4_medication_analysis

August 1, 2023

[1]:
```python
import json
import os
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import numpy as np
from drug_named_entity_recognition import find_drugs
import json5
import sys

parent_dir = os.path.abspath("..")
if parent_dir not in sys.path:
    sys.path.append(parent_dir)
from path import DATA_PROCESSED_DOCUMENTS_DIR
```

[2]:
```python
chief_complaint = "fever"
folder_location = os.path.join(
    DATA_PROCESSED_DOCUMENTS_DIR / chief_complaint / "black-or-african-american"
)
b_docs = []
w_docs = []
for filename in os.listdir(folder_location):
    file_location = os.path.join(folder_location, filename)
    if os.path.isfile(file_location):
        with open(file_location) as d:
            try:
                file_contents = d.read()
                content = json.loads(file_contents)
                b_docs.append(content)
            except Exception as e:
                try:
                    # pull of first and last line, gpt sometimes response with␣
 ↪a leading ```json and ends with ```
                    tmp = file_contents.splitlines(True)
                    while "{" not in tmp[0]:
                        tmp = tmp[1:]
                    while "}" not in tmp[-1]:
                        tmp = tmp[:-1]
```

```
                    tmp = "".join(tmp)
                    content = json5.loads(tmp)
                    b_docs.append(content)
                except Exception as e:
                    # print(f"{file_location} Error: {e}")
                    pass

folder_location = os.path.join(
    DATA_PROCESSED_DOCUMENTS_DIR / chief_complaint / "white-or-caucasian"
)
for filename in os.listdir(folder_location):
    file_location = os.path.join(folder_location, filename)
    if os.path.isfile(file_location):
        with open(file_location) as d:
            try:
                file_contents = d.read()
                content = json.loads(file_contents)
                w_docs.append(content)
            except Exception as e:
                try:
                    # pull of first and last line, gpt sometimes response with␣
 ↪a leading ```json and ends with ```
                    tmp = file_contents.splitlines(True)
                    while "{" not in tmp[0]:
                        tmp = tmp[1:]
                    while "}" not in tmp[-1]:
                        tmp = tmp[:-1]
                    tmp = "".join(tmp)
                    content = json5.loads(tmp)
                    w_docs.append(content)
                except Exception as e:
                    # print(f"{file_location} Error: {e}")
                    pass
```

[3]:
```
print(len(b_docs))
print(len(w_docs))
```

```
4955
4962
```

[4]:
```
b_normalized_medications = []
for doc in b_docs:
    if doc.get("medications") is not None:
        res = []
        res = doc.get("medications").split(" ")
        try:
            res.remove("other")
```

```
        except ValueError:
            pass
        res = find_drugs(res, is_ignore_case=True)
        b_normalized_medications.append(res)
len(b_normalized_medications)
```

[4]: 4898

[5]:
```
b_normalized_medications[:2]
```

[5]: [[], []]

[6]:
```
w_normalized_medications = []
for doc in w_docs:
    if doc.get("medications") is not None:
        res = []
        res = doc.get("medications").split(" ")
        try:
            res.remove("other")
        except ValueError:
            pass
        res = find_drugs(res, is_ignore_case=True)
        w_normalized_medications.append(res)
len(w_normalized_medications)
```

[6]: 4912

[7]:
```
# For each patient, parse out the medications and normalize them. De-dup them␣
 ↪so each patient has each medication listed only once.
b_just_names = list(
    map(
        lambda n: set(list(map(lambda m: m[0].get("name"), n))),
        b_normalized_medications,
    )
)
b_normalized_medications_names = [
    element for sublist in b_just_names for element in sublist
]
w_just_names = list(
    map(
        lambda n: set(list(map(lambda m: m[0].get("name"), n))),
        w_normalized_medications,
    )
)
w_normalized_medications_names = [
    element for sublist in w_just_names for element in sublist
]
```

```
b_just_names[:5]
# print(len(b_normalized_medications_names))
# print(len(w_normalized_medications_names))
```

```
[7]: [set(), set(), set(), set(), set()]
```

```
[8]: b_cv = CountVectorizer(analyzer="word")
b_cv_fit = b_cv.fit_transform(b_normalized_medications_names)
b_word_list = b_cv.get_feature_names_out()
b_count_list = b_cv_fit.toarray().sum(axis=0)

b_word_freq = dict(zip(b_word_list, b_count_list))



w_cv = CountVectorizer(analyzer="word")
w_cv_fit = w_cv.fit_transform(w_normalized_medications_names)
w_word_list = w_cv.get_feature_names_out()
w_count_list = w_cv_fit.toarray().sum(axis=0)

w_word_freq = dict(zip(w_word_list, w_count_list))
```

```
[9]: b_word_freq_df = pd.DataFrame(
    b_word_freq.items(), columns=["word", "b.frequency"]
).sort_values(by="b.frequency", ascending=False)
w_word_freq_df = pd.DataFrame(
    w_word_freq.items(), columns=["word", "w.frequency"]
).sort_values(by="w.frequency", ascending=False)
```

```
[10]: wf_df = w_word_freq_df.merge(b_word_freq_df, how="inner", on="word")
```

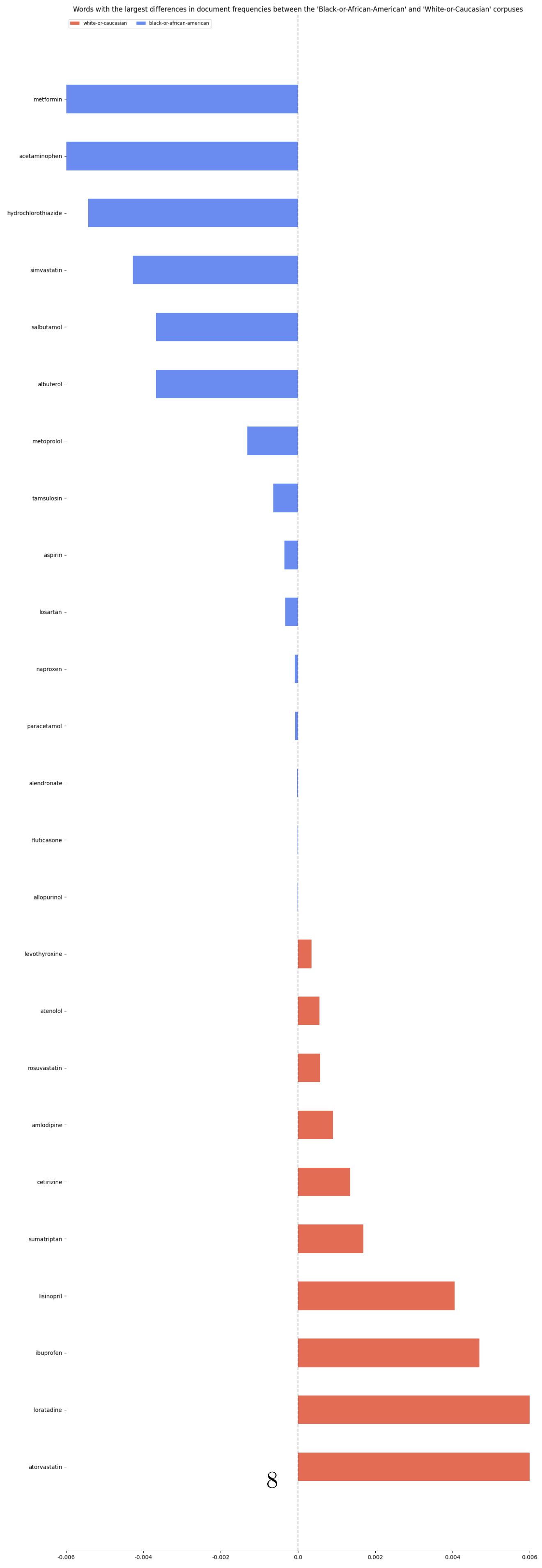
```
[11]: wf_df["w.frequency_pct"] = wf_df["w.frequency"] / wf_df["w.frequency"].sum()
wf_df["b.frequency_pct"] = wf_df["b.frequency"] / wf_df["b.frequency"].sum()
wf_df["frequency_pct_diff"] = wf_df["b.frequency_pct"] - wf_df["w.
  ↪frequency_pct"]
wf_df["frequency_pct_diff_abs"] = wf_df["frequency_pct_diff"].abs()
# Sort by largest values in absolue difference
wf_df.sort_values(by="frequency_pct_diff", ascending=False).head(10)
```

```
[11]:                   word  w.frequency  b.frequency  w.frequency_pct  \
4             metformin          201          277         0.058910
2         acetaminophen          426          434         0.124853
13  hydrochlorothiazide           19           36         0.005569
7            simvastatin           99          109         0.029015
9             albuterol           75           84         0.021981
10           salbutamol           75           84         0.021981
8            metoprolol           81           82         0.023740
21           tamsulosin            2            4         0.000586
```

```
19          aspirin          4            5          0.001172
11         losartan         27           27          0.007913

    b.frequency_pct  frequency_pct_diff  frequency_pct_diff_abs
4          0.084606            0.025696                0.025696
2          0.132560            0.007706                0.007706
13         0.010996            0.005427                0.005427
7          0.033293            0.004277                0.004277
9          0.025657            0.003675                0.003675
10         0.025657            0.003675                0.003675
8          0.025046            0.001306                0.001306
21         0.001222            0.000636                0.000636
19         0.001527            0.000355                0.000355
11         0.008247            0.000334                0.000334
```

[12]:
```
# First order frequencies by magnature of difference (absolute value), take the␣
 ↪top 200 words with the greatest difference,
# then re-sort by actual difference so when we plot the values will be␣
 ↪sequential from smallest to largest bars
most = (
    wf_df.sort_values(by="frequency_pct_diff_abs", ascending=False)
    .head(200)
    .sort_values(by="frequency_pct_diff", ascending=False)
)

chart_data = {}

# Create a map with the word as the frequency, and the magnitude vector as the␣
 ↪value\
# a vector of [0, n] will plot a blue bar
# a vector of [n, 0] will plot an orange bar
# a vector with a negative n [-n, 0] will plot a bar on the left
# a vector with a positive n [n, 0] will plot a bar on the right
# {"word": [-1, 0]} will plot an orange bar for "word" on the left of 0 with␣
 ↪length 1
# {"word": [0, 0.5]} will plot a blue bar for "word" on the right of 0 with␣
 ↪length 0.5
# in order to generate a good Positive Negative bar chart, we assign b freq to␣
 ↪the left side (negative)
# and w freq to the right side (positive)
for row in most.iterrows():
    if row[1]["w.frequency_pct"] > row[1]["b.frequency_pct"]:
        # orange bars
        chart_data[row[1]["word"]] = [
            row[1]["w.frequency_pct"] - row[1]["b.frequency_pct"],
            0,
        ]
    else:
        # blue bars
        chart_data[row[1]["word"]] = [
            0,
            -(row[1]["b.frequency_pct"] - row[1]["w.frequency_pct"]),
        ]
```

[13]:
```
# Positive Negative Bar Chart to better visualize where word frequencies␣
 ↪diverge between data sets
# Based on https://stackoverflow.com/a/69976552/11407943
import numpy as np
import matplotlib.pyplot as plt


category_names = ["white-or-caucasian", "black-or-african-american"]
results = chart_data


def survey(results, category_names):
    """
    Parameters
    ----------
    results : dict
        A mapping from question labels to a list of answers per category.
        It is assumed all lists contain the same number of entries and that
        it matches the length of *category_names*. The order is assumed
        to be from 'Strongly disagree' to 'Strongly aisagree'
    category_names : list of str
        The category labels.
    """

    labels = list(results.keys())
    data = np.array(list(results.values()))
    data_cum = data.cumsum(axis=1)
    middle_index = data.shape[1] // 2
    offsets = 0  # data[:, range(middle_index)].sum(axis=1) # + data[:,␣
 ↪middle_index]/2

    # Color Mapping
    category_colors = plt.get_cmap("coolwarm_r")(np.linspace(0.15, 0.85, data.
 ↪shape[1]))

    fig, ax = plt.subplots(figsize=(15, 50))

    # Plot Bars
    for i, (colname, color) in enumerate(zip(category_names, category_colors)):
        widths = data[:, i]
        starts = data_cum[:, i] - widths - offsets
        rects = ax.barh(
            labels, widths, left=starts, height=0.5, label=colname, color=color
        )

    # Add Zero Reference Line
    ax.axvline(0, linestyle="--", color="black", alpha=0.25)

    # X Axis
    ax.set_xlim(-0.006, 0.006)
    # ax.set_xticks(np.arange(-0.0035, 0.0035, 0.003))
    ax.xaxis.set_major_formatter(lambda x, pos: str(x))

    # Y Axis
    ax.invert_yaxis()

    # Remove spines
    ax.spines["right"].set_visible(False)
    ax.spines["top"].set_visible(False)
    ax.spines["left"].set_visible(False)

    # Ledgend
    ax.legend(
        ncol=len(category_names),
        bbox_to_anchor=(0, 0.99),
        loc="lower left",
        fontsize="small",
    )

    # Set Background Color
    fig.set_facecolor("#FFFFFF")

    return fig, ax


fig, ax = survey(results, category_names)
plt.title(
    "Words with the largest differences in document frequencies between the␣
 ↪'Black-or-African-American' and 'White-or-Caucasian' corpuses"
)
plt.show()
```


Words with the largest differences in document frequencies between the 'Black-or-African-American' and 'White-or-Caucasian' corpuses
white-or-caucasian
black-or-african-american
metformin
acetaminophen
hydrochlorothiazide
simvastatin
salbutamol
albuterol
metoprolol
tamsulosin
aspirin
losartan
naproxen
paracetamol
alendronate
fluticasone
allopurinol
levothyroxine
atenolol
rosuvastatin
amlodipine
cetirizine
sumatriptan
lisinopril
ibuprofen
loratadine
atorvastatin
-0.006
-0.004
-0.002
0.0
0.002
0.004
0.006

[14]:
```
import scipy
from sklearn.feature_extraction import text
from collections import Counter
```

[15]:
```
b_just_names_lower = [list(map(lambda x: x.lower(), arr)) for arr in␣
 ↪b_just_names]
b_list_of_doc_counter = list(map(Counter, b_just_names_lower))
# element for sublist in w_just_names for element in sublist
w_just_names_lower = [list(map(lambda x: x.lower(), arr)) for arr in␣
 ↪w_just_names]
w_list_of_doc_counter = list(map(Counter, w_just_names_lower))
b_just_names_lower
b_medications_names_counter = Counter(
    [element for sublist in b_just_names_lower for element in sublist]
)
w_medications_names_counter = Counter(
    [element for sublist in w_just_names_lower for element in sublist]
)
```

[16]:
```
b_medications_names_counter
```

[16]:
```
Counter({'lisinopril': 1119,
         'atorvastatin': 493,
         'acetaminophen': 434,
         'metformin': 277,
         'amlodipine': 235,
         'ibuprofen': 142,
         'simvastatin': 109,
         'loratadine': 90,
         'albuterol': 84,
         'salbutamol': 84,
         'metoprolol': 82,
         'hydrochlorothiazide': 36,
         'losartan': 27,
         'levothyroxine': 19,
         'cetirizine': 9,
         'naproxen': 7,
         'paracetamol': 6,
         'aspirin': 5,
         'sumatriptan': 5,
         'tamsulosin': 4,
         'alendronate': 2,
         'atenolol': 2,
         'fluticasone': 1,
         'allopurinol': 1,
         'lansoprazole': 1,
         'glipizide': 1,
         'mefloquine': 1,
         'rosuvastatin': 1,
         'chlorambucil': 1,
         'furosemide': 1})
```

[17]:
```
w_medications_names_counter
```

[17]:
```
Counter({'lisinopril': 1180,
          'atorvastatin': 612,
          'acetaminophen': 426,
          'amlodipine': 248,
          'metformin': 201,
          'ibuprofen': 164,
          'loratadine': 129,
          'simvastatin': 99,
          'metoprolol': 81,
          'albuterol': 75,
          'salbutamol': 75,
          'losartan': 27,
          'levothyroxine': 21,
          'hydrochlorothiazide': 19,
          'cetirizine': 14,
          'sumatriptan': 11,
          'naproxen': 7,
          'paracetamol': 6,
          'atenolol': 4,
          'aspirin': 4,
          'rosuvastatin': 3,
          'enalapril': 2,
          'omeprazole': 2,
          'alendronate': 2,
          'tamsulosin': 2,
          'fexofenadine': 1,
          'montelukast': 1,
          'allopurinol': 1,
          'fluticasone': 1,
          'amiloride': 1,
          'latanoprost': 1})
```

[18]:
```
total_keys = list(
    set(
        list(w_medications_names_counter.keys())
        + list(b_medications_names_counter.keys())
    )
)
new_counts = {}
aa = []
ca = []
for k in total_keys:
    # [aa,ca]
    new_counts[k] = [
        b_medications_names_counter.get(k, 0),
        w_medications_names_counter.get(k, 0),
    ]
    aa.append(b_medications_names_counter.get(k, 0))
    ca.append(w_medications_names_counter.get(k, 0))

c_table = pd.DataFrame.from_dict(new_counts)
c_table.rename(index={0: "b.freq"}, inplace=True)
c_table.rename(index={1: "w.freq"}, inplace=True)
c_table
```

[18]:
```
        paracetamol  lisinopril  mefloquine  ibuprofen  lansoprazole  \
b.freq            6        1119           1        142             1
w.freq            6        1180           0        164             0

        furosemide  alendronate  atorvastatin  amlodipine  allopurinol  …  \
b.freq           1            2           493         235            1  …
w.freq           0            2           612         248            1  …

        enalapril  latanoprost  levothyroxine  chlorambucil  acetaminophen  \
b.freq          0            0             19             1            434
w.freq          2            1             21             0            426

        cetirizine  fexofenadine  metformin  metoprolol  naproxen
b.freq           9             0        277          82         7
w.freq          14             1        201          81         7

[2 rows x 36 columns]
```

[19]:
```
class bcolors:
    HEADER = "\033[95m"
    OKBLUE = "\033[94m"
    OKCYAN = "\033[96m"
    OKGREEN = "\033[92m"
    WARNING = "\033[93m"
    FAIL = "\033[91m"
    ENDC = "\033[0m"
    BOLD = "\033[1m"
    UNDERLINE = "\033[4m"
```

[20]:
```
sig_results = []
# Chi square independence test
# https://www.dir.uniupo.it/pluginfile.php/138296/mod_resource/content/0/
 ↪22-colloc-bw.pdf
for k in list(set(total_keys)):
    # For AA [Number of instances of current word, Number of instances of all␣
 ↪other words]
    x1 = [c_table[k].iloc[0], c_table.iloc[0].sum() - c_table[k].iloc[0]]
    # For CA [Number of instances of current word, Number of instances of all␣
 ↪other words]
    y1 = [c_table[k].iloc[1], c_table.iloc[1].sum() - c_table[k].iloc[1]]
    test = scipy.stats.chi2_contingency([x1, y1])
    word = c_table[k].name
    if test.pvalue < 0.05:
        sig_results.append(word)
        print(f"{bcolors.OKGREEN}{bcolors.BOLD}Medication: {k}{bcolors.ENDC}")
        print(f"AA: {x1}")
        print(f"CA: {y1}")
        print(
            f'There {bcolors.OKGREEN}is a significant difference{bcolors.ENDC}␣
 ↪in the use of medication "{word}" between the groups with a p-value of␣
 ↪{bcolors.OKGREEN +"{:0.3f}".format(test.pvalue) + bcolors.ENDC}'
        )
        print(f"")
if len(sig_results) == 0:
    print(
        f"{bcolors.BOLD}{bcolors.FAIL}No significant differences in any␣
 ↪conditions between groups found{bcolors.ENDC}"
    )
```

```
Medication: atorvastatin
AA: [493, 2786]
CA: [612, 2808]
There is a significant difference in the use of medication
"atorvastatin" between the groups with a p-value of 0.002

Medication: hydrochlorothiazide
AA: [36, 3243]
CA: [19, 3401]
There is a significant difference in the use of medication
"hydrochlorothiazide" between the groups with a p-value of 0.020

Medication: loratadine
AA: [90, 3189]
CA: [129, 3291]
There is a significant difference in the use of medication "loratadine"
between the groups with a p-value of 0.022
```

```
Medication: metformin
AA: [277, 3002]
CA: [201, 3219]
There is a significant difference in the use of medication "metformin"
between the groups with a p-value of 0.000
```