David Strube - dstrube3@gatech.edu

CS 7641 - Machine Learning

Problem Set 1

2020-09-26

Question 2

Design a two-input perceptron that implements the boolean function $A \wedge \neg B$. Design a two-layer network of perceptrons that implements $A \oplus B$ ($\oplus$ is XOR).

For these problems, I started with an article by Francesco Cicala from 2018 in Towards Data Science in which he explores what perceptrons are and how to make them. (Cicala. (2018.)) For the first half, I basically stole the code for these functions (and put them into a util class because they would also be needed for the second half): *unit_step, perceptron, NOT_percep, AND_percep, OR_percep*. Next I wrote a function *A_AND_NOT_B_percep* that would make use of the functions *AND_percep* and *NOT_percep*.

This is in the file *Problem_Set_1_Q2a.py*

For the second half, I did something a little different from how the Towards Data Science article did it. I copied the aforementioned *A_AND_NOT_B_percep* and used the function *AND_percep*, *NOT_percep*, and *OR_percep*. I modeled the XOR function in a way that I saw in the lecture notes from a class taught in 2012 by Floriano Zini entitled "Machine Learning: Algorithms and Applications" (Zini. (2012.)), like so :

A XOR B = (A $\wedge \neg$B) $\vee$($\neg$A $\wedge$B)

This is in the file *Problem_Set_1_Q2b.py*

Question 4

Explain how you can use Decision Trees to perform regression? Show that when the error function is squared error, then the expected value at any leaf is the mean. Take the Boston Housing dataset (https://archive.ics.uci.edu/ml/datasets/Housing) and use Decision Trees to perform regression.

For this problem, I started with an article by Lorraine Li from 2019 in Towards Data Science in which she demonstrates how to use a Decision Tree (DT) to perform regression on the Boston Housing dataset. (Li. (2019.)) In this article, she explained that maximizing the information gain (IG) is the goal at each split of a DT. Impurity Measure (IM) is a component of the equation for finding IG. The class to use for classification using a DT is *DecisionTreeClassifier*; the class to use for regression using a DT is *DecisionTreeRegressor*. In the constructor of *DecisionTreeClassifier* & *DecisionTreeRegressor*, IM is the *criterion* parameter. If the objective of the DT is classification, then IM is based on entropy (according to the article); if the objective is regression, then IM is based on weighted mean squared error (MSE) instead.

If $\hat{y}_t$ is the predicted target value (sample mean) given by this equation:

$$\hat{y}_t = \frac{1}{N_t} \sum_{i \in D_t} y^{(i)}$$

and N_t is the number of training samples at node t, and D_t is the training subset at node t, and $y^{(i)}$ is the true target value, then MSE is given by this equation:

$$\mathrm{MSE}(t) = \frac{1}{N_t} \sum_{i \in D_t} (y^{(i)} - \hat{y}_t)^2$$

Li then proceeded to demonstrate this using *DecisionTreeRegressor* to plot one column from the dataset: LSTAT (% lower status of the population). She did so using a *max_depth* of 3. Tweaking this value in my code, I found that this is an optimal setting- any more would be overfitting and any less would be underfitting. I also wanted to see if this *max_depth* applied to the other columns, so I wrote some additional code to plot them. I found that for some columns, max_depth of 3 was still optimal; but for others, a *max_depth* of 2 was better suited (*max_depth* of 3 was overfitting). One column, CHAS (Charles River dummy variable) didn't make sense to include in the plots because it was always a 1 or 0.

Yet another column was particularly interesting for other reasons: B ("1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town"). Setting the *max_depth*=3 was overfitting; but setting it to 2 gave rise to a different kind of problem. At *max_depth*=2, the line goes up at about 125, and again at about 350, but at 400, there is a drop. This seems to indicate that there is an optimal value between 350 and 400; anymore is suboptimal. This is beyond the scope of this assigned question for this problem set, but it's worth pointing out. This is problematic in a kind of way that I haven't seen covered thus far in this class, but I have seen mentioned elsewhere: the problem of racial bias exposed in and/or by Machine Learning. This bias is sometimes unintentional, but still this raises some important questions. It makes me wonder how the data was collected; I wonder if there was some inherent systemic racial bias in the data collection that caused this drop after 400. There is a lot of data in the 400 range - maybe the drop is the result of noisy data that this algorithm isn't well equipped to handle. Then again, the line does drop to a level that is higher than the line between 125 and 350. Maybe this kind of drop would be seen in

other racially classified columns. If *max_depth*=1, there is no drop in the line at the end, but there is less nuance up to that point, and it looks like an underfit. At any rate, I left the *max_depth* for this column set to 2 so that it could be considered by future readers.

My code can be found in *Problem_Set_1_Q4.ipynb*

Question 6

Imagine you had a learning problem with an instance space of points on the plane and a target function that you knew took the form of a line on the plane where all points on one side of the line are positive and all those on the other are negative. If you were constrained to only use [a] decision tree or nearest-neighbor learning, which would you use? Why?

It depends on what I was trying to determine: if I was trying to determine where to place the line or how far each point was from the line; if I was trying to determine how many points are on one side of the line or the other; if I was trying to determine the average distance of all points from one side of the line or the other. Without knowing what is the objective of the task, it's hard to say what is the best way to solve it.

Let's assume the objective is to know where to draw the line. In this case, a series of random trial-and-error drawings of lines may be best suited by a Decision Tree, as that is used for the purpose of deciding which attribute is the best answer to a question, and each randomly drawn line can be seen as a node in the tree.

On the other hand, if the task involved determining the distance of one or more points from a given line, a nearest-neighbor learning algorithm would be more suitable. Nearest-neighbor learning is lazy (as opposed to linear regression which is eager), and thus has a runtime of lg n + k (as opposed to linear regression's runtime of n).

References

Cicala, Francesco. (2018.) Perceptrons, Logical Functions, and the XOR problem [https://towardsdatascience.com/perceptrons-logical-functions-and-the-xor-problem-37ca5025790a].

Zini, Floriano. (2012.) Machine Learning: Algorithms and Applications. [http://www.inf.unibz.it/~zini/ML/]. Bolzano Area, Italy: Free University of Bozen-Bolzano.

Li, Lorraine. (2019). Classification and Regression Analysis with Decision Trees [https://towardsdatascience.com/https-medium-com-lorrli-classification-and-regression-analysis-with-decision-trees-c43cdbc58054].