Daniel Fiume

Professor Zsom

DATA 1030

15 December 2024

Predicting Data Breach Severity on Publically Reported Breaches

Affiliation: Brown University

Github Repo: https://github.com/dfiume1/data-breach-ml.git

Important Note: Because the dataset used is not publicly available, there is no /data folder. I can email it to you if you would like to reproduce the code.

I : INTRODUCTION

When a company such as Google has a data breach, it often makes national news. Headlines along the lines of "300 million uses data compromised in today's hack." However, thousands of these breaches happen yearly and are publicly reported on by law in most cases.

As someone interested in data privacy, I wanted to know more about these lesser known breaches. I discovered that the non-profit organization Privacy Rights Clearinghouse has a similar dataset that is much more expansive [1]. The team gave me free access to this non-public dataset for this project. I do not have previous work to compare to for this project.

I was interested to see if it was at all possible to use just information about the breach itself to determine its significance. This project aims to predict the severity of a publicly reported data breach; more explicitly, our target variable is the number of records that were stolen in the breach. The dataset contains 35,167 data breaches (only those that are publicly reported by government entities, so this analysis is limited to those and should not be generalized) with 28

features. Additionally, these breaches have to do with companies that house government information, and thus not all private company data breaches are represented. Most of these features are text based, and the features I used for my models were the type of organization breached, the type of breach (ex: hack, physical theft, accidental leak), the types and quantities of information and whether it was encrypted, relevant dates, which government body reported the breach, and the state of the organization breached.

II: EXPLORATORY DATA ANALYSIS

For each feature I explored its data type, distribution, the range of its values, and the number of missing data points. Some features had up to 70% of the data missing, which was a large challenge in this project. I removed rows without a value for my target variable as well as duplicated rows (to prevent data leakage), which left me with 24,586 breaches. The visualization of my target variable can be found in Figure 1. One thing that surprised me initially is that most of these reported data breaches were relatively small, with only a couple percent of the data representing breaches over a million records affected. Due to the variable's large range, I found it made more sense to plot these figures on a log scale, so in my final model the actual predicted value is the *log10* of the number of records breached. Since I am interested in predicting the relative size, a prediction of 100 records should be equally as far away from a prediction of 10,000 records if the true value is 1,000.

In Figure 2, I show an example of one of my categorical features, breach type, with the target variable. A lot of the large data breaches end up being hacks, but it seems clear that data breaches of most sizes can happen in a variety of different scenarios. In Figure 3, I show an interesting feature of this data set that shows the number of breaches reported in this data set

increase up until 2020, where they rapidly decrease. I found this very confusing, but it would make sense that the government bodies reporting these breaches were occupied by larger issues.

Figure 1: Target Variable Visualization

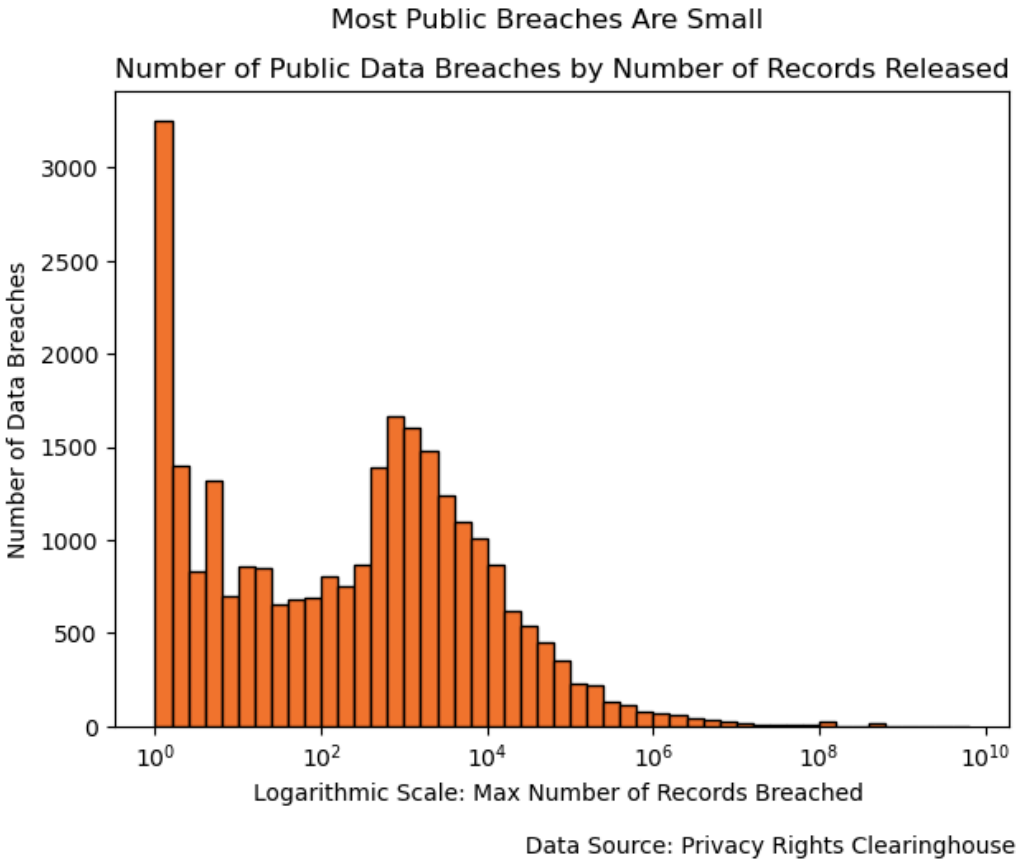
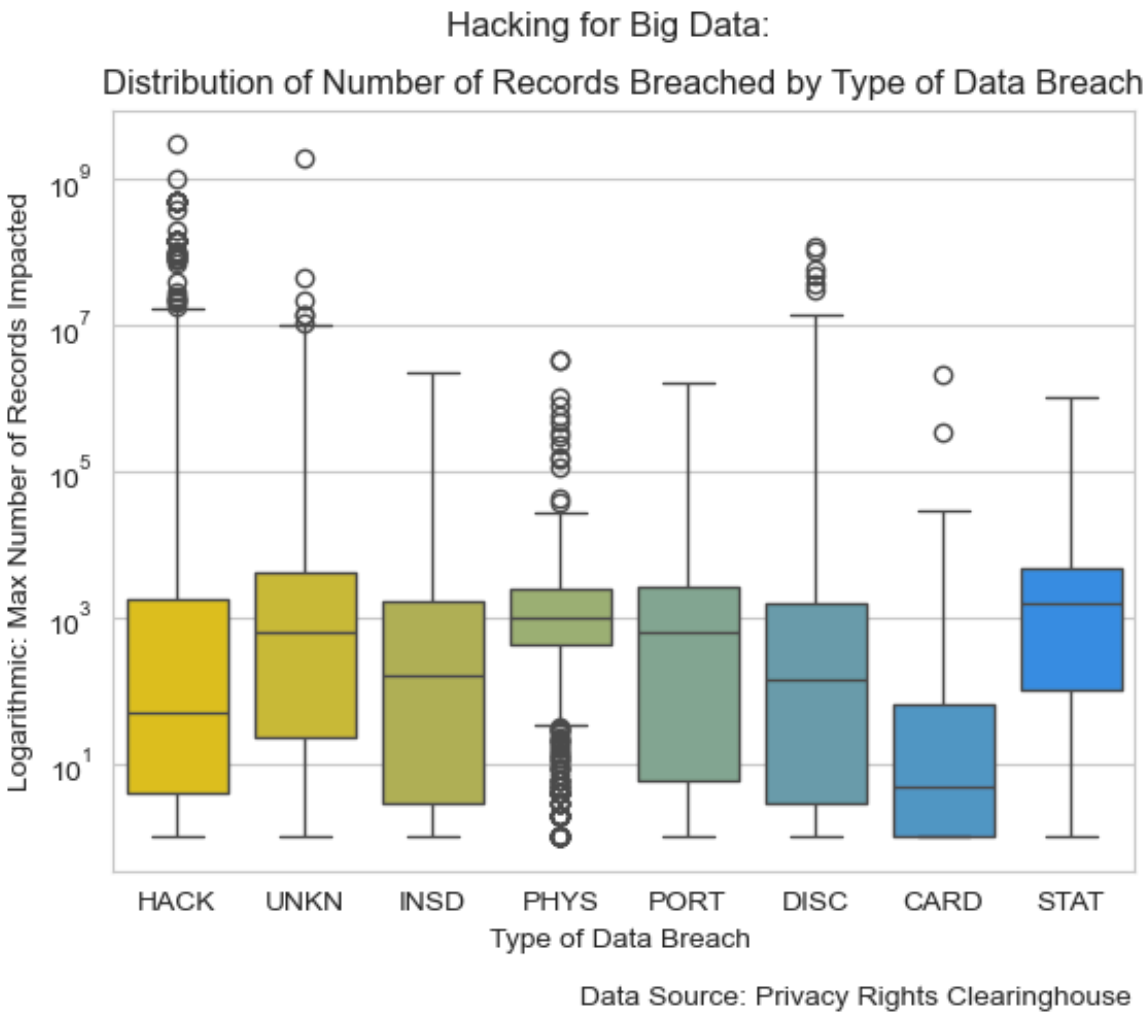


Figure 1: A histogram shows the data counts by the number of records breached on a log scale.

Figure 2: Number of Records by Breach Type and Breach Type Key



- **CARD**: Breaches involving credit or debit card fraud, not related to hacking activities.
- **HACK**: Breaches due to hacking or malware attacks by external parties.
- **INSD**: Insider breaches caused by employees, contractors, or customers.
- **PHYS**: Physical breaches involving paper documents that are lost, discarded, or stolen.
- **PORT**: Breaches involving the loss or theft of portable devices like laptops, smartphones, memory sticks, etc.
- **STAT**: Loss or theft of stationary computers or servers not designed for mobility.
- **DISC**: Unintended disclosures not involving hacking, intentional breaches, or physical loss.
- **UNKN (Unknown)**: Used when the breach type cannot be definitively determined.

Figure 2: A description of each type of data breach from the Privacy Rights Clearinghouse alongside their correlation with the target variable in box plot form.

Figure 3: Year Feature EDA

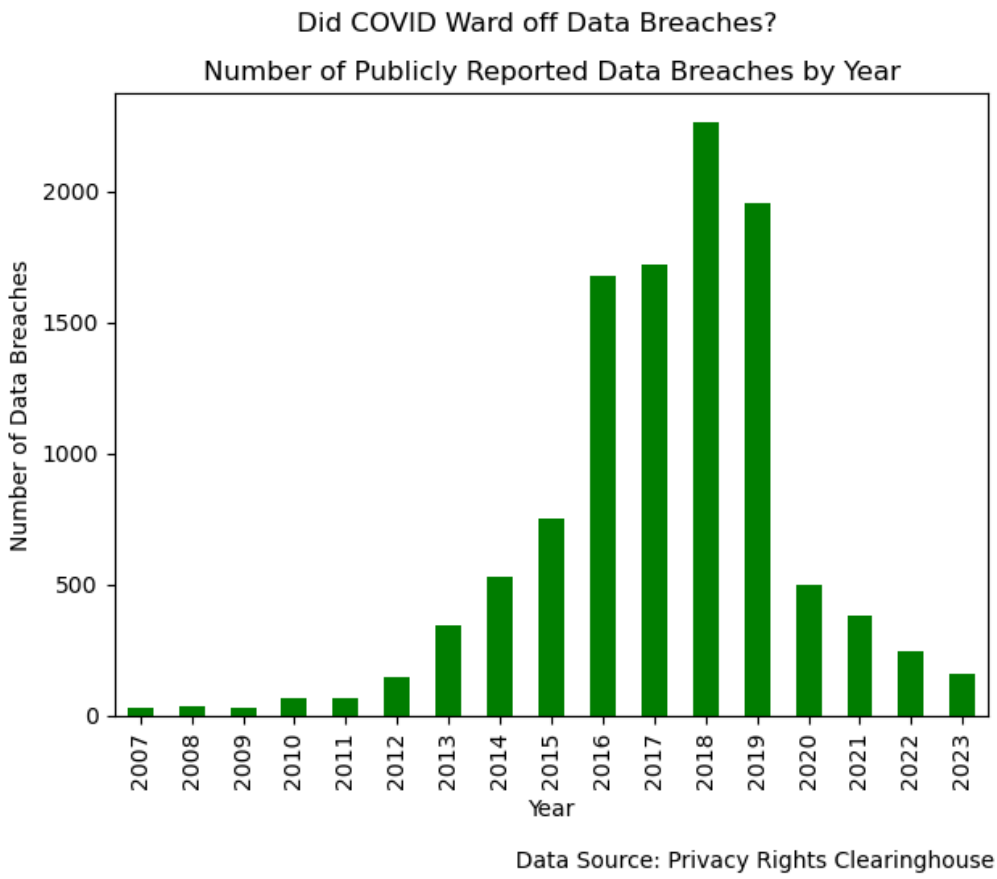


Figure 3: A histogram with the number of publicly reported breaches by the date of the breach.

Feature Engineering:

I performed feature engineering using the date features and the information types field. The dates I had available were the date of the breach, the date the breach was reported, and the date of the end of the breach. From these dates, I thought it would be interesting to see if there

was any correlation between how long the breach lasted as well as how long it took the breach to be reported. These are shown in Figure 4. It is noteworthy that the end date of the breach field had many missing values, and I assumed that a missing value meant the breach was instantaneous. Zeros are filled in for this field in the preprocessing section. In addition to these two date features, I added the year and month of the data breach as features.

I also performed some data cleaning with the information types field, as it was a json that listed the types of information breached as well as whether it was encrypted or not. Since each breach could have many types of information breached as well as multiple types of information, I decided to count the number of each type of information for each breach.

Figure 4: Engineered Date Features by Number of Records Breached

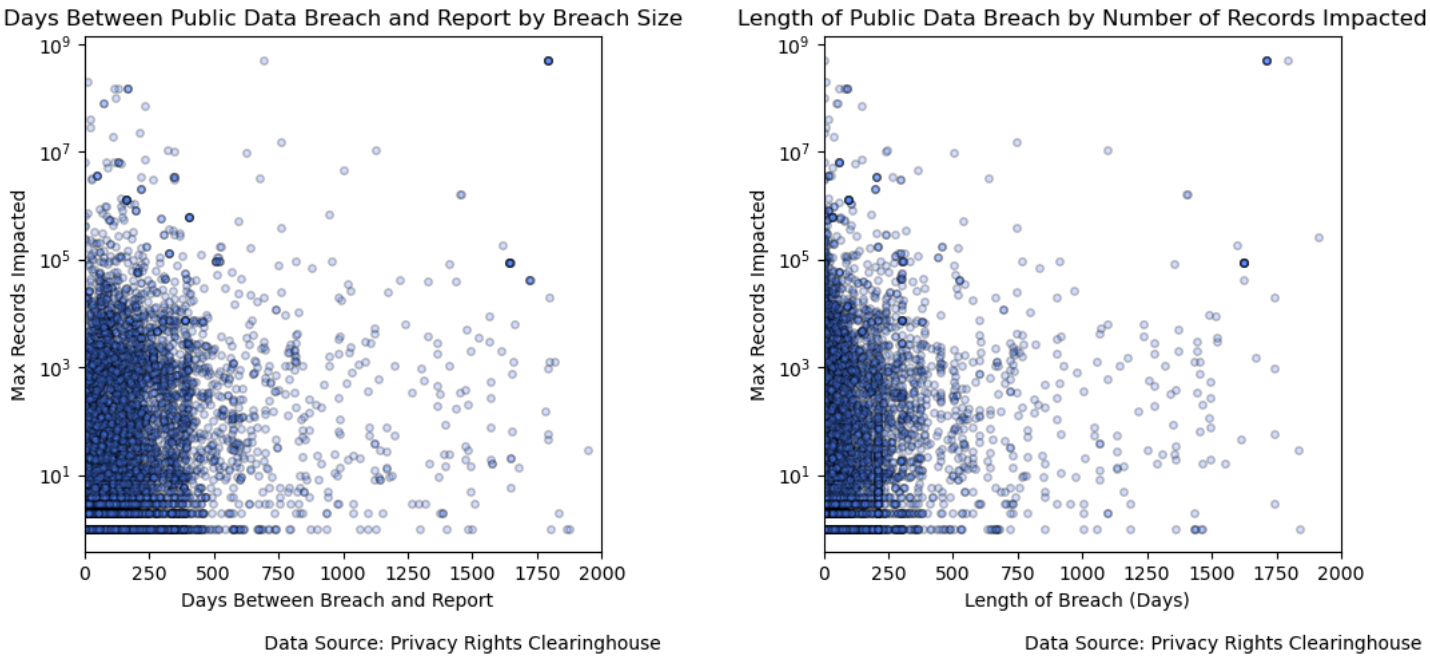


Figure 4: Scatter plots of the new engineered date features with the number of records breached.

III: METHODS

After the data cleaning and feature engineering, I had 24,586 data points with 29 features. First, I set up my preprocessors: my categorical features were processed using OneHotEncoders, and my numerical features were preprocessed using Standard Scalars. Missing values for my categorical data were handled by creating a new category, "UNKN", for each feature. Missing values for my numerical data were imputed as zero, as the only missing values were for the engineered date features as explained above.

My pipeline aided in model selection and hyperparameter tuning by performing 5-fold cross validation in combination with using grid search [2]. The first step of this pipeline is to split the data, where I decided to perform an 80-20 split for other and test respectively. The normal train test split here over many random states is fine for this problem because my problem is a regression problem, and it would not make sense to use a stratified or group split. I then used the "other" dataset to perform the 5-fold cross validation and grid search functions to select the best hyperparameters given a grid of hyperparameters based on the test score.

My test score for this project was RMSE, as the problem is a regression problem and it can be directly interpreted as the number of orders of magnitude on average the model is wrong in predicting the exact number of records affected by the breach. I also report the R2 score in my results for generalizability.

I decided to test a variety of linear and nonlinear models, and the results of my hyperparameter tuning and model selection are displayed in Figure 5. I used varying parameter grids over random seeds using my pipeline. If the best hyperparameters were at the edge of the grid, I changed the bounds, as well as narrowing in the values several times as well. An example of this for the XGBoost model was max_depth. I started with [1,3,5,10,30,100], found out my

best models were with 3, 5, and 10, and then changed the grid to [3,4,5,6,8,10,12,15], found out the best models were with 6, and narrowed it once again to [6,7] and then chose 6. This was done in conjunction with the other hyperparameters for this model as well. For all the final values in the hyperparameter table, values much larger and much smaller were tried.

In my pipeline for my final model testing, I use an 80-20 train/test split to train and test the model over 5 random seeds. This pipeline preprocesses the data identically as the first pipeline, and is identical except that it does not perform cross validation or grid search, and that it saves the models, predictions, and test sets.

Figure 5: The Models and their Hyperparameters

Model	**Hyperparameters Tuned**	**Final Values**
Linear Regression (L1 Regularization)	Regularization Parameter alpha	alpha = 0.001
Linear Regression (L2 Regularization)	alpha	alpha = 3
Random Forest Regressor	Number of Estimators, Max Tree Depth	n_estimators = 150 max_depth = 15
SVM Regressor	gamma, C	Gamma = 0.1, C = 100
KNN Regressor	Number of Neighbors	n_neighbors = 15
XGBoost Regressor	max_depth, min_child_weight,, col_sample_bytree subsample, gamma	max_depth = 6, min_child_weights = 3, sample = 0.8, gamma = 0.1
MLP Regressor	hidden_layer_sizes, alpha, max_iter	Hidden Layer sizes =(150,1) alpha=.0003, max_iter=300

Figure 5: Displays the model types and the hyperparameters that were tuned. The hyperparameters that were selected during the cross validation process are shown in the final values column.

IV: RESULTS

The model results are provided in Figure 6.The XGBoost models performed the best, followed closely behind the Random Forest models. The neural network performed better than the linear models, which performed better than some of the non-linear models in the SVM and KNNs. The XGBoost model results can be directly interpreted as mispredicting the number of records breached by one magnitude on average. This is visualized in Figure 7. The model does decently well on identifying mid-sized and large breaches, with only moderate success on the smallest breaches. Compared to the baseline model, the XGBoost performs on average 52 standard deviations better. While the models are able to identify indicators of larger breaches, at the end of the day, all organizations are at risk for data breaches in these variety of ways.

Figure 6: Model Results

Model (Over 5 Random Seeds)	RMSE Mean	RMSE Std	R2 Mean	R2 Std
Linear Regression (L1 Regularization)	1.132	0.013	0.445	0.007
Linear Regression (L2 Regularization)	1.131	0.013	0.445	0.006
Random Forest Regressor	**1.029**	0.018	0.541	0.008
SVM Regressor	1.186	0.009	0.387	0.015
KNN Regressor	1.144	0.008	0.432	0.008
XGBoost Regressor	**0.999**	0.010	0.567	0.005
MLP Regressor	**1.062**	0.017	0.510	0.012
Baseline	1.519	0.010	0	0

Figure 6: Means and Standard Deviations of the models performance over random seeds. The baseline model is calculated with the respective training set mean. Values rounded to 3 decimal places.

Figure 7: Displaying the Best Model's Predictions

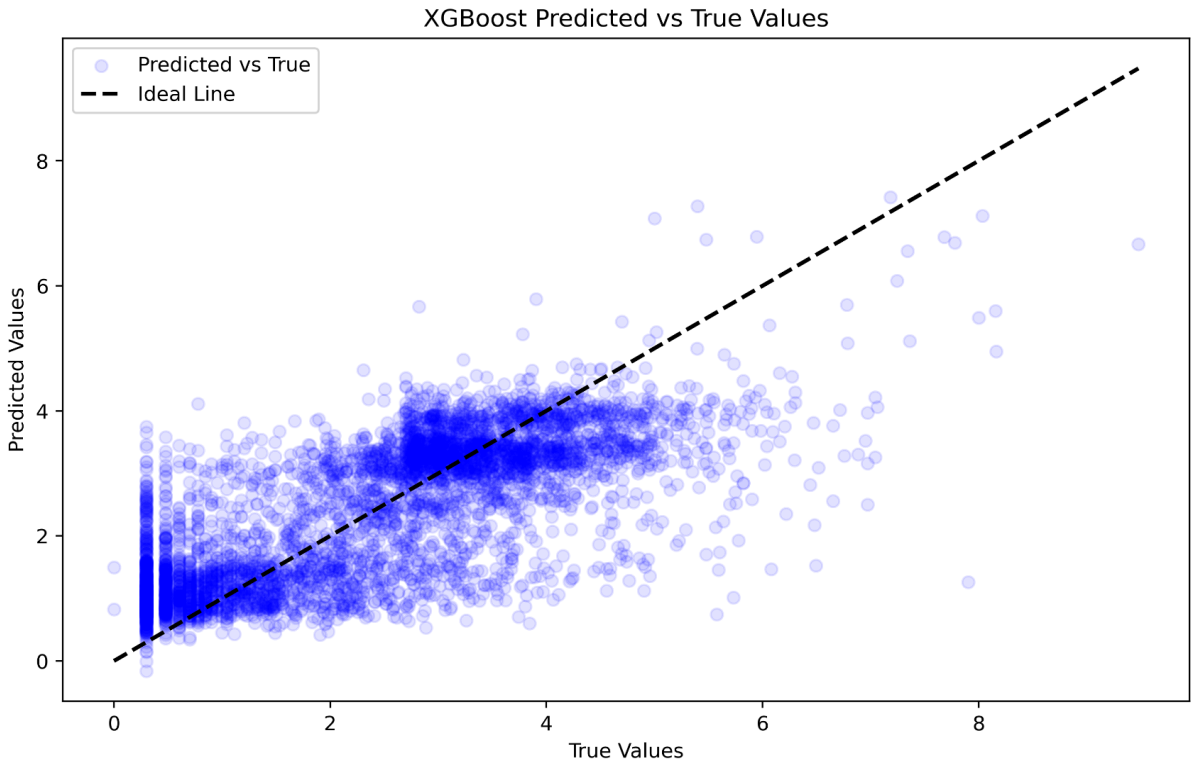


Figure 7: The log10 of the number of records predicted by the first XGBoost Model and the corresponding true value in the test set. The points are semi-translucent, so darker areas represent higher data density.

Global Feature Importance:

To further interpret these results, I calculated the global feature importance metrics of perturbation shown in Figure 8 and the five built in XGBoost metrics shown in Figure 9. The categorical feature of Source (which gov agency reported the breach) was surprisingly the most important feature holistically, but it is shown that all of the types of features (dates, types of information, location, types of breach/org) rank in the top10 in some categories. This balance of

feature importance by metric is interesting, and given the r2 scores of the results, is unsurprising since the model is not extremely successful.

Figure 8: Perturbed Feature Scores

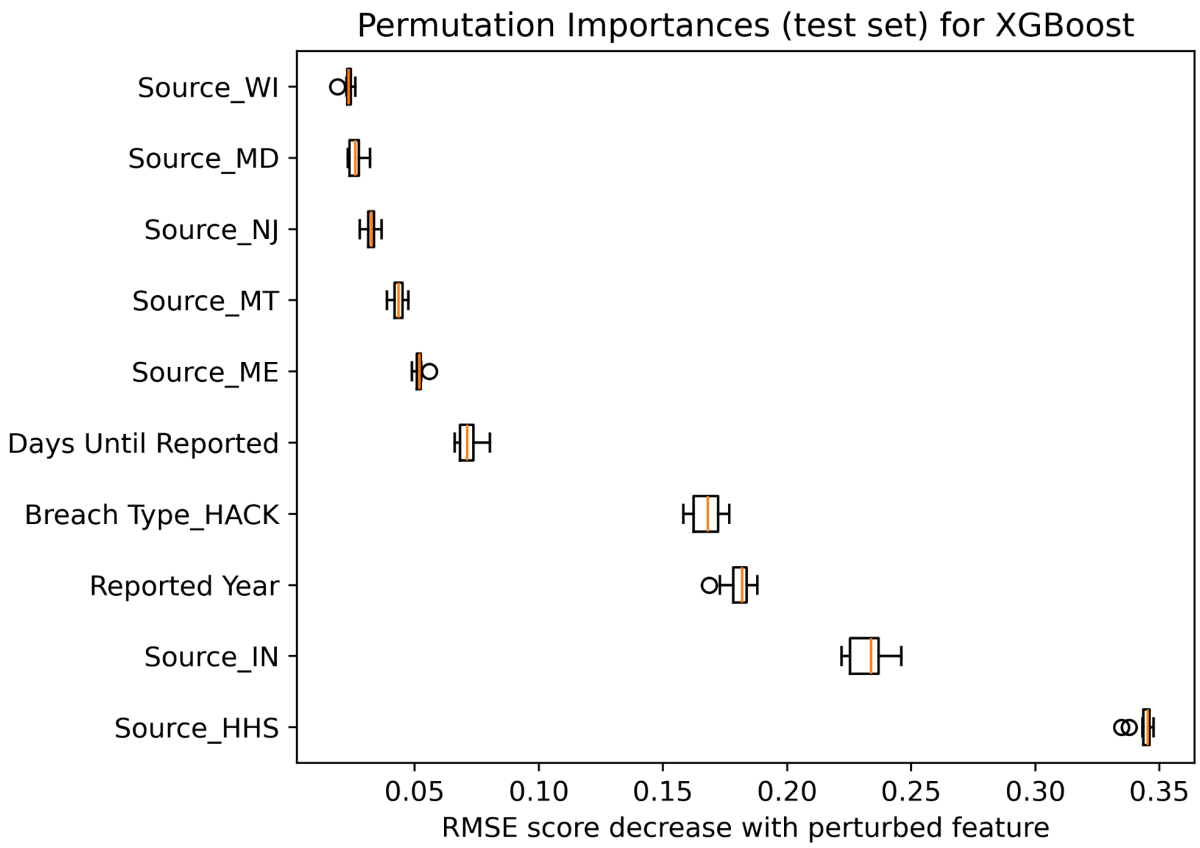
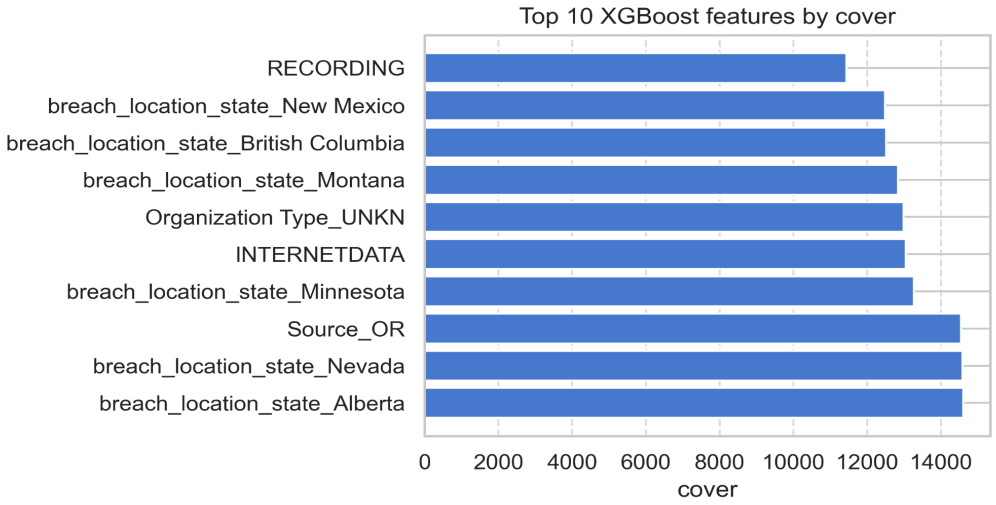
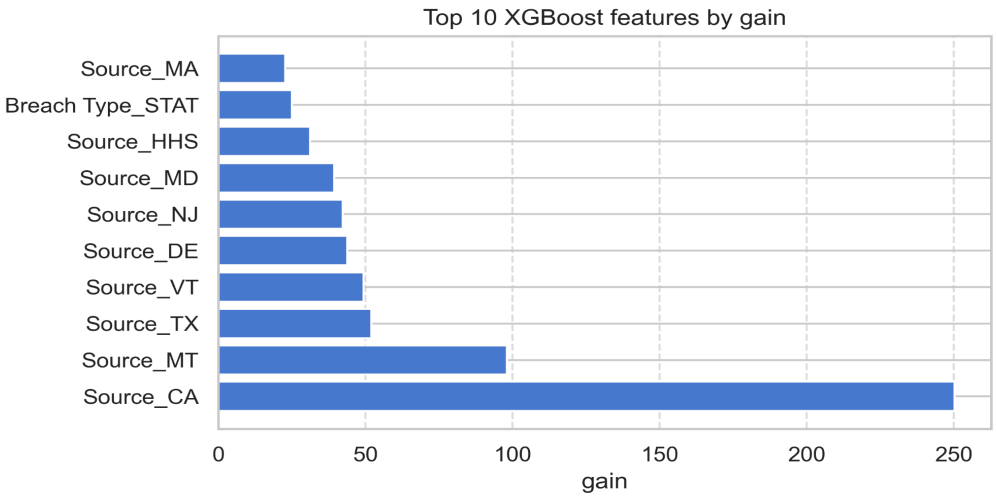
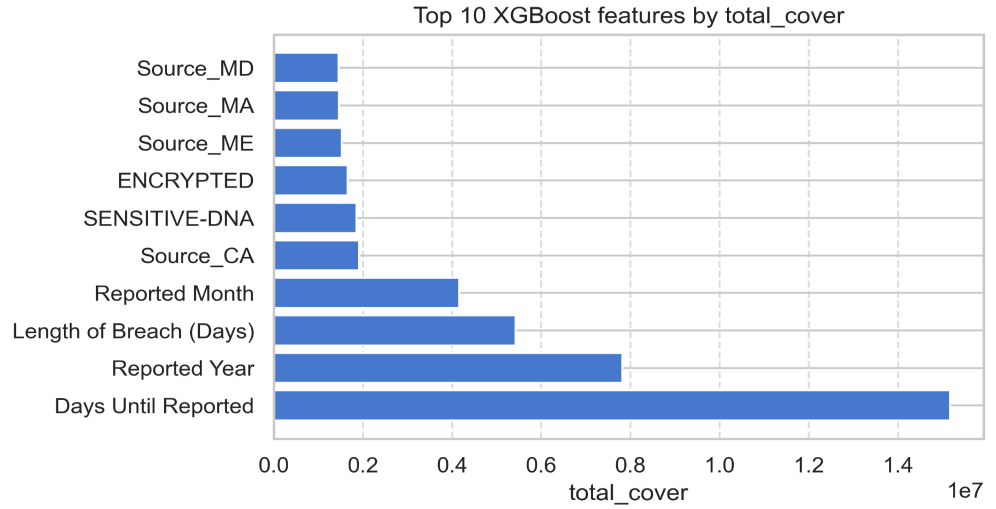


Figure 8: A box plot of the decrease in RMSE when the given features values are shuffled in the test set. Only the Top 10 feature decreases shown.

Figure 9: XGBoost Global Feature Importance Metrics


Top 10 XGBoost features by cover
RECORDING
breach_location_state_New Mexico
breach_location_state_British Columbia
breach_location_state_Montana
Organization Type_UNKN
INTERNETDATA
breach_location_state_Minnesota
Source_OR
breach_location_state_Nevada
breach_location_state_Alberta
0
2000
4000
6000
8000
10000
12000
14000
cover


Top 10 XGBoost features by gain
Source_MA
Breach Type_STAT
Source_HHS
Source_MD
Source_NJ
Source_DE
Source_VT
Source_TX
Source_MT
Source_CA
0
50
100
150
200
250
gain


Top 10 XGBoost features by total_cover
Source_MD
Source_MA
Source_ME
ENCRYPTED
SENSITIVE-DNA
Source_CA
Reported Month
Length of Breach (Days)
Reported Year
Days Until Reported
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e7
total_cover

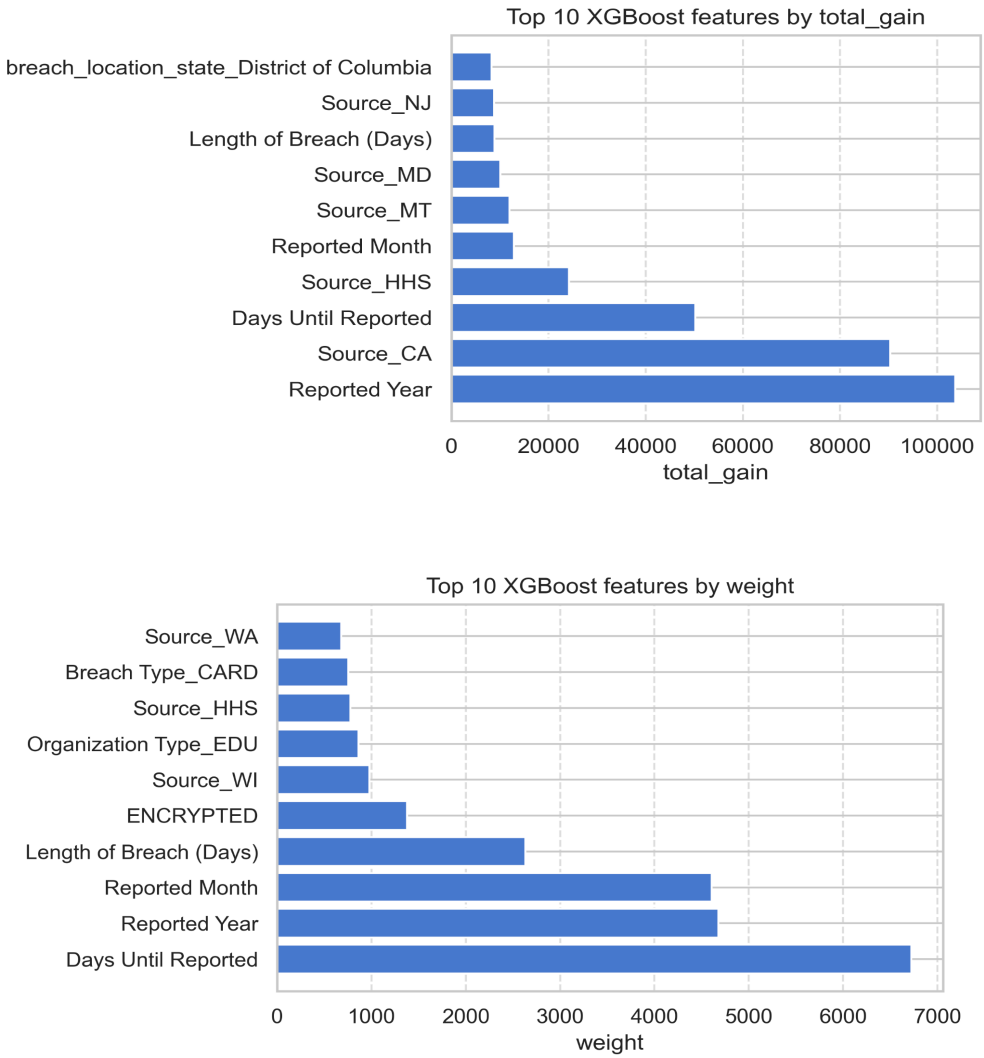


Figure 9: A bar chart for each XGBoost feature importance metric for the Top 10 features.

Local Feature Importance:

This model can also be interpreted per data point using SHAP values, as shown in Figure 10. Here, a small data breach and a medium sized data breach are given to the model. It is interesting that HHS (Department of Health/Human Services), is a strong indicator of a larger breach, as well as if the breach is a hack or if it has occurred in a recent year.

Figure 10: Local Feature Importance with SHAP Values

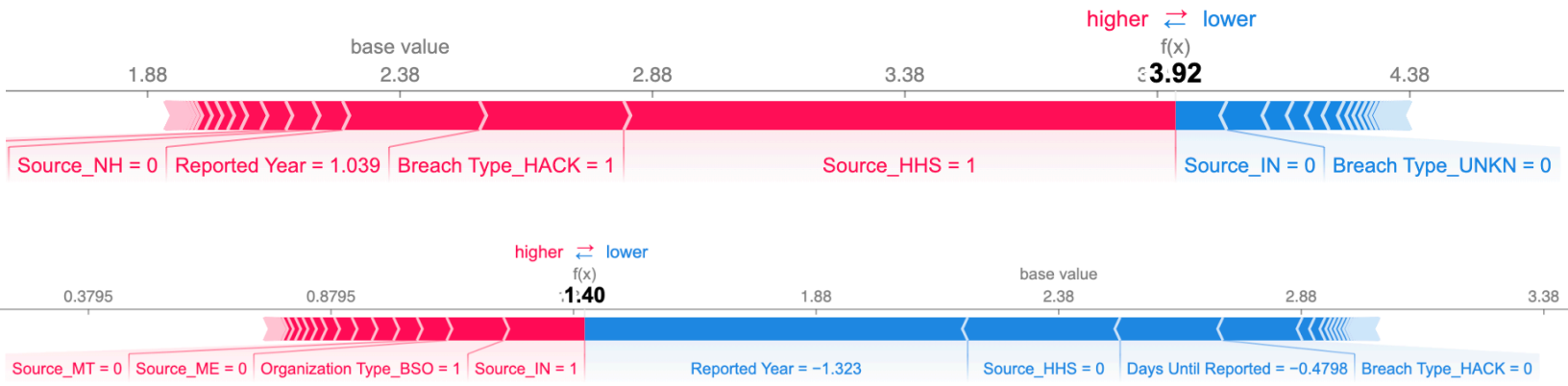


Figure 10: SHAP Force Plots for two data points. The true values for the points are 0.301 and 3.77 respectively.

V: OUTLOOK

An improvement that could be made in the feature engineering process is turning the information type json into a set of numerical features instead of pseudo one hot encoded categorical features. My current model knows if a breach had three types of medical information breached, where it might have made more sense to just know whether there was medical information breached at all. Other ways I could have improved the model is to try out more advanced neural networks, or use external computers to have been able to do even more precise hyperparameter tuning. One aspect of the data I did not use was the name of the organization breached. I could have scraped data from stock exchanges, government databases, or other sources of information to gather more data about the company itself rather than just relying on the breach information.

VI: REFERENCES

[1**]** “Data Breach Chronology.” *Data Breach Chronology | Privacy Rights Clearinghouse*, 2024, privacyrights.org/data-breaches

[2] Pedregosa, Fabian, et al. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, 2011, pp. 2825-2830. *Journal of Machine Learning Research*, http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.