Article Classification using k-Nearest Neighbors

Group: Grant Baer, Logan Baker, Blake Silton, and Kassi Winter

Project Overview

The goal of this project is to classify articles using the k-Nearest Neighbors (k-NN) algorithm. We will split our documents into training, testing, and validation sets with a 70-20-10 split. Labels will be encoded as -1, 0, and 1, where **left** is inherently further away from **right** than from **center**. The classification process involves several key steps:

1. **Data Preparation:**
    - Read in each document per labeled group.
    - Split the documents into training, testing, and validation sets.
    - Load the documents into respective collections.
2. **Feature Extraction:**
    - Apply TF-IDF to find the 500 most common words.
3. **Hyperparameter Tuning:**
    - Evaluate different values of k using the validation set to find the best performing k.
4. **Classification:**
    - Use the k-NN algorithm to classify the documents based on the chosen k.
    - Calculate and analyze the F1 score to determine the classification accuracy.

Architecture Design

Structure

```
├─ data/
│   ├── raw/
│   │   ├─ left/
│   │   ├─ center/
│   │   └─ right/
│   └── processed/
│       ├─ left/
│       ├─ center/
│       └─ right/
└─ src/
   ├─ DataProcessing/
   │   └─ DataProcessor.java
   ├─ DocumentClasses/
   │   ├─ CosineDistance.java
   │   ├─ DocumentCollection.java
   │   ├─ <interface> DocumentDistance.java
   │   ├─ DocumentVector.java
   │   ├─ <abstract> TextVector.java
   │   └─ files/
   │       └─ stopwords-en.txt
   └─ main/
       ├─ ArticleClassification.java
       └─ KNearestNeighbors.java
```

Detailed Class Descriptions

ArticleClassification (Main Class)

- **Purpose:** Orchestrate the overall workflow from data preparation to model evaluation.
- **Functions:**
    - Build document collections for each label.
    - Separate data into training, validation, and testing sets.
    - Normalize the sets relative to the training set.
    - Tune the value for `k`.
    - Test the model with the tuned `k`.
    - Print metrics.

KNearestNeighbors Class

- **Purpose:** Implement the k-NN algorithm for document classification.
- **Attributes:**
    - `private final DocumentCollection trainingSet`
    - `private final DocumentCollection validationSet`
    - `private final DocumentCollection testingSet`
    - `private Map<Integer, Map<Integer, TextVector>> trainingDocsByLabel`

The following classes are references the classes from lab 2 and 3, with slight variations

DocumentCollection Class

- **Purpose:** Manage collections of documents for training, validation, and testing.
- **Updates:**
    - Modify the constructor to loop through files in a directory, initializing each `TextVector` with a label based on the directory name.
    - Only add non-stopwords to each `TextVector`.
    - Add a method to combine `DocumentCollection` instances.

TextVector Class

- **Purpose:** Represent text documents as vectors for comparison and classification.
- **Updates:**
    - Add a label attribute initialized based on the parent directory of the file.
    - Handle stop words within the `TextVector` class.

Results

- **Precision**
  - Precision was consistently our highest metric leveling out around .6
- **Recall**
  - Our recall typically suffered worse, leveling out around .52
- **F1 Score**
  - Our highest F1 score was around .58, however this required thousands of documents. A more realistic F1 score with a few hundred documents was .55
- **K**
  - We tried various values and approaches for tuning k such as stopping when we increased below a threshold, stopping when the absolute change was below a threshold, and simply taking the value of k which achieved the best F1 score. All methods produced similar F1 scores, and a good value for k was around 350.

Obstacles and Learnings

Throughout the project, we faced several obstacles, including data challenges, determining the optimal number of common words, and tuning the hyperparameter `k`. Each obstacle provided valuable learning experiences, such as the importance of data preprocessing and the intricacies of model validation.

Obstacles with our data included data repetition, which wasn't caught until far later in the development process. We had several articles which were present in many files, with one such article appearing in 71 different files. Proportional data partitioning served as an additional challenge to ensure equal distribution of labels in all sets.

There were also many difficulties during the debugging portion. We had several minor implementation issues that didn't show themselves in an obvious way, but contributed to poor initial results. It wasn't until a thorough analysis of the output as the program was tuning and testing that the issues were noticeable, and a fix could be implemented. This has shown us the importance of thorough code reviews, and careful consideration of design at every step of the way.