

Health Analytics Leveraging Snap ML

Divya Prakash Kulkarni
Sr. Data Scientist
ISDL, IBM
Bengaluru, India
divya.kulkarni1@ibm.com

Chandra Shekhar Reddy Potula
Lead AI on zSystems Solution Team
ISDL, IBM
Bangalore , India
chandra.shekhar@in.ibm.com

Abstract— This paper presents Snap machine learning based health Insurance claims prediction system. The objective is to utilize both Snap ML classifier and regressor for the creation of predictive models in the realm of Health Insurance. Snap ML offers very powerful, multi - threaded CPU solvers, as well as efficient GPU solvers.

Keywords— AI on Mainframes, Snap ML, IBM Z, Health Analytics

I. INTRODUCTION

In medical insurance organizations, the medical claims amount that is expected as the expense in a year, plays an important factor in deciding the overall financial growth of the company. It enables insurers and policymakers to navigate the complex landscape of healthcare financing and make informed decisions that benefit both the insurance industry and the consumers it serves.

In the health insurance industry, Artificial Intelligence (AI) has emerged as a transformative force, reshaping various aspects of healthcare operations and decision-making. The integration of AI brings about a paradigm shift in how insurers evaluate risks, optimize processes, enhance customer experience, and ultimately improve healthcare outcomes.

Snap ML [2] is a library that provides high speed training and inference of popular machine learning models. It is developed & maintained by IBM Research and is fully compatible with the scikit-learn Python API. It supports accelerated scoring of scikit-learn, XGBoost and LightGBM trained models when exported or converted to: PMML, JSON, ONNX. SnapML is supported on the platforms (1) Ubuntu: Intel™ (x86_64), IBM Power™ (ppc64le) and IBM Z™ (s390x) architectures. (2) RHEL/CentOS: Intel™ (x86_64), IBM Power™ (ppc64le) and IBM Z™ (s390x) architectures. (3) IBM Z™ (s390x). (4) MacOS: Intel™ (x86_64) architecture. (5) Windows: Intel™ (amd64) architecture. This helps to train SnapML model on above platforms where data might be existing and later deploy trained models on IBM Z [3]. IBM Z servers are a family of high-performance mainframe computers designed and manufactured by IBM. IBM Z servers are known for their reliability, security, and scalability, making them a popular choice for large enterprises and organizations with demanding computing needs.

Here we use Snap ML based method to solve the health insurance claims prediction task. The objective of this paper is to discuss Snap ML based model that provide yearly medical claim expense of an insurance company, gain valuable insights into claim patterns, and optimize their operations to provide efficient and cost-effective healthcare coverage.

The rest of the paper is divided as sections. Section II discusses the problem statement that we are dealing with in this paper, section III talks about the approach taken, dataset description is provided in section IV along with details of various features, section V depicts exploratory data analysis and visualization. In section VI we have discussed the methodology used. Section VII represents the results and detailed discussion. Finally, in the last section VIII we end the paper with conclusion.

II. PROBLEM STATEMENT

According to McKinsey & Company, Global Insurance Report 2022, twelve insurers that write homeowners coverage in Louisiana were declared insolvent between July 2021 and February 2023. Insurers have paid out more than 23 billion dollars in insured losses from over 800,000 claims filed from the two years 2021 & 2022[1]. In medical insurance organizations, the medical claims amount that is expected as the expense in a year, plays an important factor in deciding the overall financial growth of the company. This amount needs to be included in the yearly financial budgets. Inappropriate estimating generally has negative effects on the overall performance of the business. These claim amounts are usually high in millions of dollars every year. An increase in medical claims will directly increase the total expenditure of the company thus affecting the profit margin.

III. APPROACH

Traditional claims prediction methods either rely on past claim patterns or use probability distribution functions to match an insurance company's claim. Prediction assumes that small claims happen more often, while large claims occur less frequently. To enhance accuracy, large and small claims are analyzed separately using different distribution functions to better match their frequency. Traditional methods may not accurately reflect changing trends, emerging risks and non-linear patterns within the data leading to less precise predictions. Traditional approach is also time-consuming, as it may involve manual data preparation which can delay decision-making processes.

As an alternative, scikit-learn machine learning-based claims forecasting offers better accuracy, adaptability, and efficiency compared to traditional approaches. It leverages advanced analytical techniques enabling insurers to make more informed decisions.

Here, we propose using a Snap ML-based solution, which is more advanced than scikit-learn, to predict medical claim costs for an insurance company. This system will determine if an individual policyholder is likely to make a claim or not.

Snap ML provides the following benefits over scikit-learn models. (1) Speed and Efficiency: Snap ML is designed for high performance, particularly on large datasets. It leverages parallel processing and hardware acceleration, such as GPUs, to speed up training and inference times. This can significantly reduce the time required to develop and deploy machine learning models. (2) Reduced Infrastructure Costs: By optimizing model training and inference processes, Snap ML can lead to reduced infrastructure costs. Faster processing times and efficient resource utilization can result in lower cloud computing expenses or reduced hardware requirements. (3) Scalability: Snap ML is built to handle large-scale machine learning tasks. It can efficiently process and analyze massive datasets, making it suitable for big data applications and scenarios where scalability is crucial. (4) Real-time Inference: Snap ML is designed for real-time and low-latency inference, making it suitable for applications that require quick decision-making, such as fraud detection or recommendation systems. (5) Competitive advantage: Leveraging Snap ML's performance and scalability can provide organizations with a competitive advantage by enabling them to develop and deploy machine learning models faster and more efficiently.

The proposed approach offers the following advantages to the insurance company: (1) Helps insurance company to prepare for real-time liquidity. (2) Identifying policyholders who are likely to submit claims, allowing insurers to enroll them in health management programs, thereby enhancing customer satisfaction (3) Helps improve insurance companies bottom line and overall profit (4) Insurance company can use this solution as an underwriting support to determine the claim risk.

IV. DATASET PREPARATION

The health insurance claims prediction dataset is sourced from the Kaggle website, and it is subsequently augmented by incorporating the claim amount data (https://www.kaggle.com/datasets/grvmishra7/health-insurance-prediction?select=train_Df64bvy.csv). There are 50883 observations in the dataset. The two variables Response and claim_amount in the dataset are the target variables which are to be predicted. Dataset description is provided in Table I. There are 7 numeric, 9 categorical and 1 boolean variable types.

TABLE I. DATASET DESCRIPTION

S.No	Name of the features	Datatype	Possible Values
1	ID	Real number	Policy holder unique id
2	City_Code	Categorical	City pin codes
3	Region_Code	Real number	Region codes
4	Accommodation_Type	Categorical	Owned, Rented
5	Reco_Insurance_Type	Categorical	Individual, Joint
6	Upper_Age	Real number	18-75
7	Lower_Age	Real number	16-75
8	Is_Spouse	Categorical	Yes, No
9	Health Indicator	Categorical	X1-X9 (worst health to best health)
10	Holding_Policy_Duration	Categorical	1-14, 14+

S.No	Name of the features	Datatype	Possible Values
11	Holding_Policy_Type	Categorical	1-4
12	Reco_Policy_Cat	Real number	1-22
13	Reco_Policy_Premium	Real number	2280-43350.4
14	Response	Boolean	0, 1
15	Claim amount	Real number	134520-2212358

V. EXPLORATORY DATA ANALYSIS AND VISUALIZATION

Exploratory data analysis operations are performed on the dataset using Python as the programming language, pandas and numpy libraries. Data Visualization is performed using matplotlib and sea born libraries.

Data is read and loaded into dataframe using read_csv library from pandas. There are no duplicate observations in the dataset. The variable Health Indicator has 23% missing values. The two variables Health Indicator and Holding_Policy_type are highly correlated with Response variable. The three variables Holding_Policy_type, Reco_Insurance_Type and Health Indicator has medium correlation with claim_amount. The correlation between variables is identified using heatmap as shown in Fig. 1.

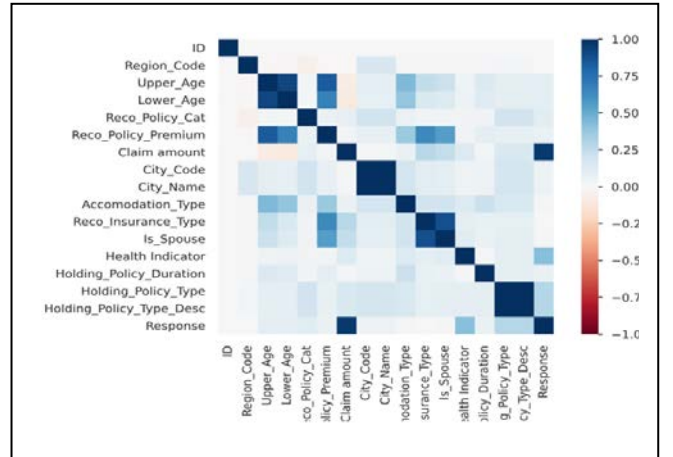


Fig. 1: Heatmap of Claims dataset

The dataset is divided into a training and testing split with a ratio of 70% for training and 30% for testing. Data pre-processing operations such as missing value imputation, OneHotEncoding, Label Encoding, and Normalization is performed inside a pipeline as shown in Fig. 2.

VI. METHODOLOGY USED

We have utilized two Snap ML models. The initial Snap ML model serves as a classifier-based model, tasked with predicting the Response variable, while the second Snap ML model takes on a regression-based role, focused on predicting the claim_amount. Snap ML can be easily installed through pip and is compatible with Python versions 3.7, 3.8, and 3.9.

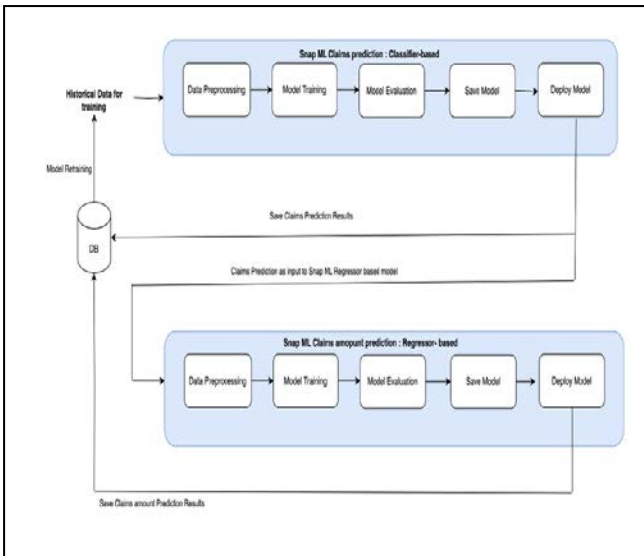


Fig. 2: Architecture diagram

As shown in Fig. 2, historical data undergoes through a series of pre-processing steps to prepare it for analysis. These steps involve removing duplicates, outliers, unrelated and highly correlated fields, and encoding categorical data. Additionally, the data is normalized. Once the data preparation is complete, we utilize it to train a Snap ML classifier-based model. Data pre-processing and model building are carried out within a pipeline, as illustrated in Fig. 5. The model's performance is assessed using metrics like accuracy, AUC, recall, precision, and F1 score, and we fine-tune its hyperparameters for optimal results. Afterward, the model is saved and can be deployed on any inference platform. we have deployed on Z servers. Additionally, model is retrained for any incorrect prediction.

If the model predicts a '1' in the Response variable, it indicates the policyholder is likely to make a claim, '0' indicates less likelihood. Only where '1' is predicted, the data goes to the second Snap ML regression-based model, along with historical data, to predict the claim amount.

The Snap ML regressor-based model also undergoes data preparation, model construction, evaluation, hyperparameter tuning, deployment and model retraining (for any incorrect predictions)

By combining Snap ML classifier and regressor methods this way, we can estimate both the number of claims and the total claim amount for the specified group of policyholders.

Here we show step by step process along with supporting screenshots.

- (1) Loading the libraries and modules: All the required libraries and modules are loaded.

```
# Import required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.metrics import confusion_matrix, roc_auc_score, precision_score, recall_score, f1_score, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, roc_auc_score
from sklearn.metrics import classification_report
```

Fig. 3: Load libraries

- (2) Loading data: Data downloaded from Kaggle is read as a csv file into a dataframe using pandas method read_csv()

```
#Read the dataset
df = pd.read_csv('/home/divya/hicp/data/HICP_TrainV2.csv')
```

Fig. 4 : Load the csv file into data frame.

- (3) Pipeline creation: Pipeline is created with steps to perform data preprocessing and model building.

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=40)

pipeline = PMMLPipeline([
    ('mapper', DataFrameMapper([
        (('City_Code',[LabelEncoder])),
        (('Region_Code',[SimpleImputer])),
        (('Accommodation_Type',[OneHotEncoder(drop='first')])),
        (('Reco_Insurance_Type',[OneHotEncoder(drop='first')])),
        (('Upper_Age',[SimpleImputer])),
        (('Lower_Age',[SimpleImputer])),
        (('Is_Spouse',[OneHotEncoder(drop='first')])),
        (('Health_Indicator',[SimpleImputer(missing_values=np.nan, strategy='most_frequent')]),OneHotEncoder(drop='first')),
        (('Holding_Policy_Duration',[SimpleImputer(missing_values=np.nan, strategy='most_frequent')])),
        (('Holding_Policy_Type',[SimpleImputer(missing_values=np.nan, strategy='most_frequent')])),
        (('Reco_Policy_Cut',[LabelEncoder])),
        (('Reco_Policy_Premium',[SimpleImputer])),
        (('Reco_Policy_Premium',[SimpleImputer])),
    ]), input_df=True, df_out=True),
    ('Classifier', SnapDecisionTreeClassifier(max_depth=16, n_jobs=4, random_state=42))
])

pipeline.fit(X_train,y_train)
```

Fig. 5: Pipeline with data pre-processing and Snap ML classifier-based model

- (4) Model evaluation: Evaluate the model predictions for their correctness using various performance metrics.

```
#Train the snap ML model on the transformed train data
#Snap ML - RANDOM FOREST CLASSIFIER
import time
from snapml import RandomForestClassifier as SnapRandomForestClassifier

snapmlRFC = SnapRandomForestClassifier(max_depth=16, n_estimators=50, n_jobs=4, random_state=42, use_histogram=True)
snapmlRFC.fit(train_data_X, y_train)
t_fit_sklearn = time.time()-t0
score_snapml = score(y_test, snapmlRFC.predict_proba(test_data_X)[1,:])

print("*****Snap ML Random Forest Classifier*****")
print("Training time (SnapML RFC): %f seconds" % (t_fit_sklearn))
print("ROC AUC score (snapml RFC): %f" % (score_snapml))

snapml_model_predict=snapmlRFC.predict(test_data_X)
confusion_matrix=snapmlRFC.confusion_matrix(y_test,snapml_model_predict)
print(classification_report(y_test, snapml_model_predict))

*****
Snap ML Random Forest Classifier
*****
Training time (SnapML RFC): 0.31 seconds
ROC AUC score (snapml RFC): 0.8449
precision recall f1-score support
0 0.76 0.93 0.84 5264
1 0.90 0.68 0.78 4913

accuracy
macro avg 0.83 0.81 0.81 10177
weighted avg 0.83 0.81 0.81 10177
```

Fig. 6: Model Evaluation

(5) Hyperparameter tuning: Fine tune hyperparameters using cross validation.

```
# Hyper parameter tuning of SnapML Random Forest Classifier

from sklearn.model_selection import RandomizedSearchCV
random_grid = {
    'max_depth': [10, 10, 10, 40, 50, 60],
    'n_estimators': [100, 200, 300, 400, 500]}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = SnapRandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42)
# Fit the random search model
rf_random.fit(train_data_X, y_train)
score_snapml = score(y_test, rf_random.predict_proba(test_data_X)[0,:])

Fitting 3 folds for each of 30 candidates, totalling 90 fits

#Print the best param and the optimized model score
print(rf_random.best_params_)
score_snapml = score(y_test, rf_random.predict_proba(test_data_X)[0,:])
print('ROC AUC score (snapml_RFC): %.4f' % (score_snapml))

{'n_estimators': 500, 'max_depth': 20}
ROC AUC score (snapml_RFC): 0.8512
```

Fig. 7: Hyperparameter tuning.

(6) Saving the model: using Snap ML export function.

```
# save the RFC model into a file
snapML_RFC.export_model('/home/divya/ncip/models/HealthInsClaimsSnapML_RFC.pkl', output_type='pml')
print('SnapML RFC model saved successfully.')
```

Fig. 8: Save the model.

VII. RESULTS

Snap ML models, along with several scikit-learn models, is constructed using the claims dataset. It is observed that Snap ML outperforms other scikit-learn models in terms of both performance metrics and execution time. The comparison between Snap ML and other scikit-learn models for claims prediction and claims amount prediction is illustrated in Figure 9 and Figure 10, respectively.

Model	Accuracy	AUC	Recall	Prec.	F1	TT(Sec)
Snap Random Forest Classifier	0.81	0.8440	0.81	0.83	0.81	0.33
Snap Logistic Regression Classifier	0.80	0.8288	0.80	0.81	0.80	1.37
Snap Decision Tree Classifier	0.77	0.7928	0.77	0.78	0.77	0.08
Extreme Gradient Boosting	0.8084	0.8685	0.6888	0.8862	0.7750	7.4420
Random Forest Classifier	0.8073	0.8531	0.6709	0.9019	0.7694	1.2820
Ridge Classifier	0.7907	0.8000	0.7025	0.8347	0.7628	0.3760
Linear Discriminant Analysis	0.7906	0.8264	0.7025	0.8347	0.7628	0.4720
Ada Boost Classifier	0.7769	0.8344	0.7277	0.7905	0.7577	0.7210
Logistic Regression	0.7428	0.7884	0.6975	0.7496	0.7218	1.3960
Decision Tree Classifier	0.7387	0.7384	0.7329	0.7249	0.7288	0.4030
Naive Bayes	0.7030	0.8251	0.7926	0.6579	0.7189	0.6070
Quadratic Discriminant Analysis	0.6886	0.8101	0.7826	0.6443	0.7066	0.4530
K Neighbors Classifier	0.5010	0.4952	0.4607	0.4785	0.4694	1.9890
SVM - Linear Kernel	0.4989	0.0000	0.6333	0.4009	0.4331	0.3350

Fig. 9: Snap ML classifier vs scikit-learn classifier.

In our development environment, when training with the scikit-learn Random Forest Classifier, it takes approximately .2820 seconds, whereas, with Snap ML, the training time is notably reduced to just 0.33 seconds as shown in Fig. 9. This means that Snap ML is approximately 3.8 times faster than scikit-learn in this context. It's important to note that actual execution times may vary based on factors such as the underlying infrastructure, available RAM, system workload, etc. For our training and testing, we utilized the IBM Z servers.

Model	R2	TT (Sec)
Snap Boosting Machine Regressor	0.9576	0.3628
Snap Random Forest Regressor	0.9254	0.6549
Extra Trees Regressor	0.9080	1.2600
Random Forest Regressor	0.9037	0.9050
Gradient Boosting Regressor	0.8743	0.3860
Decision Tree Regressor	0.8262	0.5330
Linear Regression	0.7783	0.3950
Lasso Regression	0.7783	0.4540
Ridge Regression	0.7783	0.5650
AdaBoost Regressor	0.7317	0.3450
Elastic Net	0.4830	0.3300
Bayesian Ridge	0.0066	0.3360

Fig. 10: Snap ML regressor Vs scikit-learn regressor.

Likewise, as shown in Fig. 10 when using scikit-learn Gradient Boosting Regressor, the training time is recorded as 0.3860 seconds and accuracy achieved is 87.43%. However, when applying Snap Boosting Machine Regressor training time is reduced to 0.3628 seconds and performance significantly improves to 95.76%.

VIII. CONCLUSION

Traditional claims forecasting methods have limitations in terms of accuracy, adaptability, and their ability to capture complex patterns, which can impact the effectiveness of risk management and decision-making in insurance industries.

Snap ML is a framework for fast training of generalized machine learning models. It is developed with the vision to remove training time as a bottleneck for machine learning applications. Snap ML supports a large number of classical machine learning models and scales gracefully to datasets with billions of records. When compared to scikit-learn, Snap ML offers a robust set of features and benefits, making it a valuable tool for organizations looking to accelerate their machine learning projects, improve model accuracy, and efficiently process large datasets.

The results as illustrated in section VII, demonstrate that Snap ML not only provides superior predictions but also significantly reduces training duration. This paper elucidates that Snap ML leads to a substantial enhancement in training efficiency.

REFERENCES

- [1] McKinsey & Company ,Global Insurance Report 2022 :
<https://www.mckinsey.com/industries/financial-services/our-insights/creating-value-finding-focus-global-insurance-report-2022>
- [2] C. Dünner, T. Parnell, D. Sarigiannis, N. Iouannou, A. Anghel and H. Pozidis, “SnapML: A Hierarchical Framework For Machine Learning,”NeurIPS, 2018.
- [3] E. C. McCain, P. Bastien, B. F. Belmar, B. Bhattacharya, “IBM Z development transformation” published in IBM Journal of Research and Development (Volume: 64, Issue: 5/6, Sept.-Nov. 2020)
- [4] Snap ML documentation : <https://snapml.readthedocs.io/en/latest/#>
- [5] Example notebooks to demonstrate how to use Snap machine learning (Snap ML) library : <https://github.com/IBM/snapml-examples>
- [6] M. A. Morid, K. Kawamoto, T. Ault, J. Dorius, and S. Abdelrahman, “Supervised Learning Methods for Predicting Healthcare Costs: Systematic Literature Review and Empirical Evaluation,” AMIA Annual Symposium Proceedings, vol. 2017, p. 1312, 2017.
- [7] PolitiMC, Shacham E, Barker AR, George N,Mir N, Philpott S, et al. A Comparison Between Subjective and ObjectiveMethods of Predicting Health Care Expenses to Support Consumers’
- [8] Health Insurance Plan Choice. MDM Policy & Practice. 2018; 3(1):238146831878109. doi:10.1177/2381468318781093.
- [9] M. Kumar, R. Ghani, and Z. S. Mei, “Data mining to predict and prevent errors in health insurance claims processing,” in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 65–74, Washington, DC, USA, July, 2010