Non-Linear Binary Classification using Support Vector Machine

ECE 4424 Machine Learning Project

By Dominick Perini

Abstract

In this project I chose to implement a Support Vector Machine (SVM) to perform a binary classification. SVMs are able to classify datapoints by finding a hyperplane in n^{th} dimensional space, where n is the number of features per datapoint. Classification is performed by determining the datapoint's relationship to the hyperplane: if the point is "above" the plane then the prediction is class 1, if the point is "below" the hyperplane then the prediction is class 0. This model was used to perform binary classification on the UCI Adult dataset, which is a dataset describing individual people with different backgrounds and the output label is their salary as either greater than or less than $50k. Because the dataset is not linearly separable, soft margins as well as the kernel trick had to be included in the approach in order to produce a useful classifier. The model was optimized using an algorithm called Sequential Minimal Optimization, which is a constrained optimization method specifically for the dual representation of the SVM. Over the training data, accuracy was calculated to be 88.89% with 603 samples. Over the testing data, accuracy was 76.35% with 29559 samples.

Theory

Support Vector Machines are a fascinating application of mathematics. Using high-dimensional vectors to optimize a hyperplane that separates the data is a very abstract concept and is certainly not very intuitive to implement, however the model is (in my opinion) one of the easiest to conceptually explain to anyone no matter their mathematical background.

The SVM in general computes a linear classifier as follows:

$$f(x) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{1}$$

By finding all possible dividing hyperplanes between the datapoints, optimization (in many different forms) finds the hyperplane with the greatest distance between each "support vector", which are the datapoints that are closest to the decision boundary. By maximizing this value, the hyperplane will have the greatest margin and therefore divide the classes as evenly as possible. In the figure below, the left plot shows a group of possible decision boundaries. All are perfectly acceptable for classifying the training data, however finding the optimal hyperplane requires optimization. The "trained" and optimized hyperplane divides the data

with the greatest margin on both sides, an example is shown in the right plot with any data above the plane being classified as “blue” or class 1, and any data below the plane being classified as “red” or class 0 [1].

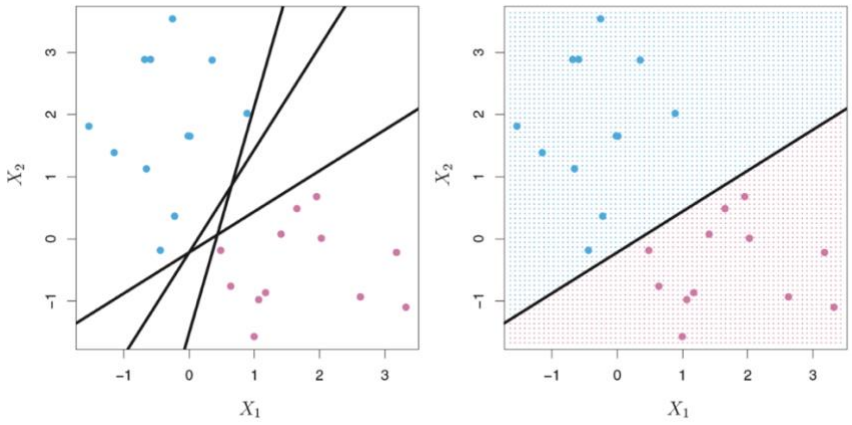

Figure 1: Support Vector Decision Boundaries

In order to use this classifier for the binary classification problem, the output of $f(x)$ can be interpreted as follows: $f(x) \geq 0$ results in a class 1 prediction, $f(x) < 0$ results in a class 0 prediction. The dual representation of SVM shows that this function can also be expressed in terms of Lagrange multipliers and an inner product between the input feature vector and itself.

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, \boldsymbol{x} \rangle + \boldsymbol{b} \tag{2}$$

In my implementation it was clear that the data was nonlinear. With 12 features it is hard to describe what a 12-dimensional hyperplane would “look like” but by using the figure above as a conceptual anchor we can imagine that it still has to separate the data. In order to separate nonlinear data, a new tool must be introduced: the kernel.

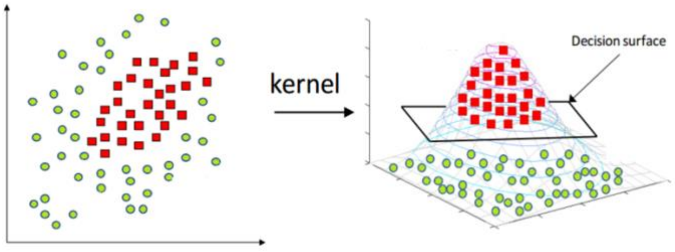

Figure 2: The Kernel Trip

As figure 2 shows [1], data that cannot be linearly separated at first can be transformed into a higher dimensional space and bisected with a hyperplane. There are a multitude of kernels and they all change the dimensionality (or don't) differently. For my implementation I choose the Gaussian kernel or the Radial Basis Function (RBF) because the kernel itself depends on hyperparameters that the programmer chooses, and it is widely documented.

$$K(x,z) = e^{-\frac{||x-z||^2}{2\gamma^2}} \tag{4}$$

The optimization problem overall can be written by substituting the RBF kernel into the dual problem and expressing the constraints on the Lagrange multipliers (alpha vector), as they are optimized to produce the largest margin W around the hyperplane.

$$max_\alpha \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j K\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\rangle \tag{5}$$

Subject to $$0 \leq \alpha_i \leq C, \qquad i = 1, \dots, m \tag{6}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{7}$$

This problem is different from the ones solved in class using methods like gradient descent or closed form optimization, because it has conditions that the optimization is subject to. In order to carry out the optimization, John Platt's Sequential Minimal Optimization (SMO) was used as it is one of the fastest and most efficient ways to perform the necessary constrained optimization.

The SMO algorithm selects two α parameters, α_i and α_j, and optimizes the objective value. Then it adjusts the b parameter based on the new α. This process continues until α converges. The three parts of SMO: selecting the parameters, optimizing the two alphas, and computing b are all on their own challenging problems. In this implementation, the only issue I had was with the third part, computing b. This issue will be further discussed in the implementation details and again in results.

Implementation Details

My implementation can be broken down into four sections: Preprocessing, SVM, SMO, and Accuracy. This project used Python version 3.8 and the final product "SVM.py" can be ran with no command arguments and will output the accuracy statistics of the model. More discussion on how to run the code will be provided in the Reproducibility section.

1. Preprocessing

The UCI Adult dataset consists of 32561 datapoints. In order to easily import, clean, and process all of the data, the Pandas Python library was used, and data was read in as a dataframe. This format has specific advantages when using python libraries to conduct machine learning, however only the convenience of labeled columns for organizing and cleaning the data was utilized by this package.
See the function **def read():** on line 124.

Not all of the datapoints are complete, meaning they include null values or are otherwise unclean, this required that null entries were removed, reducing the clean dataset size to 30162. Each datapoint has 12 features, some of which are categorical and some of which are integers.
See the function **def clean(data):** on line 130.

In some implementations of the classification problem, researches chose to discretize the continuous variables. In this implementation, because SVM works well with continuous variables, scaling was performed on categorical variables to make them continuous and numerical. This procedure was performed on each datapoint, resulting in feature vectors that were complete with continuous features. The last step in the data cleaning was normalization of the data. This step was relatively easy because the data was all continuous. Converting each feature from its standard range into a range of 0 to 1 was a design choice as some documented implementations choose the range -1 to 1, however because the labels of this project are 0 and 1, it was deemed best to use that scale in the features as well.
See the function **def normalize(data):** on line 138.

To use the data for training a machine learning model, a train-test split must be carrier out. Because of the nature of the dual representation of the SVM, training time (and prediction time) increases with the number of datapoints - instead of the number of features, as most models do - therefore a small training percentage of <2% was used in order to achieve reasonable training and prediction times. With a SVM, it is only important that there are enough examples of each class to form the hyperplane for training. Because the support vectors are those examples closest to the hyperplane, having thousands more on either side will only marginally increase the accuracy while exponentially increasing the training time. Using about 600 datapoints randomly selected from the normalized data produced sufficient results.
See the function **def trainTestSplit(data, train_frac):** on line 153.

2. Support Vector Machine

The two functions implemented to do the SVM computations necessary for the model are the classifier function and the kernel function. The classifier is simply equation 2 as detailed in this report, and the kernel is the RBF kernel as shown in equation 4.
See the function **def classifier(alpha, dpi, features, labels, b):** on line 34.
See the function **def kernel(x, z):** on line 26.

While the classifier function does not use any hyperparameters, the kernel function uses the hyperparameter Gamma. Gamma is the variable that determines how far from the decision boundary the SVM will see influence from the data. With a lower Gamma, data that is further away from the hyperplane will influence its shape and be considered for the optimization of the margin size. By using a lower value of Gamma, overfitting is generally avoided at the cost of some incorrect predictions closest to the decision boundary. Figure 3 shows a grid search performed as a part of PCA on the Adult dataset via SVM by researcher Alina Lazar in the paper "Income Prediction via SVM" [4]. The paper concluded that the optimal set of parameters was Gamma = 2^{-3} and C=2^{5}, where C is the regularization parameter, I used these values in my code.

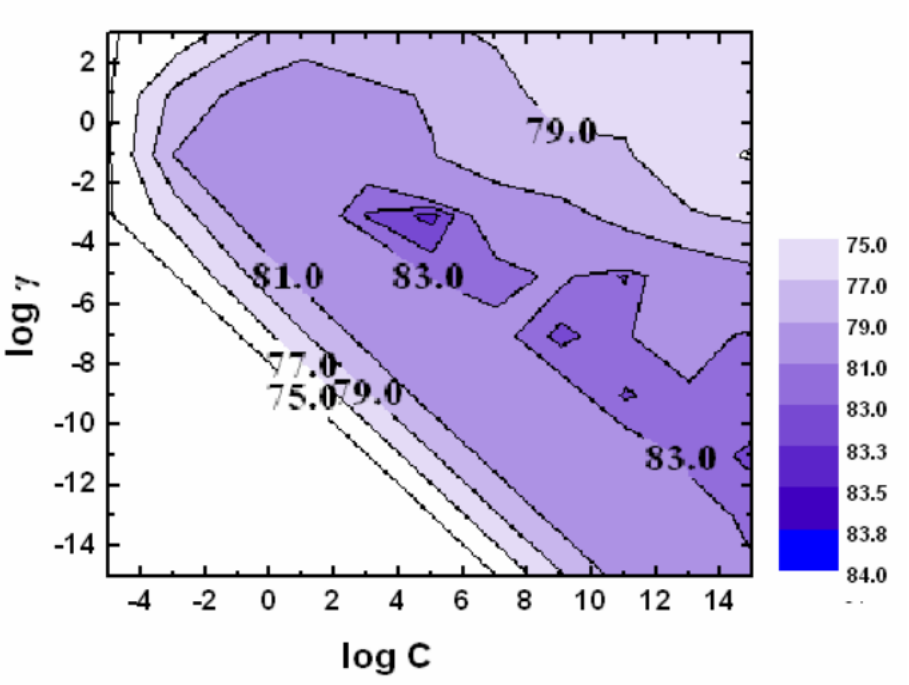

Figure 3: Grid Search of Hyperparameters Gamma and C

3. Sequential Minimal Optimization

The SMO algorithm was the most intensive component of the project to implement. By following Stanford's Simplified SMO pseudocode [3], in which alpha parameter optimization is made less computationally costly, the optimization is not guaranteed to find the global

minima/maxima as it would with John Platt's SMO algorithm [2]. This is acceptable in this instance because a strictly global minima/maxima of the optimization is not required to achieve meaningful results.

As the SMO algorithm runs, it is changing the Lagrange multipliers alpha, but must satisfy the Karush-Kuhn-Tucker conditions (KKT) which are necessary and sufficient conditions for an optimal point of a positive definite QP problem. The KKT conditions are as follows:

$$u = \sum_{i=1}^{N} y_i \alpha_i K\langle \boldsymbol{x}, \boldsymbol{x} \rangle - \mathrm{b} \tag{8}$$

$$\alpha_i = 0 \iff y_i u_i \geq 1 \tag{9}$$

$$0 < \alpha_i < C \iff y_i u_i = 1 \tag{10}$$

$$\alpha_i = C \iff y_i u_i \leq 1 \tag{11}$$

These conditions were embedded in the SMO algorithm. For each step in the Sequence of optimizations, if an alpha did not meet these conditions then it was not changed. For each pass where an alpha is not changed, the counter passes increases, if an alpha is changed then passes reset to 0. If the number of passes increases to where passes is greater than or equal to max passes, where max passes is a hyperparameter selected to be 10, then the loop ends and SMO returns the "optimal" Lagrange vector.

4. Accuracy

Once the optimized model is returned and stored, the prediction based on the model must take place. In order to predict, again equation 2 is used but now with the trained alpha vector and b value. In the predictAndCalcError function a vector of predictions is compared to the label vector, when a mismatch between the prediction and label entry is found, a 1 is marked. The sum of 1's (mismatches) divided by the total number of predictions made gives the error rate of the model. One minus the error rate gives the accuracy, and this value is returned as the output of the SVM.py function.
See the function **def predictAndCalcError(features, labels, alpha, b):** on line 166.

One issue that I found was that when applying the trained alpha and b model to the test data was that it was failing to predict any class 1 examples. By inspecting the actual predictions before they were cut off where a prediction of f(x) > 0 predicts class 1 and prediction of f(x) < 0 predicts class 0, the values were varying as expected but all well below the decision threshold. To solve this, I found that by reducing the influence of the b value, the accuracy increased significantly for both training data and the testing data. This takes place on the fourth line of the predictAndCalcError function, where b divided by 6 is passed into the classifier instead of b.

Reproducibility

The UCI data can be found at this link: http://archive.ics.uci.edu/ml/datasets/Adult

To use the SVM.py file, first change the variable ADULT_DATA_PATH (line 14) to be the file path of the adult dataset. To use the pretrained model that is saved as "alpha_trained.npy" and "b_trained.npy", change the PRETRAINED variable to True. This will avoid the 1 hour 35 minute training time for the SVM model.

By using 0.02 for the train_frac variable which should be used to produce the following results, the prediction and accuracy time should take approximately 3 minutes. By using a memory allocation tracker, computation time is increased by a factor of 3. The python memory allocation tracker caused the 603 training data size model to take too long to train, so only an activity monitor readout of PyCharm during the program running was used as an estimate of memory usage.

Results

Accuracy Results of the model:

Size of Training Data	Training Set Accuracy	Test Set Accuracy
151	88.7%	76.9%
603	88.89%	76.35%

Computational Time and memory of the model:

Size of Training Data	Train Time	Prediction Time (Train + Test)
151	35 minutes	1 minute
603	1 hour 35 minutes	2 minutes 48 seconds

Size of Training Data	Train Memory	Prediction Memory Usage (Train + Test)
603	1.17 GB allocated	20MB Peak instantaneous usage, 1.15GB allocated

Overall, with the pretrained model that used 603 training datapoints, the model is able to achieve an accuracy of 76.35% on the testing data. This was a great result as I was able to confirm that the model was neither predicting all class 0s nor predicting all class 1s. This means that the hyperplane created was actually dissecting the data meaningfully, and the SMO optimization was mostly successful! The training accuracy is higher with the smaller training data set because there are less class 1 examples in the training data to predict incorrectly. There are two ways to solve this, first is a stratified search, the second is increasing the training data to include more class 1 examples. I implemented the second method which was not ideal because training time and prediction time increase with the size of the training data, this should be explored in future versions of this algorithm.

References

[1] G. Zhang, “What is the kernel trick? Why is it important?,” Medium, 11-Nov-2018. [Online]. Available: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d.

[2] J. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines”. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

[3] “The Simplified SMO Algorithm,” CS229, Aug-2009. [Online]. Available: http://cs229.stanford.edu/materials/smo.pdf.

[4] A. Lazar, “Income Prediction via Support Vector Machine,” PSU.EDU. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.387.5068&rep=rep1&type=pdf.

[5] “Support Vector Machines,” MIT.edu. [Online]. Available: https://ai6034.mit.edu/wiki/images/SVM_and_Boosting.pdf.

[6] A. Ng, “Kernel Methods,” Stanford CS 229 Lecture Notes, Aug-2019. [Online]. Available: http://cs229.stanford.edu/notes2019fall/cs229-notes3.pdf.