OTDM: Assignment 2 - Support Vector Machines

Marc Maynou Yélamos Kleber Enrique Reyes Illescas

November 2022

This second assignment consisted primarily on the development of the primal and dual implementation for support vector machines. The code for the two methods can be found in, respectively, *primal.mod* and *dual.mod*. Both of the implementations follow the standard formulation presented in the lectures.

In order to test the correctness of the code, we generated three different sets of data with the provided generator, with three different sizes: 100, 500 and 2.000. However, the generated data needs to be formatted, in order to define the parameters that we are going to use in the models. To fulfill that purpose, we developed *formatData.py*, which takes the raw data generated and shapes it accordingly and stores it into a *.dat* file.

Alongside the two *.mod* files, we developed two *.run* files, one per implementation of the SVM. In these files we load both the model as well as a *.dat* file, we solve the optimization problem defined by the formulation and the data and display the obtained hyperplane (which consists on the normal to the separation hyperplane, *w*, and its location with respect to the origin, γ).

The primal formulation generates the hyperplane automatically (i.e. its components are obtained just by optimizing the problem). However, the dual formulation requires some further processing to define the separation plane, which was also presented in the lectures. There is one implementation caveat worth mentioning when calculating the γ, which is that when evaluating the points to find a support vector (i.e. the lagrange multiplier, λ, for the equality constraints in between 0 and ν) indicating strict comparison $\lambda > 0$ and $\lambda < \nu$ does not work. Instead, we need to slightly modify the values:

```
lambda[i] > 0.001 and lambda[i] < nu-0.001
```

Once the two hyperplanes were built for the two implementations, we made sure that they coincided, which they did for our three *.dat* files. In the following table we have the values obtained for *w* and *gamma*. As we said before, the values coincide for both primal and dual.

sample size	w values	gamma value
100	[2.2459, 2.25604, 1.24609, 2.10406]	-3.73047
500	[3.49547,3.80283,3.31768,3.27376]	-6.82671
2000	[4.32688,4.22863,4.19464,4.42746]	-8.64249

The next step was to validate that the hyperplanes were indeed correct. To do so, we partitioned the *.dat* files into a training set and test/validation set, with an approximate 75/25 split. We would build the hyperplane based on the training data and check whether we could generalize its use to the test set. To validate the predictions, we compared the predicted values with the expected results with a simple accuracy metric. For the 100, 500 and 2.000 data files, we obtained accuracies of 84%, 92.8% and 94.6%, respectively.

It is worth noticing that the bigger the dataset, the better the accuracy. This is likely due to having more data to compute the plane for the latter cases, which generates a plane that takes into account more limit cases and, therefore, can generalize better. Another aspect to highlight is that in our executions the dual solution took more time than the primal execution, which does not make sense given that the primal formulation is designed to be faster. We attribute this to two aspects: not having enough data points and dimensions for the dual formulation to be really effective and to the need of the additional processing to generate the plane.

In order to further test the implementations, we applied it to another dataset. The chosen dataset can be downloaded from https://archive.ics.uci.edu/ml/datasets/Ionosphere. This dataset has 351 instances, 34 continuous attributes and 1 binary attribute as our target. In order to process the source file we develop a simple script, *formatCsv.py*, which converts the ionosphere.data file to one that our first script is capable of processing.

Once again, the hyperplanes coincided for both implementations, and the obtained accuracy was 86.21, which corroborated the results obtained for the generated datasets.