code

March 25, 2023

[ ]:
```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import re, nltk, spacy, gensim
import os
from nltk.corpus import stopwords



# lda
import lda

# Gsdmm
from gsdmm import MovieGroupProcess


# Sklearn
from sklearn.decomposition import LatentDirichletAllocation, TruncatedSVD
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.preprocessing import normalize

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from pprint import pprint
```

[2]:
```
nlp = spacy.load('en_core_web_sm')
```

[3]:
```
def preprocessing():
    directory = 'comments1k'
    comments = []
    filenames = []
```

```
    for filename in os.listdir(directory):
        if filename.endswith(".txt"):
            with open(os.path.join(directory, filename)) as file:
                comments.append(file.read().strip())
                filenames.append(filename)
    df = pd.DataFrame({'Filename': filenames, 'comments': comments})

    df.head()
    df['comments'] = df['comments'].str.replace('&\w+;'," ")
    df['comments'] = df['comments'].apply(lambda x: re.sub('<.*?>','', x))

    df['comments'] = df['comments'].str.lower()

    df['comments'] = df['comments'].str.replace('[^\w\s]',' ')
    # Load the stop words from NLTK
    nltk.download('stopwords')
    stop_words = set(stopwords.words('english'))

    df['comments'] = df['comments'].apply(lambda x: " ".join([w for w in x.
 ↪split() if w not in stop_words]))

    return df
```

[4]:
```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))
```

[5]:
```
# create N-grams
def make_n_grams(texts):
    bigram = gensim.models.Phrases(texts, min_count=5, threshold=100)  # higher␣
 ↪threshold fewer phrases.
    bigram_mod = gensim.models.phrases.Phraser(bigram)
    trigram = gensim.models.Phrases(bigram[texts], threshold=100)
    trigram_mod = gensim.models.phrases.Phraser(trigram)
    bigrams_text = [bigram_mod[doc] for doc in texts]
    trigrams_text =  [trigram_mod[bigram_mod[doc]] for doc in bigrams_text]
    return trigrams_text
```

[6]:
```
def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in␣
 ↪allowed_postags])
    return texts_out
```

[7]:
```
def matrix_1(df):
    docs = df['comments'].values.tolist()
    vectorizer = CountVectorizer()
    bag_of_words = vectorizer.fit_transform(docs)
    x = bag_of_words.toarray()
    return x
```

[8]:
```
def create_vocab(df):
    docs = df['comments'].values.tolist()
    vectorizer = CountVectorizer()
    new_vocab = vectorizer.fit(docs).get_feature_names_out()
    return new_vocab
```

[9]:
```
def lda(df):
    import lda
    df1 = df
    matrix = matrix_1(df)
    vocab = create_vocab(df)
    titles = df['comments'].tolist()
    model = lda.LDA(n_topics=10, n_iter=1500, random_state=1)
    model.fit(matrix)
    topic_word = model.components_
    n_top_words = 8
    for i, topic_dist in enumerate(topic_word):
        topic_words = np.array(vocab)[np.argsort(topic_dist)][:-(n_top_words+1):
↪-1]
        print('Topic {}: {}'.format(i, ','.join(topic_words)))
    doc_topic = model.doc_topic_
    data = []
    docs = df.shape[0]
    for i in range(docs):
        filename = df1['Filename'][i]
        title = titles[i]
        top_topic = doc_topic[i].argmax()
        data.append({'filename': filename,'Title': title, 'Top Topic':␣
↪top_topic})
        final_df= pd.DataFrame(data)
    return final_df
```

[10]:
```
def gsdmm(df, reviews_lemmatized):
    n_topics = 10
    mgp = MovieGroupProcess(K=n_topics, alpha=0.01, beta=0.01, n_iters=2)

    vocab = set(x for review in reviews_lemmatized for x in review)
    n_terms = len(vocab)
    model = mgp.fit(reviews_lemmatized, n_terms)
```

```
    return mgp
```

```
[11]: def top_words(cluster_word_distribution, top_cluster, values):
          for cluster in top_cluster:
              sort_dicts =sorted(mgp.cluster_word_distribution[cluster].items(),␣
      ↪key=lambda k: k[1], reverse=True)[:values]
              print("\n Topic %s : %s"%(cluster,sort_dicts))
```

```
[12]: def create_topics_dataframe(df,data_text,mgp, threshold, topic_dict,lemma_text):
          result = pd.DataFrame(columns=['Filename','comments', 'Topic', 'Lemma-text'])
          for i, text in enumerate(data_text):
              result.at[i,'comments'] = df.comments[i]
              result.at[i, 'Filename'] = df.Filename[i]
              result.at[i, 'Lemma-text'] = lemma_text[i]
              prob = mgp.choose_best_label(lemma_text[i])
              if prob[1] >= threshold:
                  result.at[i, 'Topic'] = topic_dict[prob[0]]
              else:
                  result.at[i, 'Topic'] = 'Other'
          return result
```

Question 1.1 - Use Latent Dirichlet Allocation (LDA) method to discover latent topics in the dataset with the number of topics as 10. Output the top 8 words for each topic. For the document "0_9.txt" and "1_7.txt", what topics are assigned to them? Do they make sense?

```
[13]: df = preprocessing()
      X = lda(df)
      X.to_csv('output/lda.csv', index=False)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\saima_x4lzx52\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
INFO:lda:n_documents: 996
INFO:lda:vocab_size: 15901
INFO:lda:n_words: 121324
INFO:lda:n_topics: 10
INFO:lda:n_iter: 1500
INFO:lda:<0> log likelihood: -1439176
INFO:lda:<10> log likelihood: -1163684
INFO:lda:<20> log likelihood: -1138386
INFO:lda:<30> log likelihood: -1125362
INFO:lda:<40> log likelihood: -1116881
INFO:lda:<50> log likelihood: -1111641
INFO:lda:<60> log likelihood: -1107484
INFO:lda:<70> log likelihood: -1103336
INFO:lda:<80> log likelihood: -1100923
INFO:lda:<90> log likelihood: -1097638
INFO:lda:<100> log likelihood: -1095897
```

```
INFO:lda:<110> log likelihood: -1093709
INFO:lda:<120> log likelihood: -1091735
INFO:lda:<130> log likelihood: -1090065
INFO:lda:<140> log likelihood: -1087694
INFO:lda:<150> log likelihood: -1087355
INFO:lda:<160> log likelihood: -1085475
INFO:lda:<170> log likelihood: -1083549
INFO:lda:<180> log likelihood: -1082803
INFO:lda:<190> log likelihood: -1081206
INFO:lda:<200> log likelihood: -1079917
INFO:lda:<210> log likelihood: -1079104
INFO:lda:<220> log likelihood: -1078540
INFO:lda:<230> log likelihood: -1076797
INFO:lda:<240> log likelihood: -1076738
INFO:lda:<250> log likelihood: -1075786
INFO:lda:<260> log likelihood: -1075016
INFO:lda:<270> log likelihood: -1074016
INFO:lda:<280> log likelihood: -1074067
INFO:lda:<290> log likelihood: -1073245
INFO:lda:<300> log likelihood: -1072094
INFO:lda:<310> log likelihood: -1072620
INFO:lda:<320> log likelihood: -1071517
INFO:lda:<330> log likelihood: -1071474
INFO:lda:<340> log likelihood: -1071982
INFO:lda:<350> log likelihood: -1071262
INFO:lda:<360> log likelihood: -1071067
INFO:lda:<370> log likelihood: -1070996
INFO:lda:<380> log likelihood: -1071421
INFO:lda:<390> log likelihood: -1070239
INFO:lda:<400> log likelihood: -1070005
INFO:lda:<410> log likelihood: -1069335
INFO:lda:<420> log likelihood: -1069678
INFO:lda:<430> log likelihood: -1068764
INFO:lda:<440> log likelihood: -1068907
INFO:lda:<450> log likelihood: -1068737
INFO:lda:<460> log likelihood: -1068766
INFO:lda:<470> log likelihood: -1068908
INFO:lda:<480> log likelihood: -1068408
INFO:lda:<490> log likelihood: -1067698
INFO:lda:<500> log likelihood: -1067212
INFO:lda:<510> log likelihood: -1066807
INFO:lda:<520> log likelihood: -1066101
INFO:lda:<530> log likelihood: -1066276
INFO:lda:<540> log likelihood: -1065602
INFO:lda:<550> log likelihood: -1066296
INFO:lda:<560> log likelihood: -1066151
INFO:lda:<570> log likelihood: -1066265
INFO:lda:<580> log likelihood: -1065822
```

```
INFO:lda:<590> log likelihood: -1065354
INFO:lda:<600> log likelihood: -1065673
INFO:lda:<610> log likelihood: -1065428
INFO:lda:<620> log likelihood: -1065661
INFO:lda:<630> log likelihood: -1065510
INFO:lda:<640> log likelihood: -1066067
INFO:lda:<650> log likelihood: -1065222
INFO:lda:<660> log likelihood: -1064638
INFO:lda:<670> log likelihood: -1065069
INFO:lda:<680> log likelihood: -1064106
INFO:lda:<690> log likelihood: -1063943
INFO:lda:<700> log likelihood: -1064723
INFO:lda:<710> log likelihood: -1063575
INFO:lda:<720> log likelihood: -1063417
INFO:lda:<730> log likelihood: -1063683
INFO:lda:<740> log likelihood: -1063523
INFO:lda:<750> log likelihood: -1063071
INFO:lda:<760> log likelihood: -1062974
INFO:lda:<770> log likelihood: -1063114
INFO:lda:<780> log likelihood: -1063109
INFO:lda:<790> log likelihood: -1062822
INFO:lda:<800> log likelihood: -1062302
INFO:lda:<810> log likelihood: -1062977
INFO:lda:<820> log likelihood: -1062589
INFO:lda:<830> log likelihood: -1062948
INFO:lda:<840> log likelihood: -1062610
INFO:lda:<850> log likelihood: -1062163
INFO:lda:<860> log likelihood: -1062319
INFO:lda:<870> log likelihood: -1062275
INFO:lda:<880> log likelihood: -1062499
INFO:lda:<890> log likelihood: -1062112
INFO:lda:<900> log likelihood: -1062031
INFO:lda:<910> log likelihood: -1061938
INFO:lda:<920> log likelihood: -1062485
INFO:lda:<930> log likelihood: -1062448
INFO:lda:<940> log likelihood: -1061701
INFO:lda:<950> log likelihood: -1062131
INFO:lda:<960> log likelihood: -1061754
INFO:lda:<970> log likelihood: -1062017
INFO:lda:<980> log likelihood: -1062236
INFO:lda:<990> log likelihood: -1061925
INFO:lda:<1000> log likelihood: -1062443
INFO:lda:<1010> log likelihood: -1062213
INFO:lda:<1020> log likelihood: -1061624
INFO:lda:<1030> log likelihood: -1062160
INFO:lda:<1040> log likelihood: -1061052
INFO:lda:<1050> log likelihood: -1061532
INFO:lda:<1060> log likelihood: -1062170
```

```
INFO:lda:<1070> log likelihood: -1061889
INFO:lda:<1080> log likelihood: -1061743
INFO:lda:<1090> log likelihood: -1061517
INFO:lda:<1100> log likelihood: -1061235
INFO:lda:<1110> log likelihood: -1061681
INFO:lda:<1120> log likelihood: -1062032
INFO:lda:<1130> log likelihood: -1061347
INFO:lda:<1140> log likelihood: -1060862
INFO:lda:<1150> log likelihood: -1060986
INFO:lda:<1160> log likelihood: -1061145
INFO:lda:<1170> log likelihood: -1060742
INFO:lda:<1180> log likelihood: -1060640
INFO:lda:<1190> log likelihood: -1060935
INFO:lda:<1200> log likelihood: -1060756
INFO:lda:<1210> log likelihood: -1061058
INFO:lda:<1220> log likelihood: -1061034
INFO:lda:<1230> log likelihood: -1061238
INFO:lda:<1240> log likelihood: -1060856
INFO:lda:<1250> log likelihood: -1060659
INFO:lda:<1260> log likelihood: -1060128
INFO:lda:<1270> log likelihood: -1060107
INFO:lda:<1280> log likelihood: -1061199
INFO:lda:<1290> log likelihood: -1060200
INFO:lda:<1300> log likelihood: -1060742
INFO:lda:<1310> log likelihood: -1060856
INFO:lda:<1320> log likelihood: -1060910
INFO:lda:<1330> log likelihood: -1060407
INFO:lda:<1340> log likelihood: -1060358
INFO:lda:<1350> log likelihood: -1060188
INFO:lda:<1360> log likelihood: -1060396
INFO:lda:<1370> log likelihood: -1060683
INFO:lda:<1380> log likelihood: -1060273
INFO:lda:<1390> log likelihood: -1060025
INFO:lda:<1400> log likelihood: -1060458
INFO:lda:<1410> log likelihood: -1060496
INFO:lda:<1420> log likelihood: -1060336
INFO:lda:<1430> log likelihood: -1060562
INFO:lda:<1440> log likelihood: -1060103
INFO:lda:<1450> log likelihood: -1060521
INFO:lda:<1460> log likelihood: -1059861
INFO:lda:<1470> log likelihood: -1059968
INFO:lda:<1480> log likelihood: -1060543
INFO:lda:<1490> log likelihood: -1060326
INFO:lda:<1499> log likelihood: -1059940

Topic 0: brosnan,man,david,robert,life,brother,river,fantasy
Topic 1: stewart,jeff,ned,james,gannon,kelly,western,john
Topic 2: film,one,story,two,life,man,well,character
```

```
Topic 3: war,world,young,miike,yokai,kids,film,school
Topic 4: game,carla,chess,paul,french,luzhin,alexandre,read
Topic 5: star,series,show,luke,wars,episode,new,battle
Topic 6: school,high,ramones,matthau,burns,rock,best,comedy
Topic 7: christmas,scrooge,one,scott,von,version,europa,trier
Topic 8: movie,one,like,good,see,film,great,really
Topic 9: davies,great,show,comedy,people,marion,star,price
```

[14]:
```
X
```

[14]:
```
       filename                                              Title  Top Topic
0       0_9.txt  bromwell high cartoon comedy ran time programs...          6
1     100_7.txt  scott bartlett offon nine minutes pure crazine...          2
2     101_8.txt  imdb lists 1972 reason sources seen including ...          8
3    102_10.txt  first heard film 20 years ago kid grade school...          8
4     103_7.txt  read comment decided watch movie first cast sp...          8
..          ...                                                ...        ...
991   997_7.txt  agree posts comedy drama leaned little much to...          8
992   998_7.txt  really interesting movie action movie comedy m...          8
993  999_10.txt  amazed movie others average 5 stars lower crap...          8
994    99_8.txt  christmas together actually came time raised j...          8
995     9_7.txt  working class romantic drama director martin r...          8

[996 rows x 3 columns]
```

[15]:
```
X.iloc[0]
```

[15]:
```
filename                                               0_9.txt
Title        bromwell high cartoon comedy ran time programs...
Top Topic                                                    6
Name: 0, dtype: object
```

[16]:
```
X.iloc[111]
```

[16]:
```
filename                                               1_7.txt
Title        like adult comedy cartoons like south park nea...
Top Topic                                                    8
Name: 111, dtype: object
```

1.1 Answer:

- Topic 0: It seems that this topic is related to movies or TV shows that feature actors like Brosnan, David, and Robert, as well as words like “life,” “brother,” and “fantasy.” This topic may be discussed in reference to fantasies involving characters played by these actors or movies or TV shows that center on familial bonds, particularly brotherhood.
- Topic 1: This topic appears to be related to western films or television programs because it contains the phrases “Stewart,” “Jeff,” “Ned,” “James,” “Gannon,” “Kelly,” and “John.” This subject could involve discussing particular western films or TV episodes, or it could explore the themes, characters, and cliches more broadly.

- Topic 2: This topic is broader and covers a variety of movies and TV shows that use the phrases "film," "story," "life,", "man,","well," and "character." Without more background, it is challenging to determine the topic's precise focus.
- Topic 3: Words like "world," "young," "miike," "yokai," "kids," "film," and "school" appear to be references to war films or TV series that feature young characters. It's possible that this subject will examine how war is portrayed in films and television shows, particularly from the viewpoint of characters who are younger.
- Topic 4 seems to be about chess because it includes words like "carla," "chess," "paul," and "french," which are game-related. It also includes nouns with athletic connotations like "Luzin" and "Alexandre."
- Topic 5 appears to be focused on the Star Wars series because it contains phrases like "Luke," "Wars," "Episode," and "Battle" that are associated with characters and stories.
- Topic 6 appears to be about high school because it has phrases like "Ramones," "Mathaw," "Burns," "Rock," and "Best" that are associated with the time period. Students are also mentioned in words like "comedy" and "marion."
- Topic 7 appears to be focused on the film A Christmas Carol because it includes phrases like "Scrooge," "One," "Scott," "Von," and "Europa." I recognize it. Character-related words like "Christmas" and "ghost" are also present.
- Topic 8 appears to be about movies because it includes terms like "1," "like," "good," and "watch" that refer to movie quality. It also contains phrases like "movie," "amazing," and "truly" that are associated with entertainment.
- Topic 9 appears to be focused on the comedy series Davies because it contains phrases like "excellent", "show", "comedy", "people", "Marion", and "star" that are associated with the program. seems like Words pertaining to the cast are also included, and so on. "Price" in B.

Question 1.2: Because of the data sparsity, short text may not provide enough context to adequately inform topic modeling.Try Biterm, GSDMM or other short text topic model for our dataset. Compare the topic modelling results with LDA, any improvement?

Using GSDMM

[17]:
```
df
```

[17]:
```
        Filename                                           comments
0        0_9.txt  bromwell high cartoon comedy ran time programs...
1      100_7.txt  scott bartlett offon nine minutes pure crazine...
2      101_8.txt  imdb lists 1972 reason sources seen including ...
3     102_10.txt  first heard film 20 years ago kid grade school...
4      103_7.txt  read comment decided watch movie first cast sp...
..           ...                                                ...
991    997_7.txt  agree posts comedy drama leaned little much to...
992    998_7.txt  really interesting movie action movie comedy m...
993   999_10.txt  amazed movie others average 5 stars lower crap...
994     99_8.txt  christmas together actually came time raised j...
```

```
995     9_7.txt  working class romantic drama director martin r...

[996 rows x 2 columns]
```

[18]:
```python
df = preprocessing()
tokens_reviews = list(sent_to_words(df['comments']))
tokens_reviews = make_n_grams(tokens_reviews)
reviews_lemmatized = lemmatization(tokens_reviews, allowed_postags=['NOUN',␣
 ↪'VERB', 'ADV'])
mgp = gsdmm(df,reviews_lemmatized)
doc_count = np.array(mgp.cluster_doc_count)
print('Number of documents per topic :', doc_count)

# topics sorted by the number of document they are allocated to
top_index = doc_count.argsort()[-10:][::-1]
print('\nMost important clusters (by number of docs inside):', top_index)
topic_dict = {}
topic_names = [1,2,3,4,5,6,7,8,9,10]
for i, topic_num in enumerate(top_index):
    topic_dict[topic_num]=topic_names[i]
# show the top 5 words in term frequency for each cluster
top_words(mgp.cluster_word_distribution, top_index, 8)
result = create_topics_dataframe(df,data_text=df.comments, mgp=mgp, threshold=0.
 ↪3, topic_dict=topic_dict, lemma_text=reviews_lemmatized)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\saima_x4lzx52\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
INFO:gensim.models.phrases:collecting all words and their counts
INFO:gensim.models.phrases:PROGRESS: at sentence #0, processed 0 words and 0
word types
INFO:gensim.models.phrases:collected 115874 token types (unigram + bigrams) from
a corpus of 120425 words and 996 sentences
INFO:gensim.models.phrases:merged Phrases<115874 vocab, min_count=5,
threshold=100, max_vocab_size=40000000>
INFO:gensim.utils:Phrases lifecycle event {'msg': 'built Phrases<115874 vocab,
min_count=5, threshold=100, max_vocab_size=40000000> in 0.15s', 'datetime':
'2023-03-24T23:11:43.122601', 'gensim': '4.3.1', 'python': '3.10.6
(tags/v3.10.6:9c7b4bd, Aug  1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.22621-SP0', 'event': 'created'}
INFO:gensim.models.phrases:exporting phrases from Phrases<115874 vocab,
min_count=5, threshold=100, max_vocab_size=40000000>
INFO:gensim.utils:FrozenPhrases lifecycle event {'msg': 'exported
FrozenPhrases<297 phrases, min_count=5, threshold=100> from Phrases<115874
vocab, min_count=5, threshold=100, max_vocab_size=40000000> in 0.20s',
'datetime': '2023-03-24T23:11:43.324772', 'gensim': '4.3.1', 'python': '3.10.6
(tags/v3.10.6:9c7b4bd, Aug  1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.22621-SP0', 'event': 'created'}
```

```
INFO:gensim.models.phrases:collecting all words and their counts
INFO:gensim.models.phrases:PROGRESS: at sentence #0, processed 0 words and 0
word types
INFO:gensim.models.phrases:collected 116120 token types (unigram + bigrams) from
a corpus of 117088 words and 996 sentences
INFO:gensim.models.phrases:merged Phrases<116120 vocab, min_count=5,
threshold=100, max_vocab_size=40000000>
INFO:gensim.utils:Phrases lifecycle event {'msg': 'built Phrases<116120 vocab,
min_count=5, threshold=100, max_vocab_size=40000000> in 0.35s', 'datetime':
'2023-03-24T23:11:43.671249', 'gensim': '4.3.1', 'python': '3.10.6
(tags/v3.10.6:9c7b4bd, Aug  1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.22621-SP0', 'event': 'created'}
INFO:gensim.models.phrases:exporting phrases from Phrases<116120 vocab,
min_count=5, threshold=100, max_vocab_size=40000000>
INFO:gensim.utils:FrozenPhrases lifecycle event {'msg': 'exported
FrozenPhrases<246 phrases, min_count=5, threshold=100> from Phrases<116120
vocab, min_count=5, threshold=100, max_vocab_size=40000000> in 0.19s',
'datetime': '2023-03-24T23:11:43.865631', 'gensim': '4.3.1', 'python': '3.10.6
(tags/v3.10.6:9c7b4bd, Aug  1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)]',
'platform': 'Windows-10-10.0.22621-SP0', 'event': 'created'}

In stage 0: transferred 857 clusters with 10 clusters populated
In stage 1: transferred 351 clusters with 10 clusters populated
Number of documents per topic : [  2   6   4  19  19   3  19   8   6 910]

Most important clusters (by number of docs inside): [9 6 4 3 7 8 1 2 5 0]

 Topic 9 : [('film', 1883), ('movie', 1685), ('see', 920), ('make', 780),
('story', 613), ('time', 602), ('character', 536), ('well', 535)]

 Topic 6 : [('movie', 58), ('see', 22), ('make', 13), ('really', 11),
('character', 11), ('horror', 10), ('actor', 9), ('love', 9)]

 Topic 4 : [('movie', 41), ('see', 21), ('show', 17), ('love', 13), ('find',
13), ('get', 11), ('year', 9), ('time', 9)]

 Topic 3 : [('get', 24), ('see', 23), ('movie', 20), ('make', 18), ('time', 17),
('film', 16), ('character', 13), ('comedy', 13)]

 Topic 7 : [('movie', 8), ('watch', 6), ('crawford', 6), ('dance', 5),
('stooge', 5), ('shemp', 5), ('make', 4), ('work', 4)]

 Topic 8 : [('movie', 13), ('film', 8), ('time', 7), ('ramone', 7), ('see', 5),
('rock_roll_high_school', 4), ('life', 3), ('watch', 3)]

 Topic 1 : [('movie', 14), ('film', 5), ('make', 5), ('get', 4), ('story', 3),
('time', 3), ('love', 3), ('thing', 3)]
```

```
Topic 2 : [('kid', 6), ('miike', 6), ('movie', 5), ('bit', 5), ('make', 5),
('life', 5), ('pretty', 5), ('time', 4)]

Topic 5 : [('think', 11), ('movie', 5), ('feel', 4), ('act', 3), ('much', 2),
('make', 2), ('character', 2), ('like', 2)]

Topic 0 : [('chess', 3), ('film', 2), ('play', 2), ('become', 2), ('scene', 1),
('love', 1), ('life', 1), ('rather', 1)]
```

[19]:

```
result
```

[19]:

```
       Filename                                           comments  Topic  \
0       0_9.txt  bromwell high cartoon comedy ran time programs...      1
1     100_7.txt  scott bartlett offon nine minutes pure crazine...  Other
2     101_8.txt  imdb lists 1972 reason sources seen including ...      1
3    102_10.txt  first heard film 20 years ago kid grade school...      1
4     103_7.txt  read comment decided watch movie first cast sp...      2
..          ...                                                ...    ...
991   997_7.txt  agree posts comedy drama leaned little much to...      1
992   998_7.txt  really interesting movie action movie comedy m...      1
993  999_10.txt  amazed movie others average 5 stars lower crap...  Other
994    99_8.txt  christmas together actually came time raised j...      1
995     9_7.txt  working class romantic drama director martin r...      1

                                            Lemma-text
0    [comedy, run, time, school, life, teacher, yea...
1    [minute, craziness, assault, psychedelic, puls...
2    [list, reason, source, see, include, program, ...
3    [first, hear, film, kid, grade, school, happen...
4    [read, comment, decide, watch, movie, first, c...
..                                                 ...
991  [agree, post, comedy, drama, lean, much, comed...
992  [really, movie, action, movie, comedy, foxx, t...
993  [movie, other, star, lower, movie, average, st...
994  [together, actually, come, time, raise, song, ...
995  [work, class, drama, director, ritt, come, yet...

[996 rows x 4 columns]
```

[20]:

```
result.iloc[0]
```

[20]:

```
Filename                                                0_9.txt
comments      bromwell high cartoon comedy ran time programs...
Topic                                                         1
Lemma-text     [comedy, run, time, school, life, teacher, yea...
Name: 0, dtype: object
```

[21]:

```
result.iloc[111]
```

```
[21]: Filename                                                  1_7.txt
      comments      like adult comedy cartoons like south park nea...
      Topic                                                         1
      Lemma-text    [adult, comedy, cartoon, park, nearly, format,...
      Name: 111, dtype: object
```

```
[22]: result.to_csv('output/gsdmm.csv', index=False)
```

Question 2.1 When there is no (enough) labelled corpus to train a machine learning based NLP model, we need to create a training text dataset as golden standard through manual annotation. Choose a text annotation tool to finish the following two text annotation tasks:

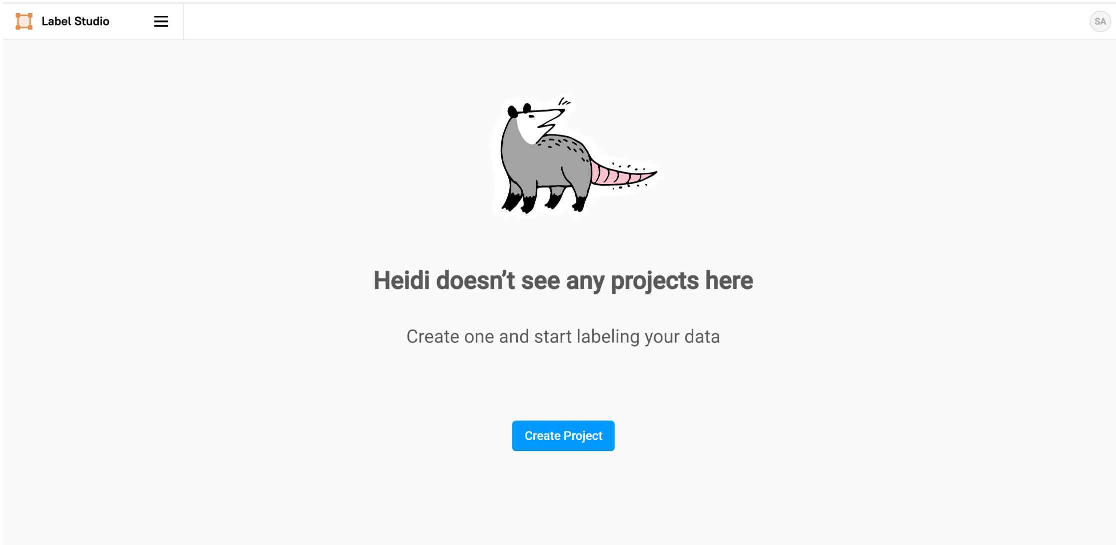
Entity Annotation: "Barack Obama was the 44th President of the United States. He was born in Hawaii and studied law at Harvard University."
Annotation Results:
Barack Obama PERSON 44th CARDINAL the United States GPE Hawaii GPE Harvard University ORG

Sentiment Annotation: "De Niro has the ability to make every role he portrays into acting gold. He gives a great performance in this film and there is a great scene where he has to take his father to a home for elderly people because he can't care for him anymore that will break your heart. I will say you won't see much bette acting anywhere."
Annotation Results: Positive

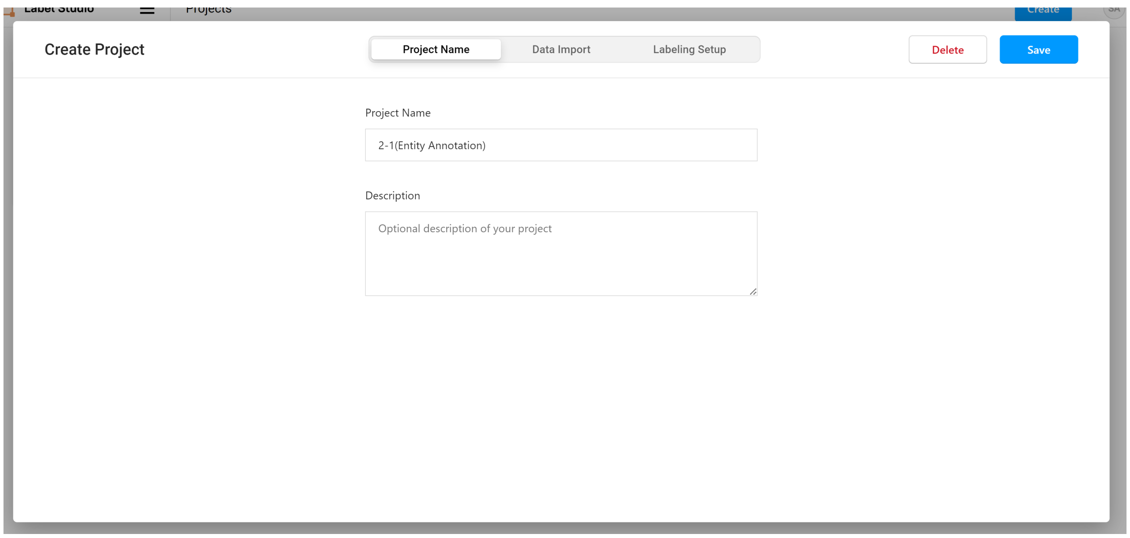
Answer #### For this task, I used label studio to perform annotation.

- Label Studio: It is an open-source application for data annotation that supports a variety of annotation kinds, including object identification, named entity recognition, and text classification.
- Source: https://github.com/heartexlabs/label-studio/
- We can install label-studio in anaconda environment, by below steps:
- step 1: conda create –name label-studio
- step 2: conda activate label-studio
- step 3: pip install label-studio

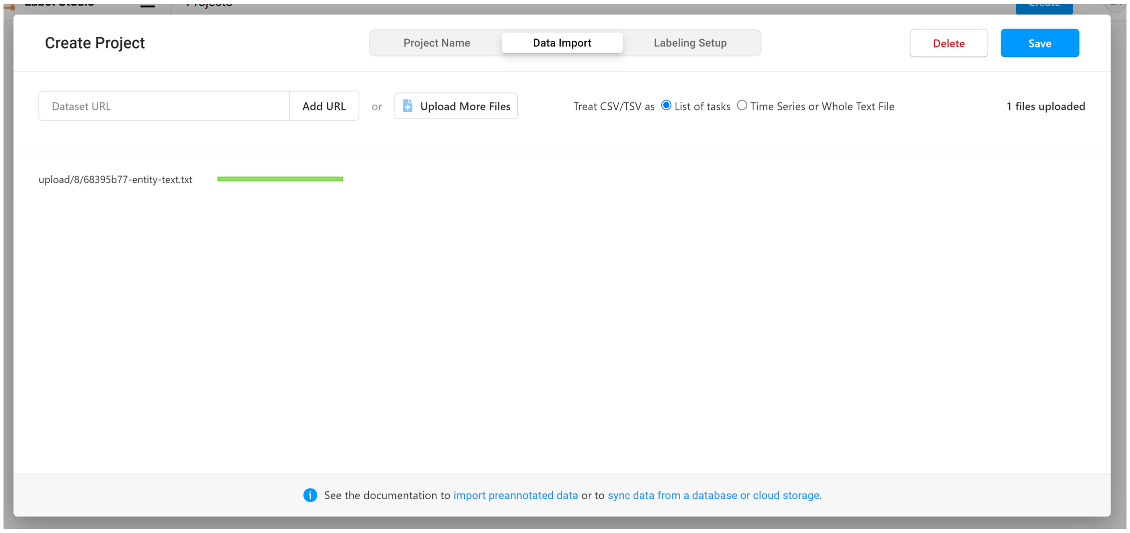
After installation you can navigate to the google chrome browser and signup for a new account. Click on the New project on landing page.



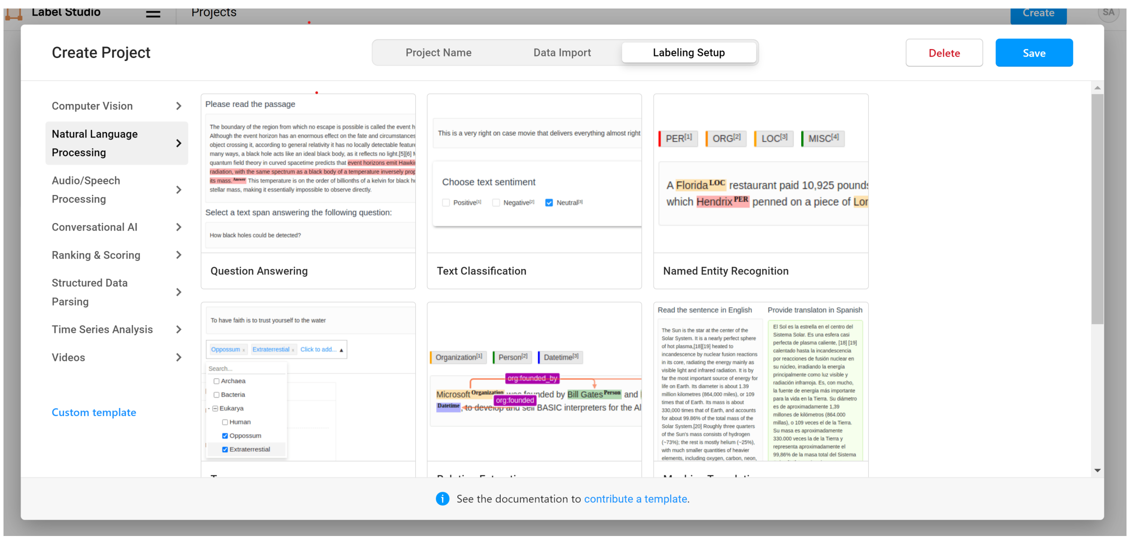
You will get below popup window and Enter the project name



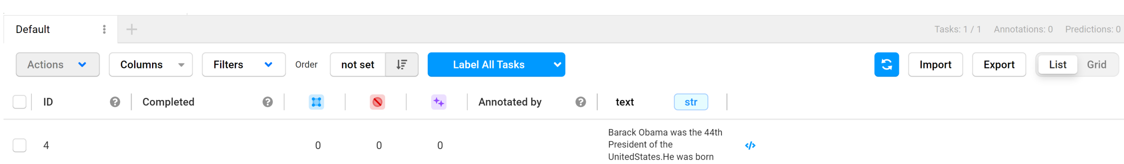
Now Click on the Data import and import the desired text file



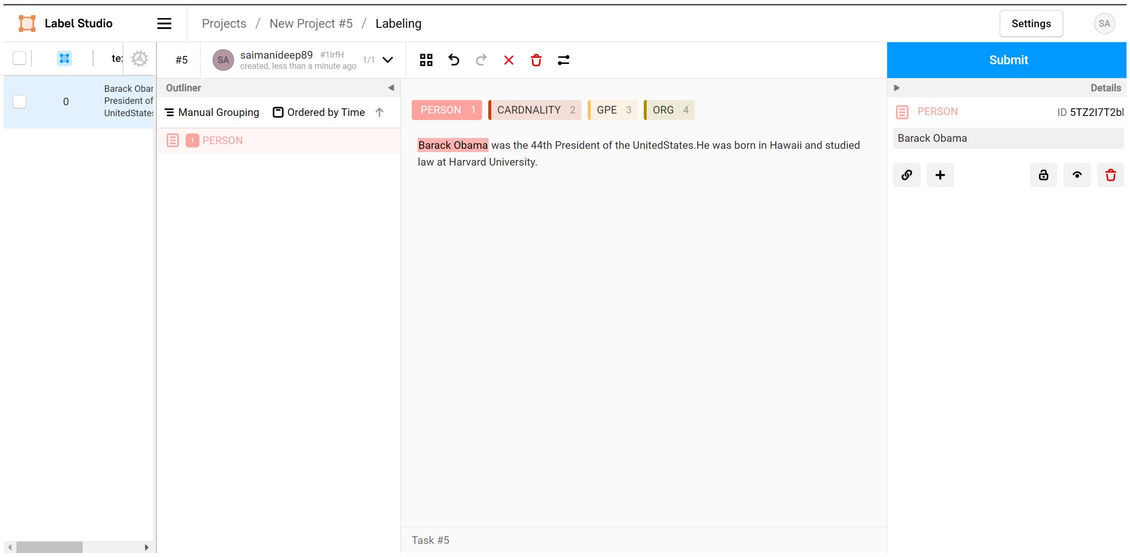
Now click on the Labeling setup and select the Natural Language Processing and select the Named Entity Recognition tab.



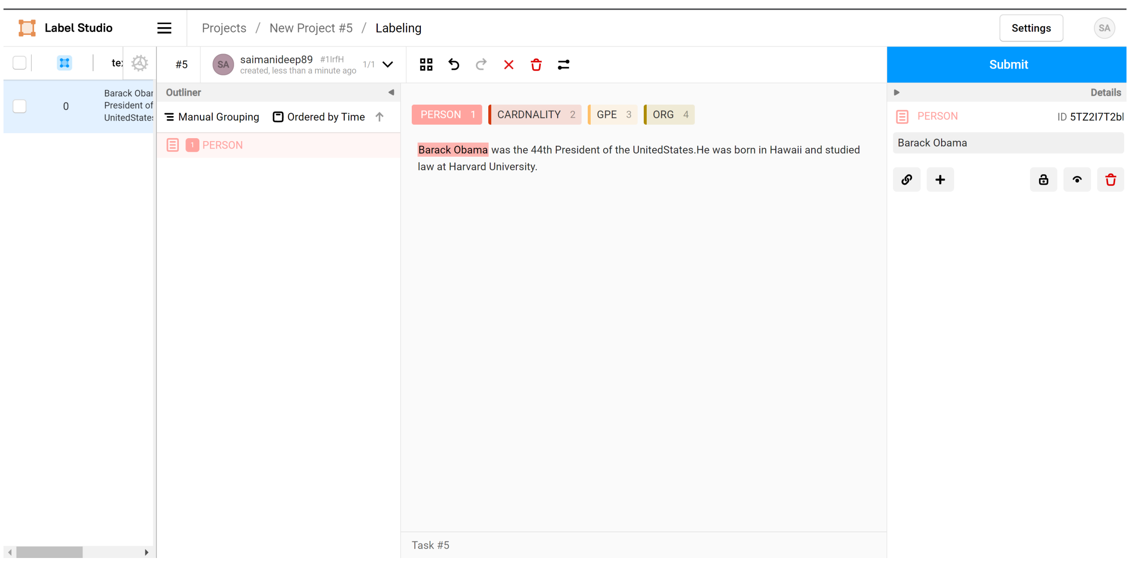
- You will be navigated to new page and Remove all the pre existing labels and add desired labels
- Select the Configure date as import file from the dropdown.
- After completion of two steps click the Save button.



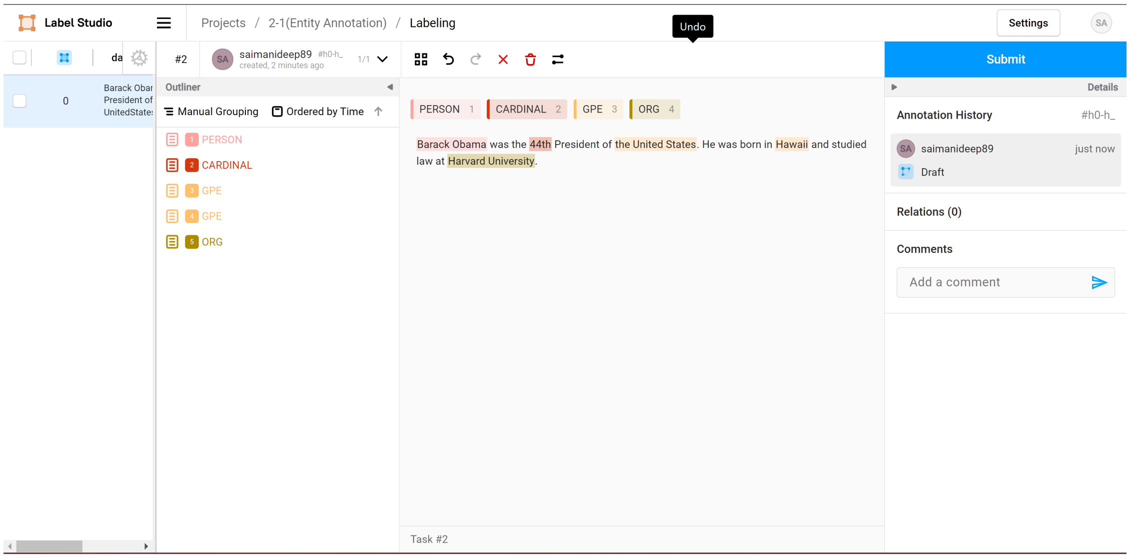
After clicking on the save button, You will be redirected to the new page to labeling the text.



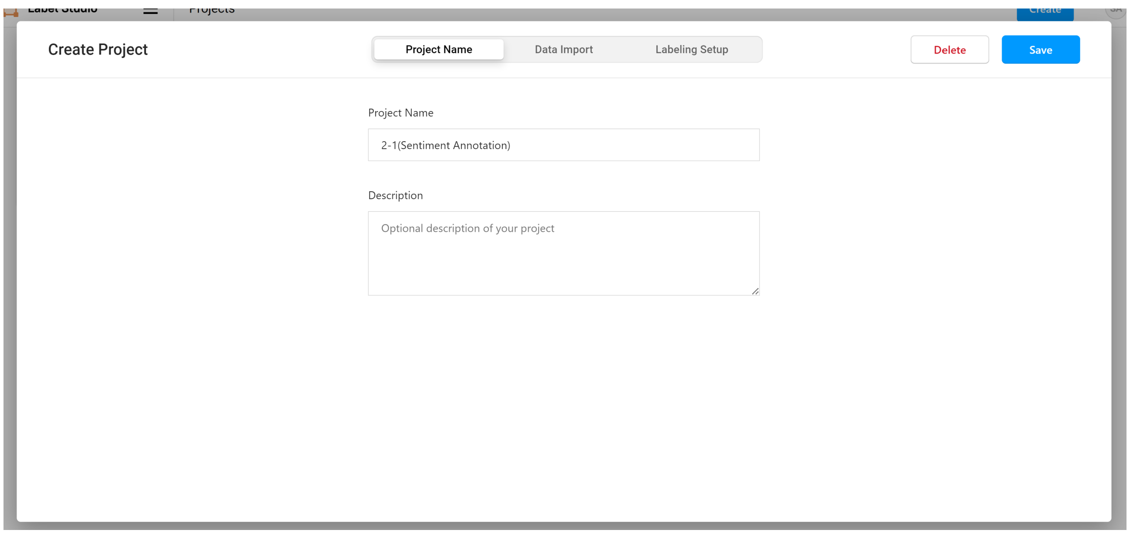
Select the ID for the label You will be redirected to new page for labeling. Follow this steps for labeling: Select the label and highlight the desired text and click enter

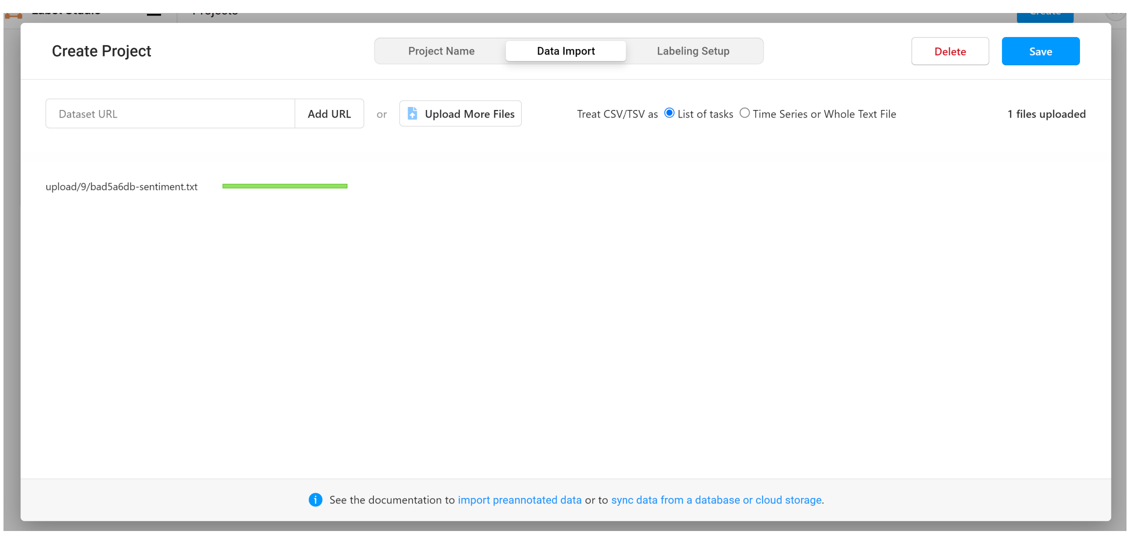


Now repeat all the steps for the remaining labels

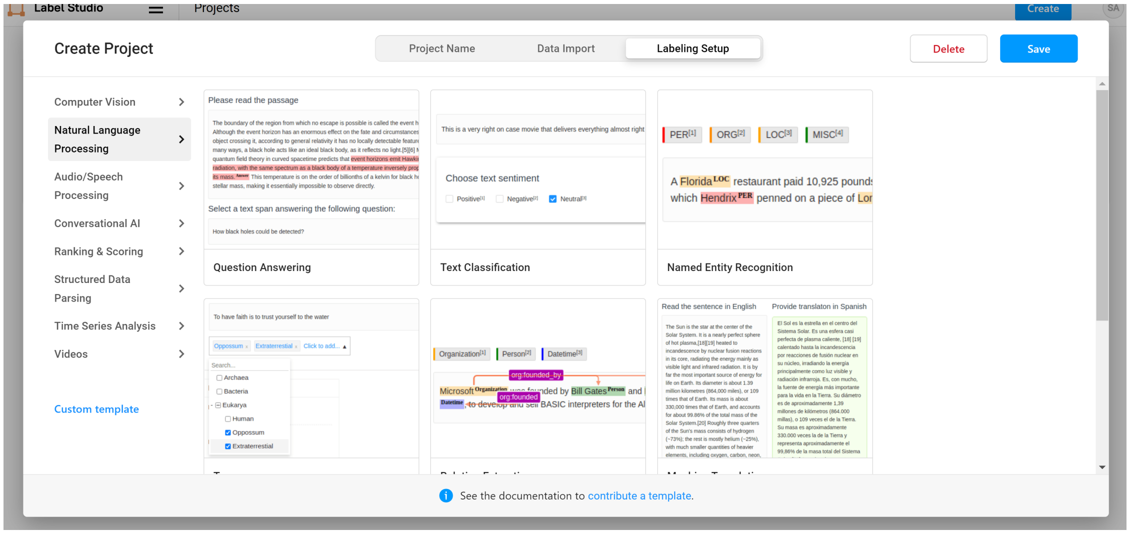


Repeat the same process for sentiment annotation for upto data importing

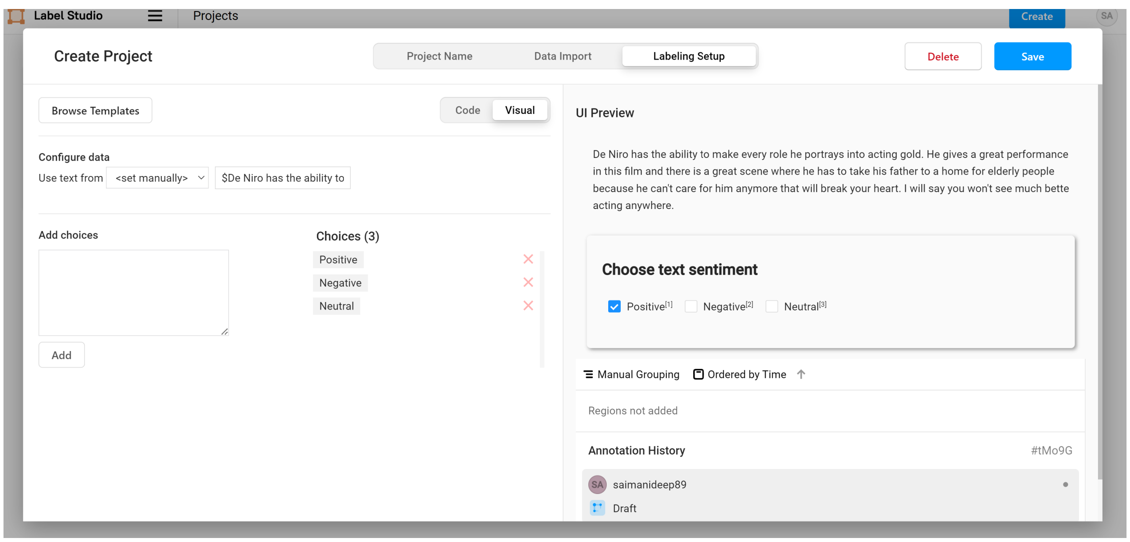




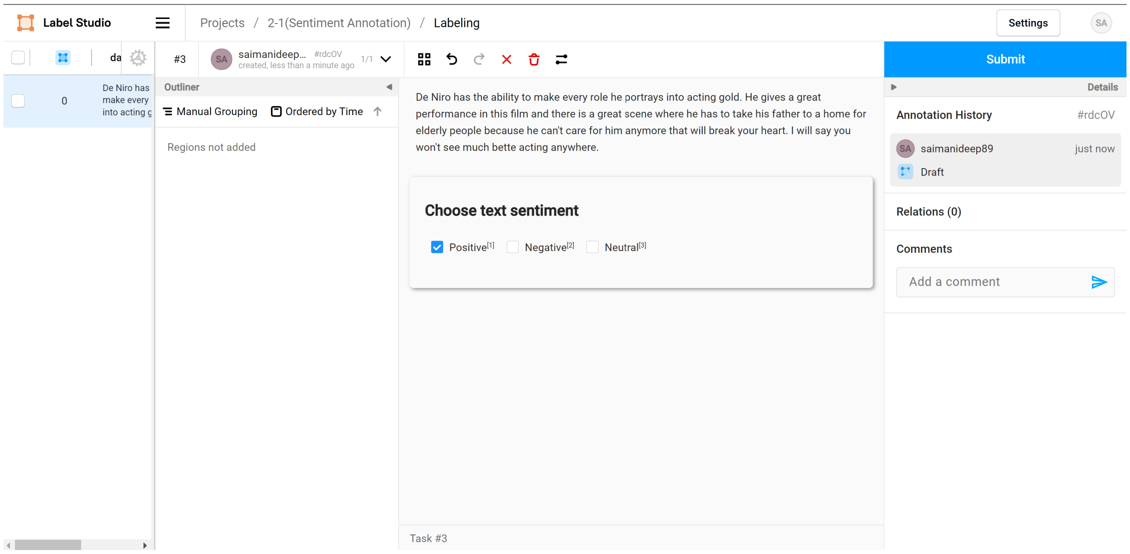
Now, select the labeling method, select the Natural Language Processing and select the text classification



After selecting, you will be redirected to the next page. Click on the Save button.



You will be redirected to the new page, Now select the ID and Select the type of the sentiment Click on the submit button to save the annotation



Thus, Entity and Sentiment Annotation is done by label-studio.

2-2 Active learning is a method to improve annotation efficiency. The following code imitates an active learning process.

[23]:
```
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2,␣
 ↪random_state=42)

# Split the dataset into initial training set and pool set
X_train, X_pool, y_train, y_pool = train_test_split(X, y, test_size=0.9,␣
 ↪random_state=42)

# Initialize the active learning loop
iterations = 10
batch_size = 10
model = LogisticRegression(random_state=42)

for i in range(iterations):
    print("Iteration {}:".format(i+1))

    # Train the model on the current training set
    model.fit(X_train, y_train)

    # Predict the labels of the unlabeled instances in the pool set
    y_pool_pred = model.predict(X_pool)

    ### below
    y_pool_prob = model.predict_proba(X_pool)
    entropy = -np.sum(y_pool_prob * np.log(y_pool_prob), axis=1)
    query_idx = np.argsort(entropy)[-batch_size:]
    ### above
```

```
    X_query = X_pool[query_idx]
    y_query = y_pool[query_idx]

    # Add the labeled instances to the training set and remove them from the
↪pool set
    X_train = np.concatenate([X_train, X_query])
    y_train = np.concatenate([y_train, y_query])
    X_pool = np.delete(X_pool, query_idx, axis=0)
    y_pool = np.delete(y_pool, query_idx)

    # Compute and print the accuracy of the model on the test set
    y_test_pred = model.predict(X_pool)
    accuracy = accuracy_score(y_pool, y_test_pred)
    print("Accuracy: {:.3f}\n".format(accuracy))
```

```
Iteration 1:
Accuracy: 0.828

Iteration 2:
Accuracy: 0.834

Iteration 3:
Accuracy: 0.851

Iteration 4:
Accuracy: 0.864

Iteration 5:
Accuracy: 0.874

Iteration 6:
Accuracy: 0.879

Iteration 7:
Accuracy: 0.881

Iteration 8:
Accuracy: 0.883

Iteration 9:
Accuracy: 0.886

Iteration 10:
Accuracy: 0.894
```

2-2(a)What is the purpose of the code between "### below" and "### above"?

Above code is Uncertainty-based sampling

This code calculates the entropy of the logistic regression model's predicted probabilities on the pool set's unlabeled examples, then chooses the most uncertain instances to be classified and added to the training set.

Especially regarding, model.predict proba(X pool) determines the predicted probability of the logistic regression model on the unlabeled instances in the pool set. The result is a 2D array with the shape (n pool, 2), where n pool denotes the quantity of examples in the pool set and the second dimension denotes the expected probability for each class (0 and 1).

For each occurrence in the pool set, the entropy of the predicted probabilities is calculated using the formula -np.sum(y pool prob * np.log(y pool prob), axis=1). The negative sum of the product of the predicted probabilities and the predicted probabilities' logarithm is known as entropy, which is a measure of uncertainty. The array's second axis, or axis=1, is used in this calculation to total the entropy values for each instance.

np.argsort(entropy)[-batch size:] The indices of the instances with the highest entropy are returned after sorting the entropy values in ascending order. The instances with the highest entropy are chosen using the [-batch size:] notation's last batch size indices. The query idx variable, which is used to choose the instances to be labeled and included to the training set, stores these indices.

Replace these code and other necessary code(as few as possible) to implement the active learning method in another strategy

I'm chossing the Query-by-committe Sampling

Query-by-committee: generates a committee of hypotheses and selects the unlabeled examples on the basis of disagreement among different hypotheses. - Suppose we create several distinct models with the same dataset. One model can be of SVM, the second model can be of Decision Tree, third can be of Logistic Regression, and so on… For this code, I'm using Logistic Regression, Random Forest, - Now among this committee of different models, we measure disagreement in the predictions for a particular data sample. - Active learner determines to query the annotator for labeling a data sample if it produces most disagreement in terms of predictions.

[24]:
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


X, y = make_classification(n_samples=1000, n_features=10, n_classes=2,
 ↪random_state=42)

X_train, X_pool, y_train, y_pool = train_test_split(X, y, test_size=0.9,
 ↪random_state=42)

iterations = 10
batch_size = 10
```

```python
classifiers = [LogisticRegression(random_state=42),␣
↪RandomForestClassifier(random_state=42)]

for i in range(iterations):
    print("Iteration {}:".format(i+1))

    committee_pred = np.zeros((X_pool.shape[0], len(classifiers)))
    for j, clf in enumerate(classifiers):
        clf.fit(X_train, y_train)
        committee_pred[:, j] = clf.predict(X_pool)

    # Compute disagreement scores
    disagreement = np.sum(committee_pred != np.expand_dims(committee_pred.
↪mean(axis=1), axis=1), axis=1)

    # Select the most uncertain instances to be labeled and added to the␣
↪training set
    query_idx = np.argsort(disagreement)[-batch_size:]

    X_query = X_pool[query_idx]
    y_query = y_pool[query_idx]

    X_train = np.concatenate([X_train, X_query])
    y_train = np.concatenate([y_train, y_query])
    X_pool = np.delete(X_pool, query_idx, axis=0)
    y_pool = np.delete(y_pool, query_idx)

    committee_pred = np.zeros((X_pool.shape[0], len(classifiers)))
    for j, clf in enumerate(classifiers):
        clf.fit(X_train, y_train)
        committee_pred[:, j] = clf.predict(X_pool)
    y_test_pred = np.round(committee_pred.mean(axis=1))
    accuracy = accuracy_score(y_pool, y_test_pred)
    print("Accuracy: {:.3f}\n".format(accuracy))
```

```
Iteration 1:
Accuracy: 0.871

Iteration 2:
Accuracy: 0.878

Iteration 3:
Accuracy: 0.885

Iteration 4:
Accuracy: 0.888
```

```
Iteration 5:
Accuracy: 0.880

Iteration 6:
Accuracy: 0.888

Iteration 7:
Accuracy: 0.895

Iteration 8:
Accuracy: 0.904

Iteration 9:
Accuracy: 0.911

Iteration 10:
Accuracy: 0.910
```

Compare these two strategies, which one is better in this example?

You can cleary see that accuracy is increasing when no. of iterations increases. Thus Query-By-Committee is better strategy compared to Uncertainty-based sampling

2-2(b)If the code is used for movie review annotation, how many reviews need to be labelled by the annotator every time?

The annotator would have to label 10 reviews per iteration if the code were used to annotate 1000 comments for movie reviews. This is due to the code's use of a batch size of 10, which divides each iteration into batches of 10 comments. In order to label 10 comments in one iteration, the annotator would have to name one comment every batch.

This is only a rough estimate, though. Depending on the caliber of the annotations, the expertise of the annotator, and the required level of model accuracy, the actual number of reviews that need to be labeled may change.

Discuss the possible pros and cons by increasing and decreasing this number.

The model's accuracy might be increased by increasing the number of reviews that must be labeled in a single iteration. This is because the model would be able to learn from a wider variety of comments and would have access to more data to train on. The annotator would need additional time to classify the remarks, though.

The annotator might save time by decreasing the amount of reviews that need to be labeled throughout each iteration. Yet it can also make the model less accurate. This is because the model would be less able to learn from a wide range of comments and would have fewer data to train on.