TEXAS
STATE CIS
DEPT

DATA MINING FINAL

Detecting Insurance Fraud
Through Linear Regression

2020

RASHA SALEH, KAMI VELARDE
ANAM NAWAB, PEDRO
BATDORF, STEPHEN KIM

EXECUTIVE SUMMARY

This proposal is an overview of our evidence-based analysis for detecting insurance fraud through building statistical models such as linear regression, logistic regression, decision trees, and a confusion matrix. We discussed and analyzed several of our findings and ended up deciding on using linear regression as it was the best fit for our particular dataset. Our team members including Rasha Saleh, Kami Velarde, Anam Nawab, Stephen Kim, and Pedro Batdorf all have similar backgrounds as computer information systems majors at Texas State University with the exception of Stephen who is studying business marketing.

As requested, we utilized our background knowledge from our data mining class to analyze our dataset and produce our findings. Starting with R-studio, we used logistic regression to calculate the fraudulence percentages and moved on to Spyder with python to gather the precision, accuracy, and specificity scores in order to draw conclusions about our dataset and ultimately detect any signs of fraud.

Our desired outcome for this analysis project was to detect the probability of fraud in our auto-insurance dataset as well as the relationship between age, injury claims, and witnesses. The idea of how fraud relates to these variables was important for our team to analyze and test. Our proposed solution was to utilize linear regression and decision trees to get the desired outcome.

After utilizing the auto-insurance dataset we are confident enough to prove that there is a reasonable amount of fraud happening inside the auto-insurance community. By implementing our statistical models, our team was able to magnify the occurring issues within the association of fraud ratios to erode the fabric of less secure systems.

PROBLEM & DATA DESCRIPTION

While insurance fraud is a growing problem each year, our recent technology has gained the ability to detect it early on by analyzing datasets with data mining. With data analysis, we are able to recognize patterns, discrepancies, and correlations within datasets to determine whether or not insurance fraud has been committed. If our strategies prove to be successful, insurance companies will be able to save money each year.

Given our auto-insurance dataset, we are able to pinpoint the specific percentages of fraudulence within the values. Our initial investigation began with logistic regression in R-Studio to determine the percentages of fraud within our dataset. We are able to analyze that age has a 22.5% effect on fraud. Out of the claims with injury, 19% reported have fraud reported and 24% of claims that had witnesses have fraud. We then moved to Python to split our dataset into test (x) and train (y) which allowed us to create a confusion matrix which gives the specificity of (99%) along with the precision (74.33%) and accuracy (74.33%) scores.

DESCRIBING & VISUALIZING DATA

Our insurance claims dataset features 40 columns of information regarding auto insurance claims such as months-as-customers, age, policy deductible, as well as information about the incident occured. With the problem we chose to solve and the dataset that went along with that, preprocessing was not involved. Initially, our analysis was in R-studio for the fraud percentages but ultimately switched over to python in order for us to create a decision tree.

74.33%
precision & accuracy

99%
specificity

ANALYSIS OF R CODE

Within R-Studio, we started the process by assigning our downloaded dataset to the value "Fraud". Our "Fraud" variable was attached to the file and then a contingency table was returned with the name "fraud_reported". Our table function gathers the frequencies. Next, the column bind function we used will bind the columns we chose out of our dataset. Using multiple independent variables such as age, injury_claim, witnesses, we obtain the lowest AIC value. We assign "log1", "log2", "log3" to determine the lowest AIC value for each independent variable in order to achieve a perfect model. For the "num", we are adding three independent variables which we calculated after adding and removing many columns from our dataset. That gave us the lowest AIC value for "injury_claim" as our independent variable with "fraud_reported" being the dependent. We used a linear regression equation to find out how those independent variables affected the dependent variable. (See appendix A for our code)

ANALYSIS OF PYTHON CODE

With the completion of our R-studio portion of analysis, we then move on to Python in Spyder. We started by reading the dataset file into Spyder and assigning it to the value "OSI". Next we replaced our "fraud_reported" column name to "Target" instead. We used the dropna function to clean our dataset and drop the NA values. Next we assigned X to our independent variables of "age", "injury_claim", "witnesses" and our dependent variable, "Target" to Y. After this, we split our data into train and test. We decided on using .3 for the test size because this seemed like the best fit along with "random_state" set to 100. Next, we assigned "clftitan_entropy" to a decision tree classifier function in order to measure the quality of our data that is now split to minimize the effects of data in-similarities and better understand the aspects of the model. Next, we broke down our dataset into a smaller subset and to get a better decision tree we used a decision tree classifier. We used sklearn to create a decision tree (See appendix D). After this we printed our confusion matrix, accuracy, and the report (See appendix C). From this we calculated the specificity (99%) precision (74.33%) and accuracy (74.33%) scores. (See appendix B for our code)

SUMMARY

Insurance fraud has been a recurring issue, and insurance companies lose thousands of dollars of revenue each year. To detect fraud we used the age, injury claim, and witnesses as our independent variables. We picked those variables because age will let us know what age group will be more likely to commit fraud in the future. We used the variable injury claim to detect fraud, because it let us know if anyone got hurt. In most car accidents injuries happen, so if there's a accident that reports no injury, it should be flagged for review to see if other data like a police report or witnesses were reported. We used the variable witnesses because when witnesses are reported, they have actual people that saw the accident occur which means that fraud is less likely and if there's no witnesses reported it should be flagged for further review to see if fraud is reported.

To summarize our findings, we ended up creating a contingency table that exposed our values in a fraud report format. From this, we were able to obtain our lowest AIC value which in turn, gave us a perfect model to reflect our outcome. We have discovered that roughly a quarter of fraud has infiltrated the auto-insurance system. As far as variables go, we ended up using the age, injury claim, and witnesses as our independent and fraud reported as our dependent. By doing this, the comparison between those two values becomes a more simple process.

Moving forward, we shifted our analysis to Spyder using python to clean our dataset and eventually split it into train and test values. Doing this will ensure that we can better understand our data and overall simplify the model. Next, we created a decision tree which helped to predict our target value. Our decision tree helped us to conclude the correct entropy number. As the decision tree classifier broke down our data, we were able to print our confusion matrix and calculate the precision, accuracy, and specificity scores. Our precision score of 74.33% tells us how accurate the model was, which was almost 75% accurate. The specificity score of 99% tells us that our model is correctly categorizing our data samples. Along with these two values, the accuracy score of 74.33% is in the acceptable range for how well our model functions as a predictor.

APPENDIX A
R CODE

READ FILE/ASSIGNING VALUES

```
Fraud<-read.csv(file="C:/Users/rasha/Desktop/Big Data/insurance.csv", sep=",", header=TRUE)
attach(Fraud
table(fraud_reported)
```

LOGISTIC REGRESSION

```
o=cor(cbind( age, injury_claim,witnesses,number_of_vehicles_involved,fraud_reported))
num<- glm(fraud_reported~age+injury_claim+witnesses, family= binomial(link ="logit" ))
log1<- glm(fraud_reported~age, family= binomial(link ="logit" ))
log2<- glm(fraud_reported~injury_claim, family= binomial(link ="logit" ))
log3<- glm(fraud_reported~witnesses, family= binomial(link ="logit" ))
```

CALCULATIONS - FRAUD VALUES

```
A<-1/(1+exp(-1*(-1.27190+  0.10348)))
# 0.2371407 so 24% of claims that have witnesses have Fraud
A<-1/(1+exp(-1*(-1.445e+00+  4.297e-05)))
# 0.1907789 so 19% of claims with injury reported have fraud
A<-1/(1+exp(-1*(-1.234529+  0.003072)))
# 0.2259265  so the age has 22.5% effect on fraud
```

PYTHON CODE

```
OSI =pd.read_csv("C:/Users/abcd/Desktop/Fall 2020/QMST/insurance_claims.csv",sep=",")
OSI['Target']=OSI['fraud_reported']*1
Z=OSI[['age','injury_claim','witnesses','Target']].dropna()
X = Z[['age','injury_claim','witnesses']]
Y = Z[['Target']]
```

APPENDIX B
PYTHON CODE

SPLITTING DATA

```
X_train, X_test, y_train, y_test = train_test_split(  X, Y, test_size = 0.3, random_state = 100)

clftitan_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100, max_depth
= 3, min_samples_leaf = 5)

clftitan_entropy.fit(X_train, y_train)
y_pred = clftitan_entropy.predict(X_test)
```

CREATING DECISION TREE

```
from sklearn.tree import plot_tree
plot_tree(clftitan_entropy,filled=True,rounded=True, feature_names = feature_cols,class_names=['X','NX'], fontsize=8)
```

CALCULATIONS - CONFUSION MATRIX/ACCURACY

```
print("confusion Matrix : ",
      Confusion_matrix(y_test, y_pred))
print("Accuracy : ",
      Accuracy_score(y_test, y_pred))
print("Report : ",
      classification_report(y_test, y_pred))
```

APPENDIX C
CONFUSION MATRIX

```
In [15]: print("Confusion Matrix: ",
    ...:       confusion_matrix(y_test, y_pred)) # Specificity: 222/(222 + 3)= 97%
    ...: print ("Accuracy : ",
    ...:        accuracy_score(y_test,y_pred)*100) # Accuracy: 74%
    ...: print("Report : ",
    ...:       classification_report(y_test, y_pred))# Recall: Weighted Average = 65%
Confusion Matrix:  [[222   3]
 [ 74   1]]
Accuracy :  74.33333333333333
Report :                precision    recall  f1-score   support

           N       0.75      0.99      0.85       225
           Y       0.25      0.01      0.03        75

    accuracy                           0.74       300
   macro avg       0.50      0.50      0.44       300
weighted avg       0.62      0.74      0.65       300
```

APPENDIX D
DECISION TREE