assignment2

December 2, 2024

1 SIADS 516: Homework 2

- **Dr. Chris Teplovs**, School of Information, University of Michigan
- **Kris Steinhoff**, School of Information, University of Michigan

[156]:
```
# The AutograderHelper class provides methods used by the autograder.
from autograder_helper import AutograderHelper
```

[157]:
```
# Autograder cell. This cell is worth 0 points.
# This cell has hidden code used to configure the autograder.
```

2 Using the Spark RDD API to analyze text

Data are from https://www.kaggle.com/nzalake52/new-york-times-articles

2.1 Objectives

1. To gain familiarity with PySpark
2. To learn the basics of the Spark RDD API
3. To practice solving a real-world problem

2.2 Overview

This project was inspired by an actual event that was experienced by a UMSI student. This student was applying for a job with a large multi-national corporation (let's call it XYZ, Inc.). XYZ Inc. was looking for someone who could conduct an analysis of a massive (terabyte-size) text dataset. They had heard about Spark and planned on investigating it but hadn't yet found someone internally who had the skill set required to tackle the problem. The UMSI student indicated that they had experience with Spark and could likely handle the task. The hiring supervisor then provided a non-Spark script and asked the student to demonstrate how that script could be translated to work in a Spark environment. The student was able to do the conversion and, pending completion of their degree, will have secured a job at XYZ, Inc.

This assignment simulates that exact situation. **In this assignment you will take a python-based script that does part-of-speech tagging on a large dataset and convert it, as much as possible, to use a pyspark-based approach.**

2.2.1 Task: Review non-Spark code

The original script was written by Luke Petschauer and a forked version is available in this notebook: NP_chunking_with_the_NLTK.ipynb.

It provides a detailed explanation of the original code and an excellent overview and justification for the use of part-of-speech tagging and a super-gentle introduction to Natural Language Processing (NLP).

Let's use some of the code from that notebook here…

We'll start by importing the required packages, and making sure the NLTK collections are downloaded to your environment.

[158]:
```
import nltk
import re
import pprint
from nltk import Tree

nltk.download(
    "book"
)  # NOTE: this should be unnecessary for Coursera image (should be preloaded)
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /home/jovyan/nltk_data…
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package brown to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package chat80 to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package cmudict to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package conll2000 to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package conll2000 is already up-to-date!
[nltk_data]    | Downloading package conll2002 to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package conll2002 is already up-to-date!
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package dependency_treebank is already up-to-date!
[nltk_data]    | Downloading package genesis to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to
[nltk_data]    |     /home/jovyan/nltk_data…
```

```
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package ieer to /home/jovyan/nltk_data…
[nltk_data]    |   Package ieer is already up-to-date!
[nltk_data]    | Downloading package inaugural to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package nps_chat to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package nps_chat is already up-to-date!
[nltk_data]    | Downloading package names to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package ppattach to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package ppattach is already up-to-date!
[nltk_data]    | Downloading package reuters to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package reuters is already up-to-date!
[nltk_data]    | Downloading package senseval to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package senseval is already up-to-date!
[nltk_data]    | Downloading package state_union to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package state_union is already up-to-date!
[nltk_data]    | Downloading package stopwords to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package swadesh to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package swadesh is already up-to-date!
[nltk_data]    | Downloading package timit to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package timit is already up-to-date!
[nltk_data]    | Downloading package treebank to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package toolbox to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package toolbox is already up-to-date!
[nltk_data]    | Downloading package udhr to /home/jovyan/nltk_data…
[nltk_data]    |   Package udhr is already up-to-date!
[nltk_data]    | Downloading package udhr2 to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package udhr2 is already up-to-date!
[nltk_data]    | Downloading package unicode_samples to
```

```
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package unicode_samples is already up-to-date!
[nltk_data]    | Downloading package webtext to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package webtext is already up-to-date!
[nltk_data]    | Downloading package wordnet to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_treebank_pos_tagger to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package maxent_treebank_pos_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package universal_tagset to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package universal_tagset is already up-to-date!
[nltk_data]    | Downloading package punkt to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package tagsets to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package tagsets is already up-to-date!
[nltk_data]    | Downloading package panlex_swadesh to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package panlex_swadesh is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /home/jovyan/nltk_data…
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection book
```

[158]: True

[159]:
```
# Download necessary NLTK data
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt_tab to /home/jovyan/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /home/jovyan/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /home/jovyan/nltk_data…
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /home/jovyan/nltk_data…
[nltk_data]   Package words is already up-to-date!
```

[159]:
```
True
```

The code in the next cell is from the "Final Code" section in the NP_chunking_with_the_NLTK.ipynb notebook (linked above). This implementation doesn't use Spark.

[160]:
```
# This is the original (non-Spark) script

patterns = """
    NP: {<JJ>*<NN*>+}
    {<JJ>*<NN*><CC>*<NN*>+}
    """

NPChunker = nltk.RegexpParser(patterns)


def prepare_text(input):
    sentences = nltk.sent_tokenize(input)
    sentences = [nltk.word_tokenize(sent) for sent in sentences]
    sentences = [nltk.pos_tag(sent) for sent in sentences]
    sentences = [NPChunker.parse(sent) for sent in sentences]
    return sentences


def parsed_text_to_NP(sentences):
    nps = []
    for sent in sentences:
        tree = NPChunker.parse(sent)
        for subtree in tree.subtrees():
            if subtree.label() == "NP":
```

```
                t = subtree
                t = " ".join(word for word, tag in t.leaves())
                nps.append(t)
    return nps


def sent_parse(input):
    sentences = prepare_text(str(input))
    nps = parsed_text_to_NP(sentences)
    return nps
```

[161]:
```
text_to_be_analyzed = """\
WASHINGTON - Stellar pitching kept the Mets afloat in the first half of last␣
 ↪season despite their offensive
woes. But they cannot produce an encore of their pennant-winning season if␣
 ↪their lineup keeps floundering
while their pitching is nicked, bruised and stretched thin.

"We were going to ride our pitching," Manager Terry Collins said before␣
 ↪Wednesday's game. "But we're not
riding it right now. We've got as many problems with our pitching as we do␣
 ↪anything."

Wednesday's 4-2 loss to the Washington Nationals was cruel for the␣
 ↪already-limping Mets. Pitching in Steven
Matz's place, the spot starter Logan Verrett allowed two runs over five innings.
 ↪ But even that was too large
a deficit for the Mets' lineup to overcome against Max Scherzer, the Nationals'␣
 ↪starter.

"We're not even giving ourselves chances," Collins said, adding later, "We just␣
 ↪can't give our pitchers any
room to work."

The Mets did not score until the ninth inning, when a last-gasp two-run homer␣
 ↪by James Loney off Nationals
reliever Shawn Kelley snapped a streak of 23 scoreless innings for the team.
"""


nps = sent_parse(text_to_be_analyzed)

# Print a list of noun phrases found in text_to_be_analyzed
print(nps)
```

```
['Stellar pitching', 'afloat', 'first half', 'last season', 'encore', 'pennant-
winning season', 'lineup', 'pitching', 'thin', 'pitching', 'game', 'pitching',
```

```
'anything', '4-2 loss', 'place', 'spot starter', 'deficit', 'lineup', 'starter',
'room', 'ninth inning', 'last-gasp two-run homer', 'reliever', 'streak', 'team']
```

You will be taking a similar approach to analyze a large set of news articles from the New York Times using pyspark.

Before you continue to the next task, you should: - read through and study the NP_chunking_with_the_NLTK.ipynb notebook (linked above) - study and run the cells above.

---

[162]:
```
# Import packages and set up a Spark Session and Context.

from pyspark.sql import SparkSession

spark = (
    SparkSession.builder.master("local[*]")
    .appName("SIADS 516 Homework 2")
    .getOrCreate()
)

sc = spark.sparkContext
sc._conf.set("spark.default.parallelism", 2)
```

[162]:
```
<pyspark.conf.SparkConf at 0x7c2ce4416920>
```

2.3 Question 1 – Part-of-Speech Count

2.3.1 Task: Create an RDD pipeline to show the count of each part-of-speech tag sorted in descending order

Complete the implementation of the `pos_counts()` function below so that it uses an RDD pipeline (i.e. sequence of transformations) to: 1. filter out blank lines 2. filter out lines starting with 'URL' 3. create a single list (using flatMap) that applies the `pos_tag_counter()` function (this is defined below for you below) to each line 4. map each resulting element to show the part of speech (which is the second element returned from the pos_tag_counter) 5. convert each resulting element to a pairRDD with POS tags as keys and values of 1 6. reduce the resulting RDD by key, adding up all the 1s (like the lecture and lab examples) 7. sort the resulting list by the counts, in descending order

[163]:
```
# This is the function you will use with flatMap in your pipeline.

TOKEN_RE = re.compile(r"\b[\w']+\b")


def pos_tag_counter(line):
    toks = nltk.regexp_tokenize(line, TOKEN_RE)
    postoks = nltk.tag.pos_tag(toks)
```

```
    return postoks
```

[164]:
```
def pos_counts(rdd):
    # Filter out blank lines
    filtered_rdd = rdd.filter(lambda line: line.strip() != "")
    # Filter out lines starting with 'URL'
    filtered_rdd = filtered_rdd.filter(lambda line: not line.startswith('URL'))
    # Create a single list using flatMap that applies pos_tag_counter() to each␣
 ↪line
    pos_tagged_rdd = filtered_rdd.flatMap(pos_tag_counter)
    # Map each element to show the part of speech (second element)
    pos_only_rdd = pos_tagged_rdd.map(lambda x: (x[1], 1))
    # Reduce the RDD by key, adding up all the 1s
    pos_count_rdd = pos_only_rdd.reduceByKey(lambda a, b: a + b)
    # Sort the resulting list by the counts in descending order
    pos_total_sorted = pos_count_rdd.sortBy(lambda x: x[1], ascending=False)
    return pos_total_sorted
# This should be the final stage of your pipeline, an RDD with the
# count of each part-of-speech tag sorted in descending order.
```

Let's start by trying your code on a small data set. The `text_to_be_analyzed` from the cells above will do nicely. We can use the parallelize() method to turn it into a RDD, pass that to your function, and then take() the first ten entries:

[165]:
```
small_text = sc.parallelize(text_to_be_analyzed.split("\n"))

small_pos_counts = pos_counts(small_text)
```

[166]:
```
small_pos_counts_take_10 = small_pos_counts.take(10)
small_pos_counts_take_10
```

[166]:
```
[('NN', 30),
 ('NNP', 24),
 ('IN', 20),
 ('DT', 16),
 ('VBD', 11),
 ('RB', 11),
 ('NNS', 10),
 ('JJ', 10),
 ('PRP$', 7),
 ('PRP', 6)]
```

[167]:
```
# Autograder cell. This cell is worth 2 points (out of 20). This cell does not␣
 ↪contain hidden tests.
```

```
# This cell deliberately includes answers to provide guidance on how this␣
 ↪question is graded.

correct = AutograderHelper.parse_spark_take(
    [
        ("NN", 30),
        ("NNP", 24),
        ("IN", 20),
        ("DT", 16),
        ("VBD", 11),
        ("RB", 11),
        ("NNS", 10),
        ("JJ", 10),
        ("PRP$", 7),
        ("PRP", 6),
    ]
)

submitted = AutograderHelper.parse_spark_take(small_pos_counts_take_10)

AutograderHelper.assert_same_shape(correct, submitted)
AutograderHelper.assert_same_rows(correct, submitted)
AutograderHelper.assert_same_columns(correct, submitted)
```

Now let's run it against a much larger data set. *The complete analysis could take about 10 minutes to run.*

[168]:
```
text = sc.textFile("../../assets/data/nytimes/nytimes_news_articles.txt")

pos_counts = pos_counts(text)
```

[169]:
```
pos_counts_take_10 = pos_counts.take(10)
pos_counts_take_10
```

[169]:
```
[('NN', 1126515),
 ('IN', 928916),
 ('NNP', 853093),
 ('DT', 761492),
 ('JJ', 498482),
 ('NNS', 437116),
 ('VBD', 379509),
 ('PRP', 282603),
 ('RB', 271053),
 ('CC', 231491)]
```

[170]:
```
assert pos_counts_take_10[0] == (
    "NN",
    1126515,
), "The first item in the result is not correct."
```

[171]:
```
# Autograder cell. This cell is worth 8 points (out of 20). This cell contains␣
 ↪hidden tests.
```

2.4 Question 2 – Noun Phrase Length

2.4.1 Task: Create an RDD pipeline to show the distribution of the length of noun phrases

Complete the implementation of the `noun_phrase_length_distribution()` function below so that it uses an RDD pipeline to return a PairRDD which contains the distribution of the length of noun phrases.

- The function `get_noun_phrases()` is defined below for you.
- You can use flatMap() to apply `get_noun_phrases()` to each entry in the input RDD.
- Sorting the resulting list by the counts in descending order will make the results easier to interpret.
- Note that the filters from the previous question (PART-OF-SPEECH COUNT) are **not needed here**. That is: you *don't* need to remove blank lines or lines that start with "URL".

[172]:
```
# This cell defines the get_noun_phrases() function you will use with flatMap()

grammar = r"""
    NBAR:
        {<NN.*|JJS>*<NN.*>}

    NP:
        {<NBAR>}
        {<NBAR><IN><NBAR>}
"""


def get_noun_phrases(line):
    """
    This function returns a list of lists of tuples. Each entry (list of␣
 ↪tuples) is a
    breakdown of a noun phrase, and each tuple contains the word and a code for␣
 ↪the noun
    phrase part.

    For example, get_noun_phrases("The quick brown fox, jumps over the lazy dog.
 ↪") returns:
```

```
    [
        [('brown', 'NN'), ('fox', 'NN')],
        [('dog', 'NN')]
    ]

    """

    TOKEN_RE = re.compile(r"\b[\w']+\b")

    chunker = nltk.RegexpParser(grammar)

    toks = nltk.regexp_tokenize(line, TOKEN_RE)
    postoks = nltk.tag.pos_tag(toks)

    if len(postoks) == 0:
        return []

    tree = chunker.parse(postoks)

    return [term for term in leaves(tree)]


def leaves(tree):
    for subtree in tree.subtrees(filter=lambda t: t.label() == "NP"):
        yield subtree.leaves()
```

[173]:
```
def noun_phrase_length_distribution(rdd):
    # Use flatMap to apply get_noun_phrases to each entry in the input RDD
    noun_phrases_rdd = rdd.flatMap(get_noun_phrases)

    # Map each noun phrase to a key-value pair (length of noun phrase, 1)
    length_rdd = noun_phrases_rdd.map(lambda np: (len(np), 1))

    # Reduce the RDD by key (length) to get the count of each length
    length_counts_rdd = length_rdd.reduceByKey(lambda a, b: a + b)

    # Sort the results by counts in descending order
    sorted_length_counts_rdd = length_counts_rdd.sortBy(lambda x: x[1],␣
 ↪ascending=False)

    return sorted_length_counts_rdd
    # This should be the final stage of your pipeline, a PairRDD with the
    # distribution of the length of noun phrases.
```

[174]:
```
small_counts = noun_phrase_length_distribution(small_text)
```

[175]:
```
small_counts_take_10 = small_counts.take(10)
small_counts_take_10
```

[175]:
```
[(1, 27), (2, 11), (3, 3), (4, 2)]
```

The cell above should produce this output:

```
[(1, 27), (2, 11), (3, 3), (4, 2)]
```

This means there are 27 1-word noun phrases, 11 2-word noun phrases, 3 3-word noun phrases, and 2 4-word noun phrases in the `small_text` data set.

[176]:
```
# Autograder cell. This cell is worth 2 points (out of 20). This cell does not␣
 ↪contain hidden tests.
# This cell deliberately includes answers to provide guidance on how this␣
 ↪question is graded.

correct = AutograderHelper.parse_spark_take([(1, 27), (2, 11), (3, 3), (4, 2)])
submitted = AutograderHelper.parse_spark_take(small_counts_take_10)

AutograderHelper.assert_same_shape(correct=correct, submitted=submitted)
AutograderHelper.assert_same_rows(correct=correct, submitted=submitted)
AutograderHelper.assert_same_columns(correct=correct, submitted=submitted)
```

Now let's run it against the larger data set. *The complete analysis could take about 10 minutes to run.*

[177]:
```
text = sc.textFile("../../assets/data/nytimes/nytimes_news_articles.txt")

counts = noun_phrase_length_distribution(text)
```

[178]:
```
counts_take_10 = counts.take(10)
counts_take_10
```

[178]:
```
[(1, 1205976),
 (2, 353457),
 (3, 119065),
 (4, 35890),
 (5, 11079),
 (6, 3889),
 (7, 1400),
 (8, 543),
 (9, 257),
 (10, 112)]
```

[179]:
```
assert counts_take_10[0] == (1, 1205976), "The first item in the result is not␣
↪correct."
```

[180]:
```
# Autograder cell. This cell is worth 8 points (out of 20). This cell contains␣
↪hidden tests.
```

[ ]:
```

```