Homework 2: Support Vector Machines

Anthony Andriano

University of Colorado Colorado Springs
aandrian@uccs.edu

Planning Relax Dataset —
https://archive.ics.uci.edu/dataset/230/planning+relax
Skin Segmentation Dataset —
https://archive.ics.uci.edu/dataset/229/skin+segmentation

Introduction

Support Vector Machines (SVMs) are supervised learning models that can be used for binary classification, multiclass classification, and regression. The fundamental principle of an SVM is to find a decision boundary between two or more classes, such that the width of the decision boundary is as wide as possible. The decision boundary is sometimes referred to as a hyperplane, which separates classes in as many dimensions as the dataset requires based on the number of features. It is possible to separate the features using non-linear methods using kernel functions with non-linear constructions. However, kernel functions and non-linear classification are beyond the scope of this paper.

A key term during the discussion and use of an SVM is support vector. Support vectors are the points that lie on or near the decision boundaries. Ideally, SVMs have linearly-separable data points as support vectors, but that is not always the case, which will be discussed in later sections.

Question 1: Linear Separability

If a linear function, which is called a hyperplane in higher dimensions, exists such that the examples of a dataset can be separated by the linear function with any errors, the examples are said to be linearly separable. In a classification problem, that means all data points from one class are on a different side of the linear function than the data points from the other class. The linear function is a line when the feature space has a cardinality of 2, and the linear function is a plane or hyperplane when the cardinality of the feature space is 3 or more, respectively.

To determine if the data points are in one class or another, a discriminator function is used. A linear discriminator is a linear function, which is used to predict the class of a label according to the following definitions:

$$h_{\overrightarrow{\theta}}(x) = \theta_0 + \theta_1 x \tag{1}$$

The bias term, θ_0, is typically included in the parameter vector, which means the equation becomes:

$$h_{\overrightarrow{\theta}}(x) = \overrightarrow{\theta} \cdot \overrightarrow{x} \tag{2}$$

where:

- $x \in R^n$: the feature vector (input example)
- $\theta \in R^n$: the parameter or weight vector
- $y \in \{0, 1\}$: the binary class label

For a dataset to be linearly separable, there must exist some $\overrightarrow{\theta}$ such that:

- $\overrightarrow{\theta} \cdot \overrightarrow{x} \geq 1$ for data points in class 1
- $\overrightarrow{\theta} \cdot \overrightarrow{x} \leq -1$ for data points in class 0

Question 2: Margin of Separation

When the classes are linearly separable, a margin of separation is said to exist between the data points in each class. The boundaries defined by the discriminator are called the upper and lower margins. The goal is to find the widest distance between the upper and lower margins to maximize the classification and prediction capabilities of the SVM. Variations in the parameter vector are minimized when the margin is the widest it can be. The data points from each class on or closest to the margins are called support vectors.

The linear discriminator in a binary classification problem is sometimes called a binary discriminator. Equation 2 defines the decision boundary when the value is 0, or $\overrightarrow{w} \cdot \overrightarrow{x} = 0$. The upper boundary is defined as $\overrightarrow{\theta} \cdot \overrightarrow{x} = 0$. The upper boundary is set to $\overrightarrow{\theta} \cdot \overrightarrow{x} \geq 1$ while the lower boundary is set to $\overrightarrow{\theta} \cdot \overrightarrow{x} \leq -1$.

The discriminator and both margins form the margin of separation, which determines the classification of the data points that are evaluated using the SVM.

Width of the Margin

The width of the margin is the perpendicular distance between the upper and lower margins. The discriminator is halfway between each margin. To maximize the width of the margin, the minimum of $\|\theta\|$ must be calculated. For mathematical convenience, it is often better to minimize $\|\theta^2\|$ instead.

Question 3: Regularized Loss

It is ideal to have linearly separable data, but that is not always possible. To accurately classify data that is not perfectly linearly separable, an objective function with a loss component must be used.

The loss function must be simple enough to implement with a reasonable amount of resources and run time, while also fitting well to the data. In class, we primarily discussed hinge loss, which is a type of loss function that is piecewise differentiable. The non-differentiable point creates problems with the mathematical analysis of the model, but that issue can be worked around by picking a value for the function at the non-differentiable point. It is not as polished as the loss functions that are continuously differentiable, but it is easier to implement and the results are generally considered to be good enough for robust classification.

The first component of the loss function is the regularization component: $\frac{1}{2}\|\theta\|^2$. This component creates the widest possible margin between the classes. The second component of the loss function is the hinge loss equation, which is $\max(0, 1 - y_i(\overrightarrow{\theta} \cdot \overrightarrow{x}_i))$. The hinge loss equation must be summed over the dataset. The point of the hinge loss function is to penalize the output if a value exists within the margin or if the value has been misclassified. To control the effect of both components, a hyperparameter called C is often used. The hyperparamter balances between the width of the margin and the penalties due to misclassifications.

The combined equation is as follow:

$$\frac{1}{2}\|\theta\|^2 + C\sum_{i=1}^{N} \max(0, 1 - y_i(\overrightarrow{\theta} \cdot \overrightarrow{x}_i)) \tag{3}$$

The motivation for the combined loss function is to allow data from imperfect datasets to be correctly classified with as much confidence as possible. Real datasets are often noisy with linearly inseparable classes, but the majority of the data points in each class could still be linearly separable, which would make an SVM with a linear classifier useful.

Question 4: Stochastic Gradient Descent

To solve the SVM classification problem, we are using Stochastic Gradient Descent (SGD). The goal of SGD is to minimize equation 3, which is the equation that combines regularization and loss. To compute the gradient of the equation, partial derivatives must be calculated.

Using equation 2, which is the linear function that is used as the discriminator, determine the gradient in the continuous regions. When $x < 1$, the gradient is -1. When $x > 1$, the gradient is 0. When $x = 1$, the gradient is $[-1, 0]$.

The algorithm for SGD includes a learning rate, which is another hyperparameter that controls the weight the parameter vector. First assign random initial values to the parameter vector, and then repeat the following calculation for each data point in the dataset:

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} - \alpha\nabla\mathcal{L} \tag{4}$$

That process is continued until the output converges, which means the value of consecutive calculations becomes very small, or the number of iterations has exceeded a

Question 5: Datasets

The **Planning Relax** and **Skin Segmentation** datasets were downloaded from the UCI repository for use in this assignment. The datasets contain all numeric features, which makes binary classification straightforward to perform.

Question 6: Custom SVM Model

A custom SVM was written in Python for this assignment. The custom SVM uses the SGD algorithm to optimize the parameter vector by changing the hyperparameter, C, and the learning rate. To measure the performance of the SVM, the classification accuracy was measured as a percentage. The runtime of each iteration was also recorded for information purposes only. Each of the datasets from Question 5 were analyzed with the custom SVM as seen in the following subsections.

Relax Dataset

The accuracy of the custom SVM when using the Relax dataset was not affected by the hyperparameter, C, or the learning rate parameter. The results also remained the same regardless of how many times the program was executed using randomized training and test sets from the same dataset. Each iteration resulted in the same accuracy, which suggests the accuracy of the model is limited by data itself. Notably, the custom SVM produced the same accuracies as the library-based SVMs, which will be detailed in the next sections.

The configurations and results for the Relax dataset are as follows:

Accuracy (%)	Run Time (s)	C	Learning Rate
70.270	0.295	0.0001	0.001
70.270	0.289	0.0001	0.01
70.270	0.288	0.0001	0.1
70.270	0.290	0.5	0.001
70.270	0.305	0.5	0.01
70.270	0.310	0.5	0.1
70.270	0.291	1.0	0.001
70.270	0.308	1.0	0.01
70.270	0.311	1.0	0.1
70.270	0.295	3.0	0.001
70.270	0.309	3.0	0.01
70.270	0.291	3.0	0.1

Skin Dataset

The accuracy of the custom SVM when using the Skin dataset was not affected by the hyperparameter, C, or the learning rate parameter. The results also remained the same regardless of how many times the program was executed using randomized training and test sets from the same dataset. Each iteration resulted in the same accuracy, which suggests the accuracy of the model is limited by data itself. Notably, the custom SVM produced different accuracies as the library-based SVMs, which will be detailed in the next sections. The origin of the significant difference (over 50%) was not able to be determined; it is likely due to the way the calculations are processed and stored in the parameter vector, but no experiment was successful at locating the problematic statement or operation.

The configurations and results for the Skin dataset are as follows:

Accuracy (%)	Run Time (s)	C	Learning Rate
20.475	394.192	0.0001	0.001
20.475	393.437	0.0001	0.01
20.475	435.193	0.0001	0.1
20.475	424.507	0.5	0.001
20.475	423.897	0.5	0.01
20.475	423.798	0.5	0.1
20.475	423.978	1.0	0.001
20.475	424.812	1.0	0.01
20.475	424.845	1.0	0.1
20.475	427.708	3.0	0.001
20.475	424.084	3.0	0.01
20.475	396.809	3.0	0.1

Question 7: Library-Based SVM Models

For this part of the assignment, the following libraries were used:

- SVC from sklearn.svm
- LinearSVC from sklearn.svm
- SGDClassifier from sklearn.linear_model

To measure the performance of the library-based SVMs, the classification accuracy was measured as a percentage. The runtime of each iteration was also recorded for information purposes only. Each of the datasets from Question 5 were analyzed with the library-based SVMs as seen in the following subsections. The same library implementations were also used for question 8.

SVC

Relax Dataset The SVC library was optimized using the hyperparameter, C. As seen in the following table, the classification accuracy matches the results of the custom SVM from question 6 when using the Relax dataset. As an additional experiment, the learning rate parameter was also changed, but the effect was the same - no change in the output. Because changing C and gamma appeared to have no impact on the accuracy, the limitation of the accuracy is likely due to the nature of the data.

Accuracy (%)	Run Time (s)	C
70.27	0.003	0.1
70.27	0.004	0.5
70.27	0.006	1.0

Skin Dataset The SVC library was optimized using the hyperparameter, C. As seen in the following table, the classification accuracy exceeds the results of the custom SVM from question 6 when using the Skin dataset. The difference is substantial, which points to an issue with the custom SVM model. Compared to the other library-based SVM models, the SVC model produced the same accuracy within 1%.

Accuracy (%)	Run Time (s)	C
92.873	103.237	0.1
92.873	128.074	0.5
92.867	139.535	1.0

LinearSVC

Relax Dataset The LinearSVC library was optimized using the hyperparameter, C. As seen in the following table, the classification accuracy matches the results of the custom SVM from question 6 when using the Relax dataset. As an additional experiment, the loss function parameter was also changed, but the effect was the same - no change in the output. Because changing C and the loss function appeared to have no impact on the accuracy, the limitation of the accuracy is likely due to the nature of the data.

Accuracy (%)	Run Time (s)	C
70.270	0.014	0.1
70.270	0.012	0.5
70.270	0.012	1.0

Skin Dataset The SVC library was optimized using the hyperparameter, C. As seen in the following table, the classification accuracy exceeds the results of the custom SVM from question 6 when using the Skin dataset. The difference is substantial, which points to an issue with the custom SVM model. Compared to the other library-based SVM models, the SVC model produced the same accuracy within 1%.

Accuracy (%)	Run Time (s)	C
92.873	0.248	0.5
92.871	0.113	0.1
92.869	0.354	1.0

SGDClassifier

Relax Dataset The SGDClassifier library was optimized using the regularization parameter, alpha. As seen in the following table, the maximum classification accuracy matches the results of the custom SVM from question 6 when using the Relax dataset. When the regularization parameter is weak, however, the results are worse than the custom SVM model as well as the other library-based SVM models. As an additional experiment, the loss function parameter was also changed, but the effect was the same - no change in the output. Because changing the loss function appeared to have no impact on the accuracy, the limitation of the maximum accuracy is likely due to the nature of the data. The accuracy being lower when the regularization parameter is reduced shows a need for regularization to prevent poor fit of the model to the data. Penalizing larger weights prevents overfitting, which seems to be required in this scenario.

Accuracy (%)	Run Time (s)	Alpha
70.270	0.001	0.01
67.568	0.001	0.0001
62.162	0.001	0.001

Skin Dataset The SGDClassifier library was optimized using the regularization parameter, alpha. As seen in the following table, the classification accuracy exceeds the results of the custom SVM from question 6 when using the

Skin dataset. Unlike the results from the Relax dataset, this dataset performs well regardless of the value of the regularization parameter, which suggests a difference in the data and associated weights. As an additional experiment, the loss function parameter was also changed, but there was no effect on the output. Because changing the loss function appeared to have no impact on the accuracy, the limitation of the accuracy is likely due to the nature of the data. With this dataset, overfitting does not appear to be a problem.

Accuracy (%)	Run Time (s)	Alpha
92.892	0.067	0.001
92.853	0.061	0.01
92.555	0.123	0.0001

Summary

Relax Dataset The maximum accuracy of each SVM classification library when analyzing the Relax dataset was the same as the custom SVM: 70.27%. However, library-based implementations completed 80 to 100 times faster due to being implemented in a lower-level language than Python.

Skin Dataset The maximum accuracy of each SVM classification library when analyzing the Skin dataset was significantly higher than the custom SVM: approximately 92% versus approximately 20%. In addition, library-based implementations completed 80 to 100 times faster due to being implemented in a lower-level language than Python.

Question 8: Other Classifiers

The last question for the assignment requires comparisons of the custom SVM, library-based SVMs, and other classifiers.

Relax Dataset

The list of algorithms and the associated results that were compared using the Relax dataset is as follows:

Model	Accuracy (%)	Run Time (s)
Decision Tree	72.973	0.001
AdaBoost	70.27	0.210
CatBoost	70.27	0.113
Custom SVM	70.27	0.310
Gaussian Naive Bayes	70.27	0.001
Gaussian Process	70.27	0.028
Linear SVC	70.27	0.014
MLP Classifier	70.27	0.150
Random Forests	70.27	0.112
SGD	70.27	0.001
SVC	70.27	0.006
XGBoost	70.27	0.042
Quadratic Disc. Analysis	67.568	0.001
K Nearest Neighbors	67.568	0.029
LightGBM	67.568	0.175

Skin Dataset

The list of algorithms and the associated results that were compared using the Relax dataset is as follows:

Model	Accuracy (%)	Run Time (s)
LightGBM	99.951	0.519
CatBoost	99.933	7.202
Decision Tree Classifier	99.931	0.164
XGBoost	99.867	0.301
MLP Classifier	99.743	29.076
XGBoost	99.541	0.292
Quadratic Disc. Analysis	96.493	0.028
SGD	92.892	0.092
Linear SVC	92.873	0.318
Gaussian Naive Bayes	92.483	0.022
AdaBoost	91.902	1.709
Random Forests	90.645	2.666
AdaBoost	89.849	1.737
Custom SVM	20.475	427.708

Analysis

Using the Relax dataset, the custom SVM model performed as well as almost every library-based model. Experimentation with various parameters for each model showed almost no performance increase; there were performance decreases, though, such as the SGD model having a minimum accuracy of 35% when the regularization parameter is too low. The highest accuracy was achieved with the Decision Tree model; I expected XGBoost or one of the other boosted models to achieve higher accuracy. It may be that the parameters were not configured correctly, which led to lower accuracies.

Using the Skin dataset, the custom SVM model performed significantly worse than every other model. I spent hours debugging and tweaking the model to no avail. A maximum accuracy of 20.475% with the custom SVM model compared to a maximum of 99.951% with LightGBM is an inexplicably larger difference. The lowest accuracy of the library-based models was AdaBoost at 89.849%, which is still more than four times higher than the accuracy of the custom SVM model. I am guessing there is a bug in the code that is yet to be identified.

Future Work

The custom SVM model could be improved by implementing an algorithm in C instead of Python, which would dramatically increase the iteration speed, and fixing the objective function to perform better with additional datasets.

References

”Implementing SVM and Kernal SVM with Python’s Scikit-Learn”. 2023. Implementing SVM and Kernal SVM with Python’s Scikit-Learn. Accessed: March 29, 2025.

”Support Vector Machines”. 2025. 1.4 Support Vector Machines. Accessed: March 30, 2025.

”Support Vector Machines with Scikit-learn Tutorial”. 2019. Support Vector Machines with Scikit-learn Tutorial. Accessed: March 30, 2025.

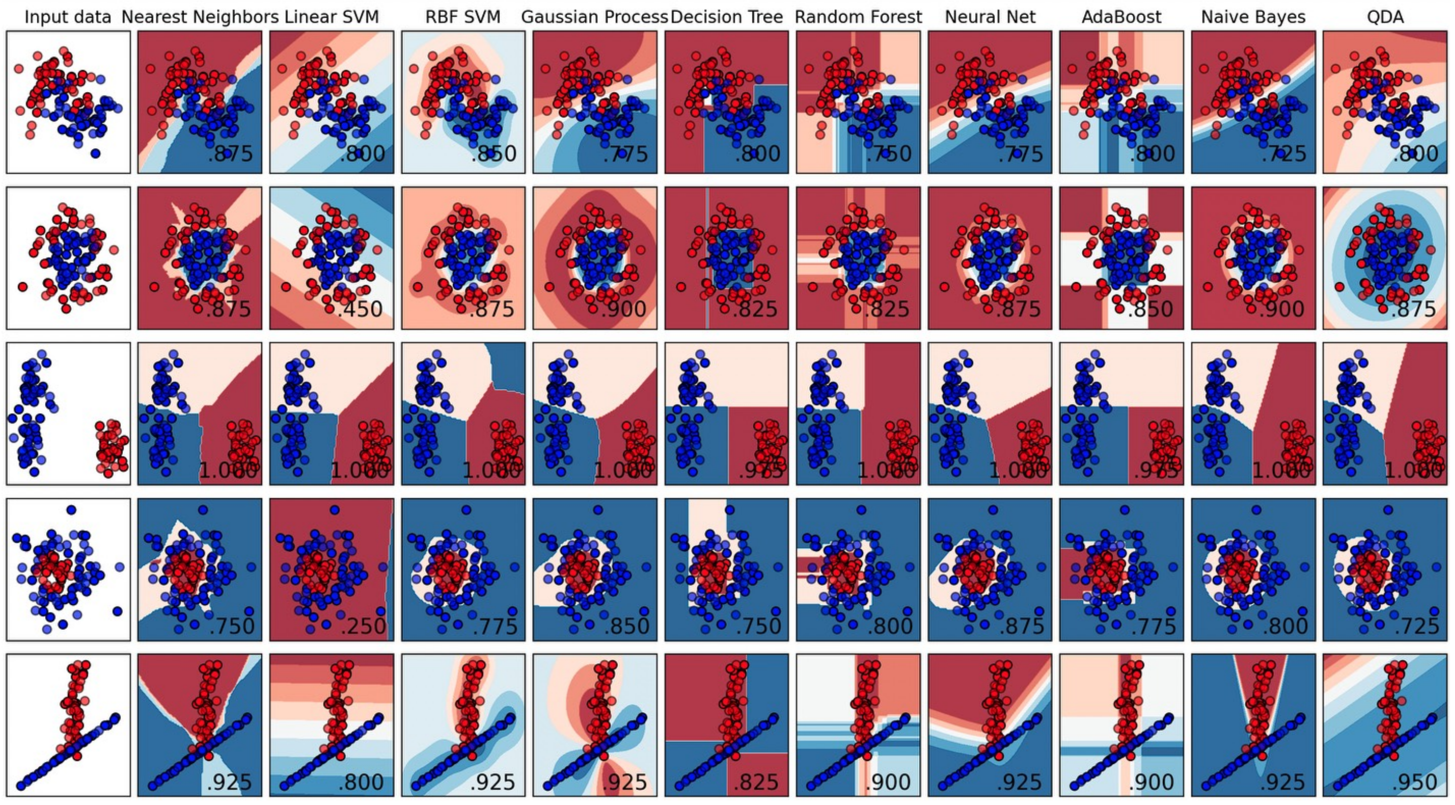
”SVM Python Without Library”. 2019. svm-python-without-library. Accessed: April 5, 2025.

Appendix

To support this effort, significant effort and time were invested into visualizing various datasets and algorithms. Shown below are examples results using the algorithms in the ”Question 8: Other Classifiers” section of the paper. The rows are different configurations of input data. From top to bottom:

- make_moons - Data points that are in the shape of partial moons, which resemble crescents
- make_circles - Data points in circular patterns with varying diameters
- make_blobs - Data points in groups that do not overlap
- make_gaussian_quantiles - Data points in densely nested concentric spheres with increasing radius
- make_classification - Data points that are normally distributed about the vertices of an n-dimensional hypercube

Here is an example of the output of the script that is responsible for comparing randomized data using many sci-kit algorithms:


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA