Khushi Raghuvanshi

Zhuowen Tu

COGS-118A

13 December 2024

Different Classifiers Evaluation

Abstract

This project evaluates six supervised learning classifiers—Random Forest, Decision Tree, K-Nearest Neighbors (KNN), Naive Bayes, Support Vector Machine (SVM), and Logistic Regression—on binary classification tasks using three datasets from the UCI Machine Learning Repository. Classifiers were assessed for accuracy across different data splits (20/80, 50/50, 80/20) using GridSearchCV for hyperparameter tuning and metrics such as training, validation, and testing errors. Results showed that SVM and Random Forest performed best, demonstrating strong accuracy and robustness. Logistic Regression also performed well, balancing simplicity with effectiveness. KNN showed variability, likely due to sensitivity to hyperparameters and larger datasets, while Decision Trees tended to overfit. Naive Bayes performed the worst, struggling with correlated features. These results highlight SVM and Random Forest as reliable classifiers for binary classification, with simpler models like Logistic Regression being effective in specific contexts.

Introduction

This project was based on a paper titled “An Empirical Comparison of Supervised Learning Algorithms” by Caruana et al. (2006). In this paper the authors present a large-scale empirical comparison of ten supervised learning algorithms, including decision trees, support vector machines, neural networks, and others, to evaluate their performance across a variety of binary classification problems. Using the information provided in this paper, I chose 6 classifiers to compare across three selected datasets from the UCI Machine Learning Repository, a popular source for publicly available datasets.

The purpose of this project was to see which classifier performs the best, resulting in highest accuracy. Based on their data, I expected to see the classifiers perform in the following order from best to worst: Random Forest, SVM, KNN, Decision Tree, Logistic Regression, and Naive Bayes. This analysis serves to explore the relative strengths and weaknesses of these classifiers in practical binary classification scenarios, providing insights into which algorithms might be most effective in different contexts.

Method

Random Forest

Random Forest is an ensemble learning method that constructs a collection of decision trees during training. Each tree is trained on a random subset of the data, and at each split, a random subset of features is considered for the decision. This helps reduce overfitting and improves the model's generalization performance. To run this classifier, I used RandomForestClassifier from sklearn.ensemble.

Hyperparameters: n_estimators: [50, 100, 200], max_depth: [None, 10, 20]

Decision Tree

A Decision Tree is a hierarchical structure where each internal node represents a feature (attribute), and each branch represents a decision rule. The tree splits the dataset at each node based on the feature that results in the best separation of classes. Decision Trees are simple but prone to overfitting, especially with deep trees, that's why we normalize the values using StandardScalar(). To run this classifier, I used DecisionTreeClassifier from sklearn.tree.

Hyperparameters: max_depth: [None, 10, 20], min_samples_split: [2, 5, 10]

K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a simple, instance-based learning algorithm. Given a test instance, KNN finds the K training examples closest to it (based on a distance metric) and assigns the most common label among them to the test instance. To run this classifier, I used KNeighborsClassifier from sklearn.neighbors.

Hyperparameters: n_neighbors: [3, 5, 10], weights: ["uniform", "distance"]

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem. It assumes that the features are conditionally independent given the class label. The model computes the probability of each class based on the likelihood of the features given the class and selects the class with the highest probability. To run this classifier, I used GaussianNB from sklearn.naive_bayes.

Hyperparameters: var_smoothing: [1e-9, 1e-8, 1e-7]

Support Vector Machine (SVM)

Support Vector Machine is a powerful classifier that finds the optimal hyperplane in a high-dimensional space that best separates data points of different classes. SVM is effective in high-dimensional spaces and particularly useful when the number of dimensions exceeds the number of samples. It aims to maximize the margin (distance) between the closest points of the classes (support vectors). To run this classifier, I used SVC from sklearn.svm

Logistic Regression

Logistic Regression is a linear model for binary classification that estimates the probability of a class label based on a logistic (sigmoid) function. It models the relationship between the features and the log-odds of the target variable, which can be interpreted as a linear combination of the input features. Despite its name, Logistic Regression is a classifier, not a regression model. To run this classifier, I used LogisticRegression from sklearn.linear_model.

Hyperparameters: C: [0.1, 1, 10], max_iter: [7000, 10000]

For each dataset, I first split the data into training and testing data using train_test_split for each split 20/80, 50/50, and 80/20. Then I created a function called evaluate_classifier() to evaluate the performance of a classifier. The function takes a classifier, a set of hyperparameters, and training and testing data as input. It begins by performing a grid search with cross-validation using GridSearchCV() to identify the best hyperparameters for the classifier, using accuracy as the scoring metric. Once the grid search is complete, it extracts the best model.

I then calculated the training error by predicting the labels for the training data and comparing them to the actual values, computing the accuracy and then the error. For the validation error, I used the best cross-validation score obtained during the grid search. Similarly, I calculated the test error by predicting on the test data and comparing the predictions to the true labels, also computing the accuracy and error.

The function prints several key metrics, including the best parameters found by the grid search, the training accuracy and error, the validation accuracy and error, and the testing accuracy and error. It also outputs the classification report and confusion matrix for a deeper analysis of the model's performance. Finally, the function returns a dictionary containing the errors for training, validation, and testing, along with the best parameters for the classifier.

I used this function across all classifiers, Random Forest, Decision Tree, KNN, Naive Bayes, SVM, and Logistic Regression to calculate the best hyperparameters and find the training accuracy, validation accuracy, and testing accuracy for the different splits.

Data

For the first dataset, I imported it from sklearn.datasets. It contains information about breast cancer where the target feature has 0 if the tumor is benign and 1 if its malignant.

I split them into target y and design matrix X, and then used them for making the splits.

For the second dataset, which is the heart_disease_uci dataset, I split the dataset into y and X again. Then to take care of the missing values, I created a function handle_missing_values(X), which uses different imputation methods to fill in the nan values. For the numerical columns, it uses the mean, and for the categorical columns it uses the mode. After that to clean the data further, I wrote a function to preprocess the data by applying transformations to both categorical and numerical features. I started by creating a ColumnTransformer called preprocessor, which included a one-hot encoder for the categorical columns and a standard scaler for the numerical columns. The one-hot encoder was set to handle unknown categories by ignoring them, and the standard scaler was applied to the numerical columns.

Next, I implemented the preprocess_data function, which took the dataset X as input. Inside the function, I used the fit_transform method of the preprocessor to apply the transformations. I then extracted the feature names for the numerical and categorical columns and combined them into a single list of feature names. Using this list, I created a new DataFrame from the transformed data while preserving the original index of the input dataset.

For cases where I wanted to keep both the transformed numerical and categorical features, I wrote an additional function, preprocess_data_full, which followed a similar process. This function also applied the transformations and combined the feature names, ultimately returning a DataFrame with the transformed data.

Then I transformed y into binary values, where 0 meant no disease and anything in the range of 1-4 was transformed to 1, meaning disease was found.

The third dataset contains the income distribution for adults, where the target feature is income with 0 being =<$50K and 1 being >$50K. This dataset did not have any missing values, but similar to data2, I transformed the data in X using the preprocess_data_full().

Experiment

As shown in the tables below, we tested each classifier for three different data splits, 20/80, 50/50, and 80/20. Using the accuracy score model from sklearn.metrics, we calculated training accuracy and test accuracy across all splits for all classifiers. The validation accuracy was calculated using the following formula: validation_error = 1 - grid_search.best_score_. Tables 1, 2, and 3 represent the above metrics across Data1, Data2, and Data3 respectively. Table 4 represents average accuracies for each dataset across classifiers with different data splits. Table 5 shows average testing accuracy for each classifier, averaged across each dataset.

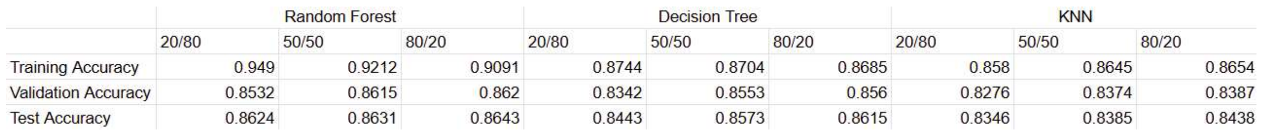
Table 1. Accuracies of all classifiers with different splits on Data1

	Random Forest			Decision Tree			KNN			Naive Bayes			SVM			Logistic Regression		
	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20
Training Accuracy	1	1	1	1	1	0.9956	0.9115	1	0.9473	0.9558	0.9401	0.9407	0.9912	0.993	0.9846	0.9646	0.9859	0.9692
Validation Accuracy	0.9644	0.9682	0.9626	0.9471	0.9471	0.9275	0.9203	0.9367	0.9494	0.9467	0.9365	0.9385	0.9822	0.9789	0.9781	0.9559	0.9648	0.9561
Test Accuracy	0.9364	0.9474	0.9474	0.8794	0.9123	0.9123	0.9101	0.9333	0.9123	0.9167	0.9439	0.9386	0.9605	0.9719	0.9825	0.932	0.9579	0.9649

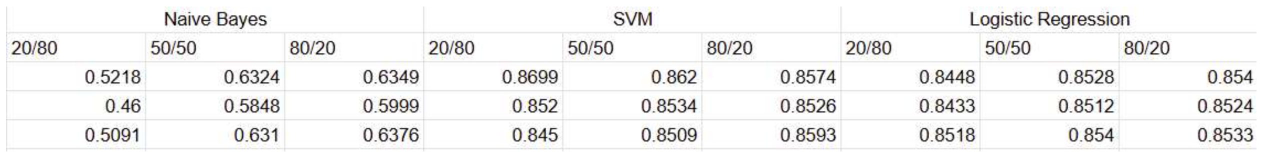
Table 2. Accuracies of all classifiers with different splits on Data2

	Random Forest			Decision Tree			KNN			Naive Bayes			SVM			Logistic Regression		
	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20
Training Accuracy	1	1	1	0.9348	0.913	0.9524	0.8804	1	0.8682	0.75	0.8109	0.8152	0.9783	0.9674	0.8438	0.8587	0.8457	0.8329
Validation Accuracy	0.8262	0.8152	0.8261	0.7664	0.7304	0.7336	0.8317	0.8391	0.822	0.7173	0.7979	0.8138	0.8099	0.8109	0.8274	0.8206	0.8152	0.8247
Test Accuracy	0.8302	0.8304	0.837	0.7962	0.75	0.7717	0.8166	0.8326	0.8315	0.7649	0.8196	0.875	0.8057	0.8196	0.8587	0.837	0.8174	0.8424

Table 3. Accuracies of all classifiers with different splits on Data3

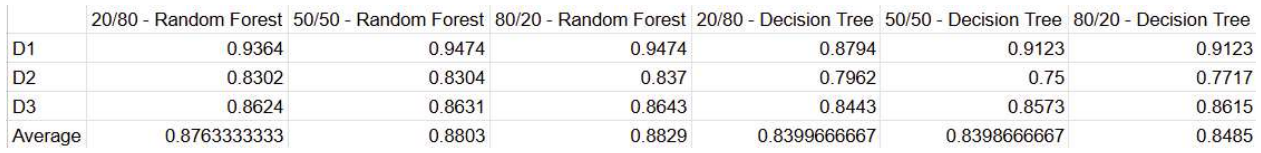


	Random Forest			Decision Tree			KNN		
	20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20
Training Accuracy	0.949	0.9212	0.9091	0.8744	0.8704	0.8685	0.858	0.8645	0.8654
Validation Accuracy	0.8532	0.8615	0.862	0.8342	0.8553	0.856	0.8276	0.8374	0.8387
Test Accuracy	0.8624	0.8631	0.8643	0.8443	0.8573	0.8615	0.8346	0.8385	0.8438

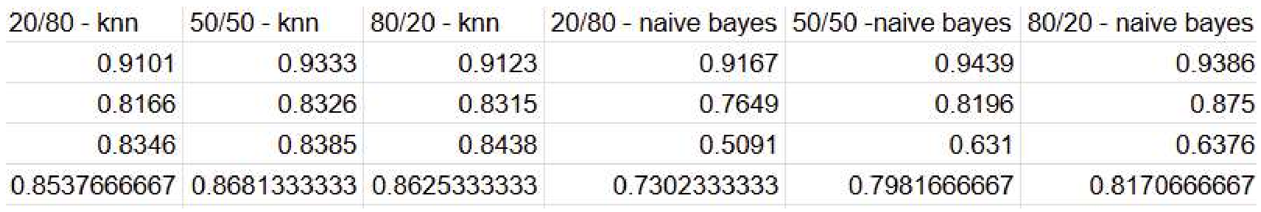


Naive Bayes			SVM			Logistic Regression		
20/80	50/50	80/20	20/80	50/50	80/20	20/80	50/50	80/20
0.5218	0.6324	0.6349	0.8699	0.862	0.8574	0.8448	0.8528	0.854
0.46	0.5848	0.5999	0.852	0.8534	0.8526	0.8433	0.8512	0.8524
0.5091	0.631	0.6376	0.845	0.8509	0.8593	0.8518	0.854	0.8533

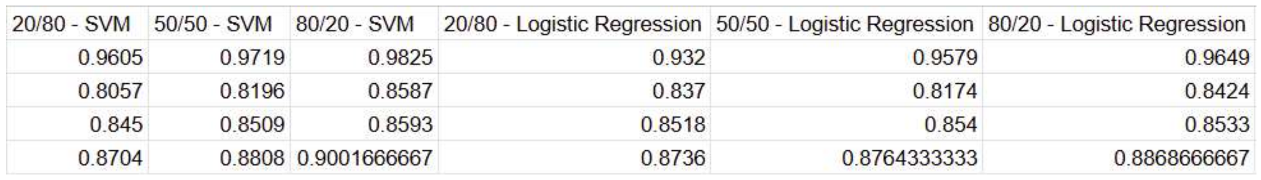
Table 4. Average Accuracies for each dataset across all classifiers with different splits



	20/80 - Random Forest	50/50 - Random Forest	80/20 - Random Forest	20/80 - Decision Tree	50/50 - Decision Tree	80/20 - Decision Tree
D1	0.9364	0.9474	0.9474	0.8794	0.9123	0.9123
D2	0.8302	0.8304	0.837	0.7962	0.75	0.7717
D3	0.8624	0.8631	0.8643	0.8443	0.8573	0.8615
Average	0.8763333333	0.8803	0.8829	0.8399666667	0.8398666667	0.8485



20/80 - knn	50/50 - knn	80/20 - knn	20/80 - naive bayes	50/50 -naive bayes	80/20 - naive bayes
0.9101	0.9333	0.9123	0.9167	0.9439	0.9386
0.8166	0.8326	0.8315	0.7649	0.8196	0.875
0.8346	0.8385	0.8438	0.5091	0.631	0.6376
0.8537666667	0.8681333333	0.8625333333	0.7302333333	0.7981666667	0.8170666667



20/80 - SVM	50/50 - SVM	80/20 - SVM	20/80 - Logistic Regression	50/50 - Logistic Regression	80/20 - Logistic Regression
0.9605	0.9719	0.9825	0.932	0.9579	0.9649
0.8057	0.8196	0.8587	0.837	0.8174	0.8424
0.845	0.8509	0.8593	0.8518	0.854	0.8533
0.8704	0.8808	0.9001666667	0.8736	0.8764333333	0.8868666667

Table 5. Table displaying the average accuracies across classifiers across training splits

Random Forest	Decision Tree	KNN	Naive Bayes	SVM	Logistic Regression
0.8829	0.8428	0.8615	0.7818	0.8838	0.879

Conclusion

After testing the six classifiers that I chose, I found that my analysis somewhat matched the expected order from the paper. My finding resulted in the following order from best to worst: SVM, Random Forest, Logistic Regression, KNN, Decision Trees and Naive Bayes. While this varies from the exact order shown in Caruana et al. (2006), there are similarities. I believe that the minor differences could be due to the smaller size of dataset I had in comparison to Caruana et al. (2006). In the paper, they are also using 10 different metrics to measure the correctness of each classifier, while I only use one, which could also be the reason for the differences.

SVM and Random Forest performed well across all datasets, likely due to their ability to handle complex relationships and overfitting (in the case of Random Forest). Logistic Regression also performed fairly well, suggesting that even relatively simple models can be competitive, especially when the data is well-behaved. KNN struggled in comparison, likely due to its computational inefficiencies with larger datasets and sensitivity to the choice of distance metric. Decision Trees showed some overfitting on certain datasets, which may explain why they did not perform as well as expected. Naive Bayes, with its assumptions about feature independence, performed the worst, particularly when the features were correlated.

Ultimately, the findings from this study provide valuable insights into the relative performance of these classifiers. In practical terms, SVM and Random Forest appear to be strong choices for a variety of binary classification problems, though simpler models like Logistic Regression may offer advantages in terms of computational efficiency and ease of interpretation.

References

Becker, B. & Kohavi, R. (1996). Adult [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5XW20.

Caruana, R., Niculescu-Mizil, A., Rich CaruanaCornell University, I., & Alexandru Niculescu-MizilCornell University, I. (2006, June 25). *An empirical comparison of supervised learning algorithms: Proceedings of the 23rd International Conference on Machine Learning*. ACM Other conferences. https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf

Janosi, A., Steinbrunn, W., Pfisterer, M., & Detrano, R. (1989). Heart Disease [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C52P4X.

Wolberg, W., Mangasarian, O., Street, N., & Street, W. (1993). Breast Cancer Wisconsin (Diagnostic) [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.