Comparative Analysis of Machine Learning Algorithms

Vikash Singh John, Abhi Singh Chadha
vxs465,axc1086

March 8, 2024

Introduction

This study delves into the application and comparative analysis of several machine learning algorithms across two distinct datasets. Emphasizing data preprocessing, we employed Min-Max Scaling, Z-Score Normalization, and Label Encoding to standardize our datasets, ensuring optimal model input. Employing a 10-fold cross-validation methodology, we rigorously evaluated each model's performance on metrics such as accuracy, precision, recall, and F1 score. The exploration spans algorithms like KNN, Naive Bayes, SVM, Decision Trees, and AdaBoost, aiming to uncover their efficacies and limitations in varied data landscapes, thus guiding informed algorithmic selections tailored to specific data characteristics.

Methodology

Data Preprocessing

Data preprocessing is tailored for two distinct datasets, incorporating normalization and encoding techniques to standardize features and facilitate model training:

1. For **Dataset 1**, normalization is employed to scale feature values:
    - *Min-Max Scaling:* Scales each feature to a range between 0 and 1.
    - *Z-Score Normalization:* Standardizes features to have a mean of 0 and a standard deviation of 1.
2. For **Dataset 2**, preprocessing includes defining custom features for clarity and handling categorical data:
    - Custom features such as 'Age', 'Tobacco', 'LDL', and others are defined to structure the dataset, improving readability and analysis.
    - Categorical variables, notably 'FamHist', are transformed into a format amenable to machine learning algorithms via one-hot encoding. This process creates separate columns for each category within 'FamHist', utilizing `pd.get_dummies` to accomplish this.

This comprehensive preprocessing approach ensures both datasets are optimally prepared, with Dataset 2 receiving additional structuring and categorical encoding to accommodate machine learning tasks.

The Min-Max Scaling :

$$x_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where x is the original value, $\min(x)$ and $\max(x)$ are the minimum and maximum values of the feature, respectively.

The Z-Score Normalization (Standardization):

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

where x is the original value, μ is the mean of the feature values, σ is the standard deviation of the feature values.

Label encoding numerically transforms categorical labels. For a feature with categories $c_1, c_2, \ldots, c_n$, it assigns an integer i to each category c_i, where $i \in \{0, 1, \ldots, n-1\}$.

Euclidean Distance

The Euclidean distance between two points x_1 and x_2 in an n-dimensional space is calculated as:

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

Evaluation Metrics

Confusion Matrix

A confusion matrix is a table used to describe the performance of a classification model. It includes:

- **TP (True Positives)**: Correct positive predictions.
- **TN (True Negatives)**: Correct negative predictions.
- **FP (False Positives)**: Incorrect positive predictions.
- **FN (False Negatives)**: Incorrect negative predictions.

Accuracy

Accuracy measures the proportion of correct predictions among the total number of cases examined.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Precision assesses the correctness of positive predictions made by the model.

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

Recall

Recall (or Sensitivity) evaluates the model's ability to detect all actual positives.

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

F1 Score

The F1 Score is the harmonic mean of precision and recall, balancing the two metrics.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

KNN Algorithm

K-Nearest Neighbors (KNN) is a non-parametric, lazy learning algorithm used for both classification and regression tasks. It classifies a new sample based on the majority vote of its k nearest neighbors for classification tasks.

- For each test point, calculate the Euclidean distance to all training points.
- Sort these distances and select the k nearest neighbors.
- The most common class label among these neighbors is assigned to the test point. This is determined by:

  $$\text{label} = \text{mode}\{y_{i1}, y_{i2}, ..., y_{ik}\}$$

  where y_{ij} represents the label of the j-th nearest neighbor.

The choice of using Euclidean distance for measuring similarity and the iterative approach for selecting k ensures the algorithm is both effective in classifying instances and robust against overfitting by avoiding too small or too large values of k.

Support Vector Machines (SVM)

The SVM implementation focuses on linear classification, with the key steps as follows:

Initialization

- Set the learning rate (lr), regularization parameter (λ), and the number of iterations (n_iters).
- Initialize the weight vector ($\mathbf{w}$) and bias (b) to zeros.

Model Training

1. Convert the labels to -1 and 1, where -1 represents one class and 1 the other.
2. For each iteration, update $\mathbf{w}$ and b as follows:

   $$\mathbf{w} \leftarrow \mathbf{w} - \text{lr} \times (2\lambda\mathbf{w} - \mathbf{x}_i y_i) \quad \text{if } y_i(\mathbf{x}_i \cdot \mathbf{w} - b) < 1$$

   $$b \leftarrow b - \text{lr} \times y_i \quad \text{if } y_i(\mathbf{x}_i \cdot \mathbf{w} - b) < 1$$

3. If the condition $y_i(\mathbf{x}_i \cdot \mathbf{w} - b) \geq 1$ is met, only the weight vector is updated:

   $$\mathbf{w} \leftarrow \mathbf{w} - \text{lr} \times 2\lambda\mathbf{w}$$

Prediction

- The prediction for a new sample $\mathbf{x}$ is given by the sign of $\mathbf{x} \cdot \mathbf{w} - b$.
- The output is -1 or 1, representing the predicted class based on the sign of the decision function.

This SVM implementation employs a linear kernel with regularization to prevent overfitting, optimizing the model's generalization to unseen data. The choice of a linear kernel simplifies the optimization problem, making the algorithm efficient for linearly separable datasets.

Naive Bayes

The Naive Bayes implementation follows these key steps:

Naive Bayes Model

1. The model initializes by calculating the prior probabilities, means, and variances for each class based on the training data.
2. For a given test instance, the posterior probability for each class is computed using the formula:

   $$P(c|x) = P(x|c)P(c) = \prod_{i=1}^{n} P(x_i|c)P(c)$$

   where $P(x_i|c)$ is the likelihood of feature i given class c, modeled using a Gaussian distribution:

   $$P(x_i|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{(x_i - \mu_c)^2}{2\sigma_c^2}\right)$$

   μ_c and σ_c^2 are the mean and variance of feature i for class c, respectively.
3. The class with the highest posterior probability is chosen as the prediction for the test instance.

This implementation of Naive Bayes focuses on binary classification, utilizing statistical properties of the features to make predictions. The model's simplicity and probabilistic nature make it effective for datasets where the features are conditionally independent given the class label.

Decision Trees

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Gini Index

The Gini index for a split is calculated as:

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2$$

where C is the number of classes and p_i is the proportion of samples in class i within the group.

Splitting Criterion

To split the data, the algorithm iterates over all features and their values, evaluating the Gini index for each potential split and selecting the one that results in the lowest Gini index.

Building the Tree

1. Starting at the root, the best split is determined using the Gini index.
2. The dataset is split into two groups based on this criterion, forming two child nodes.
3. This process is recursively applied to each child node, with the recursion stopping when either the node reaches a specified maximum depth or the number of samples at the node falls below a minimum size.
4. At each terminal node, the most frequent class among the samples in the node is chosen as the node's predicted class.

Prediction

Given a new sample, the tree routes it down to a terminal node, using the feature splits at each node to decide the path. The prediction is the class assigned to the terminal node reached by the sample.

AdaBoost with Decision Trees

Building on the Decision Trees, AdaBoost uses these as base learners in an ensemble method, enhancing classification performance. The AdaBoost algorithm iteratively adjusts the weights of incorrectly classified instances, focusing the learning on the more difficult cases.

Algorithm Overview

The algorithm initializes equal weights for all training instances. Each iteration involves fitting a Decision Tree to the weighted data, calculating its error rate, and adjusting the model and instance weights based on performance. The final model aggregates the trees, prioritizing those with lower error rates.

Prediction

A new instance is classified by summing the weighted predictions of the decision trees. The sign of the sum determines the class:

$$\text{class} = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

where T is the number of trees, α_t is the weight of tree t, and $h_t(x)$ is the prediction of tree t for instance x.

AdaBoost effectively combines multiple weak learners to form a robust classifier, with each successive learner focusing more on the instances that previous learners misclassified.

Evaluation and Performance

To assess the performance of the implemented models on both datasets, a 10-fold cross-validation approach is employed. This method divides the dataset into ten equal parts, where each part is used as a test set while the remaining parts are used as the training set. This process is repeated ten times, with each part serving as the test set exactly once. The average performance across all folds is reported for metrics such as accuracy, F1 score, precision, and recall.

Dataset 1 Performance Metrics

Model	Acc.	F1	Prec.	Rec.
KNN (K=9)	0.971	0.960	0.987	0.932
Naive Bays	0.935	0.910	0.918	0.904
SVM	0.968	0.957	0.981	0.935
Dec. Tree	0.914	0.907	0.907	0.910
AdaBoost (DT)	0.952	0.936	0.939	0.936

Dataset 2 Performance Metrics

Model	Acc.	F1	Prec.	Rec.
KNN (K=36)	0.734	0.667	0.689	0.701
Naive Bays	0.705	0.589	0.574	0.616
SVM	0.725	0.542	0.623	0.50
Dec. Tree	0.697	0.595	0.650	0.611
AdaBoost (DT)	0.672	0.433	0.431	0.454

Hyperparameter Tuning and Sensitivity Analysis

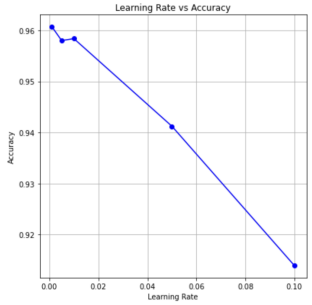
Hyperparameter Tuning: Crucial for optimizing model performance, techniques like grid and random search are employed across models (KNN, Naive Bayes, SVM, Decision Trees, AdaBoost) to find the best parameter settings.

Fine-Tuning:

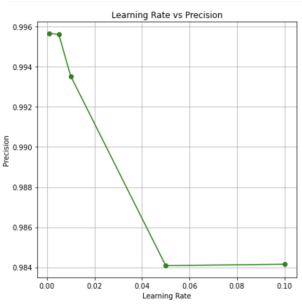
- **KNN:** Adjusting the number of neighbors (k) and exploring distance metrics.
- **Naive Bayes:** Selecting the appropriate distribution (Gaussian or Multinomial) based on data.

- **SVM:** Fine-tuning kernel type, regularization parameter (C), and kernel coefficient ($gamma$).
- **Decision Trees:** Modifying tree depth, split criteria, and minimum samples for a node split.
- **AdaBoost:** Adjusting the number of estimators, learning rate, and base estimator configuration.

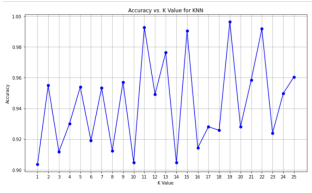
Each model's hyperparameters are meticulously fine-tuned to strike a balance between bias and variance, with cross-validation ensuring robustness and generalizability of the models.



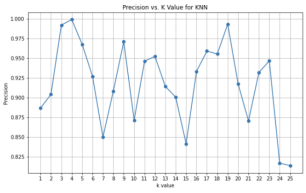
(a) SVM: Accuracy vs lr



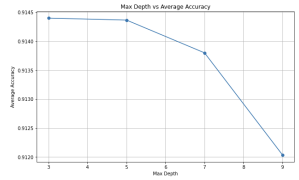
(b) SVM: Precision vs lr



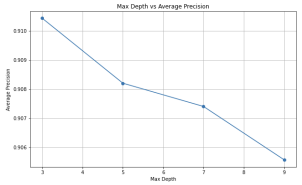
(c) KNN: Accuracy vs K



(d) KNN: Precision vs K



(e) DT: Accuracy vs max-depth



(f) DT: Precision vs max-depth

Figure 1: Performance on different hyperparameters

Kernel	Accuracy	Precision
linear	0.975407	0.975165
poly	0.903352	0.993333
rbf	0.973653	0.976151
sigmoid	0.964881	0.973332

Table 1: SVM Kernel Performance

The image presents a comparative analysis of various machine learning algorithms, highlighting their sensitivities to hyperparameter tuning. Notably, the SVM model demonstrates a pronounced reduction in accuracy and precision as the learning rate escalates, suggesting a careful calibration is required for optimal performance. In contrast, the KNN algorithm exhibits pronounced fluctuations in performance with different K values, indicating a critical dependency on the choice of neighbors. The Decision Tree's performance steadily deteriorates with increasing depth, implying a potential for overfitting. A detailed breakdown in the table below confirms that the linear kernel achieves superior accuracy and precision among the SVM kernels, while the polynomial kernel, despite its lower accuracy, achieves remarkable precision, suggesting a complex trade-off between these metrics.

Conclusion and Comparative Analysis

For Dataset 1, the KNN and SVM models show impressive accuracy and F1 scores, indicating strong performance in a possibly less complex or noisy dataset. AdaBoost, leveraging Decision Trees, also performs well, showcasing the effectiveness of ensemble methods in improving prediction accuracy.

In contrast, Dataset 2 presents a more challenging scenario, with all models experiencing a drop in performance metrics. This could be due to higher complexity, more noise, or more challenging patterns in the data. KNN's performance drop might suggest sensitivity to noise and data scale. Naive Bayes, while experiencing a decrease, maintains a balance in precision and recall, indicating resilience in handling complex data distributions. SVM's precision remains relatively high, but the drop in recall could indicate difficulty in generalizing to more complex data. Decision Trees and AdaBoost show a marked decrease in performance, suggesting possible overfitting or the need for more sophisticated ensemble techniques to handle this dataset's complexity.

Pros and Cons Based on Findings:

KNN: Highly accurate for simpler datasets but sensitive to noise and dimensionality, leading to decreased performance on more complex data.
Naive Bayes: Offers a good balance of precision and recall, indicating robustness but with limitations in handling complex data interactions.
SVM: Maintains good precision across datasets, indicating effective margin maximization, but struggles with recall in complex scenarios, suggesting limitations in capturing all positive instances.
Decision Trees: Provide interpretable models but can be prone to overfitting, as evidenced by reduced performance on more complex datasets.
AdaBoost: Enhances Decision Tree performance but still falls short in more challenging datasets, indicating potential overfitting or the need for more diverse ensembles.
These insights highlight the importance of choosing the right model based on data complexity and the specific trade-offs between precision and recall, considering the underlying data distribution and problem context.

Acknowledgment

I extend our heartfelt gratitude to Dr. Jing Ma for her invaluable instruction and mentorship in the field of Data Mining. Her expertise and insightful guidance have been pivotal in the successful completion of this project. Her dedication to fostering a deep understanding of complex concepts has profoundly enhanced my learning experience.

References

[1] A. Moore, "The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data," in *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, 2000.

[2] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[3] G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 338–345.

[4] J. R. Quinlan, "C4.5: Programs for Machine Learning," *Morgan Kaufmann Publishers Inc.*, San Francisco, CA, USA, 1993.

[5] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," in *Proceedings of the Second European Conference on Computational Learning Theory*, 1995, pp. 23–37.

[6] "Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction," *Scientific Reports*, [Online].