CS289A HW5: Decision tree

David Winer

December 24, 2016

Decision tree results

Spam

Training results

I did not add any additional features to the spam dataset. To create random forests, I sampled (with replacement) 90% of the data. With trees of depth 10 and ensembling 10 trees in the random forest case, my prediction accuracies were:

Classifier	**Metric**	**Score**
Decision tree	Training accuracy	84.0%
Decision tree	Validation accuracy	81.4%
Random forest	Training accuracy	84.2%
Random forest	Validation accuracy	81.4%

I decided to increase depth to see if I could get better accuracy. These were my results with depth 20 trees (still ensembling 10 trees in the random forest case):

Classifier	**Metric**	**Score**
Decision tree	Training accuracy	87.2%
Decision tree	Validation accuracy	81.4%
Random forest	Training accuracy	87.4%
Random forest	Validation accuracy	81.4%

Unsurprisingly, training accuracy increased, but validation accuracy stayed the exact same in both cases. I concluded that, at least with the given features, it would be difficult for me to breach validation accuracy of $81 - 82\%$.

Kaggle

My best Kaggle score was 0.78484 using a random forest of 10 trees of depth 20 each.

Walking down the tree

I pulled the first example (index 0), which was a negative example and examined its path down the tree. It correctly classified this example based on the following path (note that ≤ 0 implies equality):

- ! > 0
- meter ≤ 0
- & ≤ 0
- money ≤ 0

- $\$ \leq 1$
- message ≤ 0
- prescription ≤ 0
- volumes ≤ 0
- ; ≤ 0
- pain ≤ 0
- (≤ 0
- # ≤ 1
- other ≤ 1
- bracket ≤ 0
- business ≤ 1

Most common root splits

In this case, the only split at the root among all my trees in my random forest was ! > 0. I was surprised at this result, so I re-created my random forests while sampling less of the data (80%, 50%) and still got the same result. Evidently, exclamation points give us a lot of information!

Census data

Data processing

I started by importing the provided data using `pandas` and separating out the labels from the rest of the data. I was able to use `pandas` to parse out the variables that were integers and those that were strings (the categorical variables). I then imputed the value of the unknown (`?`) variables by assigning each to be the mode of their respective dimensions.

Finally, I then converted the data into a Python dictionary with `pandas` and used `DictVectorizer` to vectorize the categorical variables.

Training results

To create random forests, I sampled (with replacement) 90% of the data. With trees of depth 10 and ensembling 5 trees in the random forest case, my prediction accuracies were:

Classifier	Metric	Score
Decision tree	Training accuracy	91.8%
Decision tree	Validation accuracy	84.3%
Random forest	Training accuracy	92.1%
Random forest	Validation accuracy	85.1%

Kaggle

My best Kaggle score was 0.85184 using a random forest of 5 trees with depth 15 each. For this outcome, I decided to increase the level randomness in my forests and only sampled 50% of the data for each tree.

Walking down the tree

I pulled the first example (index 0), which was a negative example and examined its path down the tree. It correctly classified this example based on the following path (note that ≤ 0 implies equality):

- marital-status=Married-civ-spouse ≤ 0
- capital-gain ≤ 6849
- education-num ≤ 12
- hours-per-week > 40
- capital-loss ≤ 2205
- occupation=Handlers-cleaning ≤ 0
- marital-status=Never-married > 0
- relationship=Not-in-family > 0
- education-num ≤ 10
- workclass=Self-emp-not-inc ≤ 0

Most common root splits

Again, I only found one split at the root among all my trees in my random forest: marital-status=Married-civ-spouse > 0. I was surprised at this result, so I re-created my random forests while sampling less of the data (50%) and still got the same result.

Techniques used

Decision trees

For my decision trees, I used fairly simple stopping criteria – I continued to recurse down the tree, building more nodes unless the current node was pure (one class) or the current node was at a maximum depth value that I set (I learned from trial and error that on both datasets, I stopped gaining incremental validation accuracy after a depth of 15-20). The exact depth parameters I used in each case are listed above.

As explained above, I imputed missing attributes by taking the mode of their respective dimensions.

Finally, I cross validated on different parameters (mostly just tree depth) by setting aside 20% of my data as a validation set and only training on the remaining 80%.

Random forests

I created randomness in my forests' trees by sampling a subset of my data to create the training set for each tree. I played with 50%, 80%, and 90% of the data and found a small (1%) jump in my accuracy from 50% to 80% and an almost insignificant jump (<0.5%) from 80 to 90.

Additionally, for predicting a given example, I decided on which class to assign by taking the most commonly assigned class among my random forest trees.