Updates

December 19, 2023

1 Overview

As of our last meeting, I had finished implementing the method described in Chen et al 2023. The next steps were to train simpler machine learning models to perform the prediction task. This is because our optimization model uses the prediction model's output to design the contracts. The next step will be to use our method, along with these predictions to design contracts and compare those contracts to the contracts designed by Chen's method. Additionally, we had hypothesized that the improvements seen by Chen et al 2023 had mostly been from having a more complex prediction model. I adapted the Chen's neural network to predict loss, and I compared it's performance to simpler algorithms. By this, I mean that I kept the exact same architecture, and just trained it to predict loss instead of training it to decide the payout. I found that simpler algorithms generally predicted losses better and were more cost efficient. In the rest of the document, I describe the performance metrics I developed for evaluating these prediction algorithms, the algorithms I used, as well as the results.

2 Performance Metrics

The end goal for our prediction models is to use them in an index insurance program. The guiding principle in developing these metrics was to capture what would be most relevant and interpretable to farmers and policy makers. From the farmer side, given the worry about basis risk, I think the most relevant aspect is the loss coverage. More specifically, what percentage of their losses can the farmer expect the model to correctly predict. From the policymaker's side, I think the relevant worry is cost.

Evaluation Metrics

In the binary classification setting, recall is the share of all positive samples that you classified as positive, and precision is the share of all of the samples classified as positive that were actually positive. Let TP be true positives, FP be false positives, TN be true negatives, and FN be false negatives. Precision and recall are then:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$

This metrics are useful when the dataset we are working with is imbalanced because they help to rule out naive models that could achieve high accuracy in these settings. For example, suppose we are predicting a disaster event that happens once every 5 years. Then a naive model that simply always says there is no disaster will have an accuracy of 80%. The performance metrics I developed are inspired by these two metrics.

Loss Recall This is the average ratio of payouts to losses. This is analogous to the recall described above. We can think of this as the share of all losses that the model catches or predicts. Let n_ℓ be the total number of samples with positive loss, y be the true loss, $\hat{y}$ be the predicted loss. Here, we are implicitly assuming a strike value of 0 for simplicity. The Loss Recall is then defined as:

$$\text{Loss Recall} = \frac{1}{n_\ell} \sum_{i:y>0} \frac{\min\{y, \hat{y}\}}{y}$$

We include the min in the numerator to prevent the model from inflating this score by giving large payouts. For example, suppose that the strike value is 0 and that there are two farmers. Let $y_i, \hat{y_i}$ be the true and predicted loss for farmer i. Suppose $y_1 = y_2 = 1$, and that $\hat{y_1} = 2, \hat{y_2} = 0$. The total loss recall would be $\frac{1}{2}(1/1+0) = 0.5$. However, if we didn't include the min the total loss recall would be $\frac{1}{2}(2/1+0) = 1$. I think this is a good measure of basis risk from the point of view of the farmer. Basis risk is often measured as the probability that the payout is less than the loss suffered by the farmer, however I think the share of the loss that will be covered is a more intuitive and more relevant metric from the point of view of the farmer making a decision to purchase the insurance.

Payout Precision Intuitively, this captures what share of all of the payouts you gave were valid or should have been issued. So, for example, suppose the strike value is 0, $y = 0.5$, and $\hat{y} = 0.7$. In this case, you issued a payout of 0.7, but only 0.5 of that payout was valid. Or in other words, only 0.5 of the 0.7 *should have* been paid out. Let $n_{s'}$ be the total number of samples where the predicted loss is greater than 0: $\hat{y} > 0$. The Payout Precision is then

$$\text{Payout Precision} = \frac{1}{n_{s'}} \sum_{i:\hat{y}>0} \frac{\min\{y, \hat{y}\}}{\hat{y}}$$

I included this performance metric because it might be of interest, but I'm not sure what benefit it would provide over just including the average cost.

Expected Cost This is the expected payout across all samples:

$$\text{Expected Cost} = \frac{1}{n} \sum \max\{\hat{y}, 0\}$$

Cost per Coverage This can be the ratio of expected cost vs total coverage. It can tell us about the cost efficiency of a particular model.

$$\text{Cost per Coverage} = \frac{\text{Expected Cost}}{\text{Total Loss Recall}}$$

3 Methods

In this section, I describe the methods I used for building the prediction model. There are, generally speaking, two steps in training a prediction model. The first step is feature extraction. That is, how do we convert the raw weather data into features to be used by a machine learning algorithm. For example, we could create a feature that includes the number of consecutive months with below average rainfall, or the average temperature in each season. This is an optional step, as the algorithm can simply take the raw weather data as input. However, feature extraction is generally an important step in prediction. The second step is to use these features to train a machine learning model. I describe the approaches for both in the following.

3.1 Feature Extraction

The raw data used by Chen et al 2023 consisted of 6 weather sources observed once per month. The six weather sources were precipitation, min/max temperature, min/max pressure deficit, and dew point. The features used by Chen to predict are: $(ppt_1, ppt_2, ..., ppt_{12}, min\_temp_1, ..., min\_temp_{12}, ...)$ where ppt_i is the precipitation in month i. In total, they have 72 features (6 variables times 12 months). Chen didn't do any feature extraction, they simply fed these features to a neural network. I used two main algorithms for feature extraction: Catch22 and RoCKeT. I provide brief descriptions of both below.

3.1.1 Catch22

Canonical Time-Series Characteristics (Catch22) is a set of 22 features developed by Lubba et al. (2019) that were shown to have good predictive performance on a variety of time series classification tasks. These features were selected from a potential set of over 7500 features provided by the *hctsa* toolbox (Fulcher, Little, and Jones (2013)). These features cover many aspects of a time series, such as mean, spread, auotcorrelation, entropy and many other attributes. The features in Catch22 were selected based on their individual predictive performance on several tasks. To reduce redundancy, the features were then clustered based on their performance on different tasks. A representative feature was taken from each cluster until only 22 remained. In the paper, the authors show that training a traditional machine learning classifier on these features achieves competetive performance on a variety of time series classification datasets.

3.1.2 ROCKET: Random Convolutional Kernel Transform

The Random Convolutional Kernel Transform (RoCKeT) classifier developed by Dempster, Petitjean, and Webb (2020) uses random convolutions to create 10,000 features for each training sample. These convolutions vary in dilation. Dilation is used to "spread" a kernel over the input. For dilation, d, a given kernel is convolved with every d^{th} element of the input (Dempster, Petitjean, and Webb (2020)). Larger dilation values allow the features to be computed across a longer time span. The maximum and the proportion of positive values is computed for each kernel, and these end up being the features for each time series. A ridge regression classifier is then trained on top of these features. The RoCKeT classifier was the best performing classifier amongst those evaluated in Ruiz et al. (2021), and was also the fastest to train.

3.2 Regression Algorithms

Random Forest Regression This is an ensemble method that uses an ensemble of decision trees to make predictions. This method makes predictions by aggregating the predictions of many trees, additionally the different trees are created to be independent. This makes the method more robust and reduces variance.

Ridge Regression This is a modification of linear regression where a penalty term is added to the weight vector to prevent overfitting. The objective is: $\min_w ||y - Xw||_2^2 + C||w||_2^2$.

Support Vector Regression The objective of this algorithm is to find a function that will fit every data point within an accuracy of ϵ. In the classification setting, it is also known as the maximum margin classifier, since it tries to find the hyperplane that yields the largest separation between positive and negative samples.

Gradient Boosting Machines This is another ensembles of decision tress, similar to Random Forests. However, in this algorithm, the trees are trained iteratively and each successive tree is places higher weight on the data points misclassified by previous trees. In other words, each successive tree is focused on learning the points that the previous trees failed to learn.

4 Experiment

Recall that our approach involves first training a prediction model, and then using an optimization model to design the insurance contracts based on the prediction model's predictions on historical data. Chen et al's method trains the neural network directly to decide payouts. They claim that this is better than the current method of predicting loss and then using a linear payoff contract. In their paper, they compare their method to a linear and polynomial contracts using 1, 5, and all 72 of the variables they use. They don't specify how they create these linear and polynomial contracts. It is unclear if they simply fit a polynomial to predict loss, or if they somehow included their utility-based objective when fitting the polynomials.

Either way, I suspect that their NN approach does better because it is using a more complex model to predict loss. They don't predict loss explicitly, but we can think of it as an implicit step done by the neural network when deciding the payout. I suspected that their method would only be marginally better than using the method of training a prediction model first and next designing contracts, given that the prediction model used was of a similar complexity to the neural network they use. Intuitively, I think that if a traditional ML model is as good as a neural network at predicting loss, then using the predict-then-design framework should yield somewhat similar results to the Chen method. If the traditional ML model can outperform a neural network, then the predict-then-design framework could outperform the Chen method. Furthermore, since traditional ML methods generally have fewer parameters than deep neural networks, they are better suited for settings with less data.

Training the machine learning models is an intermediate step in our evaluation, but an informative one. We need a good prediction model to use our optimization method to design the contracts, and finding the best prediction model was the goal of this exercise. However, it also allowed us to compare the predictive performance of traditional ML models to deep neural networks in this setting. For the exercise below, I used the neural network described by Chen et al, except that instead of training it to decide the payouts, I trained it to predict loss. Other than that, the two

Transform	Algorithm	Loss Recall	Payout Precision	Average Cost	Cost Per Coverage	Time	MSE
catch22	SVR	0.95	0.96	837.35	881.05	0.07	13249
catch22	Chen NN	0.94	0.96	832.61	881.09	0.35	13138
rocket	SVR	0.95	0.97	839.38	885.46	3.78	13906
rocket	Gradient Boosting	0.96	0.97	846.72	886.09	33.08	10460
rocket	Random Forest	0.96	0.97	849.26	887.71	2806	11655
catch22	Gradient Boosting	0.96	0.96	855.57	891.37	0.12	11226
chen	SVR	0.96	0.97	855.09	892.96	0.12	10552
catch22	Random Forest	0.96	0.95	859.71	894.02	0.39	10903
rocket	Ridge	0.91	0.97	814.73	894.05	0.06	19131
chen	Random Forest	0.96	0.97	857.63	897.10	0.71	8095
catch22	Ridge	0.95	0.95	860.00	900.48	0.00	14451
rocket	Chen NN	0.96	0.95	867.53	902.09	3.13	11397
chen	Gradient Boosting	0.96	0.96	865.13	902.44	0.18	8866
chen	**Chen NN**	0.88	0.96	805.90	914.18	0.30	28683
chen	Ridge	0.96	0.94	901.27	934.92	0.00	14054

models are identical. I found that traditional ML algorithms, when coupled with the time-series feature extraction algorithms, outperform Chen's neural network when predicting loss. I think this is not super surprising, since for machine learning tasks with relatively simple data (e.g. tabular data, see here for brief overview) , the feature engineering tends to matter a lot more than the specific algorithm used. I think deep neural networks outperform other methods mainly when they are dealing with highly complex tasks that involve hundreds of thousands of training samples (e.g. natural language processing, computer vision, and reinforcement learning).

I also included a more typical measure of the model's predictive performance (MSE) for completeness. However, even using that as a performance metric, the Chen NN is not the best performer. I think the results highlight that MSE is not necessarily a good performance metric in this case, since models can have the same loss recall and payout precision, while having vastly different MSEs. The top performer in terms of cost per coverage is support vector regression combined with Catch22 for feature extraction. The best performer in terms of MSE is the Random Forest combined with the raw features used by Chen. When it comes to cost efficiency, the best performance is from methods using one of the time-series feature extraction algorithms.