preprocessing_analysis_final

December 1, 2024

1 Portfolio Assignment 3 - Preprocessing and Sentiment Analysis

[ ]:
```
# import packages
import json
import string
import nltk
from nltk.tokenize import word_tokenize
import re
from nltk.stem import WordNetLemmatizer
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
import numpy as np
import pandas as pd
from datetime import datetime
import seaborn as sns
import matplotlib.pyplot as plt
import urllib.request
import csv
from scipy.special import softmax
```

2 Loading in data

[ ]:
```
# loading the data in for liberals
with open('reddit_data_new/democrat_comments.json') as f:
    data_lib = json.load(f)

# loading the data in for conservative
with open('reddit_data_new/republican_comments.json') as f:
    data_con = json.load(f)
```

3 Preparing the twitter-roberta-base model for sentiment analysis

[ ]:
```
# load the model in and specify its task as sentiment
task='sentiment'
MODEL = f"cardiffnlp/twitter-roberta-base-{task}"
```

```
# load huggingface's tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

# Preprocess text (username and links)
def preprocess(text):
    new_text = []

    for t in text.split(" "):
        t = '@user' if t.startswith('@') and len(t) > 1 else t
        t = 'http' if t.startswith('http') else t
        new_text.append(t)
    return " ".join(new_text)

# map the numeric labels onto negative, neutral and positive from a document
labels=[]
mapping_link = f"https://raw.githubusercontent.com/cardiffnlp/tweeteval/main/
 ↪datasets/{task}/mapping.txt"
with urllib.request.urlopen(mapping_link) as f:
    html = f.read().decode('utf-8').split("\n")
    csvreader = csv.reader(html, delimiter='\t')
labels = [row[1] for row in csvreader if len(row) > 1]

# look at labels
print("Labels:", labels)
```

4 Extract comments from the json-file, which contains comments from r/conservative

[ ]:
```
# Get comments from the "comment" column, including submission_id
def extract_comments(data_con):
    comments_with_timestamps = []
    for entry in data_con:  # Loop through each object in the JSON
        if "comment" in entry and "created" in entry and "submission_id" in␣
 ↪entry:  # Check if all keys exist
            comments_with_timestamps.append({
                "comment": entry["comment"],
                "created": entry["created"],  # Extract the created timestamp
                "submission_id": entry["submission_id"]  # Extract the␣
 ↪submission ID
            })
    return comments_with_timestamps

# Use the function
text = extract_comments(data_con)
```

```
# Let's see what it looks like
for comment in text:
    print(comment)
```

5 Sentiment Analysis for conservatives

[ ]:
```
# Empty list to store the results
results = []

# Loop through each comment
for entry in text:  # Assuming text contains dictionaries with "comment",␣
 ↪"created", and "submission_id"
    comment = entry.get("comment")  # Extract the comment
    created = entry.get("created")  # Extract the created timestamp
    submission_id = entry.get("submission_id")  # Extract the submission ID

    if comment and isinstance(comment, str) and comment.strip():
        preprocessed_comment = preprocess(comment)

        # Tokenize and prepare input
        encoded_input = tokenizer(preprocessed_comment, return_tensors='pt',␣
 ↪truncation=True, max_length=512)
        output = model(**encoded_input)

        # Get the logits and apply softmax
        scores = output[0][0].detach().numpy()
        scores = softmax(scores)

        # Find the top sentiment
        ranking = np.argsort(scores)[::-1]
        top_sentiment = labels[ranking[0]]
        top_score = np.round(float(scores[ranking[0]]), 4)

        # Save the result
        results.append({
            "Comment": comment,
            "Created": created,  # Add the original timestamp from the JSON
            "Submission ID": submission_id,  # Add the submission ID
            "Sentiment": top_sentiment,
            "Score": top_score,
            "Negative Probability": np.round(float(scores[0]), 4),
            "Neutral Probability": np.round(float(scores[1]), 4),
            "Positive Probability": np.round(float(scores[2]), 4),
        })

# Add Subreddit column
```

```
for item in results:
    item["Subreddit"] = "r/Republican"

# Convert to DataFrame
df = pd.DataFrame(results)

# Check the DataFrame
print(df)

# Save to CSV
df.to_csv("republican_sentiment_checck.csv", index=False)
print("Results saved to 'republican_sentiment.csv'")
```

6 Extract comments from the json-file, which contains comments from r/liberal

[ ]:
```
# use the function
text_lib = extract_comments(data_lib)

# lets see what it looks like
for comment in text_lib:
    print(comment)
```

7 Sentiment Analysis for liberals

[ ]:
```
# Loop through each comment
for entry in text_lib:  # Assuming text_lib contains dictionaries with␣
 ↪"comment", "created", and "submission_id"
    comment = entry.get("comment")  # Extract the comment
    created = entry.get("created")  # Extract the created timestamp
    submission_id = entry.get("submission_id")  # Extract the submission ID

    if comment and isinstance(comment, str) and comment.strip():
        preprocessed_comment = preprocess(comment)

        # Tokenize and prepare input
        encoded_input = tokenizer(preprocessed_comment, return_tensors='pt',␣
 ↪truncation=True, max_length=512)
        output = model(**encoded_input)

        # Get the logits and apply softmax
        scores = output[0][0].detach().numpy()
        scores = softmax(scores)

        # Find the top sentiment
```

```
        ranking = np.argsort(scores)[::-1]
        top_sentiment = labels[ranking[0]]
        top_score = np.round(float(scores[ranking[0]]), 4)

        # Save the result
        results.append({
            "Comment": comment,
            "Created": created,  # Add the original timestamp from the JSON
            "Submission ID": submission_id,  # Add the submission ID
            "Sentiment": top_sentiment,  # Properly assigned from labels
            "Score": top_score,
            "Negative Probability": np.round(float(scores[0]), 4),
            "Neutral Probability": np.round(float(scores[1]), 4),
            "Positive Probability": np.round(float(scores[2]), 4),
        })

# Add Subreddit column
for item in results:
    item["Subreddit"] = "r/Democrats"

# Convert to DataFrame
df = pd.DataFrame(results)

# Check the DataFrame
print(df)

# Save to CSV
df.to_csv("democrat_sentiment_checckk.csv", index=False)
print("Results saved to 'democrat_sentiment.csv'")
```

8 load the data in again

[ ]:
```
dfc = pd.read_csv('republican_sentiment_checck.csv')
dfl = pd.read_csv('democrat_sentiment_checckk.csv')
```

9 Removing bot comments

[ ]:
```
# Remove rows where the 'Comment' contains "I am a bot"
dfc = dfc[~dfc['Comment'].str.contains("I am a bot", na=False)]
dfl = dfl[~dfl['Comment'].str.contains("I am a bot", na=False)]
```

10 Proportion of negative and positive comments before aggregating

[ ]:
```
colours = ['red', 'gray', 'green']

# Calculate sentiment proportions
sentiment_counts = dfc.groupby(["Subreddit", "Sentiment"]).size().
 ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",␣
 ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/Republican")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1)  # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

[ ]:
```
# Calculate sentiment proportions
sentiment_counts = dfl.groupby(["Subreddit", "Sentiment"]).size().
 ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",␣
 ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/Democrats")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1)  # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

11 Conservative line chart

[ ]:
```
colors = ['green', 'red', 'blue']


#  datetime format
dfc["Created"] = pd.to_datetime(dfc["Created"])  # Use 'Created' from JSON
dfc["Date"] = dfc["Created"].dt.date  # Extract date for grouping
```

```
# filter data for the specific date range
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfc[(dfc["Date"] >= start_date) & (dfc["Date"] <= end_date)]

#  x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot it
plt.figure(figsize=(12, 6))
sns.lineplot(data=filtered_dfl, x="Date", y="Score", hue="Sentiment",␣
 ↪style="Subreddit", markers=True, palette = colors)
plt.title("Republican Sentiment between 6/11-18/11")
plt.xlabel("Date")
plt.ylabel("Sentiment Score")

# Set x-axis ticks
plt.xticks(date_range, rotation=45)

# Set the x-axis limits explicitly
plt.xlim(start_date, end_date)

plt.tight_layout()
plt.legend(title="Sentiment")
plt.show()
```

[ ]:
```
from matplotlib import rcParams

# change font
rcParams['font.family'] = 'Georgia'

# Define colors for sentiment categories
colors = {"positive": "green", "negative": "red", "neutral": "blue"}

# datetime
dfc["Created"] = pd.to_datetime(dfc["Created"])  # Use 'Created' from JSON
dfc["Date"] = dfc["Created"].dt.date  # Extract date for grouping

# Filter
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfc[(dfc["Date"] >= start_date) & (dfc["Date"] <= end_date)]

# x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot sentiment scores
```

```
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=filtered_dfl,
    x="Date",
    y="Score",
    hue="Sentiment",
    style="Subreddit",
    markers=True,
    palette=colors  # Use the specified color palette
)

# change titles
plt.title("Republican Sentiment between 6/11-18/11", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sentiment Score", fontsize=12)

# Set x-axis ticks to every date in the range
plt.xticks(date_range, rotation=45)

# Set the x-axis limits explicitly
plt.xlim(start_date, end_date)

# Remove unnecessary stuff
handles, labels = plt.gca().get_legend_handles_labels()
filtered_legend = [(h, l) for h, l in zip(handles, labels) if l in colors.
 ↪keys()]
plt.legend(*zip(*filtered_legend), title="Sentiment")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

[ ]:
```
from matplotlib import rcParams

# change font
rcParams['font.family'] = 'Georgia'

# Define colors
colors = {"positive": "green", "negative": "red", "neutral": "blue"}

# datetime
dfc["Created"] = pd.to_datetime(dfc["Created"])  # Use 'Created' from JSON
dfc["Date"] = dfc["Created"].dt.date  # Extract date for grouping

# Filter
```

```
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfc[(dfc["Date"] >= start_date) & (dfc["Date"] <= end_date)]

# x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot the daily sentiment scores
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=filtered_dfl,
    x="Date",
    y="Score",
    hue="Sentiment",
    style="Subreddit",
    markers=True,
    palette=colors  # Use the specified color palette
)

# change titles
plt.title("Republican Sentiment between 6/11-18/11", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sentiment Score", fontsize=12)

# Set x-axis ticks to every date in the range
plt.xticks(date_range, rotation=45)

# Set the x-axis limits explicitly
plt.xlim(start_date, end_date)

# Remove unnecessary legend stuff
handles, labels = plt.gca().get_legend_handles_labels()
filtered_legend = [(h, l) for h, l in zip(handles, labels) if l in colors.
 ↪keys()]
plt.legend(*zip(*filtered_legend), title="Sentiment")

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

12 Liberal line chart

[ ]:
```
# Set font family
rcParams['font.family'] = 'Georgia'

# Define colors
colors = ['blue', 'green', 'red']

# datetime
dfl["Created"] = pd.to_datetime(dfl["Created"])  # Use 'Created' from JSON
dfl["Date"] = dfl["Created"].dt.date  # Extract date for grouping


# filter data for range
start_date = pd.to_datetime("2024-11-06").date()
end_date = pd.to_datetime("2024-11-18").date()
filtered_dfl = dfl[(dfl["Date"] >= start_date) & (dfl["Date"] <= end_date)]

#  x-axis ticks
date_range = pd.date_range(start=start_date, end=end_date, freq="D")

# Plot the filtered data
plt.figure(figsize=(12, 6))
sns.lineplot(
    data=filtered_dfl,
    x="Date",
    y="Score",
    hue="Sentiment",
    style="Subreddit",
    markers=True,
    palette=colors
)

#  title and labels
plt.title("Democratic Sentiment between 6/11-18/11", fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sentiment Score", fontsize=12)


plt.xticks(date_range, rotation=45)

# Set explicit x-axis limits
plt.xlim(start_date, end_date)

#aAdjust layout
plt.tight_layout()
```

```
plt.legend(title="Sentiment")

# Lets check the plot
plt.show()
```

13 lets aggregate the data

[ ]:
```
# Aggregating data by 'Submission_ID'
def most_common_sentiment(series):
    # Get the most common sentiment
    if len(series) > 1:
        return series.value_counts().idxmax()
    else:
        return series.iloc[0]

# aggregatomg
dfc = dfc.groupby('Submission ID').agg(
    Score=('Score', 'mean'),  # mean for each Submission_ID
    Subreddit=('Subreddit', 'first'),  # Take the first Subreddit
    Sentiment=('Sentiment', most_common_sentiment),  # Take the first encoded␣
 ↪value
    Created=('Created', 'first')  # Take the first created timestamp
).reset_index()

# View the aggregated data
print(dfc)
```

[ ]:
```
# Perform the aggregation
dfl = dfl.groupby('Submission ID').agg(
    Score=('Score', 'mean'),  # mean for each Submission_ID
    Subreddit=('Subreddit', 'first'),  # Take the first Subreddit
    Sentiment=('Sentiment', most_common_sentiment),  # Take the first encoded␣
 ↪value
    Created=('Created', 'first')  # Take the first created timestamp
).reset_index()
# View the aggregated data
print(dfl)
```

14 Proportion of negative, netural and positive comments for republicans

[ ]:
```
colours = ['red', 'gray', 'green']

# Calculate sentiment proportions
sentiment_counts = dfc.groupby(["Subreddit", "Sentiment"]).size().
 ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",␣
 ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/Republican")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1)  # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

15 Proportion of negative, netural and positive comments for democrats

[ ]:
```
# Calculate sentiment proportions
sentiment_counts = dfl.groupby(["Subreddit", "Sentiment"]).size().
 ↪reset_index(name="Count")
total_counts = sentiment_counts.groupby("Subreddit")["Count"].transform("sum")
sentiment_counts["Proportion"] = sentiment_counts["Count"] / total_counts

# Plot proportions
sns.barplot(data=sentiment_counts, x="Subreddit", y="Proportion",␣
 ↪hue="Sentiment", palette=colours)
plt.title("Sentiment in r/democrats")
plt.xlabel("Subreddit")
plt.ylabel("Proportion")
plt.ylim(0, 1)  # Set the y-axis range from 0 to 1
plt.legend(title="Sentiment")
plt.show()
```

16 Save the data

[ ]:
```
# get the necessary columns
data_rep = dfc[['Score', 'Subreddit', 'Sentiment', 'Submission ID']]
data_dem = dfl[['Score', 'Subreddit', 'Sentiment', 'Submission ID']]

# lets make them into 0 & 1
data_rep['Subreddit_encoded'] = 0  # Republican encoded as 0
data_dem['Subreddit_encoded'] = 1  # Democrat encoded as 1
```

[ ]:
```
# save
combined_df.to_csv("combined_df.csv", index=False)

combined_df
```