SafeRoad

Final Report

Prepared for CS 4624
Virginia Tech
Blacksburg, VA 24061
Version 1.1.2
5/9/2016

Project Team: Matthew Morrison
Matthew Longobardi Paradiso
Julio Suriano Siu

Customer: Xuan Zhang, Virginia Tech

Supervisor: Dr. Edward Fox

Contents

Document Status Sheet

Version	Date	Authors	Summary
1.0.0	2/19/2016	M. Paradiso, M. Morrison, J. Siu	Document created
1.0.1	3/4/2016	M. Paradiso, M. Morrison, J. Siu	Made suggested edits and restructuring to chapters 1-3 Implementation plan added
1.0.2	3/30/2016	M. Paradiso, M. Morrison, J. Siu	Prototype report added
1.0.3	4/4/2016	M. Paradiso, M. Morrison, J. Siu	Report restructured into summary, user manual, and developer's manual. Section 2.5 updated with more detail about system requirements. Section 2.6 added with instructions for running program. Prototype and refinement integrated into developer's manual, beginning with section 3.11
1.0.4	4/18/2016	M. Paradiso, M. Morrison, J. Siu	Testing report added, sections 3.14, 3.15, and 3.16
1.1.0	4/26/2016	M. Paradiso, M. Morrison, J. Siu	Final report
1.1.1	5/4/2016	M. Paradiso, M. Morrison, J. Siu	Final report with all revisions made from initial submission
1.1.2	5/9/2016	M. Paradiso, M. Morrison, J. Siu	Final report with further revisions made

Table of Figures

Table of Tables

Chapter 1

Executive Summary

1.1 Purpose

The purpose of this report is to document the development of the SafeRoad software application and provide reference for future work. This project was completed as a capstone requirement for CS 4624 (Multimedia, Hypertext, and Information Access) at Virginia Tech, guided by the client Mr. Xuan Zhang.

SafeRoad is a software application designed to analyze the NHTSA vehicle complaint database, determine the most common complaints, and predict recalls based on these complaints. The goal of the application is to make our vehicles and roads safer and prevent the loss of life.

The software has been developed to be used by data analysts for automotive manufacturers and governing agencies such as the NHTSA. The software can be run either preemptively or in response to a complaint or series of complaints regarding an automobile or one of its components. The results of the program can lead to the issuing of a recall before more serious consequences occur.

The project has been developed using Java, database connectivity, and machine learning algorithms. A classifier training set has been created and included with the source code. The final product has proven to predict recalls with an accuracy level that is significantly higher than what was required.

1.2 Scope

Automobiles are an important part of life in the United States and the rest of the world. According to a 2015 census by IHS Automotive, there were 256 million vehicles registered in the United States, with about 1.8 vehicles per household[1]. With the number of vehicles on the road, it is important that these vehicles are as safe as possible, to prevent injuries and fatalities.

The purpose of the SafeRoad project is to build a software application that will:

- Prevent fatalities, injuries, and accidents resulting from defective manufacturing
- Summarize the most common complaints made to the National Highway Transportation Safety Administration (NHTSA)
- Predict whether or not common complaints will result in a recall
- Help identify fixes to common complaints
- Alert manufacturers of quality issues to help them correct problems
- Assist in the decision to issue Technical Service Bulletins (TSBs)
- Reduce instances of lawsuits and legal fines for automotive manufacturers resulting from defective manufacturing

Every year, automotive manufacturers issue recalls on vehicles for defective parts that can present a dangerous scenario for the occupants of the vehicle. One of the most recent recalls to make the news is the recall of Takata airbags. Due to a manufacturing defect, these airbags could potentially shoot metal shrapnel upon inflation, placing vehicle occupants at risk of injury or death. In the US alone, 19 million vehicles sold by 12 different manufacturers were equipped with these defective airbags[2]. Although the recall was issued in late 2014, the affected vehicles go as far back as those manufactured in the year 2000. It should be of great concern to automotive consumers that it took nearly 15 years as well as several injuries and deaths before any recall was issued.

There are several resources available that could have helped to prevent the casualties of the Takata airbag recall, and most other recalled products for that matter. The most immediate resources are the

NHTSA's databases. These databases contain detailed information about automotive complaints and recalls. While this information is available, the problem lies in the fact that these databases are too large to be easily analyzed by humans. The solution to this problem is SafeRoad, which will have the ability to analyze these large databases, determine the most common complaints, and make predictions of future recalls.

The advantage of the NHTSA databases is that they are organized in a format that allows data to be easily mined via common database software. By using these tools in conjunction with the growing number of machine learning tools that are becoming available, a successful implementation is certainly within reach.

There are many parties that can benefit from SafeRoad. The highest benefit is placed on the automotive consumer since the application aims to make roads safer. The intended users of SafeRoad, however, are automotive manufacturers and government agencies such as the NHTSA. Automotive manufacturers have a lot to benefit from the use of SafeRoad. Manufacturers often face lawsuits resulting in large fines as a result of accidents, injuries, fatalities, and general repair costs as a result of automotive defects. By taking advantage of recall predictions made by SafeRoad, automotive manufacturers can potentially avoid lawsuits and the costs involved. Another recent recall in recent years was that of the faulty GM ignition switches that led to 124 deaths and 273 injuries[3]. GM was fined $900 million for damages caused by the defective ignition switches. Had GM had a tool like SafeRoad, they could have caught this problem much sooner, issued a recall, and avoided these large fines. Not only can automotive manufacturers benefit from the potential avoidance of fines, they can also protect their image and retain buyers of their vehicles. Consumers are much less likely to buy a vehicle from a company that is being spotlighted in the media over a dangerous or defective product. Government agencies such as the NHTSA can also benefit from SafeRoad because it can help them warn automotive manufacturers of potential recalls, helping them to achieve their main goal of keeping roads and travelers safe.

1.3 Tasks and Deliverables

Upon completion of the project, deliverables will include this report, Java source code, and a classifier training set that will support the functionality of the Java application. Much of the labor of the project will be focused on the creation of the training set. The training set will contain a set of complaint records, where each record has been classified as a recall or not. The classifications will be based on whether or not a recall already exists for each complaint in the training set. The purpose of the training set is to "teach" the classifiers what patterns and attributes in the data are linked to a recall being issued. A well designed training set will is essential to the final product making accurate recall predictions. The availability of a high quality training set alone will be a useful tool in protecting lives.

Before the development team can begin writing the application, each member has been instructed by the client to become educated on the topic of machine learning. Each member is required to complete the Machine Learning course on coursera.org per the client's request. It is also the responsibility of the development team to research and become educated on the different machine learning sources available.

Chapter 2

User's Manual

2.1 Overview

This chapter provides a general description of the SafeRoad software application. Information in this chapter includes information about the current solutions to the issues addressed, general product function, information about the end user, system requirements for running SafeRoad, and how to run the SafeRoad application.

2.2 Product Perspective

Currently, there is no application that is distributed for the intent of predicting automotive recalls. Automotive manufacturers have, however, mined their own data from dealership complaints and supply chain records (to determine where defective parts were used) to pinpoint vehicles affected by defective equipment. One major problem with this is that each manufacturer is left to implement their own process identifying affected vehicles. Another problem is that these manufacturer implementations tend to address the problem after a significant number of incidents relating to the defective part have occurred. The focus of these applications is to identify vehicles equipped with the defective equipment; however, they do not do anything to predict recalls before the problem becomes widespread.

2.3 Product Functions

SafeRoad has three main functions. In the simplest terms, the functions are as follows:

1. Analyze the NHTSA complaint database to identify and summarize the most common complaints.
2. From the most common complaints, determine if any of them would likely result in a product recall from the automotive manufacturer.
3. Analyze the recall predictions for accuracy and correctness.

2.4 User Profile

Given that this application will be used mostly by automotive manufacturing companies and government safety agencies, the end users will likely be data analysts. While programming experience is not necessary for the end user, he or she will be required to be knowledgeable in running Java programs and in file I/O.

2.5 System Requirements

The recommended systems requirements for running SafeRoad are:

- Windows, MacOS, or Linux operating system
- 2 GHz processor or faster
- 4 GB RAM or more
- Java Runtime Environment installed (version 1.7 or newer)
- Reliable high-speed internet connection
- NHTSA complaint and recall databases on local machine
  Available from: http://www-odi.nhtsa.dot.gov/downloads/

- MySQL server installed (version 5.5 or newer)
  Available from: http://dev.mysql.com/downloads/mysql/
- Java Machine Learning Library bin file
  Available from: https://sourceforge.net/projects/java-ml/files/
- Connector/J driver
  Available from: http://www.mysql.com/products/connector/

2.6 Running SafeRoad

The source code for the SafeRoad application is available from:

https://github.com/mparadiso2/SafeRoad

Before the application can be run, the previously recommended systems requirements should be addressed. Installation of all required software is very easy; documentation can be found at each respective website. NHTSA database files can be downloaded from:

http://www-odi.nhtsa.dot.gov/downloads/index.cfm

For this application, the “flat_cmpl.zip” and “flat_rcl.zip” files will need to be downloaded. Once a MySQL server is installed and NHTSA databases have been downloaded, the user will need to import the NHTSA files into a local MySQL server database with the same name as the original file. Detailed information regarding the schema of the databases can be found in the Appendix of this document and should make importing the databases very simple with minimal knowledge of MySQL server.

Once the source code is downloaded and requirements are addressed, SafeRoad can be run in any Java Integrated Development Environment (IDE), or by simply navigating to the folder in the command prompt. Individual stages of the program can be run, or SafeRoadMain.java can be run to perform all steps at once. Most common complaints will be compiled to “CommonComplaints.csv” and predicted recalls will be compiled to “PredictedRecalls.csv”. Both files will be generated in the same folder as the SafeRoad source code. Refer to section 3.7 for more detailed information about these output files.

2.7 Definitions

IDE – Integrated Development Environment, interface for software development.
Examples include Eclipse, Netbeans, and many others.
JDBC – Java DataBase Connection, an open-source library that allows Java to connect with relational databases, perform queries, and gather results
JML – Java Machine Learning library, an open-source library for machine learning algorithms with extensive documentation
NHTSA - National Highway Transportation Safety Administration
ODI - Office of Defects Investigation
OVSC - Office of Vehicle Safety Compliance
Recall - voluntary repair made by automotive manufacturer due to a defective product
SQL – Simplified Query Language
SRS - software requirements specification
TSB - Technical Service Bulletin. Information regarding repairs to common complaints that have not been found to present a dangerous situation or be the result of manufacturing defects.
VIN - Vehicle Identification Number. A unique 17-digit alphanumeric serial number.

Chapter 3

Developer's Manual

3.1 Overview

This chapter provides information to guide the development process for the SafeRoad application. Information in this chapter includes specific information about the end user, usage scenarios, specific functionality, tools utilized, class design, time requirements, and time frame for the life of the project.

3.2 User Scenarios

Scenario 1:
Mr. Smith is a data analyst for General Motors. He was recently made aware of a complaint made at a Chevrolet dealership by the owner of a 2014 Chevrolet Cruze. The customer stated that he stepped on the brake pedal in an attempt to stop, however the brake pedal felt soft and went to the floor and the brakes failed to engage. Mr. Smith decides to consult with SafeRoad to help him determine if a recall may be necessary. Mr. Smith asks one of the IT guys with some SQL experience to download the NHTSA complaints database and deliver to him the subset of all GM vehicle complaints. Upon receiving the database of only GM complaints, he runs SafeRoad on this database and finds that SafeRoad predicts a recall on the brake system due to previous complaints and resulting injuries. Mr. Smith brings this to the attention of his supervisors, who issue a recall. As a result of the recall, the brake system is repaired on affected vehicles and further injuries and potential fatalities are prevented.

Scenario 2:
Mrs. Jones is a quality control manager for Ford Motor Co. She has been forwarded a few recent complaints of vibrations on the last generation of Ford Mustangs at speeds above 80 mph. She uses a simple SQL query to return all Ford Mustang complaints, then runs the results through SafeRoad. She finds that this is a relatively common complaint, however it never had any negative consequences beyond annoying the customer. She determines that a recall is not necessary, but does forward her findings to the manufacturing department. The manufacturing department determines the cause of the vibration to be a design flaw in the tie rods. The manufacturing department begins testing and research to develop a tie rod that does not exhibit these problems. Once the new tie rod is developed, they issue a Technical Service Bulletin (TSB) explaining that replacing the tie rod with the new design will fix the vibration. A TSB is not a recall, it is a bulletin that provides mechanics with information to quickly and easily diagnose and repair problems that are not a threat to safety.

3.3 Specific Functionality

The structure of SafeRoad is a series of Java applications. These applications can be run from the operating system's command terminal or through an integrated development environment such as Eclipse. There is no GUI planned at this time.

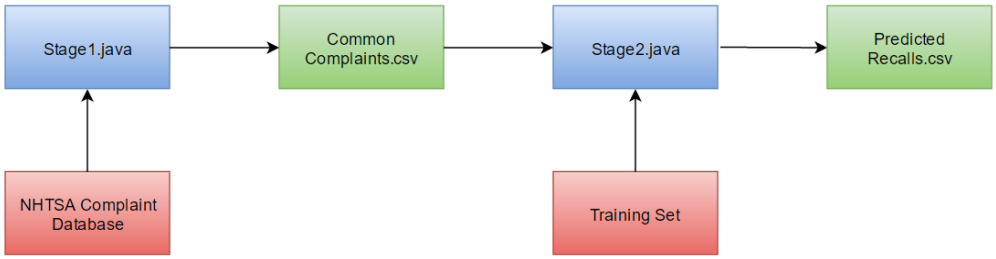
The function of the first application, which will be called "Stage1.java", is to identify and summarize the most common complaints. This application will make use of the Java DataBase Connection (JDBC) library to connect to the NHTSA complaint database. The JDBC library allows SQL queries to be called from the Java application. Queries can be called to return complaints based on vehicle year, make and model, number of reported complaints, and occurrence of crash, fire, injury, or death. The application will return a database file containing the subset of the most common complaints.

The second application, called “Stage2.java”, will work on the set of complaints returned by Stage1. First, a test set of complaints will be chosen at random. Given the large size of the complaints database and recommended test set size of 30% of the total set, a section of code will likely need to be developed to match existing recalls to complaints. Complaints that have already been addressed by an existing recall will be classified as such. Stage2 will utilize the Java Machine Learning (JML) library to learn about the most relevant factors of the complaints that are associated with recalls. The JML’s classification functionality will be applied to the results from Stage1, and common complaints will be classified into two classifications, likely recalls and unlikely recalls. The likely recalls will then be summarized in an output file.

The third application, called “PredictionsTest.java”, will work to analyze the results of Stage2. Using another randomly selected set of complaints, this set will be run through Stage2, then generate and report an F-test score based on the accuracy of the classification. The acceptable F-test score generated by PredictionsTest.java should be no lower than 60%, as specified by the customer.

Given the flexibility of Java applications, each stage of SafeRoad can be run individually, or through a main program that will run each stage sequentially. This main program will also be included as “SafeRoadMain.java”.

Figure 1. SafeRoad Workflow



3.4 Information Design

The two most important components of this application are the vehicle complaints and vehicle recalls. Both of these must initially be mined from their respective NHTSA databases. The choice to use the NHTSA databases allows specific subsets to be easily selected using simple SQL functions. There are several benefits to this choice including the ability to analyze specific qualities of complaints (such as manufacturer, component type, etc.) as well as faster running times resulting from a smaller data set. Each component class will be modeled after the most relevant data from their respective databases.

Table 1. Vehicle Complaint

TYPE	FIELD	DESCRIPTION
String	make	vehicle make
String	model	vehicle model
int	year	year of vehicle manufacture
Int	crash	1 if incident resulted in crash, 0 if not
int	fire	1 if incident resulted in fire, 0 if not

int	injured	number of people injured in incident
int	deaths	number of fatalities in incident
String	comp	defective component responsible for incident
String	desc	description of complaint
Enum	classification	“R” for recall, “N” for no recall

Table 2. Recall

TYPE	FIELD	DESCRIPTION
String	make	vehicle make
String	model	vehicle model
int	year	year of vehicle manufacture
String	compName	component description
String	defect_desc	summary of defect
String	consq	summary of consequences of defect
String	correction	summary of corrective actions taken

For a complete description of all properties of the NHTSA databases, see the Appendix about the official NHTSA database documentation.

3.5 Running Time Requirements

Given the large size of the NHTSA databases, there are no requirements on the running time of the application.

3.6 Deadlines and Deliverables

The following timeline has been established to keep the project on track and delivered by the deadline of April 26, 2016. All deliverables will be submitted to the client (Mr. Xuan Zhang) and supervisor (Dr. Edward Fox).

- February 19, 2016: Submit completed Requirements and Design Report.
- March 4, 2016: Submit Implementation report and corrections to Requirements and Design Report.
- March 16, 2016: Have Stage1.java functional (returning most common complaints from NHTSA complaints database).
- March 30, 2016: Submit Prototype and Refinement Report.
- April 6, 2016: Have Stage2.java functional (predicting recalls from common complaints) and ready for testing. PredictionsTest.java should be in progress.
- April 15, 2016: Submit Testing Report

- April 26, 2016: Submit Final Project Report and deliver completed SafeRoad application
- April 28, 2016: Presentation and demonstration of SafeRoad application

3.7 Implementation

This section contains information about the implementation plan for the SafeRoad software application. Information in this chapter includes team member roles, tasks, schedules, and progress history.

The SafeRoad application will need to query important information from the complaint database, and summarize that information so that it can be used to find a correlation with the recall database. The application will do this by taking counts of complaints by make and identifying the most commonly complained about components for each make. This data will be saved into a container class and added to a list. By default, the application will summarize the top 5 (roughly 10%) most complained about vehicle makes, and the top 3 most commonly failed components for each make. The number of vehicle makes to be summarized can be changed in the "cap" variable in Stage1.java.

The complaints table includes fields for the year, make, model, failed component, if a crash occurred, if a fire occurred, number of injuries, number of deaths, and complaint description (as well as several other fields that are unused in this application). The recall table includes columns for year, make, model, and failed component as well. The number of occurrences will be used in querying the complaints table to get an accurate count for how many times a complaint on a component for a make of car has happened. Once the most common complaints are compiled, they will then be passed into Stage2.java to be classified by the Java Machine Learning (JML) library.

The JML library can build classifiers around a data set using several different machine learning algorithms. The classifiers learn about the data set as they are built, and then a testing set can be run through the classifier. Like the training set, each testing set will contain complaint records that have been classified as a recall or no recall. The testing procedure will be to instantiate a new classifier, build it around the training set, then run the testing set through the classifier. Since each record in the testing set will already be classified, the classifier's prediction for each record can be compared to the existing classification to determine whether or not each prediction was correct. The development team plans on starting with K-Nearest Neighbors and Naïve Bayes for classification in the prototype and testing phase. Due to inconsistencies between the complaint and recall databases, the training set and testing sets for the machine learning algorithms must be created manually.

Each classifier being used in the testing phase will be analyzed for accuracy, and the best performing classifier will be chosen for the final application. The client, Mr. Zhang, has stated that the test procedure should calculate an F-test score and has specified a score or 60% or higher as acceptable. The F-test score will be generated for each test run. Specific information regarding the calculation of the F-test score can be found in section 3.14.

The entire SafeRoad application will be written in Java and SQL. The JDBC will be used to connect the Java program code to the MySQL database. All JDBC classes set up will use SQL queries to retrieve information from the database. Once the database information and summaries have been acquired, the JML library will be used to generate a machine learning algorithm to find correlations between given complaints and recalls, and to run an analysis on those correlations. A Github repository with all the source code will be used to coordinate coding efforts across the team. Frequent commits and pushes will be needed to insure that merge conflicts don't become a problem.

3.8 Points of Contact

Table 3. Points of Contact

Task	Point of Contact
JDBC Setup	Matt Paradiso
Query Construction/Analysis	Matt Morrison/Matt Paradiso*
Classifier Generation	Julio Siu
Classifier Testing	Julio Siu
Results Analysis	Matt Morrison

*Since this is a larger task, Matt Morrison will be POC for query construction and Matt Paradiso will be POC for query analysis.

For each task that the team completes, there will be a point of contact, who takes charge of the work on that task. The other team members will assist them with any work that the POC decides needs to be done. This system, where one team member is in charge of a task, puts someone personally responsible for each phase of development to be completed. This also means that someone is making the decisions of how the development is happening, which will make sure that everybody is on the same page when writing their code and making their commits to Github.

3.9 Major Tasks

For each major task that the team needs to complete for SafeRoad, an incremental development approach will be taken. The team will all contribute to the code base using a Github repository, and the team will need to commit and push their changes to the repository often to insure that few merge conflicts are created. Once an iteration of development has taken place, the code for that task will be tested thoroughly, and if any modification needs to be made, the task will go back into development. If the testing shows that no further changes need to be made to that part of the program, then the next task will go into development. This code development process will make it so that at each milestone, where a phase of the program has been coded, the SafeRoad program will function properly up through that phase.

The first major task that needs to take place is creating the JDBC classes that will be used to represent and communicate with the Complaint and Recall tables in the given database. At this stage any areas for SQL queries will either be left blank, or will be a general query that returns the entire database table as a placeholder. These classes will need to be tested for a proper connection to the database so that queries can be run. Once the general structure of the first phase has been defined and decided on in this step, the SQL queries will be ready to be written.

The second task to finish off the first phase of the program will be to write the actual queries that will be used to get the necessary information from the database. These queries will be filled into methods in the JDBC classes. Once the queries have been written, the methods will be returning a series of collections, single results, and counts. These query results will need to be organized, either in the output of the query methods themselves, or after the queries have been run in execution, so that a proper set of output variables can be passed on to the next phase. This task will conclude the first milestone of the project. The first milestone was completed before the March 16, 2016 deadline.

The third task, also the second milestone, is to set up a classification generator through the JML library. A class will need to be created for the machine learning algorithm to run and collect information to build a classifier from a certain number of data entries in the set. This class will take in the summarized data set from the first phase, and use that information when setting up how the classifier will be created and taught to find connections between complaints and recalls. This milestone has also been completed ahead of schedule.

The fourth task is to test the generated classifiers to make sure that they meet the specified 60% F-test score. Methods will need to be written that run a test of at least 100 random sample records from the database, some that do fit the classification and some which don't, and run them through the several machine learning classifiers. For each classifier being tested, the program should report the F-test score from running the classifier on the testing set, and the best performing classifier will be chosen for the final product. If no classifier achieves the specified 60% F-test score, then the training set will need to be improved until an acceptable score is reached.

3.10 Implementation Schedule

Figure 2. Development Schedule

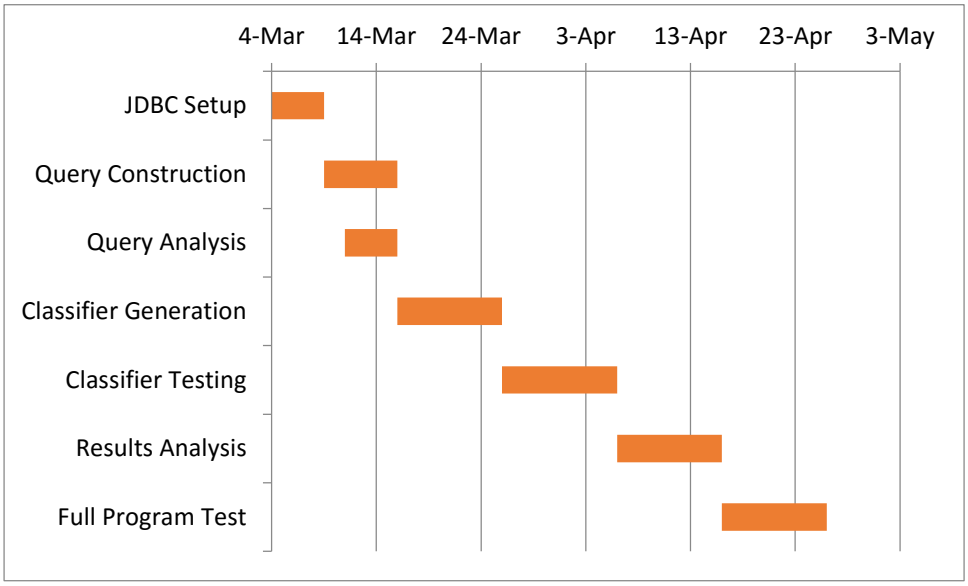


Figure 2 shows the schedule for each phase of development. The first phase of development will begin immediately following the submission of the Implementation Document on March 4. The setup of the JDBC classes will take 5 days, at which point the Query Construction will begin. On March 11, several days into the query work and when several queries should be written and functional, work will begin for another 5 day period to write the JDBC code for returning the proper information that was collected into data structures from the queries. This first phase of development will be completed by the March 16 milestone.

The second phase of SafeRoad development will start on March 16, with the Java Machine Learning class to develop the classifiers for analyzing any correlation between the database table entries. This development will take 10 days, and then an 11 day development period will run from March 26 to April 6 during which the code will be added that runs the tests on the classifiers. The second phase will be completed for a second milestone on April 6.

The final phase of development is the results analysis and full program testing. The results analysis will run from April 6 to April 16. By April 16, the last iteration cycle of the program will be developed, and the full program will be able to run. The final 10 days of development will be used for full test runs of SafeRoad, and the elimination of any bugs and problems that are encountered in the full testing of the program. After the full tests have been completed, the program will be completely ready for use on April 26.

3.11 Prototype Functionality

As of March 30, 2016, a working prototype has been developed for the SafeRoad application. The prototype currently compiles the most common complaints from the complaint database, with all complaints initially unclassified, and then classifies each complaint according to whether or not a recall should be issued for the given complaint.

In the first stage of the application, the JDBC library is used to connect to the NHTSA complaint database, which has been imported into a MySQL Server database. Initially, the database was on a remote server. After initial testing, the decision to import the database to a local host was made since the client is required to have their own copy of the databases. Upon making the switch from a remote to a local MySQL server, we immediately saw the average run time reduce from four minutes to just over one minute. For reference, the machine used for testing has a 2.50 GHz Intel Core i5 processor.

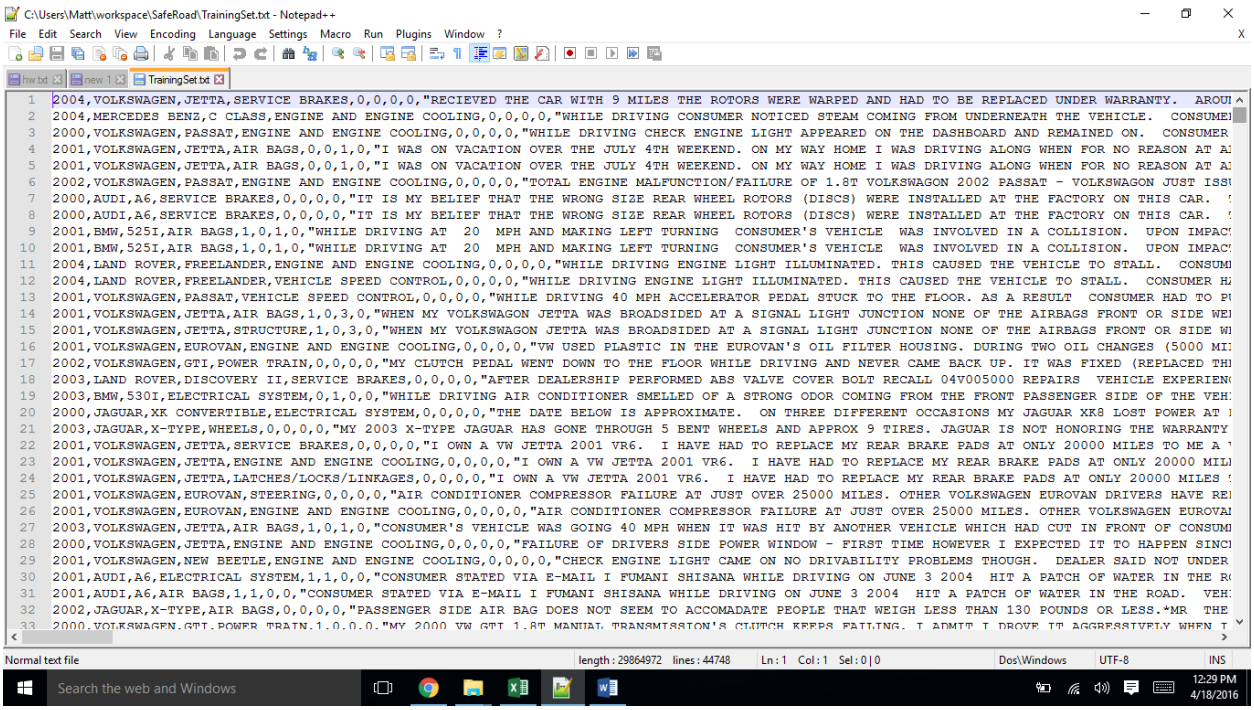
In order to recognize the most common complaints, a SQL query is run on the complaints database to determine each unique vehicle make, then count the number of complaints for each vehicle make. The top 5 vehicle makes (roughly the top 10%) with the highest complaint counts are then saved in a list. The next step then determines the component systems that are most commonly complained about for each make. Another SQL query is run on the complaints database for each vehicle make saved in the list that counts each unique component system and the number of complaints received for that specific component system. Given the relatively small fixed number of component systems, the team decided on saving the top 3 component systems that were most commonly complained about for each of the vehicle makes. Once is result set is returned, each vehicle make and its corresponding top 3 most commonly complained about component systems are saved into a container class object, and each of those is added into another list. From this list of container objects, a SQL query is generated to return all complaints that match the vehicle makes and the corresponding component systems.

Once the result set is returned from the final SQL query, the most relevant information is then parsed and printed to a file. We have determined the most relevant information from each complaint to be the vehicle year, make, model, failed component system, number of injuries, number of deaths, if a crash occurred, if a fire occurred, and the complaint description. The final results are printed to a file titled “CommonComplaints.csv” and all records are classified as “U” for unclassified.

The second stage of the prototype deals with the machine learning aspects of the application. In the development process, the team has researched and tested several classification algorithms available in the Java Machine Learning (JML) library. For this application, the team has found the Naïve Bayes Classification to be the most appropriate and has showed some promising results using it.

In this stage of the application, a Naïve Bayes Classifier object is instantiated, then given a training set to identify and learn patterns that constitute the need for a recall. The training set is structured in a comma separated values (.csv) format. This format has been chosen for its ease of use and compatibility with database software including Microsoft Office Excel and MySQL server. By using comma separated value format, these files can easily be imported into MySQL database tables. Another benefit of this format is the compatibility with Microsoft Office Excel, which allows for easy sorting, visualization, analysis, and running of macros on the data. Once the classifier is instantiated, it is then run on an unclassified dataset, and classifies each complaint in the dataset as either “R” for recall, or “N” for no recall. For each complaint that is classified as “R”, the complaint is printed to a file titled “PredictedRecalls.csv”. Figure 2, seen on the next page, is a screen dump of the training set data.

Figure 3. Training Set Data



One of the biggest obstacles has been the generation of a training set. As a general rule in machine learning, the size of a training set should be about 10% of the size of the data set to be analyzed. Given that the size of the NHTSA database is roughly 1.25 million records and growing, this task is nearly impossible without a very large team of people to divide the work amongst. The current training set has 350 records, however, the team is still working on developing and consolidating a much larger training set in order to achieve more accurate results. Each record in the training set contains information about the year, make, model, failed component system, whether or not a crash occurred, whether or not a fire occurred, number of injuries, number of deaths, complaint description, and classification. For the classification attribute, each record is simply labeled as “R” for recall, or “N” for no recall. For the training set, these classifications are manually assigned by the development team and are based on recalls that already exist in the NHTSA recall database. With the current training set size of 350 records, roughly 0.024% of the size of the NHTSA complaints database, the prototype has still been able to correctly classify roughly 40% of the complaint records in the test set. This issue has been addressed with the client, who has specified that the training set should contain at least 1,000 classified records.

3.12 Prototype Challenges

Just like any other software development project, the development of the SafeRoad application has also had its challenges and obstacles. In this section, current challenges and strategy plans to work around them will be discussed.

One of the biggest challenges has been how to apply the machine learning techniques and which resources to utilize. Each machine learning resource has its advantages and disadvantages. The initial plan was to utilize the Java Machine Learning library. The JML library has shown some promising results, however, it is not without its limitations. The team researched other machine learning resources in an effort to see if there is another package that is more suitable for the goals of the SafeRoad application.

One alternative machine learning resource that exists is the Datumbox Machine Learning Framework (http://www.datumbox.com/machine-learning-framework). The team attempted to try using this framework due to its wide range of capabilities, extensive documentation, and general praise from the open source community. The problem encountered with this resource was that it required many other supporting applications and a complex configuration that made the whole process too complicated for the intended user of the SafeRoad application.

Another resource that was researched and tested was Classifier4J (http://classifier4j.sourceforge.net/). The team was drawn to this resource due to ease of integration into the existing project. However, the documentation was lacking for this resource and further learning time would not have fit into the timeline of this project. Likewise, the consideration of implementing our own machine learning was not feasible due to time constraints.

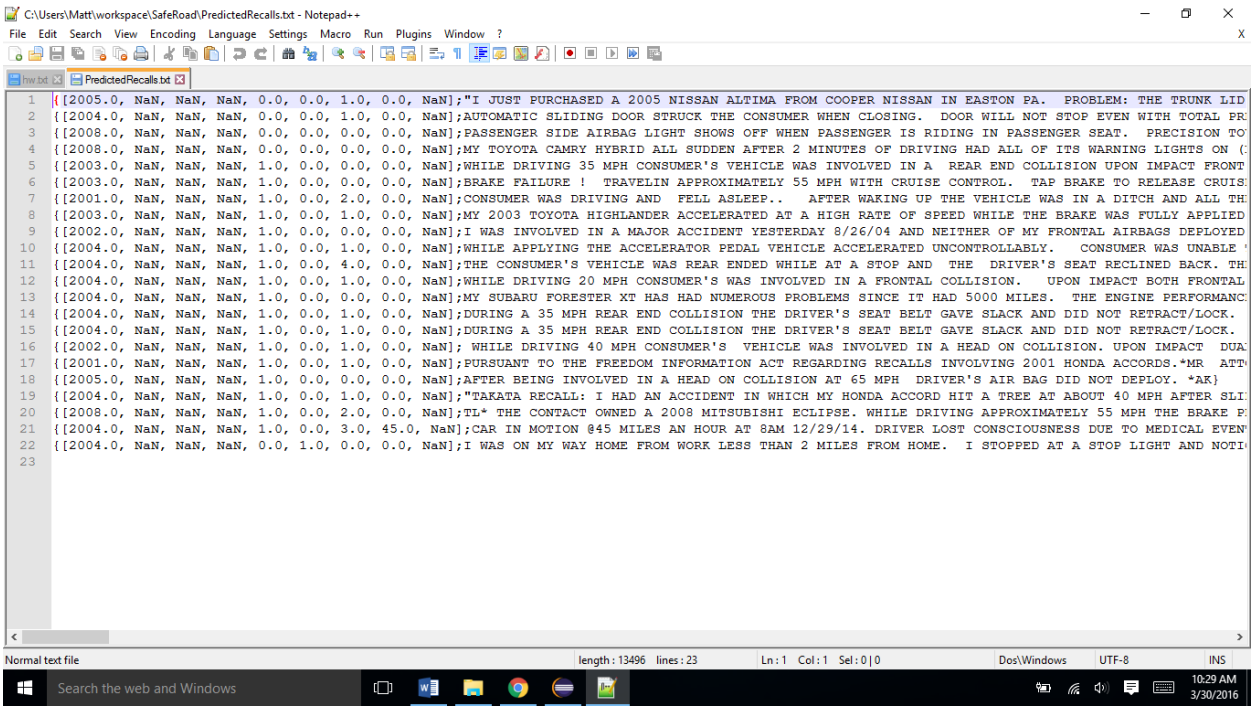
Having researched other machine learning resources, the team decided that the JML library was indeed the best choice for this application. The JML library offers very easy integration into the project, and so avoids any complexities for the end user. The JML library is a very well documented project. Thanks to the documentation, learning how to work with the library is much easier than other resources and comes quickly. The thorough documentation and quick learning curve make it the best choice for the team to stay within the time constraints of the project. The JML library also offers many different machine learning algorithms, and suffices to satisfy the requirements of the project.

While the JML library is well suited to the requirements of the SafeRoad project, there are some limitations. One of the biggest problems currently is that the classifiers require data to be represented by numbers, and excludes any information that is not in a number format. As stated before, each complaint record that is run through the classification algorithm contains a vehicle year, make, model, component system, whether or not an accident occurred, whether or not a fire occurred, number of injuries, number of deaths, and the complaint itself. Being that the classifier algorithms are only compatible with numbers, the only data being used to classify complaints are those in numeric form. In the case of vehicle complaints, the classifier was initially only using the vehicle year, number of deaths, and number of injuries to categorize complaints.

The team has identified a work around for the limitations of the JML library classifiers. Since the occurrence of an accident and a fire can only have two values (yes or no), a simple code fix was implemented to represent yes as 1 and no as 0. With this simple fix, the classifiers now consider two more fields in each complaint when predicting classifications. The goal is to be able to represent all fields as numbers. While fields such as make, model, and component are not binary, there is still a fixed number of possibilities. The planned approach is to generate a table of all makes, models, and components and then assign an integer value to each. With this approach, the classifiers will be able to consider all fields of the complaint except the actual complaint description. While this will greatly improve the accuracy of the classifier, it still presents the issue of not considering the actual complaint.

Figure 4, shown below, shows recall predictions on a small test set. To the left of the data matrix that the classifier uses to predict whether or not a complaint is cause for a recall. The fields labeled “NaN” represent fields that are currently in text format, and are therefore not being considered in the prediction algorithm. The goal is to have all fields represented by a number so that all fields are considered when predicting recalls.

Figure 4. Recall Predictions – Prototype



Since the complaint description is a text field with unlimited values, it cannot be represented by a number like the other fields in the complaint. The team has addressed this issue with the client, Mr. Xuan Zhang. As per Mr. Zhang's suggestion, a potential fix for this is to restructure how the common complaints are compiled in the first stage of the application. Rather than taking counts of unique vehicle makes and counts of component failures for each make, he would like for the application to take a bag-of-words approach. The plan would be to search through the NHTSA complaints database and determine the most common words. Once the most common words are determined, only complaints containing those words will be retrieved and processed by the classifier.

This will allow for all fields to be considered in the whole process of the application. The only challenge that will arise from this approach is to filter out words that will be present in most cases, such as "the", "is", "my", etc. These are referred to as "Stop Words". If this method is chosen, it will be implemented in addition to the current method.

The other challenge, as stated in the previous section, is the size of the training set. Several recommendations have been made by Mr. Zhang to counter this problem. One option is to build a training set from the recall database and reduce the number of attributes for each complaint. Both databases have fields for year, make, model, manufacturer, component system, and a description. Using only these attributes, it is possible to add every existing recall to the training set with very little effort. Mr. Zhang has suggested to try this approach and test the accuracy, then incrementally add more attributes such as crash, fire, injuries, and deaths. Another suggestion was to reduce the size of the NHTSA databases by removing irrelevant complaints such as from outdated data. The complaint database contains complaints for vehicles as old as model year 1949, and the recall database likewise contains some much older vehicle recalls as well. Since it is highly unlikely in the year 2016 that a recall will be issued for a vehicle from 1949, complaint records such as these have been removed. The team decided on removing all records for vehicles before the year 2000. By removing these complaints, both databases were nearly halved in size while still giving a window of 17 years to go back and issue a recall. Given that most recalls are issued within a few years of the vehicles year of manufacture, the 17 year window is more than adequate. While these measures do drastically reduce the size of the databases, a very large training set is still required and the team will continue to compile the best possible training set in the time given.

3.13 Refinement

The previous section stated the current challenges of the project and plans to overcome them. This section will focus on other changes to be made to the existing project and tasks that are yet to be accomplished.

The SafeRoad prototype is currently making its classifications using the Naïve Bayes Classifier. Per Mr. Zhang's request, he would like to see the final application use several different classifiers and generate accuracy scores for each. This should be attainable as soon as the make, model, and component can be represented as integer values. Classification algorithms that will become available with this fix include K-Nearest Neighbors, Bagging, and Self-Organized Map (SOM) Clustering. Another issue that will be addressed if time permits, is the representation of data in the PredictedRecalls.csv file. For ease of interpretation by the end user, it is preferable if the application can print the values of make, model, and component. If this proves to be too difficult, or if time prevents this development, then a lookup table will be included so that the user can easily identify values that are not explicit in the prediction file.

As stated in the previous section, the development team will begin to explore the bag-of-words approach proposed by Mr. Zhang in addition to the current method of determining the most common complaints. The way this works is that each line of text is treated as a document, occurrences of each unique word are counted and normalized, turning each word count into the probability that it will occur in the document. In order to exclude very common words such as "the", a term frequency-inverse document frequency correction is applied to give weight to relevant words. Currently, the development team is researching existing bag-of-words implementations that can easily be applied to this project.

While part of the team works to make these changes, the other part of the team will begin work on the testing and evaluation phase of the project. The initial prototype has shown some promising results, however, there is still a lot of work to be done to ensure the target accuracy requested by the client.

To meet this end, the program will need to be evaluated on classifier and query accuracy, runtime, and input/output of the program. The testing part of the team will perform both JUnit tests and manual tests on the project to learn where the program needs to be fixed or modified.

Since the product is written almost completely in Java, JUnit testing will be used as the primary testing practice for this project. Test classes will be written for each Java class, with each test class having a variety of methods to test each individual method with a number of inputs. Testing methods will also be written to test the entirety of the program from start to finish. JUnit tests allow for the quickest and most repeatable testing procedures to be written. Once a test method has been written, it is very easy to modify the input being put into the program, so writing and running many different tests can be done with speed and efficiency. Methods testing specific sections of code ensure that it is easy to locate where errors exist within the program.

In addition to the JUnit testing that will be performed by the team, a certain amount of manual testing of the program will take place. Manual tests will be performed both on the database itself as well as through complete program runs. Tests run on the database will ensure that the queries are gathering all of the necessary database entries that the algorithm will need to perform its classifications. Since the team is changing its approach to compiling common complaints, these tests will be important for creating the proper query approach. The full run through tests serve to make sure that the input and output are intuitive to the user, and that the run of the program takes place within a reasonable amount of time. While these manual tests will take longer than the JUnit tests will, they will provide valuable information on where the project could use improvements.

During the cycles of testing for the program, the team will also be working on improvements and bug fixes based on the findings of the tests. JUnit testing will be done more heavily in the earlier testing cycles, so that the team can remove any bugs that lie in the various stages of java code. The later testing cycles will be more centered on the manual testing to check full program accuracy, runtime, and ease of use. At that stage, most of the actual errors should be fixed and improvements will be made in terms of speed

and usability. However, the written JUnit tests will still be run to be sure that no changes made will break what has already been properly implemented.

While the prototype of SafeRoad has shown the team a variety of problems with the implementation, the code reworks should not be so severe that the project won't be completed on time. The primary changes to be made on how the program runs are to update the query strategy so that term frequency within complaints is used to gather common complaints, and to add classifiers to the machine learning algorithm so that multiple classifiers are used to check for a recall on the complaint. Currently, there is not an extremely large amount of code, so writing the JUnit tests should be quick. Once those are written, the entire testing effort can be centered upon the manual testing of the queries and for the intended user experience.

3.14 Testing Methods

The testing phase of the SafeRoad application is currently in progress. The development team has created a training set of complaints and classified them accordingly. Currently, the training set contains just over 350 records. Complaints that have existing recalls to fix them are classified as "R" and complaints that would definitely not lead to a recall are classified as "N". Complaints classified as "N" are complaints that are not directly related to safety. Examples include A/C not blowing cold, radio not working, and complaints that were obviously caused by the driver and not a manufacturing fault. The training set is structured in a comma separated values (.csv) format. Each line of text in the file corresponds to a complaint. Each complaint is structured in the following order:

```
Year, make, model, failed component, crash, fire, injuries, deaths, complaint
```

Two examples of a complaint are shown below, with the first corresponding to no recall and the second corresponding to a recall:

```
2004,Honda,Element,Service Brakes,0,0,0,0,"Vibration when braking",N
2007,Nissan,Altima,Steering,1,0,1,0,"Couldn't steer car and crashed",R
```

Since the last report, the team has implemented a hash map that maps all makes, models, and components to a numerical value, making it compatible with the JML classification algorithms. As a result, the two complaints will be converted to the following before being processed by the classifiers:

```
2004,7,201,103,0,0,0,0,"Vibration when braking",N
2007,6,270,115,1,0,1,0,"Couldn't steer car and crashed",R
```

A complete index of each word to their numerical values is located in the source code folder with the name "`Lookup Table Key Value Pairs.txt`". The same training set was also modified in two ways to create two more training sets to analyze how each attribute affects the accuracy of the classification algorithms. The first modified set, referred to as the short dataset, removes the crash, fire, injuries, and deaths attributes. These fields were chosen because they are all common to both the recall and complaint database. This way, the development team can easily extract all data from the recall database and structure it in this format with the simple SQL query "`select YEARTXT, MAKETXT, MODELTXT, COMPNAME, CONSEQUENCE_DEFECT from flat_rcl`". Performing this one query will easily create a training set of every known recall in just one step. The issues, however, are that non-recall complaints will need to be added and that attributes removed to achieve this are rather important in determining a recall.

The second modified training set does the same as the last. However, it also removes the complaint description. The benefit is that everything is represented in numerical form, so it is compatible with the K-Nearest Neighbors classifier in the JML library. It also has the benefit of being able to construct a large training set with one SQL query as explained in the previous set.

To summarize the above, the Naïve Bayes classifier will be applied to the original training set and the short dataset. The K-Nearest Neighbors will be applied to the numerical dataset. Since K-Nearest Neighbors compares each complaint to a specified number of neighbors, several test runs will be performed with different numbers of comparisons.

In order to determine the accuracy of the classifiers, the test procedure will calculate the F-test score for each classifier on their corresponding sets. The F-test score is the standard measurement for determining quality of binary classifiers. The F-test considers correct predictions as well as false-positives and false-negatives. As specified by the client, Mr. Zhang, the goal is to achieve an F-test score of 60% or higher.

The formula for calculating the F-test score is as follows:

```
F = 2 * (P * R) / (P + R)
```

Where:

```
P = correct recall classifications / all recall classifications
R = correct recall classifications / expected number of recall
    classifications
```

For example, if a test set has 4 real recalls, and it predicts 3, but only 2 are correct predictions, then:

```
F = 2 * (.66 * 0.50) / (.66 + .50) = 57%.
```

Another useful feature that is built into the JML library is gain ratio analysis. Each attribute in a complaint (model, component, fire, etc.) is given a weight between 0 and 1. The higher the weight, the more important it is in making the decision of how to classify the complaint. Analyzing these numbers can give clues on how to tweak and optimize the training set to give more accurate predictions.

3.15 Testing Results

The following is the output from running the test cases on our training set:

```
Testing Naive Bayes Classification...
Naive Bayes Classifier F-Test Score: 0.6103896

Naive Bayes Attribute Gain Ratios:
Year: 0.008309330247688679
Make: 0.009242350044789648
Model: 0.026993063163164556
Component: 0.019119437573581498
Crash: 0.2834499470420936
Fire: 0.1559737688785263
Injuries: 0.19940323953898803
Deaths: 0.18971145095079223

Testing Naive Bayes Classification on short dataset...
Naive Bayes Classifier F-Test Score: 0.65088755

Naive Bayes Attribute Gain Ratios (Short Dataset):
Year: 0.008309330247688679
Make: 0.009242350044789648
Model: 0.026993063163164556
Component: 0.019119437573581498
```

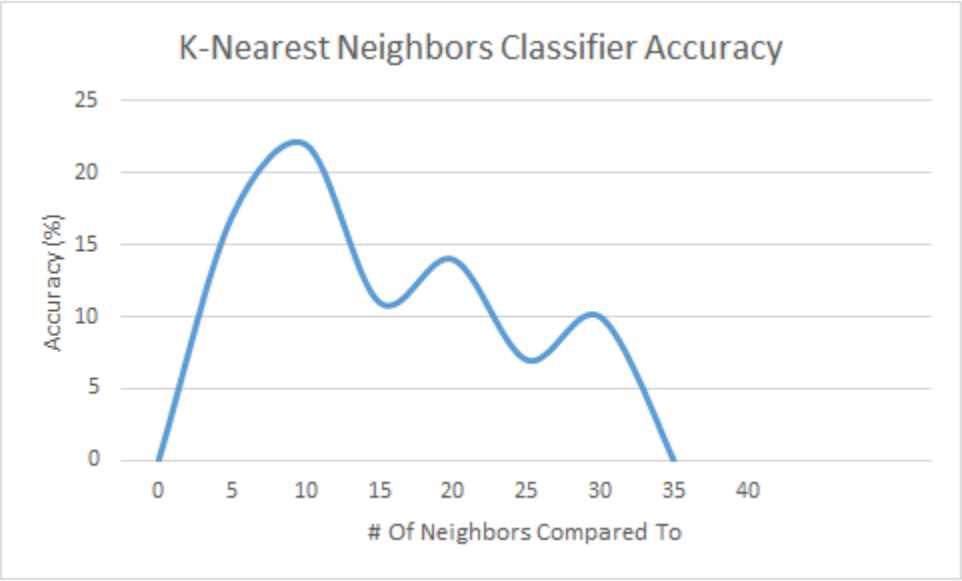
```
Testing K-Nearest Neighbors Classification...
K-Nearest Neighbors Classifier F-Test Score: 0.2208589

K-Nearest Neighbors Attribute Gain Ratios:
Year: 0.008309330247688679
Make: 0.009242350044789648
Model: 0.026993063163164556
Component: 0.019119437573581498
```

The analysis shows that the Naïve Bayes classification on the normal training set does barely pass the acceptable F-test score with 61%. Naïve Bayes on the short dataset improves to 65%, however, the gain is small at the cost of removing such important attributes. As the training set grows larger, the development team believes that the gap in accuracy will decrease due to false-positives and false-negatives. For example, a new vehicle model may have many complaints that resulted in deaths, however the lack of the make in the training set may cause a false-negative. Likewise, a complaint may match the attributes of an existing recall, however the complaint may not be serious, resulting in a false-positive. The gain ratios in the above results show the importance of crash, fire, injuries, and deaths in determining if a complaint should result in a recall. It should also be noticed that the gain ratios for year, make, model, and component attributes are identical for each classifier and are very small. This again stresses the importance of the extra attributes in the original training set. Due to the high gain ratios of these attributes, the development team will continue to use them in the final product. With the importance of crash, fire, injuries, and deaths attributes, the classifier has a much higher chance of correctly predicting recalls for new makes and models not in the training set.

Finally, the K-Nearest Neighbors proved to be far from acceptable, with its highest score of 22%. K-Nearest Neighbors was tested with several values, and comparing the 10 nearest neighbors yielded the highest score. Figure 5, shown below, shows the accuracy with respect to the number of neighbors compared to.

Figure 5. K-Nearest Neighbors Classifier Accuracy



3.16 Further Refinement

While the SafeRoad application is reaching its performance goals, there is still room for improvement. The most feasible path at this stage in development in to keep adding to the training set to improve classifier accuracy. With the size of the development team and the deadline approaching soon, there is no way that the team can reach the 10% mark, i.e., 125,500 complaints in the training set. This issue has been addressed with the client and an acceptable training set size of 1,000 complaints has been specified. Achieving this is the number one goal for the development team at this stage.

While the current 61% F-test score is technically acceptable to our client, we strive to deliver an excellent product and are aiming for at least an 80% F-test score by the time of delivery.

Chapter 4

Conclusions

4.1 Final Results

Since the testing phase, the development team has increased the size of the training set to 1,000 records. As stated in the previous section, the Naïve Bayes classifier was chosen for the classification of complaints because it provided promising F-test scores. The original dataset containing the crash, fire, injuries, and deaths attributes was chosen over the short dataset because of the high gain ratios of those attributes. In further testing, the development team's prediction of false positives and negatives were increased when these attributes were excluded. False negatives were produced when makes and models not in the training set were introduced into the test set. Likewise, false positive results were produced for non-serious issues that shared make, model, and component with other records that were classified as recalls in the presence of crashes, fires, injuries, and deaths.

The development team was able to overcome the compatibility issues with the classifiers by implementing a Hash Map to map vehicle makes, models, and components to numerical values. An inverse Hash Map was also created to map the numerical values back to their text values as well. This allows for text to be converted to numerical form for use with the classifiers, and then be converted back to text in the output files after the records have been classified. Another benefit of using the Hash Map is the constant search time, which allows for the conversion between text and numerical values without any noticeable difference in program running time.

In final testing with the training set of 1,000 records and the test set containing the crash, fire, injuries, and deaths attributes, the Naïve Bayes classifier achieved an F-test score of 89%. We are very pleased with these results, as the requirement was 60%.

We are also quite pleased with the running time and size of the predicted recalls file with respect to the complaint database being analyzed. The complaint database currently contains about 1.25 million complaints with a file size around 650 megabytes. The SafeRoad application completes in less than two minutes on the test machine with a 2.50 GHz Intel Core i5 processor. The predicted recalls file generated by the most recent run contained around 18,000 complaints with a file size of 11 megabytes. In comparison to the original complaints database, the predicted recalls file is roughly 1.5% of the size.

4.2 Future Development

With the project time frame and the size of the development team, the final product is somewhat limited in terms of what can be achieved with machine learning and natural language processing. Much of the time put into this project was spent on understanding machine learning and generating the training set for the classifiers.

For future developments, we would like to see the implementation of bag-of-words, which was suggested late in the development process. We would also like to see the implementation of textual K-Nearest Neighbors, which was also suggested late in the development process. This approach would take common keywords based on the recall database, and predict recalls based on their proximity to other words in the complaint description. For example, in complaints containing the word "airbag", it would predict a recall when "airbag" is followed by the words "did not deploy".

4.3 Lessons Learned

The development of the SafeRoad application has been very educational for the development team. Every member of the team went into the project with no knowledge of machine learning and natural language processing, and very limited knowledge of databases. We have each become more skilled with analyzing data with SQL and have learned how to connect to SQL databases with Java, providing a means for even more advanced database analysis and manipulation. Each of us has also gained a better understanding of machine learning that we all hope to continue building upon.

Throughout the life of the project we faced a few problems and learned a lot from both our mistakes and solutions. While we were able to overcome the issue of representing text as numbers for the JML classifiers, we would have liked to use some of the more compatible machine learning resources that we found late in the development phase. We initially chose the JML library for its ease of use and thorough documentation. We initially came up with our plan to compile the most common complaints based on which vehicle makes, models, and components were most frequent in the complaint database. Having our initial plan, we became so focused on implementing the project based on that plan. Rather than backtracking and trying new procedures, we continued to push on with the initial plan. While our project did produce good results, we don't know if another approach may have produced better results. Looking back, it may have been more effective to have each team member become familiar with a specific machine learning resource, then moved on with the project once we had results from each. Instead we took a linear approach which made it harder to go back and change aspects of the project.

4.4 Inventory

Table 4 contains a description of all files included in the final submission

Table 4. Inventory of Final Submission

SafeRoadMain.java	Main program
Stage1.java	Java program to compile most common complaints
Stage2.java	Java program to classify most common complaints
PredictionsTest.java	Java test class to analyze classifiers
LookupTableKeyValuePairs.txt	Text file containing map of text values to numerical values for classifiers
TrainingSet.csv	training set containing year, make, model, component, crash, fire, injuries, deaths, complaint, and classifier attributes
TrainingSetShort.csv	Training set containing only year, make, model, component, and complaints attributes
TrainingSetShort2.csv	Training set containing only year, make, model, component, and complaints attributes with new makes and models introduced
Training_Set_No_Text.csv	Training set containing only year, make, model, and component. Used with K-Nearest Neighbors classifier
Javaml-0.1.7	Java Machine Learning Library
Mysql-connector-java-5.1.38	Java Database Connection Library

Acknowledgements

The entire SafeRoad development team would like to thank our client, Mr. Xuan Zhang, for the opportunity to work on this project. We would like Mr. Zhang for his guidance and help throughout the entire development process.

We would also like to thank Dr. Edward Fox for all of the help, understanding, and being so flexible throughout the entire project.

Appendix

Complaint File Characteristics:

Field#	Name	Type/Size	Description
1	CMPLID	CHAR(9)	NHTSA'S INTERNAL UNIQUE SEQUENCE NUMBER. IS AN UPDATEABLE FIELD,THUS DATA FOR A GIVEN RECORD POTENTIALLY COULD CHANGE FROM ONE DATA OUTPUT FILE TO THE NEXT.
2	ODINO	CHAR(9)	NHTSA'S INTERNAL REFERENCE NUMBER. THIS NUMBER MAY BE REPEATED FOR MULTIPLE COMPONENTS. ALSO, IF LDATE IS PRIOR TO DEC 15, 2002, THIS NUMBER MAY BE REPEATED FOR MULTIPLE PRODUCTS OWNED BY THE SAME COMPLAINANT.
3	MFR_NAME	CHAR(40)	MANUFACTURER'S NAME
4	MAKETXT	CHAR(25)	VEHICLE/EQUIPMENT MAKE
5	MODELTXT	CHAR(256)	VEHICLE/EQUIPMENT MODEL
6	YEARTXT	CHAR(4)	MODEL YEAR, 9999 IF UNKNOWN or N/A
7	CRASH	CHAR(1)	WAS VEHICLE INVOLVED IN A CRASH, 'Y' OR 'N'
8	FAILDATE	CHAR(8)	DATE OF INCIDENT (YYYYMMDD)
9	FIRE	CHAR(1)	WAS VEHICLE INVOLVED IN A FIRE 'Y' OR 'N'
10	INJURED	NUMBER(2)	NUMBER OF PERSONS INJURED
11	DEATHS	NUMBER(2)	NUMBER OF FATALITIES
12	COMPDESC	CHAR(128)	SPECIFIC COMPONENT'S DESCRIPTION
13	CITY	CHAR(30)	CONSUMER'S CITY
14	STATE	CHAR(2)	CONSUMER'S STATE CODE
15	VIN	CHAR(11)	VEHICLE'S VIN#
16	DATEA	CHAR(8)	DATE ADDED TO FILE (YYYYMMDD)
17	LDATE	CHAR(8)	DATE COMPLAINT RECEIVED BY NHTSA (YYYYMMDD)
18	MILES	NUMBER(7)	VEHICLE MILEAGE AT FAILURE
19	OCCURENCES	NUMBER(4)	NUMBER OF OCCURRENCES
20	CDESCR	CHAR(2048)	DESCRIPTION OF THE COMPLAINT
21	CMPL_TYPE	CHAR(4)	SOURCE OF COMPLAINT CODE: CAG =CONSUMER ACTION GROUP CON =FORWARDED FROM A CONGRESSIONAL OFFICE DP =DEFECT PETITION,RESULT OF A DEFECT PETITION EVOQ =HOTLINE VOQ EWR =EARLY WARNING REPORTING INS =INSURANCE COMPANY IVOQ =NHTSA WEB SITE LETR =CONSUMER LETTER MAVQ =NHTSA MOBILE APP MIVQ =NHTSA MOBILE APP MVOQ =OPTICAL MARKED VOQ RC =RECALL COMPLAINT,RESULT OF A RECALL INVESTIGATION RP =RECALL PETITION,RESULT OF A RECALL PETITION SVOQ =PORTABLE SAFETY COMPLAINT FORM (PDF) VOQ =NHTSA VEHICLE OWNERS QUESTIONNAIRE
22	POLICE_RPT_YN	CHAR(1)	WAS INCIDENT REPORTED TO POLICE 'Y' OR 'N'
23	PURCH_DT	CHAR(8)	DATE PURCHASED (YYYYMMDD)
24	ORIG_OWNER_YN	CHAR(1)	WAS ORIGINAL OWNER 'Y' OR 'N'
25	ANTI_BRAKES_YN	CHAR(1)	ANTI-LOCK BRAKES 'Y' OR 'N'

26	CRUISE_CONT_YN	CHAR(1)	CRUISE CONTROL 'Y' OR 'N'
27	NUM_CYLS	NUMBER(2)	NUMBER OF CYLINDERS
28	DRIVE_TRAIN	CHAR(4)	DRIVE TRAIN TYPE [AWD,4WD,FWD,RWD]
29	FUEL_SYS	CHAR(4)	FUEL SYSTEM CODE: FI =FUEL INJECTION TB =TURBO
30	FUEL_TYPE	CHAR(4)	FUEL TYPE CODE: BF =BIFUEL CN =CNG/LPG DS =DIESEL GS =GAS HE =HYBRID ELECTRIC
31	TRANS_TYPE	CHAR(4)	VEHICLE TRANSMISSION TYPE [AUTO, MAN]
32	VEH_SPEED	NUMBER(3)	VEHICLE SPEED
33	DOT	CHAR(20)	DEPARTMENT OF TRANSPORTATION TIRE IDENTIFIER
34	TIRE_SIZE	CHAR(30)	TIRE SIZE
35	LOC_OF_TIRE	CHAR(4)	LOCATION OF TIRE CODE: FSW =DRIVER SIDE FRONT DSR =DRIVER SIDE REAR FTR =PASSENGER SIDE FRONT PSR =PASSENGER SIDE REAR SPR =SPARE
36	TIRE_FAIL_TYPE	CHAR(4)	TYPE OF TIRE FAILURE CODE: BST =BLISTER BLW =BLOWOUT TTL =CRACK OFR =OUT OF ROUND TSW =PUNCTURE TTR =ROAD HAZARD TSP =TREAD SEPARATION
37	ORIG_EQUIP_YN	CHAR(1)	WAS PART ORIGINAL EQUIPMENT 'Y' OR 'N'
38	MANUF_DT	CHAR(8)	DATE OF MANUFACTURE (YYYYMMDD)
39	SEAT_TYPE	CHAR(4)	TYPE OF CHILD SEAT CODE: B =BOOSTER C =CONVERTIBLE I =INFANT IN =INTEGRATED TD =TODDLER
40	RESTRAINT_TYPE	CHAR(4)	INSTALLATION SYSTEM CODE; A =VEHICLE SAFETY BELT B =LATCH SYSTEM
41	DEALER_NAME	CHAR(40)	DEALER'S NAME
42	DEALER_TEL	CHAR(20)	DEALER'S TELEPHONE NUMBER
43	DEALER_CITY	CHAR(30)	DEALER'S CITY
44	DEALER_STATE	CHAR(2)	DEALER'S STATE CODE
45	DEALER_ZIP	CHAR(10)	DEALER'S ZIPCODE
46	PROD_TYPE	CHAR(4)	PRODUCT TYPE CODE: V =VEHICLE T =TIRES E =EQUIPMENT C =CHILD RESTRAINT
47	REPAIRED_YN	CHAR(1)	WAS DEFECTIVE TIRE REPAIRED 'Y' OR 'N'
48	MEDICAL_ATTN	CHAR(1)	WAS MEDICAL ATTENTION REQUIRED 'Y' OR 'N'
49	VEHICLES_TOWED_YN	CHAR(1)	WAS VEHICLE TOWED 'Y' OR 'N'

Recall File Characteristics:

Field#	Name	Type/Size	Description
1	RECORD_ID	NUMBER(9)	RUNNING SEQUENCE NUMBER, WHICH UNIQUELY IDENTIFIES THE RECORD.
2	CAMPNO	CHAR(12)	NHTSA CAMPAIGN NUMBER
3	MAKETXT	CHAR(25)	VEHICLE/EQUIPMENT MAKE
4	MODELTXT	CHAR(256)	VEHICLE/EQUIPMENT MODEL
5	YEARTXT	CHAR(4)	MODEL YEAR, 9999 IF UNKNOWN or N/A
6	MFGCAMPNO	CHAR(20)	MFR CAMPAIGN NUMBER
7	COMPNAME	CHAR(256)	COMPONENT DESCRIPTION
8	MFGNAME	CHAR(40)	MANUFACTURER THAT FILED DEFECT/NONCOMPLIANCE REPORT
9	BGMAN	CHAR(8)	BEGIN DATE OF MANUFACTURING
10	ENDMAN	CHAR(8)	END DATE OF MANUFACTURING
11	RCLTYPECD	CHAR(4)	VEHICLE, EQUIPMENT OR TIRE REPORT
12	POTAFF	NUMBER(9)	POTENTIAL NUMBER OF UNITS AFFECTED
13	ODATE	CHAR(8)	DATE OWNER NOTIFIED BY MFR
14	INFLUENCED_BY	CHAR(4)	RECALL INITIATOR (MFR/OVSC/ODI)
15	MFGTXT	CHAR(40)	MANUFACTURERS OF RECALLED VEHICLES/PRODUCTS
16	RCDATE	CHAR(8)	REPORT RECEIVED DATE
17	DATEA	CHAR(8)	RECORD CREATION DATE
18	RPNO	CHAR(3)	REGULATION PART NUMBER
19	FMVSS	CHAR(10)	FEDERAL MOTOR VEHICLE SAFETY STANDARD NUMBER
20	DESC_DEFECT	CHAR(2000)	DEFECT SUMMARY
21	CONEQUENCE_DEFECT	CHAR(2000)	CONSEQUENCE SUMMARY
22	CORRECTIVE_ACTION	CHAR(2000)	CORRECTIVE SUMMARY
23	NOTES	CHAR(2000)	RECALL NOTES
24	RCL_CMPT_ID	CHAR(27)	NUMBER THAT UNIQUELY IDENTIFIES A RECALLED COMPONENT.

Note: The above two schema, mentioned in Section 2.6, are taken from NHTSA official documentation (http://www-odi.nhtsa.dot.gov/downloads/)

References

1. M. Culver. (2015, Jul 29). *Average Age of Light Vehicles in the U.S. Rises Slightly in 2015 to 11.5 years* [Online]. Available: http://press.ihs.com/press-release/automotive/average-age-light-vehicles-us-rises-slightly-2015-115-years-ihs-reports

2. NHTSA. (2016, Feb 12). *Takata Airbag Recalls* [Online]. Available: http://www.safercar.gov/rs/takata/takatalist.html

3. P. Williams. (2015, Sep 17). *GM to Appoint Monitor, Pay $900M Fine Over Faulty Ignition Switches* [Online]. Available: http://www.nbcnews.com/storyline/gm-recall/gm-appoint-monitor-pay-900m-fine-over-faulty-ignition-switches-n429136

4. T. Abeel, (2012, Jul 10). *Java Machine Learning Library* [Online]. Available: http://java-ml.sourceforge.net/

5. *Lesson: JDBC Basics* (2016, Apr 27). [Online]. Available: https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

6. *Office of Defects Investigation* (2016, May 4). [Online]. Available: http://www-odi.nhtsa.dot.gov/downloads/

7. *Datumbox Machine Learning Framework* (2016, May 4). [Online]. Available: http://www.datumbox.com/machine-learning-framework/

8. *Classifier4J* (2016, May 4). [Online]. Available: http://classifier4j.sourceforge.net/