DATA 695: Capstone Project

FINAL PROJECT REPORT

Oil Spill: Image Detection using Deep Learning Techniques and Statistical Analysis with ML Algorithms

Group 5:

Jannatul Naeema (30198919)

Md. Nazmus Sajid (30156854)

Contents

Introduction

Oil spills are catastrophic events that result in severe environmental consequences, posing significant threats to ecosystems, wildlife, and humans. They can result from various sources such as offshore drilling, transportation accidents, or pipeline leaks. Throughout this project we wanted to build a neural network model through deep learning to identify oil spills from geological/satellite images. We also aimed to study the oil spill accident data to find out the best model for future prediction using several machine learning algorithms.

Datasets

Dataset 1: Oil Pipeline Accidents [(1)]

The database contains information on every reported oil pipeline leak or spill from 2010 to 2017 across various states in the United States. This dataset is in tabular format; it is a csv file which contains 48 columns and 2795 rows of data. According to acknowledgement of Kaggle, these oil pipeline accident reports were collected and published by the DOT's Pipeline and Hazardous Materials Safety Administration and available for public use.

Dataset 2: Oil Spill Detection [(2)]

This oil spill detection classification dataset contains labelled images of oil spill at sites in the Niger Delta region of Nigeria. This dataset is a collection of images of oil spill. This spill dataset is already divided into 300 images of test and 112 images of train data. The images are labelled as No Spill and Oil spill.

Part 1: Image Classification using Dataset-2 (Images)

1.1 Image Detection of Oil spill using CNN

Procedure: For Convolutional Neural Network we have used 471 images from where there are 237 oil spill images and 234 no spill images.

1.1.1 Loading Data using Keras Utils

For loading data, we have used the **'image_dataset_from_directory'** function from the **'tf.keras.utils'** module from TensorFlow. Here, we have used the function for the **'Spill_Data'** directory we created in our local machine. Then we have loaded our images in a variable known as **'data'** which configures the directory into two classes (Spill images and No Spill images). Next, the **'data_iterator'** is created by calling the **'as_numpy_iterator'** method on the **'tf.data.Dataset'** object called **'data'**. This iterator allows us to iterate over the dataset and retrieve one batch at a time. Inside the loop, each batch returned by the iterator is unpacked into images and labels. These variables hold the NumPy arrays representing the batch of images and their corresponding labels. You can then perform any necessary operations on these arrays, such as feeding them into a machine-learning model for training or evaluation.

1.1.2 Data Pre-processing

For data pre-processing, we have scaled our data and split it to training, validation and test sets. In the scaling stage, we used the map function to apply a lambda function to each element of the dataset data. In this specific case, the lambda function takes two inputs, denoted as x and y, which represent the input images and their corresponding labels, respectively. The lambda function divides the input images x by 255, effectively performing a normalization operation. Dividing the images by 255 is to scale the pixel values from the original range of 0-255 to a normalized range of 0-1. Again,

we have used the **'as_numpy_iterator'** method to our normalized data to grab our iterator and **next ()** to access another batch.

1.1.3 Building the Convolutional Neural Network Model

We have created a Sequential model in TensorFlow's Keras API for a Convolutional Neural Network (CNN) architecture. First, we have imported the specific layer classes required to create the CNN architecture. Then, we added the 2D convolutional layers with 16 filters of size 3x3, a stride of 1, and 'relu' activation. It also specifies the input shape of the images as (256,256,3). We also added 2D max-pooling layers with a default pool size and stride of 2. These steps are repeated to add two more sets of convolutional and pooling layers with increasing filter sizes (32 and 16) but the same configuration. **'model.add(Flatten())'** flattens the output from the previous layers into a 1D vector, preparing it for the fully connected layers.

Afterwards, we added the fully connected layers one with 256 units and **'relu'** activation and other with 1 unit and **'sigmoid'** activation. Finally, compiled the model with the **'adam'** optimizer, **'BinaryCrossentropy'** loss function, and **'accuracy'** as the evaluation metric. **'model.summary()'** prints a summary of the model's architecture, including the layer types, output shapes, and the total number of trainable parameters. We plotted the loss values for training and validation sets, which show declination of values and proves the model is not overfitted. We have done a similar visualization for measuring accuracy for both training and validation sets, which reaches the maximum of 1 after completion of 20 epochs.

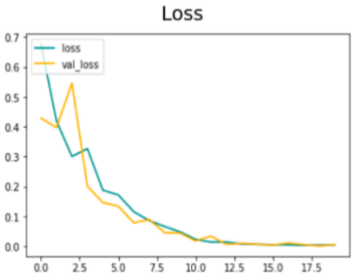


Fig 1: The training and validation loss values over the epochs.

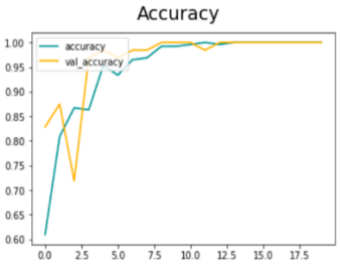


Fig 2: The training and validation accuracy values over the epochs.

1.1.4 Evaluation of the Model

For evaluating our model, we have imported some key metrics from t**ensorflow.keras.metrics** which are precision. recall and binary accuracy. They are used for classification problems. We have tested our model on the test set and got maximum values for all the metrics I.e., 1.0.

Fig 3: Original Image

Fig 4: CNN model predicting no oil spill

Fig 5: Original Image

Fig 6: CNN model predicting oil spill

Part 2: Statistical Analysis with ML Algorithms using Dataset-1

2.1 Exploratory Data Analysis and visualization (plots); Data cleaning and Wrangling

We have completed this section of work, for the data cleaning process, we handled empty cells by replacing them with NaN values. Additionally, we split the Accident date/time column into separate year, month, and day columns, enabling us to analyze the data more effectively. We also eliminated a few unnecessary columns, such as Accident Date/Time, Accident Year, Supplemental Number, and Report Number, to streamline the dataset and focus on the most relevant information.

We also focused on analyzing various aspects of the dataset. We created bar plots to visualize the trends of Net loss, Injuries, Fatalities, and Costs over the years, months and days separately for an 8-year period. We have also observed the quantity of Intentional and Unintentional Liquid Release in barrels over the years along with Liquid recovery and Net Loss in barrels. The generated bar charts are presented in Appendix A Fig A.1.

2.2 Principal Component Analysis (PCA)

PCA is unsupervised dimensionality reduction technique and doesn't rely on any labels. It transforms a dataset with possibly correlated features into a new set of uncorrelated features called principal components. We aimed to reduce dimensionality to 5, but the results showed that only 27.7% of the variables were explained by PC1, and 13.87% were explained by PC2, 12.6% by PC3, 9.3% by PC4 and 7.9% by PC5, total of 71.36% of the variables were explained by the five principal components. Since lots of data are lost with this procedure, so we decided to implement supervised machine learning techniques.

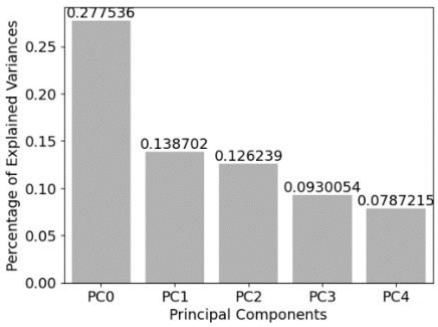


Fig 7: Principal components and percentage of explained variances

2.3 Classification

The goal of classification is to assign input data to a specific category or class. The output is discrete and represents a class label. For example, classifying liquid explosion has two class: YES or NO; so, either there is a liquid explosion or not. Similarly, Liquid ignition is also classified as YES or NO, same goes for Pipeline Shutdown. In our dataset for all three-input data can be categorized as binary output. For this classification we have three target columns Liquid Ignition, liquid explosion and Pipeline Shutdown. The rest 15 columns are feature columns; we also encoded the feature columns

which had categorical values into numerical value. We splitted the data into training and test dataset in 80/20 ratio.

- **2.3.1 Logistic Regression**
- **2.3.2 Decision Trees for Classification**
- **2.3.3 Random Forests**
- **2.3.4 Naive Bayes**

2.3.1 Logistic Regression

For logistic regression we have taken Liquid Explosion as our target variable because it can be one of the major reasons that can cause oil spill. For training our model we have 8 categorical variables and 6 numeric variables. For converting the categorical variables into numeric variables, we have used Label Encoder. Here for no explosion, it is encoded as 0 and for liquid explosion it is encoded as 1. After that we plotted a correlation heatmap for all the variables that are used in the model. From the heatmap we can say apart from few variables the correlation between all the variables is insignificant. This dismantles the multicollinearity between the independent variables.

2.3.1.1 Building the Logistic Regression Model:

We have separated the target variable and divided the dataset into training and test sets. The size of the test is 20% of the entire dataset. For maximizing the performance of the model, we have used Grid Search CV and defined the hyperparameter grid. The grid specifies different values for the hyperparameters '**penalty', 'C',** and '**solver'** to be used in the logistic regression model. 'penalty' specifies the type of regularization to be applied to the logistic regression model. The options we have provided are 'l1' (Lasso regularization) and 'l2' (Ridge regularization). **'C'** hyperparameter is the inverse of the regularization strength. Smaller values of C result in stronger regularization, while larger values result in weaker regularization. We have specified a list of possible values [0.001, 0.01, 0.1, 1, 10, 100] for 'C'. The '**solver'** hyperparameter specifies the algorithm to use for optimization. The options we have provided are 'liblinear', 'lbfgs', and 'saga'. With this hyperparameter grid, Grid Search CV will systematically explore different combinations of these hyperparameters to find the best combination that optimizes the performance of the logistic regression model. We then used this param grid in conjunction with Grid Search Cross Validation from scikit-learn to perform the grid search. For our model, the best hyperparameters are 'C': 10, 'penalty': 'l1' and 'solver': 'liblinear' respectively.

2.3.1.2 Evaluation of the Model:

For evaluating our model, we have tested the model on our test set and got around 99.6% accuracy. We have also created a confusion matrix where the value of True positive is 555 and in test set the value is 556 that means the model predicted just one value inaccurately.

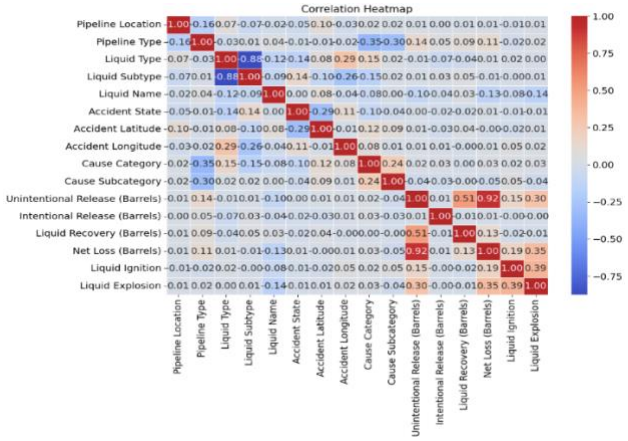


Fig 8: Correlation Heatmap for Logistic Regression

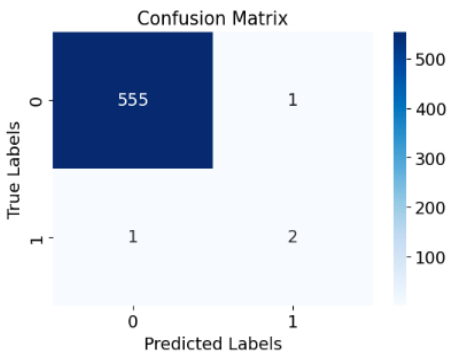


Fig 9: Confusion Matrix for Logistic Regression

2.3.2 Decision Trees for Classification

Decision trees for classification is a supervised machine learning algorithm used for solving classification problems. From scikitlearn **sklearn.tree** we imported **DecisionTreeClassifier,** then we created an instance for classifier and started with default hyperparameters. Then we did hyperparameter tuning with **GridSearchCV** to find the best combination of hyperparameters. We have noted the best hyperparameters in the table below. Next, we trained the classifier instance using these best parameters and evaluated its performance on the test dataset. We used accuracy score for evaluation.

2.3.3 Random Forests Classifier

Random Forest classifier use multiple decision tree to improve classification performances. After pre-processing of data, train-test split and target-feature separation we imported an **RandomForestClassifier** instance with default hyperparameter. We used **GridSearchCV** to find best combinations of hyperparameters for optimal performance, trained the classifier and tested on test dataset to evaluate its performance for three target variables.

2.3.4 Naive Bayes Classifier

Naive Bayes is a simple and effective probabilistic classification algorithm which is based on Bayes theorem, and it works well with substantial number of features. The assumption of naïve bayes is all features are independent which was not true in our dataset, as it is almost impossible for real world dataset to get features independent of each other. We did hyperparameter tuning and training and testing using this classifier, but we know that this classifier might not be applicable for our dataset because of the assumption.

Comparison of Different Classification Models

Accuracy: Accuracy refers to the measure of how effectively a classification model correctly predicts the category of data instances. It quantifies the proportion of correctly predicted instances out of the total instances in the dataset. The comparison of accuracy values are presented and discussed in Result and Discussion section.

2.4 Features Selection

We also implemented decision trees to find the most important features for Liquid Explosion, Liquid Ignition and Pipeline Shutdown. The results are shown in the results and discussion section.

2.5 Regression Models

Regression is a statistical and supervised machine learning technique used for modelling the relationship between a dependent variable (target variable) and few independent variables (feature variables). In our dataset we have three target variables and seventeen independent variables. Our target variables are Unintentional Release, Liquid Recovery and All Costs due to Accidents. Here Unintentional Release represents the spill quantity in barrels; Liquid Recovery represents the amount in barrels recovered after the spill and All Costs indicate the total cost incurred due to different damages, fatality, or injuries.

In this section, we implemented five regression models to find the best model to predict the target variables with minimum error. As the first step we moved the target variables to the last of data frame, then using label encoder we encoded the categorical variables to some numbers. Then we did the split of the dataset into training and test set in 80/20 ratio.

These are the regression models that we applied on our dataset:

2.5.1 Linear Regression
2.5.2 KNN
2.5.3 Random Forest
2.5.4 XGBoost
2.5.5 AdaBoost

2.5.1 Linear Regression

Our first step was to perform linear regression, but we wanted to check the pre-requirement; and we checked for Multicollinearity, Linearity and Normality; and there were some multicollinearities in our data, also this dataset failed for both linearity and normality check. So decided to move on to perform next regression model.

2.5.2 KNN

We used sklearn.neighbors from scikitlearn for the KNN regression and we created an instance of the KNeighborsRegressor class and choose number of neighbours K= 5, as a larger K value is important to get smoother predictions. Then we fit it to the training data. Then we used the trained model to make predictions on the test data set and calculated the evaluation metrics Mean Squared Error (MSE) to assess the model's performance.

2.5.3 Random Forest

We used sklearn.ensemble from scikitlearn for the Random Forest regression and we created an instance of the RandomForestRegressor class where we specified number of trees in the forest as n_estimators = 10. Then similarly we fit it to the training data. Then we used the trained model to make predictions on the test data set to calculate Mean Squared Error (MSE).

2.5.4 XGBoost

XGBoost is a powerful gradient boosting algorithm used for regression; for this part we used xgboost library; also performed hyperparameter tuning for that imported GridSearchCV from scikitlearn. We defined parameter grid to search for the best parameters and then fit the grid search object on the training data. Then we performed hyperparameter tuning to find the best model and the best model parameters are: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 150}. Then as other regression models we made predictions on the testing set and calculated MSE.

2.5.5 AdaBoost

AdaBoost is Adaptive Boosting; an ensemble learning algorithm that combines multiple weak learners (usually decision trees) to create a strong learner. We used sklearn.ensemble from scikitlearn to import adaboost regressor, created an instance of the AdaBoostRegressor class. We also performed hyperparameter tuning with GridSearchCV from scikitlearn. We also performed hyperparameter tuning to find the best model. The best model parameters are: {base_estimator= DecisionTreeRegressor (max_depth=3), learning_rate=0.1, random_state= 42} for unintentional release and learning_rate=0.2 and 0.01 for Liquid Recovery and All cost respectively while other parameters are exact same. In next step we made predictions on the testing set with best model and calculated MSE.
The best Hyperparameters for three different Target Variables with Different Classifications methods are displayed in Appendix A Table A.1.

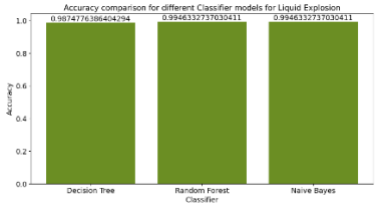
Comparison of Different Regression Models

Mean Squared Error: Mean Squared Error (MSE) is used to measure the average squared differences between the predicted values and the actual values in a regression problem. The lower values of MSE indicate that the model's predictions are closer to the actual values, while higher values indicate larger discrepancies between predictions and actual data. The comparison of MSE values are presented and discussed in Result and Discussion section.

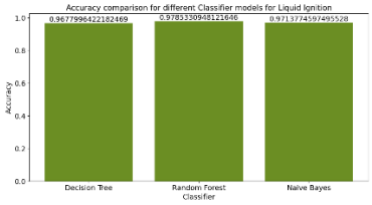
Results and Discussion

For image recognition through CNN, the batch size is 32 I.e., each batch contains 32 images. The height and width of each image is 256 pixels. The color channels for each image are 3 (e.g., red, green, and blue channels for RGB images). In data pre-processing, we split our data into 11 batches. Each batch contains 32 images. For training the CNN model we have allocated 8 batches, for validation set we have taken 2 batches and test set consists of 1 batch. We trained our model using epochs = 20 specifies the number of times the entire training dataset will be passed through the model during training. In this case, the model will be trained for 20 epochs, meaning it will go through the training dataset 20 times. Here we have also used the 2 batches of validation set which is used to evaluate the model's performance and monitor its generalization during training. The validation set contains input images and their corresponding labels that are separate from the training set. For testing the accuracy of our model, we have tested our model on some random images from the internet and our model was predicting the images efficiently for oil spills and non-spill of oil in different waterbodies.

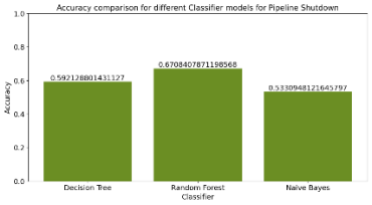
In the next part, with Dataset-1 we explored the details of spill-related incidents, specifically focusing on three significant outcomes: Liquid Explosion, Liquid Ignition and Pipeline Shutdown. These outcomes are immediate results of spills, each have two categories: YES or NO. We used four distinct classifiers for Liquid Explosion: Logistic Regression, Decision Tree, Random Forest Classifier, and Naive Bayes. We implemented the last three classifiers for Liquid Ignition and Pipeline Shutdown. We employed cleaned and modified Dataset-1 and implemented these classifier algorithms. Finally, we found that the Random Forest Classifier displayed remarkable performance across all three outcomes. Specifically, it achieved an accuracy rate of 97.85% for Liquid Explosion, 99.46% for Liquid Ignition and 67% for Pipeline Shutdown. This outcome emphasizes that the Random Forest Classifier is the most effective in classifying instances across these three spill-related outcomes.



To classify the category of Liquid Explosion, Random Forest Classifier has the highest accuracy

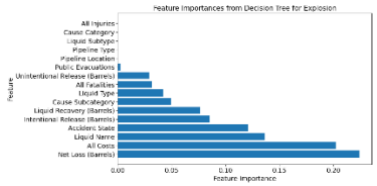


For Liquid Ignition, Random Forest Classifier has the highest accuracy

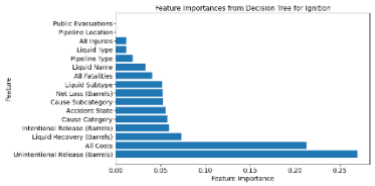


For the Pipeline Shutdown classification, Random Forest Classifier has the highest accuracy

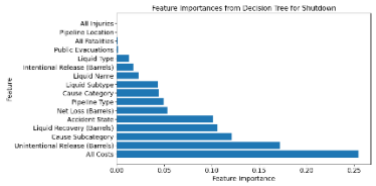
In our next step we figured out the most important features to classify the three target variables are as below:



For liquid explosion the most important features are Net Loss of liquid (Barrels) and All cost incurred due to spill

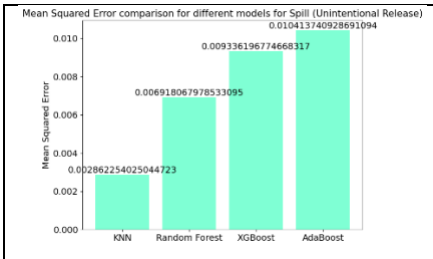
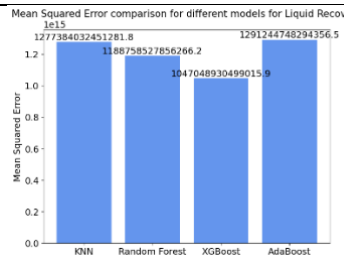
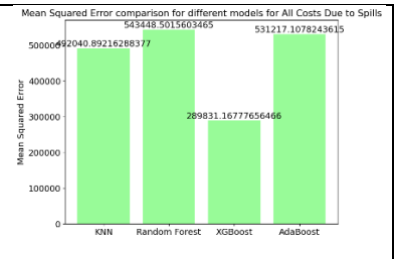


For liquid ignition the most important features are Unintentional Release of liquid (Barrels) and All cost incurred due to spill



For Pipeline Shutdown the most important features are Unintentional Release of liquid (Barrels) and All cost incurred due to spill

Next with Dataset-1, we selected three distinct target variables alongside seventeen independent variables. These target variables are Unintentional Release, Liquid Recovery and All Costs due to Accidents. As mentioned above, Unintentional Release denotes the quantity of spillage in barrels; Liquid Recovery signifies the liquid volume in barrels successfully recovered after the spill; and All Costs indicates the comprehensive expenses incurred because of damages, fatalities, or injuries occurred due to the spill. While Unintentional Release refers to the spill quantity, both Liquid Recovery and All Costs remain relevant to the spill scenario. In this part, we implemented four distinct regression models (KNN, Random Forest, XGBoost, AdaBoost), aimed at identifying the most optimal model capable of predicting the target variables with the least possible error. We considered applying linear regression but the requirement for its successful implementation were not fulfilled, that is why we moved on with the four regression models. For the prediction of Unintentional Release or spill quantities, the K-Nearest Neighbors (KNN) model provided best result compared to other models, yielding predictions with minimal error in spill estimation. In the prediction of Liquid Recovery, the XGBoost regression model exhibited the lowest prediction error in comparison to other regression models. This makes XGBoost the most accurate predictor for estimating the amount of liquid successfully recovered post-spill. Lastly, when forecasting "All Costs" attributed to spills, the XGBoost regression model was the most effective method for predicting the comprehensive costs incurred due to spill-related damages, fatalities, and injuries.

		
For Unintentional Release or Spill KNN performed better than other models to predict any spill with minimum error	In predicting Liquid Recovery XGBoost performed best with lowest error than any other regression models	For predicting All costs due to spill XGBoost performed better than other models

References

1. *Oil pipeline accidents, 2010-present*. (n.d.). Retrieved May 05, 2023, from https://www.kaggle.com/datasets/usdot/pipeline-accidents

2. *Spill_data*. (n.d.). Retrieved May 03, 2023, from https://www.kaggle.com/datasets/damingo1/spill-data

Appendix A

Exploratory Data Analysis and visualization in section 2.1 (plots) with Dataset-1

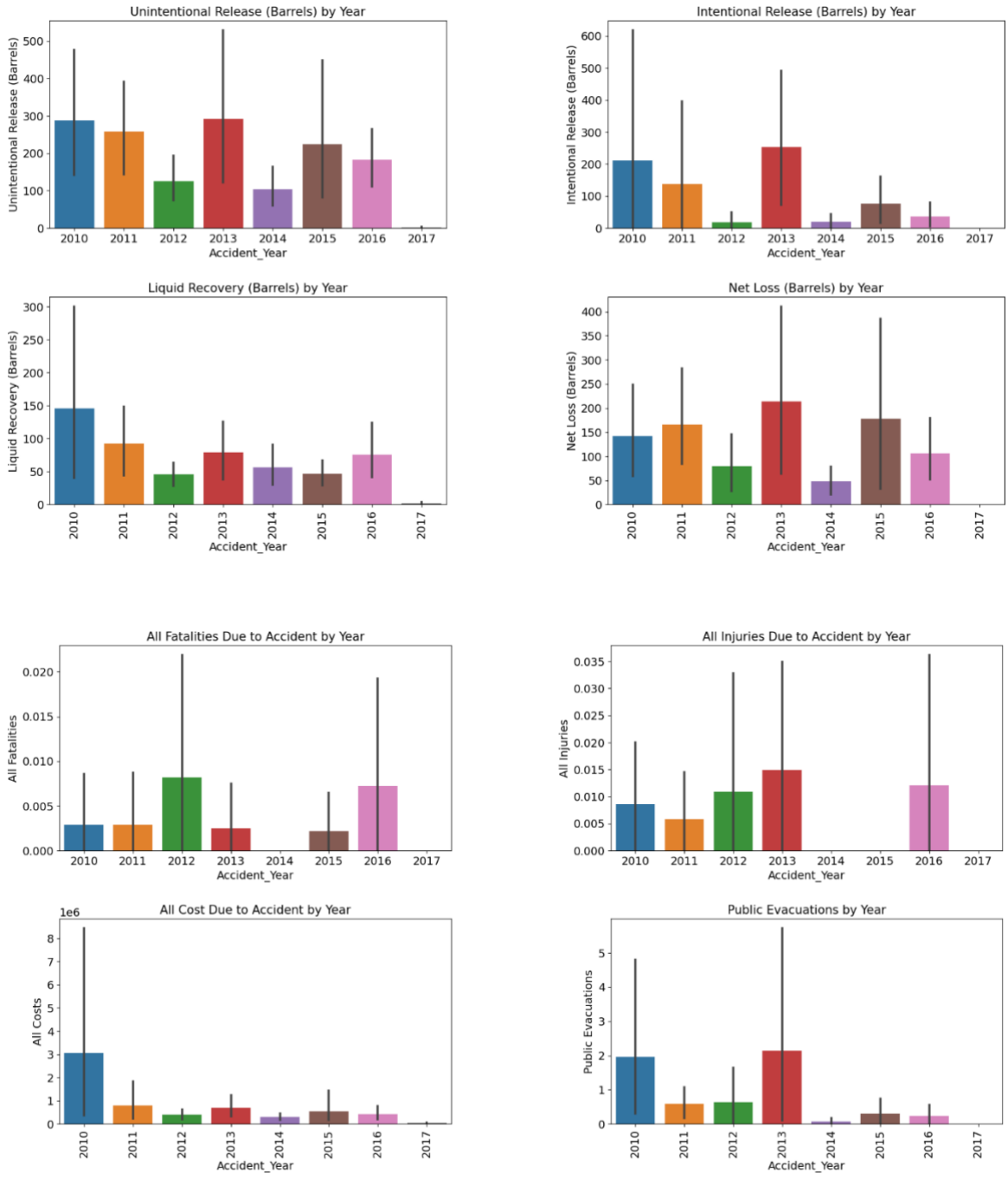


Fig A.1: Visualization of Different Features over the Years (2010-2017)

Table A.1: Best Hyperparameters for three different Target Variables with Different Classifications shown in Section 2.3

	Best Hyperparameters for three different Target Variables with Different Classifications		
	Ignition	**Liquid Explosion**	**Pipeline Shutdown**
Decision Trees	'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10	'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2	'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10
Random Forests Classifier	'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100	'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100	'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200
Naive Bayes Classifier	'var_smoothing': 1e-05	'var_smoothing': 1e-05	'var_smoothing': 1e-08