Insurance Fraud Detection using Machine Learning

By Unknown unkown

WORD COUNT	1998	TIME SUBMITTED	27-APR-2023 12:32AM
		PAPER ID	99141843

Insurance Fraud Detection Using Machine Learning

<student name>

<University name>

<Course Name>

<Professor Name>

Introduction

The goal of this project, or work is to identify a machine learning model which achieves the best results out of a set of other machine learning models. The goal of this report, or final deliverable is to provide a baseline model that could be used as a reference to understand how well my other models are performing and then actually develop models which outperform this baseline. For detecting fraudulent insurance claims!

The task at hand is a classification task, given a particular sample of dataset, the goal is to predict if the insurance claim made is fraudulent. Hence my target variable within the dataset is: "fraud_reported."

Methodology

The dataset that I've chosen from Kaggle, contains close to 1000 samples within it, and has 39 features. Out of these a total of 19 columns are categorical in nature! In order fit this to any machine learning model we know that we have to perform some data pre-processing, hence I decided to use OneHotEncoder provided within the Sklearn package to encode these features and the resulting set has 1145 features.

This data is also split into 70% - 30% with 705 of the data being used as training set, and rest as Test set.

I used 5 fold cross validation, while determining the best hyperparams for my models. So each model would be trained, and validated against single test train split and then GridSearchCV is performed to find the hyperparams of each of the models that we shall use.

I used the following machine learning algorithms while trying to find the best of all the following:

- NaiveBayes Classifier (Multinomial)
- DecisionTreeClassifier
- Support Vector Machines
- Logistic Regression.

Note: all the Precisions, Recalls and F1-scores are weighted averages for both 0, 1 classes.

Results and Discussion

I will try to obtain results for each of my models and then walk through as to why I might have got such results. Let's start with my baseline model. I shall use the same training data for each of these models, and then test them to obtain some basic results, and then search for the optimal params which yield the best results for each model and note the change in performance.

Baseline Model (Majority Class based Classifier):

The model that I decided to as a reference/baseline model for this task, is Strawman baseline model. Primary reason being the fact that Strawman baseline model is a very simple classifier that is used for predicting target labels based on majority class for binary classification. It counts the classes, and determine the majority class and then predicts the unseen sample or test sample.

To my surprise! I received some promising metrics, although I mainly believe this is because of the fact that it is easier to predict 0's in this case (fraud_not_reported) because majority of the entire samples are 0's.

Strawman Baseline Majority Classifier			
Accuracy	**Precision**	**Recall**	**F1**
0.73	**0.54**	**0.73**	**0.62**

Classification Metrics: Weighted Avg scores for majority baseline.

Although the accuracy is surprisingly decent, the other metrics clearly indicate that is an in efficient way of classification detection of fraud.

NaiveBayes Classifier (Multinomial):

NaiveBayes is a simple machine learning algorithm. It is a probabilistic model that is based on Bayes theorem, it assumes that each feature is conditionally independent provided with label. It is a simple yet very popularly used classifier!

The NaiveBayes Multinomial Classifier is a simple and efficient version of the Bayes theorem, which works well with high-dimensional features! Since we have exactly 39! Features, I decided to use this variant of the NaiveBayes.

Upon fitting this model and performing a single test train split validation we get the following results:

NaiveBayes with Default Params			
Accuracy	**Precision**	**Recall**	**F1**
0.74	**0.69**	**0.74**	**0.65**

Classification Metrics: Weighted Average scores for MultinomialNB

Immediate observation can be made that our model out performs the baseline model and provides close below avg precision, recall and f1 scores! But still way better than those offered by our baseline model.

Now let's see if there's any improvement if we perform GridSearchCV with 5 fold cross validation to find and then fit the optimal version of MultinomialNB. Upon doing so, we get the following metrics:

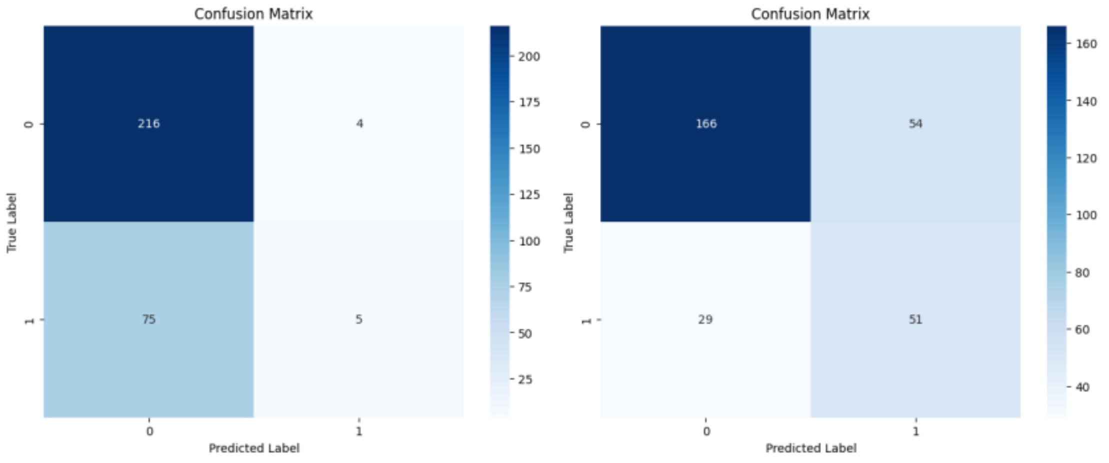
The hyperparams obtained from GridSearchCV are: {'alpha': 0.1, 'fit prior': True}

NaiveBayes with Hyperparams			
Accuracy	**Precision**	**Recall**	**F1**
0.72	**0.75**	**0.72**	**0.73**

Classification Metrics: Weighted Average scores for MultinomialNB with Hyperparams

Clearly! We have better and stable results in terms of all metrics! Although the accuracy did fall a bit, we achieved some decent Precision, Recall and F1 scores for the hyperparams!

Let's see how each of these classified the test samples:



Confusion Matrices for MultinomialNB without (left) and with (right) Hyperparameters.

The MultinomialNB instance without hyperparams fails to identify any fraud detected hence the lower scores for other important metrics like Precision, Recall, and F1. However the one with Hyperparameters does a decent job of this classification!

Logistic Regression:

This is one of the most popular and versatile machine learning algorithm used today! Logistic Regression is a powerful model, which can be used for binary classification, it maps the input features to the probability of the binary outcome.

Logistic Regression can also handle a wide range of input data types, including continuous and categorical. This can be used to handle multi-class classification problems however, we already have a dataset that has been transformed to continuous!

I trained a simple Logistic Regression model, with no params. And here are the results that I obtained for it:

Logistic Regression with Default Parameters			
Accuracy	**Precision**	**Recall**	**F1**
0.74	**0.72**	**0.74**	**0.72**

Classification Metrics: Weighted Average scores for LogisticRegression

The logistic regression model obtains better results than both NaiveBayes and our Strawman baseline!

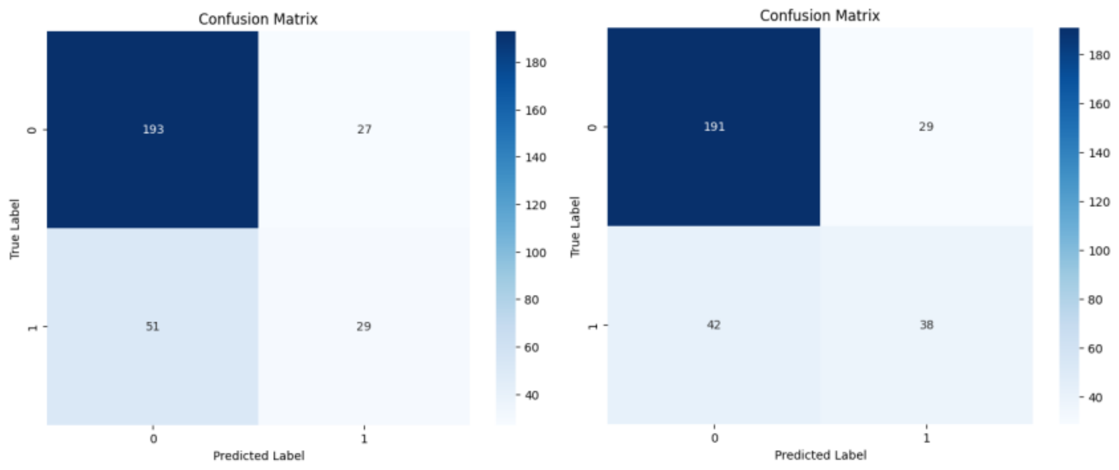
Now that I saw some better results this encouraged me to push for finding the hyperparams with GridSearchCV while performing 5 fold Cross validation here are the results that I obtained. I also manually lowered the threshold and was happy with the results obtained:

Logistic Regression with Hyperparams			
Accuracy	**Precision**	**Recall**	**F1**
0.79	**0.83**	**0.79**	**0.80**

Classification Metrics: Weighted Average scores for LogisticRegression with Hyperparams

These are some very acceptable/good metrics! The higher precision is a great sign, this is because we lowered the threshold so that more samples can be classified as True/1/fraud_reported, and this in turn increases our Precision and Recall. So in order to achieve this we had to manually ***lower the threshold to 0.03! And set the hyperparams to: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}***

We can see a clear distinction in classification within these variants of logistic regression within our confusion matrix:



Confusion matrices for LogisticRegression without (left) and with Hyperparameters.

Support Vector Machines:

The main reason why I decided to use Support Vector Machines, for this binary classification tasks is that they can be used with linear and non-linear decision boundaries. They work by finding the hyperplane.

They are also effective in handling high dimensional data, with this in mind I trained a simple SVM with goal of achieving higher metrics, here are the metrics obtained for basic svm without hyperparams and Cross Validation:

Support Vector Machines Default Params with Probability			
Accuracy	**Precision**	**Recall**	**F1**
0.74	**0.73**	**0.74**	**0.73**

Classification Metrics: Weighted Average scores for SVM without Hyperparams

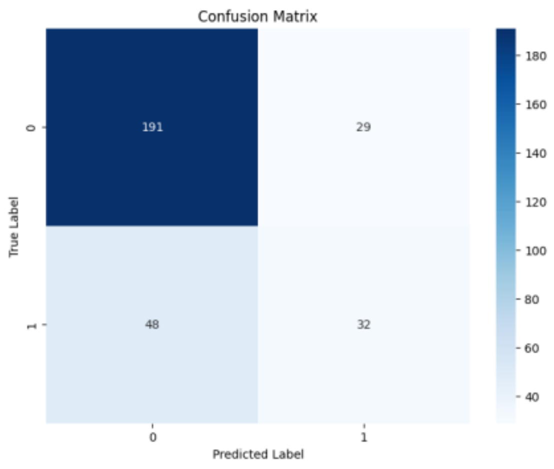
Now that we have the metrics for default model, we can say that it performs better than our baseline, however it still underperforms when compared to the LogisticRegression model.

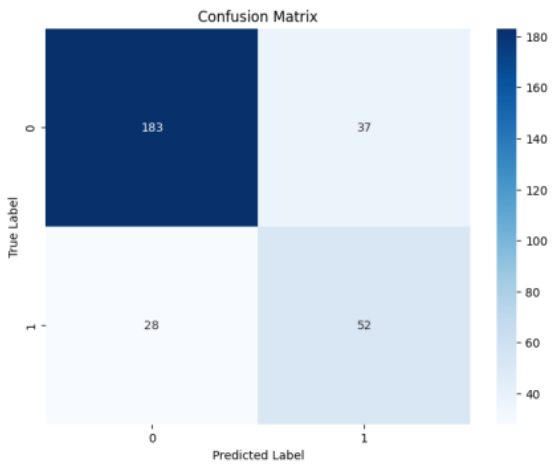
To see if things improve when we find the hyperparams for SVM: ***{'C': 1, 'degree': 2, 'kernel': 'linear'}, while lowering the threshold to 0.299***, here are the metrics obtained:

Support Vector Machine with Hyperparams			
Accuracy	**Precision**	**Recall**	**F1**
0.79	**0.83**	**0.79**	**0.80**

Classification Metrics: Weighted Average scores for SVM with Hyperparams

We can see a clear improvement in performance however these scores are still not better than the balanced/hypertuned Logisitic Regression model. We can see the classification performance within the confusion matrix:





Confusion Matrices for SVMs without (left) and with (right) hyperparams

Decision Tree Classifier:

The primary reasoning for including this classifier is the fact that Decision Trees are simple they are versatile like Logistic Regression they can handle both categorical and continuous data, they can also be used to fit upon unbalanced data, making it an easy choice to include it within our analysis.

They can also prevent bias towards the majority class! This is very important in our dataset as much of the samples are actually 0/negative/fraud_not_reported.

With that being said, I trained a new instance of a Decision Tree without any Hyperparameters. And here are the metrics that I obtained:

DecisionTree Classifier with Default Params			
Accuracy	**Precision**	**Recall**	**F1**
0.81	**0.80**	**0.81**	**0.80**

Classification Metrics: Weighted Average scores for DecisionTree Classifier without Hyperparams

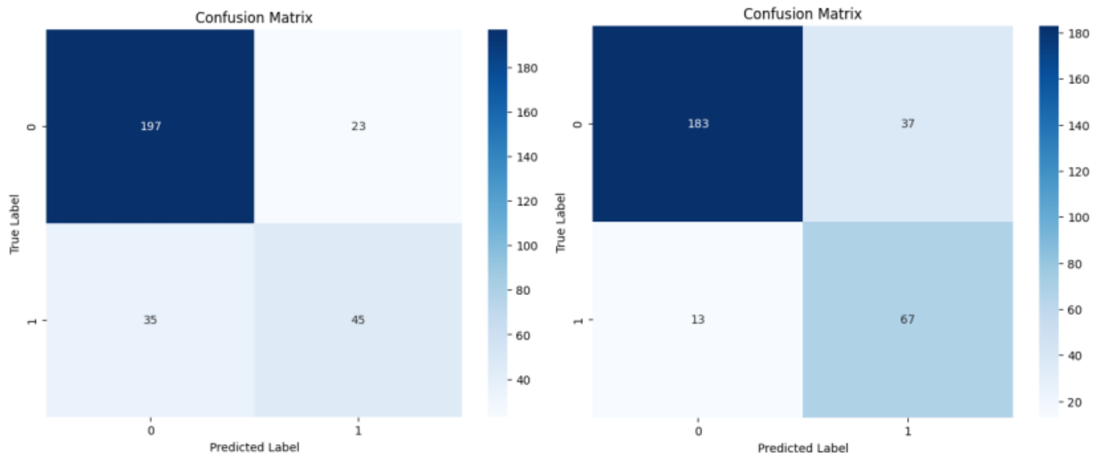
These are great scores! When compared to our baseline and other models, DecisionTreeClassifier did a great job when compared to other models as per the reason I mentioned earlier, they can prevent bias towards negative class, which is the reason why we are getting decent but not exceptional scores the lack for more positive samples within our dataset.

Let's see if we can push these metrics even so slightly if possible by performing GridSearchCV to find the Hyperparameters, and then lowering the threshold manually to see if it helps. Here are the ***Hyperparameters: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 2} and threshold that gives us the best result: 0.2***

DecisionTree Classifier with Hyperparameters			
Accuracy	**Precision**	**Recall**	**F1**
0.83	**0.86**	**0.83**	**0.84**

Classification Metrics: Weighted Average scores for DecisionTree Classifier without Hyperparams

Lets take a look at their classification of each label with and without Hyperparameters:



Confusion Matrices for DecisionTreeClassifier without (left) and with (right) hyperparameters.

I finally am satisfied with the results obtained! To be honest, the dataset lacks data, as in it only has 1000 samples, out of which we use only 700 of these for our training, which causes some of the models to clearly under fit! Hence the lower scores. However, we can see great and well balanced scores of Accuracy, Precision, Recall and F1.

So if we were to summarize our findings, we'd have the following table of results for each model.

Results						
Model	**Cross Validation**	**Hyperparameters**	**Accuracy**	**Precision**	**Recall**	**F1**
Strawman Baseline	**None**	**None**	**0.73**	**0.54**	**0.73**	**0.62**
NB	**None**	**None**	**0.74**	**0.69**	**0.74**	**0.65**
NB	**5 Fold**	**True**	**0.72**	**0.75**	**0.72**	**0.73**
LR	**None**	**None**	**0.74**	**0.72**	**0.74**	**0.72**
LR	**5 Fold**	**True**	**0.79**	**0.83**	**0.79**	**0.80**

SVM	**None**	**None**	**0.74**	**0.73**	**0.74**	**0.73**
SVM	**5 Fold**	**True**	**0.79**	**0.83**	**0.79**	**0.80**
DT	**None**	**None**	**0.83**	**0.80**	**0.81**	**0.80**
DT	**5 Fold**	**True**	**0.83**	**0.86**	**0.83**	**0.84**

Table: Comprising results of all the models with and without CV + Hyperparameters, and Threshold

Conclusion

The primary thing that I received help with in terms of taking up this project is having to think like a machine learning engineer, the ability to experiment, make mistakes roll back and then better results, this gave me some hands on experience to work on some real dataset. I personally feel that I've gained some good practice as to what should I look for within the dataset, in order then select a model for the task at hand.

In order to improve the performance of this model, we could possibly collect more samples in the feature format we have. This would help us prevent under fit some of the models that did not work to their truest potential.

Finally, in order to meet the defined objective of my proposal I find Decision Tree, with Hyperparameters and Lowered threshold after 5 fold cross validation to be the best fit for this task and this particular dataset, with an accuracy score of 0.83. Considering the fact that we have such high count of features and still being able to get such results is quite good. I am happy with the result obtained.

References

ARPAN. (2022). Insurance Fraud Detection [Dataset]. Indian Institute of Management Calcutta & Kaggle. Retrieved from Insurance Fraud Detection | Kaggle

Insurance Fraud Detection using Machine Learning

ORIGINALITY REPORT

1%
SIMILARITY INDEX

PRIMARY SOURCES

1	www.matf.bg.ac.rs Internet	12 words — 1%
2	www.coursehero.com Internet	10 words — < 1%

EXCLUDE QUOTES	OFF	EXCLUDE SOURCES	OFF
EXCLUDE BIBLIOGRAPHY	OFF	EXCLUDE MATCHES	OFF