Comparison of Tree, Perceptron, and Support Vector Machine Classifier Performance on Boolean Functions for Autonomous Obstacle Avoidance

Matt Fleetwood

ECE 479 Intelligent Robotics II
Portland State University
Maseeh College of Engineering and Computer Science
February 2018

Version 1.9

Contents

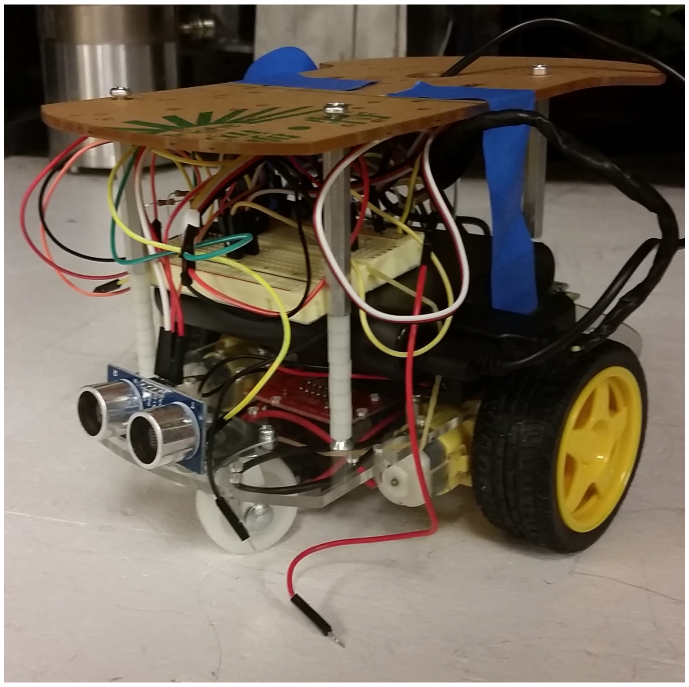

Figure 1: Vikingbot with extra modifications.

1 Abstract

The purpose of this work is to compare the performance of three classifiers: Support Vector Machines (SVM), Perceptrons, and Decision Tree Classifiers. The classifiers are responsible for detecting when an obstacle is present using data from the SR-04 sensor on a Vikingbot. Each classifier is simulated using scikit-learn and trained on obstacle avoidance data based on the HC-SR04 sensor. The Tree performed worst, with an average accuracy of 0.5 while the Perceptron and SVM predicted targets perfectly.

2 Background

Obstacle avoidance is an ongoing problem in computer vision. Machine Learning offers a potential automatic solution to this problem by using classifiers. The Vikingbot was selected because it uses Python on the RaspberryPi 2, which is relatively simple to script and simulate projects in. Python and RaspberryPi also have a large community for support and significant open-source libraries like OpenCV and scikit-learn [1, 5]. As the authors of *Data Mining - Practical Machine Learning Tools and Techniques* argue in Chapter 7 *Data Transformations*, “successful data mining involves far more than selecting a learning algorithm and running it over your data ... in all cases, the right choices depend on the data itself” [6]. Here the author means “right choices” about selecting classifiers (e.g. trees, SVMs, perceptrons) and the respective parameters modified for optimization. The Vikingbot was also ideal because it is mobile, which means it can potentially interact with the environment more interestingly than a stationary robot. For instance, if emotive behavior such as LEDs, LCDs, or other actuators were desired to interact with the environment then it would require little overhead for this application.

2.1 Decision Tree Classifiers

The Tree classifier is commonly used for many applications in machine learning and robotics, the popular flavors being C4.5 and CART [3]. Decision trees utilize ”divide-and-conquer” methods and a common alternative to trees are called rules [6].

2.2 Perceptrons

The Perceptron classifier is a linear model [2]. It is sometimes regarded as more difficult to use than classifiers such as the SVM [2, 3]. Unlike tree classifiers, which make splits depending on the input features, perceptrons use the perceptron learning rule [6]. The goal of this algorithm is to separate classes using a hyperplane whose weights are iteratively modified during training.

2.3 Support Vector Machines

SVMs are currently viewed as being one of the easiest classifiers to apply to most problems [2]. This linear model also takes advantage of instance-based learning, which allow it to create "quadratic, cubic, and higher-order decision boundaries" [6]. These boundaries are made by using the maximum-margin hyperplane. Similar to perceptrons the SVM exhibits issues with computational complexity, meaning the learning time can become impractical, and with overfitting, meaning the model cannot generalize well beyond their training set [6].

3 Data and Experiments

3.1 Goals

The experiments for this project can be summarized as follows:

- 1. Create, simulate, and evaluate the performance of an SVM classifier in Python using Sci-Kit learn.
- 2. Create, simulate, and evaluate the performance of a Perceptron classifier in Python using Sci-Kit learn.
- 3. Create, simulate, and evaluate the performance of a Tree classifier in Python using Sci-Kit learn.
- 4. Compare and evaluate the performance of the SVM, Perceptron, and Tree classifiers to determine which did the best.

3.2 Problem Design

The problem is formulated into several scenarios by using one threshold and one Boolean function. The Boolean mappings are shown in the table below.

Note that since the tree classifier cannot have more members in each class than there are splits or features in a node, the tree was trained on a different mapping than Table 1, shown in Table 2.The threshold function returns one if the input sensor value is within some threshold. For instance, the HC-SR04 sonic sensors report a value of 4 cm if an object is essentially on top of them. On the other hand, the sensors can report up to 300 cm (approximately 3 m).

Table 1: Truth table for the Perceptron and SVM.

Row	A	B	C	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0	0
3	0	1	1	0	0	1	0	0	0	0
4	1	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	1

Table 2: Truth table for the Tree.

Row	A	B	C	Y1	Y2
0	0	0	0	0	1
1	0	0	1	0	1
2	0	1	0	1	0
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	0	1
7	1	1	1	1	0

The threshold function was used for ranges between 4 cm and 12 cm, and 4 and 30 cm. This means if the sensors reported a value between 4 cm and 12 cm, assuming this was the range used, then they return one and otherwise zero.

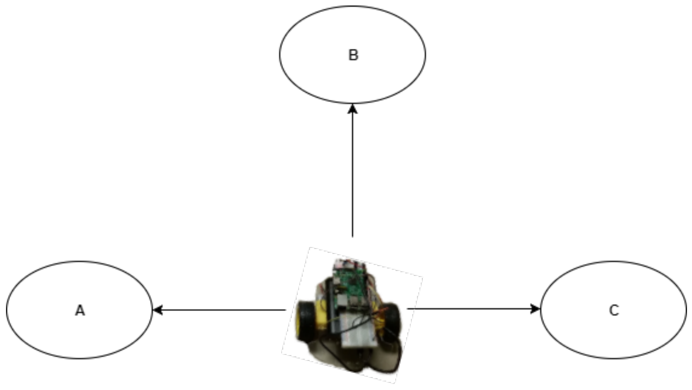

Figure 2: Input features for the Vikingbot detected using the HC-SR04 sonic sensor.

The Vikingbot was used to report three different sensor values: one for forward object scanning (B in the tables above) and one each for scanning to the left and to the right (A and C, respectively). This setup is displayed in Figure 2 above. After the Vikingbot checks for objects in front of it such as B in Figure 2, it turns to the left and checks for objects such as object A. It then turns right twice to check for objects where C is shown. The targets, Y1 - YN, are treated as a class. For example, since the Perceptron and SVM could use one-hot encoding for the target labels, then Y0 in this case represents no object present to the front, left or right of the Vikingbot, Y1 represents (for example) only an object to the right, etc.

4 Results and Conclusions

The results for each classifier is given in the table below. As shown, the average accuracy was the most convenient statistical method for evaluating classifier performance using scikit-learn. In comparison, other ML libraries such as Orange have several metrics that are commonly calculated after an evaluation scheme is chosen [4].

The Tree performed worst at each split by having an average accuracy of 50 per-cent. The Tree was also more complicated to use because it required somewhat strict rules compared to the Perceptron and SVM classifiers. For instance, in the image below, it's shown that there cannot be fewer members in a class than there are inputs or splits in a node. This presents a

Table 3: Classifier performance.

Classifier	Avg. Accuracy
Tree	.50
Perceptron	1
SVM	1

more interesting and perhaps challenging design problem, since it cannot be straightforwardly described in terms of a one-hot encoded target label or class.

The Boolean function used to train the Perceptron and SVM on the other hand had more flexibility in this regard. The class mapping or output could be a simple one-hot representation, and in fact seemed to increase the performance of the SVM and Perceptron although more experiments on more robust data needs to be conducted to conclude that. Generally it is regarded that SVMs, and Perceptrons perhaps to a lesser degree, are treated as "off-the-shelf" components that can be used with little need for optimization because results are decent to begin with. The experiments provided in this project agree with these views. It was trivial to use the Perceptron and SVM, but the Tree required extra tuning.

This work naturally calls for more study into comparing Trees, Perceptrons, SVMs and their performances across different datasets and applications. Research that compares these structures is useful for educational and industrial purposes because it allows users to define their problems and map it to classifiers with known properties more easily. It's also worth noting that the Vikingbot had issues. In particular, the Vikingbot required a pickle protocol that was current to the Pi model used. This means if pickle protocol 3 was used, it would not work for RasperryPi 2. Another issue with the Vikingbot is that it was unable to install SciPy and scikit-learn. This meant scikit-learn classifier's predictions would have to be hard-coded or installed another way. Assuming this issue is resolvable, then the classifiers from scikit-learn could easily be added to the code created so far. The algorithms and code can be found on GitHub. The final video showcasing all of the necessary footage can be found on YouTube.

5 References

[1] A. Rosebrock, "OpenCV panorama stitching," pyimagesearch, 2016.
https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/

[2] Hsu et al, "A Practical Guide to Support Vector Classification," Department of Computer Science, National Taiwan University, Taipei 106, Taiwan 2016. https://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.

[3] Kohavi, Ronny and Quinlan, J. Ross, "Data Mining: Practical Machine Learning Tools and Techniques," Data Mining Tasks and Methods: Classification: Decision-tree Discovery, 267–276, 2002.
http://dl.acm.org/citation.cfm?id=778212.778254

[4] Orange Tool Box, https://docs.orange.biolab.si/3/visual-programming/widgets/evaluation/testa

[5] scikit-learn.com

[6] Whitten, Frank, Hall, "Data Mining - Practical Machine Learning Tools and Techniques," Third Edition.