CISC-5800-L01 MACHINE LEARNING FINAL PROJECT REPORT

SPECTF Heart Data Set Classification

Presented by:

Donovan Miner

Carlos Ferrer

1. INTRODUCTION

The aim of this project is to enhance the accurate diagnosis of heart disease, ensuring that patients in need receive timely and effective treatment, improving their outcomes. To achieve this, we focus on classifying cardiac Single Photon Emission Computed Tomography (SPECT) images as either normal or abnormal.

This report explores the application of Support Vector Machines (SVM) with a newly introduced kernel function on a dataset comprising over 267 instances sourced from the UCI Machine Learning Repository. We implemented SVM from scratch and extensively experimented with hyperparameters, including learning rate, slack variable, and kernel type, to optimize the model's performance. Additionally, a Bayes classifier was used for comparison. Model performance was evaluated using accuracy, with preliminary results showing promising outcomes.

2. DATASET

Description

The dataset for this project was sourced from the UCI MAchine Learning Repository and focuses on diagnosing cardiac Single Photon Emission Computed Tomography (SPECT) images. It includes data from 267 patients, each represented by 44 features. Each patient is classified into one of two categories, normal or abnormal.

The dataset is divided into a training set, SPECTF.train, containing 80 instances, and a testing set, SPECFT.test, containing 187 instances. The training set is balanced, with an equal distribution of 40 normal and 40 abnormal instances, which is ideal for training machine learning models. On the other hand, the testing set is imbalanced, it has 15 normal and 172 abnormal instances.

Data Preparation

An advantage of this dataset is that it has no missing values, reducing the need of additional preprocessing. The dataset was preprocessed using min/max normalization to scale all features to a range between 0 and 1. This normalization ensures that individual attributes are on the same scale, which is particularly important for algorithms like Support Vector Machines to perform effectively.

$$\boldsymbol{x_{i,f}} = (x_{i,f} - \min(x_f)) \,/\, (\max(x_f) - \min(x_f))$$

3. Learning Methods

Support Vector Machines

Support Vector Machines (SVMs) are a supervised machine learning algorithm widely used for classification and regression tasks. The main idea of SVM is to find the optimal hyperplane that separates data points belonging to different classes with the maximum margin in the feature space. SVM is particularly effective for high-dimensional data and is well-suited for scenarios with a limited number of training samples. In addition to linear classification, SVMs can effectively handle non-linear classification using the kernel trick, which implicitly maps input data into high-dimensional feature spaces. We define a linear classifier, h(x)=sign(wTx+b), using binary classification labels {+1,−1}.

$$f(\chi_i) = \begin{cases} 1, & \text{if } \omega \cdot \chi + b \geq 1 \\ -1, & \text{if } \omega \cdot \chi + b \leq -1 \end{cases}$$

In learning the parameters of the SVM, the dual form of the optimization problem was implemented. By using the Representer Theorem, the separating hyper-plane can be learned by first finding the α's for each data point. The resulting non-zero α values resulting from the learning process are the learned support vectors, which can be used directly to classify new data points, or to learn the primal form of the SVM. In this process, the α values were learned, then used to calculate the primal *w* and *b* offset in order to classify the test data.

Representer: $w = \sum_i^n a^i y^i x^i$ Dual Classifier: $sign(\sum_i^n a^i y^i k\langle x^i, x\rangle + b)$

In experimenting with the SVM, ultimately 4 factors were taken into account: kernel functions, slack variables, step size in gradient descent, and number of iterations in gradient descent. The values tested are shown below.

Kernel Function	Linear	Radial Basis Function	Quadratic Function	—	—
Slack Constraint	unbound	1	5	10	100
Step Size	0.01	0.05	0.1	0.5	1
Iterations	10	35	50	100	—

The α values were learned via minimizing error through gradient descent. Hinge loss modified for the dual SVM and the gradient update rule are shown below in their matrix representations, where $\boldsymbol{\alpha}$ represents the vector of alpha values and $\boldsymbol{K}$ represents the Gram matrix, where $K_{i,j} = k\langle \mathbf{x}^n, \mathbf{x}^m \rangle$ (Bishop, 2006, 291-293).

$$L = 1/2\,(\alpha K\alpha) - \sum_i^n \alpha$$

$$\Delta\alpha = \epsilon\,(1 - \alpha K\alpha) \text{ subject to } 0 \leq \alpha \leq C$$

Initial tests of the SVM and analysis of the dataset revealed that many features of the two classes were located relatively closely together, meaning that a linear separator would likely be unable to separate the two classes efficiently. As a result, a quadratic kernel function was tested for its effect on separation. Due to the significant overlap between the classes, slack variables were also introduced to allow for a certain degree of misclassification in the hopes that an overall better separator would be learned. An unbound classifier was also tested.

The self-built SVM was constructed almost entirely with NumPy, largely for ease of matrix operations. Pandas was used to load the dataset, as well as for some transformations and exporting result metrics. The remaining miscellaneous packages were the 'inf' maximum integer placeholder and the *exp* function from the Python 'math' module. For more detailed information on the program and functions, please refer to the 'README' document.

Bayes Classifier

The Bayes classifier is a probabilistic algorithm used for solving classification problems. It's based on Bayes's theorem, which calculates the posterior probability of a hypothesis given prior knowledge and evidence. The algorithm assumes that the features in the dataset are conditionality independent of each other, a naive assumption that simplifies computation and makes the method highly efficient.

4. Results

Support Vector Machine (without Libraries)

Listed in the table below are some metrics taken from various iterations of the tested SVM. Note that the linear and quadratic kernels list their first four unconstrained iterations, as these are fairly representative of their overall performance. The radial basis kernel function saw the most variation in its performance, so differing metrics were chosen to represent it accordingly. The full list of metrics can be found in ‘metrics.xlsx’.

Most notably is the model's inability to generalize or properly separate the data. The model tends to overfit on one class or the other, either properly classifying all or nearly all of the negative classes properly, while also returning a significant number of false negatives, or correctly identifying a majority of the positive class and none of the negative class, while still returning a significant number of false positives. One reason for this could be the difference in the training and testing datasets. While the training data is balanced with 40 instances of each class, the testing data only has 15 instances of the negative class. It could be possible that the data is overlearning the negative data and improperly classifying as a result. The more likely explanation is that the two classes are simply not being separated enough by the chosen kernel methods, and the classifier is unable to properly distinguish the two. The features themselves lie fairly close to each other in their absolute terms, and min-max normalization was used to try to address this, but was not enough. More would need to be learned about the relationships between features and to infer the best method to separate the two classes. RBF and quadratic were the highest order functions used in these tests, so it could be the case that the data is separable in higher dimensions. The final issue that could arise is simply erroneous implementation affecting the trained model’s support vectors.

Kernel	Slack	Iterations	Step Size	TPos	TNeg	FPos	FNeg	Run Time
Linear Kernel	inf	10	0.01	13	15	0	159	0.64011
Linear Kernel	inf	35	0.01	104	0	15	68	2.260109
Linear Kernel	inf	50	0.01	13	15	0	159	3.264147
Linear Kernel	inf	100	0.01	13	15	0	159	6.487422
Quadratic Kernel	inf	10	0.01	13	15	0	159	0.726702
Quadratic Kernel	inf	35	0.01	104	0	15	68	2.480295
Quadratic Kernel	inf	50	0.01	13	15	0	159	3.700943

Quadratic Kernel	inf	100	0.01	13	15	0	159	7.124124
RBF Kernel	inf	100	0.01	105	1	14	67	8.235525
RBF Kernel	inf	10	0.5	14	15	0	158	0.811922
RBF Kernel	5	10	0.01	104	2	13	68	0.827091
RBF Kernel	10	100	1	13	15	0	159	8.174072

Support Vector Machine (using Libraries)

In this project, we also used the scikit-learn library to implement the Support Vector Machine (SVM) classifier. We focused on tuning key hyperparameters: the kernel type, and penalty coefficient (C). When experimenting with specific parameters, we retained the default values for other parameters not being tested to ensure controlled comparisons. The hyperparameters were tested across the following ranges, and the resulting metrics of the SKLearn implementation are shown below for comparison.

Epsilon	0.001	0.01	0.1	1	10
Penalty Coefficient (C)	0.1	1	10	100	1000
Kernel Type	Linear	RBF	Polynomial		

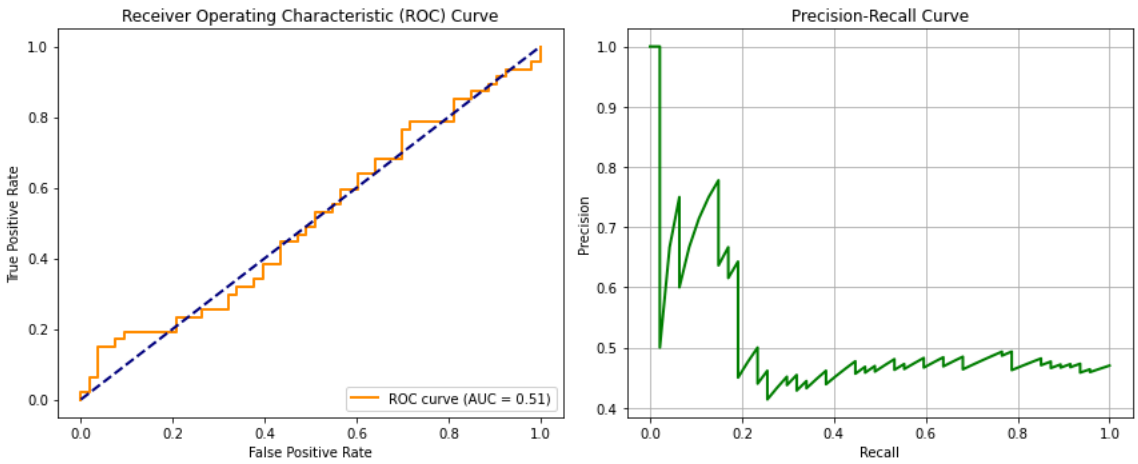
Kernel	C	Accuracy	Runtime
Linear	0.1	0.679144	0.003986
Linear	1	0.71123	0.00598
Linear	10	0.716578	0.00598
Linear	100	0.716578	0.00598
Linear	1000	0.716578	0.005924
Polynomial (Quadratic)	0.1	0.213904	0.001992
Polynomial (Quadratic)	1	0.336898	0.00299
Polynomial (Quadratic)	10	0.427807	0.002992
Polynomial (Quadratic)	100	0.411765	0.002992
Polynomial (Quadratic)	1000	0.411765	0.002994
Rbf	0.1	0.716578	0.006916
Rbf	1	0.743316	0.004531
Rbf	10	0.754011	0.00299
Rbf	100	0.754011	0.003987
Rbf	1000	0.754011	0.003986

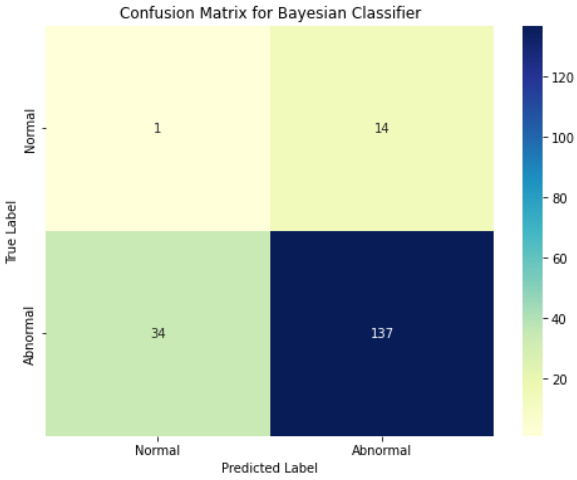
[Best accuracy/runtime highlighted for each parameters]

Bayesian Classifier

The Bayesian classifier was implemented as an additional method for comparative analysis in this project. The simplicity and probabilistic approach of the Naive Bayes classifier make it an effective baseline for classification problems, particularly when dealing with small datasets like the SPECT dataset used here.

The Naive Bayes classifier achieved an accuracy of 74%, higher than the SVM's 75%. While the Bayesian model showed promising performance, it struggled with class imbalances present in the test data, as reflected in its precision and recall scores. The ROC curve highlights a trade-off between sensitivity and specificity, with an area under the curve (AUC) of 0.51, compared to the SVM's).





5. Conclusion

In this project, we implemented and evaluated two classification algorithms Naive Bayes and Support Vector Machine (SVM) to classify SPECT images as normal or abnormal. Our analysis revealed that the SVM classifier outperformed the Naive Bayes classifier in terms of accuracy and robustness. This aligns with existing literature, which suggests that SVMs, with their ability to find an optimal hyperplane for data separation, often achieve higher accuracy in complex datasets. In contrast, the Naive Bayes classifier, which operates under the assumption of feature independence, may not capture intricate relationships between features, leading to comparatively lower performance.

These findings suggest that SVMs are a more suitable choice for this classification task, providing valuable insights for future research and practical applications in medical image analysis.

6. References:

1. Cios, K., Kurgan, L., & Goodenday, L. (2001). SPECTF Heart [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5N015.
2. Anshul. (2021, October 12). Guide on Support Vector Machine (SVM) Algorithm. Analytics Vidhya.https://analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
3. Wilimitis, D. (2019, February 21). The Kernel Trick. Medium. https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f
4. Lecture 9: SVM. (n.d.). Www.cs.cornell.edu. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html
5. Naive Bayes Classifier in Machine Learning. (2023, September 9). Www.guru99.com. https://www.guru99.com/naive-bayes-classifiers.html
6. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).
7. Scikit-learn developers. (n.d.). Support vector machines. Scikit-learn. Retrieved December 10, 2024, from https://scikit-learn.org/stable/modules/svm.html
8. Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.
9. J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.
10. Data structures for statistical computing in python, McKinney, Proceedings of the 9th Python in Science Conference, Volume 445, 2010.
11. Bishop, C. (2006). *Pattern Recognition and Machine Learning* (pp. 291 - 293). Springer.