Elliot Greenlee
egreenle@vols.utk.edu

Assignment 1

1 Collaborators

Discussed parameter setup for text parsing and shared final accuracy results with Carl Britt and Razieh Baghbaderani.

2 Approach

This report covers the design and implementation for a classification of a set of news data. After preprocessing the data, Multinomial Naive Bayes is run in Spark on article topic and body pairs to predict the topic of previously unseen bodies. This simple classifier is then tested using different settings, with the goal of maximizing accuracy.

In Bayesian theory, one calculates the posterior probability of an input belonging to a particular class by taking the product of prior probability and likelihood of that event occurring and dividing it by the evidence. Methods based on this expression are a popular solution in pattern recognition.

$$P(w_j|x) = \frac{p(x|w_j)P(w_j)}{p(x)}$$

Classes are chosen based on the highest a-posteriori probability $P(w_j|x)$, calculated using the a-prior probability of a given class $P(w_j)$, the evidence $p(x)$, and the probability distribution function for the feature with respect to the given class $p(x|w_j)$.

There are two options for estimating the probability distribution of features: parametric methods and nonparametric methods. In parametric methods, the distribution is assumed to be known; for example, we assume a normal Gaussian distribution. In nonparametric methods, there is no assumption of the form of the distribution. In this project, a multinomial model implemented in the Spark Naive Bayes algorithm is used to estimate the distribution using extracted features.

The raw dataset presented consists of twenty-two .sgm files containing Extensible Markup Language (XML) data. XML is similar in look to HTML, but uses tags to carry the data, rather than to display it. Tags are created by the author to best describe the data, and are not predefined. In the case of this project, the Python library BeautifulSoup4 is used to parse the files.

The .sgm files contain 21577 news articles with the ”reuters” tag, 10377 of which are extracted. The rest are discarded due to a missing topic or body. For each topic of an article, indicated by a D tag, a topic and body pair is added to a list of 13090 total entries. When instances of two topics in one D tag separated by a dash are included, this results in 14383 total entries. Duplicate topics are removed. The full results are presented in table 1.

	Articles	Non-Empty	Topic Entries	+ Dashed Topic Entries
Total	21577	10377	13090	14383

Table 1: The number of articles and entries parsed.

After extracting the data, certain topics of the 135 total are filtered out for use in classification. These are "money", "fx", "crude", "grain", "trade", "interest", "wheat", "ship", "corn", "oil", "dlr", "gas", "oilseed", "supply", "sugar", "gnp", "coffee", "veg", "gold", "soybean", "bop", "livestock", and "cpi". After filtering, 6438 entries remain. Next, the body texts are cleaned by removing newline, slash, and dash characters. Each text has the most common words removed to reduce the noise in the data, using the Stanford NLTK "English" stop words. Each word is stemmed using PorterStemmer, converting words like "running" to "run" for simplicity. After these common text preprocessing operations, the data is written to a .txt file in .csv format. The 10 example entries are in example_data.txt. Additionally, each string topic was converted to an integer representation.

The chosen feature to extract is the Term Frequency-Inverse Document Frequency (TFIDF). The TFIDF is defined as the term frequency, which is either just the count of a term t appearing in a certain document d

$$tf = f_{t,d}$$

or is the frequency of a term appearing in a certain document

$$tf = \sum_{t'} f_{t',d}$$

multiplied by the inverse document frequency, which is the log of the inverse of the frequency of documents containing that term

$$idf = \log \frac{N}{n_t}$$

as

$$tfidf = tf * idf$$

This is performed in two separate methods, one using Spark, and one using Scikit Learn. The Spark method is implemented with dataframes using the Tokenizer, HashingTF, and IDF functions with a minimum document frequency of 1. On average over 10 runs, this method took 1.43 seconds. This method uses raw counts for the term frequency, and also fails to provide a one-to-one matching of the term to the data because it hashes each word. The final representation for each document is two arrays, one of hashes from the document and one of TFIDFs. The non-Spark method is implemented using TfidfVectorizer, also using a minimum document frequency of 1. On average over 10 runs, this method took 0.81 seconds. This method uses actual frequency rather than count for the term frequency, and the final representation for each document is an array with frequencies for each of the total words from all documents, rather than just the single document.

3 Results

The data was split using the PySpark RandomSplit function into 90% training and validation, and 10% testing data. Next, three separate separations of data were modeled and compared for trends in accuracy: 50% training and 40% validation, 60% training and 30% validation, and 70% training and 20% validation. The average over 10 runs was compared for each split, and each model was evaluated on the testing data to find the best one.

Overall, only a small difference can be observed between different amounts of training data; there is a higher variation between runs of the same data than between averages of different data. However, the results point to a declining of accuracy when the amount of training data is increased. This is unusual, but is likely due in this case to an oddity with multiple topics being assigned to a single article body. If instead accuracy is measured by seeing if any of the topics match the predicted topic, indeed accuracy improves. The best model found is the third 50% training data model. The results are presented in table 2 below.

Testing Run	50% Training	60% Training	70% Training
1	32.8	31.0	30.9
2	33.9	32.0	30.1
3	35.7	31.8	32.2
4	33.8	30.9	31.0
5	34.4	30.4	31.2
6	33.0	31.8	31.4
7	31.8	31.2	30.4
8	31.2	31.5	30.0
9	33.1	31.2	29.0
10	32.3	32.5	31.2
Average Validation	33.9	32.7	32.1

Table 2: The accuracy results from the Naive Bayes classifiers on average from the validation data and each run on the test data.

4 Code

The code for this project is written in a Jupyter notebook using Python3 and PySpark. Findspark is used to integrate Spark, and will need to be altered or removed to run on another system. A pip requirements file is provided as requirements.txt.