Support Vector Machines:
Utilizing Polynomial Kernelization for Iris Species Classification

ABSTRACT

The Iris plant is composed of different species and is easy to tell apart visually, but these species are not easily differentiated based on specific measurements. Present in our dataset, we have two separate Iris species, the Iris-Setosa and Not-Iris-Setosa. Along with these classifications, the dataset also includes each plant's sepal length, sepal width, petal length, and petal width. It is pertinent for the accurate classification of these plants, as each species flourishes in slightly different soil conditions, and different water and sunlight requirements. This report aims to use the given features and known classifications of the dataset to build, train, and fit a support vector machine (SVM) to pick up the latent information in our dataset, and thus, correctly classify our data points. Through cross-validation techniques and hyper-parameter tuning, it was found that using an SVM with polynomial kernelization offers the most accurate and precise classification for our dataset.

INTRODUCTION

Support vector machines are a supervised and discriminative machine learning classification algorithm that uses decision boundaries to segment and classify our entries, depending on the features we are fitting the model with. These models offer a variety of use cases, as there are multiple hyper-parameters that alter the way the SVM classifies our data. The background section contains further information regarding support vector machines, how they work, and why their use is optimal in our case. The discussion section outlines the outcomes of my implemented solution, its relation to the species of the Iris, and whether or not our model's performance is acceptable. The conclusion reiterates the pertinence of proper classification for the Iris' survival.

BACKGROUND

As is the case with all machine learning algorithms, the objective of our SVM algorithm is to 'learn' latent information present in our dataset. In the case of SVMs, the outcome of the model is a decision boundary, typically a line or hyper-plane, that aims to perfectly separate the different classes in the dataset. SVMs operate under three key initiatives:

- The use of optimization to find a solution (hyper-plane or line) with few errors
- Seeks a large margin separator (maximize the distance between decision boundary and respective data points)
- Uses the kernel trick to make large feature spaces more computationally efficient

These three key ideas are the main guidelines that SVM uses to output our decision boundary. The first idea can more generally be defined as allowing for slack. Very rarely, in real-world circumstances, will we have a dataset that is perfectly separable by a single decision boundary, and thus, this allowance addresses the issue of infinite oscillation of our boundary. The next key idea, maximizing margin separation, encourages and requires a more robust and generalizable output of our model. Finally, kernelization allows the algorithm to classify datasets with larger feature spaces by mapping the data to a higher dimension. This last initiative was critical in solving the problem, as our data is comprised of four features and is not easily classified by a single line.

Additionally present in SVMs are hyper-parameters that further alter the way these classifications are made. One of these hyper-parameters is the kernelization method used. In our case, we opted to select a polynomial kernelization method to map our dataset to a higher feature space, which sacrifices interpretability for the sake of improving the accuracy and precision of our classification. For any general SVM implementation, the optimal kernelization method is dependent on the nature of the dataset and desired model outcomes. The final hyper-parameter present in polynomial kernelized SVMs is the 'C' parameter. This hyper-parameter adds a penalty associated with each erroneous classification and is what allows our algorithm to have slack in our classification.

DISCUSSION

With a deeper understanding of how SVMs make classifications and how the hyper-parameters influence the model's classification, we can now delve into my implementation and the intricacies behind it. As aforementioned, hyper-parameters vastly differentiate how a particular algorithm performs. From the kernelization method, different values of C, and the degree of the polynomial decision boundary, each permutation of these values will greatly modify the model's results. In order to test each permutation of these hyper-parameters and identify the optimal solution, a grid search was conducted. This grid search cross-validated the kernelization method, C-value, and degree of the polynomial, and outputted the optimal combination of these hyper-parameters. At the end of all cross-validation and hyper-parameter tuning, it was found that the optimal SVM model for our dataset used a polynomial kernel, a C value equal to 0.1, and a polynomial degree equal to three.

Going further into the implementation decisions made, I used a 70/30 train-test split. Using this split and the parameters mentioned above, the model had perfect predictive power.

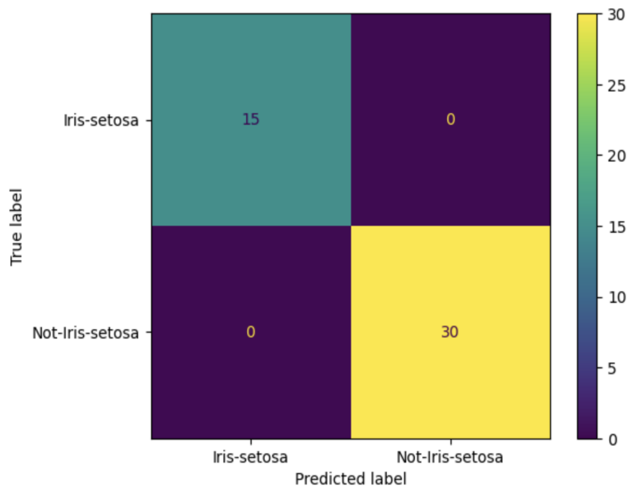


Figure 1

Figure 1 confirms and visualizes that our model's predictions on unseen data are perfect. There are 15 total true positives and 30 total true negatives, meaning that every piece of testing data was correctly classified.

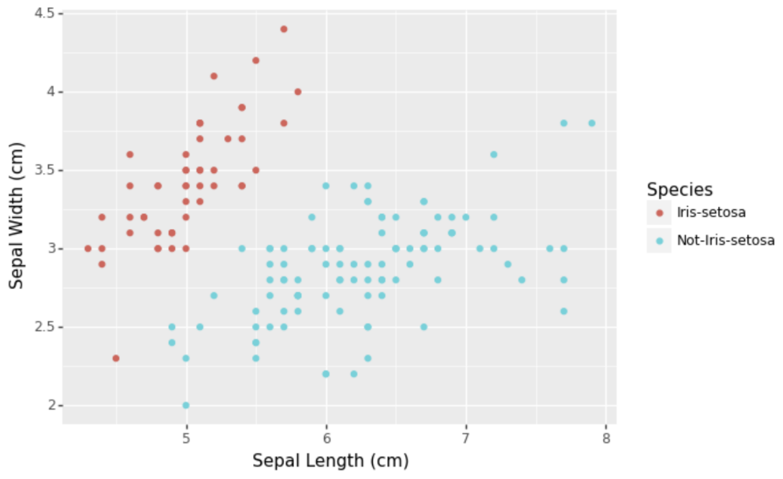


Figure 2

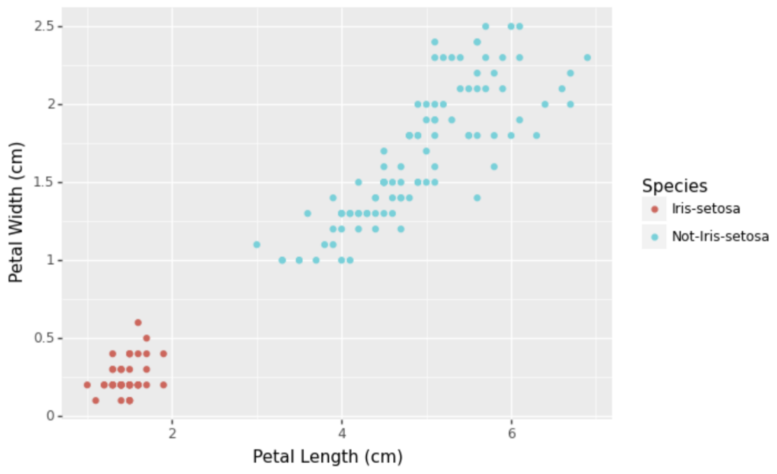
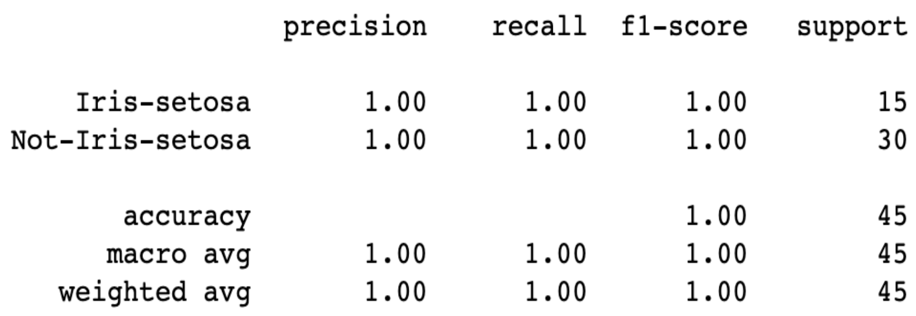


Figure 3

Although there are no visual decision boundaries (as I was unable to implement a plot that did), it becomes apparent from both Figures 2 and 3, that our classes of data are both separate and cohesive. Due to this, it is highly reasonable that our more complex SVM is able to perfectly classify our data, and thus, we can confirm that our model is not overfit.



```
                  precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        15
 Not-Iris-setosa       1.00      1.00      1.00        30

        accuracy                           1.00        45
       macro avg       1.00      1.00      1.00        45
    weighted avg       1.00      1.00      1.00        45
```

Figure 4

Along the same narrative of the first three figures, figure 4 is the final figure that denotes the complete perfection of our model. The combination of precision, recall, and f1 give a further understanding of our model's predictive power and accuracy. All of these metrics' maximum values are one, and our model was able to match these perfect metric scores.

CONCLUSION

From this dataset, and the model I abstracted and implemented, it is apparent that the Iris species is easily classifiable. Using the correct permutation of hyper-parameters SVMs prove to be more than proficient at classification for our given dataset. Both the confusion matrix outputted, and the performance metrics above, it's very apparent that my model is acceptable and should be put into practice.

REFERENCES

Kumar, A. M. (2018, December 17). C and Gamma in SVM. Medium. https://medium.com/@myselfaman12345/c-and-gamma-in-svm-e6cee48626be

Plot different SVM classifiers in the iris dataset — scikit-learn 0.18.2 documentation. (n.d.). Scikit-Learn.org. https://scikit-learn.org/0.18/auto_examples/svm/plot_iris.html

Scikit-learn developers. (2019). sklearn.svm.SVC — scikit-learn 0.22 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Support Vector Machines (SVM). (n.d.). Retrieved February 22, 2023, from https://www.datasciencesmachinelearning.com/2019/01/support-vector-machines-svm.html

SVM Hyperparameter Tuning using GridSearchCV. (2020, March 10). Velocity Business Solutions Limited. https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/

The Iris Dataset. (n.d.). Scikit-Learn. https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html