**FINAL PROJECT -- Jingnan Qi**

**Intro**

The dataset is from Kaggle, called Tweets Blogs News – Swiftkey Dataset 4million. It includes tweets, blogs, and news articles total of 4 million text entries, and it could be a perfect dataset for NLP. I only use the US English language editions.

**Data source:** <https://www.kaggle.com/crmercado/tweets-blogs-news-swiftkey-dataset-4million#de_DE.twitter.txt>

**Question**:

1. When I first saw the dataset, I thought it could be interesting to figure out what are the most frequently used words for Americans in their daily life.
2. I was also wondering if there would be any difference among the words used in tweets, blogs, and news articles.
3. I use Twitter a lot and it can only type in 140 characters at a time, which could be annoyed sometimes, so I was also curious if this character limit would do any change to people’s language.
4. In addition, as more and more young generations are into using abbreviations, such as idk, lol, idr, nbd…, maybe I could see if those innovative language are increasingly used.
5. Furthermore, derived from the results, there could be potential suggestions and innovative thoughts about implementing future predictive text application.

**Code Optimization**

1. **Performance Factors Charts**

|  |  |
| --- | --- |
| **Serialization Method** | **RunTime** |
| None | 4:45 |
| Pickle | 4:46 |
| Marshal | 4:51 |

|  |  |
| --- | --- |
| **Compression Method** | **RunTime** |
| None | 4:45 |
| Gzip | 4:40 |
| Bzip2 | 4:13 |

|  |  |
| --- | --- |
| **Checkpoints Method** | **RunTime** |
| None | 4:45 |
| Times | 4:41 |

|  |  |
| --- | --- |
| **Memory Management Method** | **RunTime** |
| None | 4:45 |
| MEMORY\_ONLY\_SER | 3:48 |
| MEMORY\_AND\_DISK | 4:12 |

|  |  |
| --- | --- |
| **Scheduling Algorithm** | **RunTime** |
| None | 4:45 |
| Schedule | 4:38 |

1. **Pivot table with most efficient combinations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Serialization** | **Compression** | **Checkpoints** | **Memory**  **Management** | **Scheduling Algorithm** | **RunTime** |
| Marshal | None | Times | MEMORY\_AND\_DISK | Schedule | 4:17 |
| None | Gzip | Times | MEMORY\_AND\_DISK | Schedule | 3:57 |
| Pickle | Bzip2 | None | MEMORY\_ONLY\_SER | None | 3:34 |

By using Bzip2 compression scheme, with in MEMORY\_ ONLY\_SER memory management, in combination with Pickle serialization protocol, I achieved top performance from my code.

Pickle is more efficient than using Marshal. Bzip2 is more space-saved than Gzip. Besides, Bzip2 allows me to access to the txt.bz file after the compression, which means I did not have to read the original big txt file. It highly saved the memory and space. I set the compression level as 5 from 0-9. Although 9 seems like the most space-saved number to choose but it also needs the longest compression time, in order to better balance the run time and efficiency, I chose 5. MEMORY\_ONLY\_SER is more efficiently than MEMORY\_AND\_DISK, because it stored RDD as serialized objects, which is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read. As for checkpoints and scheduling algorithm, although they seemed to contribute a lot to the efficiency of the code, they did not contribute more to the reduction of the run time, hence I did not put them in the combination at last.

From none of the performance factor, the run time is initially 4:45 (285s), the best combination result is 3:34(214). Reduced by 24.912%.

**Summary**

1. Results and answers:

Top 10 word count (high to low):

[(4704698, 'the'), (2732578, 'to'), (2371971, 'and'), (2354107, 'a'), (1997143, 'of'),

(1613801, 'in'), (1571509, 'i'), (1083919, 'for'), (1043594, 'is'), (979101, 'that')]

As we can see from the results, the top 10 word count is not representative and meaningful enough to answer the questions I want to answer.

So I take a look at the top 100 words, then I found something interesting:

(152143, '-'), (114433, '&'), (85209, ':)'), (69771, ':'), (67463, '--'), (56211, '2'), (54682, '–'),

(52584, '—'), (39863, '.'), (35022, '1')

As the top 10 most frequently used punctuations and numbers. Besides,

(150794, 'good'), (142654, 'love'), (108701, 'great'), (78907, 'thanks'), (69548, 'best'), (59808, 'happy'), (53081, 'lol'), (52620, 'better'), (47075, 'thank'), (33262, 'please')

We can tell that although in the news, blogs, and tweets, people still tend to use positive words. Seen from those findings, there is a trend and pattern to dig deeper to explore the language and words expression changing in the modern society.

I was kind of anticipated the results considering the most frequently used words are surely prepositions and pronouns. The top 2-word counts are ‘the’ and ‘to’ for all 3 media sources. So, I dig furthermore beyond top 20 to see if there are any more changes to notice. In the top 100 mostly used word in tweets, I saw the appearance of ‘RT’, ‘:)’, and ‘lol’ those relatively more modern language and emojis have been highly used, thus the character limit of 140 does in some ways motivate the usage of emojis and abbreviations. Derived from the results, it could provide some insights to predictive text applications producers with how to improve the user experience and efficiency when texting, tweeting, and writing.

1. Challenges:
2. Hard to find suitable dataset. Although there is no perfect dataset, it took me a lot of time to look for the reasonable dataset to do the project. Word count sounds like a easy function but in order to reveal useful application and answer meaningful questions, it is important to find a good dataset as the basis.
3. Hard to clean and read data. Considering the core concepts of this course—big data, it seems only if we could process more than one gigabyte data hence utilize the usefulness of the technology. But the data is not all in good and identical format, which also took me some efforts to research how could I get a clean dataset after loading and filtering.
4. Relatively longer time to run the code. Sometimes the code worked but sometimes it did not, so I had to look up in the internet to find solutions. When the data is too large the code did not go through.