

Decision tree and overfitting*

Guillaume Wisniewski
guillaume.wisniewski@limsi.fr

January 2018

In this lab, we'll explore using decision trees to make classification decisions on one simple binary classification task: sentiment analysis (is this review a positive or negative evaluation of a product?).

We'll use for prediction are simply the presence/absence of words in the text. If you look in data/sentiment.tr, you'll see training data for the sentiment prediction task. The first column is zero or one (one = positive, zero = negative). The rest is a list of all the words that appear in this product review. These are **binary** features: any word listed has value "=1" and any word not listed has value "=0" (implicitly... it would be painful to list all non-occurring words!).

1 Before you begin ...

Make sure SciKit-Learn is correctly installed:

```
>>> from sklearn.tree import DecisionTreeClassifier
```

If that doesn't work, something is wrong with your installation.

2 Understanding what decision trees are doing

Train a decision tree of (maximum) depth 2 on the sentiment data.

First, we need to load the data:

```
>>> from data import *
>>> X, Y, dictionary = loadTextDataBinary('data/sentiment.tr')
>>> X.shape
(1400, 3473)
>>> Y.shape
(1400,)
```

*Based on the lecture of Hal Daumé III

We have successfully loaded 1,400 examples of sentiment training data. The vocabulary size is 3,473 words; we can look at the first ten words (arbitrarily sorted):


```
>>> dictionary[:10]
['hanging', 'woody', 'originality', 'bringing', 'wooden', 'woods', 'stereotypical', 'sho
```

Now, we can train a depth one decision tree (aka “decision stump”) on this data:

```
>>> from sklearn.tree import DecisionTreeClassifier
>>> dt = DecisionTreeClassifier(max_depth=1)
>>> dt.fit(X, Y)
DecisionTreeClassifier(compute_importances=None, criterion='gini',
                        max_depth=1, max_features=None, min_density=None,
                        min_samples_leaf=1, min_samples_split=2, random_state=None,
                        splitter='best')
>>> showTree(dt, dictionary)
bad?
-N-> class 1 (333 for class 0, 533 for class 1)
-Y-> class 0 (358 for class 0, 176 for class 1)
```


This shows that if you only have one question you can ask about the review it’s that you should ask if the review contains the word ”bad” or not. If it does not (”N”) then it’s probably a positive review (by a vote of 533 to 333); if it does (”Y”) then it’s probable a negative review (by a vote of 358 to 176).

Your first task is to build a depth two decision tree.

1. Draw it on a piece of paper.
2. Convince yourself whether or not it is useful to go from depth one to depth two on this data. How do you know? 
3. It’s important to recognize that decision trees are essentially learning **conjunctions** of features. In particular, you can convert a decision tree to a sequence of if-then-else statements, of the form:

```
if    A and B and C and D then return POSITIVE
elif  A and B and C and !D then return NEGATIVE
elif  ...
```

This is called a “decision list”. Write down the decision list corresponding to the tree that you learned of depth 2.

4. Build a depth three decision tree and “explain” it. In other words, if your boss asked you to tell her, intuitively, what your tree is doing, how would you explain it? Write a few sentences. 
5. It’s not enough to just think about training data; we need to see how well these trees generalize to new data. First, let’s look at training accuracy for different trees:

- Depth 1:

```
>>> np.mean(dt.predict(X) == Y)
0.63642857142857145
```

(Brief explanation: `dt.predict(X)` returns one prediction for each training example. We check to see if each of these is equal to `Y` or not. We want the average number that are equal, which is what the mean is doing. This gives us our training accuracy.)

- Depth 2:

```
>>> np.mean(dt.predict(X) == Y)
0.66000000000000003
```

So the depth two tree does indeed fit the training data better. What about test data?

```
>>> X_test, Ydtest, _ = loadTextDataBinary('data/sentiment.te', dictionary)
>>> np.mean(dt.predict(Xtest) == Ytest)
0.62
```

Note: when we load the development data, we have to give it the dictionary we built on the training data so that words are mapped to integers in the same way!

Here, we see that the accuracy has dropped a bit.

3 Underfitting and overfitting

6. For all possible depths from depth 1 to depth 20, compute training error, development error and test error for the corresponding decision tree. Plot these three curves.
7. What trend do you observe for the training error rates? Why should this happen?
8. If you were to choose the depth hyperparameter based on TRAINING data, what TEST error would you get? If you were to choose depth based on the DEV data, what TEST error would you get? Finally, if you were to choose the depth based on the TEST data, what TEST error would you get. Precisely one of these three is “correct” – which one and why?

4 Implementing a decision tree

9. Implement a `train` method to implement the training algorithm described during the lecture.