CSC–4220/5220
Dr. Akond Rahman
Project Update 3 Report
Jonathan A. Gibson, Andrew P. Worley, & Raunak S. Hakya

1. Introduction

This project update is the final portion of the CSC–4220/5220 course and is an extension of project update 2. It encompasses the use of the same dataset used in previous update, partitions the dataset into four sub-groups (based on *REPO_TYPE*), and evaluates six different classification models with transfer learning. Five of the classification models were the same from the previous project update, but we added a sixth classification model by building, training, and testing a Deep Neural Network (DNN).

2. Motivation

To us, it was motivating to work on this project because we could see how all the machine learning algorithms we have learned about this semester can be applied in a real-world project. Even the thought of automatically extracting lots of useful information, from a large dataset, was also very motivating.

Another motivation is the potential for additional employment opportunities after graduation, given our acquired skillset. Data analysis is quite a hot topic nowadays. Data scientists and machine learning software developers are in high demand. Companies have massive amounts of data and require people with our type of skills to extract knowledge and hidden insights from these datasets.

Furthermore, this practical first approach to learning and having our own data analysis project in our resume will provide us with satisfaction and confidence during job hunting and interviews.

3. Objective

1) Fit classification models for all four partitioned datasets using the Decision Tree (DTC), Random Forest (RFC), k-Nearest Neighbor (KNN), Artificial Neural Network (ANN), and Naive Bayes (NBC) algorithms. Test and evaluate pairs of datasets for transfer learning.
2) Build a DNN with Keras, train the network on all four partitioned datasets, and test/evaluate pairs of datasets for transfer learning, as shown in Figure 1.
3) Oversample the minority class (*INSECURE*) and undersample the majority class (*NEUTRAL*) for class rebalancing using *SMOTE*.
4) Repeat steps (1) and (2) with the resampled datasets and compare the results.
5) To use a static security analysis tool to discover potential security vulnerabilities.

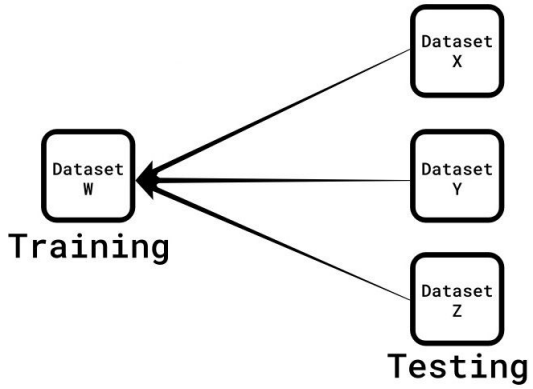

Fig. 1: Concept of transfer learning between similar domains.

4. Methodology

The dataset and classifier requirements required several steps of preparation to complete. Figure 2 shows the general flow of our entire program for update 3.

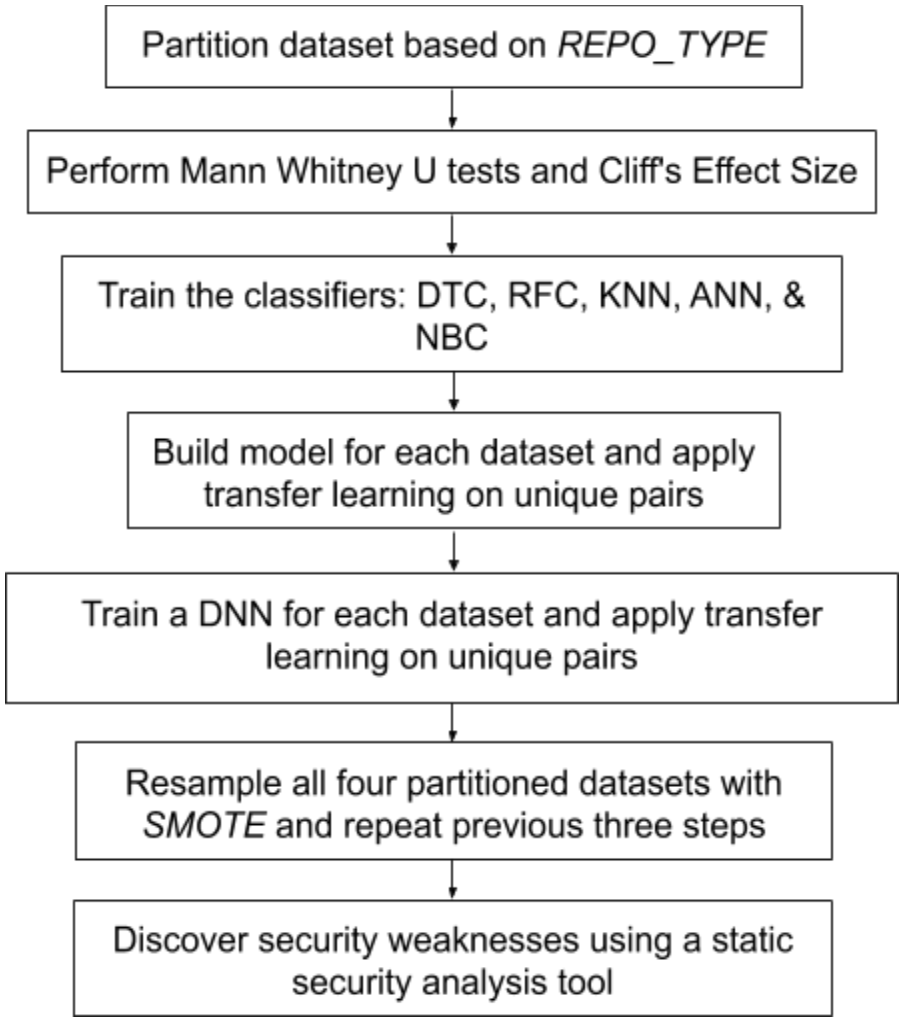

Fig. 2: Different phases of project update 3.

4.1. Dataset Preprocessing

First, we generated partitioned datasets from the original dataset based on the value of the *REPO_TYPE* column. We accomplish this by looping through the original dataset multiple times, with each iteration extracting all records that match one of the predetermined values (*ComputationalChemistry, Astronomy,* & *ComputationalBiology*). The extracted records are dropped from the main dataset, allowing sequential iterations to proceed quicker. Once all predetermined subsets are found, the remaining records are collected into a subset called *Others*. We saved all four of the partitioned subsets as entries in the <*datasets*> array. For ease of classification, the *SECU_FLAG* column of each subset is encoded to '0' for *NEUTRAL* and '1' for *INSECURE*.

We then examined the total count of both the *NEUTRAL* and *INSECURE* records for each of the new datasets, including their percentages, to clearly represent the class imbalance that existed before resampling. Moreover, we perform Mann–Whitney U tests and Cliff's Delta Effect Size calculations for each of the seven valid features from the datasets, to compare differences between the two classes. The columns that we used as features were *ADD_LOC*, *DEL_LOC*, *TOT_LOC*, *DEV_EXP*, *DEV_RECENT*, *PRIOR_AGE*, & *CHANGE_FILE_CNT*.

4.2. Transfer Learning with Classification Models

After all four subsets are generated, each is used to train five different classifiers. We built a model object for each and stored them in a matrix called <*model_list_matrix*>, as shown in Table 1. This format allows easy access to each model object, of the prior learners, for the next stages.

Row	0					1					2					3				
	CHM					AST					BIO					OTH				
	DTC	RFC	KNN	ANN	NBC	DTC	RFC	KNN	ANN	NBC	DTC	RFC	KNN	ANN	NBC	DTC	RFC	KNN	ANN	NBC
Column	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4

Table 1: Layout of <*model_list_matrix*>

We then applied transfer learning by testing trained subset models against each other. For example, we trained a DTC with the CHM dataset but evaluated the results of testing the AST, BIO, and OTH datasets against the CHM DTC object. We applied this concept for all possible pairs and received 60 result sets. We accomplish this with three loops. The outer two loops iterated through each of the subsets, skipping any combination that would test a dataset against itself. The innermost loop iterated over all five classifiers. After each test, the innermost loop prints the confusion matrix and classification report for the current combination. The results allowed us to calculate and view the values of accuracy, precision, recall, and F-score for all result sets.

4.3. Deep Neural Network Construction

Next, we built a DNN using the Keras library and a Tensorflow backend. In this step, we create a Sequential model and add a stack of fully connected layers to it one at a time. Here, we defined three fully connected network layers using the "Dense" type. We specified the number of input features while adding the first layer with the <*input_dim*> argument and setting it to the total number of full model features. This means that the model expects rows of data with seven input variables (because there are seven full-model features). Also, we specified the *ReLU* activation function using the activation argument on the first two layers. However, in the case of the third layer which is the output layer, we used a *sigmoid* activation function. This is to ensure that the output of our deep neural network is normalized between '0' and '1'. Table 2 shows the structure of our DNN built with Keras.

Table 2: Structure of the Deep Neural Network

Model: "sequential_1"		
Layer (type)	**Output Shape**	**Param #**
Input (Dense)	(None, 12)	96
Hidden (Dense)	(None, 7)	91
Output (Dense)	(None, 1)	8
Total params: 195 Trainable params: 195 Non-trainable params: 0		

After defining the model, we configured the learning process to train the model. This is done via the compile method. We set the optimizer argument to *adam*, which is an efficient stochastic gradient descent algorithm. And since this is a binary classification problem, we used cross-entropy; for this, we define the loss argument to *binary_crossentropy*. Also, we set the metrics argument to *["acc"]* in order to gather and report the classification accuracy. Furthermore, we compared the output predictions of the DNN with the ground truth to calculate the true positives, true negatives, false positives, and false negatives. We used these counts to calculate the values of precision, recall, and F-score for the DNN, while also returning the confusion matrix.

Next, we trained the DNN model on our datasets. For this, we defined the number of epochs and the batch size to be 50 and 10 respectively. This means that the training process will run for a total of 50 iterations through the dataset and will consider 10 dataset rows before the model weights are updated within each epoch. Thus, each epoch will involve 5 (50/10) updates to the model weights.

4.4. Application of SMOTE

In this stage, we applied the *SMOTE* algorithm to each of the datasets. We used the *SMOTENN* function, with the sampling strategy parameter set to *"all"* to perform under and over-sampling while using the Edited Nearest Neighbours (ENN) algorithm to clean up the decision boundary between the two classes. The new class distribution of the *SECU_FLAG* column was printed. The new resampled datasets are then used to train a new set of classifiers which were then cross-tested with transfer learning in an identical manner to the original, unsampled datasets. Table 3 shows the class distribution before and after *SMOTE* resampling.

Table 3. Class distribution before and after SMOTE resampling.

Original								
Subsets	**ComputationalChemistry**		**Astronomy**		**ComputationalBiology**		**Others**	
	Count	%	Count	%	Count	%	Count	%
NEUTRAL	49397	94.28	62496	92.80	159216	91.01	5151	95.67
INSECURE	2998	5.72	4848	7.20	15717	8.99	233	4.33
Total	52395	100.00	67344	100.00	174933	100.00	5384	100.00
Resampled								
Subsets	**ComputationalChemistry**		**Astronomy**		**ComputationalBiology**		**Others**	
	Count	%	Count	%	Count	%	Count	%
NEUTRAL	34174	46.25	41222	45.71	109019	46.18	3601	46.13
INSECURE	39721	53.75	48964	54.29	127036	53.81	4205	53.87
Total	73895	100.00	90186	100.00	236055	100.00	7806	100.00

4.5. Security Analysis

In order to run a static security analysis, we used "Bandit" which is a tool designed to find security issues in Python code. This tool processes each file, builds an Abstract Syntax Tree (AST) from it, and runs appropriate plugins against the AST nodes. And after finishing the scanning of all the files, it generated a report. We ran Bandit against all three of our project update folders and the utility folder that defined all of our helper functions, while excluding the dataset and output files. Bandit found no security weaknesses. Figure 3 shows the results of our Bandit security analysis.

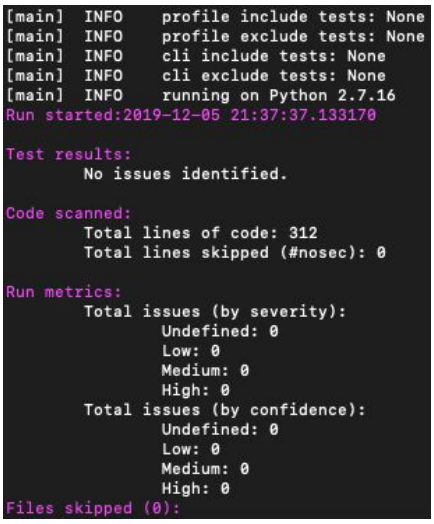

Fig. 3: Result of the Bandit security analysis.

5. Results

Table 4: Accuracy, NEUTRAL F-Score, & INSECURE F-Score for Prior Learners before resampling.

Using Original Datasets

	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
DTC	Training	Chemistry		(0.85, 0.92, 0.08)	(0.85, 0.92, 0.10)	(1.00, 1.00, 0.99)
		Astronomy	(0.87, 0.93, 0.07)		(0.85, 0.92, 0.10)	(1.00, 1.00, 0.99)
		Biology	(0.87, 0.93, 0.07)	(0.85, 0.92, 0.08)		(1.00, 1.00, 0.99)
		Others	(0.87, 0.93, 0.07)	(0.85, 0.92, 0.08)	(0.85, 0.92, 0.10)	

	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
RFC	Training	Chemistry		(0.93, 0.96, 0.00)	(0.91, 0.95, 0.00)	(1.00, 1.00, 0.98)
		Astronomy	(0.93, 0.96, 0.03)		(0.91, 0.95, 0.00)	(1.00, 1.00, 0.98)
		Biology	(0.93, 0.96, 0.03)	(0.93, 0.96, 0.00)		(1.00, 1.00, 0.98)
		Others	(0.93, 0.96, 0.03)	(0.93, 0.96, 0.00)	(0.91, 0.95, 0.00)	

	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
KNN	Training	Chemistry		(0.92, 0.96, 0.02)	(0.90, 0.95, 0.01)	(0.96, 0.98, 0.32)
		Astronomy	(0.92, 0.96, 0.06)		(0.90, 0.95, 0.01)	(0.96, 0.98, 0.32)
		Biology	(0.92, 0.96, 0.06)	(0.92, 0.96, 0.02)		(0.96, 0.98, 0.32)
		Others	(0.92, 0.96, 0.06)	(0.92, 0.96, 0.02)	(0.90, 0.95, 0.01)	

	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
ANN	Training	Chemistry		(0.83, 0.91, 0.09)	(0.74, 0.85, 0.08)	(0.93, 0.97, 0.10)
		Astronomy	(0.76, 0.86, 0.09)		(0.74, 0.85, 0.08)	(0.93, 0.97, 0.10)
		Biology	(0.76, 0.86, 0.09)	(0.83, 0.91, 0.09)		(0.93, 0.97, 0.10)
		Others	(0.76, 0.86, 0.09)	(0.83, 0.91, 0.09)	(0.74, 0.85, 0.08)	

	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
NBC	Training	Chemistry		(0.26, 0.36, 0.13)	(0.20, 0.24, 0.16)	(0.71, 0.83, 0.11)
		Astronomy	(0.28, 0.40, 0.10)		(0.20, 0.24, 0.16)	(0.71, 0.83, 0.11)
		Biology	(0.28, 0.40, 0.10)	(0.26, 0.36, 0.13)		(0.71, 0.83, 0.11)
		Others	(0.28, 0.40, 0.10)	(0.26, 0.36, 0.13)	(0.20, 0.24, 0.16)	

The results of the first transfer learning stage were not as expected. There was a distinct difference between testing with the *Other* dataset and the remaining sets. As shown in Table 4, the best results were found while classifying the *Others* group with all metrics of nearly 1.00 for the decision tree and random forest classifiers, a substantial difference compared to the low scores with other datasets. There are three reasons why this might be the case. First, the *Others* dataset is far smaller than the other datasets and may be able to test better on training sets that are much larger than itself. Second, it may be the case that the decision tree random forest classifiers are very well suited for this type of data. Third, it may be that the combination of two repository types caused the models to distinguish between classes better. Further investigation and experimentation will be needed to validate any of these theories. All of the other classification models did not learn the features of the minority class very well. However, this result is to be expected. When there is a large class imbalance present in a dataset, the majority class typically has far more rows available for the model to learn from. As a result, the classification models will predict the majority class well, and only occasionally misclassify a majority class example as a minority class example, while being unsure of how to properly predict the minority class. Although the accuracy and *NEUTRAL* F-scores for the other datasets were relatively high, these metrics can be deceiving. The metric that we are most interested in is the *INSECURE* F-score. This value indicates how well the models were able to classify the minority class, which is the goal.

Table 5: Accuracy, NEUTRAL F-Score, & INSECURE F-Score for Prior Learners after resampling.

Using Resampled Datasets						
DTC	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.64, 0.65, 0.62)	(0.60, 0.63, 0.56)	(1.00, 1.00, 1.00)
		Astronomy	(0.61, 0.64, 0.56)		(0.60, 0.63, 0.56)	(1.00, 1.00, 1.00)
		Biology	(0.61, 0.64, 0.56)	(0.64, 0.65, 0.62)		(1.00, 1.00, 1.00)
		Others	(0.61, 0.64, 0.56)	(0.64, 0.65, 0.62)	(0.60, 0.63, 0.56)	
RFC	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.62, 0.66, 0.57)	(0.59, 0.66, 0.48)	(1.00, 1.00, 1.00)
		Astronomy	(0.61, 0.67, 0.54)		(0.59, 0.66, 0.48)	(1.00, 1.00, 1.00)
		Biology	(0.61, 0.67, 0.54)	(0.62, 0.66, 0.57)		(1.00, 1.00, 1.00)
		Others	(0.61, 0.67, 0.54)	(0.62, 0.66, 0.57)	(0.59, 0.66, 0.48)	
KNN	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.50, 0.54, 0.46)	(0.47, 0.57, 0.33)	(0.99, 0.99, 0.99)
		Astronomy	(0.51, 0.58, 0.40)		(0.47, 0.57, 0.33)	(0.99, 0.99, 0.99)
		Biology	(0.51, 0.58, 0.40)	(0.50, 0.54, 0.46)		(0.99, 0.99, 0.99)
		Others	(0.51, 0.58, 0.40)	(0.50, 0.54, 0.46)	(0.47, 0.57, 0.33)	
ANN	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.56, 0.53, 0.58)	(0.41, 0.45, 0.36)	(0.72, 0.63, 0.78)
		Astronomy	(0.53, 0.43, 0.60)		(0.41, 0.45, 0.36)	(0.72, 0.63, 0.78)
		Biology	(0.53, 0.43, 0.60)	(0.56, 0.53, 0.58)		(0.72, 0.63, 0.78)
		Others	(0.53, 0.43, 0.60)	(0.56, 0.53, 0.58)	(0.41, 0.45, 0.36)	
NBC	(Acc, N_F1, I_F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.54, 0.02, 0.70)	(0.54, 0.03, 0.70)	(0.55, 0.07, 0.70)
		Astronomy	(0.53, 0.07, 0.69)		(0.54, 0.03, 0.70)	(0.55, 0.07, 0.70)
		Biology	(0.53, 0.07, 0.69)	(0.54, 0.02, 0.70)		(0.55, 0.07, 0.70)
		Others	(0.53, 0.07, 0.69)	(0.54, 0.02, 0.70)	(0.54, 0.03, 0.70)	

One of the side effects that we expected from resampling was the sacrifice of majority class results for the betterment of minority class results; this expectation was met, as shown by the results in Table 5. The action of resampling all of the datasets left the class distribution at a near 50/50 split. As a result, the classification models were able to predict the minority class much better than before, but the deletion of majority class examples hindered the learning ability for the *NEUTRAL* class. However, the decision tree and random forest classifiers maintained their high quality. One of the most surprising changes was the change made by the KNN classifier trained on the *Others* dataset. Although it returned unsatisfactory before resampling, it returned near-perfect results for the *INSECURE* class, while still maintaining its results for the *NEUTRAL* class, after resampling. This proves to us that the KNN classifier is also a valid candidate for this type of data.

Table 6: Accuracy, precision, recall, and F-score for the DNN before resampling.

Using Original Datasets						
DNN	(Acc, Prec, Rec, F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.93, 0.00, 0.00, 0.00)	(0.91, 0.00, 0.00, 0.00)	(0.96, 0.00, 0.00, 0.00)
		Astronomy	(0.94, 0.00, 0.00, 0.00)		(0.91, 0.00, 0.00, 0.00)	(0.96, 0.00, 0.00, 0.00)
		Biology	(0.94, 0.00, 0.00, 0.00)	(0.93, 0.00, 0.00, 0.00)		(0.96, 0.00, 0.00, 0.00)
		Others	(0.94, 0.00, 0.00, 0.00)	(0.93, 0.00, 0.00, 0.00)	(0.91, 0.00, 0.00, 0.00)	

Table 7: Accuracy, precision, recall, and F-score for the DNN after resampling.

Using Resampled Datasets						
DNN	(Acc, Prec, Rec, F1)		Testing			
			Chemistry	Astronomy	Biology	Others
	Training	Chemistry		(0.55, 1.00, 0.55, 0.70)	(0.54, 1.00, 0.54, 0.70)	(0.54, 1.00, 0.54, 0.70)
		Astronomy	(0.54, 1.00, 0.54, 0.70)		(0.54, 1.00, 0.54, 0.70)	(0.54, 1.00, 0.54, 0.70)
		Biology	(0.54, 1.00, 0.54, 0.70)	(0.54, 1.00, 0.54, 0.70)		(0.54, 1.00, 0.54, 0.70)
		Others	(0.54, 1.00, 0.54, 0.70)	(0.54, 1.00, 0.54, 0.70)	(0.54, 1.00, 0.54, 0.70)	

The results of training our DNN on the initial datasets were insightful. As shown in Table 6, all the tests had high accuracy; however, closer inspection reveals that the accuracy perfectly matches the ratio of the *NEUTRAL* entries in the datasets. This is the result of the DNN predicting that every record was *NEUTRAL*. This also appears in the confusion matrix as a complete lack of both true positives and false positives. This type of behavior is expected for a dataset with such harsh class imbalance. As the DNN tries to minimize its loss function, one easy way to achieve minimization with high accuracy is to simply classify every example as the majority class.

We expected that after resampling that the DNN results would be slightly more balanced. The results did match our prediction as more true positives were found; however, further review revealed a similar problem to the initial testing sets. In contrast to the previous results, the DNN decided that almost all entries were *INSECURE*; only one to four entries per dataset were correctly labeled as *NEUTRAL*. While using the rebalanced datasets did increase our scoring metrics, this was not the behavior we desired.

Table 8: P-Value and Cliff's Effect Size

(p-val, ∂)	P-Value & Cliff's Delta						
	ADD_LOC	DEL_LOC	TOT_LOC	DEV_EXP	DEV_RECENT	PRIOR_AGE	CHANGE_FILE_CNT
Chemistry	(1.00, 0.12)	(0.99, 0.07)	(1.00, 0.10)	(0.00, 0.07)	(0.00, 0.03)	(0.03, 0.02)	(0.99, 0.05)
Astronomy	(1.00, 0.07)	(0.99, 0.03)	(0.99, 0.05)	(0.00, 0.05)	(0.00, 0.07)	(0.72, 0.01)	(0.03, 0.01)
Biology	(1.00, 0.10)	(0.99, 0.03)	(1.00, 0.07)	(1.00, 0.13)	(1.00, 0.17)	(0.02, 0.01)	(1.00, 0.05)
Others	(0.99, 0.17)	(0.99, 0.12)	(0.99, 0.13)	(0.99, 0.13)	(0.58, 0.01)	(0.00, 0.11)	(0.97, 0.07)

The purpose of calculating the P-values and the Effect Size is to compare the similarities between features or feature values, with a certain confidence. When comparing features, one of the goals is to find features that are very similar to each other so that they can be omitted from the model. Features that are too similar will not benefit the model predictions, but rather cause unwanted bias. We, however, compared the *NEUTRAL* and *INSECURE* values against each other for each of our seven features. By doing this, we can see whether there are stark differences between the properties of a *NEUTRAL* class example and an *INSECURE* class example. A value close to '0' means that the values are similar, while a value close to '1' means that the values are dissimilar. Unfortunately, the ∂ values that were returned showed that all of the values for our features were rather similar. This means that the decision boundaries for binary classification models were not obvious. The effect of this can be seen in Table 4, where the classification models performed very poorly, with regards to predicting the minority class. The classification models were able to slightly redeem themselves after the resampling of the data by learning some of the class intricacies, but more data may be required to achieve better results.

6. Conclusions and Possible Improvements

From the presented results, we can safely draw a few conclusions. Unbalanced datasets can have distinct detrimental effects on the accuracy of classifiers trained on the data. In addition, the effect can vary greatly depending on the algorithm used for the training. We can also conclude, for this particular dataset, that the *SMOTE* algorithm drastically improved the quality of the training for a majority of the classifiers tested.

Additionally, we can conclude that transfer learning is a viable method to see if classification models are able to generalize during the learning process (rather than overfitting to the data they trained on).

From our results, it is clear that the current DNN far from optimal. There are several things that we could adjust in the future to modify the performance. We could change the optimizer, add additional layers, adjust the size of the layers, change the connections between the layers, optimize the parameters, calculate loss based on the recall score rather than the accuracy, change the number of epochs, change the batch size, and the like.

There are also some obvious optimizations that we could apply to the prior learner classifiers and the *SMOTE* algorithm. For the most part, we allowed *scikit-learn to* use the default parameters when we used API calls. There are many parameters available to tune for the classifier and *SMOTE* objects that we did not attempt to modify.

Finally, we could also use effect size calculations to compare our features with each other directly, rather than just comparing the difference between our binary classes. If two or more of the features are similar, it is possible that we could see immediate improvements in our models by removing a subset of our, currently seven, features.

7. Project Code Execution Instructions

- Install a Python2 interpreter, preferably Python 2.7.
- If you don't have these packages already, run "pip install <*package_name*>" where <*package_name*> is:
    - imbalanced-learn==0.4.2 // the latest version does not work with Python2
    - keras
    - tensorflow
    - bandit
    - statistics
    - scipy
    - tqdm
    - pandas
    - numpy
    - sklearn
- Clone the repository from https://gitlab.csc.tntech.edu/apworley42/csc-5220---group-project/tree/Devel by direct download, by using SSH (git@gitlab.csc.tntech.edu:apworley42/csc-5220---group-project.git), or HTTP (https://gitlab.csc.tntech.edu/apworley42/csc-5220---group-project.git). The appropriate source code exists in the "Devel" branch.
- You can run the Bandit security analysis by using "make run" from the root directory.
- You can run update 3 by using "make run" from the "update_3/" directory.
- The program is currently limited on the number of acceptable names for the dataset. If you misformat the program execution, the program will tell you the problem and how to fix it. You can also look in the source code or the *Makefile* for specifics on how to properly run the program.