CS 6190: Probabilistic Machine Learning Spring 2024

Final Report: Kurt Namini

The implementation can be found here.

1 Introduction

I'm interested in the applications of probabilistic learning techniques in the insurance industry, which in the past has been dominated my more traditional, or frequentist, approaches. There are a number of different areas that might be applicable, however I am curious about fraud detection for auto insurance claims. This project aims to do that through the use of probabilistic models such as Bayesian Logistic/Probit Regression, Bayesian Networks, etc. Additionally, I want to establish a baseline on performance with basic Machine Learning techniques to further determine whether the Bayesian approach is better or worse than more typical Machine Learning models.

2 Motivation

This is an important question for companies that provide insurance where fraud is common, such as auto, health, and worker's comp insurance. In recent cases, the Utah Insurance Department reported that they have "investigated and prosecuted cases identifying over \$80 million in losses to victims." By leveraging this sort of analysis, an insurer may better detect fraudulent claims, which will help decrease avoidable losses.

3 Data

The dataset used is a collection of auto insurance claims from 1994 to 1996, which was obtained from Kaggle: Vehicle Insurance Claim Fraud Detection. The features include data on the date of an accident and the resulting claim, vehicle information, personal information of the claim filer, and insurance policy information of the claim filer. The original dataset has a total of 33 features, including the target label, and there is no missing data.

3.1 Data Preparation

As is required by learning models, the data was processed to convert all features into numerical values. Numerical features were standardized by converting the values into standard Gaussian values. For non-numerical data, two approaches were used: ordinal encoding and one-hot encoding.

3.1.1 Ordinal Encoding

Ordinal encoding was used for two kind of variables in the data preparation stage. The first is date feature., which include the month, the week of that month, and the day of that week for the date when the accident occured, and the date when the claim was filed. The second is a collection of features that would otherwise be considered continuous, but were collected as discretized categories with both uneven and even intervals. These features include the age of the claimee's vehicle, the price of that vehicle, how many vehicles the claimee owns, the number of days between the start of the insurance policy and the accident and claim, etc.

In both of these cases, I used ordinal encoding starting with zero to convert into numerical features. The reason I used ordinal encoding instead of one-hot encoding here is that both of the feature cases mentioned above have a natural ordering. This makes sense for the date features, but is less intuitive for the uneven discretized continuous features. At this point, I have no other alternative to encode the discretized features in a way that accounts for the uneven intervals between categories.

3.1.2 One-Hot Encoding

I used one-hot encoding for features that are naturally categorical, with no intrinsic ordering. These features include the vehicle make and type, policy type and base policy type, marital status, insurance agent type, accident area type, etc.

3.1.3 Processed Data

To complete processing the original dataset, various string binary features were converted into numerical binary features(0,1). Following this step, all features were converted to floating-point datatype. The resulting dataset has 66 total features and 15,420 rows. Roughly 6% of the claims in the dataset are labeled as fraudulent, a very small proportion.

4 Train-Test Split

With the extremely low percentage of fraudulent claims, a typical procedure for generating train-test splits in the data did not allow for proper model training. Specifically, the models learned to predict only non-fraudulent claims, and by doing so achieve a roughly 94% prediction accuracy on the test data. To handle this, I implemented my own train-test split function. This function generated the training data by selecting a fraction (function argument) of all fraudulent cases to include in the training data. This was done to increase the fraction of fraudulent cases in the training set above 6%. By doing this, the trained models will actually learn to predict fraud, instead of just achieving a high test prediction accuracy. Unfortunately, this limits the size of the training set drastically.

5 Baseline Analysis

To determine a simple baseline to beat with a probabilistic model, I used non-Bayesian Logistic Regression, and a standard 2 layer feed-forward neural network. Given that only 6% of the data contained fraudulent claims, a method other than overall prediction accuracy was used to determine each model's performance. With the context of detecting fraud, the goal is to have a high true-positive prediction rate, and low false-negative prediction rate (when normalized appropriately, these rates are compliments). Specifically, we want to identify as many fraudulent claims as possible. This sort of analysis is not intended to be the sole determinant for fraud detection, it is simply meant to assist in fraud detection. For example, claims that are identified as fraudulent will require additional research. For those claims that are falsely identified, it will simply be a dead-end. In the case where claims are falsely unidentified as fraudulent, then there are simply no next steps. Which is exactly why the aim is to maximize the true positive rate, which will then allow true fraud to be investigated thoroughly.

The final train set size is 923, with 35% fraudulent claims and 65% non-fraudulent claims.

5.1 Logistic Regression

The logistic regression used was run for 1000 max iterations, with default settings, and an additional bias/intercept variable. The overall test accuracy is 78.53% and confusion matrix results are below:

	Predicted Negative	Predicted Positive
Actual Negative	0.7621	0.1965
Actual Positive	0.0182	0.0232

Table 1: Confusion Matrix: Log Reg

5.2 Neural Network

The network architecture used was a two-layer feed-forward neural network. There are 50 hidden nodes per layer, and hidden ReLu activations. The final activation is the sigmoid function. The network was trained for 250 epochs, with 13 batches per epoch. The overall test accuracy is 74.87% and confusion matrix results are below:

	Predicted Negative	Predicted Positive
Actual Negative	0.7522	0.2065
Actual Positive	0.0190	0.0224

Table 2: Confusion Matrix: NN

6 Final Analysis

The final Bayesian model used is a Variational Bayesian Neural Network. There are two hidden layers, with 20 nodes each. The hidden activations are ReLu, and final activation is sigmoid. A factorized Gaussian posterior is used for the network weights, and a standard normal Gaussian prior is used for all weights. The network is trained over 200 epochs with learning rate 0.001. The overall test prediction accuracy is 95.22%, and the convergence results and confusion matrix are below:

	Predicted Negative	Predicted Positive
Actual Negative	0.9519	0.0067
Actual Positive	0.0411	0.0003

Table 3: Confusion Matrix: BNN

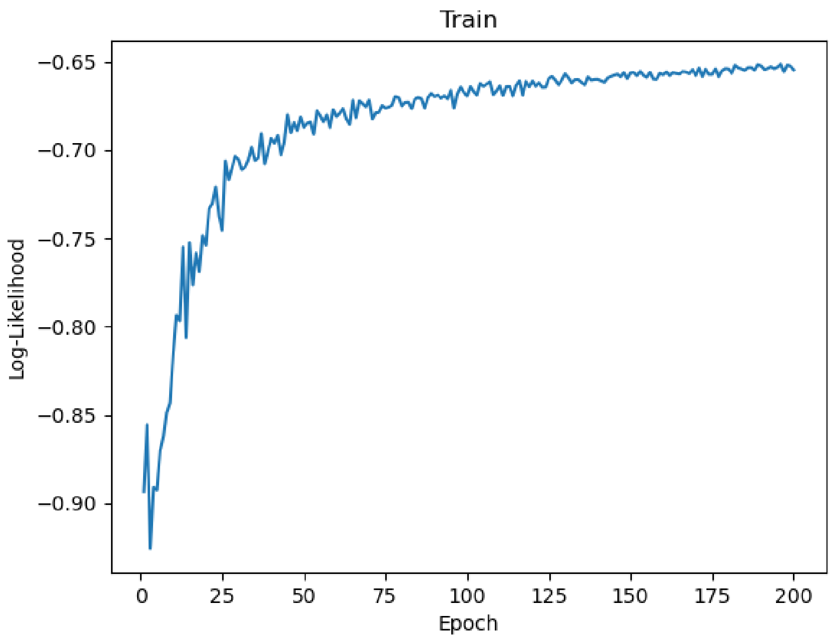

Figure 1: BNN convergence

6.1 Comparison of Results

From the baseline results, we see that both standard machine learning models accurately detected at least 2% of the total 6% of fraudulent claims. Additionally, the logistic regression had overall accuracy of 78% and the neural network had overall accuracy of 75%. Comparing to the final Bayesian neural network, both had a much higher true-positive prediction rate. The BNN had overall accuracy of 95%, which is much higher than the baseline models. In conclusion, the Bayesian model clearly performed worse when looking at true-positive rate, but better for overall prediction accuracy. Given that the overall goal was to maximize true-positive accuracy, the baseline models fit the data much better than the Bayesian model.

7 Next Steps

Although the Bayesian model did not perform as well as I would have hoped, there are various ways to improve its performance. If given more time to conduct analysis using Bayesian models, there are a few aspects of the modelling process I would prioritize:

1. Attempt various ratios to generate the train-test splits, assessing the impact via true-positive prediction rates.

2. Explore more methods to handle the train-test split to ensure the train set size is large enough, while also accounting for the low rate of fraudulent claims in the data, which is the goal of prediction in this analysis.

3. Use different BNN architectures to determine the best non-linear Bayesian model.

4. Incorporate various new methods for incorporating prior knowledge and uncertainty into the Bayesian model.