Supervised Learning Algorithms for Classification: A Comparative Study

Jen-Lung Hsu
Institute of Data Science, National
Cheng Kung University
Tainan, Taiwan
RE6121011@gs.ncku.edu.tw

***Abstract*—This research paper delves into the implementation and evaluation of various supervised learning algorithms for classification tasks. The study focuses on manual implementation of linear classifiers, k-nearest neighbors (K-NN) classifiers, and decision tree classifiers, including pruning. Additionally, it explores feature engineering techniques and assesses feature importance using SHAP values. The stability of classifiers is verified through k-fold cross-validation, and a novel algorithm for merging the predicted results from k classifiers is presented. The paper concludes by comparing the performance of models and justifying the best approach.**

Keywords—supervised learning, feature engineering, k-fold cross-validation

I. Introduction

In the era of machine learning and data-driven decision-making, supervised learning algorithms are pivotal for classification tasks. They enable the development of models that can effectively categorize data based on labeled examples, offering solutions for diverse applications such as image recognition, sentiment analysis, and fraud detection. This research embarks on a journey to explore and understand the nuances of these supervised learning algorithms. The focus is on manual implementation, deliberately avoiding established libraries like scikit-learn. The aim is to gain in-depth insights into how these algorithms work, their strengths, and their limitations in classification tasks.

To ensure a meaningful examination, a real-world dataset from the insurance domain is used. This data serves as a practical context to assess algorithm performance. The dataset is divided into training, validation, and test sets, ensuring that the algorithms are rigorously evaluated while avoiding overfitting. The study commences with the implementation of basic linear classifiers and gradually advances to more complex approaches, including k-nearest neighbors (K-NN) with various distance metrics and decision trees, both naïve and pruned versions. The comparison of these algorithms forms the core of our analysis. Feature engineering is explored to assess the impact of new features on classification accuracy. Cross-validation is employed to test classifier stability across different k values, illuminating their adaptability and optimal configurations.

In summary, this research aims to unveil the inner workings of supervised learning algorithms in classification tasks. It offers insights into their applications, strengths, and potential challenges. By evaluating and comparing these algorithms in various scenarios, this study provides a valuable resource for data professionals, enabling them to make informed choices when selecting classifiers for specific needs in the realm of supervised learning.

II. Problem Description

The dataset used for this study is derived from Kaggle's "Car Insurance Claim Prediction" dataset. It comprises comprehensive information about policyholders and the attributes associated with them, offering a rich source of data for exploring and solving real-world problems related to insurance claim prediction.

The dataset encompasses various features, including but not limited to:

- **Policy Tenure:** The duration for which a policyholder has been enrolled.
- **Car Age:** The age of the insured vehicle.
- **Car Owner's Age:** The age of the car owner.
- **Population Density:** The density of the population in the city where the policyholder resides.
- **Car Make and Model:** The specific make and model of the insured vehicle.
- **Power:** A measure of the vehicle's power.
- **Engine Type:** The type of engine in the insured car.
- And many more.

The dataset also includes a crucial target variable, which serves as the focal point of this study. This binary target variable indicates whether a policyholder is likely to file an insurance claim within the next 6 months or not. Predicting this claim occurrence accurately is of paramount importance to insurance companies, as it directly influences their risk assessment, pricing strategies, and operational decisions.

The problem at hand can be framed as a binary classification task, where the goal is to build machine learning models that can effectively predict whether a policyholder will file an insurance claim within the specified timeframe. This has practical implications for the insurance industry in terms of optimizing their claim management, fraud detection, and overall policyholder experience. To tackle this problem, we delve into the implementation and evaluation of supervised learning algorithms, with a specific focus on manual implementation, thereby gaining a deeper understanding of their functioning and performance. The choice of algorithms includes basic linear classifiers, k-nearest neighbors (K-NN) using various distance metrics, and decision trees, both with and without pruning. We also explore feature engineering techniques to enhance the predictive power of the models.

Furthermore, cross-validation is employed to assess the stability and generalization capability of these classifiers, addressing the need for robust and reliable insurance claim prediction models. By addressing these challenges in the

context of the "Car Insurance Claim Prediction" dataset, this research aims to provide valuable insights for the insurance industry, ultimately contributing to improved risk assessment and claim management strategies.

III. Data Preprocessing

In this section, we detail the preprocessing steps undertaken to prepare the dataset for our supervised learning classification tasks. The dataset[1] is retrieved from a real-world source, and we employ several operations to clean, enhance, and address class imbalance issues.

A. Handling Continuous Variables[2]

1) Removal of Uninformative ID Column: The first step in data preprocessing is to remove the **`policy_id`** column as it lacks any meaningful information for our prediction task.

*2) Processing **`max_torque`**:* The **`max_torque`** column contains information about torque values and the corresponding revolutions per minute (RPM). We extract the torque and RPM values and convert them to numeric data types. Subsequently, we calculate the 'torque to RPM ratio' for each entry to gain insights into the relationship between torque and RPM. The original **`max_torque`**, **`rpm`**, and **`torque`** columns are removed from the dataset.

*3) Processing **`max_power`**:* Similar to the **`max_torque`** column, the **`max_power`** column is processed to extract power and RPM values, which are then converted to numeric data types. We calculate the 'power to RPM ratio' for each entry. The **`power`**, **`rpm`**, and **`max_power`** columns are removed from the dataset.

B. Handling Categorical Variables[2]

1) Conversion of 'Yes' and 'No' to 1 and 0: In the dataset, categorical variables are represented as 'Yes' and 'No'. To make them suitable for machine learning algorithms, we transform these values into binary format: 'Yes' to 1 and 'No' to 0.

2) Dummy Encoding for Remaining Categorical Variables: To handle the remaining categorical variables, we employ dummy encoding. Notably, we use **`drop_first=True`** to avoid multicollinearity, ensuring one column is omitted to prevent redundancy.

C. Addressing Class Imbalance

The original dataset exhibits class imbalance with significantly more samples in the **`is_claim`** category labeled as '0' compared to '1'. To mitigate this issue, we employ Synthetic Minority Over-sampling Technique (SMOTE).

SMOTE is employed to oversample the minority class (**`is_claim`** = 1) by generating synthetic examples. The number of samples in the minority class is increased to match the number of samples in the majority class (**`is_claim`** = 0). The final preprocessed dataset contains a balanced distribution of classes, with an equal number of '0' and '1' labels for the **`is_claim`** target variable.

In summary, the preprocessing steps described in this section aim to enhance the dataset's suitability for training machine learning models, particularly in the context of classification tasks. The balanced dataset ensures that both classes are equally represented, allowing for more reliable model training and evaluation.

In the following sections, we will delve into the manual implementation of various classification algorithms, feature engineering, and model evaluation using cross-validation techniques.

IV. Classification Algorithms

A. Linear Classifier

The provided code includes a manual implementation of a basic linear classifier, specifically the Perceptron. It allows for the training of a binary classification model that learns to classify data into one of two classes. Key elements of this implementation include:

- **`Perceptron`** class: The class encapsulates the Perceptron model, offering methods for training and prediction.
- **`fit`**: This method is used to train the Perceptron model using the training data. It iteratively updates the model's weights to minimize classification errors.
- **`predict`**: After training, this method predicts the class label for input data based on the learned weights.
- **`predict_shap`**: An additional method is provided for use with SHAP (SHapley Additive exPlanations), a tool for interpreting machine learning models.

B. K-NN Classifier

The code includes two K-Nearest Neighbors (K-NN) classifiers that are manually implemented with different distance metrics:

- **`KNN`** class: This implementation uses the Euclidean distance metric.
- **`KNN_manhattan`** class: This implementation uses the Manhattan (L1) distance metric.
- **`KNN_chebyshev`** class: This implementation uses the Chebyshev ($L\infty$) distance metric.

Key functions in the code:

- **`fit`**: These classes include a **`fit`** method for training the K-NN model with training data.
- **`predict`**: The **`predict`** method predicts the class label for input data by finding the k-nearest neighbors and selecting the majority class.
- Each K-NN implementation supports the use of different distance metrics, which may be chosen based on the problem's characteristics.

C. Naïve Decision Tree Classifier

In this section, a basic decision tree classifier, referred to as the **`NaiveDecisionTree`**, is manually implemented. This classifier creates a binary decision tree without any pruning. Key components include:

- **` NaiveDecisionTree`** class: This class provides methods for fitting the decision tree to the data and making predictions.
- **`fit`**: This method grows a decision tree by recursively splitting nodes based on the Gini impurity criterion, without any pruning.

- `**predict**`: After training, the predict method classifies new data samples by traversing the tree.
- `**compute_feature_importance**`: The model also computes feature importance scores during training.

D. Decision Tree with Pruning

In this extension of the decision tree classifier, referred to as the `**PrunedDecisionTree**`, a pruning algorithm is incorporated to prevent overfitting. Key features of this implementation include:

- `**PrunedDecisionTree**` class: An extension of the `**NaiveDecisionTree**`, but with added pruning capability.
- `**__init__**`: The constructor allows you to specify parameters like `**max_depth**` and `**min_samples_split**` for controlling the depth of the tree and minimum samples required to split a node.
- `**fit**`: The `**fit**` method grows a decision tree by recursively splitting nodes based on the Gini impurity criterion. Pruning is applied based on the specified criteria, such as a minimum number of samples required to split a node.
- `**predict**`: After training, the `**predict**` method classifies new data samples by traversing the tree.
- The `**PrunedDecisionTree**` enhances the basic decision tree's generalization capabilities by preventing overfitting through pruning.

V. Feature Engineering

In this section, we delve into feature engineering, a crucial aspect of the machine learning pipeline, which encompasses techniques for enhancing the dataset's informativeness and the model's performance. Our focus lies on determining feature importance, leveraging SHAP (SHapley Additive exPlanations) values for assessment, and proposing novel feature engineering strategies based on observed patterns.

A. Determining Feature Importance

To discern the importance of features, we employ two different approaches: one for linear classifiers and one for decision trees.

1) Linear Classifiers: Perceptron: We implement a Perceptron model with specified parameters. After fitting the model, we extract the model's weights for each feature. Feature importance is determined by the magnitude of the weights. We present the top 5 important features in Table I, and proceed with training the Perceptron on these features to evaluate its classification performance.

TABLE I. TOP 5 IMPORTANT FEATURES OF PERCEPTRON

Perceptron	
Top 5 important features	Feature importance
`age_of_car`	14762.5005
`height`	7488.2541
`make`	-6255.6044
`width`	-4627.5573
`area_cluster_C20`	2790.0995

2) Decision Trees: Naïve Decision Tree (NDT): For decision trees, we introduce a Naïve Decision Tree (NDT) model with a limited depth. The model is used to identify the top 5 important features based on the structure of the tree and presented in Table II. These important features are then selected for subsequent model training, leading to a more concise representation of the dataset.

TABLE II. TOP 5 IMPORTANT FEATURES OF NAÏVE DECISION TREE

Naïve Decision Tree	
Top 5 important features	Feature importance
`policy_tenure`	120.0
`age_of_car`	15.0
`segment_B2`	5.0
`make`	4.0
`area_cluster_C8`	4.0

B. SHAP Values for Feature Importance

SHAP values provide an alternative perspective on feature importance. We apply SHAP to both the Perceptron and NDT models. For the Perceptron, we create an SHAP explainer and calculate SHAP values to visualize the impact of each feature on predictions in Fig. 1. For the NDT, we encapsulate the model in an SHAP-compatible wrapper to perform a similar analysis and present in Fig. 2.

C. Proposing New Features

Based on our observations and domain knowledge, we introduce new features to potentially enhance model accuracy. Specifically, we calculate a `**volume**` feature as the product of `**height**`, `**width**`, and `**length**`—a meaningful indicator of a vehicle's dimensions. We include this new feature and all the important features in our dataset and retrain both the Perceptron and NDT models with a selection of features, observing any improvements in classification performance.

D. Results and Evaluation

In all cases, after retraining the models with the selected features, we evaluate their performance using accuracy. The results of these experiments are presented in Table III and IV.

TABLE III. ACCURACY OF PERCEPTRON & NAÏVE DECISION TREE WITH ONLY 5 IMPPORTANT FEATURES

Only 5 important features	
	Accuracy
Perceptron	0.50
Naïve Decision Tree	0.82

TABLE IV. ACCURACY OF PERCEPTRON & NAÏVE DECISION TREE WITH ALL THE IMPPORTANT FEATURES

All the important features	
	Accuracy
Perceptron	0.50
Naïve Decision Tree	0.84

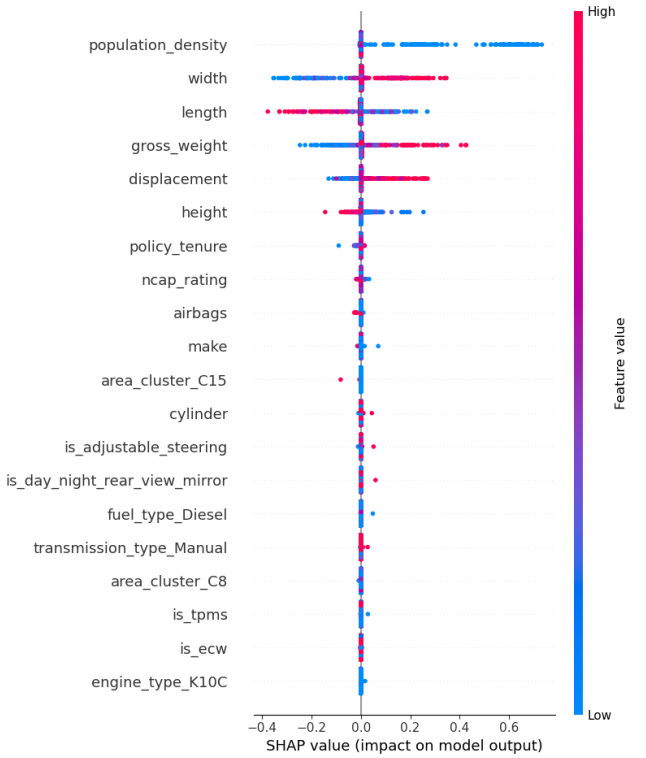

Fig. 1. SHAP value of top 20 important features of Perceptron.

Fig. 2. SHAP value of top 20 important features of Perceptron.

From the table above, it is evident that the addition of several key variables did not lead to an improvement in the performance of the perceptron model. However, there was a slight increase in accuracy observed in the Naïve Decision Tree model. Therefore, in the subsequent models, we included all the important features.

VI. Cross-Validation

A. *K-Fold Cross-Validation*

For k-fold cross-validation, we experiment with different values of k (k = 3, 5, 10). We partition the dataset into k subsets of roughly equal size, train the classifiers on k - 1 subsets, and evaluate their performance on the remaining subset. This process is repeated k times, rotating through the subsets.

TABLE V. ACCURACY OF MODELS FOR 3-FOLD CROSS-VALIDATION

K = 3				
	Perceptron	KNN	Naïve Decision Tree	Decision Tree with Pruning
Fold 1	0.4986	0.9433	0.8882	0.8882
Fold 2	0.5011	0.9426	0.8952	0.8952
Fold 3	0.4995	0.9437	0.8950	0.8950
Mean	**0.4997**	**0.9432**	**0.8928**	**0.8928**

TABLE VI. ACCURACY OF MODELS FOR 5-FOLD CROSS-VALIDATION

K = 5				
	Perceptron	KNN	Naïve Decision Tree	Decision Tree with Pruning
Fold 1	0.4981	0.9415	0.8984	0.8984
Fold 2	0.5014	0.9409	0.8940	0.8940
Fold 3	0.4976	0.9430	0.8963	0.8963
Fold 4	0.5010	0.9444	0.8983	0.8983
Fold 5	0.4978	0.9417	0.8947	0.8947
Mean	**0.4992**	**0.9423**	**0.8963**	**0.8963**

TABLE VII. ACCURACY OF MODELS FOR 10-FOLD CROSS-VALIDATION

K = 10				
	Perceptron	KNN	Naïve Decision Tree	Decision Tree with Pruning
Fold 1	0.5040	0.9460	0.8891	0.8891
Fold 2	0.4922	0.9460	0.8860	0.8860
Fold 3	0.5010	0.9421	0.8898	0.8898
Fold 4	0.5019	0.9437	0.8948	0.8948
Fold 5	0.4993	0.9435	0.8908	0.8908
Fold 6	0.4960	0.9407	0.8866	0.8866
Fold 7	0.5042	0.9481	0.8931	0.8931
Fold 8	0.4978	0.9474	0.8917	0.8917
Fold 9	0.5039	0.9415	0.8930	0.8930
Fold 10	0.5005	0.9460	0.8904	0.8904
Mean	**0.5001**	**0.9445**	**0.8905**	**0.8905**

Form the table above, we notice that as k increases, the performance of classifiers might exhibit more variance. In k = 3, each fold represents a larger portion of the dataset, leading to different subsets for training and testing. In k = 10, each fold is smaller, providing a more granular assessment of model performance. Besides, a larger k increases the computational complexity of cross-validation. With k = 10, we train and test the model 10 times, making it computationally more intensive than k = 3. However, higher values of k (e.g., k = 10) can help improve the generalization ability of models by exposing them to diverse data partitions, making them less prone to overfitting.

B. Merging Predicted Results

We have developed an algorithm for merging predicted results from k classifiers within k-fold cross-validation. This algorithm incorporates the performance of individual classifiers, assigning weighted contributions based on their accuracy. We focus on harnessing the predictive power of the Perceptron, K-Nearest Neighbors (KNN), Naïve Decision Tree, and Decision Tree with Pruning models, weighting them accordingly to achieve a final prediction for each data point.

TABLE VIII. MEAN ACCURACY OF 5-FOLD CROSS VALIDATION AND PERCENTAGE OF MEAN ACCURACY OF EACH MODEL

	Perceptron	KNN	Naïve Decision Tree	Decision Tree with Pruning
Mean	0.4992	0.9439	0.8917	0.8917
Percentage	15.44%	29.14%	27.71%	27.71%

TABLE IX. COMPARE THE PERFORMANCE IN TESTING DATASET AND COMPLEXITY OF MODELS

	Accuracy in testing data	Running time (unit: second)
Perceptron	0.4997	872
KNN	0.9434	10649
Naïve Decision Tree	0.8894	75
Decision Tree with Pruning	0.8894	75
merge the predicted results (weighted vote)	0.8894	49232

The results indicate that the weighted voting approach did not yield superior performance. This could be attributed to the similarity in feature capture among the individual algorithms, resulting in a suboptimal amalgamation during the final merging process.

C. Model Performance Comparison

To compare whether the four models are similar in performance or if one model is superior to the others, I initially conducted 5-fold cross-validation on each of the four models. Subsequently, I employed the **Friedman non-parametric test** to compare the models, with each model having five accuracy values (one for each fold) under consideration.

The null hypothesis posited that the performance of the four models is approximately equal, while the alternative hypothesis proposed that there is at least one model with a statistically significant difference in performance. This methodology enabled me to determine whether one model is indeed superior to another.

$$H_0: \eta_1 = \eta_2 = \eta_3 = \eta_4 \quad (1)$$

This expression represents a null hypothesis that assumes the performance of the four models is similar, with no significant differences. The symbol $\boldsymbol{\eta}$ denotes median. Therefore, this null hypothesis posits that the median performance of the four models is similar, indicating an equal performance level around the median.

$$T = \frac{12}{k(k+1)} \sum_{i=1}^{k} \frac{R_i^2}{b} - 3b(k+1) \sim^{H_0} \chi^2(k-1) \quad (2)$$

The equation above represents the formula for the Friedman test statistic. In (2), **k** represents the number of treatments, which, in this context, is four (referring to the four models). **b** represents the number of blocks, which, in this context, is five (related to the five accuracy values from cross-validation). $\boldsymbol{R_i}$ denotes the rank sum of treatment i.

Under the assumption of the null hypothesis, the Friedman test statistic follows a chi-squared (χ^2) distribution with degrees of freedom equal to $k-1$. This implies that if the null hypothesis is true (i.e., there are no significant differences among the models), the Friedman statistic will conform to a chi-squared distribution with $k-1$ degrees of freedom.

TABLE X. TEST STATISTIC AND P-VALUE OF FRIEDMAN TEST

Friedman non-parametric test	
Test statistic	9.2000
P-value	0.0563

Based on the table above, we observe that the p-value is greater than 0.05, indicating a non-rejection of the null hypothesis. This suggests that **there are no significant differences among the four models**. However, it is important to note that the p-value is very close to 0.05, and considering the substantial disparity in accuracy between the Perceptron model and the other three models, it is likely that the non-rejection is due to the small size of the dataset. If the experiment were to be conducted with a 10-fold cross-validation, providing each model with ten data points, it is highly probable that the null hypothesis would be rejected. This would imply that at least one of the four models exhibits significant differences in performance.

VII. CONCLUSION

Our research into supervised learning, specifically car insurance claim prediction, uncovered several key challenges. The primary difficulties revolved around selecting the right classification algorithm, feature engineering complexities, data imbalance, and deciding the optimal k-fold cross-validation.

One significant revelation was the superior performance of Decision Trees with Pruning, emphasizing the algorithm's pivotal role in predictive accuracy. Feature engineering complexities became apparent, particularly when transforming variables like 'torque' and 'power' into ratios with 'RPM'. Data imbalance presented a major hurdle, prompting the application of SMOTE to tackle skewed datasets. Cross-validation, with different fold counts (k = 3, 5, 10), highlighted the trade-off between model stability and resource

consumption. An ensemble approach, combining predictions from multiple classifiers, offered promise in overcoming individual model limitations. The implications of these findings extend to the insurance industry, offering potential improvements in risk assessment, policy pricing, and fraud detection.

For future research, advanced machine learning techniques can be explored to enhance model performance and adapt to evolving insurance trends. Emphasizing model interpretability can make these models more practical for real-world insurance scenarios. In summary, our study navigated the complexities and challenges of supervised learning in car insurance claim prediction, offering insights that can drive advancements in the insurance sector and guide future machine learning endeavors.

REFERENCES

[1] Iftesha Najnin (2022). Kaggle Data Set:Car Insurance Claim Prediction. Car Insurance Claim Prediction (kaggle.com)

[2] Zubin Relia (2023). Car Insurance Claim Classifier. Car Insurance Claim Classifier | Kaggle

[3] ChatGPT