data1010-hw-10

November 20, 2020

1 Homework 10

1.1 Brown University

1.2 DATA 1010

1.3 Fall 2020

1.4 Problem 1

Go to `https://prismia.chat/gallery` and check out the decision tree manipulative (this is the same one we used in class).

(a) Drag all of the red points into the middle and the blue points equally split to the left and right (so that, left to right in order, you encounter five blues, then 10 reds, and then the other five blues). The projections of the points onto a vertical axis should be all mixed up. Does a greedily-trained decision tree end up with training accuracy of 100% for this example? (Note: you should think about the full CART algorithm, where there is no restriction on whether each split is horizontal or vertical, rather than the restricted one in the manipulative which only allows vertical splits at the first stage and horizontal ones at the second.)

(b) Come up with a configuration of the 10 red and 10 blue points where you can achieve 100% training accuracy (and where you would also anticipate having a high test accuracy as well), but where the greedy algorithm gives worse results.

1.4.1 SOL

(a) Yes. The greedy algorithm does not plan ahead so it might lead the solution to a subtopimal stage instead of a global optimal stage. In some simple cases, step optimizations will result in a global optimization. In the first stage, the greedy algorithm will split out the blue points on thel eft or on the right to achieve lowest Gini impurity. The next stage will split out the blue points in the mixed set from the red points. And it will reach 100% accuracy.

(b) Suppose we have two circles of points with red in the inner circle and blue in the outer circle. Using a kernel SVM, we can find a hyperplane that separates the two circles perfectly and also having a high test accuracy. But the greedy CART algorithm will give worse result since it will cut the line vertically and horizontally in a greedy way. Even with a very deep tree,

we can achieve a 100% accuracy with greedy CART but it will perform very badly on test set because of overfitting.

1.5 Problem 2

Go to `https://prismia.chat/gallery` and check out the neural network manipulative (this is the same one we used in class).

(a) Starting from the good configuration of weights, biases, and input coordinates (click 'show best' to get that), move the middle bias dial all the way to the left (middle neuron in the middle column). Wiggle the sliders which govern the weights of the edges feeding into that neuron. What are the derivatives of the predicted gold probabilty (post-softmax) with respect to those two weights?

(b) Suppose that both biases in the last layer are both increased by the same amount. How does that affect the resulting (binary) prediction function?

(c) Suppose you wanted to have the same binary prediction function as the one shown when you click "show best", but you wanted the model's *probability* predictions to soften. In other words, you'd rather that the model go less quickly from nearly 100% gold to nearly 100% purple as you move across from one side of the semicircular boundary to the other.

What change could you make to the weights and biases of the network to achieve this effect?

1.5.1 SOL

(a) Since the bias is set to the left most, the derivatives will be zero since the weights do not affect the output of that neuron any more.

(b) The resulting binary prediction will be unchanged. In a function such as W_1x+b_1, W_2X+b_2in which b_1, b_2 are the biases, if we change the biases the same amount, the relative difference between the two outputs will be the same and the prediction will still be the same as before.

(c) If we want to soften the model's probability predictions, we need to shrink the influence of an input to the output between layers. This "influence" in controlled by the weight matrix but not the bias, so we need to decrease the weights passed into the next layer. However, we should not change the final output, which means the relationship between the two output probabilities should not be cahnged, but the relative difference can be smaller as the process ins softened. As a result, the weights passed to both the last layer need to be decreased with the same ratio.

1.6 Problem 3

We say that a model with a deterministic training process is **scale-invariant** if the following two procedures yield the same predictions for each row of the data frame of test data:

1. train the model on the training data
2. evaluate the model at each test input value to obtain test predictions

and

1. multiply a column of the data frame of training data by a positive number
2. train the model on the new training data, with the modified column
3. multiply that same column of the data frame of *test* data by that same positive number
4. evaluate the model at each new test input value to obtain test predictions

Which of the models we've studied in this course are scale invariant? Be sure to address linear regression for regression problems, logistic regression for binary classification, soft- and hard-margin suppor vector machines for binary classification, and decision trees for classification or regression.

1.6.1 SOL

Linear Regression: It is scale-invariant. Suppose the data is scaled by a constant C. The only thing the model need to do is to rescale the coefficients of that feature by $\frac{1}{C}$ to achieve the same regression output as the previous model. Proof:

$$B = (X^T X)^{-1} X^T Y$$

scaling the data columns

$$X^S = XD$$

$$(DX^T XD)B^s = D^T X^T y$$

hence

$$B^s = D^{-1} B$$

and

$$y^s = X^s B^s = XDD^{-1} B = XB = y$$

Logistic Regression: It is scale-invariant. The maximum likelihood of a logistic regression is

$$l(\beta|y) -> \prod_i (\frac{exp(x_i'\beta)}{1 + exp(x_i'\beta)})^{y_i} (\frac{1}{1 + exp(x_i'\beta)})^{1-y_i}$$

and the MLE is the arg max of that likelihood. When you scale a regressor, you also need to accordingly scale the coefficients to achieve the original maximal likelihood.

Decision Tree: It is scale-invariant, because its training process is to find a certain cut in a certain feature to split out areas. The change of the scale will only lead to a same scale of change in the cutting point but will not affect the model output.

Hard-Margin SVM: It is scale-invariant. If there is a hard-margin hyperplane that perfectly separates the dataset into two classes, then scaling a certain feature does not change that hyperplane as the we just need to scale that dimension of that feature in the hyperplane and we will have a hard margin again.

Soft-Margin SVM: It is not scale-invariant. Since both the linear kernel and rbf kernel include calculation much more complex than just multiplying a positive constant. Scaling will result in different distribution of data points after we map the data to higher dimension space, which leads to different soft margin from the previous model.

[ ]: