Decision Trees for Text Classification in CS2

Kevin Lin
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA, USA
kevinl@cs.uw.edu

Course Data Structures

Programming Language Java

Resource Type Assignment

CS Concepts Non-self-balancing binary trees

Knowledge Unit Programming Concepts

SYNOPSIS

In CS2 courses centering programming with recursion and data structures, binary trees can be used to represent hierarchical relationships between data. Drawing on a machine learning context, this assignment presents an application of binary trees toward text classification that demonstrates how the design of programming abstractions shapes social outcomes. By the end of this assignment, students will not only be able to define methods that recursively construct, traverse, and modify binary trees, but also begin to engage with ethical questions around the design and use of sociotechnical text classification systems.

ACM Reference Format:
Kevin Lin. June 2022. Decision Trees for Text Classification in CS2. In *EngageCSEdu*.ACM, New York, NY, USA 2 pages. https://doi.org/10.1145/3519938

KEYWORDS

data structures, binary trees, binary search trees, recursion, mutation

1 ENGAGEMENT HIGHLIGHTS

In natural language processing, **text classification** is the problem of assigning the correct label to a sentence or document. Text classification algorithms are commonly used in a variety of real-world contexts involving large amounts of user-generated text data, such as classifying spam emails, analyzing user sentiment, and identifying toxic social media comments. Each real-world example is embedded in social definitions of language, so the assignment frames text classification algorithms as *sociotechnical systems* that encode ideas about how the world should work by making normative judgments about language. While the programming tasks for this assignment are only slightly different from the binary tree practice problems that students solve in class, the social applications draw attention to the way that ideas about language can be encoded within binary decisions. Students are tasked to implement 4 binary tree methods that form the data structure foundations of a decision tree machine learning model.

EngageCSEdu, Open Educational Resources, Virtual

ACM ISBN 978-1-4503-9430-7/22/06.
https://doi.org/10.1145/3519938

These contexts are personally-relevant to students not only because students directly or indirectly participate in social media, but also because the assignment raises important questions regarding the consequences of the design and deployment of text classification systems. Every platform must moderate user-generated content in one way or another, but their decisions how moderation is carried out has social consequences. While the machine learning concepts are abstracted-away, this assignments creates opportunities for instructors to raise ethical questions through discussing the limits of binary logic for algorithmic decisionmaking and the social relationships that text classification algorithms reinforce.

- How might the use of binary decisions and the binary return value shape how users experience the text classification algorithm?
- What ideas about language can be encoded by a model that only makes binary decisions based on the presence or absence of a word?
- When enforced by algorithms, how do ideas about language affect people by race, gender, sexuality, class, and their intersections?
- As the complexity or depth of a decision tree model increases, how are these issues differently mitigated and/or exacerbated?
- If the data includes problematic ideas, what is the model's responsibility to counteract problems? What is our criteria for success?

Just as cutting-edge machine learning models can learn to generate racist, sexist, and homophobic text without explicit instruction, decision tree models can also learn problematic ideas about the world from its data. For these reasons and many others, decision tree models are not commonly used in practice today. But it's also precisely in these ethical questions and problems that decision trees offer a compelling context for exploring the intersection of technology and society. By developing students' ethical reasoning in introductory programming—in spaces traditionally devoid of critical counternarratives—we might also begin changing the culture of computing away from one that views technology and society as separate and toward one that views technology and society as deeply interrelated.

2 RECOMMENDATIONS

This assignment is designed to take between 4 to 8 hours after 3 or 4 prior programming lessons on recursive methods involving binary trees. In our course, students will have written 12 recursive binary tree methods that practice all the relevant concepts in general. Without this background, students might feel overwhelmed by the interfaces and classes. This assignment is designed as a culminating experience for assessing student proficiency with writing recursive binary tree methods in context with new interfaces and classes that students have not seen before. For each of the 3 learning objectives, we recommend students know how to solve analogous versions using more familiar classes.

Construction Students can define a method `readTree(Scanner in)` that constructs a binary tree from a formatted instructions text file.

Traversal Students can define several methods that traverse a tree and conditionally print-out or modify tree values.

Modification Students can define a method `tighten()` that compresses a tree by removing single-child branches and promoting their 0/2-child descendants.

Developing components for a machine learning model can be daunting, so it's important to discuss the relationship between programming concepts

and the decision tree model especially if students are not yet comfortable using libraries and code that they did not personally develop. A high-level overview in class that emphasizes the programming task and how all the pieces fit together can be helpful, but even then expect to answer questions about the relationship between scaffold code and the programming tasks. These relationships can also be addressed in advance by posing them to students as questions during class or in the specification. For example, we might ask students to describe how different interfaces and classes will be used for each method and check that their understanding is correct before moving on to programming.

3 MATERIALS

All materials in this assignment are open source. The easiest way to acquire the materials is to download the zip archive from the website. To create the materials from source, clone the repository[1] and run `make` (or reproduce the commands listed in the Makefile).

An instructor guide is provided in the form of the README.md, instructor-guide.docx, and index.html. All three of these files contain the same exact content but in different formats depending on your preference. To create student-facing instructions, refer to the instructor guide for more details.

Students are only expected to read and write program code in TextClassifier.java; all other files contain information that is not strictly required for the assignment. As noted in the instructor guide, students have a choice of different training datasets (in addition to creating their own).

spam.tsv Labeled spam (or not-spam) message dataset.
toxic.tsv Labeled toxic (or not-toxic) comment dataset.
tiny.tsv A subset of 10 examples from toxic.tsv for testing.

Integration tests for each dataset is provided in the TextClassifierTest.java class. However, we encourage instructors to develop further support mechanisms for students working on the assignment. Students will struggle if support is only provided through the integration tests because they are relatively difficult to use for debugging.

Implementation of the `TextClassifier` class relies on the `Splitter` interface and `Vectorizer` class. Students are not expected to understand the implementations of these files.

Splitter.java Divides the given data points into left and right.
  GiniSplitter.java A strategy for splitting using information gain.
  TestSplitter.java A strategy for splitting suitable for testing.
Vectorizer.java Transforms documents into numbers for the `Splitter`.

We also include a simple web app for a more authentic way to experience the text classifier. Instructions for running the web app are described in the instructor guide.

Server.java Runnable web app server.
index.html Web app frontend hosted by the server.

[1] https://github.com/kevinlin1/text-classifier