occurring in the algorithm more deeply. The derivative of the cost function always had the same sign. A negative derivative of the cost function should mean the weights need to be increased. Hence, we now became worried that the backprop algorithm was not properly adjusting the weights. Further examination showed that the weights were monotonically increasing with each epoch. We discovered that the backpropagation might not be working because the MSE cost function assumes that the underlying data has a normal distribution. From looking at the various graphs of our data, it is safe to say that this assumption was not met, meaning the algorithm could be behaving in an unexpected way. This, coupled with the fact that the MSE function is non-convex for binary classification, means that our gradient descent is not guaranteed to minimize the cost function. Instead of using MSE, we should have used the Binary Cross-Entropy (BCE) Loss function or the likelihood of a Bernoulli random variable, both of which inherently handle binary classification data. We decided to attempt to implement the Binary Cross-Entropy (BCE) Loss function, but with little avail. We tried importing libraries for this cost function, but had difficulty getting them to agree with our established code. We also used gradient clipping, which is a technique applied when gradients tend to "explode", as is happening in our case. However, we still saw no change.

Weighted Classes to counter class-imbalance using xGBoost:

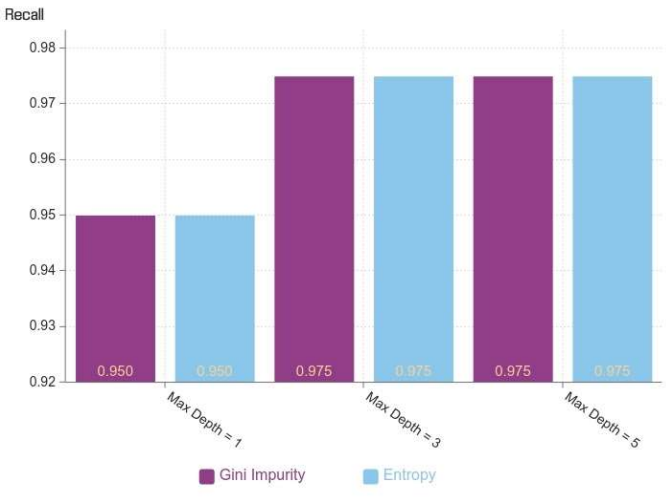
We trained an xGBoost classifier from the xgboost library on our unsampled and sampled datasets. Two classifiers were trained, one without any class weights and the other with the minority class weighed 10^4 times more than the majority class. The results were similar to those of adaBoost and Bagging in cases of unsampled data and Tomek links based sampling. TL's seem to remove the one data point which has been most liable to being classified as a false negative. The effect of class weights with xGBoost was most apparent in the cases of random oversampling and undersampling. In case of ROS, the unweighted classifier gave perfect recall and 99.6% precision, and as there was no class imbalance to address, the weighted classifier degraded precision while not affecting recall. The influence of weights was even more apparent in the case of RUS, where the unweighted classifier yielded a recall and precision of 94.1% while the weighted classifier degraded precision to 73.9% while maximizing recall which was our goal. Thus, weighted classes help address class imbalance but not to the extent resampling techniques did.

Results From Scikit Learn's Models

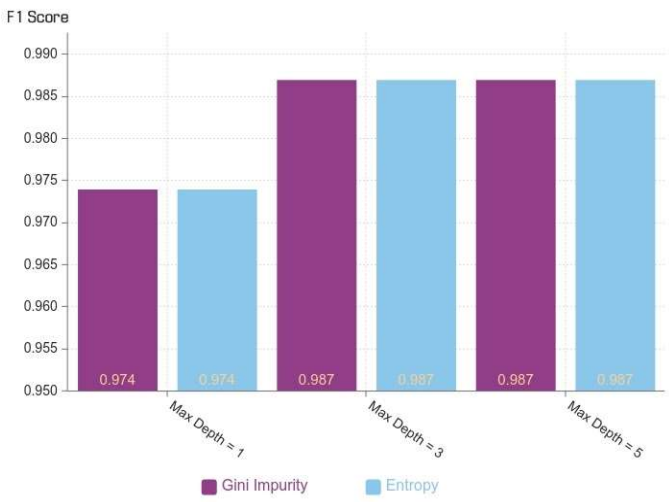
	Unsampled	NearMiss	ROS	RUS	SMOTE	TomekLinks
Neural Net Sigmoid eta =0.01	034 | 006 000 | 254 Precision: 1.0 Recall: 0.85 F1 Score: 0.919	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	258 | 016 031 | 210 Precision: 0.89 Recall: 0.94 F1 Score: 0.916	031 | 003 003 | 035 Precision: 0.91 Recall: 0.91 F1 Score: 0.912	256 | 018 034 | 207 Precision: 0.88 Recall: 0.93 F1 Score: 0.908	028 | 002 000 | 264 Precision: 1.0 Recall: 0.93 F1 Score: 0.966
Neural Net Tanh eta =0.01	034 | 006 000 | 254 Precision: 1.0 Recall: 0.85 F1 Score: 0.919	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	257 | 017 027 | 214 Precision: 0.90 Recall: 0.94 F1 Score: 0.921	032 | 002 003 | 035 Precision: 0.91 Recall: 0.94 F1 Score: 0.927	254 | 020 027 | 214 Precision: 0.90 Recall: 0.93 F1 Score: 0.915	028 | 002 000 | 264 Precision: 1.0 Recall: 0.93 F1 Score: 0.966
Neural Net RELU eta =0.01	033 | 007 011 | 243 Precision: 0.75 Recall: 0.825 F1 Score: 0.786	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	264 | 010 062 | 179 Precision: 0.81 Recall: 0.96 F1 Score: 0.88	028 | 006 007 | 031 Precision: 0.80 Recall: 0.82 F1 Score: 0.812	221 | 053 009 | 232 Precision: 0.96 Recall: 0.81 F1 Score: 0.877	025 | 005 000 | 264 Precision: 1.0 Recall: 0.83 F1 Score: 0.909
Decision Tree with Entropy Max Depth 3	039 | 001 000 | 254 Precision: 1.0 Recall: 0.975 F1 Score: 0.987	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	271 | 003 000 | 241 Precision: 1.0 Recall: 0.99 F1 Score: 0.994	034 | 000 001 | 037 Precision: 0.97 Recall: 1.0 F1 Score: 0.98	256 | 018 000 | 241 Precision: 1.0 Recall: 0.93 F1 Score: 0.966	030 | 000 000 | 264 Precision: 1.0 Recall: 1.0 F1 Score: 1.0
xGBoost w/o weighted classes 10 learners	039 | 001 000 | 254 Precision: 1.0 Recall: 0.975 F1 Score: 0.987	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score: 1.0	274 | 000 001 | 240 Precision: 0.996 Recall: 1.0 F1 Score: 0.998	032 | 002 002 | 036 Precision: 0.941 Recall:0.941 F1 Score: 0.941	NA	030 | 000 000 | 264 Precision: 1.0 Recall: 1.0 F1 Score: 1.0
xGBoost w/ minority class weighed 10^4*majority 10 learners	039 | 001 000 | 254 Precision: 1.0 Recall: 0.975 F1 Score: 0.987	034 | 000 000 | 038 Precision: 1.0 Recall: 1.0 F1 Score:1.0	274 | 000 013 | 228 Precision: 0.954 Recall: 1.0 F1 Score: 0.976	034 | 000 012 | 026 Precision: 0.739 Recall: 1.0 F1 Score: 0.85	NA	030 | 000 000 | 264 Precision: 1.0 Recall: 1.0 F1 Score: 1.0

Comparing the Hyperparameters - Without Re-sampling

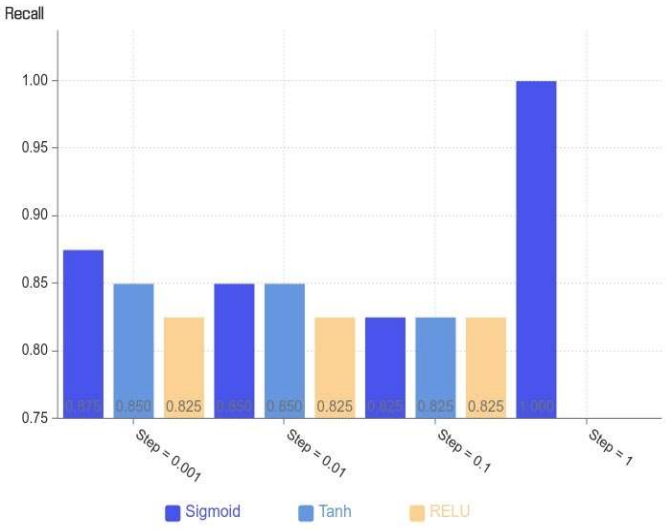
Recall for Scikit Decision Trees



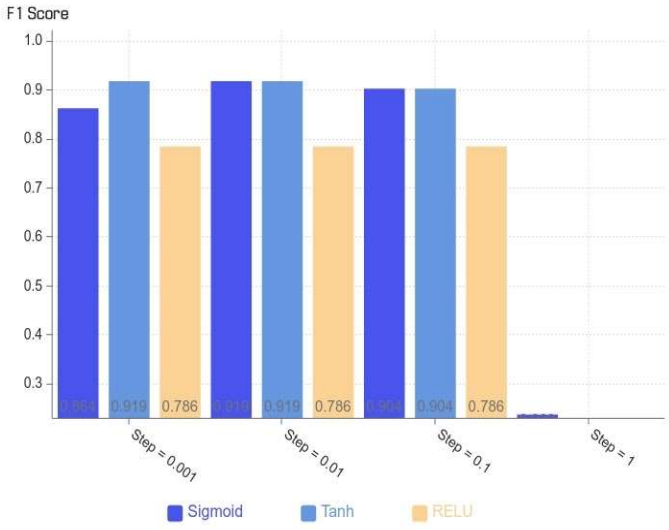
F1 Score for Scikit Decision Trees



Recall for Scikit Neural Networks

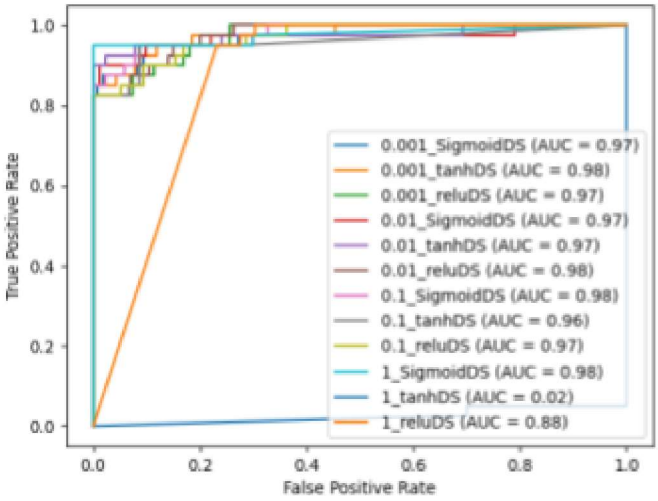


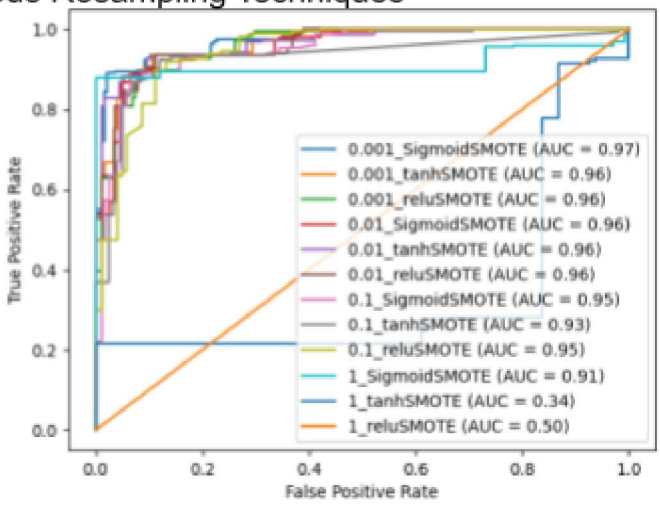
F1 Score for Scikit Neural Networks

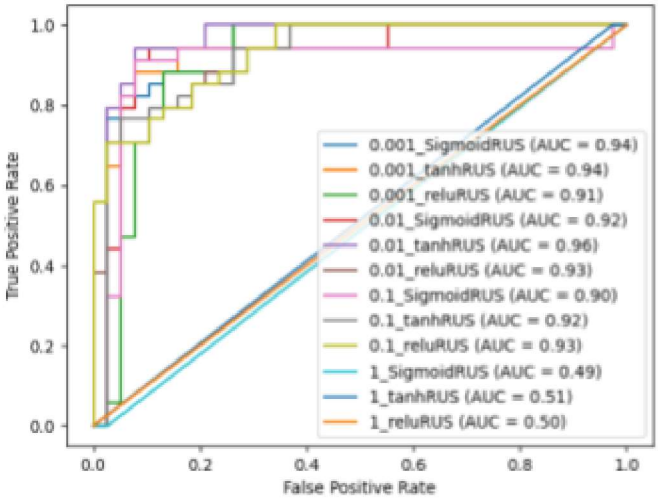


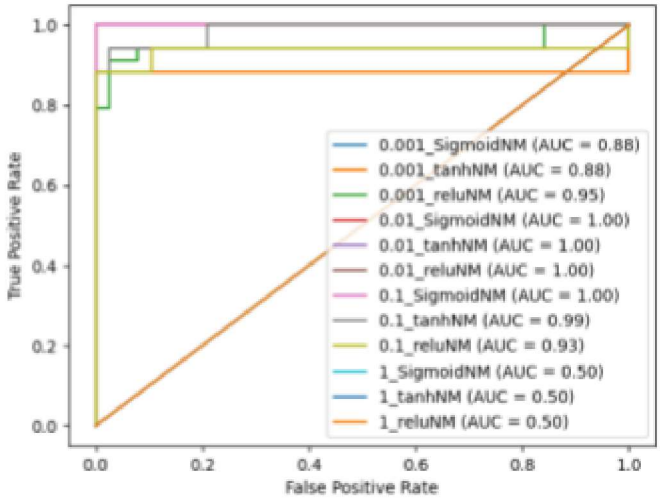
** The split criterion had no effect on the decision tree model, while the chosen activation function did produce different results for the neural networks **

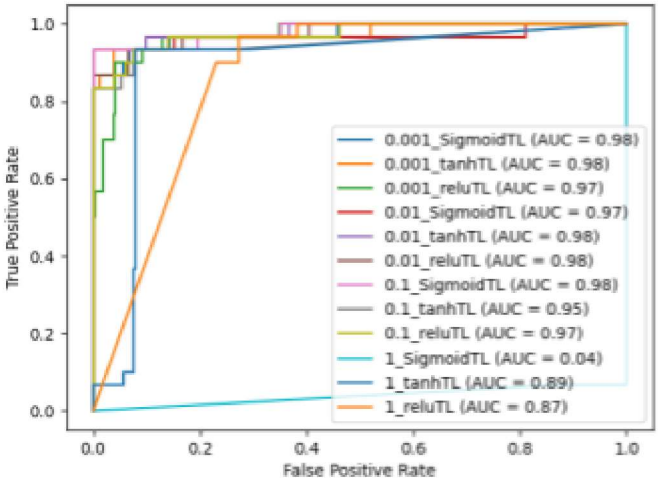
ROC Curves for the Various Resampling Techniques

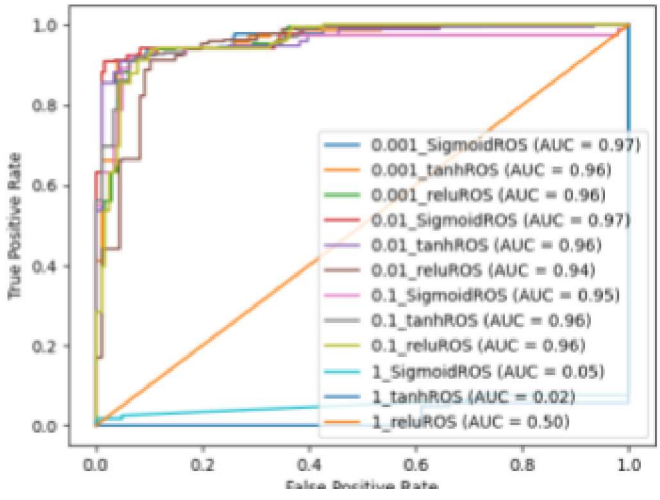












Neural Network Hyperparameters and Resampling

The models with a step size of 1 performed terribly. This is expected because when the step is that large, it is difficult to converge because it will bypass the minimum.

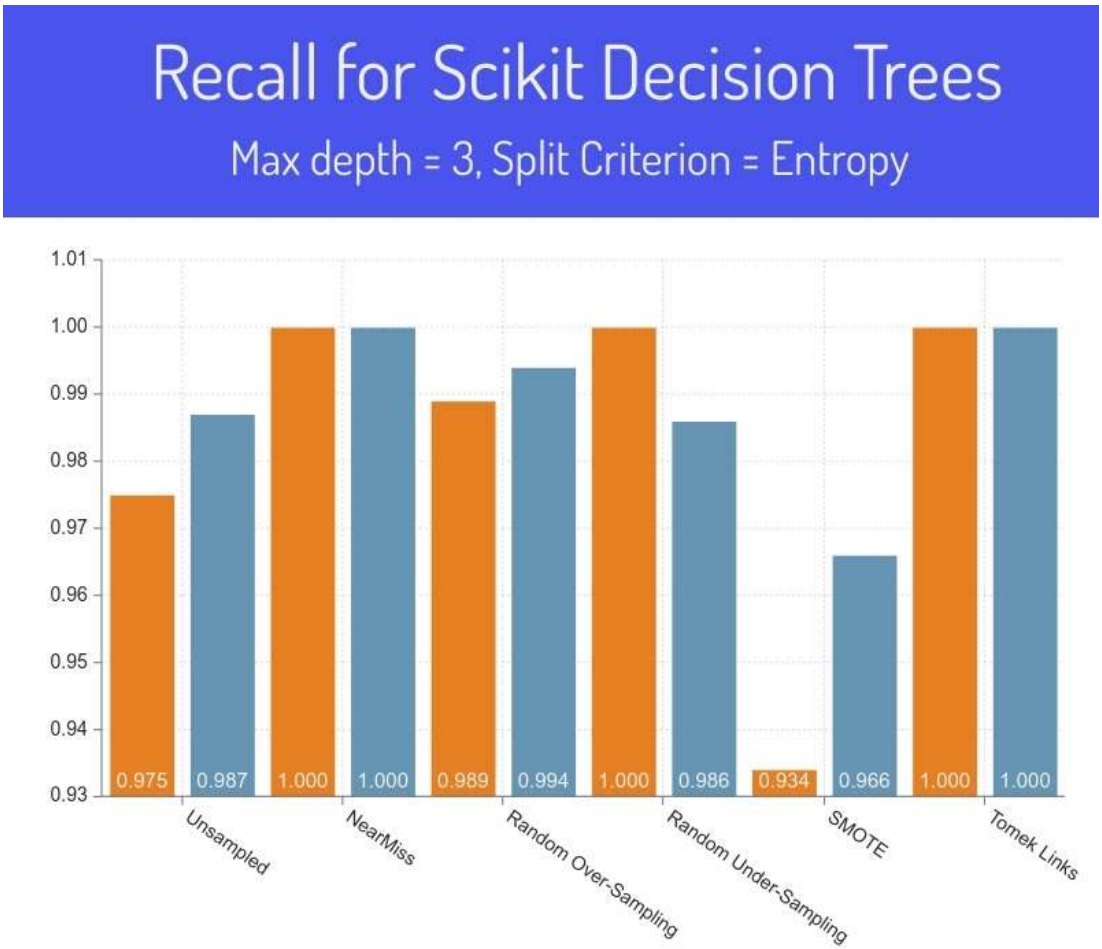
All the resampling techniques did better than the unsampled data and were comparable to each other except for tomek links. This is because, as discussed prior, there is only one identified tomek link; thus, this technique is doing little to mend the class imbalance.

ReLu performed the worst out of the activation functions. This is because in binary neural network classification, the output is the probability of the data point being in a class. The range of ReLu is [0,inf), so it is difficult to convert to a probability which would be in range [0,1], making classification harder as well. Also, ReLu makes all negative values zero regardless of their magnitude; hence, it does not map them well. Next, sigmoid and tanh had comparable results. This makes sense because the tanh function is a rescaled version of the sigmoid function

.

We did get a "*ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged yet*" warning for a few of the hyperparameter combinations. This could be because the step size is too small or too big. If the step size is too small, gradient descent will take too long to reach the minimum and will reach the maximum interactions first. On the other hand, if it is too big, the gradient will oscillate, "stepping over" the minimum.

Also, our models may be imperfect because they could be hitting a local minimum. A local minimum is guaranteed to be the global minimum when the data is linearly separable, but our data is not .

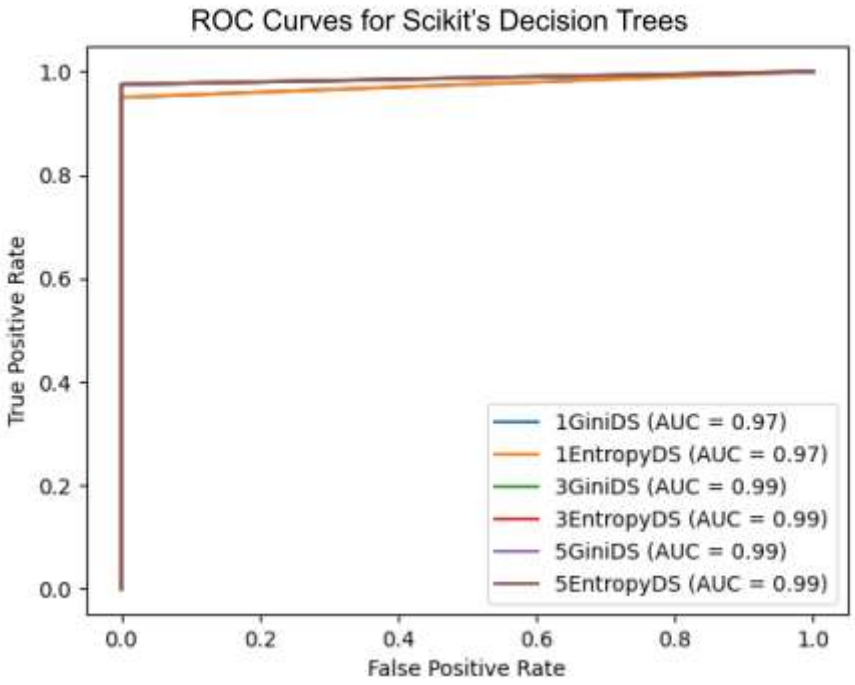


```
Using Gini Impurity
273 | 001
000 | 241
Precision: 1.0
Recall: 0.9963503649635036
F1 Score: 0.9981718464351006

Using Entropy
256 | 018
000 | 241
Precision: 1.0
Recall: 0.9343065693430657
F1 Score: 0.9660377358490566
```

To the right is the output for Scikit's Decision Tree using the SMOTE technique. This was the only instance of Gini Impurity and Entropy producing different results

The under-sampling techniques did better than the over-sampling techniques. SMOTE had the worst performance.



Potential pitfalls and Further Questions:

- The data collected is from a single sensor. Other sensors are needed to validate contact type.
- The amount of data is small, only 978 examples collected over 120 minutes.
- The presence of potential noise could cause models like adaboost to overfit.
- We are concerned that we do not have enough information to make useful predictive models. Thus, the model might not be necessarily general enough at this stage.
- We could employ Tracking Time and Contacted Device Parity: Track the exact time of contact and cross-verifying contact score with the other device's reading.
- How will the model behave on data from other sensors, i.e. testing the current model on new data + training the model on data from multiple sensors?

References

[1] B. Kumar, “Imbalanced Classification: Handling Imbalanced Data using Python,” *Analytics Vidhya*, 24-Jul-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/. [Accessed: 16-Apr-2021].

[2] R. Karim, “10 Gradient Descent Optimisation Algorithms,” *Medium*, 04-May-2020. [Online]. Available: https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9. [Accessed: 16-Apr-2021].

[3] G. Lemaitre, F. Nogueira, and C.K. Aridas (2014-2020) imbalanced-learn [Python Package] scikit-learn-contrib/imbalanced-learn: A Python Package to Tackle the Curse of Imbalanced Datasets in Machine Learning

[4] G. Lemaitre, F. Nogueira, and C.K. Aridas, “NearMiss,” *Github,* 2017. [Online] Available: NearMiss — Version 0.8.0 [Accessed: 25-Apr-2021].

[5] J. Brownlee, “Stacking Ensemble Machine Learning With Python,” *Machine Learning Mastery,* 10-Apr-2020. [Online]. Available: https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/

[6] J. Brownlee, “How to Configure XGBoost for Imbalanced Classification,” *Machine Learning Mastery*, Feb. 04, 2020. https://machinelearningmastery.com/xgboost-for-imbalanced-classification/.

[7] P. Płoński, “Xgboost Feature Importance Computed in 3 Ways with Python,” *MLJAR Automated Machine Learning*, Aug. 17, 2020. https://mljar.com/blog/feature-importance-xgboost/ (accessed May 11, 2021).

[8] S. SHARMA, “Activation Functions in Neural Networks,” *Medium*, Sep. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[9] J. Brownlee, “A Gentle Introduction to k-fold Cross-Validation,” *Machine Learning Mastery*, May 21, 2018. https://machinelearningmastery.com/k-fold-cross-validation/.

[10] R. Khan, “Nothing but NumPy: Understanding & Creating Binary Classification Neural Networks with…,” *Medium*, Dec. 08, 2020. https://towardsdatascience.com/nothing-but-numpy-understanding-creating-binary-classification-neural-networks-with-e746423c8d5c (accessed May 11, 2021).