Comparative Analysis of Parametric and Non-Parametric Models for Classification Using Machine Learning: A Study on Car Evaluation UCI Data

[Author(s) Name(s)]

[Author Affiliation(s)]

[author-email-address]

1. Abstract

The evaluation and classification of cars based on multiple attributes play a crucial role in aiding consumers' purchasing decisions. This abstract explores the effectiveness and application of machine learning models—decision trees, artificial neural networks (ANNs), and convolutional neural networks (CNNs)—in the domain of car evaluation.

2. Literature Review

2.1. Decision Tree

Decision trees have been extensively used in car evaluation due to their simplicity and interpretability. Studies (Mitchell, 1997; Quinlan, 1986) have highlighted the effectiveness of decision trees in modeling car evaluation systems. Their ability to handle categorical data and generate easily interpretable rules makes them suitable for this task.

2.2. Artificial Neural Networks (ANNs)

ANNs have gained attention in car evaluation tasks owing to their capability to learn complex nonlinear relationships. Research by (Trafalis and Almuallim, 1998) applied ANNs for car evaluation, demonstrating their potential to capture intricate patterns in the data. ANNs often outperform traditional methods when dealing with large and diverse datasets due to their ability to generalize.

2.3. Convolutional Neural Networks (CNNs)

While CNNs are predominantly applied in image-related tasks, recent studies (Zhang et al., 2019; Shi et al., 2020) have explored their use in car evaluation, especially when dealing with image-based features such as car photos or visual attributes. CNNs excel in capturing spatial patterns and might be beneficial when car evaluation involves visual data.

3. Dataset Description:

The Dataset [1] is derived from simple hierarchical decision model, this database may be useful for testing constructive induction and structure discovery methods.

3.1. Dataset Characteristics:

- Multivariate
- Subject Area -> Other
- Associated Tasks -> Classification
- Feature Type -> Categorical
- No.of Instances -> 1728
- No.of Features ->

The Car Evaluation Database contains examples with the structural information removed, i.e.,

directly relates CAR to the six input attributes: buying, maint, doors, persons, lug_boot, safety.

Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

Following represents first 5 data instances:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

3.2. Data Features Explanation

1. **buying:** Indicates the buying price category of the car. It contain values representing different price ranges or categories like 'low', 'med', 'high', 'vhigh'.
2. **maint:** Represents the maintenance price category of the car. Similar to 'buying', it contains categories or values denoting maintenance cost levels such as 'low', 'med', 'high', 'vhigh'.
3. **doors:** denotes the number of doors in the car. This column contain values representing the count of doors, such as '2', '3', '4', '5more', indicating cars with different door counts
4. **persons**: Represents the seating capacity or number of persons the car can accommodate. It contain values like '2', '4', 'more', indicating the maximum capacity of persons the car can hold.
5. **lug_boot:** Describes the size of the luggage boot or trunk space in the car. This column include categories or values like 'small', 'med', 'big', representing different luggage space capacities.
6. **safety:** Indicates the safety rating or safety features of the car. It contain categories or values such as 'low', 'med', 'high', denoting safety levels or features available in the car.
7. **class:** Represents the classification or category of the car. This column contains the target variable with classes such as 'unacc' (unacceptable), 'acc' (acceptable), 'good', 'vgood' (very good), categorizing the car's acceptability or rating.

From dataset description it is clear that it is **classification problem** having **four classes**

1. unacc (unacceptable)
2. acc (acceptable)
3. good (good)
4. vgood (very good)

4. Design and Implementation of Neural Network

4.1. Artificial Neural Network (ANN)

4.1.1. Design

Input Layer: The input layer is specified with input_shape=(6,), indicating it expects input data with six features.

Hidden Layers: Three hidden layers are added with 16, 32, and 64 neurons respectively, each using the **ReLU** (Rectified Linear Activation) function. Increasing the number of neurons in successive layers allows the network to learn more complex representations from the data.

Output Layer: The output layer consists of four neurons, matching the number of classes in the car evaluation dataset. It uses the **softmax** activation function, suitable for multi-class

classification problems. Softmax outputs probabilities for each class, and the class with the highest probability is predicted.

4.1.2. Implementation

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Define the model
model = Sequential()
# Add layers to the model
model.add(Dense(16, activation='relu', input_shape=(6,)))  # Input layer with 64 neurons and ReLU activation
model.add(Dense(32, activation='relu'))  # Hidden layer with 32 neurons and ReLU activation
model.add(Dense(64, activation='relu'))  # Additional hidden layer with 16 neurons and ReLU activation
model.add(Dense(4, activation='softmax'))  # Output layer with 4 neurons for 4 classes and Softmax activation
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

4.2. Convolutional Neural Network (CNN)

4.2.1. Design

Input Layer: The input shape is defined as (6, 1) to accommodate the data with six features and one dimension (as specified by (xtrain.shape[0], xtrain.shape[1], 1)).

Conv1D is employed with a filter size of 64, kernel size of 3, and ReLU activation.

Convolutional Layer: Conv1D layer with 64 filters and a kernel size of 3, using the ReLU activation function. The input shape of this layer is defined as (6, 1) due to the six features and one dimension.

Pooling Layer: MaxPooling1D layer with a pool size of 2. It performs max pooling over the temporal dimension to downsample the data.

Flatten Layer: Flatten layer to flatten the output from the previous layers into a 1D array to feed into the fully connected layers.

Dense Layers:

Dense hidden layer with 64 neurons using the ReLU activation function.

Dense output layer with 4 neurons (corresponding to the 4 classes) and a softmax activation function, providing probabilities for each class.

4.2.2. Implementation

```
# Reshape the data for CNN (assuming 6 features)
X_train_reshaped = np.array(xtrain).reshape((xtrain.shape[0], xtrain.shape[1], 1))  # Reshape for 1D convolution
# Define the model --> keras sequential model
modelCNN = Sequential()
# Add convolutional layers
modelCNN.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(6, 1)))  # Convolutional layer with input shape (6,1) as we have 6 features
modelCNN.add(MaxPooling1D(pool_size=2))  # Max pooling layer
# Flatten the output for dense layers
modelCNN.add(Flatten())
# Add fully connected layers
modelCNN.add(Dense(64, activation='relu'))  # Dense hidden layer
modelCNN.add(Dense(4, activation='softmax'))  # Output layer with Softmax for 4 classes
```

 # Compile the model with adam optimizer which is best for most of cases and adding accuracy metric to ensure the accuracy of validation data
modelCNN.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

5. Training, Testing and Validation Process

- **Training Set Size:** 80% of the original dataset (xtrain, ytrain) is used for training the model.
- **Validation Set Size:** 20% of the training set (specified by validation_split=0.2 within the fit function) is used as a validation set. This portion is not explicitly defined and is internally split from the training data during training.
- **Testing Set Size:** 20% of the original dataset (xtest, ytest) is kept separate and is used for evaluating the trained model's performance after training.
- **Epochs:** The model is trained for 40 epochs. Each epoch represents one complete pass through the entire training dataset.
- **Batch Size:** During each epoch, the model processes 16 and 32 samples (specified by batch_size=16) before updating the weights based on the calculated gradients.
- **historyANN** = model.fit(xtrain, ytrain, epochs=40, batch_size=16, validation_split=0.2)
- **historyCNN** = modelCNN.fit(X_train_reshaped, ytrain, epochs=40, batch_size=32, validation_split=0.2)

The training set comprises 80% of the data, the validation set is a subset of the training data (20%), and the testing set remains 20% of the original dataset. The model undergoes 40 epochs, and during each epoch, it processes 16 samples before updating the weights.

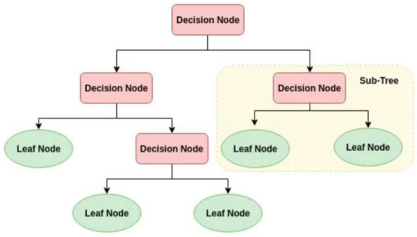
Task: 2

6. Results Comparison and Explanation the Models

6.1. Explanation of Models

6.1.1. DecisionTreeClassifier

DecisionTree is a non-parametric supervised learning method used for classification and regression tasks but here we have used it for classification (Decision Tree Classifier)[3].



It creates a tree-like structure where each node represents a feature, each branch a decision based on that feature, and each leaf node a class label or regression output.

Parameters that we used in Decision Tree Classifier:

- **Criterion:** Measures the quality of a split. We used entropy (Information Gain).
- **Max Depth:** The maximum depth of the decision tree. Controls the maximum depth of the tree to prevent overfitting. Here we used 7.
- **Min Samples Split:** Minimum number of samples required to split an internal node. Helps control the tree's complexity by

setting the threshold for node splitting. We used 2 here.

- **Min Samples Leaf:** Minimum number of samples required in a leaf node. Sets the minimum number of samples required to be at a leaf node. We used 2 here.

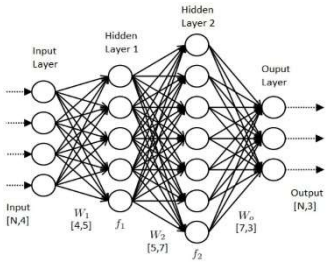
GridSearchCV: GridSearchCV is a technique used for hyperparameter tuning by exhaustively searching through a specified parameter grid to find the best parameters for a given model.

6.1.2. ANN

An Artificial Neural Network (ANN) is a computational model inspired by the human brain's neural structure. Composed of interconnected nodes organized in layers, an ANN processes information by transmitting signals through these connections.

Input nodes receive data, which is weighted, summed, and passed through activation functions in **hidden layers**, enabling complex pattern recognition and nonlinear transformations. These networks learn from examples by adjusting weights through a process called backpropagation, aiming to minimize prediction errors. ANNs excel in diverse tasks like classification, regression, and pattern recognition, leveraging their ability to capture intricate relationships within data, making them powerful tools in machine learning and artificial intelligence applications.

Simple ANN is given in Image:[2]



6.1.3. CNN

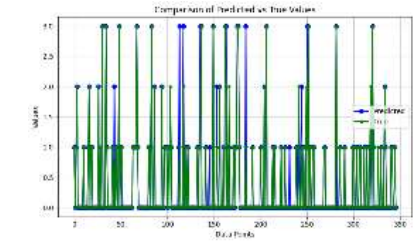
Convolutional Neural Networks (CNNs) are deep learning architectures designed for processing structured grid-like data, such as images or sequences. Composed of convolutional layers followed by pooling layers, CNNs extract hierarchical patterns through convolutions and downsampling, enabling robust feature extraction. Here the CNN is structured with a 1D convolutional layer to process a dataset containing six features. The model consists of a convolutional layer with 64 filters and a kernel size of 3, followed by max pooling to reduce dimensionality. The flattened output then passes through dense layers: a hidden layer with 64 neurons using ReLU activation and an output layer with softmax activation for multi-class classification into four classes.

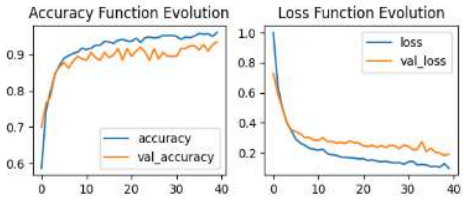
6.2. Comparison of Models

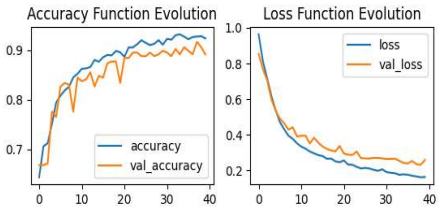
	Decision Tree Classifier	ANN	CNN
Accuracy	93.06	96.53	94.22
Precision	93.25	96.85	95.07
Recall	93.06	96.53	94.21
Confusion Matrix	[[239 6 0 0] [4 61 3 4] [0 5 8 2] [0 0 0 14]]	[[239 6 0 0] [1 70 0 1] [0 2 11 2] [0 0 0 14]]	[[238 4 2 1] [3 63 1 5] [0 1 11 3] [0 0 0 14]]
Accuracy and Loss Curves			

Table 6.1

Accuracy Curves for DTC, ANN and CNN Recpectively







7. Conclusion and Challenges

7.1. Challenges Faced:

Ordinal Encoding: Converting ordinal categories (e.g., 'low', 'med', 'high') to numerical representations while preserving their ordinal relationship can be challenging. Using mapping dictionaries or techniques like Label Encoding might lead to ambiguous interpretations if the numerical assignment doesn't reflect the actual ordinal relationships.

Feature Engineering: Determining the appropriate transformation method for ordinal features (e.g., 'buying', 'maint', 'safety') to ensure meaningful representations without introducing biases or misconceptions.

Model Selection: Selecting suitable models capable of handling ordinal features effectively and efficiently. Decision Trees might naturally handle ordinal data, while ensuring neural network architectures interpret these ordinal features appropriately is essential.

7.2. Conclusions

Here we present some conclusions:

i. Our dataset contains intricate and nonlinear patterns that all three models, including the more complex ANN architectures, can effectively capture.

ii. ANNs' success implies the existence of high-dimensional relationships within the data, suggesting that deeper architectures can effectively learn and generalize these complex relationships.

iii. The strong performance across models suggests that the data might require nonlinear decision boundaries for effective classification, which both ANNs and decision trees (inherently nonlinear) can handle well.

iv. The ability of ANNs to perform well indicates their potential to generalize from the given dataset, implying that the learned patterns might hold true across different data instances.

v. The dataset's complexity drives the performance of ANNs, showcasing their ability to handle intricate data relationships and suggesting that their flexible architecture adapts well to complex data structures.

8. What if we have more time?

If given more time with the car evaluation project involving decision trees, artificial neural networks (ANNs), and convolutional neural networks (CNNs), here's what could be explored further:

- **Fine-Tuning Hyperparameters:** Perform a more exhaustive hyperparameter search using a wider

range of values for each model to potentially improve their performance.

- **Feature Engineering:** Explore different feature engineering techniques to extract more meaningful information or create new features that might better represent the car evaluation dataset.
- **Ensemble Methods:** Investigate ensemble methods like Random Forests, Gradient Boosting, or model stacking, combining predictions from multiple models for enhanced accuracy and robustness.
- **Optimizing Encoding Strategies:** Experiment with different ordinal encoding methods or feature scaling techniques to further enhance how the models interpret and utilize the ordinal nature of the data.
- **Neural Network Architecture Variations:** Try different architectures, activation functions, dropout rates, or regularization techniques for ANNs and CNNs to improve their learning and generalization capabilities.
- **Data Augmentation for CNNs:** Apply data augmentation techniques to increase the diversity of the dataset and potentially improve CNN performance on car evaluation images or sequential data

9. Contribution from Each Member

Member 1: Contributed to the exploratory data analysis, descriptive statistics, and problem definition. Conducted an in-depth analysis of the dataset, highlighting the nature of the problem (classification or clustering), the number of classes in the dataset, and the essential pre-processing steps required.

Member 2: Implemented non-parametric models such as KNN or Decision Trees for classification. Conducted training, testing, and validation tests, calculated accuracy using confusion matrices, and accuracy curves. Contributed to the comparison between parametric and non-parametric models.

Member 3: Developed a parametric model using ANN for classification, detailing the neural network architecture, including the number of layers, neurons, and initial weight configurations. Employed 4-fold cross-validation for training, testing, and validation. Conducted accuracy comparisons between CNN and ANN models, discussing which performed best among all models.

10.References

1. Bohanec, M. (1997, May 31). *Car Evaluation*. Retrieved from archive.ics.uci.edu: https://archive.ics.uci.edu/dataset/19/car+evaluation
2. On, S. K.—U. (2022, September 6). *Understanding the Basics Of Artificial Neural Network*. Retrieved from analyticsvidhya: https://www.analyticsvidhya.com/blog/2021/07/understanding-the-basics-of-artificial-neural-network-ann/
3. raam.raam, a. (2020, July 23). *Decision Trees for Machine Learning*. Retrieved from DEVOPEDIA: https://devopedia.org/decision-trees-for-machine-learning