Faculty Code: EAR
Project Number: 2102

Exploring Positive Unlabeled Machine Learning

A Major Qualifying Project Report
Submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:

Jesse Abeyta
Nicholas Cheng
Bryan Gass
Calvin Kocienda
Vinay Nair

Date: March 28th, 2021

Approved by:

Professor Elke A. Rundensteiner

With Guidance From:

Walter Gerych And Luke Buquicchio

Acknowledgements

We would like to thank our advisor, Professor Elke Rundensteiner, for her support and guidance throughout this project. Professor Rundensteiner met with us on a weekly basis over the past six months, helping us to understand academic expectations and keeping us focused on our ultimate goal. We would also like to thank WPI PhD students Walter Gerych and Luke Buquicchio for their invaluable support in this project. Walter and Luke were truly generous with their time, sometimes answering questions for us in the middle of the night and on weekends. We are grateful for their patient mentoring and critical guidance, without which this project would not have been possible. Finally, we would like to thank WPI PhD student ML Tlachac who graciously gave us access to the Twitter depression dataset.

Abstract

Positive and unlabeled learning involves positive examples and unlabeled data. The unlabeled data can contain both positive and negative examples. PU learning has gained prevalence recently due to its newfound application in social media and medicine. The current state of the art approaches to PU algorithms face a multitude of issues. Therefore the team implemented and conducted experiments on existing algorithms such as SAR-EM and NNPU. These algorithms were modified to create a novel PU algorithm.

Executive Summary

Positive and unlabeled learning is a new field in machine learning that uses data where all positive examples are labeled and there is unlabeled data that is comprised of positive and negative examples (Bekker Davis 2020). PU learning is a semi-supervised binary classification method that recovers the labels from the unlabeled section of the data.

Due to the nature of PU learning and its gaining popularity over the recent years, there has been advancement in researching different types of PU algorithms. Since PU data arises in a multitude of scenarios, having a strong PU learning algorithm could solve many of the daily issues faced by medical administrators or social media developers. In social media, the user has no option to dislike an image and therefore PU data arises in order to understand why the user did not like the content being showed. Through robust and advanced PU learning algorithms, these issues would be easily solvable, however, due to the inherently complex nature of PU data, PU algorithms in this day and age are not good enough to solve these issues without introducing other factors such as biases.

The two main biases explored were the selected completely at random (SCAR) and the selected at random assumption (SAR). The team mainly focused on working with the SCAR assumption, as it greatly simplifies algorithms, and has been used widely in PU research. The team first implemented various PU algorithms in order to understand how PU learning works and what the current pitfalls are that the state of the art algorithms are unable to avoid. Initially the team implemented various 2-step PU learning algorithms which combine two different algorithms in order to tackle PU data. These following 2-step algorithms were implemented: One-Step Naive Bayes (NB),

One-Step Support Vector Machines, Two-Step NB-SVM, and Two Step Expectation Maximization-NB. Once these algorithms were implemented the team was able to develop a better understanding of the true challenges of PU learning. The team then tried to implement a logic-based algorithm known as PULSE with a specific regard to using the algorithm to analyze the twitter dataset concerning the identification of depressed users. PULSE is a natural language processing algorithm that showed great promise but was too complex.

Another avenue of exploration was the use of TweetBERT to analyze a twitter based dataset in order to identify depression within twitter users. TweetBERT is a state of the art NLP algorithm that is used to featurize tweets such that it can be fed into a classification algorithm. The dataset collected had the associated PHQ-9 score of each twitter user which is a medical diagnosis form that shows how depressed a patient is. TweetBERT was used to featurize the twitter depression dataset and the team intended to run our novel algorithm, more information can be found in the future works section.

The next set of PU learning algorithms we analyzed were SAR-EM and TIcE. SAR-EM is an algorithm that uses an estimated propensity score of each example in order to label the example positive or negative. It uses an EM algorithm to generate a classifier and the appropriate propensity model that will assign the estimated propensity scores from the data. Furthermore, the application of the propensity weighted risk estimator also helped increase the accuracy of the simple neural network classification algorithm that was used.

Often it is useful to be able to estimate how prevalent the positive class is in a sample. This is straight forward in the fully labeled setting, but specialized algorithms are required in the unlabeled setting. We implemented and directly tested the TIcE algorithm, which is widely considered to be the gold standard for class prior estimation in the PU setting. We tested the algorithm both under ideal and challenging conditions.

While reviewing existing PU algorithms, we found that commonly used methods assume there is no bias in the labeling process. We developed a novel PU learning algorithm that can cope with bias in the labeling process through the combination and adaptation of several existing PU algorithms, including TIcE. Our novel algorithm, known as the Subclass Prior Propensity Weighted Risk Estimator, extends existing research and provides a method of classification that makes fewer assumptions about the field.

Contents

1 Introduction

1.1 Overview of PU Learning

1.1.1 What is PU Learning

In Machine Learning (ML), special algorithms are used to find statistical patterns in datasets. These algorithms can make use of these patterns to make predictions about incomplete or hypothetical observations, or individual data points. The two main types of ML algorithms are regression and classification. Regression algorithms attempt to predict an observation's continuous feature, while classification algorithms attempt to predict which discrete class an observation belongs to.

Classification algorithms could be used to identify unknown fruits as apples, bananas, or clementines based on their size, weight, and color. In this example each fruit is an observation, and its size, color, weight, and species are its features. The objective of the classification algorithm would be to find a function that correctly identifies unknown fruits given only their size, color, and weight. The function that this algorithm generates is known as the model. In order to generate a model, the classification algorithm will analyze a dataset that lists characteristics of fruits and their species. This dataset, known as the training set, should ideally have information as to what species (class) each fruit (observation) belongs to. Once a model has been trained, you can use it to predict the species of unknown fruits if you know their size, weight, and color.

In order to generate an optimal model, many ML algorithms require the datasets they train on to be fully labeled (Ratner, A). If you were to try to train these algorithms on incomplete datasets, you would likely find that they performed poorly. However, it is not always practical to have perfectly complete data. Sometimes the only source of data is unreliable. Sometimes it is impractical or prohibitively expensive to collect complete data.

In medicine, a patient's record only lists the diseases that they were diagnosed with. If a patient contracted a disease and never visits a doctor to have it treated, it will not be listed in their record. In other words, if a patient has been diagnosed with a disease they likely have it; if they haven't been diagnosed, they may still have contracted the disease at some point. If medical researchers were trying to predict which patients had a disease based on their symptoms, they would be trying to solve a Positively Unlabeled (PU) problem.

A PU problem is a binary classification problem where only some observations from the positive class are labeled (Bekker Davis, 2020). Observations not labeled may belong to the positive class or the negative class. In our medical example, a patient (observation) would belong to the positive class if they had a particular disease. If they didn't have this disease they would belong to the negative class. If a patient had a diagnosis, they would be considered "labeled". If a patient didn't have a diagnosis, regardless of whether or not they had the disease, they would be considered "unlabeled".

Other common PU problems include predicting social media preferences, evaluating advertising effectiveness, and determining if an object is present in a picture. Social media sites allow users to like posts, but not dislike them. An ad on a website can go unclicked because the user isn't interested, or because they didn't notice it. A picture not tagged as containing an object may not contain it, or the tagging may be incomplete.

Traditional ML algorithms struggle with PU datasets, as they interpret unlabeled positive observations as belonging to the negative class (Bekker Davis, 2020). This leads to the algorithm learning to distinguish between what is labeled and what is unlabeled, rather than what is positive and what is negative. In order to avoid this issue, it is necessary to use algorithms that don't assume that unlabeled observations necessarily belong to the negative class. These are known as PU learning algorithms.

1.1.2 State of the Art Approaches

The majority of current PU learning algorithms can be sorted into three different types: two-step techniques, biased learning, and class prior incorporation. Each of these has various issues that we aim to resolve with the creation of a novel algorithm .

Two-step techniques work based on the assumption that labeled examples are similarly distributed to positive examples, and that negative examples are notably different from the labeled examples (Bekker Davis, 2020). Using this assumption, this technique breaks down the process of PU learning into two separate steps. The first step involves identifying reliable negatives, reliable positives, and "unreliable" points that don't fit into either category. In other words, the classifier in this step will attempt to find points in the dataset that share many features of the positive class (hence, "reliable" positive), and classifies points with features vastly different as part of the negative class. Points that go in the unreliable category are those that are classified as positive, even though they were part of the negative class before classification. The second step involves using any type of semi-supervised learning on the previously found reliable positives, reliable negatives, and optionally, the unreliable points. The goal of this step is to generate the best model possible to classify unlabeled points.

Biased learning methods treat all unlabeled points as if they were negative (Bekker Davis, 2020). Each of these unlabeled points is given a certain amount of noise, which is a constant value indicating the uncertainty of their negative classification. Because the level of noise is constant, the SCAR assumption can be used. Unlabeled data points that display features close to the positive class are given a greater amount of noise than those that display features different than those in the positive class. This noise can be used to penalize misclassified positive samples, or tune hyperparameters using a variety of equations. Biased learning methods are often applied in support vector machines (SVM), clustering, and matrix completion.

Class prior incorporation methods take advantage of the SCAR assumption to utilize the class prior in three main methods: post-processing, pre-processing, and method modification (Bekker Davis, 2020). In post-processing, a probabilistic classifier considers all unlabeled data as negative and changes the predicted probability based on the classifier's certainty that the point belongs to the positive class. If the classifier is highly certain that the data point belongs to the positive class, it will be labeled as positive. Pre-processing changes the dataset based on the class prior in order to create a new dataset that can be incorporated into classification methods that expect fully supervised data. This process shapes the PU dataset into a different dataset that traditional classification methods can use. Alternatively, method modification involves taking traditional classifiers, such as Naive Bayes, and tuning them such that they work with PU datasets.

Although these three approaches are all able to classify PU datasets with relative success, most of these methods require several assumptions to work properly (Bekker Davis, 2020). These assumptions tend to oversimplify complexity that shows up in real-world datasets, such as bias, or distribution overlap between the positive and negative classes.

1.2 Our Approach

Our goal this project was to find a PU learning method that has similar performance to the state of the art methods, while eliminating as many of the previously mentioned downsides as possible.

During the initial phase of the project, we surveyed existing PU learning algorithms. We first familiarized ourselves with various common assumptions used in PU learning, then studied state of the art PU algorithms. In order to build a thorough understanding of these algorithms, we implemented each one using several research papers as a reference. These algorithms helped us develop some insight into the field of PU learning, and allowed us to identify some existing problems that current algorithms face.

Next, we used this background knowledge to begin development on our novel algorithm. After reading a promising paper, we decided to work on implementing PULSE, a sentence-context classification algorithm that utilizes logic programming. At this point, our team broke up into two sub-groups: Jesse, Nick, and Bryan worked on implementing PULSE and all of its components. Unfortunately, PULSE proved to be a dead end, due to its lack of documentation and scope of our project. The team switched to implementing the TIcE and SAR-EM algorithms, as well as an algorithm found in an AAAI paper, which from here on out, we will refer to as the "Subclass Prior Estimation" algorithm. They developed a novel algorithm that built off of these algorithms. We will provide a detailed description of this algorithm in section 2.8.10. Experiments run on these algorithms show that their performance is comparable, if not better, than the state-of-the-art algorithms.

Calvin and Vinay worked to implement other supplementary features, such as developing a loss function, and implementing TweetBERT, a branch of Google's state-of-the-art natural language processing (NLP) algorithm to tokenize the tweets used in our team's novel algorithm. The group did not get to utilize the tokenized tweets in our novel algorithm, but this task is something our group can explore after this project.

2 Background

In the following section we will provide an overview of the concepts necessary to understand the basics of machine learning, and how these fundamentals are applied in state of the art ML algorithms, including ones used in PU learning. These include the general components of machine learning optimization, such as dataset preparation, loss functions, training and testing, and scoring metrics. We will then review types of models, such as standard classification algorithms and neural networks. Finally, we will tie these concepts to analyzing the architecture behind current PU learning techniques.

2.1 Machine Learning Techniques

All experiments we conduct in this paper use the parameters and methods described below.

2.1.1 Dataset Cleaning

Before any machine learning takes place, the dataset the model will use must be cleaned. "Cleaning" a dataset refers to the process of optimizing the dataset for usage in a specific experiment (Russel, S. and Norvig, P). This usually involves removing several values, such as duplicate values, values with incomplete information, or entire columns that will not contribute any meaningful information to the algorithm. For example, an experiment designer creating a model that attempts to predict whether a fruit is an apple or an orange might drop the "date bought" column, as the date the fruit was bought has no relevance to its classification.

There are other reasons for dataset cleaning. When cleaning the Tweet-BERT dataset, our group noticed that not all of our tweets were in English - some were in Spanish, Arabic, and other languages. However, the Tweet-BERT algorithm was only trained on English tweets, so many of the tweets in the dataset could not be used. There were two options: either drop the non-English tweets, or translate them into English. The easier option was to drop the non-English tweets.

2.1.2 Training and Testing

After the dataset cleaning has finished, we can start training our model on the dataset. This is often broken down into two phases, known as "training" and "testing" (Russel, S. and Norvig, P). The dataset is partitioned into two different parts, called the training set, and the testing set. The training set is usually significantly larger than the testing set, with common splits being a 70-30 split, or an 80-20 split. The aim is to use the training set as a way to tune the model parameters, and use the test set to verify that the parameters have been tuned correctly. This process also helps prevent overfitting, which is when the model has been tuned to only recognize the dataset it has been trained on.

Training works by splitting up the contents of the training set into equal "batches", which are run through the model (Russel, S. and Norvig, P). Given the parameters, the model will attempt to classify each datapoint in a batch into a certain class. Afterwards, the model uses a loss function to determine how accurate its prediction was. We will discuss loss functions in greater detail in the next section. This continues until all batches have run through the model once; this cycle is known as an epoch. The model designer can train their model for any number of epochs, and generally, the more epochs the model trains for, the more accurate the model will be. However, some epochs take a long time to run, and after a certain number of epochs, the model stops drastically improving. Therefore, designers need to strike a balance between the time and resources spent training the model, and the improvements in accuracy in the model.

After training is complete, the model will move to the testing phase (Russel, S. and Norvig, P). This testing phase works the same way as the training phase, but the model uses the test dataset instead of the training dataset. Running the model on a different dataset will help to ensure that the model is able to classify any similar dataset, and is not overfit on the training dataset. The testing accuracy is a common metric used to evaluate the performance of a model.

A problem introduced when training or testing on a PU model is that the model is not always able to check its guesses against the true labels, since some labels are missing (Bekker Davis 2020). A common way to fix this problem is to modify the dataset to abide by certain PU assumptions, such as SCAR. We will discuss these assumptions later in the paper.

2.1.3 Loss Function

When a ML algorithm makes a prediction in either the training or testing phase, it must evaluate the accuracy of its predictions for a given batch.

Loss functions are a common way for a model to evaluate its performance progressively as it moves through the dataset (Russel, S. and Norvig, P). A loss function scores how closely a prediction matches its target. This score is known as the loss. The model will use the loss to tune its parameters. By taking the derivative of the loss function, the model can step backwards through the loss function to identify the parameters that contributed the most to the current loss score. The specific value of the loss differs depending on the loss function used, but generally, a low loss means that the model classified many points in a batch well. Therefore, the ultimate goal of a loss function is to help the model tune its parameters to minimize the loss.

A problem with many loss functions is that they require knowing the true label in order to work properly. In PU datasets, the true label is often unknown. Therefore, traditional loss functions are not usable in PU models. The solution is to either use some sort of PU assumption when training the model, or use a loss function specifically built for PU learning, such as the non-negative PU loss function described later in the paper.

2.1.4 Learning Rate

Once a model determines the loss of a sample, it must adjust its parameters. Techniques such as Gradient Descent (see section 3.1.5) determines the kind of change to make to the parameters, while the learning rate controls the magnitude of the adjustment (Russel, S. and Norvig, P). Picking a good value for the learning rate is important. If it's set too small, the ML algorithm could take prohibitively long to find a solution. If it's set too high, the algorithm might overshoot the optimal model . For these reasons it is important to reduce the learning rate as gradient of the loss function flattens. This allows for us to metaphorically slow down as we get closer to a locally optimal solution. Gradient descent describes how the learning rate is used in practice to optimize a model.

2.1.5 Gradient Descent

Gradient descent is used to find the minima of a loss function (Russel, S. and Norvig, P). Given a point on the surface of the loss function (representing the performance of a model) you compute the gradient, or direction of steepest descent. This indicates which changes to your parameters should improve your model the most.

Since the loss decreases as model performance improves, the loss function corresponds to the best-performing model (Russel, S. and Norvig, P). Therefore, in order to find the best possible model, we must search for the smallest loss. Below is a rendition of a loss function for a model with two parameters. The peaks correspond to lower performing models, while the valleys correspond to higher performing models.

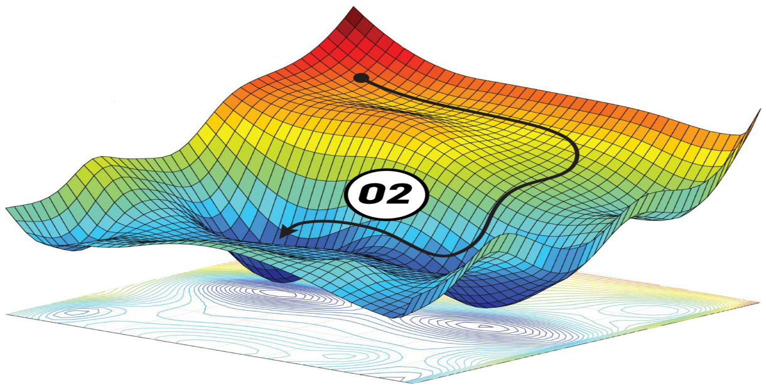


Figure 1: A depiction of gradient descent in a 3D plane. Created by Ahmed Fawzy Gad. From Paperspace.

The equation below provides an example of gradient descent (Russel, S. and Norvig, P). This exact equation may vary depending on the type of gradient descent, but these algorithms will repeat until the model converges on the local minima.

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta\theta_j} J(\theta_0, \theta_1) \tag{1}$$

Gradient descent is not guaranteed to find the global minimum, only a local one. It is common for local minima to still have high loss values, and simple gradient descent has no way of escaping them. In order to deal with this issue, several variations of gradient descent were developed. Stochastic gradient descent uses random "batches" of data to introduce a chance of leaving local minima. Additionally, k-fold cross validation involves performing gradient descent multiple times with different initial conditions, then choosing the best result (Russel, S. and Norvig, P).

2.2 Performance Evaluation

2.2.1 Accuracy

Accuracy is the simplest way of scoring a machine learning model (Google Developers, 2020). It is the ratio of correct predictions versus the total number of predictions. The major drawback of scoring models using accuracy is that it doesn't account for imbalanced data. For example, if a model predicts the positive class 100% of the time, but 90% of the dataset's points belong to the positive class, the model will be scored as 90% accurate. This is a gross overestimation of the model's true classifying capabilities. Accuracy is useful to get a overall view the model's performance, but should not be used as a definitive indicator of a model's performance.

2.2.2 Confusion Matrices

Confusion matrices can provide a more detailed insight into a model's performance (Mathworks, 2020). The confusion matrix is drawn as a chart, where each class is listed out along the x and y axis. One axis indicates the predicted classes, and the other axis indicates for the true classes. Since both axes mirror each other, it does not matter which axis corresponds to which set of classes. The two axes form intersecting cells, as shown below. These intersecting cells represent the quantity of a predicted class versus the true class. In the figure below, assuming that the x-axis represents the actual classes and the y-axis represents the predicted classes, it is easy to see that most examples were predicted correctly. For example, the model correctly classified a point as belonging to class H 36 different times. The model also incorrectly predicted a point belonging to class J as class H, twice.

During binary classification, only four cells are formed in a confusion matrix (Google Developers, 2020). Samples in these confusion matrices that were predicted as the positive class that were truly positive are called true positives. Those that were classified as negative and were truly negative are called true negatives. This logic continues for false positives and false negatives, as well.

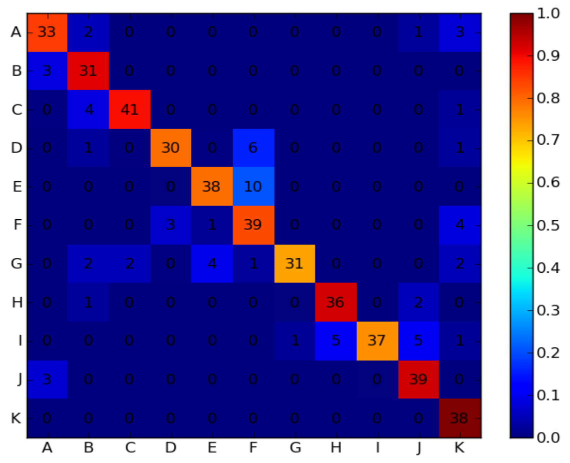


Figure 2: Example of a confusion matrix heat map. Created by Simon Zack. From Stack Overflow.

2.2.3 Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC)

The Receiver Operating Characteristic (ROC) is a plot that indicates the performance of a binary classifier at different threshold settings (Google Developers, 2020). The y-axis is labeled as the true positive rate (TPR) while the x-axis is labeled as the false positive rate (FPR). The TPR measures the ratio of total positives that were predicted as positive. The FPR measures the ratio of total negatives that were predicted as positive.

$$TPR = \frac{TP}{ActualPositive} = \frac{TP}{TP + FN} \tag{2}$$

$$FPR = \frac{FP}{ActualNegative} = \frac{FP}{TN + FP} \tag{3}$$

The ROC plots the relation between the FPR and the TPR based on a threshold (Google Developers, 2020) . The threshold determines the separation between class boundaries in cases where the binary classifier outputs continuous values. For example, suppose that a classifier outputs real values between 0 and 1, where 1 is positive and 0 is negative. A neutral threshold could be that predictions greater or equal to 0.5 are labeled as positive and all other predictions are labeled as negative. However, a threshold of 0.5 could potentially be too lax in some real-world scenarios, such as in medical diagnostics, where positive predictions carry heavy ramifications. The threshold for a prediction to be considered positive could then be raised to a higher value, such as 0.8 The TPR and FPR at the various thresholds can be graphed.

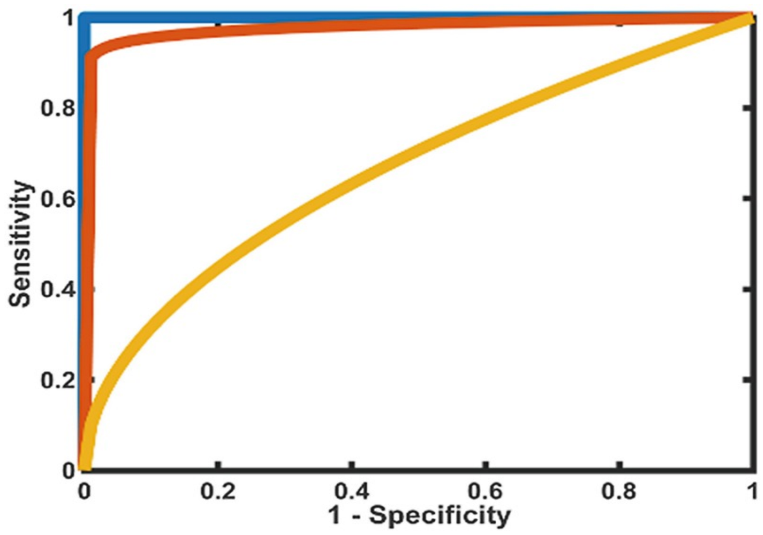


Figure 3: Example of different ROC curves, with the ideal AUC of 1 indicated in blue. Created by S. Makarov, M. Horner, et al. From NCBI.

The Area Under the Curve (AUC) is the area underneath the ROC graph (Google Developers, 2020). Higher AUC values are associated with better classifiers. An AUC value of 1 means we have a perfect classifier, while an AUC value of 0 means we have a perfectly bad (opposite) classifier. An AUC around 0.5 is the equivalent of random guesses.

2.2.4 Precision and Recall

Precision is the ratio between the true positives and the total amount of predicted positives (Machine Learning Crash Course, 2020). It is a metric that indicates how reliable a positive prediction is when predicted by the classifier as there is the possibility that the predicted positive is a false positive. In probability terms, precision is the probability that x is a true positive, given that x is predicted as a positive. Pr(x = true positive — x = predicted as positive). A classifier with a higher precision is less likely to predict a false positive.

$$Precision = \frac{TruePositive(TP)}{TruePositive(TP) + FalsePositive(FP)} \tag{4}$$

Recall measures the accuracy of the classifier in predicting true positives based on the amount of true positives and false negatives (Machine Learning Crash Course, 2020). In other words, this metric indicates how well it predicts true positives. Recall is the probability that x is predicted as a positive given that x is a true positive. Pr(x = predicted as positive — x = true positive). A classifier with a higher recall is less likely to predict a false negative.

$$Recall = \frac{TruePositive(TP)}{TruePositive(TP) + FalseNegative(FN)} \tag{5}$$

2.2.5 F1 Score

The F1 score of a classifier is the harmonic mean of the precision and recall of the classifier (Wood, T). The weight of the recall can be optionally weighted in the F1 score, denoted as beta. The beta functions as a multiplier on the importance of recall over precision.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{6}$$

When precision and recall are substituted, this becomes:

$$F_\beta = \frac{(1 + \beta^2) \cdot truepositive}{(1 + \beta^2) \cdot truepositive + \beta^2 \cdot falsenegative + falsepositive} \tag{7}$$

The F1 score is a more useful metric than accuracy when false positives or false negatives can greatly influence the model, as standard accuracy is much more heavily influenced by true positives and true negatives (Wood, T). As such, F1 scores are more useful on imbalanced datasets. Since most real-world datasets contain biased data, F1 scores are usually considered a better scoring metric than standard accuracy. Our group used F1 scores to grade our non-neural net PU classifiers, as the customer dataset we ran our classifiers on contained approximately 80% positive samples.

2.3 Machine Learning Tools

2.3.1 PyTorch

Pytorch describes itself as "An open source machine learning framework that accelerates the path from research prototyping to production deployment" (Pytorch Tutorials, 2020). It is one of the most popular tools for experimenting with machine learning. Pytorch features many popular machine learning models and functions such as support for customizable neural networks, loss functions, and activation functions. Pytorch is centered upon the usage of "Tensors" which are multidimensional arrays. Pytorch has the ability to process tensors on the GPU, giving a massive performance improvement over the CPU.

2.3.2 Scikit-Learn

Scikit-Learn bundles various tools for data analysis and machine learning (Scikit-Learn, 2020). It provides out-of-the-box functionality for data processing, pipelines, machine learning models, and other useful tools.

2.3.3 NumPy

Numpy is a python library for high performance numerical computations (NumPy, 2020). It is primarily focused on array operations and is a necessity for working with large datasets.

2.3.4 Pandas

Pandas is a library that is useful for manipulating data, especially tables and time series, for the use of data processing and analysis (Pandas, 2020). It is well known for its DataFrame objects. Data frames are two dimensional objects that represent a table. The rows contain values for each column and the columns are labeled. It allows the user complete control in manipulating rows, columns, and cells within the table. One easy way to create a DataFrame object is by loading a spreadsheet (csv file). This makes it very easy to read and write data to files.

2.4 Data Visualization Tools

2.4.1 Tableau

Tableau is a data visualization tool that's aimed at improving the flow and accessibility of data analysis (Tableau, 2020). Its ease of functionality stems from the drag-and-drop actions to create data queries using Tableau's features, known as "VizQL." Tableau has a wide range of use cases in machine learning, statistics, natural language and smart data preparation more useful to aid in human creativity in data analysis. This software has been used around the world, and will help our group identify trends when comparing different classification methods.

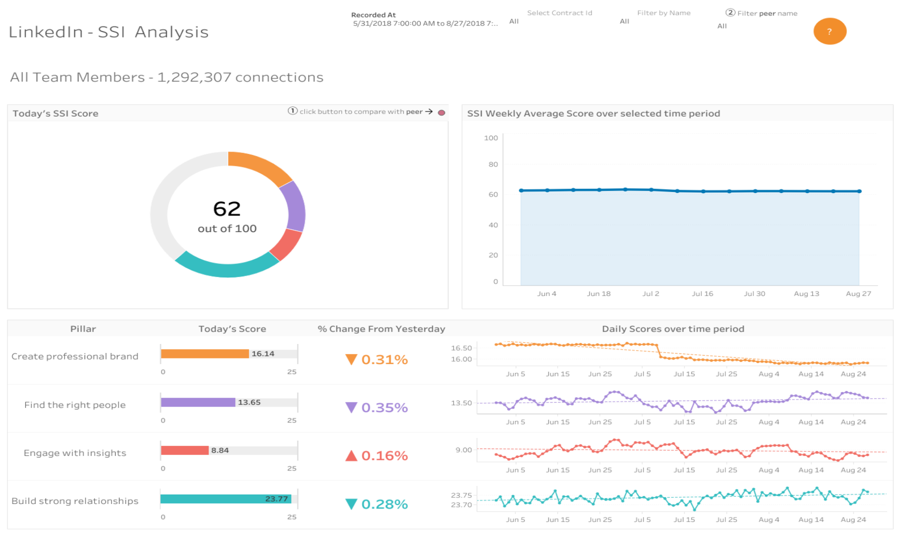


Figure 4: An example of the Tableau dashboard, alongside a visualization shows how easy and user friendly the programming interface is. An entire dashboard can be created to show various key statistics and trends that can be extracted from the dataset in question. Created by B. Staniar and T. Ngo. From Tableau.

2.4.2 Matplotlib

Matplotlib is alternative to Tableau, used by our team to generate quick visualizations while programming to better understand our datasets, and our model performance (Matplotlib, 2020). Matplotlib is a comprehensive library for creating static, animated or interactive visualizations in python. It allows the user to create publication quality plots using minimal amounts of code. The programmer can take full control over the font and axes properties as well as line style and one can export and embed to various file formats and interactive environments. It extends it usability by providing 3rd party packages that allow for further visualizations.

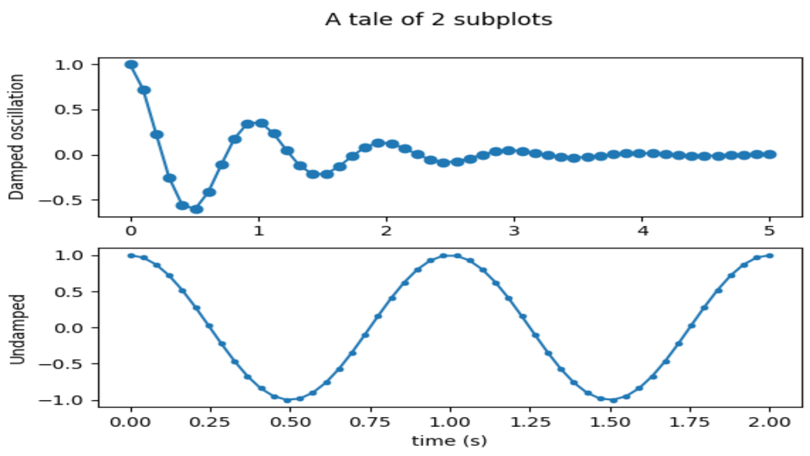


Figure 5: An example of a graph generated by matplotlib. From matplotlib.org.

2.5 Classification Algorithms

The section below details a few common classification algorithms used in machine learning. Our group attempted to implement a PU-tailored version of many of these networks onto a PU dataset. This introduced us to some of the problems faced with PU learning, such as the inability to determine a sample's true class. Implementing the PU versions of each algorithm also gave us insight into modern-day attempts at solving these PU-learning problems. To understand our theory behind modifying these classifiers, refer to section 2.1.2. Our group uses components of some of these classifiers, such as expectation-maximization, in later experiments.

2.5.1 Linear Regression

Linear regression is one of the simplest and widely used supervised machine learning algorithms. Linear regression attempts to model the relationship between an independent and a dependent variable by fitting a linear equation to the observed data (Dharani, J 2020). The linear equation that models the data is know as the line of best fit and this is the optimal linear relationship between the two variables. If there is more than one independent variable then the best fit could take multiple forms such as a plane in three dimensions or a hyperplane in higher dimensions. There are multiple ways of generating the line of best fit and one of the most common and popular methods is to use Least Squares. It takes the sum of all the distances between the line and the actual observation and finds minimum sum of the least squares.

A simple equation that approximates the linear relationship between a dependent Y and independent X variable is:

$$Y_i = \beta_0 + \beta_1 X_i \tag{8}$$

Beta 0 and Beta 1 are unknown constants where Beta 0 represents the intercept and Beta 1 represents the slope of the linear model (Dharani, J 2020). In order to calculate the loss of the linear regression model the residual sum of squares is used where the difference between the predicted data point and the actual data point is squared and added to calculate the loss of the model.

Residual Sum of Squares

$$RSS = \sum (y - \hat{y})^2 \tag{9}$$

2.5.2 Naive Bayes

Naive Bayes classifiers are a family of classifiers that follow the assumption that each feature is equal and independent when calculating the output (Hastie, T). Although this assumption allows for simple and effective classification methods, it is considered naive due to the unlikeliness of such an assumption arising in a real world scenario. Naive Bayes algorithms are based on Bayes' Theorem, which finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as:

Naive Bayes:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{10}$$

where A is the hypothesis and B is the evidence, and the probability of B is not equal to 0 (Hastie, T). This equation finds the probability of A given that B is true. P(A) is the a priori of A, which is the prior probability of A occurring before any evidence is seen. P(A—B) is the posteriori probability of B, which is the probability of an event after evidence is seen. because of the assumption of independence amongst features as that isn't always the case given real world scenarios.

There are multiple different types of naive Bayes classifiers, such as Multinomial Naive Bayes, Bernoulli Naive Bayes and Gaussian Naive Bayes (Hastie, T). Multinomial Naive Bayes are used on multinomially distributed data, and the Bernoulli Naive Bayes is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters used to predict the class variable only take up values of yes and no. Gaussian Naive Bayes classifiers are when predictors take up continuous values and aren't discrete, therefore it is assumed that these continuous predictors follow a gaussian distribution.

2.5.3 Support Vector Machines

A Support Vector Machine (SVM) is a type of classifier that builds a hyperplane, which is a decision boundary that separates data in an N-dimensional space, where N is the number of features an observation contains (Mallick, S). The hyperplane is built between two classes, and divides the data such that the distance between the hyperplane and the nearest observation in each class is maximized. The observations in each class used to define the hyperplane are called support vectors. These points are determined through the following optimization equation:

Support Vector Machine:

$$\gamma = \min_{i=1}^{N} \tag{11}$$

$$y_i \frac{w^T x_i + b}{||w||} \tag{12}$$

where x_i is a training sample, y_i is the value of binary classification of the class (1 or -1), w and b refer to the slope and intercept of the hyperplane, respectively.

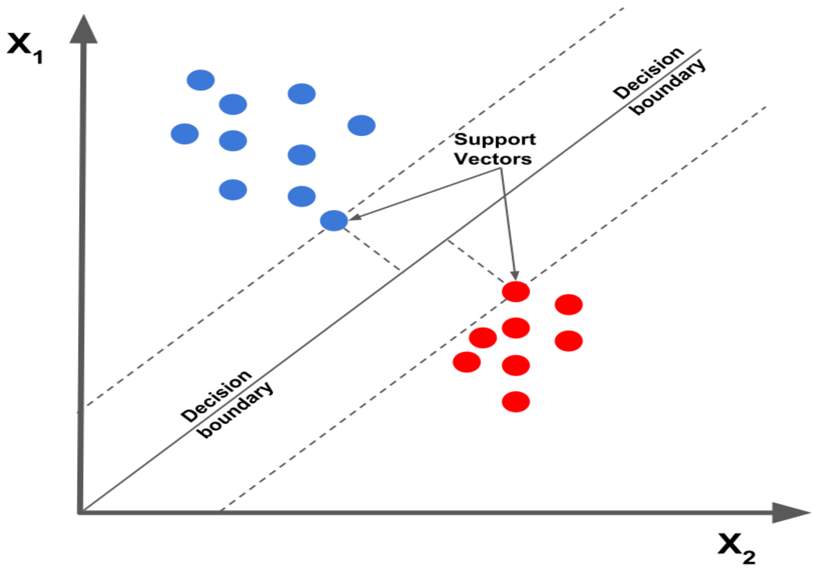


Figure 6: Image of an Support Vector Machine (SVM). Created by Lukasz Gebel. From Towards Data Science.

2.5.4 Expectation Maximization

The expectation maximization algorithm (EM) functions help to tune a model's parameters until they best fit the observed data, in cases where the model's equations cannot be directly solved (Bekker Davis, 2020). This approach works by cycling between two steps, the expectation step, and the maximization step.

The expectation step attempts to estimate the value of the missing or latent variable (Bekker Davis, 2020). Every cycle, it generates a new expected value of the log likelihood with respect to the current conditional distribution Z given X, and the current estimates of the parameters $\theta^{(t)}$.

$$Q(\theta|\theta^{(t)}) = E_{Z|X,\theta^{(t)}}[logL(\theta; X, Z)] \tag{13}$$

Next, the maximization step tries to optimize the parameters of the model (Bekker Davis, 2020). This is found by determining the value that maximizes the parameters found in the expectation step.

$$Q^{(t+1)} = arg_{\theta}maxQ(\theta|\theta^{(t)}) \tag{14}$$

The EM algorithm has a wide variety of use cases in machine learning, such as unsupervised learning problems, like clustering and density estimation (Bekker Davis, 2020).

2.6 Neural Networks

Neural networks are a common model structure used in machine learning, and make up a majority of our models used in our experiments. We used PyTorch to create our neural networks, because PyTorch allowed us to create easy, highly customizable, and powerful models for experiments. This section describes how neural networks function.

Artificial Neural Networks (often referred to as Neural Networks) are computational constructs that simulate the behavior of groups of neurons (Yiu, 2019). Much like the brain activates various neurons to control the body, neural networks use neurons to form classifiers. These networks are comprised of layers of neurons, often referred to as "hidden layers". Depending on the input to the network, some neurons in the hidden layers will fire, sending information to the neurons in next sequential hidden layer. Eventually, the final layer will contain one node, whose result will be the output of the network.

To better visualize how a neural network functions, imagine a network made to analyze a hand-drawn picture of any number from 0 to 9, and determine its value. For simplicity's sake, this network contains only two hidden layers, with three neurons in each layer. Let's say that the network receives this picture as input:

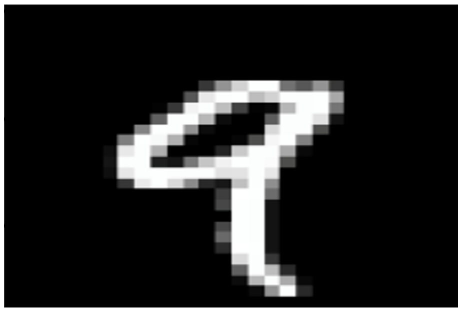


Figure 7: Hand-drawn image of the number 9. From MNIST Dataset.

One feature that a neuron might track is the amount of curvature in the middle of the picture. Some numbers, such as 9 or 8, may have this curvature pattern, similar to this drawing, while other numbers, such as 1, 2, or 7 definitely do not have the same curvature in the middle. Since there is a strong indicator that there is curvature in the middle of the image, this neuron will fire a signal into the next hidden layer. Inside the neighboring layer, there may be a neuron that tracks a vertical line on the right side of the image. Some numbers, like 2 or 5, do not have a vertical line on the right side of the image, but the number 9, and particularly this drawing of the number 9, does. After reaching the end of both hidden layers, the output neuron will receive the connections from the previous hidden layer and determine which number is represented in the picture. The network determines that the neuron for curvature in the middle and the neuron for a vertical line on the right both fired, suggesting that the number shown in the image could be the number 9.

Now that we know how neural networks work on a large scale, we can dissect how these networks perform their calculations (Yiu, 2019). As mentioned above, a neuron represents a feature, and fires based on whether the feature is strongly enough represented in the input. Given the neuron in Figure 8, this equation represents whether the neuron will fire:

$$z = a(w_1x_1 + w_2x_2 + w_3x_3 + b) \tag{15}$$

where x_i is the input to the neuron, and w_i is the weight of the neuron that determines how useful the connection is, and a is equal to g(z), where g is the activation function of the neuron. If g(z) crosses a certain threshold, the neuron will be fired.

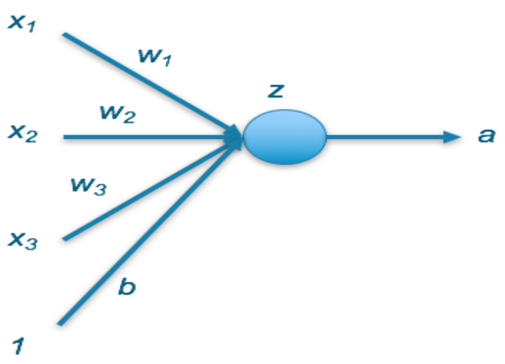


Figure 8: Image of a neuron in a neural network. Created by Stacy Ronaghan. From Medium.

A neural network of a specified architecture is similar to a function that uses a collection of weights and biases as its parameters (Russel S. and Norvig P). Neural networks effectively compute a series of linear combinations on their activation functions, with the weights acting as scaling factors and the biases acting as shifts for each connection. Linear combinations of linear activation functions always produce linear functions, so if a neural network is to approximate a nonlinear function, it must contain nonlinear activation functions.

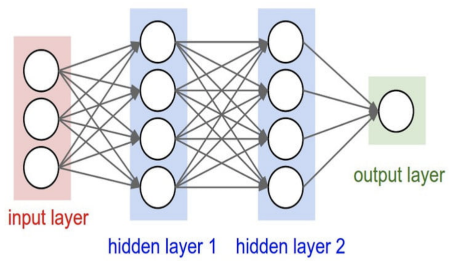


Figure 9: Image of an Artificial Neural Network (ANN) with two hidden layers. Created by Lukasz Gebel. From Towards Data Science.

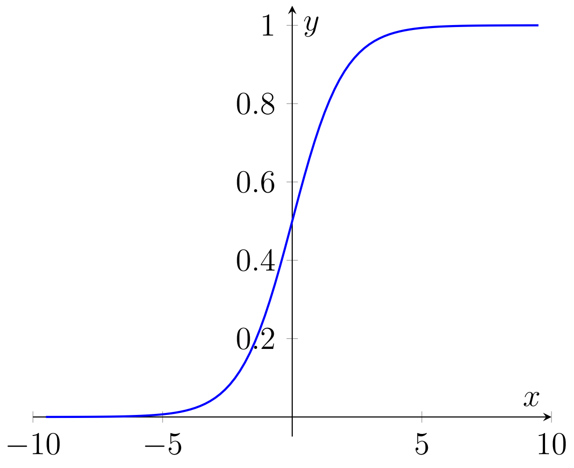
2.6.1 Activation Functions

Listed below are common activation functions used in neural networks. These functions control whether a certain neuron will fire by checking if the output of the function is greater than a predefined threshold.

Sigmoid Function The sigmoid function gives an output between 0 and 1 (Ronaghan, 2018). It is therefore commonly used in binary classification, as it is able to easily identify whether a point belongs to the positive or negative class, given a certain threshold. It is also effective at predicting probabilities.

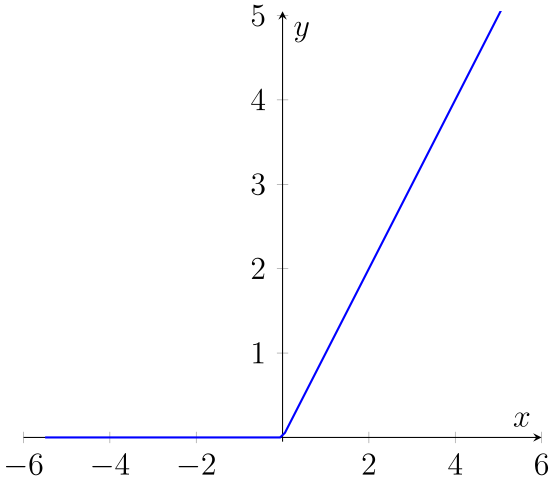
Sigmoid Activation Function:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{16}$$



ReLU Function The Rectified Linear Unit (ReLU) function returns a value between 0 and infinity (Ronaghan, 2018). It is one of the most popular activation functions, and is used for a variety of purposes.

$$Relu(x)\& = max(0, x) \tag{17}$$



Softmax Function The softmax function outputs a continuous value between 0 and 1 (Ronaghan, 2018). Softmax is similar to the sigmoid function, but in a more general form to support models that classify more than two classes.

$$softmax_i(a) = \frac{\exp a_i}{\sum \exp a_i} \tag{18}$$

2.6.2 Backpropagation

The training process of neural networks involves two steps. The first step is forward propogation, where the network calculates an output based on the input. This is the process previously described in the neural network example(Yiu, T). Next, the network performs backpropogation, which calculates the difference in error between the expected value and the actual value, and adjusts the weights of the network accordingly. This adjustment will help the network determine the value of a node firing has in predicting a certain output (Nguyen, G). Going back to the number predictor example, let's say that a neuron identified whether the handwritten number had been drawn over two pixels near the middle of the picture. This neuron would likely fire often, but because many numbers would likely cover those two pixels in the middle of the picture, the neuron is not good at narrowing down what the number could be, so the weight of this neuron will be small. Similarly, there could be a neuron that tracks for an exact match between the input and the picture of the number 9 above. That neuron wouldn't fire often, but when it did fire, the network is certain that the picture contains the number 9. As a result, the network will likely weigh this neuron highly.

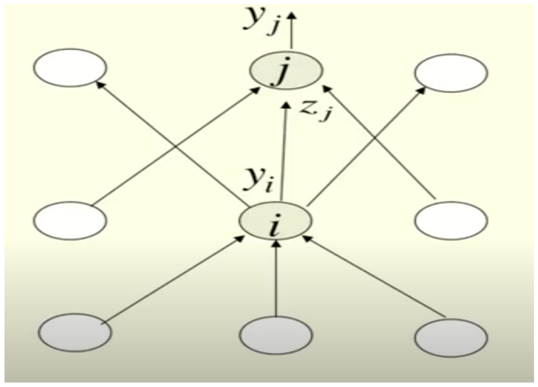


Figure 10: Image of a sample neural network during backpropogation. Created by Giang Nguyen.

Backpropogation follows these formulas, written when backpropogating E w.r.t y:

$$\frac{\delta E}{\delta z_j} = \frac{dy_j}{dz_j}\frac{\delta E}{\delta y_j} = y_j(1-y_j)\frac{\delta E}{\delta y_j} \tag{19}$$

$$\frac{\delta E}{\delta y_i} = \sum_j w_{ij}\frac{\delta E}{\delta z_j} \tag{20}$$

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta z_j}{\delta w_{ij}}\frac{\delta E}{\delta z_j} = y_i\frac{\delta E}{\delta z_j} \tag{21}$$

These equations show that the output of any node in the previous layer is equal to the weight of the connection times the derivative of the input value with respect to all total inputs the first node received (Nguyen, G). By applying this to every neuron that fired, the network can determine the specific neurons that were responsible for the greatest amount of error, and adjust their weights accordingly.

Backpropogation is necessary because professional grade neural networks can contain millions of connections, and calculating the root cause of the error using traditional gradient descent methods are often inefficient (Nguyen, G). Working backwards allows the network to not only prune neurons that did not fire in the forward pass, but also calculate the error per layer, rather than the error per neuron.

2.7 PU Algorithms

PU algorithms attempt to perform binary classification on a dataset that contains unlabeled samples, known as a PU dataset (Bekker Davis, 2020). In binary classification, the goal is to find a function f that maps an observation x to a class y, which is either positive (y = 1) or negative (y = 0). In PU learning each x has a label feature s which indicates whether or not x is positively labeled. If s = 1, y = 1. If s = 0, y = 0 OR y = 1. This forms the basis upon which all PU classification algorithms are built.

$$\begin{cases} s = 1 & y = 1 \\ s = 0 & y = 0 || y = 1 \end{cases}$$

In the section below, we will provide a detailed explanation of several assumptions that are often made when writing a PU learning algorithm. Next, we will introduce our research over the whole project into a novel PU algorithm, providing a background on how each algorithm works, and why we decided to pursue each algorithm in our research.

2.7.1 PU Assumptions

When training a PU algorithm, one has to evaluate the reasons why an sample could be unlabeled. The assumptions listed below are common assumptions that are made during PU training. We apply these assumptions because without them, it becomes nearly impossible to construct a PU learning model that will be effective on any possible dataset. Each algorithm will apply a different set of assumptions.

Selected Completely at Random (SCAR) Under the SCAR assumption, every positive sample has an equal likelihood of being labeled (Bekker Davis, 2020). The set of labeled samples can be assumed to be an independent and identically distributed sample of the positive distribution. This is an extremely simplifying assumption that does not allow for the possibility of labeling bias.

Selected at Random (SAR) Rather than an sample's propensity score being constant (as under SCAR), it is a function of that sample's attributes (Bekker Davis, 2020). This is a more realistic assumption than SCAR, as the characteristics of data points are often used to inform the labeling process. For example, people that are visibly sick are more likely to be diagnosed with a disease than those that are asymptomatic. Traditionally, the true propensity score of each example must be known for algorithms employing the SAR assumption to work.

Separable Classes The distributions of the positive and negative classes do not overlap (Bekker Davis, 2020). This implies that the unlabeled and labeled positive observations follow the same distribution.

Anchor Set There exists a subset of feature space that is wholly composed of positive observations . This subset, known as the Anchor Set, is defined by holding certain features in the observation constant. Positive and negative observations may be intermixed outside of the Anchor Set (Bekker Davis, 2020).

Separability There exists a function that defines a subdomain in the instance space where all observations are positive (Bekker Davis, 2020).

Irreducibility The negative distribution does not contain the positive distribution (Bekker Davis, 2020).

2.7.2 Single Training Set Scenario Vs Case Control Scenario

Examples of PU data within a dataset can arise from two different circumstances. The first case is the single training set in which all the positive and unlabeled examples come from a single set of examples (Bekker Davis, 2020). The case control set comes from two independently drawn data sets, one of the data sets contains the unlabeled examples and the other contains all of the positive examples.

The data samples in single training set scenario assumes that the positive and unlabeled samples is a independent and identically distributed sample from a real distribution. Only a fraction of the samples within the set are selected to be labeled and is denoted as c. This is done following their individual propensity scores denoted by e(x) and thus the data set has a fraction of c of labeled samples.

An example of this scenario can be seen in personalized advertising in which users click on a specific subset of all advertisements of interest. Therefore, the positive samples would be the ads that were clicked upon by the user from the whole population of ads shown to the user.

The case control scenario assumes that the positive and unlabeled samples originate in two independent data sets where only the unlabeled data set is an independent and identically distributed sample from the real distribution.

An example of the case control scenario can be observed in a setting where 2 data sets are used and one of the data sets only contains positive examples. In the world of healthcare, when trying to predict a patients socioeconomic states from a health record, positive examples could stem from various health centers distributed amongst upper-class areas. Therefore, guaranteeing positive examples only. The unlabeled set could stem from a random selection of health centers, therefore, containing health centers in both upper-class neighborhoods and lower class neighborhoods.

A key similarity between the two is that the observed positive examples are generated from the same distribution in the single training set and case control scenario. Both scenarios enable a set of examples that are drawn from a positive distribution through the labeling mechanism defined by the propensity score. Therefore, most PU learning methods are equipped to handle both scenarios, however, it is important to note the distinction between the two. It is imperative to consider each scenario when interpreting results and using software.

The single training set scenario is more popular than the case control scenario and has received greater attention in literally. It is possible to convert between the case control scenario to the single training set scenario.

2.7.3 PU Algorithms

This section details the PU algorithms that our group researched during the project, in chronological order.

2.7.4 PULSE

The PULSE algorithm, defined in (Blockeel, 2017) is a Relational Grounded Language Learning algorithm designed to learn the meaning of English words in natural language sentences. For the purposes of this algorithm, the meaning of a word is defined as the context (world circumstances) in which the word can be used. PULSE emerged from the field of Inductive Logic Programming (ILP), and accordingly represents contexts as statements in formal logic. Consider for instance the sentence "Sally wears a red dress and plays on the lawn." Assume for a moment that this sentence is describing a scene depicted in a photograph. This sentence asserts factual statements about the scene (That Sally is wearing a dress, and that that dress is red), but it does not necessarily rule out facts by not mentioning them. Put more concretely, the scene in question could very well include a lawn gnome, but we have no way of knowing from one sentence. What PULSE aims to do is to learn in what contexts a word can be used, knowing that the use of a word in a sentence is evidence it can be used and that the absence of a word is not evidence against its suitability. The parallels to PU learning are readily apparent. The positive class corresponds to the contexts where a word can be used. Labeled instances are sentences where a word is used, and the unlabeled instances are the sentences where the word is not used.

Statements in first order logic are derived from each sentence (either through natural language processing techniques or human labor). These logical statements, following the example given, might look like "Sally, wear, dress" or "Sally, play, lawn". From this point we can use first order logic to determine if the logical statement is true or not. This allows for the sentence to be generalized using least general generalization (LGG) to boil down the meaning of the context to the main point of what we wish to describe in the sentence, in the form of a clause. This process utilizes inductive logical programming to derive a logical foundation to sentences. Through the use of this foundation we are able to infer meaning in the most general way possible given a sentence. To understand this better consider the example "I work only on Saturdays" and "I work only on Sundays". Because both of these statements show that the person would be only willing to work on Saturdays or Sundays we can then derive the least general generalization that still keeps meaning to this person's statements. This least general generalization could be "I work only on weekends". From this point we now can understand the use of the word that might be able to be used in a sentence as a sub-cluster, or sub-clause. This sub-cluster will inhabit the overall cluster, or clause, as a logical statement that is "positive" or meets the truth we wish to meet given the context. PULSE is an algorithm that will properly find all the sub clauses that make up a clause given a context by picking different words at random and then testing.

PULSE was an area of interest for the team due in part to the limited applicability of the algorithm outside of ILP. Most notably was the efficiency of PULOR; the limitation imposed by the previous authors to remain in a discrete paradigm versus exploring a continuous one, and any kind of formal inductive logical programming library. This allowed room for the Team to expand on the state of the art work, given the time constraints in place.

Understanding of the algorithm was unquestionably sound throughout the team as well as implementation of the PULSE algorithm. Problems did rise however through the immense amount of work that came with implementing an inductive logic programming library. This was the prerequisite project to having Least General Generalization work, which is needed within PULSE. This proved to be more work than the scope of the project allowed, so we eventually scrapped PULSE in favor of TIcE.

2.7.5 TIcE

The most recent and widely used class prior estimation algorithm is Tree Induction for c Estimation (TIcE) (Bekker and Davis, 2018). TIcE performed with roughly the same level of accuracy as ALPHAMAX, while executing significantly faster. This increase in execution time, coupled with an algorithm that is easier to implement, has made TIcE the go-to algorithm for class prior estimation.

In the broadest terms, TIcE functions by using decision trees to identify subsets of data with a high number of positively labeled observations. These subsets are expected to have a proportionally large number of positive observations. From the number of positively labeled observations in these subsets, the algorithm is able to perform a series of statistical and probabilistic corrections that result in an accurate estimation of the class prior.

In greater detail, TIcE operates under the SCAR assumption and uses the stochastic nature of the labeling process to find highly positive sub domains, or regions of the data where a high proportion of observations are positively labeled. In order to find these sub domains, decision trees are utilized. In short, decision trees identify sub regions of the data set by progressively partitioning it based off characteristics of the observation. The algorithm and proof provided in the TIcE paper assumes observations have discrete attributes, and thus uses partial attribute assignment. This was explicitly done for ease of derivation, and conceptually there is no reason TIcE wouldn't work equally well in a continuous problem space. Indeed, experiments by (Ivanov 2019) indicate similar levels of accuracy in a continuous domain.

Because the label frequency and class prior is estimated from the sub domain specifically chosen to have a high number of positively labeled examples, there is an obvious risk of overestimation due to over-fitting. In order to address this, the data set is randomly split into parts, one that is used for sub domain discovery, and one that is used for parameter estimation. In other words, interesting regions are computed from one dataset and then those regions are used for analysis on another. From here, a series of bounds adjusting operations are performed to arrive at a range of possible class prior values. Although the lower bound is the most mathematically rigorous value to report, experimental results show the upper bound tends to be closer.

2.7.6 Propensity Weighted Loss and the SAR EM model

The propensity score is the probability that a positive instance is labeled positive. The crucial difference between the propensity score from causal inference and the one used in this method is that the propensity score is being conditioned on the class being positive (Bekker Davis, 2020). The method defines the propensity score as the following:

$$e(x) = Pr(s = 1|y = 1, x) \tag{22}$$

The paper represents a single set training scenario where the data can be viewed as a set of x,y,s where x is the vector of attributes (Bekker Davis, 2020). If the example is positive then it is labeled and therefore has a value of s=1 where the variable s represents whether an example is labeled. This would correlate in y=1 which means that the example is positively labeled. Therefore, anytime s=1 then y=1. However, when s=0 then it is not possible to immediately identify the value of y.

Casual inference uses inverse-propensity-scoring as a standard method where the examples are weighted with the inverse of their propensity score. This cannot be applied when working with PU data as then there is a 0 probability for labeling the negative examples (Bekker Davis, 2020). The insight is that for each labeled example $(x_i, s = 1)$ that has a propensity score e_i , there are expected to be $\frac{1}{e_i}$ positive examples, of which $\frac{1}{e_i} - 1$ did not get selected to be labeled. This insight can be used in algorithms that use counts, to estimate the correct count from the observed positives and their respective propensity scores. In general, this can be formulated as learning with negative weights: every labeled example gets a weight 1 and for every labeled example a e_i negative example is added to the dataset that gets a negative weight $1 - \frac{1}{e_i}$.

There are 2 scenarios that can arise with the propensity scores, the first is where the true propensity scores are known and the second is where the propensity scores need to be estimated based on the data. In general the propensity score can be estimated from the data however it has a bias (Bekker Davis, 2020). The estimate of the propensity score is denoted by e and from the bias the propensity score only needs to be accurate for positive examples. The propensity score is used as the labeling mechanism for the model to label whether a data point is likely to be negative or positive based on its propensity score. Through considering case 1 where the true propensity scores are known then it can directly be implemented into the algorithm and therefore avoid some of the pitfalls that occurs when the propensity score is estimated. If the propensity score is underestimated then the resulting model tends to have a higher bias. A lower propensity score would result in the model estimating the positive class to be more prevalent than it is. An incorrect propensity score has larger impact when predicted class has more extreme values (towards 0 or 1).

The SAR-EM method for PU learning utilizes the following definition to calculate the cost of the model using a propensity weighted risk estimator that takes into account the estimated propensity scores (Bekker Davis, 2020).

$$\hat{R}(\hat{y}|e,s) = \frac{1}{n}\sum_{i=1}^{n} s_i(\frac{1}{e_i}\delta_1(\hat{y}_i) + (1-\frac{1}{e_i})\delta_0(\hat{y}_i)) + (1-s_i)\delta_0(\hat{y}_i) \tag{23}$$

This equation enables the model to calculate the cost of any incorrect predictions while taking into account the propensity score. This helps to generate higher accuracy as it optimizes the model by utilizing the propensity scores (Bekker Davis, 2020).

The SAR-EM model finds a propensity function through the use of an expectation maximization algorithm. First an assumption is made that true propensity score is not known and therefore the propensity function only depends on a subset of all attributes called the propensity attributes. Then an expectation-maximization (EM) function is run that will search for both a classifier and a lower dimensional propensity function that best explains the data (Bekker Davis, 2020). The aim of the maximization part is to maximize the expected log likelihood of both models, thus it is imperative to optimize the log loss of a weighted data set. The classification model f receivers each example i twice once as positive, weighted by the expected probability of it being positive $\hat{y}_i$ and once as negative, weighted by the expected probability of it being negative (1 - $\hat{y}_i$). The propensity score model e receives each example once, positive if the observed label is positive and negative otherwise, weighted by the expected probability of it being positive $\hat{y}_i$. Due to the SAR-EM model generating the classification model and propensity function to estimate the propensity scores of the data set, it works better than traditional models that fail to factor in the propensity scores.

2.7.7 Subclass Prior Estimation

Most methods of PU learning in common use rely on the SCAR assumption. This assumption effectively means that each observation from the positive class has an equal probability of being labeled. This greatly simplifies the math behind many PU algorithms, but seriously at odds with real world complexity. It is common for the attributes of an observation to determine whether that observation is labeled. One can more closely model the real world by instead using the SAR assumption and propensity scores, but such a move rules out many of the existing PU algorithms.

In "Class Prior Estimation with Biased Positives and Unlabeled Examples", the authors propose a method for considering a data set as consisting of multiple positive sub-classes, each with a different label probability. This effectively models biases in sampling, where the under or over labeling of a group of observations is determined by their sub-class membership. Considered individually, each sub class satisfies the SCAR assumption, even though the total data set does not. This allows us to apply algorithms that rely on the SCAR assumption to individual sub-classes.

A brief description of their algorithm is as follows: first, one is to perform the K-means clustering algorithm at several different values of K on the labeled positive data. Each time this is done, the silhouette score (a measure of how well a K means model fits the data) is computed. The model with the best silhouette score is used as the final one. This determines how many sub classes are present in the data set. Then, for each positive sub class, a class prior estimator, such as TIcE, is used. The result of this estimator is returned for each discovered sub-class. Below is a more rigorous examination of the algorithm and the math behind it.

Let f_1 and f_0 represent the distributions for the positive and negative classes respectively. f represents a mixture in which unlabeled data is sampled from f_1 and f_0. α represents the class prior for the positive class of the mixture f.

$$f(x) = \alpha f_1(x) + (1-\alpha) f_0(x) \tag{24}$$

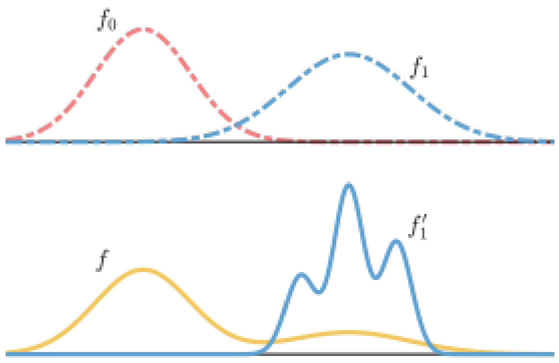


Figure 11: The top picture shows the positive f_1 distribution and a negative f_0 distribution. The lower image shows the mixture f drawing from both the positive and negatives. f_1' is the positive labeled distribution. Image from Class Prior Estimation for Biased Positives and Unlabeled Examples (Jain, S et al 2020).

The goal of this algorithm is to find the α that best describes f given a biased positive sample and an unbiased unlabeled sample. Let f_1' be the positive labeled distribution. f_1' and f_1 are related under a ”mixing bias” assumption. This means that f_1' and f_1 are assumed to be up by the same components, but may be at different mixing proportions. The two are related under a K-component mixture representation.

$$f_1' = \sum_{i=1}^{K} r_i \phi_i(x) \tag{25}$$

$$f_1 = \sum_{i=1}^{K} \gamma_i \phi_i(x) \tag{26}$$

$\phi_i(x)$ are density functions, $r_i, \gamma_i \in [0,1]$, $\sum_{i=1}^{K} r_i = 1$, and $\sum_{i=1}^{K} \gamma_i = 1$

Additionally, f_0 cannot be expressed as containing the same components ϕ as which f_1 contains. This is the ”ϕ-irreducibly” assumption. Under these assumptions, the approach of this algorithm is to divide the problem into multiple unbiased positive unlabeled sub-problems. A class prior estimation algorithm is then used to find α for each sub-problem. The results are combined via a weighted average to arrive at a final result.

2.7.8 Propensity Weighted Loss with Subclass Prior Estimation

The label probabilities of sub-classes determined using the Sub-class Prior Estimator provide critical insight into the true distribution of positive observations in the data-set, but (Jain S. et al. 2020) provides no direct suggestion as to how to use this information. While considering this algorithm, we noticed the parallels between the propensity scores from the propensity weighted estimator in SAR-EM (Bekker et al., 2019) and the label probabilities of the sub-classes. A logical next step was for us to combine the two algorithms. SAR-EM is employed, but instead of estimating the propensity score from the data, the Sub-class Prior Estimator is used to assign observations to sub-classes and each observation is assigned it's sub-class's label prior as its propensity score.

3 Related Works

This section highlights a number of other papers that explore alternative avenues of research into PU learning.

Due to their nature of building on existing algorithms, one common research path involved transforming standard ML classifiers into PU classifiers. Our group implemented a few of these, but there remain many alternative PU transformations on several other types of ML classifiers. These methods fall under three categories: two-step methods, biased learning methods, and class prior estimation methods.

During our research, we found several two-step methods that share similar methodologies to the two-step algorithms we implemented [Fung 2006, Bekker 2020, Peng Zuo , Yu]. This gave us a firm understanding of how two-step algorithms worked, and allowed us to reference these papers when writing our own methods. An example is ”Positive examples and Negative examples Labeling Heuristic”, or PNLH. Similar to the two-step methods we explored, PNLH attempts to identify reliable positives and reliable negatives. PNLH first extracts a set of reliable negatives using features that frequently occur in positive data. Next, PLNH iteratively increases the set of reliable positives and reliable negatives by classifying a data point based on its relation to the current positive or negative clusters. For example, if a point is close to the positive cluster and not close to any negative clusters, it will be added to the positive cluster (Fung, G.P.C). K-means clustering uses a similar method, but uses k-means to form the clusters, instead of using feature comparisons (Bekker, J.). Another two step technique that is used is a combination of 1-DNF and iterative SVM. 1-DNF is a classification method that finds recurring features within the labeled positive examples and classifies these as ”strong” features. A reliable negative is one where none of the strong features are present. Since there are many possible features that could constitute a positive feature, to gather a set of reliable negative features is difficult. Therefore 1-DNF avoids this complication by setting a minimum threshold with a frequencies of all the features to select the strongest ones (Peng, T, Zuo, W/81). Iterative-SVM uses an SVM classifier that is trained on the positive examples and the reliable negatives and the unlabeled examples that are classified as negatives are added to the reliable negative set and this is algorithm runs until there are no more points to be classified (Yu, H/112).

A separate field that has piqued the interest of PU learning researchers is relational learning. This process involves filling in knowledge bases with additional data, based on new relationships found inside the already existing data. Relational learning models consider all data inside the knowledge base to be true, and all data yet to be added as unlabeled. The previously mentioned PU algorithm PULSE (section 2.8.4) is an example of a relational learning method. Most existing relational learning methods make the *closed world* assumption, which considers every point not already in the dataset as negative. This assumption helps facilitate the creation of new relations in the data, by excluding the possibility of hypothetical data potentially changing the nature of the relationship. However, some recent methods have started to make the *open world* assumption, which accounts for data not inside the knowledge base by identifying it as incomplete (Bekker, J.). Although relational programming is largely a different field than machine learning, PU researchers have been able to slightly modify relational models to work in a PU setting. One such method, known as TIcER, is an open world that builds off the TIcE algorithm mentioned in section 2.8.5. Assuming the SCAR assumption holds, TIcER can estimate the class prior directly from the PU data by creating a first order logic tree. This algorithm is explored in the paper, "Positive and Unlabeled Relational Classification through Label Frequency Estimation", by the same authors as the survey paper. As with PULSE, TIcER could have been explored by our group if the scope of our project was larger. Relational learning represents a potentially untapped field for PU learning, and early experiments with algorithms such as TIcER look promising.

A common assumption in PU learning is that both the labeled and unlabeled positive instances come from the same distribution. However, in the real world, this may not always be the case. Data entries may contain attributes which bias them towards being labeled versus unlabeled. One proposed solution involves developing a robust class prior estimation algorithm using a clustered approach (Jain, S. et al). This algorithm serves as a basis for our novel algorithm. This solution relies on another algorithm, TIcE, which performs class prior estimation on a per-cluster basis.

4 Methodology

This section provides an overview of the work our group accomplished. We describe the setup for each experiment we run, and discuss our goals for running each experiment.

4.1 Data Preprocessing

4.1.1 BERT and TweetBERT

Bidirectional Encoder Representations from Transformers, or BERT, is an neural network developed and maintained by Google. It is used for natural language processing (NLP) to help the Google Search algorithm improve its precision in finding results. Its goal is to help machines understand conversational subtleties of languages that aren't obvious when reading a string of words at face value (Nayak, P). The problem with using Recurrent Neural Networks (RNN), considered to be the traditional method of language understanding tasks, is that such models operate sequentially, and read the sentence left-to-right, or right-to-left. This approach can cause some problems, especially with sentences that require a word at the end of a sentence in order to put the whole sentence into context (Uszkoreit, J). For example, the word "bark" in the sentence "The bark over there was painfully..." has a different meaning depending on the last word in the sentence. If the last word is "loud", then bark would be referring to a dog bark, but if the last word was "rough", bark would be referring to tree bark. When reading this sentence from left-to-right, the sequential model must read every word in between "bark" and "loud" or "rough" to determine the meaning of the whole sentence. A study conducted in 2015 shows that generally, the more times a network has to take several steps in order to make a decisions, such as in the sentence described above, the harder it is to train that network (Hochreiter, S., et al) . Additionally, sequential neural networks cannot make full use of graphics cards for computation, as graphics cards excel at parallel computation, instead of sequential computation, leading to an even slower training process (Uszkoreit, J).

BERT avoids this problem by utilizing transformers, or models that take each word in the context of a sentence as a whole, instead of in the context of words that come before or after (Uszkoreit, J). First, transformers run a constant number of steps, and in each step, the transformer models the relationship between each word in a sentence, regardless of the word's position in the sentence. The model uses these steps to form an attention score given to each two-word combination in the sentence. The attention score indicates how important one word is at defining another word. For example, the two-word combination "The bark" would be given a lower score than the combination "bark loud". This process is called "self-attention". From here, the model can use these relation pairs to assist in its overall NLP goal, such as optimizing the precision of the Google Search algorithm.

TweetBERT works similarly to BERT, but has instead been pre-trained on tweets. Tweets differ from the average Google Search queries in content, length, usage of slang, and usage of emojis, which makes a separate algorithm necessary. Our implementation of TweetBERT was taken from the GitHub repository: https://github.com/VinAIResearch/BERTweet.

Our group implemented TweetBERT to tokenize our tweets. This tokenizing process involves transforming the text of a tweet into a tensor of numbers that a model can understand. This allows our group to train our novel algorithm on any real-world dataset. We used this to tokenize a Twitter dataset, referenced in section 5.1.2. After tokenizing our tweets using TweetBERT, we fed them to a PyTorch neural network. The goal with creating a model was to figure out a model architecture that works well with the tokenized tweets. This would not only have helped with the creation of our novel algorithm, but it would also have served as a benchmark to compare our novel algorithm's performance. However, the scope of our project changed, and we didn't have the opportunity to implement the Twitter dataset with the tokenized tweets on our novel algorithm. This is a task we will explore in the future.

4.1.2 Performance Baselines And Data Pipeline

Determining the reliability and the quality of the results during the testing process can be a daunting task. Often times, results referenced from state of the art methods show qualities within their practices that are lacking to show the bigger picture. This bigger picture will often show information that aids in the understanding of the Team's re-implementation which can uncover any errors in our work or from authors. This process of developing baselines, to the often specialized data, can be time consuming and difficult to do. This can be especially difficult in the time consuming nature of re implantation of state of the art techniques.

In an aim to shorten this process the team has developed a series of pipelines of several algorithms and data preprocessing techniques. These series of pipelines will enable the team to handle more unique and specialized data, develop baselines using standard statistical methods and more easily work toward developing specific baselines within the algorithms we test and implement in the future.

Often times, the team met resistance when figuring out the quality of the data within a positive unlabeled learning environment. Techniques in the past have been to create simulated positive unlabeled data in the past that can confirm our findings within the work we have done. However, this process can become tedious due in part from the nature of the specialized data. Our hope is that our pipelines within our work within positive unlabeled learning will allow for us to focus less on the fine tuning of the data and more so on the implementation of future algorithms.

The challenges that the Team has met are making these pipelines as robust as possible. Meeting every edge case is a statistical impossibility but as work continues these pipelines will save the Team time toward more menial tasks within the research development cycle. This of course calls for continuous upkeep to account for new or unforeseen algorithms and data.

4.1.3 Pipeline

The classification and runtime performance of Machine Learning algorithms are highly dependent on specific implementation details. In order to provide background to our results and provide enough information to attempt an accurate replication, we have provided a detailed overview of how we implemented our pipeline and individual algorithms. All code was completed in Python, primarily because of its wide selection of Machine Learning libraries. In order to make clear the difference between how we implemented algorithms in Python and how we intend them to work in the abstract, we will be defining our algorithms using pseudo code.

In order to evaluate the performance of various Machine Learning algorithms on different datasets under different conditions, we must perform the algorithm below.

1. Load Dataset into memory
2. Standardize Dataset

3. Split Dataset into train and test sets
4. Apply transformations simulating PU assumptions (described in INSERT SECTION HERE)
5. Apply Machine Learning algorithm to the training set
6. Report out the performance of algorithm on train and test set

This general algorithm and the steps that comprise it are what is known as a Machine Learning pipeline. For every different combination of models and assumptions we wish to test, we must construct a different pipeline. Rather than write out small variations of the same algorithm dozens of times, we elected to write a single function that generates pipeline functions as needed. A function that takes other functions as arguments and/or returns functions is known as a higher order function. Make_pipeline is a higher order function that creates a pipeline function for a particular dataset and Machine Learning algorithm. Its pseudocode is listed below.

Psuedo code for make_pipeline:

```
function MAKE_PIPELINE($preprocess\_function, model\_function, postprocess\_function$)
    return function pipeline(x):
    $x = preprocess\_function(x)$
    $x = model\_function(x)$
    return $postprocess\_function(x)$
```

Each pipeline object is a function of one variable that is in turn composed of three sub-functions of one variable. This has the satisfying effect of only requiring one line of code to invoke the entire pipeline on a dataset. Steps 2-4 in the list above correspond to the preprocess_functions parameter, while steps 5 and 6 correspond to model_function and postprocess_function respectively. If one wishes to apply multiple preprocessing functions, they can be combined into a single function. The argument passed through each stage of the pipeline is the dataset, so functions of multiple parameters need to be converted into functions one parameter through one way or another. We have made extensive use of wrapper and partial functions for this purpose.

Bugs are an inevitable part of the software development process, and so we have adopted certain programming practices in an effort to manage them. One of these practices is the use of the functional programming style.

Functional programming emphasizes the use of higher order functions and the avoidance of mutable state. This allows us to write our code in a manner that is both more abstract and easier to reason about. Minimizing the use of mutable state makes it easier to determine the origin of bugs.

Although there are many benefits to functional programming, for reasons of pure practicality we have had to make a few compromises. We need to keep track of what observations were selected for the train and test datasets in order to evaluate the model's performance. In order to achieve this, we elected to pass objects into our pipeline functions that would be filled in at runtime with records of which observations were unlabeled. Our current pipeline for the Customer Intentions dataset consists of the following operations:

1. Apply a transform to the data so that the class indicator is in a known position.

2. Encode every categorical variable as a integer

3. Split the dataset into a train and test set

4. Randomly unlabel X% of positive observations. This simulates the SCAR assumption holding for the dataset.

5. Apply the PU algorithm to the train set

6. Evaluate the results of the the previous step

We created a pipeline to test each of our PU classifiers, which contains a series of functions.

4.2 Model Selection

4.2.1 One-Step NB

Our implementation of the One-Step Naive Bayes classifier relied heavily on scikit-learn's implementation of Gaussian Naive Bayes. In accordance with our inclinations towards functional programming, we have created a wrapper function classify_naive_bayes that handles the training and application of a Gaussian Naive Bayes classifier. This wrapper function creates a copy of the dataset argument so that the function causes no side-effects. The pseudocode for our implementation is below.

Psuedo code for classify_naive_bayes:

```
function CLASSIFY_NAIVE_BAYES($dataset$)
    $dataset\_copy \leftarrow copy(dataset)$
    $model \leftarrow TrainGuassianNaiveBayes(dataset)$
    $new\_labels \leftarrow model(dataset\_copy)$
    $dataset\_copy.labels \leftarrow new_labels$
    return $dataset\_copy$
```

It should be noted that prior to applying this function preprocessing functions as described in section 5.1.3 have been applied to our dataset. Thus, the Gaussian Naive Bayes function is trained on a training set with normalized data and various PU assumptions applied.

4.2.2 One-Step SVM

Implementation of the one-step support vector machine was tested to see the effects of applying the SCAR dataset to a support vector machine to gauge how well the F-score changes based off each iteration of SVM use. We found that the one-step proved to be less than favorable and turned to the use of one step Naive Bayes along with a support vector machine. This is expected to show significant improvement comparatively to its one-step support vector machine counterpart. Additional plans are to continue on with this pattern of testing with two-step counterparts to naive bayes, support vector machines and hybrids of the two.

4.2.3 Two-Step NB-SVM

Our two-step NB-SVM implementation builds upon the one-step NB from Section 5.2.1 and adds an additional SVM classifier. The SVM used is implemented through Scikit-learn's implementation of a linear support vector classifier. The pseudocode for the entire two-step NB-SVM algorithm lies below.

Psuedo code for two_step_nb_svm(dataset):

```
function TWO_STEP_NB_SVM($dataset$)
    $nb\_classified\_dataset \leftarrow classify\_naive\_bayes(dataset)$
    for entry in nb_classified_dataset do
        if entry is negative in dataset and nb_classified_dataset then
            $uncertain.append(entry)$
        if entry is positive in dataset then
            $positives.append(entry)$
    $any\_negatives \leftarrow True$
    while $any\_negatives$ and $uncertain$ do
        $train\_obs, train\_labels \leftarrow reliable_negatives + positives$
        $train\_svm\_classifier(train\_obs, train\_labels)$
        $predictions = fit\_svm\_classifier(uncertain.observations)$
        $any\_negatives \leftarrow False$
        for $prediction$ in $predictions$ do
            if $prediction$ is negative then
                Remove prediction from uncertain
                $reliable\_negatives.append(prediction)$
                $any\_negatives \leftarrow True$
```

4.2.4 Two-Step EM-NB

We implemented our two-step EM-NB classifier based on the method described in the paper, “An Evaluation of Two-Step Techniques for Positive Unlabeled Learning in Text Classification”. We built our two-step NB-SVM classifier first, and built our EM-NB classifier on the framework of the NB-SVM. The two classifiers contain many similar features, including the preprocessing functions, and the first step in the two-step process. After running the first step, the algorithm moves on to the Expectation Maximization step, which iteratively runs a Naive Bayes classifier on the “positive” and “unlabeled” datasets. With each iteration, the classifier produces new labels for each element inside the “unlabeled” dataset. Iterations continue until the dataset reaches a certain threshold.

```
function TWO_STEP_EM_NB($dataset$)
    $nb\_classified\_dataset \leftarrow classify\_naive\_bayes(dataset)$
    for entry in nb_classified_dataset do
        if entry is classified negative by both the nb_classified_dataset and dataset then
            $reliable_negatives.append(entry)$
        if entry is classified positive by nb_classified_dataset and is negative in dataset then
            $uncertain.append(entry)$
        if entry is positive in dataset then
            $positives.append(entry)$
    $any\_negatives \leftarrow True$
    while $any\_negatives$ and $uncertain$ do
        $train\_obs, train\_labels \leftarrow reliable_negatives + positives + uncertain$
        $train\_svm\_classifier(train\_obs, train\_labels)$
        $Uncertain\_obs = uncertain$
        $Predictions = fit_gaussian_naive_bayes(uncertain_obs)$
        $any\_negatives \leftarrow False$
        for $prediction$ in $predictions$ do
            if $prediction$ is negative then
                Remove prediction from uncertain
                $reliable\_negatives.append(prediction)$
                $any\_negatives \leftarrow True$
    for entry in uncertain: do
        $entry \leftarrow positive$
    $Mergereliable_negative, positives, uncertainintonewdataset$
```

!!!Source for nnPU

4.2.5 Non-Negative PU Loss

Our team implemented a non-negative loss function, which is a function that outputs a value between 0 and 1 that measures the loss during training (Kiryo et al., 2017). We purposefully ensured that our loss function was non-negative, or unable to output a value less than zero, because negative loss can lead to overfitting. Next, we decomposed the loss function to work on PU datasets. The four figures below depict the decomposition process.

$$E_{x\sim P}[L^-(x)] \quad = \quad \pi\, E_{x\sim P+}[L^-(x)] + (1 - \pi)\, E_{x\sim P-}[L^-(x)]$$

Figure 12:

Figure 12 represents our starting equation. This formula shows that the average negative loss of all points in a dataset is equal to the class prior, π, multiplied by the average negative loss over all positive points plus the inverse of the class prior multiplied average negative loss over all negative points.

$$E_{x\sim P}[L^-(x)] \quad = \quad \pi\, E_{x\sim P+}[1 - L^+(x)] + (1 - \pi)\, E_{x\sim P-}[L^-(x)]$$

Figure 13:

Figure 13 converts the average negative loss into the inverse of the average positive loss. This step will be important in Figure 15.

$$E_{x\sim P}[L^-(x)] - \pi\, E_{x\sim P+}[1 - L^+(x)] = (1 - \pi)\, E_{x\sim P-}[L^-(x)]$$

Figure 14:

Figure 14 proves that the left hand side is equal to the inverse class prior multiplied by the average negative loss over all negative points. The left hand side can be used to represent all unlabeled points in a dataset, as it contains the set of all points in a dataset minus the positive points.

$$\begin{tikzcd} E_{x\sim P}[L(x)] \quad = \quad \pi\, E_{x\sim P+}[L^+(x)] + & \boxed{(1 - \pi)\, E_{x\sim P-}[L^-(x)]} \arrow[d] \\ & \boxed{E_{x\sim P}[L^-(x)] - \pi\, E_{x\sim P+}[1 - L^+(x)} \end{tikzcd}$$

Figure 15:

Figure 15's top equation represents our initial loss function. Since we don't know the set of negative points in a PU dataset, we can replace this portion of the equation with the equation we derived in Figure 14.

Our goal with implementing non-negative PU loss was to use this as our loss function in our novel algorithm. However, we instead went with a propensity weighted loss estimator, as it more closely related to our novel algorithm's goal of estimating an unknown propensity score.

4.2.6 TweetBERT

We implemented TweetBERT by using the code found at the TweetBERT GitHub to tokenize the tweets, and using the tokenized tweets in a simple PyTorch classifier. First, the model feeds all datapoints through the tokenizer, which converts the words in a tweet into a series of real numbers. Next, the dataset is split into a training and test dataset with a 20% split, and the tokenized tweets are fed into the PyTorch classifier. The classifier's architecture consists of four hidden linear layers, with 25% dropout between each layer. The network uses Binary Cross Entropy as its loss function, and uses AdamW as its optimization function. We trained our model for a max of 300 epochs, but noted that the model's training and testing accuracy rose most sharply between 0 and 100 epochs.

4.2.7 TIcE

The implementation of TIcE was taken from a Github repository (https://github.com/dimonenka/DEDPUL), which was an adaptation the author's GitHub. The original TIcE repository (https://github.com/ML-KULeuven/SAR-PU) was created for Python 2, while our codebase was entirely in Python 3. The adaptation was necessary to incorporate the code into our existing infastructure.

4.2.8 Propensity Weighted Loss

The goal of the propensity weighted loss function is to re-weight the loss according to the propensity score of each example. To calculate the unweighted loss, we used binary cross entropy loss. We assumed two cases: one where we know the true propensity score, and one where the propensity score was unknown. In the case where the propensity score was unknown, we used an Expectation Maximization algorithm to model the propensity function. Here are the steps we took to implement and properly use the propensity weighted loss function: 1. Generate sample data. 2. Generate the propensity function. 3. Use the propensity function to generate a propensity score e for a given sample. If we are using the EM algorithm, it is assumed that the propensity function is unknown and must be estimated. If the sample is positive, unlabel it with the probability $1 - e$. Store the propensity scores such that they can be used by the loss function. 4. Apply a simple Neural Network with the propensity weighted loss function using the stored propensity scores.

Below is the pseudo-code of the propensity weighted loss function:

```
function PROPENSITY_WEIGHTED_LOSS($y$, $\hat{y}$, $e$)
    $total\_risk \leftarrow 0$
    for $y_i$, $\hat{y}_i$, $e_i$, ..., $y$, $\hat{y}$, $e$ do
        $a \leftarrow positive\_log\_loss(y_i) * \frac{1}{e_i}$
        $b \leftarrow negative\_log\_loss(y_i) * (1 - \frac{1}{e_i})$
        $c \leftarrow (y * (a + b)) + ((1 - y) * negative\_log\_loss(\hat{y}))$
        $total\_risk \leftarrow total\_risk + c$
    return $total\_risk$
```

4.2.9 Subclass Prior Estimation

First, this algorithm seeks to establish the best possible way to group positive labeled clusters of points based on the silhouette score, which is the mean silhouette coefficient across all the clusters.
The silhouette coefficient is calculated by

$$\frac{(b - a)}{max(a, b)}$$

where a is the mean distance between points within the cluster and b is the distance between the cluster to the nearest outside cluster. Points are clustered using the k-means classifier resulting in the best silhouette score. Next, unlabeled points are assigned to their respective cluster using this k-means classifier. From there, a class prior estimation algorithm, such as TIcE, can be applied to each cluster, now consisting of both positive and unlabeled examples. Once α for each cluster is found, the average of all the α, weighted by the size of the unlabeled portion of each cluster, is returned. Shown below is the pseudocode.

```
function K_MEANS_SILHOUETTE($unlabeled$, $positives$, $max\_k$)
    for n = 2,...,max_k + 1 do
        $km\_classifier \leftarrow KMeans(n\_clusters = n)$
        $pred \leftarrow km\_classifier.fit\_predict(positives)$
        $score \leftarrow silhouette\_score(positives)$
        if $score$ is greater than $best\_score$ then
            $best\_km\_classifier \leftarrow km\_classifier$
            $best\_score \leftarrow score$
    return $best\_km\_classifier, pos\_assignments$
function ESTIMATE_ALPHA($unlabeled$, $positives$, $max\_k$)
    $km\_classifier, pos\_assignments \leftarrow k\_means\_silhouette(positives, max\_k)$
    $unlabeled\_assignments \leftarrow assign\_clusters(km\_classifier, unlabeled)$
    for i = 1,...,length(pos_assignments) do
        $alphas[i] \leftarrow TIcE(pos\_assignments[i], unlabeled\_assignments[i])$
        $weights[i] \leftarrow \frac{size(unlabeled\_assignments[i])}{size(unlabeled)}$
    $\alpha^* \leftarrow dot\_product(alphas, weights)$ return $\alpha^*$
```

4.2.10 Novel Algorithm: Cluster-Bias Adjusted Clustering C-BAC

nay The method for creating and implementing this algorithm is a combination of the Subclass Prior Estimation algorithm and the Propensity Weighted Loss algorithm. The steps are similar to the Propensity Weighted Loss algorithm implementation except the propensity scores are discovered by the Subclass Prior Estimation algorithm instead of the use of the SAR-EM model. While the Subclass Prior Estimation algorithm determines the class prior for a given cluster, it is simple to derive the propensity score from that. For a given cluster, let α be the proportion of positive samples in an unlabeled mixture, and p and u be the number of positive and unlabeled samples respectively. The propensity score e of the entire cluster can be calculated as shown below:

$$e = \frac{p}{p + u\alpha} \tag{27}$$

This algorithmic workflow served as the basis of our novel algorithm as the incorporation of the subclass prior estimation algorithm using the tree induction using c estimation algorithm coupled with the propensity weighted risk loss algorithm has never been implemented before. The team's novel workflow was developed through the use of the combination of existing algorithms with the hope that it would have greater accuracy and improved classification as compared to the state of the art approaches.

5 Experimental Analysis

5.1 Datasets

5.1.1 Online Shopping Dataset

The dataset used in the experiments involving the Naive Bayes, SVM, and EM-NB PU classifiers is titled “Online Shoppers Purchasing Intention Dataset”, from the UCI Machine Learning Dataset Repository. This dataset recorded the behaviour of customers while browsing a shopping website. It contained 12,330 data points, each representing a unique user within a 1-year period. This helps the dataset avoid bias towards a particular person or time period. The dataset consists of 10 numerical and 8 categorical attributes. The “Administrative”, “Administrative Duration”, “Informational”, “Informational Related”, “Product Related”, and “Product Related Duration” categories are integer values that represent the number of different pages visited by a user in a session, and the time spent on each page. These numbers were based on the URL information of the pages a user visited. The “Bounce Rate” refers to the percentage of users that enter the site and leave without visiting any other page on the site. The “Exit Rate” displays, for any given webpage, the percentage of users for which that webpage was their final page visited before exiting their session. The “Page Value” attribute refers to the average assigned value of a webpage that a user visited before finishing an online transaction. The “Bounce Rate”, “Exit Rate”, and “Page Value” attributes were calculated using the metrics found in Google Analytics for each page on the site. The “Special Day” feature represents the closeness in time between the access date and any special holiday, such as Christmas or Valentine’s Day. The other features include the access “Month” (String), the user’s “Operating System” (Integer), “Browser” (Integer), “Region” (Integer), “Traffic Type” (Integer), “Visitor Type” (String, such as new visitor or returning visitor), “Weekend” (Boolean), and “Revenue” (Boolean).

We chose “Revenue” as our target variable, as it ultimately showed whether the user purchased a product on the site, which was the goal of the study. This feature was skewed towards the negative class, as out of the 12,330 recorded shoppers, 10,422 people (84.5%) did not make any purchases, and 1,908 people (15.5%) did make a purchase. We believe that this feature represents real world scenarios, and would be a good test for our algorithms.

5.1.2 Twitter Datasets

The Twitter dataset is split into two parts, an unlabeled dataset and a labeled dataset. The labeled datasets are split into two different datasets, one which contains a collection of tweets from a number of users, and another which contains the PHQ9 test results from those same users.

The unlabeled Twitter dataset contains three main columns which are an untitled column that contains the number of each row starting at 0, the second column is the ID of the Twitter user and the third column is the content of the tweet. The initial dataset that was presented to the team was missing filled with blank rows that contained only the row number but no ID and no content for the tweet and these blank rows significantly added to the total amount of rows in the comma separated value (csv) document. The total number of rows in the csv was 65,543 rows and the initial csv also contained multiple repeated tweets by the same user ID which were not retweets. Retweets are when user tweets the same tweet as another user, this would result in a different tweet ID as it is a new user tweeting a tweet with the same content as another user. It is also denoted in the content column as RT before the tweet itself. Furthermore, the initial csv contained multiple repeated tweets by the same user. This would cause a bias to be introduced when working with an PU models as it would adversely effect the model's performance. Therefore, in order to negate this issue, the dataset was cleaned using pandas.

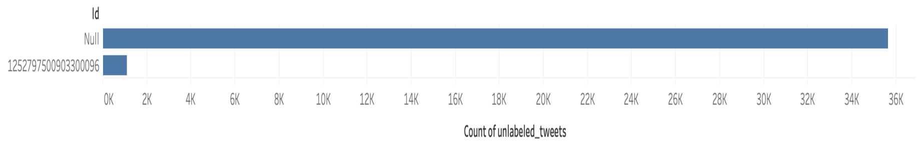


Figure 16: A visual representation of the greatest number of repeated tweets and null value tweets.

The twitter dataset has an accompanying PHQ9 dataset, which contains the test results of the user's PHQ9 score. The PHQ9 is a questionnaire which has been designed to monitor the severity of depression within a patient, and is a tool used by primary health care services. It can be used tentatively at times to help diagnose depression for at risk individuals within a certain demographic of the population. Based on the results of the test, the user will be assigned a score between 0 and 27. Any value within the range of 0-4 represents no depression, 5-9 represents mild depression, 10-14 represents moderate depression, 15-19 represents moderate to severe depression and 20-27 represents severe depression. The dataset contains the score of each question from 0-3 for each user.

5.1.3 Banknote Authentication Dataset

This dataset was found from the UCI Machine Learning Dataset and was used for experimentation regarding the Cluster-Bias Adjusted Classifier. The dataset consists of data that was extracted from images taken from geniune and forged bank-note samples. Then it was digitize using an industrial camera for print inspection. The final images are 400 by 400 pixels and wavelet transformation tools were used to extract features from the images of the bank notes.

The dataset contains the following variables in which 4 variables are continuous and 1 variable is an integer. The variables are as follows:
1. variance of Wavelet Transformed image
2. skewness of Wavelet Transformed image
3. curtosis of Wavelet Transformed image
4. entropy of image (continuous)
5. class (integer)

The class integer is representative of forged bank notes denoted by the value 0 and real, genuine bank notes denoted by the value 1.

5.1.4 HTRU2 Dataset

The dataset used in the experiment involving the Cluster-Bias Adjusted Classifier algorithm is titled "High Time Resolution Universe Survey Dataset", from the UCI Machine Learning Dataset Repository. The HTRU2 dataset describes a sample of pulsar candidates collected during a High Time Resolution Survey. Pulsars are rare types of neutron stars that produce radio emission which are detectable from earth. As the pulsar neutron star rotates, its release radio emission beams which sweep across the sky and produces a detectable pattern for broadband radio emissions. Since pulsars rotate rapidly, this periodically repeating emission pattern is looked for through the use of large radio telescopes. Identifying these periodic patterns are difficult and this is why ML algorithms are being used for rapid detection of pulsars.

The dataset contains a split of 16,529 spurious or false examples of pulsars are 1639 real pulsar examples with a total of 17,898 examples. Therefore, the dataset is unbalanced as there are 16,529 negative examples that are considered as the aforementioned fake pulsars, and 1529 positive examples which are the verified pulsars. Each pulsar datapoint is known as a candidate to be a pulsar and is described by the dataset using 8 continuous variables and 1 single class variable. The variables are as follows:
1. Mean of the integrated profile
2. Standard deviation of the integrated profile
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.
9. Class

The first 4 variables are simple statistics obtained from the integrated pulse profile, which is an array of continuous variables that help describe the frequency and signal of the pulsar over average time. The class variable is an indicator to whether the pulsar is fake represented by 0 or real represented by 1.

5.2 Evaluation Metrics

This section describes the evaluation metrics we used to measure the performance of the algorithms in our experiments.

5.2.1 Method Modification PU Learning Algorithms

Overall accuracy is a simple evaluation metric, however, it does not perform well when working with class skew, or differing cost of false negatives and false positives (Wood, T). Since our dataset had a negative skew, we decided to evaluate our algorithms using the F-score, instead.

As described in section 3.2.5, the F-score measures the harmonic mean of the model's precision and recall. This helps alleviate the issues with using overall accuracy, as it is optimized for binary classification. It is sensitive to the model's performance with regard to the positive class, unlike overall accuracy, which measured the model's ability to classify as a whole.

5.2.2 TweetBERT

Unlike other experiments, TweetBERT is a pre-trained model, so we do not need to measure its performance. Instead, we will use TweetBERT to train a separate classifier on our depression datasets. Therefore, we chose to evaluate only the validation loss and accuracy of our classifier.

5.2.3 TIcE and Subclass Prior Estimation

We reported the class prior of the unlabeled mixture alongside the expected class priors. We also reported the MSE.

5.2.4 Propensity Weighted Loss

We used accuracy as the main metric. We also compared the accuracy of the propensity weighted loss function against the accuracy from a standard MSE loss function.

5.2.5 Cluster-Bias Adjustment Classification C-BAC

We used accuracy as the main metric. We also reported the computed propensity scores for each cluster for more insight. Furthermore, we plotted the final plot that shows how the classification algorithm performed in terms of classifying each cluster. This was done to understand if the clusters were being classified in an optimal fashion.

5.3 Experiment Descriptions

5.3.1 Method Modification PU Learning Algorithms

Our experiment involved taking the PU version of , and testing their performance against each other. We hypothesized that the two-step classifiers would perform better than the one-step classifiers. To conduct our experiment, we trained each one of our classifiers on our dataset. We started the training procedure with 70% of the dataset and 0-90% label prior, using increments of 10%. We applied the SCAR assumption in our preprocessing functions in our pipeline, by converting certain increments of our positively labeled data points into unlabeled points.

After training our model, we used the remaining 30% of our dataset to test our model. Each test outputs four numbers: the label prior, the overall accuracy, the unlabeled accuracy, and the F-score. Our group decided to use the F-score as the measurement of the model's accuracy. We tested each model five times per label prior level, and recorded our results in the graphs below.

Label Prior	1 Step NB	1 Step SVM	2 Step NB-SVM	2 Step EM-NB
0%	0%	0%	0%	0%
10%	17%	18%	39%	35%
20%	34%	32%	49%	46%
30%	48%	48%	55%	56%
40%	60%	57%	61%	60%
50%	67%	68%	65%	67%
60%	73%	77%	71%	68%
70%	83%	81%	74%	71%
80%	90%	89%	75%	75%
90%	95%	94%	77%	78%

Table 1: F-Score: Label Prior vs. Algorithm

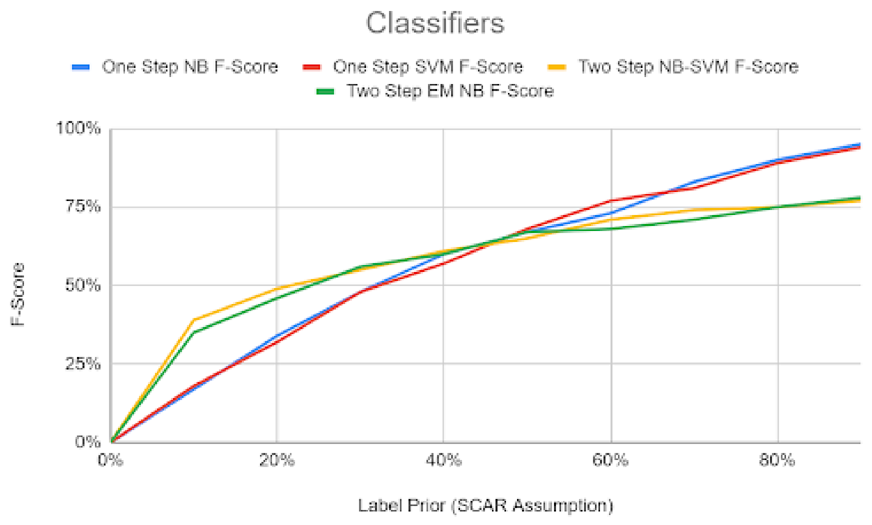


Figure 17: A chart displaying our F-scores for each type of classifier at increments of 10% label prior.

5.3.2 TweetBERT Classifier

We ran many experiments on our TweetBERT classifier in an attempt to accurately classify the tokenized tweets. The result shown in Figure 23 represents our best result, when training and testing our classifier over 300 epochs. The average training accuracy from epochs 0-150 was 0.6285, and the average training accuracy from epochs 151-300 was 0.6630.

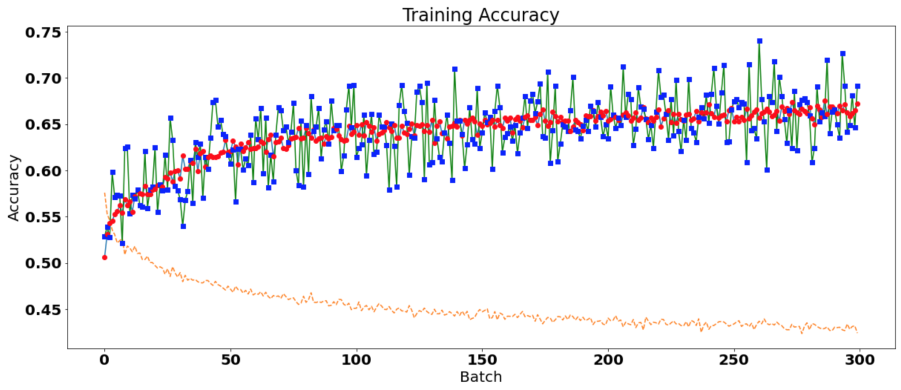


Figure 18: A graph mapping the performance of the TweetBERT classifier over 300 epochs. The blue lines and red dots represent the training accuracy, the green lines and blue squares represent the testing accuracy, and the dotted line represents the loss.

5.3.3 TIcE

The state of the art algorithm for class prior estimation under the SCAR assumption is TIcE. We elected to test TIcE under both ideal and challenging circumstances. Of particular interest to us was how TIcE performed when the positive and negative distributions overlapped. For our experiments we used Gaussian distributions for both the positive and negative classes.

Because the degree of overlap between two distributions is affected by the distance between means, two different standard deviations, and the relative size of the distributions, we have elected to only vary the true class prior and standard deviation of the negative distribution. The distance between the two distribution means was fixed at 4. We evaluated each setting at various levels of α. While it is earlier stated that α is the proportion of positives in a mixture, for our experiments, α represents the proportion of negative points in the mixture. The results for this experiment can be found in Appendix D (8.4)

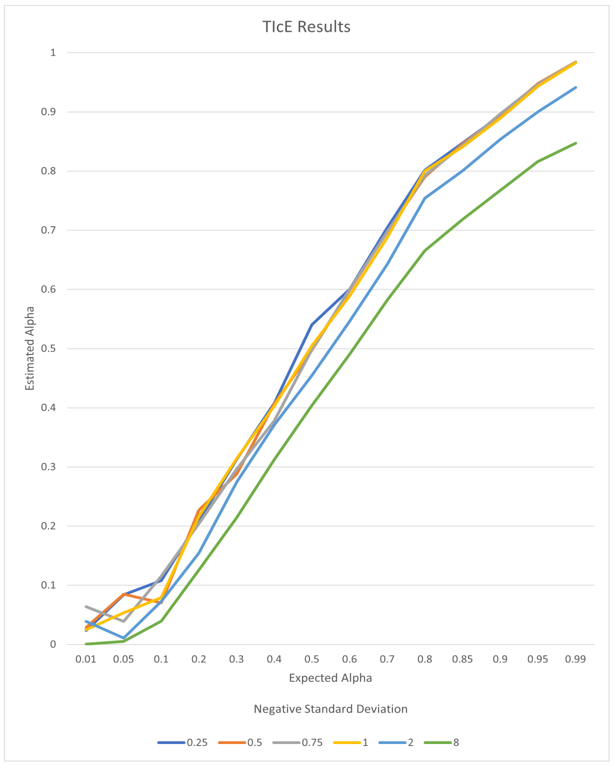


Figure 19: Imaging showing the performance of TIcE at various levels of alpha. Each differently colored line corresponds to a different standard deviation of the negative distribution

5.3.4 Propensity Weighted Loss

For our experiments, we generated a set of one-dimensional data to test how the propensity weighted loss function performs under various conditions. The training data and testing data is generated with identical parameters. Some parameters of the data varied according to the experiment and some of the parameters were fixed. the positive $F_+(x)$ and negative $F_-(x)$ distribution both contain 5000 points. Respectively, $F_+(x)$ and $F_-(x)$ are as follows:

$$F_+(x) \sim Normal(\mu = 2, \sigma = 1) \tag{28}$$

$$F_{-}(x) \sim Normal(\mu, \sigma) \tag{29}$$

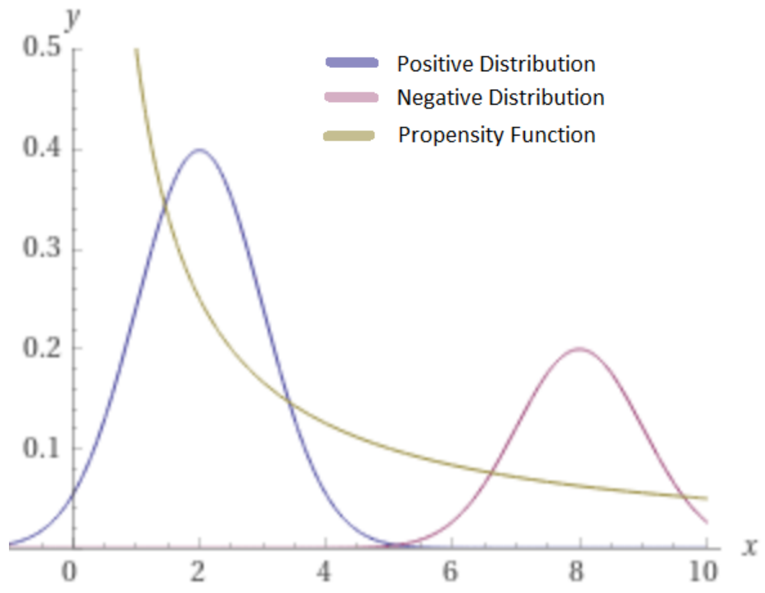


Figure 20: Imaging showing the frequency distribution of the positive and negative classes, along with a given propensity function, which determines the label probability of a given positive sample

We tested the propensity weighted loss function under two scenarios. The first scenario assumes that the propensity function is known. In our tests, let these functions be represented below:

$$f_1(x) = min(1, \frac{0.25}{x}) \tag{30}$$

$$f_2(x) = min(0.8, max(0.2, \frac{0.5}{x})) \tag{31}$$

$$f_3(x) = U(0, 1) \tag{32}$$

$$f_4(x) = 0.5 \tag{33}$$

Propensity Function	μ_-	σ_-	Accuracy	MSE Loss Function Accuracy
f_1	4	2	0.7665	0.5129
f_1	4	1	0.8397	0.5246
f_1	10	1	0.9954	0.5318
f_1	8	2	0.9776	0.5316
f_2	8	2	0.9762	0.5816
f_3	8	2	0.9771	0.7322
f_4	8	2	0.9704	0.7333

Table 2: Classification Accuracy with known Propensity Scores. A standard MSE loss function, which disregards propensity scores, was evaluated as well for comparison.

The second scenario assumes that we do not know the true propensity function when apply the propensity weighted loss function. For this, we used an Expectation Maximization algorithm to estimate the propensity function.

Propensity Function	μ_-	σ_-	Accuracy1	Accuracy2
f_1	8	2	0.9435	0.5
f_2	8	2	0.9779	0.5
f_4	8	2	0.9597	0.5

Table 3: Classification Accuracy with estimated Propensity Scores. The standard MSE loss function accuracy was not included as those were evaluated in the table above. Accuracy1 is the accuracy when adjusting for the propensity bias in the estimated propensity function. Accuracy2 has no such adjustment done.

It is important to note that the EM algorithm does not accurately model the propensity function in many cases. The effect on accuracy depends on the properties of the calculated propensity function. With the EM algorithm, underestimating the propensity values can have severe consequences on the classification accuracy. The error on the prediction of the propensity function is known as the Propensity Weighted Estimator bias (Bekker et al., 2019). To combat this bias, we bound the propensity function to an arbitrary value of our choosing such that high accuracy scores were still obtained. In our experiments we chose 0.1 as the minimum calculated propensity score. While this method is quite rudimentary, it worked with our tests and was sufficient in demonstrating the effects of propensity bias.

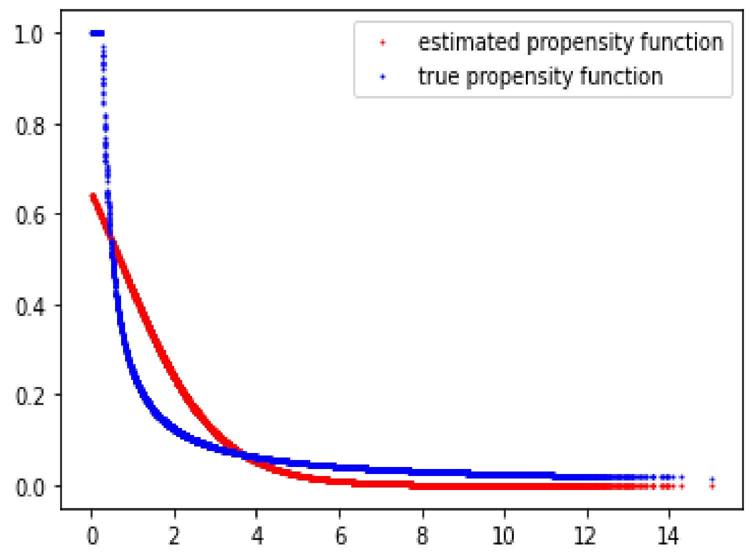


Figure 21: Image showing the propensity function found by the EM algorithm versus the true propensity function

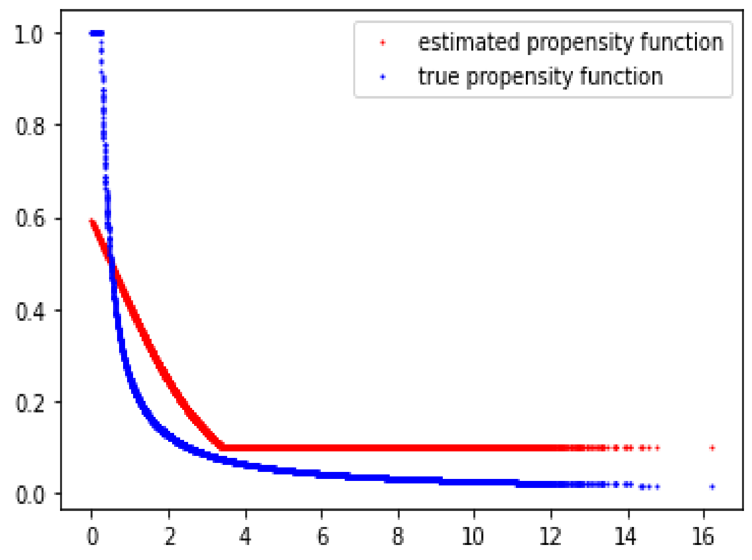


Figure 22: Image showing the propensity function found by the EM algorithm with bias correction versus the true propensity function.

The bias can be calculated using the Propensity Weighted Estimator Bias equation shown below.

$$bias(\hat{R}(\hat{\mathbf{y}}|\hat{\mathbf{e}}, \mathbf{s})) = \frac{1}{n}\sum_{i=1}^{n} y_i(1 - \frac{e_i}{\hat{e}_i})\,(\delta_1(\hat{y}_i) - \delta_0(\hat{y}_i))$$

Figure 23: Imaging showing the Propensity Weighted Estimator bias (Bekker et al., 2019)

The bias function shows that underestimated propensity scores can have an out-sized effect on bias. The EM-derived propensity functions estimated extremely small propensity scores in some parts of the function, and it was necessary to limit its effects. In our tests, severely underestimating the propensity scores caused the Neural Network to predict each and every sample as positive. This is due to the fact that incorrect and very small propensity scores will predict the positive class to be a lot more prevalent than it really is. On the contrary, while overestimating propensity scores, in theory shall still lead to bias, the error limit is bounded, thus the effects on the overall accuracy are limited. In comparison, the error from underestimating propensity scores is unbounded. This is because of $\frac{e_i}{\hat{e_i}}$ which has a range of e_i to ∞ depending on $\hat{e_i}$. In our experiments, despite the risks of overestimating the true propensity score by limiting the estimated score to not go below 0.1, the potential risks of not doing so is far greater.

5.3.5 Subclass Prior Estimation

Similarly to the data generated for the TIcE experiments, there are two main components: one positive and one negative. The difference this time is that there are multiple separate positive distributions. Because the subclass prior estimation algorithm uses k-means to find sub-classes (see section 2.7.7 for more details), the ideal data-set for testing this algorithm will have a cluster for each positive subclass. For this experiment, we generated data points points in two dimensions. Each positive cluster consists of both positive labeled data and positive unlabeled data. This cluster is generated in two steps. First a positive labeled distribution is generated. Next, positive unlabeled and negative data is generated. We can call this an unlabeled mixture. Refer to figure 11. How much of each depends on α. While it is earlier stated that α is the proportion of positives in a mixture, for our experiments, α represents the proportion of negative points in the mixture. This doesn't change anything and is essentially the same thing. The positive distribution of the mixture is created with the same mean and standard deviation of the positive labeled distribution while the negative distribution is centered elsewhere. For all cluster generations, the negative distributions are created with the exact same parameters since we only want one negative distribution in all.

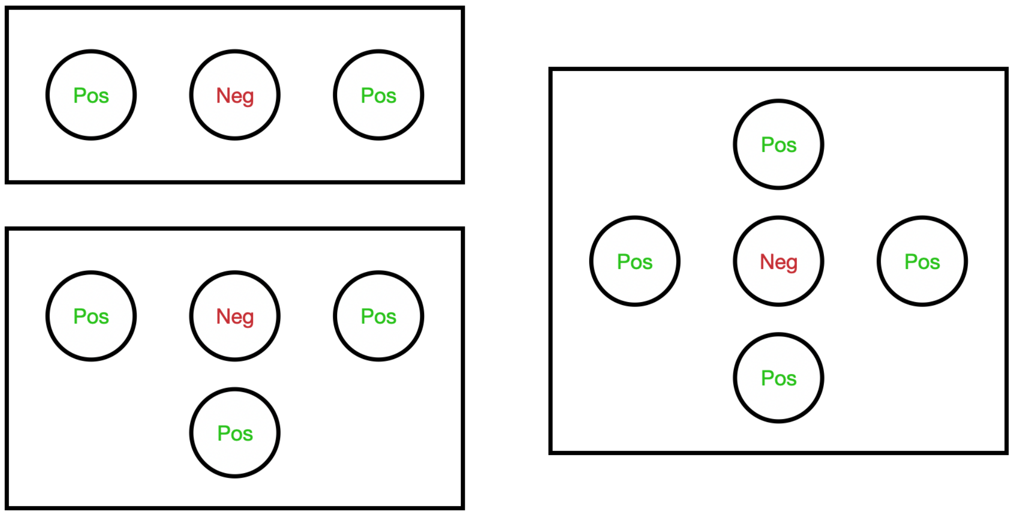


Figure 24: Imaging showing roughly how the positive clusters of size 2,3, and 4 are structured with the negative distributions in the centers.

There are many parameters in the generated data that can be changed. In our first experiment, the parameters we chose to vary were: average alpha of each positive cluster, number of positive distributions, and the standard deviation of the negative distribution. The positive and negative distributions are each Gaussian distributions. There are other parameters that may have significant impact on our results, but in order to keep the amount of experimental data manageable, these parameters were fixed. Parameters such as the standard deviation of the positive distributions, the positioning of the positive distributions relative to each other and to the negative distribution, and positive distribution sizes were fixed in this experiment. We set the size of each of the positive distributions as a random variable following a normal distribution with a mean of 3000 and a standard deviation of 500. This size refers to the number of positively labeled points to be generated for the positive cluster. The standard deviation of the positive distributions were set to 0.1. The alphas for each cluster follow a normal distribution with a mean of the target alpha, and with a standard deviation of $min(0.05\alpha, 1 - \alpha)$. The size of negative distribution varies with the number of cluster and the alphas of the positive distribution, but it has a maximum size of 10000. The means of all the positive distributions are located 16 points away from the mean of the negative distribution. Additionally, each iteration of the experiment was run 10 times and averaged to produce a more precise value. The results for this experiment can be found in Appendix A (8.1).

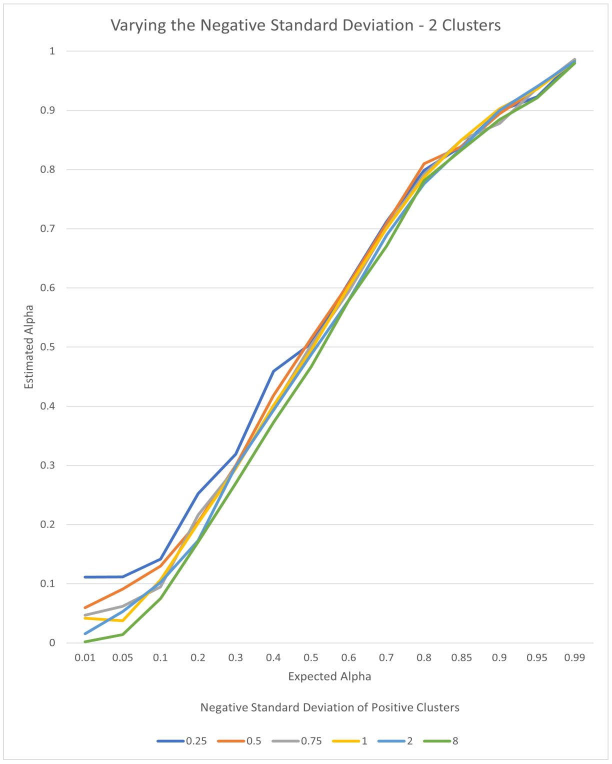


Figure 25: Algorithm performance with 2 positive clusters around the negative distribution. It shows the performance at various levels of alpha. Each differently colored line corresponds to a different standard deviation of the negative distribution.

In the second experiment, instead of varying the negative standard deviation, we varied the mean positive cluster size. The positive cluster size is the number of positively labeled points in a positive cluster. The standard deviation for the negative distribution was fixed at 0.5. The rest of the fixed experimental parameters remained the same as the first experiment. The results for this experiment can be found in Appendix B (8.2).

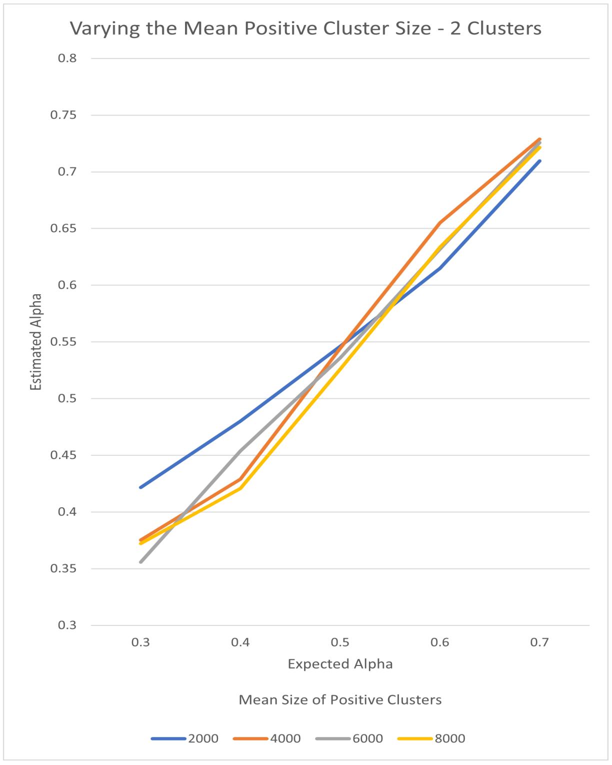


Figure 26: Algorithm performance with 2 positive clusters around the negative distribution. It shows the performance at various levels of alpha. Each differently colored line corresponds to a different mean size that is used for the positive clusters. The same mean is applied to each of the positive clusters.

In the third experiment, we varied the mean positive cluster standard deviation. The same as the second experiment, the negative distribution standard deviation was fixed to 0.5 as well. The rest of the fixed experimental parameters remained the same as the first experiment. The results for this experiment can be found in Appendix C (8.3).

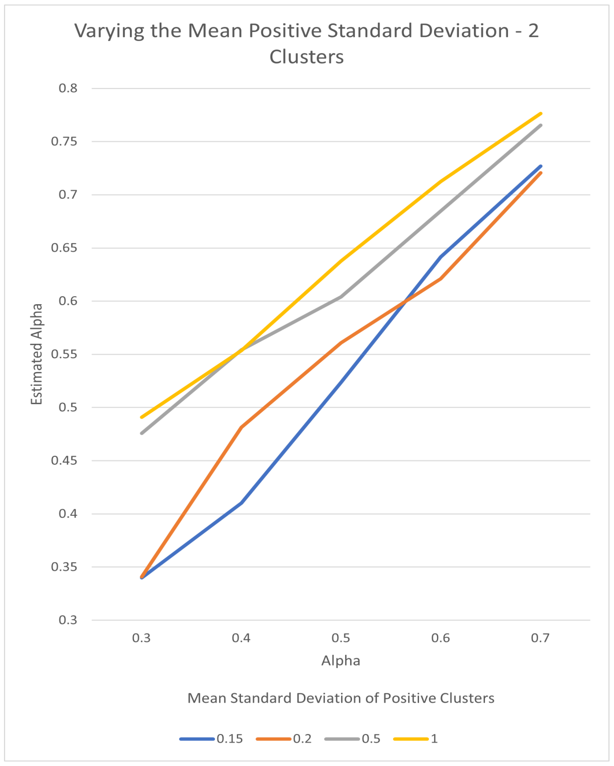


Figure 27: Algorithm performance with 2 positive clusters around the negative distribution. It shows the performance at various levels of alpha. Each differently colored line corresponds to a different mean standard deviation of the positive distributions. The same mean is applied to each of the positive clusters.

5.3.6 Cluster-Bias Adjustment Classification (C-BAC) with generated data

??

For these experiments, we generated two-dimensional data to evaluate how the propensity weighted loss function performs with propensity scores determined by the subclass prior estimation algorithm. The data generated was similar to the data generated for the experiments involving just the subclass prior estimation algorithm. The training data and testing data was generated with identical parameters. The negative data $F_{-}(x, y)$ was generated as one cluster.

$$F_{-}(x,y) \sim Normal(\mu_x, \sigma_x), Normal(\mu_y, \sigma_y) \tag{34}$$

The positive data was formed in clusters. We ran tests for the two-cluster and three-cluster scenario. For the two-cluster scenario, the positive clusters were each spaced 8 units away from the center of the negative cluster on the left and right sides. The three-cluster scenario is the same except an extra positive cluster is added 8 units away on top from the center of the negative cluster.

Each cluster $F_{+}^{i}(x,y)$ is represented as:

$$F_{+}^{i}(x,y) \sim Normal(\mu_x, \sigma_x), Normal(\mu_y, \sigma_y) \tag{35}$$

All positive clusters across all tests have the same standard deviation.

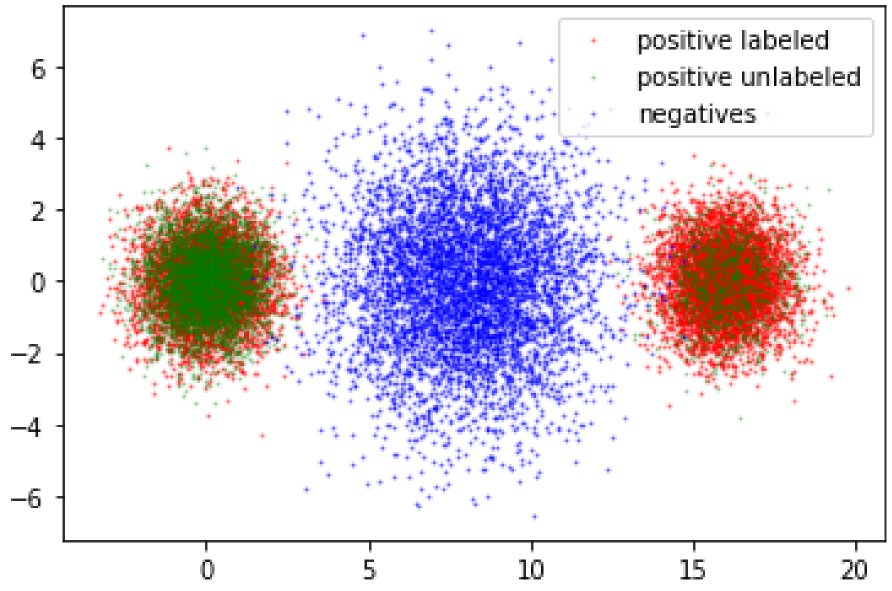


Figure 28: 2-cluster generated data with two different propensity scores

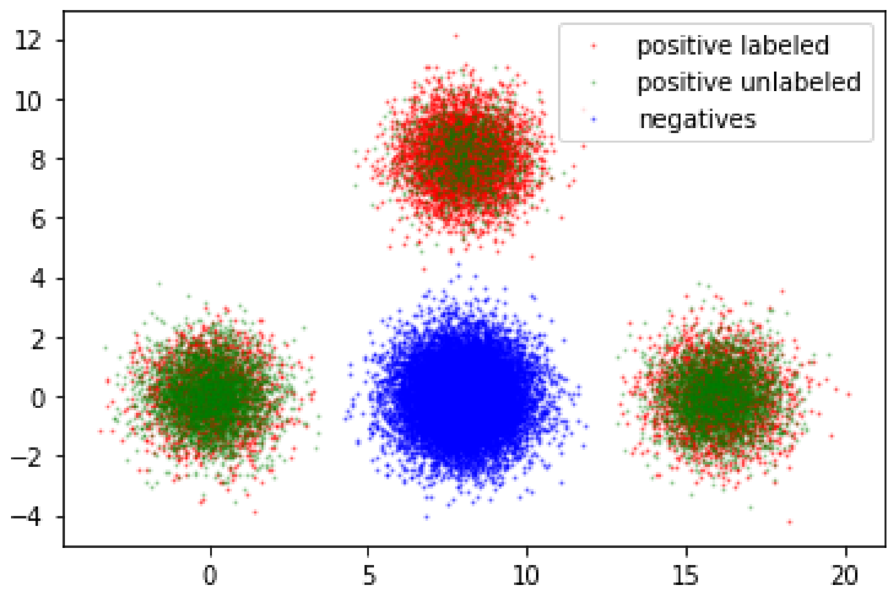


Figure 29: 3-cluster generated data with three different propensity scores

Each positive cluster contained 5000 points while the negative cluster contained the number of samples denoted as 5000 multiplied by the number of positive clusters. This ensures that there are equal amounts of positive and negative samples. Once the positive and negative samples were generated, the positive data for each cluster was unlabeled according to a chosen propensity score. This chosen propensity score varied depending on the experiment.

e_1	e_2	σ_-	Accuracy	$\hat{e_1}$	$\hat{e_2}$
0.25	0.25	1	0.99825	0.30187	0.31828
0.25	0.25	2	0.97240	0.31788	0.30772
0.25	0.25	4	0.94870	0.26784	0.24031
0.40	0.40	1	0.99755	0.54912	0.54041
0.40	0.40	2	0.98870	0.43588	0.51603
0.40	0.40	4	0.94840	0.37563	0.41687
0.60	0.60	1	0.97450	0.65727	0.72416
0.60	0.60	2	0.99455	0.63037	0.62365
0.60	0.60	4	0.93820	0.58805	0.56953
0.75	0.75	1	0.99975	0.76425	0.76921
0.75	0.75	2	0.99375	0.75369	0.74953
0.75	0.75	4	0.95300	0.72678	0.75945
0.40	0.60	1	0.99615	0.64734	0.50815
0.40	0.60	2	0.98875	0.62933	0.43918
0.40	0.60	4	0.93405	0.42117	0.56637
0.25	0.75	1	0.74215	0.76010	0.42275
0.25	0.75	2	0.87540	0.28965	0.74831
0.25	0.75	4	0.68130	0.76351	0.30947

Table 4: 2-cluster results. e is the input propensity score while $\hat{e}$ is the estimated propensity score for each cluster. e and $\hat{e}$ also do not match up on the table.

e_1	e_2	e_3	σ_-	Accuracy	$\hat{e_1}$	$\hat{e_2}$	$\hat{e_3}$
0.25	0.25	0.25	1	0.98710	0.31686	0.30942	0.30702
0.25	0.25	0.25	2	0.98867	0.29606	0.34415	0.31783
0.25	0.25	0.25	4	0.92203	0.27343	0.26786	0.25448
0.40	0.40	0.40	1	0.98677	0.49068	0.48662	0.46416
0.40	0.40	0.40	2	0.99180	0.42213	0.45664	0.49808
0.40	0.40	0.40	4	0.93057	0.41976	0.42899	0.38033
0.60	0.60	0.60	1	0.99927	0.63256	0.64308	0.71597
0.60	0.60	0.60	2	0.99217	0.63978	0.76430	0.66543
0.60	0.60	0.60	4	0.93530	0.59145	0.58011	0.62728
0.75	0.75	0.75	1	0.99940	0.77005	0.76822	0.82787
0.75	0.75	0.75	2	0.98993	0.79120	0.75878	0.77188
0.75	0.75	0.75	4	0.92823	0.73828	0.70122	0.74237
0.40	0.60	0.80	1	0.99300	0.62747	0.83086	0.43264
0.40	0.60	0.80	2	0.98823	0.79924	0.69115	0.49645
0.40	0.60	0.80	4	0.90140	0.37640	0.78180	0.62162
0.15	0.50	0.85	1	0.75333	0.59244	0.86038	0.23210
0.15	0.50	0.85	2	0.79303	0.83868	0.59731	0.21060
0.15	0.50	0.85	4	0.55140	0.82486	0.50368	0.17982

Table 5: 3-cluster results. e is the input propensity score while $\hat{e}$ is the estimated propensity score for each cluster. e and $\hat{e}$ also do not match up on the table.

One of the key takeaways from this experiment was understanding the weakness within the classification algorithm as with more positive clusters around the negative cluster with varying propensity scores, the classification algorithm was unable to accurately classify the clusters. This in turn led to a test accuracy of 0.82 as an upper limit. Thus this led to another experiment to understand where the classification algorithm was being capped in terms of performance.

5.3.7 C-BAC Classifier Accuracy

For these experiments we used the same generated data that's aforementioned in the previous section. The purpose of this experiment was to understand the bounds of the classification algorithm in terms of how it could classify each of the positive clusters depending on their positions. Multiple experiments were performed to understand the limit of the classification algorithm with respect to cluster position.

The set of experiments can be broken down into the number of positive clusters and their relative position as compared to the negative cluster. The first experiment involved two positive clusters that had a propensity score of 0.85 and 0.15 with varied positioning around the negative cluster. The first position tested was having one cluster to the left of the negative distribution and another just above the negative distribution. The next was to test 2 positive clusters to the right and the left of the negative clusters. The final outcome of the classification algorithm is shown below in a plot.

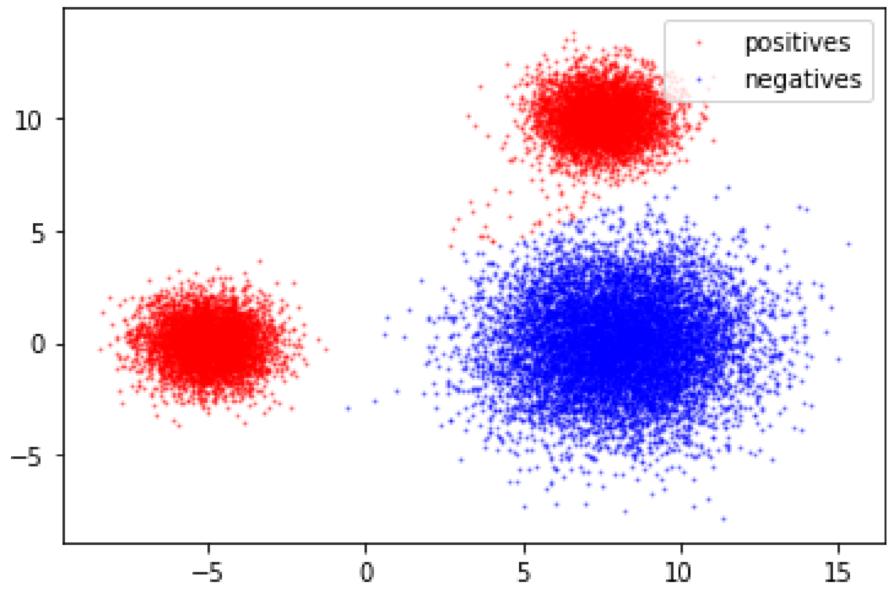


Figure 30: 2-cluster generated with one to the right and one on top of the negative distribution, figure shows the classification end result from the C-BAC algorithm

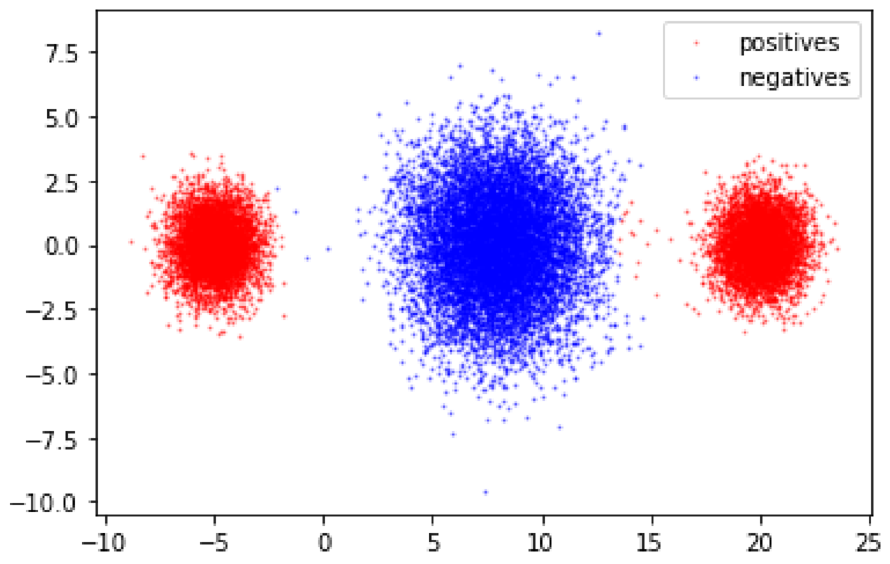


Figure 31: 2-cluster generated with one to the right and one to the left of the negative distribution, shows the classification end result from the C-BAC algorithm

The next set of experiments involved using 3 positive clusters in a triangular shape around the negative cluster. The values of the propensity scores were maintained to be more accurate to real world data and therefore 2 clusters on each side of the negative cluster had a propensity score of 0.15 and one cluster on top of the negative cluster had a propensity score 0.85.

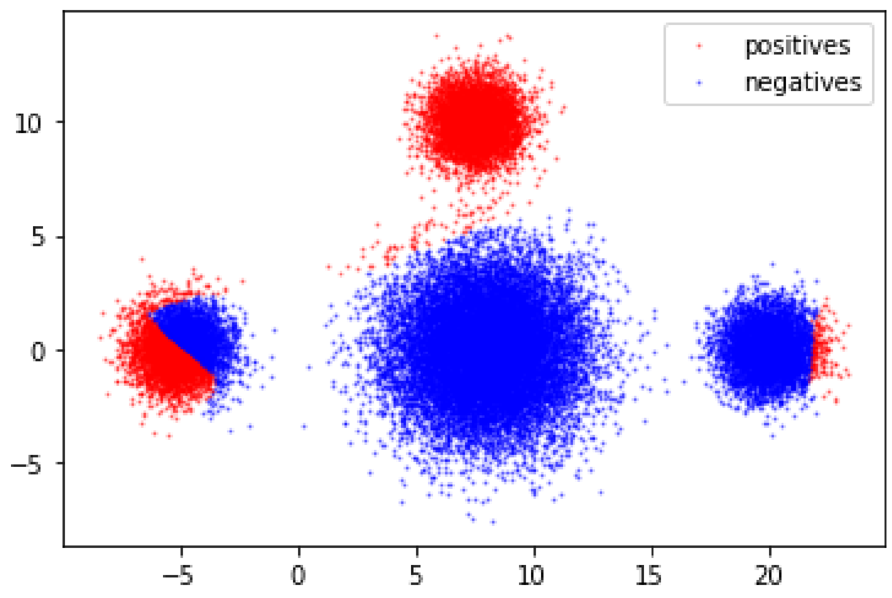


Figure 32: 3-cluster generated in triangular formation around the negative distribution, figure shows the classification end result from the C-BAC algorithm

As can be seen in figure 32, that shows the experiment above the classification algorithm is unable to accurately classify the clusters with the lower propensity scores. It was able to reach an average propensity score of 0.76 over the 5 runs of the experiment and one of the plots generated is shown as the figure above. The classification algorithm struggled to identify the clustering configuration that would be optimal. Furthermore, one more experiment was run using 4 positive clusters around the negative distribution with 2 clusters to the left and right having 0.15 propensity scores and the top and bottom clusters having 0.85.

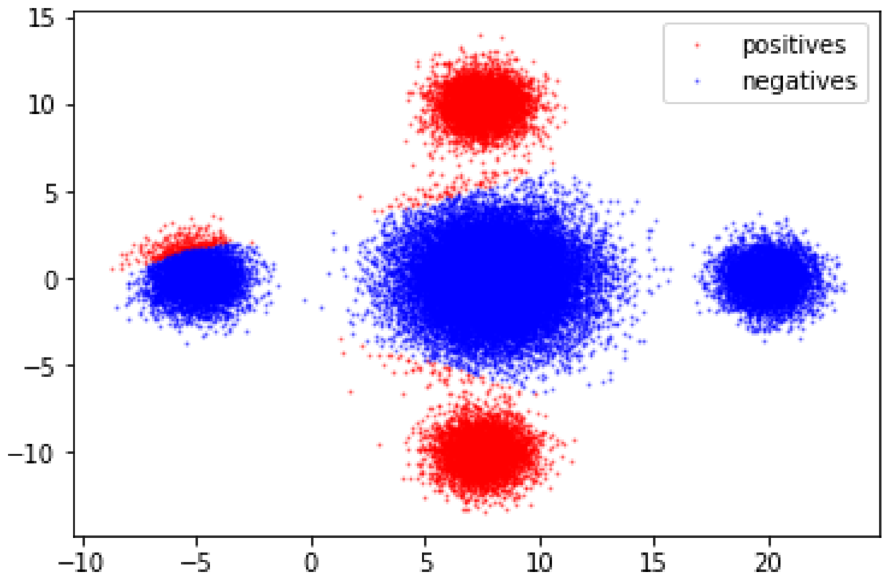


Figure 33: 4-cluster generated in a diamond formation around the negative distribution, figure shows the classification end result from the C-BAC algorithm

As with the experiment involving the 3 cluster set up in the triangular formation, a similar result can be seen in the 4 cluster diamond formation. The classifier is unable to accurately classify the clusters to the right and the left of the negative distribution accurately.

Each experiment was run for 50 epochs and the main outcome was to see how the classification algorithm classified each cluster. Furthermore, one idea that the team had to improve the classification was to cap the torch.loss at 0 therefore preventing any potential overfitting, but through the course of these experiments it proved to have no difference in terms of the testing accuracy and the classification of the clustering.

In order to address the issue that arose during this experimentation, the team worked on the code and refactored the code to ensure that they were no bugs in the code that prevented the classification algorithm from performing well. Once the code was refactored and re-written to become more understandable and neater, the classification algorithm performed with optimal accuracy and was able to classify the clusters correctly and as expected. With the new and improved code the team was ready to try and implement the C-BAC algorithm on real world datasets to test its performance.

5.3.8 C-BAC and HTRU Experimentation

After running various experiments on the C-BAC dataset using generated clustered data, the team decided to implement the algorithm on real world datasets. Therefore, the C-BAC algorithm was run on HTRU dataset to test the novel algorithm's applicability. In order to run the experiment, two clusters were set alpha values which represent their true propensity scores and these were kept constant to 0.5 and 0.75. Then the C-BAC algorithm was run that uses k-means to identify the clusters and the TiCE algorithm to estimate the alpha values. Each experiment was run for 15 epochs as after that the accuracy wouldn't increase significantly and started to reach its upper limit.

Experiment Trial	Est. α (0.5)	Est. α (0.75)	Test Accuracy
1	0.016	0.811	0.959
2	0.021	0.778	0.974
3	0.070	0.796	0.976
4	0.801	0.065	0.980
5	0.031	0.817	0.972
6	0.810	0.014	0.941
7	0.808	0.137	0.982
8	0.785	0.084	0.973
9	0.788	0.085	0.974
10	0.822	0.017	0.956

Table 6: Shows the estimated alphas values and test accuracy for the 10 experimental runs done using C-BAC on the HTRU dataset

For the 10 experimental trials that were done the mean accuracy is 0.968 with a standard deviation within the accuracy of 0.012. The C-BAC algorithm is able to achieve a high accuracy, however, since the data isn't in the optimal clustering configuration the estimated alpha values for 0.50 and 0.75 aren't as accurate or have a low standard deviation like the test accuracy. The mean of the estimated alpha for 0.5 over the 10 experimental trials was 0.495 and the standard deviation was 0.377. The mean value is close to the 0.5 true values of the 10 experiments, however, the standard deviation is high and therefore shows that none of the individual values throughout the experiments were accurate. The mean of the estimated alpha for 0.75 over the 10 experimental trials was 0.360 and the standard deviation was 0.361. Therefore, as it can be seen through the average and standard deviations of the estimated alphas for 0.75, the dataset doesn't configure well as the C-BAC algorithm isn't able to find optimal clustering in order to make accurate estimations.

5.3.9 C-BAC and Banknote Authentication Experimentation

Another real world dataset was used to run another experiment with the novel C-BAC algorithm which was the banknote authentication dataset. Again a similar experimental set up was used as aforementioned in the previous subsection. The alphas were set to 0.50 and 0.75 and the estimated alphas and test accuracy scores were generated over the use of 30 epochs as that's where test accuracy reached its upper limit.

Experiment Trial	Est. α (0.5)	Est α (0.75)	Test Accuracy
1	0.311	0.236	0.961
2	0.263	0.232	0.957
3	0.255	0.133	0.958
4	0.114	0.500	0.941
5	0.296	0.033	0.939
6	0.195	0.044	0.921
7	0.258	0.326	0.958
8	0.244	0.442	0.963
9	0.184	0.161	0.961
10	0.093	0.321	0.943

Table 7: Shows the estimated alphas values and test accuracy for the 10 experimental runs done using C-BAC on the Banknote Authentication dataset

As it can be seen from table 7, the C-BAC algorithm has a high accuracy score for the 10 experimental runs and has a mean accuracy of 0.949 with a standard deviation of 0.013. This shows the accuracy of the C-BAC algorithm is effective as it also has a low standard deviation and a high mean accuracy. However, for the estimated alphas for estimating the 0.5 alpha value, the mean was 0.221 and the standard deviation was 0.069. The mean and standard deviation for estimating the alpha value of 0.75 are 0.245 and 0.188 respectively. Again, the dataset doesn't provide the optimal clustering configurational setup for the C-BAC algorithm and therefore the estimated alpha means are inaccurate and have greater standard deviations then what would be expected.

6 Discussion

This section describes the implications of each of our experiments we performed throughout the project. We learned from this analysis to create better experiments in the future.

6.1 Method Modification PU Learning Algorithms

Our first experiments were the ones we ran on the PU versions of naive bayes, SVM, EM-NB, and EM-SVM. A key takeaway from these experiments was the inappropriate the use of accuracy as a performance metric. We used binary classifiers on a skewed dataset, and as such, each of our models accuracies were similar. Since most of our labels were negative (84.5%), our model effectively could classify each unlabeled point as negative, and be correct 84% of the time.

After these initial experiments, we switched to using F-score, which is the preferred metric for binary classifiers. The resulting F-scores showed that all of the algorithms performed poorly when the label prior was low. As the label prior increased, the F-score of every algorithm also increased. Between 0% and 15% label prior, the F1-score of the two-step algorithms increased at a rate faster than the F1-score of the one-step algorithms. After this point the F1-score of the two-step algorithms began increasing at a rate lower than those of the one-step algorithms. Between 40% and 50% label prior, the F1-scores of the one-step and two-step algorithms is approximately equal. At label prior percentages greater than 50%, one-step algorithms start outperforming the two-step algorithms. We believe this is because using two-step algorithms introduces a cost to overall classification ability. This cost is offset by increased performance on lesser label priors. At higher label priors, however, the gains are eclipsed by worse classification performance.

6.2 TweetBERT Model

Our PyTorch classifier did not yield impressive results. Although the average training accuracy of the classifier increased steadily in the first hundred epochs, its average training accuracy soon leveled out in later epochs. This poor performance could be attributed to a number of factors. Firstly, the classifier was only trained on one dataset, so the possibility of dataset bias is present. However, a much more likely reason is that many of the tweets in out Twitter dataset were in a language other than English, and TweetBERT was only trained on English tweets. This difference in language effectively leaves many of the points in the dataset unclassifiable, and causes a large amount of noise. To solve this issue, our team would either need to train our classifier on a new dataset, purge all non-English tweets from our current dataset, or find some way to translate the tweets into English. Solving this issue will be a problem our team explores in the future.

6.3 TIcE

Subsequent experiments delved further into exploring other PU algorithms that could form the basis of our novel algorithm. The implementation of the AAAI TIcE paper allowed the team to understand the effects of clustering the data for classification in a PU domain. This research led to our team implementing the propensity risk weighted estimator from the paper, "Beyond the Selected Completely at Random Assumption for Learning from Positive and Unlabeled Data". We used that loss function from this paper to train a simple classifier, and compared it against the MSE loss function. Both loss functions were used to train a classifier, and the accuracy score showed great improvement with the propensity weighted risk estimator. After forming our novel algorithm, we tested its performance in a similar manner.

Figure 19 shows that TIcE performs well at intermediary class prior levels, but one the class prior becomes too high or too low the results become unreliable. This is likely due to an over-correction in the bounds setting portion of the algorithm. Likewise we find that the algorithm consistently tracks worse with progressively higher standard deviations. This is to be expected, as having a higher degree of overlap would intuitively make it harder to reason about the positive distribution.

6.4 Subclass Prior Estimation

This experiment involved varying the standard deviation of the negative distribution, and the results showed good performance with our fixed parameter settings. When fixing the standard deviation of the positive cluster distributions, tight distributions were needed in order to achieve a good level of estimation. That is why the fixed standard deviation values for the positive clusters seem relatively small. In our results, one thing that stands out is that lower values of α, which for our purposes, is defined as the proportion of negatives in an unlabeled mixture, yields worse estimation results. In general, lower standard deviations of the negative distribution influenced the outcome to be more accurate, but this is more pronounced at lower levels of α. In the second experiment, there was somewhat of an improvement in estimation precision with larger positively labeled clusters. As with the previous experiment, high levels of expected α lead to better estimations. In the third experiment, as we varied the standard deviation of the positive cluster, smaller standard deviation values lead to better estimation precision. This is expected since tighter positive distributions means it is more easily distinguishable from the negative. Overall, across all our experiments, there is no significant difference in results between the two-cluster, three-cluster, and four-cluster experimental setups.

6.5 Propensity Weighted Loss

The results demonstrate that the propensity weighted loss function works very well in achieving high accuracy for PU data. On the other hand, a standard MSE loss function does not achieve high accuracy. This is because the MSE loss function assumes all labeled points as positives and unlabeled points as negative, which is wrong. These experiments prove that the true classes of unlabeled examples can be used in training when we account for the label probability, or propensity score, of a given sample. In the case that we assume that the true propensity scores are unknown to the propensity weighted loss function, the expectation maximization algorithm somewhat models the propensity function. While it is able to discover the general trend, the exact propensity scores can be off by quite a bit. Despite this, we saw that the demonstrated accuracy values from our experiments were still quite high. This could be due to the simplicity of the data - the data is only one dimensional. This could also be due to the fact that the propensity weighted loss function can function rather well when predicted propensity scores are off. For very small estimate propensity values, limiting the estimated propensity scores to 0.1 as a simple adjustment for bias works well, even if this means propensity scores below 0.1 are adjusted to be more inaccurate in some case. This is because the benefits out-weight the costs. The total amount of biased reduced is vastly greater than the total amount of new bias introduced.

6.6 C-BAC Algorithm Experiments

Just like the prior experiments involving the propensity weighted loss function, the accuracy is generally good for most cases. Additionally, the propensity score estimation derived from the subclass prior estimation algorithm is generally accurate. In general, larger standard deviation values for the negative distribution lead to lower accuracy results. This is expected because the positive and negative data sit closer together with looser distributions. The accuracy values are also similar across the board for most varying propensity scores when all clusters carry the same propensity scores. However, when the cluster specific propensity scores differ greatly, the accuracy drops quite substantially. The reason is unknown and could be further investigated in the future to improve the algorithm. The same general conclusion can be reached when analyzing the results for both two-cluster scenario and the three-cluster scenario.

Through the experimentation done to identify how the C-BAC algorithm was clustering the various clusters in the generated dataset it became clear that with the greater differing propensity scores and different cluster configurations that the algorithm was unable to perform well. Through the refactoring of the code this issue was able to be overcome and solved in order to allow for experimentation with real world datasets.

The experimentation involving the HTRU and banknote authentication showed that the C-BAC algorithm is able to handle real world datasets as both experiments had a high test accuracy of around 0.94 to 0.98. However, it should be noted that the datasets didn't present the optimal cluster configuration that would be best suited for the C-BAC algorithm. Therefore, either better datasets that would be ideal for an experimental set up could be used or the clustering algorithm could be improved instead of using K-means clustering

7 Conclusion

PU datasets cause problems for many classification algorithms, which are not accurately able to classify samples when a significant portion of the dataset is unlabeled. PU problems are common in the real world, such as in disease diagnosis. Our project's goal, then, was to create an algorithm that can accurately classify PU datasets. Such an algorithm could be applied to a number of real world applications.

In an effort to create our novel algorithm, we researched current PU learning methods (Bekker Davis, 2020), and chose to focus on the subclass prior estimation algorithm (du Plessis et al., 2019) and the propensity weighted loss function (Bekker et al., 2020). These algorithms have different purposes; one is a loss function(MSE Loss), and the other finds class priors of unlabeled data(Bekker Davis, 2020). Since they both showed promising results, we combined the two to form a comprehensive classification algorithm. This novel algorithm relies on the same assumptions as the subclass prior algorithm (see section 2.7.7 for more details). This is important because it allows for more complex assumptions about data. In the real world, datasets are complex, and assumptions like the SCAR assumption (Bekker Davis, 2020) are unable to take into account this complexity. Our algorithm addresses this challenge by finding these biases and sorting the data into clusters. Within each cluster, we assume that the SCAR assumption holds. This is important because many PU algorithms do not explore the relationships between data within a dataset(Jain et al., 2020).

Based on our experimental results, the Cluster-Bias Adjusted Clustering algorithm is able to estimate the propensity score within a promising margin of error (see section **??** for more details). We believe that our algorithm has potential, and merits further investigation.

7.1 Table of Accomplishments

Calvin Kocienda

- Served as team co-leader for meetings, organized team tasks and progress
- Implemented logistic regression classifier in Pytorch to run tests on datasets

- Helped implement two-step EM-NB and two-step NB-SVM classifiers to use in experiment
- Helped research ideas for a novel algorithm
- Implemented the nnPU loss function on a simple model
- Ran TweetBERT on our labeled Twitter dataset
- Served as team leader for meetings, organized team tasks and progress.
- Finalized the TweetBERT implemenation on a PyTorch classifier.

Jesse Abeyta

- Created pipeline to streamline running tests on our datasets
- Implemented automated preprocessing functions
- Implemented automated scoring metrics, with output in a .csv file
- Helped implement one-step NB classifiers to use in experiment
- Served as team co-leader, organized tasks for each group to accomplish
- Continued to update data pipeline
- Worked on developing the PULSE algorithm, before switching to implement the TIcE algorithm
- Designed experiments and aided in implementation of SAR-PU and SAR-PU + EM algorithms
- Conceptualized and designed novel algorithm, designed experiments for it

.

Vinay Nair

- Set up Tableau profile for use in future experiments
- Served as team scribe, recorded meeting minutes
- Implemented an ANN for usage in an earlier version of our experiment

- Helped research ideas for a novel algorithm
- Organized and cleaned our Twitter dataset for future use
- Researched and implemented the TwitterBERT algorithm

Nick Cheng

- Helped implement two-step EM-NB and two-step NB-SVM classifiers to use in experiment
- Served as team scribe, recorded meeting minutes
- Worked on the PULSE algorithm, which was discontinued
- Implemented the subclass prior algorithm (class prior estimation with biased positive and unlabeled examples).
- Created sample data and ran experiments for TiCE and the Subclass Prior Estimation algorithm
- Worked on SAR-PU and SAR-PU + EM algorithms and ran experiments on them
- Worked on novel algorithm and ran experiments on it

Bryan Gass

- Helped with research and understanding for potential novel algorithms
- Implemented one-step SVM classifier to use in experiment
- Worked on the creation of statistical learning algorithms to show baseline metrics to compare and reference in future endeavours.
- Helped with research and understanding for potential novel algorithms
- Aided with the understanding and implementation of PULSE and as well as supporting algorithms.
- Assisted in understanding of TIcE algorithms.
- Continued development on baselines to compare novel algorithms to
- Further worked on data preprocessing strategies for related data
- Edited and aided in novel algorithm understanding and baseline material

7.1.1 B Term

Calvin Kocienda

- Helped research ideas for a novel algorithm
- Implemented the nnPU loss function on a simple model
- Ran TweetBERT on our labeled Twitter dataset

Jesse Abeyta

- Served as team leader, organized tasks for each group to accomplish
- Continued to update data pipeline
- Worked on developing the PULSE algorithm, before switching to implement the TIcE algorithm

Vinay Nair

- Helped research ideas for a novel algorithm
- Organized and cleaned our Twitter dataset for future use
- Researched and implemented the TwitterBERT algorithm

Nick Cheng

- Served as team scribe, recorded meeting minutes
- Worked on the PULSE algorithm, which was discontinued
- Implemented the AIII algorithm (class prior estimation with biased positive and unlabeled examples).
- Created sample data and ran experiments for TIcE and the AIII algorithm

Bryan Gass

- Worked on the creation of statistical learning algorithms to show baseline metrics to compare and reference in future endeavours.
- Helped with research and understanding for potential novel algorithms

- Aided with the understanding and implementation of PULSE and as well as supporting algorithms.
- Assisted in understanding of alphamax and TIcE algorithms.

Team Bryan, Jesse, and Nick

- Researched and developed the PULSE algorithm, alongside several necessary supporting methods.
- Researched and implemented the TiCE algorithm.

Team Calvin and Vinay

- Researched and developed the non-negative PU loss function.
- Acquired and cleaned a dataset from our sister MQP group.
- Used this dataset in our implementation of TweetBERT.

7.1.2 C Term

Calvin Kocienda

- Served as team leader for meetings, organized team tasks and progress.
- Finalized the TweetBERT implemenation on a PyTorch classifier.

Nick Cheng

- Implemented and ran experiments with SAR-PU and SAR-PU + EM algorithms
- Implemented and ran experiments with novel algorithm

Jesse Abeyta

- Designed experiments and aided in implementation of SAR-PU and SAR-PU + EM algorithms
- Conceptualized and designed novel algorithm, designed experiments for it

Bryan Gass

- Continued development on baselines to compare novel algorithms to
- Further worked on data preprocessing strategies for related data
- Edited and aided in novel algorithm understanding and baseline material

Vinay

- Implemented and aided for running experiments with SAR-EM model.
- Analyzed twitter data for TweetBERT using Tableau.

7.1.3 D Term

Vinay

- Worked on identifying the bug in the C-BAC code base
- Conducted experimentation to better understand the issue with the classification part of the C-BAC algorithm
- Worked on experimentation of the C-BAC algorithm involving real world datasets.
- Authored subsections:
  5.1.3 (Banknote Authentication)
  5.1.4 (HTRU Dataset)
  5.3.7 (C-BAC Classifier Accuracy)
  5.3.8 (C-BAC and HTRU Experimentation)
  5.3.9 (C-BAC and Banknote Authentication Experiment)
- Edited (revisions and adding new paragraphs) the following subsections:
  4.2.10 (Novel Algorithm - Cluster-Bias Adjustment Classification)
  5.2.5 (Cluster Bias Adjustment Classification)
  5.3.6 (C-BAC generated data)
  6.6 (C-BAC Experimentation)
  7 (Conclusion)
  7.1 (Fixed the "Table of Accomplishments")
  7.2 (Future Works)

7.2 Future Steps

Our group plans to implement our novel algorithms on the Twitter dataset. As of now, all of our experiments were run on synthetic data, and not real-world data. Testing our algorithms on a real-world dataset will allow us to determine our model's applicability to real-world scenarios. Additionally, the novel algorithm does not perform as well when the propensity scores of the various clusters vastly differ. We plan to investigate this and find a resolution for this issue.

One of the potential future works based on the experimentation that has been conducted using the C-BAC algorithm is to try using different classification algorithms instead of k-means clustering to improve on the C-BAC algorithm in terms of applicability. This would enable the C-BAC algorithm to handle a more diverse set of datasets and tackle more real world issues. Another potential future works is to try out more real world datasets to see where the algorithm can perform optimally and deliver quality results.

Another part of the work that will be carrying on into the future is writing a conference paper publication about the C-BAC algorithm and making the aforementioned changes to ensure the success of the conference paper. Thus far Vinay has worked on a draft of the introduction section of the conference paper and plans to work with the advisors, professor and Jesse to create and submit a full publication and hopefully succeed well at a top ML/AI conference.

References

1 Artificial Neural Network - an overview — ScienceDirect Topics. (2008). Artificial Neural Network. https://www.sciencedirect.com/topics/earth-and-planetary-sciences/artificial-neural-network

2 Backpropogation — Brilliant Math Science Wiki. (2020). Backpropagation. https://brilliant.org/wiki/backpropagation/:%7E:text=Backpropagation%2C

3 Bekker, J., Davis, J. (2020). Learning from positive and unlabeled data: a survey. Machine Learning, 109(4), 719–760. https://doi.org/10.1007/s10994-020-05877-5

4 Bengio, Y., Simard, P., Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157–166. https://doi.org/10.1109/72.279181

5 Blockeel, H. (2017). PU-learning disjunctive concepts in ILP. CEUR Workshop Proceedings, 1–9. http://ceur-ws.org/Vol-2085/blockeelLBP-ILP2017.pdf

6 Classification: Accuracy — Machine Learning Crash Course. (2020). Google Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy

7 Classification: Precision and Recall — Machine Learning Crash Course. (2020). Google Developers. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

8 Classification: ROC Curve and AUC — Machine Learning Crash Course. (2020). Google Developers. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

9 Classification: True vs. False and Positive vs. Negative. (2020). Google Developers. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative

10 Create confusion matrix chart for classification problem - MATLAB confusionchart. (2018). Confusion Chart. https://www.mathworks.com/help/stats/confusionchart.html

11 Depression Statistics. (2019, July 12). Depression and Bipolar Support Alliance. https://www.dbsalliance.org/education/depression/statistics/

12 Estimating the Class Prior in Positive and Unlabeled Data through Decision Tree Induction. (2018). AAAI, 1–7. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/viewFile/16776/16539

13 FastStats. (2020). Depression. https://www.cdc.gov/nchs/fastats/depression.htm

14 Gabriel Pui Cheong Fung, Yu, J. X., Hongjun Lu, Yu, P. S. (2006). Text classification without negative examples revisit. IEEE Transactions on Knowledge and Data Engineering, 18(1), 6–20. https://doi.org/10.1109/tkde.2006.16

15 Giang Nguyn. (2013, March 9). 3 - 4 - The backpropagation algorithm [12 min]. YouTube. https://www.youtube.com/watch?v=46Jzu-xWIBkamp;feature=emb_logo

16 Hastie, T., Tibshirani, R., Friedman, J. (2021). The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics) (2nd ed.). Springer.

17 Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. (2003, March). Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.

18 Institute of Medicine. (2002). Care Without Coverage: Too Little, Too Late (1st ed.). National Academies Press.

19 ”Introduction To PyTorch — PyTorch Tutorials 1.6.0 Documentation”. 2020. PyTorch.Org. https://pyTorch.org/tutorials/beginner/nlp/pyTorch_tutorial.html.

20 Jain, S., Delano, J., Sharma, H., Radivojac, P. (2020). Class Prior Estimation with Biased Positives and Unlabeled Examples. Proceedings of the AAAI Conference on Artificial Intelligence, 34(04), 4255–4263. https://doi.org/10.1609/aaai.v34i04.5848

21 Jessa Bekker, J., Robberechts, P., Davis, J. (2019). Beyond the Selected Completely At Random Assumption for Learning from Positive and Unlabeled Data. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2019, 1–14. https://arxiv.org/pdf/1809.03207.pdf

22 Kaboutari, A., Bagherzadeh, J., Kheradmand, F. (2014). An Evaluation of Two-Step Techniques for Positive-Unlabeled Learning in Text Classification. International Journal of Computer Applications Technology and Research, 3(9), 592–594. https://doi.org/10.7753/ijcatr0309.1012

23 Kiryo, R., Niu, G., du Plessis, M., Sugiyama, M. (2017). Positive-Unlabeled Learning with Non-Negative Risk Estimator. NIPS, 1674–1684. https://www.researchgate.net/publication/314182104$_Positive - Unlabeled_Learning_with_Non - Negative_Risk_Estimator$

24 Mallick, S. (2018, July 12). Support Vector Machines (SVM) — Learn OpenCV. Learn OpenCV — OpenCV, PyTorch, Keras, Tensorflow Examples and Tutorials. https://learnopencv.com/support-vector-machines-svm/

25 Matplotlib: Python plotting — Matplotlib 3.3.4 documentation. (2012). Matplotlib. https://matplotlib.org/

26 Nayak, P. (2019, October 25). Understanding searches better than ever before. Google. https://blog.google/products/search/search-language-understanding-bert/

27 NumPy. (2020). Numpy. https://numpy.org/

28 pandas - Python Data Analysis Library. (2010). Pandas. https://pandas.pydata.org/

29 Plessis, M. C., Niu, G., Sugiyama, M. (2014). Analysis of Learning from Positive and Unlabeled Data. Conference on Neural Information Processing Systems, 1–8. https://papers.nips.cc/paper/2014/file/35051070e572e47d2c26c241ab88307f-Paper.pdf

30 Ratner, A., Sa, C. D., Wu, S., Selsam, D., Ré, C. (2016). Data Programming: Creating Large Training Sets, Quickly. NCBI, 1–40. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5985238/

31 Ronaghan, S. (2018, July 27). Deep Learning: Overview of Neurons and Activation Functions. Medium. https://srnghn.medium.com/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4

32 Russell, S., Norvig, P. (2020). Artificial Intelligence: A Modern Approach (Pearson Series in Artifical Intelligence) (4th ed.). Pearson.

33 scikit-learn: machine learning in Python — scikit-learn 0.24.1 documentation. (2019). Scikit-Learn. https://scikit-learn.org/stable/

34 Tableau: Business Intelligence and Analytics Software. (2003). Tableau. https://www.tableau.com/

35 Transformer: A Novel Neural Network Architecture for Language Understanding. (2017, August 31). Google AI Blog. https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

36 UCI Machine Learning Repository: Online Shoppers Purchasing Intention Dataset Data Set. (2018). Online Shoppers Purchasing Intention Dataset Data Set. https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset

37 Wood, T. (2020, August 7). F-Score. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/f-score

38 Yiu, T. (2019, August 4). Understanding Neural Networks - Towards Data Science. Medium. https://towardsdatascience.com/understanding-neural-networks-19020b758230

39 "Machine Learning Tools And OS - Scaleway". 2020. Scaleway. https://www.scaleway.com/en/machine-learning-images/: :text=Pytorch%20is%20an%20open%20source,ecosystem%20of %20tools%20and%20libraries.

40 Mathwords.com. 2020. Mathwords: Area Under A Curve. [online] Available at: ¡https://www.mathwords.com/a/area_under_a_curve.htm¿ [Accessed 26 October 2020].

41 Mathwords.com. 2020. Mathwords: Harmonic Mean. [online] Available at: ¡https://www.mathwords.com/h/harmonic_mean.htm¿ [Accessed 26 October 2020].

42 McGonagle, J., Shaikouski, G., Williams, C., et al. 2020. Backpropogation. [online] Available at: ¡https://brilliant.org/wiki/backpropagation/: :text=Backpropagation%2C %20short%20for%20%22backward%20propagation,to%20the%20neural %20network's%20weights¿ [Accessed 26 October 2020].

43 Ncbi.nlm.nih.gov. 2020. Effects Of Health Insurance On Health. [online] Available at: ¡https://www.ncbi.nlm.nih.gov/books/NBK220636/¿ [Accessed 26 October 2020].

44 Nguyn, G. (2013, March 8). 3 - 4 - The backpropagation algorithm [12 min] [Video file]. Retrieved from https://www.youtube.com/watch?v=46Jzu-xWIBkamp;feature=emb_logo

8 Appendix

8.1 Appendix A

num_clusters	standard_dev	alpha	alpha_average	est_alpha
2	0.25	0.01	0.01024	0.11128
2	0.25	0.05	0.04900	0.11173
2	0.25	0.10	0.10024	0.14150
2	0.25	0.20	0.20313	0.25238
2	0.25	0.30	0.30164	0.31918
2	0.25	0.40	0.40541	0.45931
2	0.25	0.50	0.50034	0.50591
2	0.25	0.60	0.59511	0.60863
2	0.25	0.70	0.70265	0.71175
2	0.25	0.80	0.79445	0.79893
2	0.25	0.85	0.83961	0.83941
2	0.25	0.90	0.90091	0.89905
2	0.25	0.95	0.92965	0.92307
2	0.25	0.99	0.99128	0.98548
2	0.50	0.01	0.01016	0.05955
2	0.50	0.05	0.05091	0.09082
2	0.50	0.10	0.10020	0.13000
2	0.50	0.20	0.20197	0.20441
2	0.50	0.30	0.29879	0.29988
2	0.50	0.40	0.40423	0.41817
2	0.50	0.50	0.50354	0.51452
2	0.50	0.60	0.60329	0.60673
2	0.50	0.70	0.70444	0.70859
2	0.50	0.80	0.81203	0.81014
2	0.50	0.85	0.84019	0.83912
2	0.50	0.90	0.89682	0.89436
2	0.50	0.95	0.95812	0.93648
2	0.50	0.99	0.98894	0.98435
2	0.75	0.01	0.01006	0.04706
2	0.75	0.05	0.05000	0.06211
2	0.75	0.10	0.09835	0.09459
2	0.75	0.20	0.20544	0.21623

2	0.75	0.30	0.29775	0.29565
2	0.75	0.40	0.40178	0.39877
2	0.75	0.50	0.49846	0.50313
2	0.75	0.60	0.59701	0.59367
2	0.75	0.70	0.70646	0.70197
2	0.75	0.80	0.79789	0.79201
2	0.75	0.85	0.85581	0.85036
2	0.75	0.90	0.88441	0.87779
2	0.75	0.95	0.95144	0.93739
2	0.75	0.99	0.99271	0.98634
2	1.00	0.01	0.01003	0.04163
2	1.00	0.05	0.04994	0.03766
2	1.00	0.10	0.09923	0.10640
2	1.00	0.20	0.20109	0.20243
2	1.00	0.30	0.30047	0.29465
2	1.00	0.40	0.40231	0.40170
2	1.00	0.50	0.49982	0.49500
2	1.00	0.60	0.60769	0.60202
2	1.00	0.70	0.70139	0.70049
2	1.00	0.80	0.79566	0.78725
2	1.00	0.85	0.85795	0.85044
2	1.00	0.90	0.90491	0.90266
2	1.00	0.95	0.96242	0.93597
2	1.00	0.99	0.98997	0.98335
2	2.00	0.01	0.00995	0.01575
2	2.00	0.05	0.04944	0.05292
2	2.00	0.10	0.10050	0.10231
2	2.00	0.20	0.19756	0.17288
2	2.00	0.30	0.30223	0.29853
2	2.00	0.40	0.40804	0.39334
2	2.00	0.50	0.49657	0.48715
2	2.00	0.60	0.59139	0.57913
2	2.00	0.70	0.69967	0.68920
2	2.00	0.80	0.78400	0.77640
2	2.00	0.85	0.84224	0.83638
2	2.00	0.90	0.91253	0.90000
2	2.00	0.95	0.95189	0.94028

2	2.00	0.99	0.98841	0.98312
2	4.00	0.01	0.00999	0.00021
2	4.00	0.05	0.05105	0.02019
2	4.00	0.10	0.09949	0.07162
2	4.00	0.20	0.19867	0.16983
2	4.00	0.30	0.29817	0.26935
2	4.00	0.40	0.39749	0.37271
2	4.00	0.50	0.50582	0.48366
2	4.00	0.60	0.60225	0.58260
2	4.00	0.70	0.68323	0.66799
2	4.00	0.80	0.80353	0.79046
2	4.00	0.85	0.83954	0.82754
2	4.00	0.90	0.91130	0.89502
2	4.00	0.95	0.93207	0.91989
2	4.00	0.99	0.98915	0.98234
2	8.00	0.01	0.00991	0.00192
2	8.00	0.05	0.04943	0.01432
2	8.00	0.10	0.10102	0.07501
2	8.00	0.20	0.20234	0.17073
2	8.00	0.30	0.29888	0.26961
2	8.00	0.40	0.40030	0.37295
2	8.00	0.50	0.49065	0.46659
2	8.00	0.60	0.59912	0.57870
2	8.00	0.70	0.69070	0.67086
2	8.00	0.80	0.79731	0.78137
2	8.00	0.85	0.84715	0.83355
2	8.00	0.90	0.90244	0.88471
2	8.00	0.95	0.93482	0.92128
2	8.00	0.99	0.98813	0.97970
2	10.00	0.01	0.01000	0.00037
2	10.00	0.05	0.05042	0.01376
2	10.00	0.10	0.09764	0.06923
2	10.00	0.20	0.20182	0.16919
2	10.00	0.30	0.29890	0.27059
2	10.00	0.40	0.40218	0.37515
2	10.00	0.50	0.50633	0.48195
2	10.00	0.60	0.58846	0.56608

2	10.00	0.70	0.68824	0.66894
2	10.00	0.80	0.80389	0.78689
2	10.00	0.85	0.85693	0.84335
2	10.00	0.90	0.90490	0.89351
2	10.00	0.95	0.95315	0.93330
2	10.00	0.99	0.98854	0.97789
3	0.25	0.01	0.00995	0.08142
3	0.25	0.05	0.05014	0.14961
3	0.25	0.10	0.10092	0.19595
3	0.25	0.20	0.19975	0.27367
3	0.25	0.30	0.29963	0.36583
3	0.25	0.40	0.39295	0.43177
3	0.25	0.50	0.49751	0.53900
3	0.25	0.60	0.59041	0.61624
3	0.25	0.70	0.69782	0.71140
3	0.25	0.80	0.80219	0.81083
3	0.25	0.85	0.84643	0.85077
3	0.25	0.90	0.90623	0.90554
3	0.25	0.95	0.95296	0.94477
3	0.25	0.99	0.99115	0.98416
3	0.50	0.01	0.01000	0.10255
3	0.50	0.05	0.04990	0.08325
3	0.50	0.10	0.09932	0.14228
3	0.50	0.20	0.19998	0.22483
3	0.50	0.30	0.29964	0.32445
3	0.50	0.40	0.40786	0.43630
3	0.50	0.50	0.50241	0.51358
3	0.50	0.60	0.59445	0.60369
3	0.50	0.70	0.71056	0.70936
3	0.50	0.80	0.79317	0.79376
3	0.50	0.85	0.83796	0.83692
3	0.50	0.90	0.90554	0.90193
3	0.50	0.95	0.93249	0.92669
3	0.50	0.99	0.98881	0.98348
3	0.75	0.01	0.00994	0.06027
3	0.75	0.05	0.04933	0.06664
3	0.75	0.10	0.10069	0.12096

3	0.75	0.20	0.19882	0.19282
3	0.75	0.30	0.30546	0.32088
3	0.75	0.40	0.39799	0.40177
3	0.75	0.50	0.50164	0.51347
3	0.75	0.60	0.60113	0.60191
3	0.75	0.70	0.69887	0.69839
3	0.75	0.80	0.79157	0.78808
3	0.75	0.85	0.84312	0.83788
3	0.75	0.90	0.90717	0.89733
3	0.75	0.95	0.93888	0.92740
3	0.75	0.99	0.99095	0.98601
3	1.00	0.01	0.01006	0.03718
3	1.00	0.05	0.05032	0.06694
3	1.00	0.10	0.09821	0.09923
3	1.00	0.20	0.20051	0.19821
3	1.00	0.30	0.29353	0.28801
3	1.00	0.40	0.40338	0.39416
3	1.00	0.50	0.49901	0.49400
3	1.00	0.60	0.60095	0.61051
3	1.00	0.70	0.70586	0.70677
3	1.00	0.80	0.80843	0.80255
3	1.00	0.85	0.86328	0.85766
3	1.00	0.90	0.90663	0.90360
3	1.00	0.95	0.95009	0.93600
3	1.00	0.99	0.98831	0.98325
3	2.00	0.01	0.00998	0.02418
3	2.00	0.05	0.04984	0.03576
3	2.00	0.10	0.10154	0.12094
3	2.00	0.20	0.20115	0.18984
3	2.00	0.30	0.30240	0.29152
3	2.00	0.40	0.39882	0.38837
3	2.00	0.50	0.50012	0.49352
3	2.00	0.60	0.59318	0.59116
3	2.00	0.70	0.70325	0.69272
3	2.00	0.80	0.79936	0.79213
3	2.00	0.85	0.85176	0.84532
3	2.00	0.90	0.89641	0.89008

3	2.00	0.95	0.95016	0.93847
3	2.00	0.99	0.98667	0.98190
3	4.00	0.01	0.00999	0.01027
3	4.00	0.05	0.04959	0.02454
3	4.00	0.10	0.09973	0.07232
3	4.00	0.20	0.19955	0.17419
3	4.00	0.30	0.29959	0.27048
3	4.00	0.40	0.40064	0.38239
3	4.00	0.50	0.49879	0.47681
3	4.00	0.60	0.60395	0.58522
3	4.00	0.70	0.69540	0.68010
3	4.00	0.80	0.80566	0.79168
3	4.00	0.85	0.85149	0.83955
3	4.00	0.90	0.90428	0.89552
3	4.00	0.95	0.92590	0.91579
3	4.00	0.99	0.99116	0.98320
3	8.00	0.01	0.00996	0.00598
3	8.00	0.05	0.05071	0.01591
3	8.00	0.10	0.09858	0.06815
3	8.00	0.20	0.19803	0.16551
3	8.00	0.30	0.29506	0.26398
3	8.00	0.40	0.39958	0.37212
3	8.00	0.50	0.49827	0.47401
3	8.00	0.60	0.60269	0.57955
3	8.00	0.70	0.70062	0.68026
3	8.00	0.80	0.79957	0.78430
3	8.00	0.85	0.84521	0.83037
3	8.00	0.90	0.89246	0.87453
3	8.00	0.95	0.95134	0.92842
3	8.00	0.99	0.99061	0.97672
3	10.00	0.01	0.01001	0.00135
3	10.00	0.05	0.05051	0.01887
3	10.00	0.10	0.09999	0.06581
3	10.00	0.20	0.19921	0.16584
3	10.00	0.30	0.29989	0.26892
3	10.00	0.40	0.40029	0.37352
3	10.00	0.50	0.51124	0.48633

3	10.00	0.60	0.60251	0.57910
3	10.00	0.70	0.70224	0.68375
3	10.00	0.80	0.80723	0.79014
3	10.00	0.85	0.85742	0.84307
3	10.00	0.90	0.90943	0.89458
3	10.00	0.95	0.94314	0.91606
3	10.00	0.99	0.98953	0.97841
4	0.25	0.01	0.00993	0.11905
4	0.25	0.05	0.05041	0.15162
4	0.25	0.10	0.09954	0.17842
4	0.25	0.20	0.19928	0.28038
4	0.25	0.30	0.29849	0.35632
4	0.25	0.40	0.40067	0.45838
4	0.25	0.50	0.50341	0.54371
4	0.25	0.60	0.59582	0.62190
4	0.25	0.70	0.69626	0.70659
4	0.25	0.80	0.80939	0.81742
4	0.25	0.85	0.84670	0.85068
4	0.25	0.90	0.90625	0.90441
4	0.25	0.95	0.96367	0.95102
4	0.25	0.99	0.99021	0.98513
4	0.50	0.01	0.00993	0.08727
4	0.50	0.05	0.04956	0.09890
4	0.50	0.10	0.09983	0.15602
4	0.50	0.20	0.19986	0.24856
4	0.50	0.30	0.30139	0.33185
4	0.50	0.40	0.39842	0.42993
4	0.50	0.50	0.50837	0.53073
4	0.50	0.60	0.60764	0.61058
4	0.50	0.70	0.69413	0.69579
4	0.50	0.80	0.79263	0.79175
4	0.50	0.85	0.85998	0.85880
4	0.50	0.90	0.89631	0.89481
4	0.50	0.95	0.95003	0.93808
4	0.50	0.99	0.99144	0.98336
4	0.75	0.01	0.01015	0.05923
4	0.75	0.05	0.04914	0.07284

4	0.75	0.10	0.10103	0.11362
4	0.75	0.20	0.19852	0.22530
4	0.75	0.30	0.29587	0.31074
4	0.75	0.40	0.40155	0.41093
4	0.75	0.50	0.50113	0.51759
4	0.75	0.60	0.60487	0.60047
4	0.75	0.70	0.69634	0.69331
4	0.75	0.80	0.79283	0.78904
4	0.75	0.85	0.84310	0.84214
4	0.75	0.90	0.89209	0.88698
4	0.75	0.95	0.95358	0.93636
4	0.75	0.99	0.99060	0.98443
4	1.00	0.01	0.01006	0.05860
4	1.00	0.05	0.04979	0.05836
4	1.00	0.10	0.09974	0.11454
4	1.00	0.20	0.20192	0.20385
4	1.00	0.30	0.29978	0.31214
4	1.00	0.40	0.40133	0.41178
4	1.00	0.50	0.49775	0.49634
4	1.00	0.60	0.60080	0.60035
4	1.00	0.70	0.70078	0.69913
4	1.00	0.80	0.79968	0.79422
4	1.00	0.85	0.85085	0.84704
4	1.00	0.90	0.89410	0.88952
4	1.00	0.95	0.95315	0.93224
4	1.00	0.99	0.98901	0.98341
4	2.00	0.01	0.00991	0.02244
4	2.00	0.05	0.05018	0.04233
4	2.00	0.10	0.09950	0.08364
4	2.00	0.20	0.20234	0.19114
4	2.00	0.30	0.29989	0.28475
4	2.00	0.40	0.40472	0.39749
4	2.00	0.50	0.50448	0.49603
4	2.00	0.60	0.59480	0.58503
4	2.00	0.70	0.69882	0.69274
4	2.00	0.80	0.80734	0.79805
4	2.00	0.85	0.83751	0.83192

4	2.00	0.90	0.89357	0.88759
4	2.00	0.95	0.95682	0.94033
4	2.00	0.99	0.99052	0.98407
4	4.00	0.01	0.01008	0.01598
4	4.00	0.05	0.05025	0.02332
4	4.00	0.10	0.09939	0.07277
4	4.00	0.20	0.20121	0.17470
4	4.00	0.30	0.29706	0.26901
4	4.00	0.40	0.40468	0.38036
4	4.00	0.50	0.50539	0.48162
4	4.00	0.60	0.59721	0.57632
4	4.00	0.70	0.70285	0.68656
4	4.00	0.80	0.79700	0.78441
4	4.00	0.85	0.84880	0.83642
4	4.00	0.90	0.88908	0.87694
4	4.00	0.95	0.95274	0.93010
4	4.00	0.99	0.99044	0.98173
4	8.00	0.01	0.00996	0.00288
4	8.00	0.05	0.05000	0.02190
4	8.00	0.10	0.10068	0.06728
4	8.00	0.20	0.20256	0.16768
4	8.00	0.30	0.29794	0.26841
4	8.00	0.40	0.40204	0.37389
4	8.00	0.50	0.50156	0.47590
4	8.00	0.60	0.59494	0.57184
4	8.00	0.70	0.69766	0.67796
4	8.00	0.80	0.80247	0.78616
4	8.00	0.85	0.84786	0.83148
4	8.00	0.90	0.90059	0.88627
4	8.00	0.95	0.94603	0.92123
4	8.00	0.99	0.99091	0.97767
4	10.00	0.01	0.01000	0.00012
4	10.00	0.05	0.05028	0.01864
4	10.00	0.10	0.09994	0.06472
4	10.00	0.20	0.20151	0.16827
4	10.00	0.30	0.30318	0.27287
4	10.00	0.40	0.39740	0.36916

4	10.00	0.50	0.49969	0.47314
4	10.00	0.60	0.59338	0.56961
4	10.00	0.70	0.70185	0.68034
4	10.00	0.80	0.80594	0.78688
4	10.00	0.85	0.85003	0.83378
4	10.00	0.90	0.89234	0.87761
4	10.00	0.95	0.96378	0.93252
4	10.00	0.99	0.98969	0.97662

Table 8: Vary negative cluster standard deviation. The MSE metric was calculated after running each setup with 10 iterations.

8.2 Appendix B

num_clusters	mean_pos_size	alpha	alpha_average	est_alpha	MSE
2	2000	0.3	0.30602	0.42167	0.01847
2	2000	0.4	0.40023	0.48002	0.00811
2	2000	0.5	0.48873	0.54611	0.00565
2	2000	0.6	0.59010	0.61480	0.00115
2	2000	0.7	0.69188	0.70976	0.00062
2	4000	0.3	0.30380	0.37532	0.00572
2	4000	0.4	0.39968	0.42906	0.00154
2	4000	0.5	0.50229	0.54464	0.00271
2	4000	0.6	0.59098	0.65519	0.00496
2	4000	0.7	0.70808	0.72894	0.00080
2	6000	0.3	0.29712	0.35572	0.00569
2	6000	0.4	0.39305	0.45369	0.00626
2	6000	0.5	0.51011	0.53607	0.00086
2	6000	0.6	0.60172	0.63179	0.00134
2	6000	0.7	0.70623	0.72582	0.00052
2	8000	0.3	0.30052	0.37215	0.00709
2	8000	0.4	0.39363	0.42070	0.00107
2	8000	0.5	0.50178	0.52617	0.00105
2	8000	0.6	0.60142	0.63363	0.00194
2	8000	0.7	0.69819	0.72153	0.00067
3	2000	0.3	0.29614	0.43301	0.02193
3	2000	0.4	0.39991	0.48295	0.00763
3	2000	0.5	0.50181	0.57509	0.00641
3	2000	0.6	0.59614	0.64164	0.00358
3	2000	0.7	0.70478	0.73604	0.00132
3	4000	0.3	0.29576	0.37044	0.00667
3	4000	0.4	0.39898	0.49499	0.01018
3	4000	0.5	0.50521	0.55672	0.00329
3	4000	0.6	0.60529	0.64247	0.00164
3	4000	0.7	0.69712	0.72960	0.00133
3	6000	0.3	0.29807	0.36706	0.00619
3	6000	0.4	0.40082	0.45278	0.00339
3	6000	0.5	0.49963	0.57359	0.00697
3	6000	0.6	0.60406	0.63795	0.00152

3	6000	0.7	0.70915	0.72549	0.00036
3	8000	0.3	0.29603	0.36591	0.00585
3	8000	0.4	0.39750	0.45963	0.00434
3	8000	0.5	0.50345	0.54318	0.00216
3	8000	0.6	0.59427	0.63207	0.00200
3	8000	0.7	0.71679	0.73421	0.00034
4	2000	0.3	0.29899	0.43508	0.01987
4	2000	0.4	0.39741	0.50344	0.01162
4	2000	0.5	0.50405	0.57700	0.00688
4	2000	0.6	0.60594	0.66598	0.00390
4	2000	0.7	0.69007	0.72953	0.00200
4	4000	0.3	0.30113	0.40784	0.01247
4	4000	0.4	0.40068	0.47783	0.00628
4	4000	0.5	0.49592	0.56537	0.00597
4	4000	0.6	0.60863	0.65368	0.00222
4	4000	0.7	0.70485	0.73163	0.00090
4	6000	0.3	0.30184	0.39145	0.00870
4	6000	0.4	0.40245	0.47871	0.00697
4	6000	0.5	0.49803	0.56412	0.00494
4	6000	0.6	0.59836	0.63921	0.00192
4	6000	0.7	0.68996	0.71655	0.00076
4	8000	0.3	0.29708	0.36369	0.00502
4	8000	0.4	0.39905	0.47198	0.00587
4	8000	0.5	0.50479	0.54552	0.00200
4	8000	0.6	0.60685	0.65384	0.00252
4	8000	0.7	0.70103	0.72715	0.00100

Table 9: Results when varying the mean positive cluster size. The MSE metric was calculated after running each setup with 10 iterations.

8.3 Appendix C

num_clusters	pos_standard_dev	alpha	alpha_average	est_alpha	MSE
2	0.15	0.3	0.29911	0.33988	0.00312
2	0.15	0.4	0.39860	0.41023	0.00052
2	0.15	0.5	0.49995	0.52372	0.00106
2	0.15	0.6	0.62470	0.64165	0.00082
2	0.15	0.7	0.71724	0.72678	0.00032
2	0.20	0.3	0.29411	0.34091	0.00362
2	0.20	0.4	0.40212	0.48153	0.00791
2	0.20	0.5	0.50307	0.56074	0.00459
2	0.20	0.6	0.60148	0.62102	0.00075
2	0.20	0.7	0.69314	0.72054	0.00148
2	0.50	0.3	0.30413	0.47565	0.03204
2	0.50	0.4	0.40362	0.55431	0.02361
2	0.50	0.5	0.50929	0.60385	0.01008
2	0.50	0.6	0.59920	0.68458	0.00854
2	0.50	0.7	0.69559	0.76536	0.00542
2	1.00	0.3	0.30626	0.49090	0.03495
2	1.00	0.4	0.38990	0.55346	0.02753
2	1.00	0.5	0.49926	0.63760	0.02115
2	1.00	0.6	0.59895	0.71233	0.01365
2	1.00	0.7	0.70424	0.77629	0.00603
3	0.15	0.3	0.30613	0.34625	0.00215
3	0.15	0.4	0.39730	0.43906	0.00202
3	0.15	0.5	0.50078	0.52899	0.00184
3	0.15	0.6	0.60582	0.63253	0.00097
3	0.15	0.7	0.69862	0.70982	0.00026
3	0.20	0.3	0.29967	0.39710	0.01158
3	0.20	0.4	0.39379	0.45193	0.00384
3	0.20	0.5	0.49675	0.54638	0.00305
3	0.20	0.6	0.58386	0.61289	0.00128
3	0.20	0.7	0.69733	0.72342	0.00086
3	0.50	0.3	0.29403	0.48825	0.03795
3	0.50	0.4	0.40093	0.54727	0.02279
3	0.50	0.5	0.49727	0.61330	0.01410
3	0.50	0.6	0.60508	0.68009	0.00654

3	0.50	0.7	0.70973	0.77368	0.00478
3	1.00	0.3	0.30351	0.50298	0.04191
3	1.00	0.4	0.39559	0.55942	0.02821
3	1.00	0.5	0.50089	0.64332	0.02127
3	1.00	0.6	0.59049	0.70232	0.01314
3	1.00	0.7	0.69188	0.77367	0.00694
4	0.15	0.3	0.29681	0.35625	0.00399
4	0.15	0.4	0.39592	0.43652	0.00229
4	0.15	0.5	0.49546	0.53052	0.00177
4	0.15	0.6	0.59968	0.62201	0.00055
4	0.15	0.7	0.68873	0.70611	0.00046
4	0.20	0.3	0.30052	0.39287	0.00976
4	0.20	0.4	0.40301	0.47179	0.00515
4	0.20	0.5	0.50271	0.55106	0.00300
4	0.20	0.6	0.61052	0.65369	0.00218
4	0.20	0.7	0.70118	0.73161	0.00112
4	0.50	0.3	0.30185	0.45721	0.02517
4	0.50	0.4	0.39975	0.53038	0.01741
4	0.50	0.5	0.49905	0.63099	0.01807
4	0.50	0.6	0.60728	0.69302	0.00822
4	0.50	0.7	0.70028	0.76138	0.00391
4	1.00	0.3	0.30079	0.48719	0.03530
4	1.00	0.4	0.40230	0.55985	0.02559
4	1.00	0.5	0.49531	0.61931	0.01583
4	1.00	0.6	0.59911	0.71447	0.01396
4	1.00	0.7	0.70124	0.77410	0.00559

Table 10: Results after varying the positive cluster standard deviation. The MSE metric was calculated after running each setup with 10 iterations.

8.4 Appendix D

standard_dev	target_alpha	est_alpha	MSE
0.25	0.01	0.02367	0.00114
0.25	0.05	0.08419	0.00443
0.25	0.10	0.10779	0.00292
0.25	0.20	0.20920	0.00069
0.25	0.30	0.31303	0.00221
0.25	0.40	0.40760	0.00197
0.25	0.50	0.54013	0.00668
0.25	0.60	0.60044	0.00062
0.25	0.70	0.70449	0.00021
0.25	0.80	0.80134	0.00032
0.25	0.85	0.84755	0.00013
0.25	0.90	0.89414	0.00006
0.25	0.95	0.94492	0.00007
0.25	0.99	0.98296	0.00005
0.50	0.01	0.02857	0.00155
0.50	0.05	0.08491	0.00834
0.50	0.10	0.07040	0.00151
0.50	0.20	0.22692	0.00959
0.50	0.30	0.28795	0.00067
0.50	0.40	0.40646	0.00166
0.50	0.50	0.49742	0.00068
0.50	0.60	0.59898	0.00078
0.50	0.70	0.69779	0.00023
0.50	0.80	0.79027	0.00017
0.50	0.85	0.84597	0.00009
0.50	0.90	0.89402	0.00007
0.50	0.95	0.94770	0.00006
0.50	0.99	0.98426	0.00003
0.75	0.01	0.06373	0.01229
0.75	0.05	0.03882	0.00194
0.75	0.10	0.11533	0.00263
0.75	0.20	0.20403	0.00115
0.75	0.30	0.29647	0.00100
0.75	0.40	0.37793	0.00137

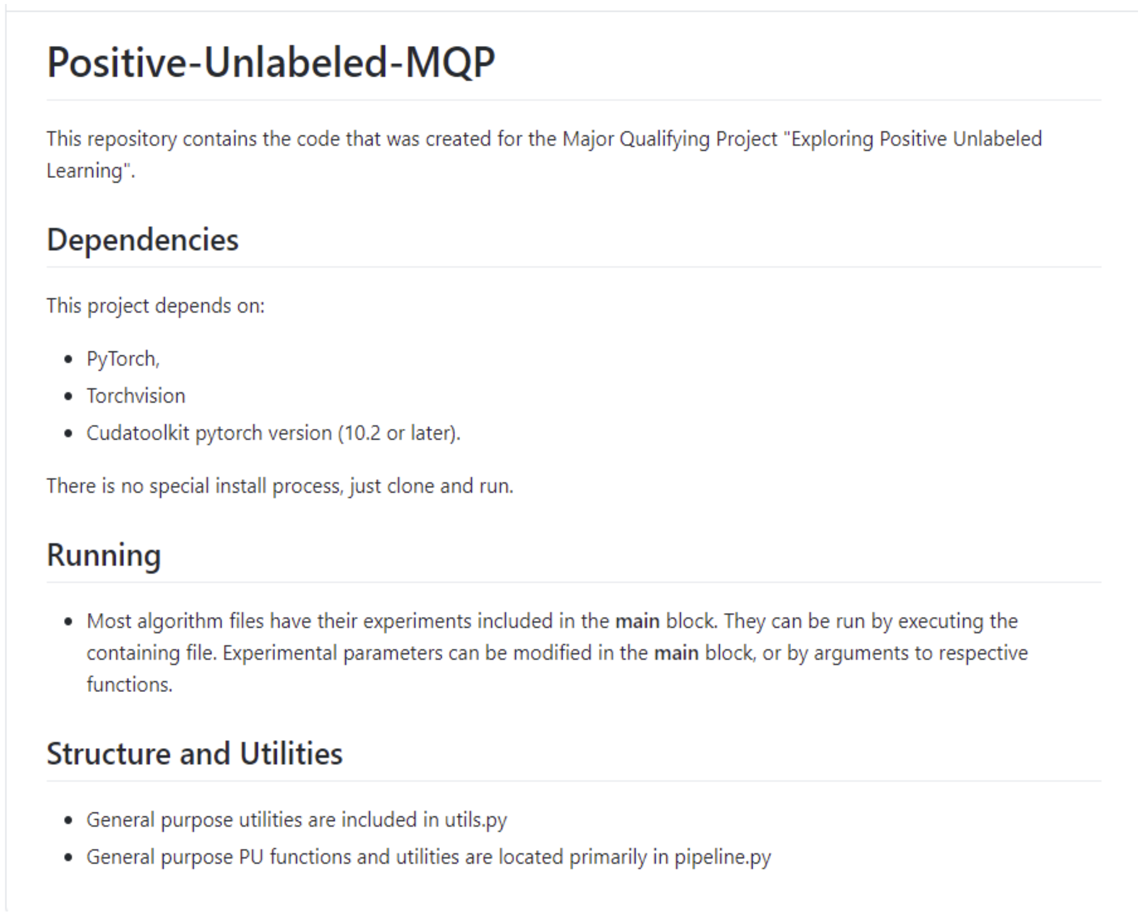
0.75	0.50	0.49823	0.00052
0.75	0.60	0.60017	0.00059
0.75	0.70	0.69745	0.00026
0.75	0.80	0.79256	0.00011
0.75	0.85	0.84348	0.00007
0.75	0.90	0.89662	0.00004
0.75	0.95	0.94652	0.00003
0.75	0.99	0.98379	0.00004
1.00	0.01	0.02477	0.00157
1.00	0.05	0.05341	0.00255
1.00	0.10	0.07953	0.00125
1.00	0.20	0.21616	0.00371
1.00	0.30	0.31414	0.00227
1.00	0.40	0.40307	0.00074
1.00	0.50	0.50459	0.00113
1.00	0.60	0.58903	0.00028
1.00	0.70	0.68827	0.00031
1.00	0.80	0.79999	0.00015
1.00	0.85	0.84073	0.00012
1.00	0.90	0.88923	0.00016
1.00	0.95	0.94329	0.00006
1.00	0.99	0.98327	0.00005
2.00	0.01	0.03920	0.00474
2.00	0.05	0.01107	0.00171
2.00	0.10	0.07276	0.00359
2.00	0.20	0.15457	0.00297
2.00	0.30	0.27445	0.00196
2.00	0.40	0.37086	0.00337
2.00	0.50	0.45452	0.00236
2.00	0.60	0.54660	0.00306
2.00	0.70	0.64254	0.00332
2.00	0.80	0.75397	0.00270
2.00	0.85	0.80057	0.00256
2.00	0.90	0.85365	0.00226
2.00	0.95	0.89990	0.00254
2.00	0.99	0.94135	0.00241
4.00	0.01	0.05570	0.01011

4.00	0.05	0.01094	0.00158
4.00	0.10	0.03865	0.00464
4.00	0.20	0.12197	0.00633
4.00	0.30	0.20601	0.00904
4.00	0.40	0.29161	0.01208
4.00	0.50	0.38574	0.01330
4.00	0.60	0.46602	0.01814
4.00	0.70	0.55164	0.02208
4.00	0.80	0.65223	0.02204
4.00	0.85	0.69332	0.02462
4.00	0.90	0.73504	0.02732
4.00	0.95	0.78706	0.02672
4.00	0.99	0.81644	0.03021
8.00	0.01	0.00034	0.00009
8.00	0.05	0.00514	0.00208
8.00	0.10	0.03938	0.00387
8.00	0.20	0.12594	0.00564
8.00	0.30	0.21440	0.00751
8.00	0.40	0.31229	0.00787
8.00	0.50	0.40387	0.00936
8.00	0.60	0.49046	0.01218
8.00	0.70	0.58185	0.01409
8.00	0.80	0.66561	0.01821
8.00	0.85	0.71853	0.01745
8.00	0.90	0.76737	0.01766
8.00	0.95	0.81612	0.01802
8.00	0.99	0.84700	0.02063
10.00	0.01	0.00758	0.00052
10.00	0.05	0.00100	0.00240
10.00	0.10	0.04217	0.00355
10.00	0.20	0.13313	0.00461
10.00	0.30	0.22987	0.00519
10.00	0.40	0.30844	0.00848
10.00	0.50	0.40707	0.00876
10.00	0.60	0.50078	0.01007
10.00	0.70	0.59415	0.01138
10.00	0.80	0.68660	0.01306

10.00	0.85	0.72438	0.01598
10.00	0.90	0.76745	0.01790
10.00	0.95	0.82347	0.01609
10.00	0.99	0.85196	0.01933

Table 11: TIcE results. The MSE metric was calculated after running each setup with 10 iterations.

8.5 Appendix E



Positive-Unlabeled-MQP

This repository contains the code that was created for the Major Qualifying Project "Exploring Positive Unlabeled Learning".

Dependencies

This project depends on:

- PyTorch,
- Torchvision
- Cudatoolkit pytorch version (10.2 or later).

There is no special install process, just clone and run.

Running

- Most algorithm files have their experiments included in the **main** block. They can be run by executing the containing file. Experimental parameters can be modified in the **main** block, or by arguments to respective functions.

Structure and Utilities

- General purpose utilities are included in utils.py
- General purpose PU functions and utilities are located primarily in pipeline.py

9 Authorship Table

Section	Author	Editor
Chapter 1: Abstract	Vinay Nair	Jesse Abeyta
1.1: Executive Summary	Vinay Nair	Jesse Abeyta
Chapter 2: Introduction		
2.1: Overview of PU Learning	Calvin Kocienda	Bryan Gass/ Jesse Abeyta
2.1.1: What is PU Learning	Calvin Kocienda	Bryan Gass/ Jesse Abeyta
2.1.2: State of the Art Approaches	Calvin Kocienda	Bryan Gass/ Jesse Abeyta
2.2: Our Approach	Calvin Kocienda	Bryan Gass/ Jesse Abeyta
Chapter 3: Background		
3.1: Machine Learning Techniques		
3.1.1: Dataset Cleaning	Calvin Kocienda	Vinay Nair
3.1.2: Training and Testing		
3.1.3: Loss Function	Nicholas Cheng	Bryan Gass
3.1.4: Learning Rate	Nicholas Cheng	Bryan Gass
3.1.5: Gradient Descent	Nicholas Cheng	Bryan Gass
3.2: Performance Evaluation		
3.2.1: Accuracy	Nicholas Cheng	Calvin Kocienda
3.2.2: Confusion Matrices	Nicholas Cheng	Calvin Kocienda
3.2.3: Receiver Operating Characteristic (ROC) and Area Under theCurve (AUC)	Nicholas Cheng	Calvin Kocienda
3.2.4: Precision and Recall	Nicholas Cheng	Calvin Kocienda
3.2.5: F1 Score	Nicholas Cheng	Calvin Kocienda
3.3: Machine Learning Tools		
3.3.1: PyTorch	Vinay Nair	Jesse Abeyta/ Bryan Gass
3.3.2: Scikit-Learn	Calvin Kocienda	Vinay Nair
3.3.3: NumPy	Jesse Abeyta	Jesse Abeyta
3.3.4: Pandas	Vinay Nair	Jesse Abeyta
3.4: Data Visualization Tools		
3.4.1: Tableau	Vinay Nair	Calvin Kocienda
3.4.2: Matplotlib	Vinay Nair	Calvin Kocienda

3.5: Classification Algorithms		
3.5.1: Linear Regression	Bryan Gass	Vinay Nair
3.5.2: Naive Bayes	Bryan Gass	Jesse Abeyta
3.5.3: Support Vector Machines	Vinay Nair	Jesse Abeyta
3.5.4: Expectation Maximization	Nicholas Cheng	All
3.5.5: BERT and TweetBERT	Calvin Kocienda	Vinay Nair
3.6: Neural Networks	Nicholas Cheng	Calvin Kocienda/ Bryan Gass
3.6.1: Activation Functions	Nicholas Cheng	Calvin Kocienda
3.6.2: Back Propagation	Nicholas Cheng	Calvin Kocienda/ Bryan Gass
3.7: PU Algorithms	Calvin Kocienda	
3.7.1: PU Assumptions	Jesse Abeyta	Calvin Kocienda
3.7.2: Single Training Set Scenario vs Case Control Scenario	Vinay Nair	Jesse Abeyta
3.7.3: PU Algorithms		
3.7.4: P.U.L.S.E	Bryan Gass	Jesse Abeyta
3.7.5: TIcE	Jesse Abeyta	Nicholas Cheng
3.7.6: Propensity Weighted Loss		
3.7.7: Subclass Prior Estimation	Jesse Abeyta	Nicholas Cheng
3.7.8: Propensity Weighted Loss with Subclass Prior Estimation	Jesse Abeyta	Nicholas Cheng
Chapter 4: Related Works	Calvin Kocienda	Jesse Abeyta/ Vinay Nair
Chapter 5: Methodology		
5.1: Data Preprocessing *5.1.1: BERT and TweetBERT*	Calvin Kocienda	Vinay Nair
5.1.2: Performance Baselines and Data Pipeline	Bryan Gass	Jesse Abeyta
5.1.3: Pipeline	Jesse Abeyta	Bryan Gass
5.2: Model Selection		
5.2.1: One-Step NB	Bryan Gass	Calvin Kocienda
5.2.2: One-Step SVM	Vinay Nair	Calvin Kocienda
5.2.3: Two-Step NB-SVM	Jesse Abeyta	Calvin Kocienda
5.2.4: Two-Step EM-NB	Nicholas Cheng	Calvin Kocienda
5.2.5: Non-Negative PU Loss	Calvin Kocienda	Nicholas Cheng
5.2.6: BERT and TweetBERT	Calvin Kocienda	Vinay Nair
5.2.7: TiCE	Nicholas Cheng	Jesse Abeyta
5.2.8: Propensity Weighted Loss	Nicholas Cheng	Jesse Abeyta

5.2.9: Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
5.2.10: Propensity Weighted Loss with Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
Chapter 6: Experimental Analysis		
6.1: Datasets		
6.1.1: Online Shopping Dataset	Calvin Kocienda	Vinay Nair
6.1.2: Twitter Datasets	Vinay Nair	Calvin Kocienda
6.2: Evaluation Metrics		
6.2.1: Method Modification PU Learning Algorithms	All	Calvin Kocienda
6.2.2: TweetBERT	Calvin Kocienda	Vinay Nair
6.2.3: TIcE and Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
6.2.4: Propensity Weighted Loss	Nicholas Cheng	Jesse Abeyta/ Bryan Gass
6.2.5: Propensity Weighted Loss with Class Prior Estimation	Nicholas Cheng	Jesse Abeyta
6.3: Experiment Descriptions		
6.3.1: Method Modification PU Learning Algorithms	All	Calvin Kocienda
6.3.2: TweetBERT Classifier	Calvin Kocienda	Vinay Nair
6.3.3: Propensity Weighted Loss	Nicholas Cheng	Jesse Abeyta
6.3.4: TIcE	Jesse Abeyta	Nicholas Cheng
6.3.5: Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
6.3.6: Propensity Weighted Loss with Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
Chapter 7: Discussion		
7.1: Method Modification PU Learning Algorithms	All	Calvin Kocienda
7.2: TweetBERT Model	Calvin Kocienda	Vinay Nair
7.3: TIcE	Jesse Abeyta	Nicholas Cheng
7.4: Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
7.5: Propensity Weighted Loss	Nicholas Cheng	Jesse Abeyta
7.6: Propensity Weighted Loss with Subclass Prior Estimation	Nicholas Cheng	Jesse Abeyta
Chapter 8: Conclusion	Calvin Kocienda	Nicholas Cheng
8.1: Table of Accomplishments	All	All
8.2: Future Steps	Calvin Kocienda	All