**Insurance Fraud Data Analysis and Classification**

**By Jabed Miah**

**Introduction**

There are many other Kaggle projects that have investigated insurance fraud using the dataset mentioned later, but they are severely lacking. The results provided by the authors are very poor and at times are the worst when it comes to fraud classification. Moreover, the conclusions are non-existent: [9, 10]. There’s no effort towards understanding the domain knowledge of the insurance fraud dataset: they just take a model and apply it to the data—sometimes they will use ten or more models a la [10], but how can they use a model without understanding how the model works. There is a very bare minimum explanation of why the authors in the Kaggle notebooks choose to do things the way they do.

My project hopes to address these issues: I will explore a singular Standard Vector Machine (SVM) model. I will elucidate on results, conclusions, and model parameters. The goal for this project is to have a well-written deeply explored analysis of the data with a robust discussion of metrics and a working model.

**Preprocessing and Investigation**

The dataset used for the analysis is from Kaggle, Auto Insurance Claims Data ([bit.ly/3JwvwTk](https://bit.ly/3JwvwTk)). It is not a “perfect” dataset as there is no description of what each column means; currently, the dataset consists of 40 columns and 1,000 rows. Tools I will use are popular modules from Python such as pandas, NumPy, seaborn, scikit-learn, matplotlib, mlextend, and imbalanced-learning. The code for the project is available in a Google Collaboratory Jupyter notebook ([bit.ly/408gadf](https://bit.ly/408gadf)); however, it is not as organized as this document. Also, some codes for plotting graphs are left out but are available in the notebook.

The document is structured in this manner: variables, columns, and functions will be italicized. In hopes of preventing confusion between variables and functions: functions include “()” parentheticals. Code will be included and if output is needed it will immediately follow. Long URLs that take up space are shortened using bit.ly. Output from the code that takes up a large amount of vertical real estate are snipped and horizontally stitched.

Before I begin the analysis, the best thing to do is to preprocess the data for null or missing values as per machine learning purposes as well as analysis. In Fig. 1, there are null values for the columns: *collision\_type*, *property\_damage*, *police\_report\_avaliable*, *\_c39*; the last column isn’t shown in the output since it has been dropped beforehand after seeing that all it offered is null values.

insurance\_claims\_df.isna().sum()

A picture containing text

Description automatically generated

**Figure 1:** Stitched output of all the summed null values in each column.

The column *police\_report\_avaliable* has missing data where each one is inputted as “?”—prior to the imputation I have converted each “?” value to NaN using Numpy’s *replace()* function.

insurance\_claims\_df = insurance\_claims\_df.drop('\_c39', axis=1)

import numpy as np

insurance\_claims\_df = insurance\_claims\_df.replace('?', np.NaN)

These missing values are imputed using the mode of the rows and after this step the data can now be analyzed since all the null values have been removed; however, I performed an analysis using the rows without null values before executing any of the code mentioned in this section—just how my brain tends to work.

insurance\_claims\_df = insurance\_claims\_df.fillna(insurance\_claims\_df.mode().iloc[0])

The Kaggle data also did not include the types of values that are given to each categorical column: the code below filters the data types of each column in the data frame by the parameter “object” and prints every unique value.

cat\_cols = min\_df.select\_dtypes(include=['object'])

for col in cat\_cols:

print(f"{col}: {cat\_cols[col].unique()}")

In this instance, *min\_df* is a modified version of the insurance dataset: the modifications will be mentioned later. Table. 1 gives an overview of each column’s unique options where each entry is separated by a space because of shear laziness on my part—certain columns are left out which will be explained later.

**Table 1**: Features that have multiple unique entries for the data insurance\_claims dataset.

|  |  |
| --- | --- |
| Features | Unique Entries |
| policy\_csl | '250/500' '100/300' '500/1000' |
| insured\_sex | 'MALE' 'FEMALE' |
| insured\_education\_level | 'MD' 'PhD' 'Associate' 'Masters' 'High School' 'College' 'JD' |
| insured\_occupation | 'craft-repair' 'machine-op-inspct' 'sales' 'armed-forces' 'tech-support'  'prof-specialty' 'other-service' 'priv-house-serv' 'exec-managerial'  'protective-serv' 'transport-moving' 'handlers-cleaners' 'adm-clerical'  'farming-fishing' |
| insured\_hobbies | 'sleeping' 'reading' 'board-games' 'bungie-jumping' 'base-jumping' 'golf'  'camping' 'dancing' 'skydiving' 'movies' 'hiking' 'yachting' 'paintball'  'chess' 'kayaking' 'polo' 'basketball' 'video-games' 'cross-fit'  'exercise' |
| insured\_relationship | 'husband' 'other-relative' 'own-child' 'unmarried' 'wife' 'not-in-family' |
| incident\_type | 'Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'  'Parked Car' |
| collision\_type | 'Side Collision' 'Rear Collision' 'Front Collision' |
| incident\_severity | 'Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage' |
| authorities\_contacted | 'Police' 'None' 'Fire' 'Other' 'Ambulance' |
| property\_damage | 'YES' 'NO' |
| police\_report\_available | 'YES' 'NO' |
| fraud\_reported | Y' 'N' |

Before starting a classification project, checking the balance of the datasets does not hurt: the labels in the dataset are binary for *fraud\_reported* which contains a value of Y and N—these will be later encoded to 0 and 1 for machine learning purposes. There are 247 fraudulent cases and 753 non-fraudulent cases—a bar graph is shown in Fig. 2. A clear imbalance is visible which can be a problem depending on the type of model that is used for the machine learning section, but that is not the only obstacle: the data is meager and may not be the best for machine learning later the data is resampled to increase the minority class.

Logo

Description automatically generated with medium confidence

**Figure 2**: The amount of data per class, Fraudlent being the minority class and Non-Fradulent being the majority class.

Let’s get some domain knowledge and then analyze the dataset to find some meaningful insights before I get ahead of myself.

**Auto-Insurance Domain Knowledge**

The goal for this section is to fill gaps in my knowledge regarding the dataset. The *policy\_csl* and *umbrella\_limit* features are features that I did not understand: the formatting for the former are numbers separated with slashes. A bit of research has gone a long way to build an understanding of this topic—the abbreviation CSL stands for Combined Single Limits where the insurance policy limits the coverage of all components of a claim to a single dollar amount. CSL maxes out the amount of money that is paid out which covers bodily injury and property damage; interestingly, the limit splits between involved parties of the accident or claim.

If let’s say a CSL policy is 500/500, the first 500 before the slash is if an accident is caused by you the insurance will pay out up to $500,000 per person to anyone that you injured while the second 500 is the total per accident payout which is limited to a maximum of $500,000. However, additional coverage may need to be added that is required by the state or insurer.

If someone were to go over the maximum provided by their coverage, the umbrella limit would kick in. The limit in-question has no applicability in the coverage of bodily injuries or property damage. But this is not the only functionality of the umbrella limit: umbrella insurance can cover legal costs in cases of libel or slander, liabilities when traveling overseas, and expenses that are related to one self’s psychological harm and mental anguish.

Also, the *policy\_deductable* is another feature that has eluded me since I am not familiar with automobile insurance deductibles: it looks like policy deductibles are paid out of pocket on a claim which would be the basic definition. However, there is a bit more to it: a higher deductible would mean more is paid out of pocket, but the car insurance rate is lower, and a lower deductible would mean the car insurance rate is higher, but less is paid out of pocket—the inverse of the former.

I believe that another feature can be engineered after considering the expenses covered by the insurance provider, but due to time constraints that is something I will leave for the future if I do change my mind.

**Visualizing the Dataset in Python**

The gender binary split brings into questions: How many of the customers that are Male or Female commit insurance fraud and how many do not? Customers that are female who commit insurance fraud are 126 and their Male counterparts are 121: this clearly shows that females commit insurance fraud at a slightly higher amount in this specific dataset. However, if we look at it the other way in-terms of the ones who do not commit insurance fraud, we find that most females about 411 of them do not while 342 males trail behind. A bar graph of the results are shown in Fig. 2.

print(f"Fradulent Females: {((insurance\_claims\_df['fraud\_reported'] == 'Y') & (insurance\_claims\_df['insured\_sex'] == 'FEMALE')).sum()} \n"

f"Non-fradulent Females: {((insurance\_claims\_df['fraud\_reported'] == 'N') & (insurance\_claims\_df['insured\_sex'] == 'FEMALE')).sum()}\n"

f"Fradulent Males: {((insurance\_claims\_df['fraud\_reported'] == 'Y') & (insurance\_claims\_df['insured\_sex'] == 'MALE')).sum()}\n"

f"Non-fradulent Males: {((insurance\_claims\_df['fraud\_reported'] == 'N') & (insurance\_claims\_df['insured\_sex'] == 'MALE')).sum()}")

Chart, bar chart

Description automatically generated

**Figure 3**: Distribution of classes based on gender.

Moreover, the education level feature allows me to show at what level of education customers are most likely or least likely to commit insurance fraud.

education\_levels = insurance\_claims\_df['insured\_education\_level'].unique()

for education in education\_levels:

print(f"{education}: {((insurance\_claims\_df['insured\_education\_level'] == education) & (insurance\_claims\_df['fraud\_reported'] == 'Y')).sum()}")

Using Pandas *unique()* function, I assign the list it returns for the column *insured\_education\_level* to the declared variable *education\_levels*: the function returns all the unique values it finds in that specific column. Next, I iterate through the *education\_levels* printing a Boolean expression. The expression in-question looks to each education level and its connection to fraudulent activity being ‘Y’ which is added. Do note that the print line of code is one long line—the word warping has given a portion of the code a newline which would be a Python syntax error.

Chart, bar chart

Description automatically generated

**Figure 4**: Insurance fraud committed based on insured’s level of education.

The horizontal bar graph as shown in Fig. 4 reveals that customers with the education level of Juris Doctor (JD) commit the most amounts of insurance fraud: this data does not however give an idea of why this would be the case. There could be other extraneous factors in play, reusing the same line of code for education levels: we can get a picture of (1) insurance fraud committed based on policy holder’s state, (2) insurance fraud committed based on the customer’s vehicle, and (3) insurance fraud committed based on customer age.

Icon

Description automatically generated

**Figure 5**: Rates of insurance fraud based on the policy holder's state.

Given that there are only three states in the dataset, customers that are from Ohio commit the most insurance fraud compared to customers in Indiana and Illinois as shown in Fig. 5.

In the original dataset, there are errors regarding the names of two car models such as Subaru and Acura where the original spells them mistakenly as “Accura” and “Suburu”. I have chosen to not change those entries in the data instead the names are changed in the plot, Fig. 6. There are 14 brands of vehicles that everyone in the dataset uses.

Chart, bar chart

Description automatically generated

**Figure 6**: Brands of vehicles driven by fraudulent drivers.

Mercedes and Ford drivers are neck-in-neck while Jeep drivers are the lowest in-terms of insurance fraud. Next, if we look at the range of ages in Fig. 7, the dataset goes up to a maximum of age 61. The minimum age is 19 years old.

Chart, histogram

Description automatically generated

**Figure 7**: Age range of customers that commit insurance fraud.

People who are the age of 41 commit the most insurance fraud with 16 total cases.

**Feature Selection using Seaborn**

I will now use seaborn’s heatmap feature to see the correlation between features to assess which features can be dropped: the parameter fmt=’.2g’ will move the decimal to two significant figures.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize = (18, 12))

correlation = insurance\_claims\_df.corr()

sns.heatmap(data=correlation, annot=True, fmt ='.2g', linewidth=2)

plt.show()

Chart, treemap chart

Description automatically generated

**Figure 8**: Heat map of features and their correlation.

There are a few highly correlated features such as *age* and *months\_as\_customer*, *total\_claim\_amount* and *vehicle\_claim*, *total\_claim\_amount* and *property\_claim*, and *total\_claim\_amount* and *injury\_claim*. I will drop the column *total\_claim\_amount* since it is the addition of all the other claim amounts, and I will also drop *age*. An issue with having highly correlated features is that they may become a problem for the linear model I plan to use; however, the dropping of highly correlated features is not as a cut and dry. I will drop features that I believe to be useless—or too complex to deal with as of now—in the model such as *auto\_make*, *auto\_model*, *incident\_city*, *incident\_location*, *policy\_state*, *policy\_bind\_date*, *incident\_date*, *incident\_state*, *policy\_number*, and *insured\_zip*.

drop = ['auto\_make', 'auto\_model', 'incident\_city', 'incident\_location',

'auto\_year', 'total\_claim\_amount', 'policy\_state', 'age',

'policy\_bind\_date', 'incident\_date',

'incident\_state', 'policy\_number', 'insured\_zip']

min\_df = insurance\_claims\_df.copy()

min\_df.drop(drop, inplace=True, axis=1)

**Encoding Categorical Columns and Scaling Numerical Columns using Scikit-learn’s StandardScaler()**

Prior to encoding the labels, getting all the categorical columns into their own dataframe would help separate categorical and numerical columns. The *select\_dtypes()* comes to the rescue since the datatypes of the categorical columns are all “object” changing the include parameter to it will filter out all the categorical columns: this was performed previously to discover the unique options for each feature. We will just reuse *cat\_cols*. Pandas library comes with a *get\_dummies()* function which will encode the values for the categorical columns to 1’s and 0’s while also creating separate new columns for each option i.e. *insured\_education\_level* would split into *insured\_education\_level\_College*, *insured\_education\_level\_HighSchool* etc.

import pandas as pd

cat\_cols = pd.get\_dummies(cat\_cols, drop\_first=True)

Furthermore, scaling the numerical columns is an important step: if this isn’t performed, the model will be biased towards certain features—scaling makes this bias less inequitable and brings every feature down to the same level. Thankfully, scikit-learn is packaged with a scaler: the StandardScaler() class which with its *fit\_transform()* function that takes in a dataframe will fit as well as transform the data. There are functions that perform these two operations separately.

from sklearn.preprocessing import StandardScaler

# Getting all the numerical columns.

num\_cols = min\_df.select\_dtypes(include=['int64'])

# Scaling all the numerical columns

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(num\_cols)

scaled\_num\_df = pd.DataFrame(data = scaled\_data, columns = num\_cols.columns)

scaled\_num\_df

Maybe, there should be a function that performs scaling on numerical columns as well as encoding on categorical columns called *scaling\_encoder()* since that isn’t the case; the two dataframes here will need to be combined using Pandas’s *concat()* function.

combined\_df = pd.concat([scaled\_num\_df, cat\_cols], axis=1)

**Pair-Plotting using Seaborn**

A pair-plot gives a good idea of the relationship between features, notice that in Fig. 9 there is no linear separability between features. Also, there’s no visible relationship or trend between the selection of features. I’ve limited the number of features to five: it takes a very long time for plots like this to be produced. The combined dataframe consists of 73 columns where the extra number of columns are dummy columns.

A picture containing calendar

Description automatically generated

**Figure 9**: Pair-plot of select features.

**Fitting the Machine Learning Model**

A good way to make sure things work out as planned including making the model is robust is to split the data into a train and a test set. The split is performed using scikit-learn’s *train\_test\_split()* functionality.

from sklearn.model\_selection import train\_test\_split

y = combined\_df['fraud\_reported\_Y'].to\_numpy()

X = combined\_df.drop('fraud\_reported\_Y', axis=1).to\_numpy()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1, stratify=y)

The target values are set to the *y* variable, and the features are assigned to the *X* variable. They are converted to NumPy arrays so that they can be accepted by the *train\_test\_split()* function. Parameters set for the *train\_test\_split()* function make it so the test set of data is 30% while stratify preserves the proportion of the distribution of the data. Given the apparent imbalance of the data, it is necessary to use an algorithm that takes this property into account. In this case, I am considering using a weighted Standard Vector Machine (SVM) with a heuristic weighting parameter that does not require manual tweaking. The data is also fitted using the Standard Vector Classifier.

from sklearn.svm import SVC

svc = SVC(gamma='scale', class\_weight="balanced")

svc.fit(X\_train, y\_train)

y\_pred = svc.predict(X\_test)

Now, all that is left is to calculate the results of the model using scikit-learn once again: the results of the code are shown in Fig. 10.

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

svc\_train\_acc = accuracy\_score(y\_train, svc.predict(X\_train))

svc\_test\_acc = accuracy\_score(y\_test, y\_pred)

print(f"Training accuracy of Support Vector Classifier is : {svc\_train\_acc}")

print(f"Test accuracy of Support Vector Classifier is : {svc\_test\_acc}")

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

Calendar

Description automatically generated with medium confidence

**Figure 10**: Output of a training accuracy score, a testing accuracy score, a confusion matrix, and a classification report.

It is shown that the classification training accuracy is 94%: the test accuracy however is a lowly 79%. Metrics such as these are not particularly important when the data is imbalanced hence the usage of confusion matrix and classification report. I have graphed the confusion matrix to get a clearer picture of what is going on.

A picture containing chart

Description automatically generated

**Figure 11**: Confusion matrix of the model, plotted using matplotlib and mlxtend.

The model correctly predicted the positive class no fraud 191 times as seen from the True Positive (TP) value: furthermore, the model correctly predicted the negative class fraud 47 times. When we look at the remaining diagonal values, there are 35 occasions where the model predicted no fraud incorrectly and 27 occasions or false positives where the model predicted fraudulent activity incorrectly.

If we take a gander at the classification report, the first class has good results in all the categories. The second class tells a different story with abysmal scores—even though the model is weighted it remains apparent that there is just not enough data for the second class, to remedy this problem resampling the dataset would be the next step.

**Resampling using ADASYN**

The reasoning behind why I chose such a sampling method is because someone else who also did a similar topic but for credit card fraud had shown me the value of such a resampling method with their impressive results. ADASYN instead of copying the same minority data generates synthetic data for examples that are harder to learn which has two advantages: reduction of bias presented by class imbalance and the shifting of the classification boundary to consider difficult examples. Let’s resample the data using the imbalanced learning library.

from imblearn.over\_sampling import ADASYN

ada = ADASYN(random\_state=42)

X\_res, y\_res = ada.fit\_resample(X, y)

print(f"X\_res: {X\_res.shape}, y\_res: {y\_res.shape}")



**Figure 12**: Shape of x\_res and y\_res--showing that they are now balanced.

ADASYN successfully balanced the data set as shown in Fig. 12 which means there is no need to use a weighted SVM instead I will use the normal SVM. I will once again split the dataset: this time without any specific parameters: I seem to run into certain odd errors if I use any parameters. The shape of the split data can be viewed in Fig. 13.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_res, y\_res)

print(f"X\_train: {X\_train.shape}, X\_test: {X\_test.shape}")

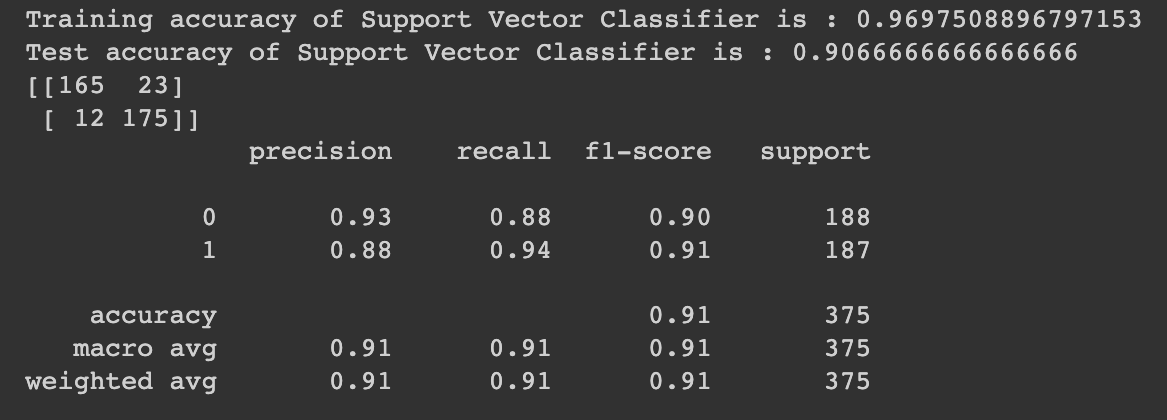
print(f"y\_train: {y\_train.shape}, y\_test: {y\_test.shape}")

A black background with white text

Description automatically generated with low confidence

**Figure 13**: Shape of x\_train, X\_test, y\_train, and y\_test after splitting the data.

Next, I will fit the data onto the model using the same code previously, and the results are shown below.



**Figure 14**: Metrics for the model.

The model performed better in the second class increasing the values of the categories by a huge portion: the same can be said of the test accuracy that has gone from 79% to 90%.

A picture containing treemap chart

Description automatically generated

**Figure 15**: Confusion matrix plot of the model after resampling.

If we look at the new confusion matrix, there is a steep reduction in false positive values. From the precision metric, we can see that the model predicts insurance fraud with 88% certainty. The results are much better than our previous bout with machine learning, but I believe we can do much better by tuning the hyperparameters.

**Tuning SVM Hyperparameters using GridSearchCV**

The GridSearchCV class in the scikit-learn library can help us in attain better results: GridSearchCV is a brute force approach to finding the best hyperparameters for the model. Let’s take a quick overview of SVM’s parameters such as Kernels, C, and Gamma. The primary functionality of kernels is to take low dimensional space and transform it into a higher-dimensional space. As discussed before about the dataset, there is a lack of linear separability between some features and this kernel is useful for such scenarios. C parameter tells the SVM, in-terms of optimization, how much should be avoided when misclassifying each training example. If C is a large value, the optimizer chooses a smaller-margin hyperplane: the optimizer considers its effectiveness in getting all training points classified correctly. A very small C value results in the optimizer looking for a larger-margin separating hyperplane; consequently, it misclassifies more points. Gamma calculates the plausibility of a line of separation. When gamma is high, the nearby points have a high influence. However, if it is low, far away points are considered.

param\_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}

grid = GridSearchCV(SVC(),param\_grid,refit=True,verbose=2)

grid.fit(X\_train, y\_train)

print(grid.best\_estimator\_)



**Figure 16**: GridSearchCV's best optimized parameters for the model.

After applying the algorithm, GridSearchCV finds the best hyperparameters to be *C=10* and *gamma=0.1* for the SVM model. I will now predict the new results using the tuned model.

grid\_predictions = grid.predict(X\_test)

print(confusion\_matrix(y\_test, grid\_predictions))

print(classification\_report(y\_test, grid\_predictions))

Calendar

Description automatically generated

**Figure 17**: Results of the model after using the best parameters.

The precision score is much better than what the model achieved in the two previous iterations with a 92% certainty for predicting insurance fraud. A summary table of all three previous results are shown in Table 2.

**Table 2**: Summary of metrics for all three model variants.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Data | Class | Precision | Recall | F1-Score | Accuracy | TP FN  FP TN | |
| Weighted SVM | Imbalanced | 0  1 | 0.88  0.57 | 0.85  0.64 | 0.86  0.60 | 0.79 | 191  27 | 35  47 |
| SVM | Balanced w/ ADASYN | 0  1 | 0.93  0.88 | 0.88  0.94 | 0.91  0.91 | 0.91 | 165  12 | 23  175 |
| SVM | Balanced w/ ADASYN | 0  1 | 0.97  0.92 | 0.91  0.97 | 0.94  0.94 | 0.94 | 172  6 | 16  181 |

Resampling and tuning the hyperparameters worked well to increase the metrics of every iteration.

**Conclusion**

The final SVM model after resampling using ADASYN and then tuning it’s hyperparameters using the brute force approach of GridSearchCV gave exceptional scores across the board for precision, recall, f1-score, and accuracy. The model achieved an accuracy score of 94% compared to the previous two models. The 97% recall score for Class 1 shows that the model is correctly identifying most of the fraudulent transactions: the F1-score of 94% suggest that the model is achieving a good balance between precision and recall. However, the rate of false positives values now stands at a value of six. A common issue with fraud classification is the difficulty or challenge of reducing false positives which leaves open the opportunity to try other models; nonetheless, that opportunity was partially explored in this project. In the notebook, I’ve tried other models such as Naïve Bayes and Decision Tree—perhaps at a later more opportune time those two avenues can be considered fully. A consideration such as that would require an understanding of the specific needs and requirements of a business or organization that may use it. Alas, the project has come to a close.

**References**

1. Kagan, J. (2022, December 7). *Combined single limits: Definition, example, benefits, vs. split*. Investopedia. Retrieved March 17, 2023, from https://www.investopedia.com/terms/c/combined-single-limits.asp#:~:text=Combined%20single%20limits%20are%20a,property%20damage%20in%20an%20incident.
2. Person. (2022, March 30). *What is an umbrella limit in insurance?* WalletHub. Retrieved March 17, 2023, from https://wallethub.com/answers/ci/umbrella-insurance-limits-2140787625/
3. *What does 500/500 insurance mean?* Policygenius. (n.d.). Retrieved March 17, 2023, from https://www.policygenius.com/auto-insurance/what-does-500-500-insurance-mean/
4. *Car insurance deductibles explained*. Progressive. (n.d.). Retrieved March 17, 2023, from https://www.progressive.com/answers/car-insurance-deductible/#:~:text=What%20is%20a%20car%20insurance,have%20a%20car%20insurance%20deductible.
5. Google. (n.d.). *Classification: Accuracy  |  machine learning  |  google developers*. Google. Retrieved March 20, 2023, from https://developers.google.com/machine-learning/crash-course/classification/accuracy
6. *Adasyn: Adaptive Synthetic Sampling Approach for ... - IEEE xplore*. (n.d.). Retrieved March 22, 2023, from https://ieeexplore.ieee.org/document/4633969
7. Nian, R. (2019, December 13). *An introduction to ADASYN (with code!)*. Medium. Retrieved March 22, 2023, from https://medium.com/@ruinian/an-introduction-to-adasyn-with-code-1383a5ece7aa#:~:text=ADASYN%20(Adaptive%20Synthetic)%20is%20an,%E2%80%9Charder%20to%20learn%E2%80%9D%20examples.
8. The\_Data\_Guy. (1962, November 1). *Parameter "stratify" from method "Train\_test\_split" (scikit learn)*. Stack Overflow. Retrieved March 22, 2023, from https://stackoverflow.com/a/72092663/6513998
9. Buntyshah. (2018, October 9). *Insurance fraud claims detection*. Kaggle. Retrieved March 22, 2023, from https://www.kaggle.com/code/buntyshah/insurance-fraud-claims-detection
10. niteshyadav3103. (2022, March 4). *Insurance fraud detection (using 12 models)*. Kaggle. Retrieved March 22, 2023, from https://www.kaggle.com/code/niteshyadav3103/insurance-fraud-detection-using-12-models
11. *SVM hyperparameter tuning using GRIDSEARCHCV*. Velocity Business Solutions Limited. (2020, February 19). Retrieved March 22, 2023, from https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/