Northeastern University, Khoury College of Computer Science

CS 6220 Data Mining — Assignment 3

Due: February 1, 2023(100 points)

YOUR NAME
YOUR GIT USERNAME
YOUR E-MAIL

1 Preprocessing Raw Data from Multiple Sources

The purpose of this section is to build training pre-processing code that would enable us to do machine learning. Typically there are multiple data sources, and in order to create training samples, we'll need to join several of them together. Also, in subsequent lectures, we'll learn that several NLP algorithms rely on self-supervision (using the existing data in the absence of labels) in order to train large scale neural networks. In this assignment we'll explore the process of getting multiple raw data sources into the form of training data, a common task that engineers spend a significant amount of time doing.

1.1 The Dataset

News articles are commonly aggregated from multiple sources. The landscape of news has been evolving ever since social media has amplified its effects, and a big topic that has come up in Congress is the idea of bias. That is, news articles may cover news stories with differing types of verbiage and language at varying frequencies.
The data that we will be using today comes from Kaggle, and it is available here. There are two CSV files that we wish to join this week's homework:

- `data/id_titles.csv`
- `data/id_publishers.csv`

As there name suggests, these are publishers and titles paired with data, along with a lot of other data. Each record as an ID associated with it and it is possible to join the two data sources.

1.2 The Tensorflow Example

One of the most common ways to pass messages from one place to another in real-time is through protocol buffers, *protobufs* for short. Protobufs are language-neutral and developed by Google to serialize data (convert into bytes), efficiently stored (in binary form), flexibly be defined, and subsequently loaded into memory quickly.
Tensorflow Records (TFRecords) or TFExamples are a protobuf that has been the standard for many machine learning serving systems, and take up significantly less space than the original data on distributed file systems, often partitioned into multiple files (i.e., `SSTables` through `BigTables`.) They can be read in with parallel I/O, which can be useful for training machine learning algorithms on large scale data.
If you look at Tensorflow Example's proto structure [1], they use the `Feature` proto [3]:

```
import "tensorflow/core/example/feature.proto";

message Example {
  Features features = 1;
}
```

Each feature can be of type `bytes`, `float`, or `int64`. As well, you can have arrays of variable length of the above data types (that's what meant by the `repeated` in the `*.proto` message. Please review Tensorflow Example's proto, and there are good resources on how to read and write examples in Python [2]. At a high level, the following is an example of how Tensorflow Examples are structured:

```
example = text_format.Parse('''
   features {
       feature {
         key: "my_feature"
         value {int64_list {value: [1, 2, 3, 4]}}
       }
       feature {
         key: "string_array"
         value {bytes_list {value: ["a", "b", "c", "d"]}}
       }
   }''', tf.train.Example())
```

(Please use more standard data ingestion rather than creating a string in protobuf format.)

1.3 Data Exploration

In most data exploration, we would like to take the distribution of the data. In this case, we can use a histogram of the words. Oftentimes, scientists will cut off extremely frequent words and extremely infrequent words.
In this subsection, take the distribution of the words in all of the titles in the dataset. Plot out the distribution, and provide some design decisions as to what words you're deciding to include in the distribution.

1.4 Create Training Data

Word2Vec [5] is a popular self-supervised algorithm that windows surrounding words in order create analytics, a technique called the skip-gram. That is to say, the training set consists of

- An anchor word
- Its surrounding words in an array [word1, word2, word3, ...]. These surrounding words are centered around the anchor word.

So, for example, let's say we had the following sentence

The rain in Spain falls mainly on the plains.

After preprocessing the sentence, for a window of size 5, the corresponding `tf.Examples` for the anchor `Spain` would be:

```
example = text_format.Parse('''
   features {
       feature {
         key: "anchor_word"
         value {bytes_list: "spain"}
       }
       feature {
         key: "surrounding_words"
         value {bytes_list {value: ["rain", "in", "falls", "mainly"]}}
       }
       feature {
         key: "news_source"
         value {bytes_list: "<news_source>"}
       }
       feature {
         key: "time"
         value {int64_list: 204}
       }
   }''', tf.train.Example())
```

Because the window is five, we pick two words on either side of `Spain` (which we have lowercased as `spain`). On the left side, these words are `rain` and `in`. On the right side, these words are `falls` and `mainly`. Therefore the feature array would be {`rain`, `in`, `falls`, `mainly`}.
Very often you will find that the window will come up against the edge of text. Only include words in the window, and if the window surpass the end or beginning of the title, you need not include anything. For example, the anchor `plains` has no words on the right side of the anchor. Therefore, the feature array would be {`on`, `the`}, an array of size two. Likewise, the anchor `rain` will have only a single word `the` to the left side of the anchor in the window. Therefore, the feature array would be {`the`, `in`, `spain`}, an array of size three.
Notice there are other features. Those will become clearer in the next section.

1.5 Adding in News Source and Time

You'll notice two sources of data. You'll need to join the two to understand who wrote what article. The common field between the two is ID. Explain what type of `join` you used and why. Provide examples and samples of data as supporting evidence.
With the joined data, add in the feature of who wrote the story. It is at your discretion which field to use for this data (i.e., `PUBLISHER`, `URL`, or `HOSTNAME`). Be able to justify your design decision. Provide some analysis (histograms, etc.).
Finally, add in time information. A common format for timing in computers is UTM time [6], which is the provided `TIMESTAMP`. This is often not useful as a feature for machine learning algorithms. Think back to a business use case of how we'd like to use our data. For example, would we want to know about trending words (e.g., "millenial", "covid", etc.) or countries (e.g., "korea", "taiwan", "mexico", etc.) on an hourly basis? A monthly basis? There is no correct answer, so please justify your design decision. (Sometimes it helps to understand the trends in the data.)

2 Submission Instructions

- Commit your code to main in Github, and provide the link. We will snapshot and download it. If you created your code in Colab, you can download your Colab as an iPython Notebook by going to `Download` and selecting `Download .ipynb`.
- Include 10,000 samples in your repository and call it `output.tfrecord`.
- Your documentation and code's legibility is part of our grading criterion, so please make sure it's readable.
- In your writeup (i.e., `assignment3.pdf`), include a pipeline description with a diagram.

References

[1] Tensorflow Example Protobuf: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/example/example.proto

[2] Example Reading and Writing Examples in Tensorflow https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/load_data/tfrecord.ipynb

[3] Tensorflow Feature Protobuf: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/example/feature.proto

[4] Google's Docs on Protocol Buffers: https://developers.google.com/protocol-buffers

[5] Mikolov, Tomas; et al. (2013). "Efficient Estimation of Word Representations in Vector Space". arXiv:1301.3781

[6] The UNIX Timestamp https://en.wikipedia.org/wiki/Unix_time