Hepatitis C Diagnosis using Machine Learning Classification Techniques

Ioannis Mystakidis - 1115201600113 - Undergraduate[1]

[1] *Department of Informatics and Telecommunications, University of Athens*

Abstract

Accurate disease prediction and classification are crucial in medical science as they play a significant role in preventing disease spread and identifying infected areas in their early stages. Machine learning (ML) techniques have emerged as an effective tool for predicting and classifying diseases, and their application in this field has gained considerable attention. In this study, we assessed the performance of distinct classifiers on a dataset of hepatitis C patients and healthy blood donors to develop a systematic strategy for disease diagnosis. To optimize the hyperparameters of the classifiers and evaluate their performance, we employed nested cross-validation. The classifiers used in this study were Logistic Regression (LR), Gaussian Naive Bayes (GNB), K-Nearest Neighbors (KNN), Linear Discriminant Analysis (LDA), and Support Vector Machines (SVM). We evaluated the performance of these classifiers in two different scenarios - with and without feature selection. We employed Univariate Feature Selection, Recursive Feature Elimination, and Tree-based Feature Selection as feature selection methods, which were treated as hyperparameters and optimized during nested cross-validation. Our results showed that the LR classifier was the most promising classifier when Feature Selection was ommited, whereas SVM performed better when Feature Selection was employed.

Keywords: Hepatitis C, Machine Learning, Supervised, Classification, Nested Cross-Validation

1 Introduction

Hepatitis C is a global health problem affecting an estimated 58 million people worldwide, with approximately 290,000 deaths attributed to the disease each year. It is a viral infection that primarily affects the liver and can lead to serious health complications such as cirrhosis, liver cancer, and liver failure. The disease is caused by the hepatitis C virus (HCV), which is primarily spread through blood-to-blood contact, such as sharing needles or receiving blood transfusions (World Health Organization 2023).

The treatment for hepatitis C has improved significantly over the years, with the development of antiviral drugs that can effectively cure the disease. However, despite these advancements, accurate prediction of disease progression and treatment outcomes remains a challenge. This is particularly important given that HCV infection is often asymptomatic, with up to 80% of infected individuals remaining undiagnosed and at risk of developing chronic liver disease (World Health Organization 2023).

Machine learning (ML) techniques offer a promising approach to effectively diagnose and classify patients with Hepatitis C, thereby enabling the development of more effective treatment protocols. In this study, we aim to develop a classifier that can accurately distinguish between Hepatitis C patients and healthy donors based on various features and laboratory test values.

In this study, we compare the results of various algorithms for this task using nested Cross Validation and try out the effect of Feature Selection on the improvement of the classification. The remainder of the study is structured as follows: Section 2 introduces the materials and research techniques. Section 3 discusses the results. Section 4 provides a conclusion. Section 5 contains supplementary material about the reproduction of the results.

2 Materials and Methods

2.1 Data Set

In this study, the Hepatitis C data were given to us as a curated dataset. The data set contains laboratory test values of blood donors and Hepatitis C patients and demographic values like age and sex. The target attribute for classification is a binary category (healthy blood donors vs. Hepatitis C). The data samples have the following features: Age, Sex, ALB, ALP, ALT, AST, BIL, CHE, CHOL, CREA, GGT, and PROT. Each sample has a label with 1 and 0 for disease or absence of disease, respectively. A small sample of the dataset is shown in Figure 1.

	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT	label
0	32	0	43.2	52.0	30.6	22.6	18.9	7.33	4.74	80.0	33.8	75.7	0
1	45	0	41.7	73.2	43.6	29.4	6.4	8.89	5.31	71.0	67.4	70.3	0
2	55	0	41.5	59.5	15.4	16.2	6.8	6.35	5.22	80.0	12.4	69.9	0
3	53	0	37.8	98.1	30.5	21.1	4.0	5.02	4.42	94.0	23.2	65.2	0
4	56	1	39.7	66.0	14.2	20.8	3.5	7.48	5.88	66.0	7.2	67.2	0

Figure 1. The first 5 rows of our dataset.

Exploratory Data Analysis: EDA is an essential part of machine learning and is the first step that starts after acquiring data (Dodge 2008). In this process, the original data is explored with as few a priori assumptions as possible, summarizing the structure of the data and presenting specific patterns. We've performed EDA with the help of the *pandas_profiling* package in python.

As shown in Figure 2, our dataset contains only a small number of samples. There are 13 variables, 12 features and 1 target variable, out of which 11 are numeric and 2 are categorical (Sex and Label). Our dataset is already pretty clean as it does not contain any missing values or duplicate rows.

Dataset statistics

Number of variables	13
Number of observations	204
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	20.8 KiB
Average record size in memory	104.6 B

Variable types

Numeric	11
Categorical	2

Figure 2. The main statistics of our dataset, e.g. number of samples, number of features, types of features etc.

In Figure 3, we can see the value of Pearson's r correlation coefficient between the different variables in our dataset. The correlation coefficient is a statistical measure of how strongly two variables are related to one another. By multiplying the two deviations from their respective means, the product-difference approach yields the correlation coefficient; this method is especially useful for calculating the linear single correlation coefficient. To be precise, -1 to +1 describes the range of the correlation coefficient. Positive correlation between two variables is shown by an r value greater than zero, whereas negative correlation is indicated by an r value less than zero. When $|r| = 1$,

there is a perfect linear correlation between the two variables; in other words, they are functionally related. If $r = 0$, then there is no linear relationship between the two metrics. Whenever $0 < |r| < 1$, linear correlation exists between the two variables. As $|r|$ approaches 1 (perfect linearity), the relationship strengthens; as it approaches 0 (poor linearity), the relationship weakens.

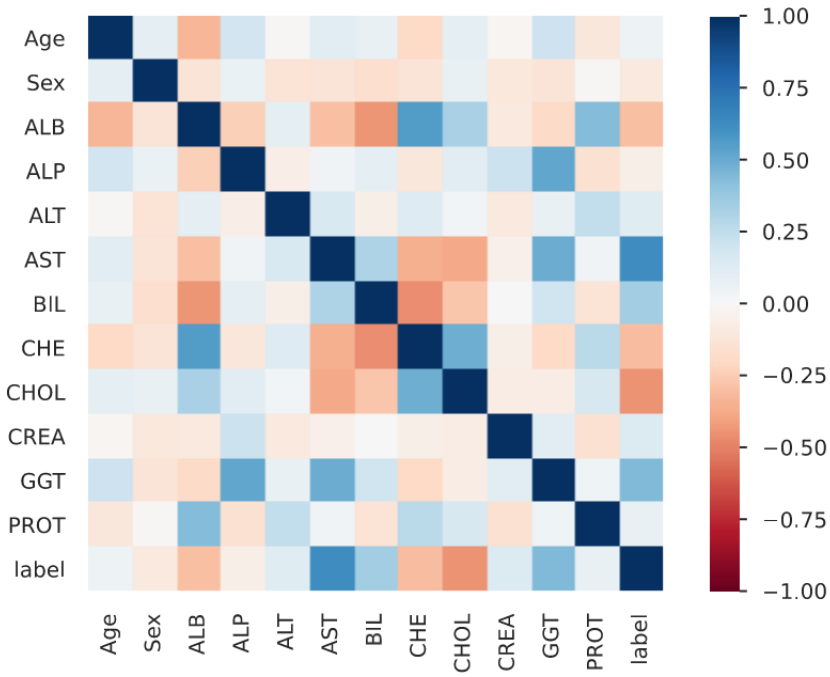
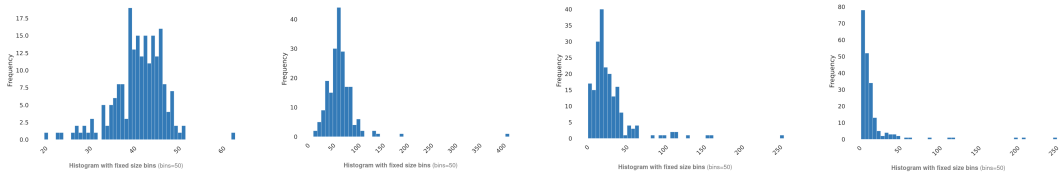


Figure 3. Pearson's r correlation coefficient between the dataset's variables.

The correlation analysis revealed that the feature AST is very important for the final target label. There are also BIL and GGT that contribute to some extent. Apart from the target label, there is a clear correlation between features ALB and CHE, ALB and PROT, and AST and GGT.

There also seems to be some noise/outliers in our data, as evidenced by the histograms of some the features, shown in Figure 4. However, because of the limited amount of data in this dataset, each patient's data information is very precious, so we avoid removing those outliers.



(a) ALB (b) ALP (c) ALT (d) BIL

Figure 4. Histograms of the values of certain features of the dataset.

2.2 Preprocessing

As mentioned in Subsection 2.1, our dataset was found to be exceptionally clean, devoid of missing values or duplicate rows, and outliers were not removed. Hence, minimal preprocessing was required. The preprocessing steps involved feature scaling and feature selection, which were treated as hyperparameters and optimized simultaneously with the estimator.

2.3 Pipeline

We are essentially treating the preprocessing steps as hyperparameters and try to optimize those too when peforming Cross Validation. We provide two options for the Feature Scaling:

1. Normalization: Using the MinMaxScaler function of scikit-learn.
2. Standardization: Using the StandardScaler function of scikit-learn.

For Feature Selection, we provide three options:

1. Univariate Feature Selection: Using the SelectKBest function of scikit-learn, which removes all but the k highest scoring features. We only give the option of using two different scoring functions:
   (a) *f_classif*, which computes the ANOVA F-value.
   (b) *mutual_info_classif*, which estimates the mutual information.
2. Recursive Feature Elimination: Using the RFE function of scikit-learn, with a Logistic Regression estimator.
3. Tree-Based Feature Selection: Using the SelectFromModel function of scikit-learn, with a Random Forests estimator.

We experiment with two different pipelines, one that includes a Feature Selection step (3.2) and one that does not (3.1). So, the pipelines essentially contain the following steps:

- Feature Scaler
- Feature Selector (optional)
- Estimator

The estimator is a machine learning model like Logistic Regression or Linear Discriminant Analysis fit on the data. By combining the preprocessing steps and the estimator into a single pipeline we achieve two things:

- We can treat the preprocessing steps, Feature Scaling and Feature Selection, as hyperparameters too, so that, when we optimize the model's hyperparameters, we can also find the best Feature Scaling and Feature Selection methods at the same time.
- We avoid Data Leakage, which is a phenomenon that occurs when models are trained with information outside of the training data, i.e. validation and test data (Kaufman *et al.* 2012). Unfortunately, it is easy to cause data leakage when performing k-fold cross-validation. The k-fold cross-validation only works when the models are trained only with data they should have access to. This rule can be violated if the data is processed improperly prior to the sampling. Transformations like standardization use the entire data distribution when determining how each value should be altered. Performing such techniques *before* the training data is split into k folds will mean that the training set will be influenced by the validation set, thereby causing data leakage. This is essentially what happens when carrying out hyperparameter tuning methods that incorporate a cross-validation splitting strategy, such as the grid search or the random search. There is a simple solution to avoiding data leakage when performing k-fold cross-validation, which is to perform such transformations after the training data is split into k-folds. We can accomplish this using scikit-learn's *Pipeline* to create objects that chain together every step of the workflow.

2.4 Performance Evaluation and Optimization

The goal of this study was to determine the best performing classification algorithms for the given data set. We have used nested Cross-Validation to evaluate the performances of the different algorithms, but also to optimize the hyperparameters of the models (and the preparation steps in our case) at the same time. As mentioned, we produced results for two different situations, one in which we don't do feature selection and one in which we do. We use 10 folds for the outer loop (evaluation of optimized model) and 3 for the inner loop (hyperparameter optimization). We repeat this nested Cross-Validation 5 times, one for each algorithm. The hyperparameter optimization is

performed with 100 trials of the OptunaSearchCV function from the Optuna package.

Each outer fold splits the data into a Train and a Test set. For each such outer fold, the inner fold breaks the Train set further into a Train and a Validation set. Each pipeline (preparation steps and estimation) gets trained on the inner Train set and evaluated on the Validation set. Once we find the optimal hyperparameters for the pipeline, we re-train the model on the inner Train and Validation sets combined. We then evaluate the optimized model on the Test set of that particular fold by computing different metrics like Matthews Correlation Coefficient, F1 Score, F2 Score, Precision and more. We repeat this same procedure on each of the 10 outer folds to obtain an unbiased evaluation of the performance of each algorithm, not subject to the initial Train-Test split, since we now split, train and evaluate multiple times (Cawley and Talbot 2010).

The inner folds use the f1_score (macro) for the evaluation when doing the hyperparameter optimization. In an imbalanced dataset, F1 micro can be biased towards the majority class, as it gives equal weight to each instance, regardless of the class distribution. F1 macro, on the other hand, gives equal weight to each class, which can help to prevent the bias towards the majority class and give a better overall picture of the model's performance.

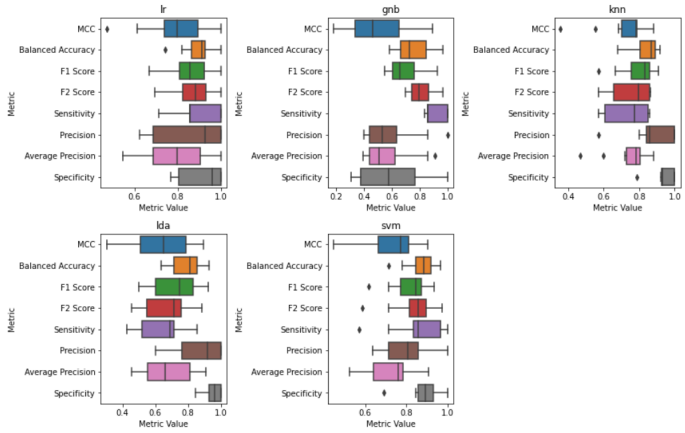
For the outer folds, we compute various metrics for all outer folds' test sets and to select the best performing algorithm between LR, GNB, KNN, LDA and SVM, we compute the mean of the Matthews Correlation Coefficient (MCC) score over all these test sets. Various studies consider MCC to be a very reliable metric for binary classification on imbalanced datasets (Chicco and Jurman 2020; Chicco, Warrens, and Jurman 2021; Chicco, Tötsch, and Jurman 2021; Chicco and Jurman 2023), especially in Bioinformatics, although some others seem to contradict that (Zhu 2020).

3 Results and Discussion

For each outer fold, we compute the evaluation metrics shown in subfigures (a) and produce each metrics' boxplot. We repeat this for every algorithm. At the end, we compute the mean MCC score over all folds for each algorithm, as shown in subfigures (b).

3.1 Main Results

In Figure 5 we see the results of the optimization of the pipeline *without* feature selection. Logistic



(a) Algorithm Performance Metrics over the Outer Folds' Test Sets

	Algorithm	Mean MCC	with Std
0	lr	0.796602	0.157224
1	gnb	0.513823	0.233441
2	knn	0.715927	0.144821
3	lda	0.647799	0.187278
4	svm	0.727265	0.136843

(b) Summarized Algorithm Performance Comparison of MCC Scores

Figure 5. Performance metrics over the different folds' test sets (Without Feature Selection).

Regression achieved the highest average MCC score, 0.796602, over the 10 outer folds of nested Cross Validation. The lowest score was achieved by Gaussian Naive Bayes at 0.513823.

As LR performed better by a wide margin, we chose this one as the 'winner' algorithm. We (re-)trained a pipeline (without feature selection) on the whole dataset, including training and test,

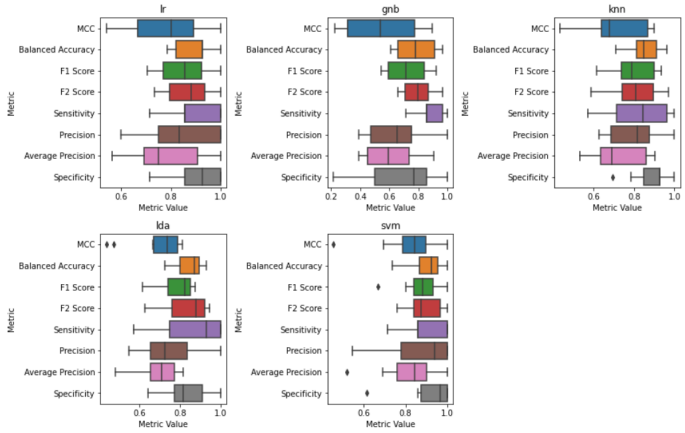
using LR as the estimator. We evaluated and optimized its hyperparameters using simple Cross Validation and the Optuna packages with 100 trials. The selected (optimized) hyperparameters for the preparation and estimation steps were:

- Data Preprocessing:
  - Feature Scaling: Standardization
- LR Estimator Hyperparameters:
  - Penalty: L2
  - C: 1.846
  - Fit_Intercept: True

The mean cross-validation score of the best estimator was: 0.9320

3.2 Feature Selection Results

In Figure 6 we see the results of the optimization of the pipeline *with* feature selection. In contrast



(a) Algorithm Performance Metrics over the Outer Folds' Test Sets

	Algorithm	Mean MCC	with Std
0	lr	0.787379	0.152932
1	gnb	0.550173	0.254412
2	knn	0.714694	0.147188
3	lda	0.691727	0.127964
4	svm	0.819576	0.152865

(b) Summarized Algorithm Performance Comparison of MCC Scores

Figure 6. Performance metrics over the different folds' test sets (With Feature Selection).

to the main results, adding a Feature Selection step in the pipeline improved the performance of all algorithms, except for Logistic Regression. In fact, the SVM classifier now achieved the highest mean MCC score.

As SVM outperformed LR in this case, we chose SVM as the 'winner' algorithm. We (re-)trained a pipeline (with feature selection) on the whole dataset, including training and test, using SVM as the estimator. We evaluated and optimized its hyperparameters using simple Cross Validation and the Optuna packages with 100 trials. The selected (optimized) hyperparameters for the preparation and estimation steps were:

- Data Preprocessing:
  - Feature Scaling: Standardization
  - Feature Selection: Tree-Based
- LR Estimator Hyperparameters:
  - C: 32.737
  - Gamma: 0.024

The mean cross-validation score of the best estimator was: 0.9553.
The top 5 feature selected from the Tree-Based feature selection method were ALP, ALT, AST, BIL and GGT, which aligns with out initial analysis 2.1 that these were some of the most correlated features with the label variable.

4 Conclusion

In this study, the performance of several classification algorithms was evaluated using nested cross-validation and the Matthews correlation coefficient (MCC) as the evaluation metric. Logistic Regression (LR) achieved the highest average MCC score, while Gaussian Naive Bayes had the lowest score. However, when a feature selection step was added to the pipeline, the Support Vector Machine (SVM) classifier outperformed LR. The optimized hyperparameters for the preparation and estimation steps of the LR pipeline were determined using cross-validation and Optuna packages, with a mean cross-validation score of 0.9320. For the SVM pipeline, the hyperparameters were also optimized using cross-validation and Optuna packages, with a mean cross-validation score of 0.9553. The top 5 selected features were ALP, ALT, AST, BIL, and GGT, which were found to be the most correlated features with the label variable. Overall, these results suggest that SVM with feature selection is the best algorithm for this classification task.

5 Supplementary Material

5.1 Notes

Logistic Regression was the most promising of the models when we didn't use feature selection. However, for Logistic Regression we've used L2 regularization, which in some sense can be considered as a feature selection method too. So for Logistic Regression we've probably used some form of feature selection in both pipelines.

5.2 Reproduction

Along with this study, we have also provided a jupyter notebook that contains the associated python code for reproducing the aforementioned results.

The notebook can be run on a cloud collaborative notebook environment like Kaggle (tried and tested) or Google Colab with all dependencies included. The only libraries that needed to be downloaded in our case on these environments were *Optuna* for hyperparameter optimization and *pandas-profiling* for exploratory data analyis. We've included a *!pip install* command for both packages in the first cell of the notebook.

5.3 Evaluate on Hold-Out Set

As a result, we have produced two best pipelines after all this training. One without feature selection (with a Logistic Regression estimator) and one with feature selection (with a Support Vector Machines estimator). These pipelines are optimized in terms of both the preparation steps and the estimator step. Passing the feature values (X) of the hold-out set into their .predict() methods will not only predict on the hold-out set but also transform the data, as it is a pipeline that does both. The last section of the jupyter notebook we've provided uses these two pipelines to predict on the hold-out set and then visualizes the results. All you have to do is provide the correct path to the hold-out set in the second cell of the last section. We've provided both pipelines, but commented out the pipeline without the feature selection because it achieved a lower mean cross-validation score than the one with feature selection. You can use it by uncommenting the corresponding line in the cell. In case you don't want to run the whole notebook to reproduce the two best pipelines from scratch, we have also included the .pkl of these two pipelines along with the code. You can load them by uncommenting the first cell of the last section and changing the paths to the .pkl files to your local paths instead.

References

Cawley, G. C., and N. L. C. Talbot. 2010. “On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation.” *Journal of Machine Learning Research* 11 (70): 2079–2107. http://jmlr.org/papers/v11/cawley10a.html.

Chicco, D., and G. Jurman. 2020. “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation.” *BMC Genomics* 21 (1): 6. https://doi.org/10.1186/s12864-019-6413-7.

———. 2023. “The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification.” *BioData Mining* 16 (1): 4. https://doi.org/10.1186/s13040-023-00322-4.

Chicco, D., N. Tötsch, and G. Jurman. 2021. “The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation.” *BioData mining* 14 (1): 13. https://doi.org/10.1186/s13040-021-00244-z.

Chicco, D., M. J. Warrens, and G. Jurman. 2021. “The Matthews Correlation Coefficient (MCC) is More Informative Than Cohen’s Kappa and Brier Score in Binary Classification Assessment.” *IEEE Access* 9:78368–78381. https://doi.org/10.1109/ACCESS.2021.3084050.

Dodge, Y. 2008. “Exploratory Data Analysis.” In *The Concise Encyclopedia of Statistics.* New York, NY: Springer. https://doi.org/10.1007/978-0-387-32833-1_136.

Kaufman, S., S. Rosset, C. Perlich, and O. Stitelman. 2012. “Leakage in Data Mining: Formulation, Detection, and Avoidance.” (New York, NY, USA) 6 (4). ISSN: 1556-4681. https://doi.org/10.1145/2382577.2382579.

World Health Organization. 2023. *Hepatitis C [Fact Sheet].* https://www.who.int/news-room/fact-sheets/detail/hepatitis-c.

Zhu, Q. 2020. “On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset.” *Pattern Recognition Letters* 136:71–80. ISSN: 0167-8655. https://doi.org/https://doi.org/10.1016/j.patrec.2020.03.030.