**Group Project**

Wong, Yu Kwan

Ngan, Tsang Kwong

Kirstin Megga Ramos

Pillai, Rani Krishna

Ailin Zhang

24W --Supervised Learning (SEC. 001)

Merlin James

April 08, 2024

[Executive summary 1](#_Toc289553357)

[Introduction 2](#_Toc1329928028)

[Data exploration 3](#_Toc1664394346)

[Data modelling 5](#_Toc636360746)

[Predictive model building 7](#_Toc2065345340)

[Naive Bayes 7](#_Toc1393390056)

[Logistic regression 9](#_Toc1193301478)

[SVM 10](#_Toc2049315410)

[Voting classifier 12](#_Toc734182643)

[Random forest 13](#_Toc294675711)

[Deploying the model 13](#_Toc1439439710)

[Key findings 13](#_Toc1066790079)

[Conclusion 13](#_Toc419834834)

# Executive summary

In pursuit of enhancing road safety and reducing fatalities, this project developed a predictive AI solution leveraging machine learning and API technology. The dataset utilized spans a six-year period from the Toronto police department, providing valuable insights into traffic collisions. The project is divided into several key sections:

1. Data Exploration: Section 1 involves comprehensive data exploration, encompassing data loading, statistical analysis, assessment of missing data, and graph visualization.
2. Data Modeling: Section 2 focuses on data modeling, including transformations, feature selection, data splitting, and preprocessing using pipeline classes.
3. Model Implementation: Section 3 explores the application of various machine learning algorithms and classifiers, such as logistic regression, SVM, Naive Bayes, voting classifier, and random forest. Techniques like grid search and randomized grid search are employed for model fine-tuning.
4. API Development: Section 4 entails transforming the selected machine learning model into an analytics API using the Flask framework and the Pickle module for deployment on a local host.
5. Key Findings: Section 5 highlights the key findings from the project, including model scores, accuracies, and recommendations based on a comparative analysis of different models.
6. Conclusion: Section 6 presents the conclusion drawn from the project, summarizing the effectiveness of the developed predictive solution and its potential impact on enhancing road safety and reducing fatalities.

Overall, this project represents a significant step towards leveraging data-driven insights to address critical challenges in road safety and lays the foundation for future advancements in predictive analytics and AI-driven solutions for traffic management.

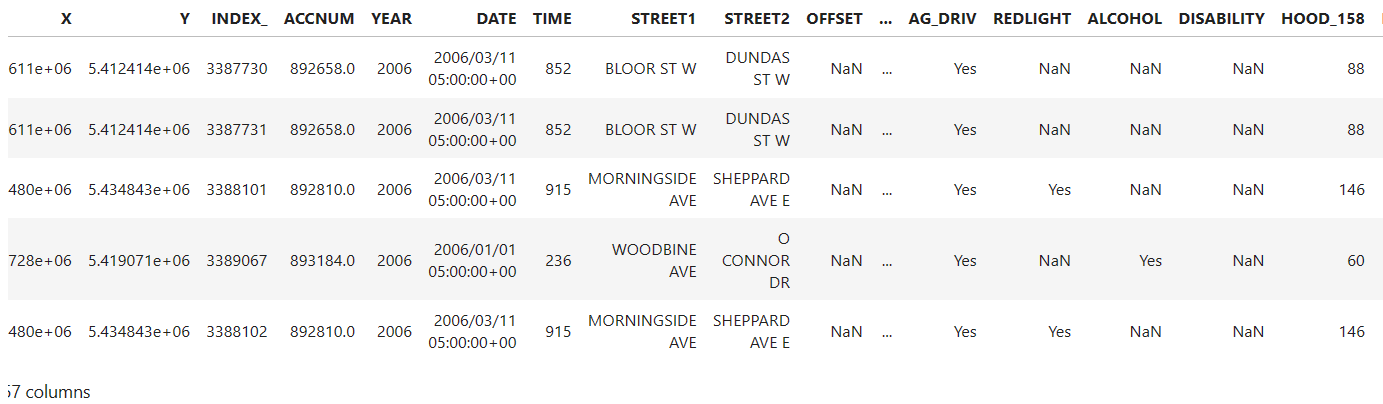
# Introduction

With the aim of enhancing road safety and reducing fatalities, both the police department and the individuals can benefit from a predictive AI solution by assessing the likelihood of fatal collisions. For police departments, such a tool aids in implementing effective security measures and planning road conditions in high-risk areas. Similarly, for individuals, a predictive tool allows them to evaluate the need for additional precautions based on specific factors like time, weather conditions, and neighborhood.

This project developed a predictive software by using machine learning and API technology. The dataset utilized in the project is from the Toronto police department over a six-year period. Section 1 in the project report involves data exploration, including data loading, statistical analysis, assessment of missing data, and graph visualization. In Section 2, the focus shifts to data modeling, transformations, feature selection, data splitting and preprocessing using pipeline class. Section 3 delves into the application of various algorithms and classifiers, such as logistic regression, SVM, Naive Bayes, voting classifier, and random forest, to build predictive models. Furthermore, grid search and randomized grid search techniques are employed for model fine-tuning. Following a comparative analysis of model scores and accuracies, the report recommends the most effective model. Subsequently, the selected machine learning model is transformed into an analytics API using the Flask framework and the Pickle module for deployment on a local host in Section 4. The key findings are illustrated in Section 5. Section 6 the conclusion was be drawn.

# Data exploration

In this section, our primary focus was on exploring the dataset thoroughly, aiming to gain comprehensive insights into its structure and characteristics. We initiated by loading the data and meticulously describing each data element. The first 5 row is:



This dataset comprises 18,194 entries and 57 columns. The target column, ACCLASS, needs to be transformed into binary values, with "Fatal" mapped to 1, and "Non-Fatal Injury" and "Property Damage Only" mapped to 0. Additionally, 5 NaN values from this column are considered for removal.

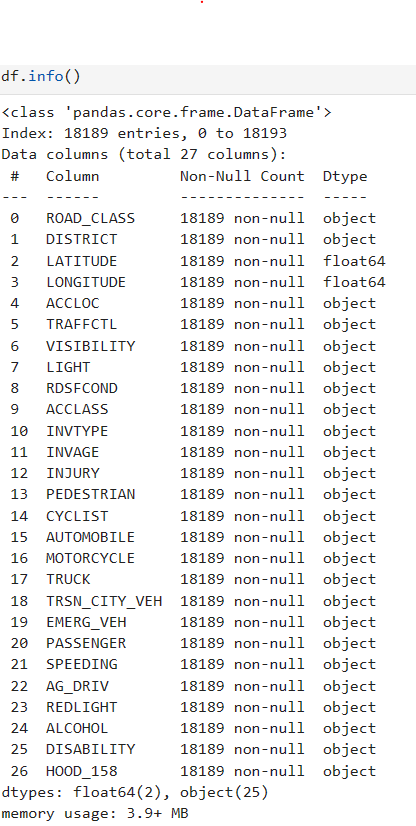
Upon closer examination of the dataset, it was determined that the following columns are unnecessary for analysis: ObjectId, HEIGHBOURHOOD\_158, HEIGHBOURHOOD\_140, CYCLISTYPE (due to excessive categories and NaN values), PEDCOND (due to excessive categories and NaN values), PEDACT (due to excessive categories and NaN values), PEDTYPE (due to excessive categories and NaN values), DRICOND (as it includes "other," signifying a lack of accuracy), DRIVACT (as it includes "other," signifying a lack of accuracy), MANOEUVER (as it includes "other," signifying a lack of accuracy), FATAL\_NO, INVTYPE, DATE, YEAR, ACCNUM, INDEX\_, STREET1, STREET2, OFFSET, X, Y, and INJURY. The clean code for the column determines is shown in below.



Dealing with columns which contain many categories, we adopted some strategies in the dataset for better analysis.

1. Simplification and mapping: The LIGHT column was categorized into Daylight, Dark, Dusk, Dawn, and Other. Similarly, the INVAGE column was grouped into age ranges: 0 to 20, 20 to 40, 40 to 60, 60 to 80, and over 80. The RDSFCOND column was condensed into Dry, Wet, Snow, Ice, and Other. Additionally, we simplified the ROAD\_CLASS column to Major Arterial, Minor Arterial, Collector, Local, and Other. The TRAFFCTL column was categorized as No Control, Traffic Signal, Stop Sign, and Other. Furthermore, the ACCLOC column was categorized into At Intersection, Non-Intersection, and Other. Lastly, the VISIBILITY column was simplified to Clear, Rain, Snow, and Other, while the INVTYPE column was categorized as Driver, Pedestrian, Passenger, Vehicle Owner, Cyclist, and Other.
2. Filling the missing values: In the RDSFCOND cloumn, we also filled the missing values with other. In DISTRICT column, we filled them with the most frequent value.
3. Dropping: we dropped some columns. Redlight, Speeding, Ag\_Driv, Alcohol, Disability are representing DRIVACT column, so we decided to drop it. INITDIR column has 5051 missing values, we also dropped it.

Finaly, we got a clean data frame, which includes 27 columns and 18189 entries.



Through this comprehensive exploration, we gained a clear view about the dataset for modeling and analysis tasks.

# Data modelling

In the data modeling section, we focused on several key aspects to prepare our dataset for machine learning algorithms. Firstly, we performed preprocessing for categorical data by using oneHotencoder from scikit-learn, which is able to encode categorical data into a binary format.

Subsequently, we bundled the preprocessing for numerical and categorical data using the ColumnTransformer class. Hence, we can apply different transformations to different columns in our dataset efficiently. For categorical data, we utilized the cat\_transformer pipeline defined earlier.

After transforming the data, we resampled it to address any class imbalance issues using the Synthetic Minority Over-sampling Technique (SMOTE). This technique generates synthetic samples for the minority class to balance the class distribution in the dataset. We instantiated the SMOTE object with a specified random seed for reproducibility and applied it to both the transformed features (X\_transformed) and the target variable (y).

Overall, this preprocessing pipeline ensures that our categorical data is appropriately encoded for machine learning algorithms, and any class imbalances are addressed to prevent biased model training.

The code in this section is shown below.



# Predictive model building

## Naive Bayes

One of the classifiers we used to train our model is Naive Bayes. With its probabilistic nature, this algorithm makes it well suited for computing probabilities for classification decisions and in our case, determining whether an accident is fatal or not. Another thing is that Naïve Bayes is simple and efficient. It can handle large datasets which is what our KSI dataset is. Our dataset involves many categorical variables and Naïve Bayes is particularly effective with this kind of dataset. Finally, it provides a good baseline model and can be particularly beneficial in the initial phases of our project to establish performance benchmarks.

Below are several reasons why we have chosen Gaussian Naïve Bayes over Multinomial Naïve Bayes to train our model:

* Data type: Gaussian is more suitable for datasets where features are continuous and assumed to follow a normal distribution. This is ideal for our dataset since it has coordinates that take a wide range of values.
* Negative values: Multinomial Naïve Bayes is typically used when dealing with count data or features that represent counts or frequencies. In our case, our dataset has negative numerical values such as transformed features and error will occur if Multinomial is used for this instance.

A screen shot of a computer code

Description automatically generated

Figure 1: Error encountered using MultinomialNB

* Gaussian’s flexibility: Since our chosen algorithm does not restrict the sign of the feature values, it allows to work with any form of numerical data whether they’ve been standardized or scaled. This flexibility makes it more suitable for datasets that include a mix of positive and negative feature values.

Evaluation of the model scoring and improvement of performance accuracy

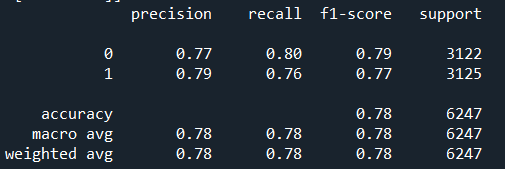
* If maximizing overall accuracy is the main goal: Model with SMOTE, K-Fold, and Adjusted Threshold offers the highest accuracy (0.906). This model might be the best choice if the overall rate of correct predictions is most important, especially in scenarios where each correct prediction (both true positives and true negatives) is valuable.
* If minimizing false positives is crucial (maximizing precision): Again, the Model with SMOTE, K-Fold, and Adjusted Threshold is superior, achieving a precision of 0.904. This model is suitable when the cost of a false positive is high, such as in spam detection or medical testing where false alarms are costly or risky.
* If catching as many positive cases as possible is vital (maximizing recall): The Model with SMOTE (without adjusted threshold) has better recall (0.724) compared to the model with the adjusted threshold (0.685). Choose this model if it is crucial to identify as many true positives as possible, even if it results in some false positives. This scenario is common in fraud detection or disease screening where missing a positive case can have severe consequences.
* If seeking a balance of precision and recall (high F1 Score): The Model with SMOTE, K-Fold, and Adjusted Threshold again performs best in terms of F1 Score (0.740), which balances precision and recall. This model would be appropriate for situations where you need a good balance between catching as many positives as possible and maintaining high accuracy in your positive predictions.

Conclusion on Naïve Bayes model:

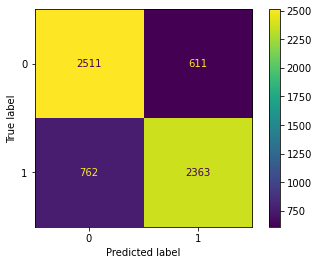
Given the metrics, the Model with SMOTE, K-Fold, and Adjusted Threshold appears to be the best overall, providing a robust balance of accuracy, precision, and F1 Score. It optimizes performance across various aspects, making it versatile for diverse application needs unless specific circumstances dictate a higher priority for recall over other metrics.

## Logistic regression

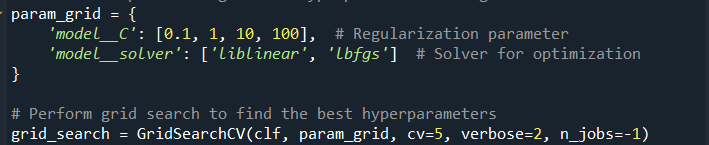
Logistic regression is used to solve this prediction issue. In the begining, the solver is set as ‘lbfgs’ and max\_iter equals to 1000. The training and test accuracy are 0.790 and 0.780, respectively. The other performance is shown in the below figure.

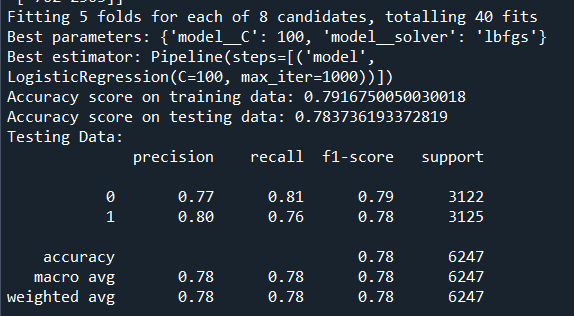


And the confusion matrix is:



Secondly, the parameter grid is set for hyperparameter tuning by grid search.

After total 40 fits, the best parameter, best estimator and accuracy score on both of training data and test data are summarized:



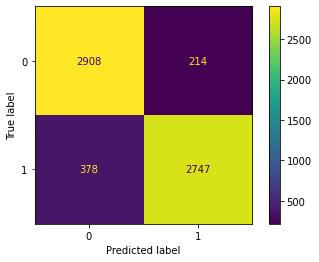
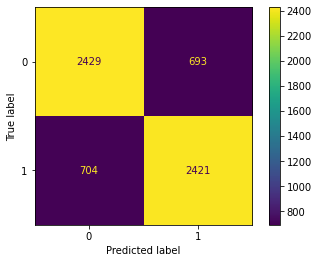
The accuracy scores are both 0.791 and 0.783, respectively. They show a slight improvement compared to the initial attempt

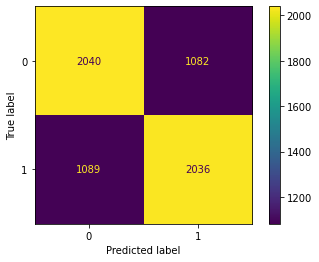
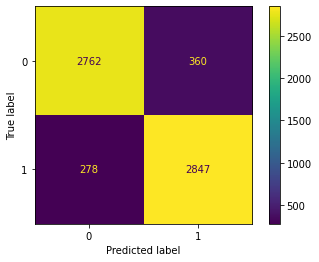
## SVM

In this subsection, firstly we employed an **SVM classifier** using four distinct kernels: **linear, rbf, poly, and sigmoid**. The parameter C was specifically set to 0.1 for the linear kernel, while default settings were applied to the other three kernels. Accuracy scores for both the training and test sets are summarized in Table 1 below. Additionally, Figure 1 illustrates the confusion matrix corresponding to each kernel. Notably, the rbf kernel achieved accuracy scores of 0.917 and 0.905 on the training and test sets, respectively. Similarly, the poly kernel exhibited accuracies of 0.911 and 0.898 on the training and test sets. Comparatively, the linear kernel yielded accuracies of 0.787 and 0.776, while the sigmoid kernel attained scores of 0.662 and 0.652 on the training and test sets, respectively. Based on these findings, it is evident that the **rbf and poly** kernels demonstrated superior performance in terms of accuracy.

Table 1 Comparison of accuracy on training set and test set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kernel | Linear | rbf | poly | sigmoid |
| Accuracy on training set | 0.787 | 0.917 | 0.911 | 0.662 |
| Accuracy on testing set | 0.776 | 0.905 | 0.898 | 0.652 |

 (a) (b)

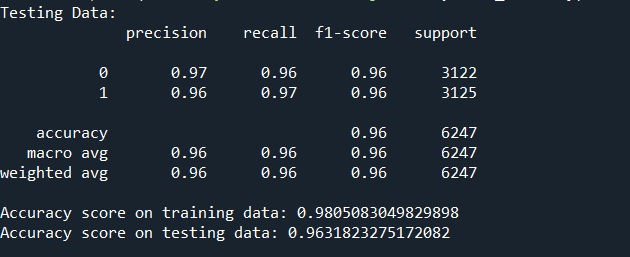


(c) (d)

Figure 1: confusion matrix: (a) linear, (b) rbf, (c) poly, (d) sigmoid

After fine-tuning the SVM models using GridSearchCV to determine the optimal parameters and best estimator, the process involved a trade-off between computational resources and model performance. The parameter grid was defined as follows: 'classifier\_\_kernel': ['poly', 'rbf'], 'classifier\_\_C': [0.1, 1, 10, 100], 'classifier\_\_gamma': [0.01, 0.1, 1.0].

A total of 120 fits were conducted, with 5-fold cross-validation for each of the 24 candidate parameter combinations. The best parameters identified were **{'classifier\_\_C': 10, 'classifier\_\_gamma': 1.0, 'classifier\_\_kernel': 'rbf'}**, and the corresponding best estimator was a Pipeline containing an SVM classifier with these parameters. The accuracy scores obtained on both the training and testing data were noteworthy, with an accuracy score of **0.981** on the training data and **0.963** on the testing data.

 This indicates that the model generalizes well to unseen data, demonstrating strong predictive performance.

## **Voting classifier**

We chose a voting classifier which includes hard voting and soft voting, to investigate the prediction in this subsection. The performance features, such as accuracy, precision, recall, F1-score, confusion matrix and ROC are summarized. Firstly, the performance features of hard voting are as follows:

A screenshot of a computer screen

Description automatically generated  
A graph of blue squares with numbers and labels

Description automatically generatedA graph with a line and a point

Description automatically generated  
  
Secondly, the performance features of soft voting are shown in the below figures.A screenshot of a computer

Description automatically generated  
A blue squares with white text

Description automatically generatedA graph with a line and a line

Description automatically generated

## **Random forest**

In this subsection, **Random Forest Classifier** was utilized to perform the prediction by four attempts.

**Attempt 1**: Firstly, the initial parameters are shown as follows:

A computer screen shot of a program code

Description automatically generated

In the first attempt of the model training, the min\_samples\_leaf was fixed as 1 only, and below is the best model performance:

A screenshot of a computer program

Description automatically generated  
A computer screen shot of a black screen

Description automatically generated  
The best model achieved an accuracy of over 95% on the test dataset while the one on the training dataset is 98%. These performances on accuracy suggest this model have strong predictive capabilities. However, further analysis revealed a variance between the performance on the training and cross-validation sets when plotting the learning curve. This discrepancy indicates a potential issue of overfitting, where the model may have learned to fit the training data too closely, resulting in reduced generalization ability to unseen data.

A screen shot of a computer

Description automatically generatedA graph of a graph showing the number of training scores

Description automatically generated with medium confidence

The tree plot will be very huge with min\_sample\_leaf:

  
It almost expends the result for each case, which means that if we try to use some brand-new data to perform the prediction, the accuracy might not be ideal.

**Attempt 2:** Secondly, the parameters below are set and the min\_sample\_leaf is at least 2.

A computer screen with text and images

Description automatically generated  
After building the model, the best parameters and the accuracy are obtained:

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A graph of a graph showing the results of a training

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated  
Since the max\_depth parameter increases, the decision tree becomes more complex. However, the training data may decrease with the increasing max\_depth parameter. This complexity may lead to overfitting. Hence, it is crucial to select the appropriate max\_depth value to strike a balance between model complexity and generalization. The ideal max\_depth value is the one that optimizes the trade-off between training error reduction and test data accuracy, ensuring the decision tree fits the data well without overfitting.

**Attempt 3:** From the third attempt, we tried to minimize max depth to 20 and refit the model.

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen with text

Description automatically generated



Attempt 4: Finally, max\_depth is set to 4, 6, 8,10. The maximum value is 10.

A computer screen shot of a program code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen with text and numbers

Description automatically generated

A graph showing the results of a training course

Description automatically generated

A screen shot of a computer program

Description automatically generated

# Deploying the model

In this section, we focus on API development, which entails transforming the selected machine learning model into an analytics API using the Flask framework and the Pickle module for deployment on a local host.

A screenshot of a computer

Description automatically generated

# Key findings

In this section, we compare the accuracy scores of each best model generated by various machine learning algorithms. By evaluating the performance metrics of each model, we identify and select the best-performing one. Subsequently, we present the key findings derived from our analysis,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | Naive Bayes | Logistic Regression | SVM | Voting Classifier  (hard voting) | Random Forest  (1st attempt) |
| Accuracy | 0.906 | 0.784 | 0.963 | 0.951 | 0.955 |
| Precision | 0.902 | 0.780 | 0.960 | 0.970 | 0.960 |
| Recall | 0.686 | 0.780 | 0.960 | 0.931 | 0.960 |
| F1 Score | 0.741 | 0.780 | 0.960 | 0.950 | 0.950 |

According to the score generated from these five machine learning algorithms, some finding can be summarized:

Naive Bayes: Shows a high accuracy score of 0.906, but comparatively lower precision, recall, and F1 score, indicating that it may be classifying some instances incorrectly.

Logistic Regression: Has a lower accuracy score compared to Naive Bayes but shows similar precision, recall, and F1 score.

SVM: Demonstrates the highest accuracy score of 0.963, along with high precision, recall, and F1 score, indicating its effectiveness in classification tasks.

Voting Classifier (hard voting): Achieves a high accuracy score of 0.951, with excellent precision, recall, and F1 score, suggesting that it combines the strengths of multiple classifiers effectively.

Random Forest (1st attempt): Shows a competitive accuracy score of 0.955, with high precision, recall, and F1 score, making it a strong contender for classification tasks.

Overall, SVM and the Voting Classifier (hard voting) stand out as the top performers in terms of accuracy and overall performance across precision, recall, and F1 score. Random Forest also performs well but slightly lower than SVM and the Voting Classifier. Naive Bayes and Logistic Regression show lower accuracy and performance metrics compared to the other algorithms.

# Conclusion

In conclusion, this project exemplifies the power of predictive analytics in enhancing road safety and mitigating fatalities. Through the utilization of machine learning and API technology, we have developed a robust predictive software capable of assessing the likelihood of fatal collisions based on various factors. By leveraging a dataset sourced from the Toronto police department spanning six years, we were able to explore, model, and fine-tune predictive models using machine learning algorithms and classifiers.