MA5851 Assignment 3, Document Three

April 25, 2021

1 Named Entity Recognition

1.1 Rational for Utilising Named Entity Recognition

As mentioned in Document Two, 541 out of 1,287 rows of extracted news data did not possess tags listing which states or territories those articles discussed. This was perceived as an issue since this data was necessary for estimating the relative strengths of the property markets across all states and territories in Australia. Hence, Named Entity Recognition (NER) was required to identify all place locations in the text corpus.

1.2 Named Entity Recognition Literature Review

NER refers to the task of identifying key information, or entities, within a text corpus such as a location, person, organisation or time (Marshall, 2019). There exist several distinct approaches to NER which include:

- **Rule-Based**: detects entities in a text corpus if they match a list of known entities and any other hand-crafted rules. The advantage of this approach is that it does not rely on any annotated data and can demonstrate good robustness and coverage of the obtained results (Chahira et al., 2017 & GeeksforGeeks, 2020). According to current literature however, this approach has been progressively abandoned due to the high time and manual work requirements, reduced learning capacity of the system and potential existence of overly complicated patterns in the data (Chahira et al., 2017; Li et al., 2020 & Thanaki, 2017).

- **Unsupervised Learning**: uses a model previously trained through unsupervised tasks, such as masked language modelling and sentence prediction, to detect entities based on the term and the context of the sentence (Rajasekharan, 2020). The *Bidirectional Encoder Representations from Transformers* (*BERT*) model is an example of an unsupervised learning approach to NER (Horev, 2018). The advantages of the unsupervised learning approach is that it does not rely on a large amount of high-quality annotated data and can attain high accuracy on unseen data (Li et al., 2019). However, the disadvantage of this approach is that state-of-the-art model solutions that require fine-tuning can require high computational power (Hui, 2019).

- **Feature-Based Supervised Learning**: this is the most common and leading approach in NER which uses training data and their features as the input to generate a trained shallow feedforward neural network model. *Spacy's* pre-trained NER model is an example of a simple, supervised classifier (Jack, 2019). This trained model can then be used to detect similar entities in new data. The advantage of this approach is that it can attain high accuracy on unseen data (Toral, 2015). However, the disadvantage of this approach is that it relies on a large amount of high-quality annotated data (Li et al., 2019).

- **Deep-Learning**: uses multiple processing layers in a neural network to learn non-linear mappings between the input and the output via non-linear activation functions. The advantage of being able to learn complex and intricate features in this way is that the deep-learning model can remove the need for domain expertise and sophisticated feature extraction while still achieving high accuracy. That is, the model can accept raw data (Mahapatra, 2018; Maheshkar, 2020 & Vilariño, 2020). Again, the disadvantage of this approach is that it relies heavily on a large amount of high-quality annotated data and is computationally expensive to train (Vilariño, 2020).

Some of the existing challenges faced during NER, and even sentiment analysis, involve the presence of slang words, new accents and grammatical and spelling mistakes (Devi, 2020). However, published news articles are typically quality checked for these items.

1.3 Chosen Named Entity Recognition Method

Spacy's pre-trained, english NER model, *en_core_web_sm*, was the selected NER model for detecting locations in the Australian news articles due to the absence of a list of all known location entities, lack of high-quality annotated data and because the location entities were deemed to be simple to identify in grammatically correct text. Furthermore, it was conceived that implementing a rule-based approach would not have been possible without this list of known location entities and that a deep-learning approach would have been excessive for this particular task. Moreover, the unsupervised learning approach was excluded as the *BERT* model would have required more than 16GB GPU memory, which exceeded the available resources of 8GB GPU memory (Hui, 2019). The following figure displays an example result of *Spacy's* NER model applied to a portion of the body text in an Australian news article.

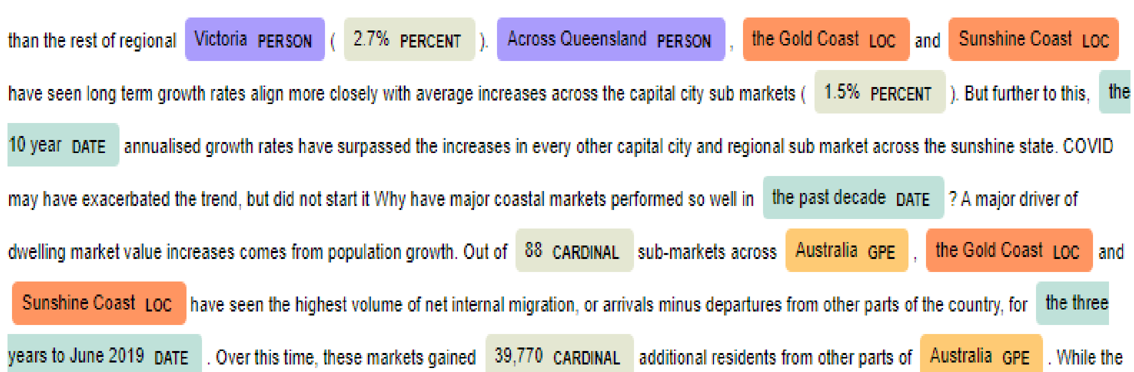


Figure 1: NER Example

As can be seen from the above figure, *Spacy's* NER model identified different categories of key information within the body text corpus, however, the only entities of interest for this task consisted of the locations (LOCs) and geopolitical entities (GPEs). Consequently, the entities were later filtered to only include LOCs and GPEs.

Additionally, it was rendered clear from the above figure that the entities identified were not always at the state and territory level (for example, Gold Coast was a city, not a state or territory). Hence, it was deemed necessary to employ the *Nominatim* geocoding software, from within Python, to compute the full addresses from these LOC and GPE tags and then return the associated states or territories, if applicable.

1.4 Named Entity Recognition Performance Results

In order to validate the NER method implemented, the predicted state or territory for each news article was compared with the true tag labels extracted from the *onthehouse* news article pages. Since not all of the articles contained tags, these records were removed from the comparison. Furthermore, only records predicted to include at least one of the states or territories in Australia were kept for this task. This comparison revealed that the implemented NER method possessed an accuracy of 73%. This accuracy was regarded as a good baseline to improve upon for future iterations of this NLP pipeline.

2 Sentiment Analysis

2.1 Rational for Utilising Sentiment Analysis

Polarity-based sentiment analysis was incorporated into the NLP pipeline in order to gain the necessary insights into the relative strengths of housing across all states and territories in Australia. Valence-based sentiment analysis was not actively pursued as this would have posed challenges in trialling machine learning-based approaches (Artiles, 2017).

2.2 Sentiment Analysis Literature Review

Sentiment analysis has been used in literature to gauge the attitude of the writer towards a particular subject. This traditionally has involved predicting whether the text positively, neutrally or negatively reflects the subject (Devi, 2020). The two main approaches towards sentiment analysis consist of:

- **Lexicon-Based**: which assumes that each word in the text corpus possess an emotion and that the quantification of these emotions would be indicative of the writer's attitude towards the subject. A popular example of a lexicon-based approach includes the Valence Aware Dictionary and sEntiment Reasoner (VADER) (Artiles, 2017). The advantage of this approach is that labelled data and the procedure of training a classifier is not required. However, this approach can suffer from a recall and overall accuracy perspective (Isabelle et al., 2018).

- **Machine Learning (ML) Algorithms**: involves using a trained classifier to predict whether the text corpus positively, neutrally or negatively reflects the subject (Pajupuu, 2016). The Logistic Regression (LR) model, Support Vector Machine (SVM) model, Naive Bayes (NB) model, and even the BERT model, are traditional examples of machine learning algorithms employed for sentiment analysis tasks (Molla, 2018 & Pajupuu, 2016). The key advantage of this approach is that it has demonstrated high accuracy in literature over the lexicon-based approach in a number of cases. However, its disadvantage involves its reliance on good feature extraction, large amount of high-quality annotated data and reduced performance in other domain applications (Devika et al., 2016).

2.3 Chosen Sentiment Analysis Method

Both lexicon-based and machine learning-based approaches were trialled before deciding on the best performing model to use in production. This included trialling the VADER model; a weighted, multinomial LR model; a weighted, multiclass SVM model and a multinomial NB model. In order to support multi-class classification for the LR and SVM models, the *One-vs-Rest* (*OvR*) strategy was used to generate one binary classification problem per class. The BERT model was excluded due to the previously mentioned resource limitations.

To prepare for these trials, the text corpuses needed to be further processed. This was because it was deemed highly likely that multiple states and territories would be mentioned in any given news article which could also mean multiple, differing sentiments. Hence, it was important to be able to individually identify only the relevant sentences containing each state or territory. This involved exploding the DataFrame using the NER place tags collected earlier, filtering on the single sentences from each article that contained the identified LOC or GPE entity and then collecting the previous 110 sentence characters and the subsequent 110 sentence characters for additional context during the sentiment analysis. This produced 3,896 records in the DataFrame indexed by either a state or territory.

Additionally, the text corpuses needed to be cleaned. This involved:

- **Removing excess white spaces**: which served as unwanted noise.
- **Converting to lowercase**: so that identical words with casing differences could be treated the same.
- **Removing numbers**: since this was difficult to contextualise.
- **Removing stopwords**: to provide more focus on the important features in the text.
- **Lemmatisation**: to reduce inflectional forms and derivationally related forms. Lemmatisation was chosen over stemming as the root form would still be a word recognised by the VADER model (Heidenreich, 2018).

Moreover, a portion of the data needed to be labelled, as either positive, neutral or negative, in order to train the ML models and to convert the VADER compound scores into these discrete classes for comparison purposes. Due to time constraints, only 500 records could be labelled. With a 90/10 split, this left 450 records for training the ML models and 50 for testing. After labelling, it was observed that the dataset was slightly imbalanced. This is shown in the figure below. To account for the slightly imbalanced dataset, the 'balanced' hyperparameter setting was used across the ML models to adjust the weights inversely to the class frequencies (Scikit Learn, 2021).

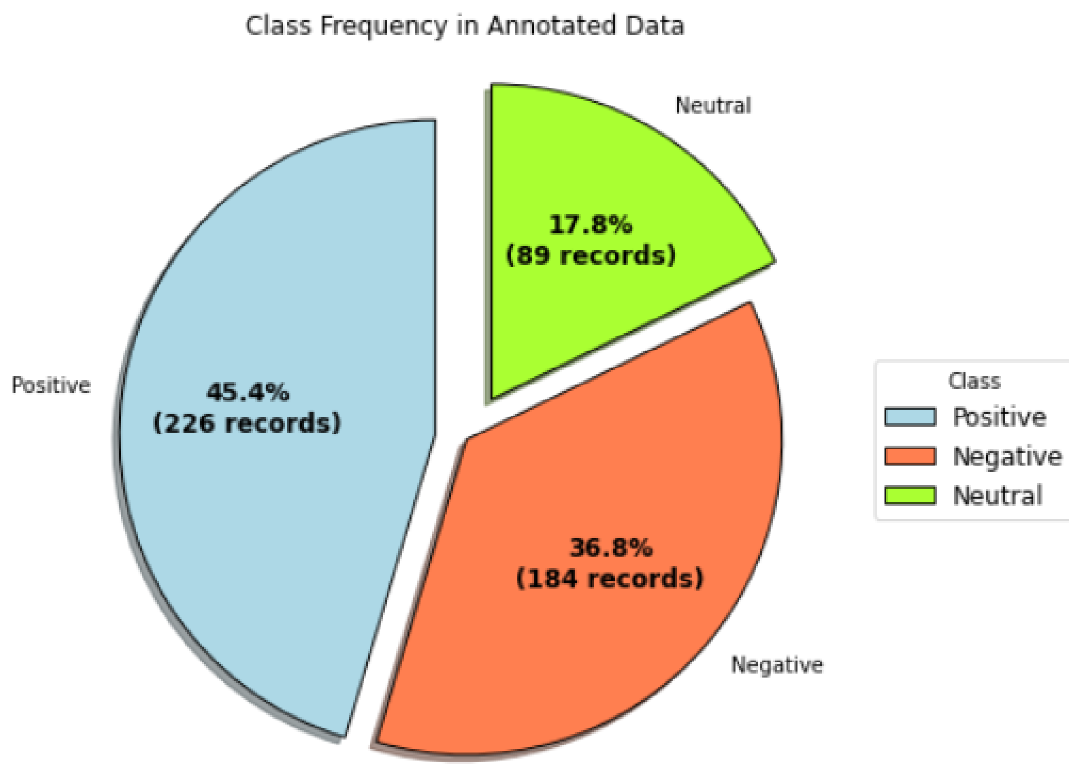


Figure 2: Imbalance in Annotated Dataset

To convert the VADER compound scores into discrete classes, the scores were compared with the manual labels and the best possible thresholds were decided based on class separability. As shown in the figure below, class separability was poor, however, the threshold for positive was any score greater than 0; the threshold for negative was any score less than -0.1 and neutral was assigned to any scores in between these two thresholds.

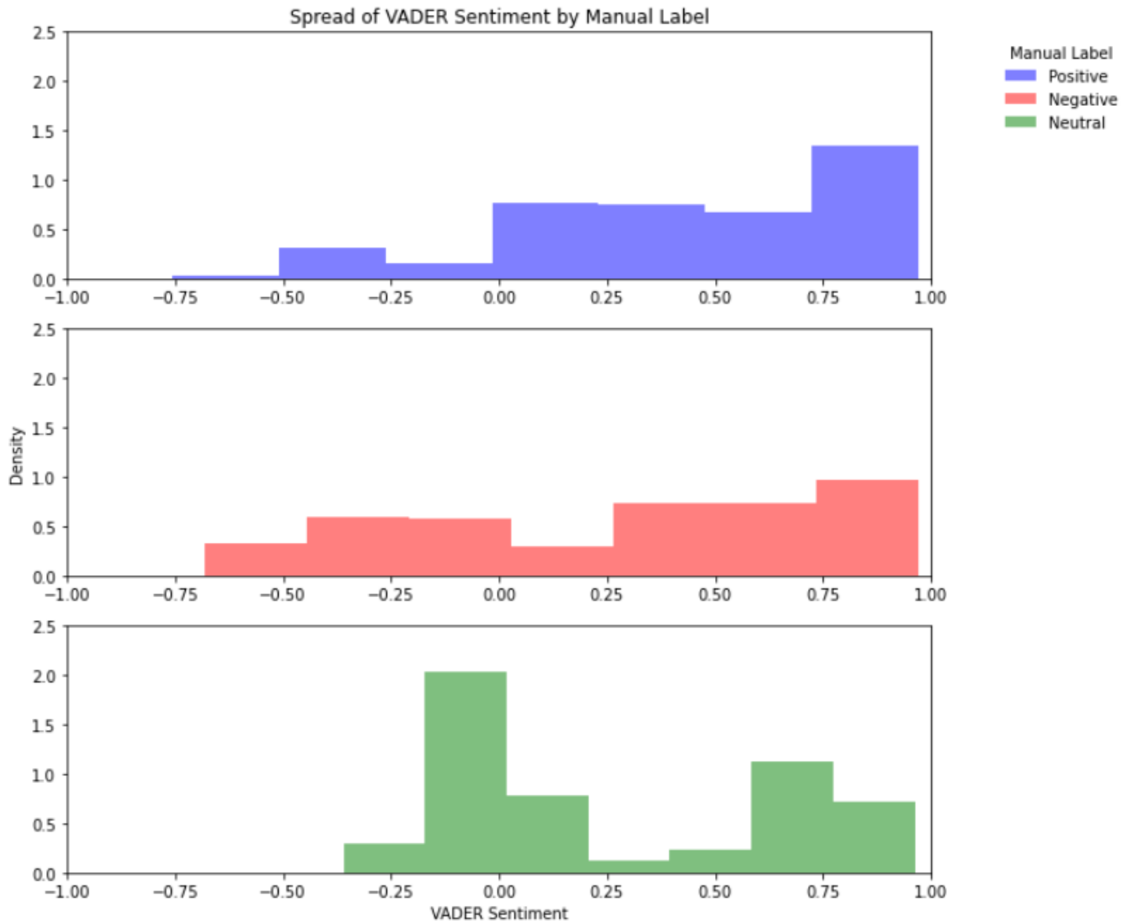


Figure 3: VADER Compound Scores by Manual Label

In order to train and make predictions using the ML models, feature extraction using Term Frequency-Inverse Document Frequency (TF-IDF) was used to convert the training and test data into normalised term frequency arrays. This normalised term frequency array, as opposed to a regular frequency array using Bag of Words (BoW), was specifically required for the LR and SVM models.

To optimise the hyperparameters and to avoid overfitting the ML models during training, *GridsearchCV* with a *Stratified K-Fold* strategy was implemented. This decision to use cross-validation over the use of a validation set meant that the small training set would not be reduced even further (Stack Exchange, 2019). Regularisation of the LR model was achieved using the L2 penalty as this was the only option supported for the 'multinomial' setting (Scikit Learn, 2021). Additionally, low 'C' parameters for the SVM model were investigated in order to soften the SVM margin and reduce overfitting (Scikit Learn, 2021). Moreover, a high 'alpha' setting was investigated for the NB model to reduce overfitting as well (Tadagoppula, 2020).

The multinomial NB model was found to be the best performing model. Thus, this model was saved with *pickle* for use in production. Further details on the model performances are included in the subsequent subsection.

The predictions from the sentiment analysis classifier were then used as scoring weights to the frequency of published news articles per state or territory. These scores were then plotted as a function of time to gain the necessary insights into the relative strengths of housing across all states and territories in Australia. This end deliverable is further detailed in Section 3.

2.4 Sentiment Analysis Performance Results

To assess the model performances, the precision, recall and macro F1-scores were analysed. As can be seen from the tables below, the LR model possessed the highest accuracy of 68%, followed by the NB model at 64%, the SVM model at 62% and the VADER model at 46%. The macro average F1-score was used for comparison instead of the accuracy as the dataset was slightly imbalanced.

VADER

	precision	recall	f1-score	support
Neutral	0.45	0.38	0.41	89
Positive	0.50	0.77	0.61	227
Negative	0.58	0.24	0.34	184
accuracy			0.51	500
macro avg	0.51	0.47	0.46	500
weighted avg	0.52	0.51	0.48	500

SVM Model

	precision	recall	f1-score	support
Neutral	0.45	0.56	0.50	9
Positive	0.74	0.61	0.67	23
Negative	0.65	0.72	0.68	18
accuracy			0.64	50
macro avg	0.61	0.63	0.62	50
weighted avg	0.65	0.64	0.64	50

LR Model

	precision	recall	f1-score	support
Neutral	0.56	0.56	0.56	9
Positive	0.67	0.78	0.72	23
Negative	0.86	0.67	0.75	18
accuracy			0.70	50
macro avg	0.69	0.67	0.68	50
weighted avg	0.72	0.70	0.70	50

NB Model

	precision	recall	f1-score	support
Neutral	1.00	0.44	0.62	9
Positive	0.61	0.87	0.71	23
Negative	0.69	0.50	0.58	18
accuracy			0.66	50
macro avg	0.77	0.60	0.64	50
weighted avg	0.71	0.66	0.65	50

Figure 4: Classification Reports for Sentiment Analysis

While the LR model was found to have the highest macro F1-score, it was excluded from selection because it displayed evidence of overfitting when comparing the k-fold macro F1-scores between the train and test data. That is, the macro F1-score in the training data was significantly higher than the macro F1-score in the test data.

LR Model

K-Fold Split	Train	Test
1	97.1%	39.8%
2	96.6%	66.7%
3	96.7%	68.1%
4	97.3%	70.4%
5	97.0%	73.8%
Mean ± Std Dev	96.9% ± 0.3%	64.4% ± 9.4%

SVM Model

K-Fold Split	Train	Test
1	64.7%	36.7%
2	65.9%	68.3%
3	63.0%	64.0%
4	64.0%	47.8%
5	63.7%	61.8%
Mean ± Std Dev	64.9% ± 1.9%	54.6% ± 9.4%

NB Model

K-Fold Split	Train	Test
1	63.1%	41.0%
2	66.5%	71.7%
3	63.8%	56.5%
4	64.8%	49.1%
5	64.4%	61.5%
Mean ± Std Dev	64.7% ± 1.7%	54.0% ± 8.9%

Figure 5: Evidence of Overfitting

Consequently, the multinomial NB model was selected for use in production as it was the next most accurate model and did not show evidence of overfitting. The model's accuracy of 64% was also regarded as a good baseline to improve upon for future iterations of this NLP pipeline.

3 High-Level Assessment of Results and Output

As can be seen from the figure below, the proposed NLP pipeline successfully provided an estimate of the relative strengths of the property market across all states and territories in Australia, as a function of time. The figure revealed that at the start of 2021, Victoria was the strongest property market in Australia, followed by NSW, QLD, ACT, WA, TAS, SA and then NT. Since the NLP task accuracies were not 100%, these sentiment scores require frequent checks and readjustments, using a separate and reliable source, to avoid errors from compounding over time.

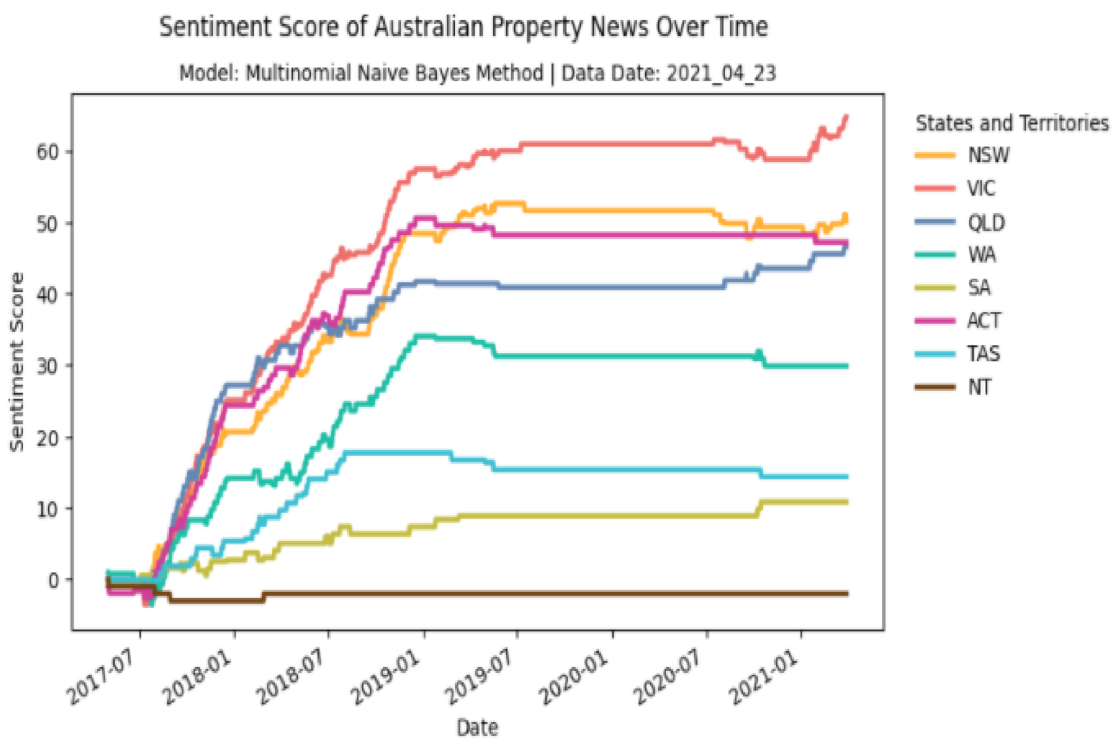


Figure 6: End Deliverable

4 Natural Language Processing Pipeline Code

4.1 Named Entity Recognition Model

4.1.1 Identify Places

[22]:
```
# import packages
import warnings # suppress warnings
warnings.filterwarnings('ignore') # suppress warnings
import pandas as pd # for creating dataframes
pd.options.mode.chained_assignment = None  # to suppress SettingWithCopyWarning
from datetime import datetime # for getting current date
import matplotlib.pyplot as plt # for plotting
from matplotlib.pyplot import figure # for plotting
import spacy # for NER
from spacy import displacy # for displaying NER results

# loading trained, enlighs NER model
nlp = spacy.load('en_core_web_sm')
```

[23]:
```
# import packages
import warnings # suppress warnings
warnings.filterwarnings('ignore') # suppress warnings
import pandas as pd # for creating dataframes
import re # use regex to perform acquisition of selective sentence

# loading trained NER model
nlp = spacy.load('en_core_web_sm')

# read in csv saved previously
data2 = pd.read_csv(r'C:/Users/Imran/Desktop/Assignment_3_Repo/data2.csv')

# define function for extracting geographical entities (GPE) and locations␣
↪(LOC) using NER
def get_place_list(x):
    # apply nlp to news article
    article = nlp(x)
    # extract GPE entities from article
    place_list = [(X.text) for X in article.ents if (X.label_ == 'GPE') or (X.
↪label_ == 'LOC')]
    # sort alphabetically and de-duplicate GPE list
    place_list = sorted(list(set(place_list)))
    return place_list
# apply function
data2['Place'] = data2['Body_Transformed'].apply(get_place_list)

# duplicate article for each GPE identified so sentimenet can be calculated per␣
↪GPE
```

```python
data2 = data2.explode('Place').reset_index(drop = True)

# select relevant sentences for each place
def get_sentences(place, text):
    # extract only the sentences containing the places
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text)
    sentences_list = [word for word in sentences if str(place) in word]
    # use list comprehension convert list to string
    #sentences_string = ' '.join([str(elem) for elem in sentences_list])
    #return sentences_string
    return sentences_list
data2['Relevant_Body'] = data2.apply(lambda x: get_sentences(x['Place'],␣
↪x['Body_Transformed']), axis=1)

# duplicate article for each GPE identified so sentimenet can be calculated per␣
↪GPE
data2 = data2.explode('Relevant_Body').reset_index(drop = True)
```

4.1.2 Get Additional Context for Sentences Containing Place Tags

[24]:
```python
# define function for extracting geographical entities (GPE) and locations␣
↪(LOC) using NER
def get_additional_context(text, sentence):
    index = text.find(str(sentence))
    if index < 0:
        string = sentence
    else:
        start = index - 110
        end = index + len(str(sentence)) + 110
        if start <= 0:
            start = 0
            string = text[start:end]
            end_new = string.rfind(" ")
            string = string[start:end_new]
        else:
            start = start
            string = text[start:end]
            start_new = string.find(" ")
            end_new = string.rfind(" ")
            string = string[start_new:end_new]
    return string
# apply function
data2['Relevant_Body2'] = data2.apply(lambda x:␣
↪get_additional_context(x['Body_Transformed'], x['Relevant_Body']), axis=1)

#example_text = data2['Body_Transformed'].iloc[102]
#displacy.render(nlp(example_text), jupyter=True, style='ent')
```

4.1.3 Use Nominatim Software to get Full Address from Place Tags

[25]:
```python
# import packages
import time # for implementing time delays
from geopy.geocoders import Nominatim # for open-source geoencoding

# get unique values of geographical entities as repeated queries using␣
↪Nominatim is not allowed per the usage policy
data2_unique = data2[['Place']].drop_duplicates(keep = 'first')
# specify user agent as per usage policy
geolocator = Nominatim(user_agent="geoapiExercises")

# define function to get the full address of Geographical Entity
def get_full_address(x):
    location = geolocator.geocode(x)
    # limit requests to 1 every 1 second as per usage policy
    time.sleep(1)
    try:
        address = location.address
```

```
    except:
        address = 'None'
    return address
# the line below is commented out to prevent running Nominatim for previously␣
 ↪obtained locations as per usage policy
#data2_unique['Full_Address'] = data2_unique['Place'].apply(get_full_address)

# save addresses to csv (serves to cache the results so as to not repeat␣
 ↪queries as per usage policy of Nominatim)
#data2_unique.to_csv(r'C:/Users/Imran/Desktop/Assignment_3_Repo/
 ↪standardised_addresses.csv', index = False)
```

4.1.4 Roll-Up the Full Address to the State and Territory Level

[26]:
```
# read in csv saved previously
standardised_addresses = pd.read_csv(r'C:/Users/Imran/Desktop/Assignment_3_Repo/
 ↪standardised_addresses.csv')

# merge full address to dataframe by the place variable
data3 = data2.merge(standardised_addresses, on = 'Place', how = 'left')

# identify state
def get_Key_Place(x):
    if ('Australia' in str(x)) and ('New South Wales' in str(x)):
        Key_Place = 'NSW'
    elif ('Australia' in str(x)) and ('Queensland' in str(x)):
        Key_Place = 'QLD'
    elif ('Australia' in str(x)) and ('South Australia' in str(x)):
        Key_Place = 'SA'
    elif ('Australia' in str(x)) and ('Tasmania' in str(x)):
        Key_Place = 'TAS'
    elif ('Australia' in str(x)) and ('Victoria' in str(x)):
        Key_Place = 'VIC'
    elif ('Australia' in str(x)) and ('Western Australia' in str(x)):
        Key_Place = 'WA'
    elif ('Australia' in str(x)) and ('Northern Territory' in str(x)):
        Key_Place = 'NT'
    elif ('Australia' in str(x)) and ('Australian Capital Territory' in str(x)):
        Key_Place = 'ACT'
    elif ('Australia' in str(x)):
        Key_Place = 'National'
    else:
        Key_Place = "Overseas"
    return Key_Place
data3['Place_Tag'] = data3['Full_Address'].apply(get_Key_Place)
```

4.1.5 Assess Place Model Accuracy

[27]:
```
def place_matched(Place_Tag, Tags_Transformed):
    if str(Tags_Transformed).find(str(Place_Tag)) >=0:
        return 'Matched'
    else:
        return 'No Match'

# apply function
data3['Place_Matched'] = data3.apply(lambda x: place_matched(x['Place_Tag'],␣
↪x['Tags_Transformed']), axis=1)

# subset to only include labelled records
place_labelled = data3[data3['Tags_Transformed'].notna()]

# subset records to only include places which have been indexed by a state or␣
↪territory
place_labelled = place_labelled[(place_labelled['Place_Tag'] != 'Overseas') &␣
↪(place_labelled['Place_Tag'] != 'National')]

# compute accuracy
total = len(place_labelled)
matched = len(place_labelled[place_labelled['Place_Matched']=='Matched'])
Accuracy = matched/total
print("Place Model Accuracy:", "{0:.0%}".format(Accuracy))
```

```
Place Model Accuracy: 73%
```

4.2 Sentiment Analysis

4.2.1 Data Preprocessing Continued

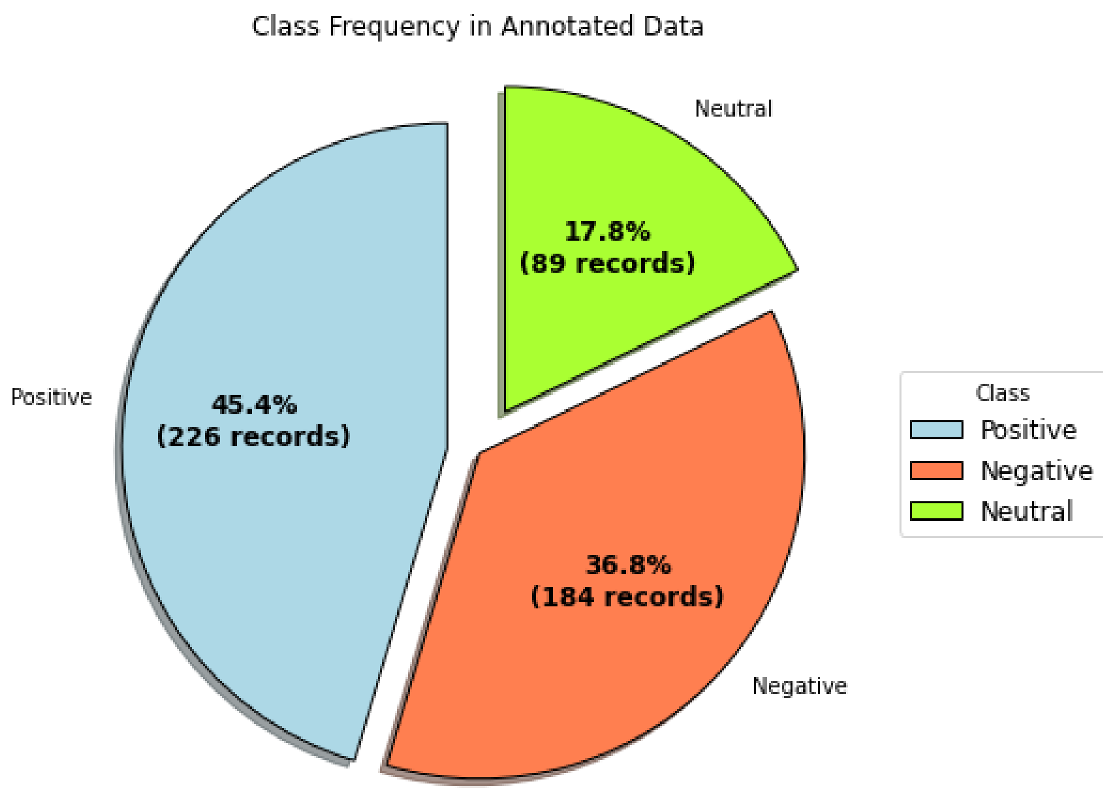
[28]:
```
# import packages
import nltk # import nltk for sentiment analysis
from nltk.stem import WordNetLemmatizer # for lemmatisation
from nltk.tokenize import word_tokenize # tokenize words for removal of␣
↪stopwords
from nltk.tokenize.treebank import TreebankWordDetokenizer # detokenize text

# remove excess white space
data3['Body_Cleaned'] = data3['Relevant_Body2'].astype(str).apply(lambda x: ' '.
↪join(x.split()))

# convert to lowercase
data3['Body_Cleaned'] = data3['Body_Cleaned'].str.lower()

# remove punctuation
```

```
data3['Body_Cleaned'] = data3['Body_Cleaned'].str.replace(r'[^\w\s]', '', regex␣
↪= True)

# remove stopwords
stopwords = nltk.corpus.stopwords.words('english')
def remove_stopwords(text):
    word_tokens = word_tokenize(text)
    filtered_tokenized_text = [word for word in word_tokens if word not in␣
↪stopwords]
    filtered_text = TreebankWordDetokenizer().
↪detokenize(filtered_tokenized_text)
    return filtered_text
data3['Body_Cleaned'] = data3['Body_Cleaned'].apply(lambda x:␣
↪remove_stopwords(x))

# apply lemmatisation
lemmatizer = WordNetLemmatizer()
def apply_lemmatisation(text):
    word_tokens = word_tokenize(text)
    filtered_tokenized_text = [lemmatizer.lemmatize(w) for w in word_tokens]
    filtered_text = TreebankWordDetokenizer().
↪detokenize(filtered_tokenized_text)
    return filtered_text
data3['Body_Cleaned'] = data3['Body_Cleaned'].apply(lambda x:␣
↪apply_lemmatisation(x))

# remove numbers
data3['Body_Cleaned'] = data3['Body_Cleaned'].str.replace(r'\d+','', regex=True)
```

4.2.2 Label Sample Dataset

[29]:
```
# import packages
import matplotlib.pyplot as plt # for plotting
from matplotlib.pyplot import figure # for plotting
import numpy as np

# save data to be labelled
data3.to_csv(r'C:/Users/Imran/Desktop/Assignment_3_Repo/Unlabelled_Data.csv',␣
↪index = True)

# read in csv saved previously
model_data  = pd.read_csv(r'C:/Users/Imran/Desktop/Assignment_3_Repo/
↪Annotated_Data.csv')

# plot pie chart of imbalanced dataset
Positive = len(model_data[model_data['Label']==1])
```

```python
Negative = len(model_data[model_data['Label']==2])
Neutral = len(model_data[model_data['Label']==0])
Class = ['Positive', 'Negative', 'Neutral']
Data = [Positive, Negative, Neutral]
explode = (0.1, 0.0, 0.15)
colors = ("#ADD8E6", "#FF7F50", '#AAFF32')
wp = { 'linewidth' : 1, 'edgecolor' : "black" }
def func(pct, allvalues):
    absolute = int(pct / 100.*np.sum(allvalues))
    return "{:.1f}%\n({:d} records)".format(pct, absolute)
fig, ax = plt.subplots(figsize =(10, 7))
wedges, texts, autotexts = ax.pie(Data,
                                  autopct = lambda pct: func(pct, Data),
                                  explode = explode,
                                  labels = Class,
                                  shadow = True,
                                  colors = colors,
                                  startangle = 90,
                                  wedgeprops = wp,
                                  textprops = dict(color ="black"))
ax.legend(wedges, Class,
          title ="Class",
          loc ="center left",
          prop={"size":12},
          bbox_to_anchor = (1, 0, 0.5, 1))

plt.setp(autotexts, size = 12, weight ="bold")
ax.set_title("Class Frequency in Annotated Data")
plt.show()
```



4.2.3 Implementing a Valence Aware Dictionary and sEntiment Reasoner (VADER)

Acquiring Sentiment Scores

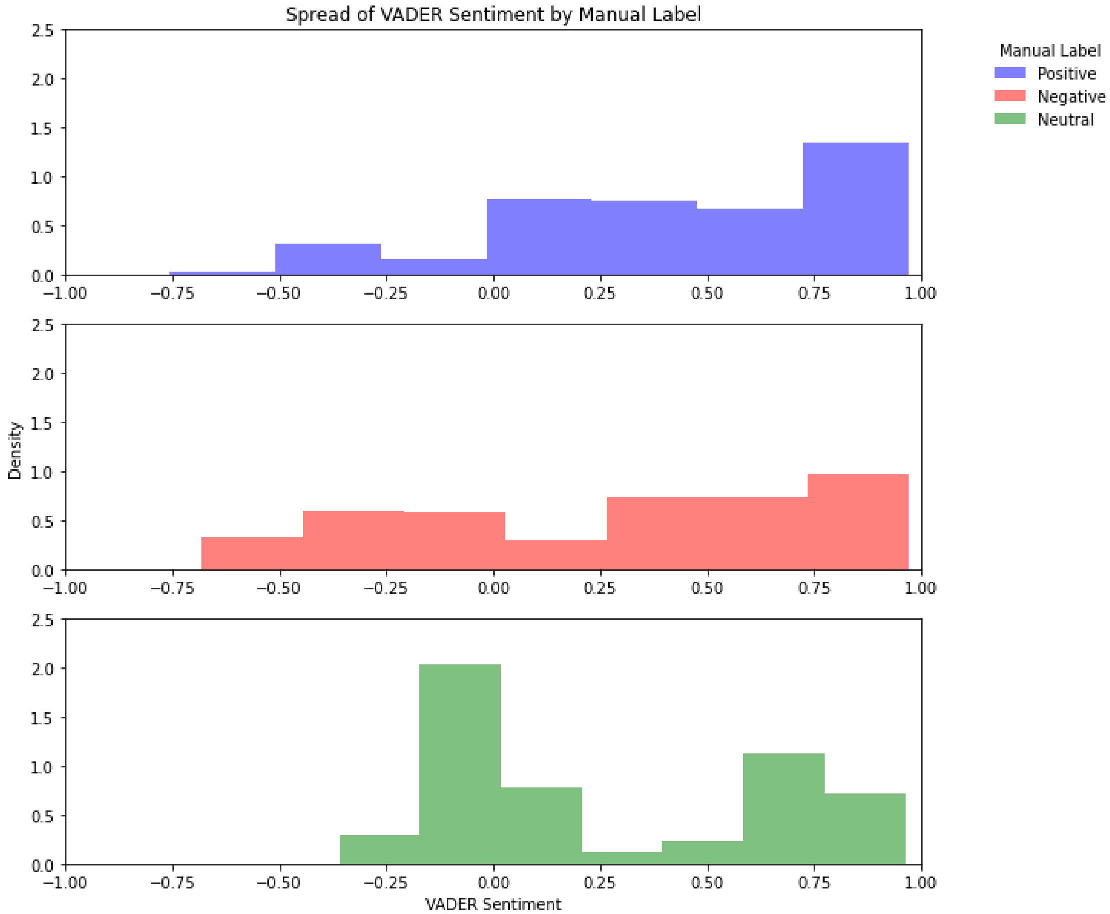
[30]:
```
# import packages
from nltk.sentiment.vader import SentimentIntensityAnalyzer # import nltk for␣
 ↪sentiment analysis
#nltk.download('vader_lexicon')

# call the SentimenetIntensityAnalyser object
analyser = SentimentIntensityAnalyzer()

# define function to return compound sentiment scores
def sentiment_analyzer_scores(sentence):
    score = analyser.polarity_scores(sentence)['compound']
    return score

# get sentimenet for text
model_data['Body_Sentiment'] = model_data['Body_Cleaned'].
 ↪apply(sentiment_analyzer_scores)
```

```
# subset dataframe to assess VADER accuracy
model_data = model_data[['Body_Cleaned', 'Label', 'Body_Sentiment']]

# subset to only include labelled records
model_data = model_data[model_data['Label'].notna()]

# data type conversion
model_data['Label'] = model_data['Label'].astype(int)
```

Converting VADER Sentiment Scores to Discrete Classes

[31]:
```
# define function to return compound sentiment scores
def get_Class(Label):
    if Label == 1:
        return 'Positive'
    if Label == 2:
        return 'Negative'
    else:
        return 'Neutral'

# get sentimenet for text
model_data['Class'] = model_data['Label'].apply(get_Class)

# set bins
bins = 7

# separate dataset into manual label classes
b1 = model_data[model_data['Class'] == 'Positive']
c1 = b1['Body_Sentiment']
b2 = model_data[model_data['Class'] == 'Negative']
c2 = b2['Body_Sentiment']
b3 = model_data[model_data['Class'] == 'Neutral']
c3 = b3['Body_Sentiment']

# define subplots
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax1 = fig.add_subplot(311)
ax2 = fig.add_subplot(312)
ax3 = fig.add_subplot(313)

# Turn off axis lines and ticks of the big subplot
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_color('none')
ax.spines['left'].set_color('none')
ax.spines['right'].set_color('none')
ax.tick_params(labelcolor='w', top=False, bottom=False, left=False, right=False)
```

```
# plot subplots
ax1.hist(c1, bins = bins, color="blue", alpha = 0.5, density = True)[0]
ax2.hist(c2, bins = bins, color="red", alpha = 0.5, density = True)[0]
ax3.hist(c3, bins = bins, color="green", alpha = 0.5, density = True)[0]

# set x and y limits
ax1.set_xlim(-1, 1)
ax1.set_ylim(0, 2.5)
ax2.set_xlim(-1, 1)
ax2.set_ylim(0, 2.5)
ax3.set_xlim(-1, 1)
ax3.set_ylim(0, 2.5)

# Set common labels
ax.set_xlabel('VADER Sentiment')
ax.set_ylabel('Density')
ax1.set_title('Spread of VADER Sentiment by Manual Label')

# Create the legend
line_labels = ["Positive", "Negative", "Neutral"]
fig.legend([ax1, ax2, ax3],
           labels = line_labels,
           loc="upper right",
           bbox_to_anchor = (1.08, 0.88),
           title = "Manual Label",
           frameon = False)

# show plot
plt.show()
```



[32]:
```python
# get VADER label
def get_VADER_label(i):
    if float(i) > 0:
        return 1
    elif float(i) >= -0.1 and float(i) <=0:
        return 0
    else:
        return 2

# get label from VADER sentiment
model_data['VADER Prediction'] = model_data['Body_Sentiment'].
 ↪apply(get_VADER_label)

# data type conversion
model_data['VADER Prediction'] = model_data['VADER Prediction'].astype(int)
```

Assess the Precision and Recall of the VADER Model

[33]:
```
# import packages
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score, f1_score, mean_squared_error # for analysing classification␣
 ↪results

# assess results
target_names = ['Neutral', 'Positive', 'Negative']
print("VADER")
print(classification_report(model_data['Label'], model_data['VADER␣
 ↪Prediction'], target_names = target_names))
```

```
VADER
              precision    recall  f1-score   support

     Neutral       0.45      0.38      0.41        89
    Positive       0.50      0.77      0.61       227
    Negative       0.58      0.24      0.34       184

    accuracy                           0.51       500
   macro avg       0.51      0.47      0.46       500
weighted avg       0.52      0.51      0.48       500
```

4.2.4 Preparing Data for Training New Models

[34]:
```
# import packages
from sklearn.feature_extraction.text import TfidfVectorizer # for generating␣
 ↪TF-IDF matrix
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold, cross_val_score # for model assessment

# split training and test data
train, test = train_test_split(model_data[['Body_Cleaned', 'Label']], test_size␣
 ↪= 0.1, stratify = model_data['Label'], random_state = 101)

# Create feature vectors
vectorizer = TfidfVectorizer(min_df = 5,
                             max_df = 0.8,
                             sublinear_tf = True,
                             use_idf = True)
train_vectors = vectorizer.fit_transform(train['Body_Cleaned'])
test_vectors = vectorizer.transform(test['Body_Cleaned'])
```

4.2.5 Implementing a Weighted, Multinomial Logistic Regression Model

Logistic Regression Model Hyperparamter Tuning

[35]:

```
# import packages
from sklearn.linear_model import LogisticRegression # for logistic regression␣
↪modelling
import pickle # for saving and loading best model from hyperparameter tuning
from tabulate import tabulate # for printing results in a tabular format

# define estimator
LR_estimator = LogisticRegression(multi_class = 'multinomial',
                                  class_weight='balanced',
                                  penalty = 'l2',
                                  random_state = 101)

# define range of parameters to optimise the estimator
LR_parameters = {'solver': ['lbfgs', 'sag', 'saga', 'newton-cg'],
                 'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# define function for optimising the estimator estimator
def optimised_LR_model(estimator, parameters):
    grid_search = GridSearchCV(estimator= estimator,
                               param_grid = parameters,
                               scoring = 'f1_macro',
                               cv = 10,
                               return_train_score = True,
                               verbose=True)

    grid_search.fit(train_vectors, train['Label'])

    optimised_LR_model = grid_search.best_estimator_

    pickle.dump(optimised_LR_model, open('optimised_LR_model.sav', 'wb'))

    # print results to check for overfitting
    #print("params", grid_search.cv_results_["params"])
    #print("best_estimator_", optimised_LR_model)

    data = [
    ["1",
     str(round(grid_search.cv_results_["split1_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split1_test_score"][-1]*100,1)) + '%'],
    ["2",
     str(round(grid_search.cv_results_["split2_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split2_test_score"][-1]*100,1)) + '%'],
    ["3",
     str(round(grid_search.cv_results_["split3_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split3_test_score"][-1]*100,1)) + '%'],
    ["4",
     str(round(grid_search.cv_results_["split4_train_score"][-1]*100,1)) + "%",
```

```
     str(round(grid_search.cv_results_["split4_test_score"][-1]*100,1)) + '%'],
    ["5",
     str(round(grid_search.cv_results_["split5_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split5_test_score"][-1]*100,1)) + '%'],
    ["Mean \u00B1 Std Dev",
     str(round((grid_search.cv_results_["mean_train_score"][-1])*100,1)) + "%␣
↪\u00B1 " \
     + str(round((grid_search.cv_results_["std_train_score"][-1])*100,1)) + "%",
     str(round((grid_search.cv_results_["mean_test_score"][-1])*100,1)) + "%␣
↪\u00B1 " \
     + str(round((grid_search.cv_results_["std_test_score"][-1])*100,1)) + "%"]]
    #print(tabulate(data, headers=["Split", "Train", "Test"]))

    with open("LR_output.txt", "a") as LR_output:
        print(tabulate(data, headers=["K-Fold Split", "Train", "Test"]), file =␣
↪LR_output)

## run function
#optimised_LR_model(LR_estimator, LR_parameters)
LR_output = open(r'C:/Users/Imran/Desktop/Assignment_3_Repo/LR_output.txt', "r")
print("LR Model")
print(LR_output.read())
```

```
LR Model
K-Fold Split    Train         Test
--------------  ------------  ------------
1               97.1%         39.8%
2               96.6%         66.7%
3               96.7%         68.1%
4               97.3%         70.4%
5               97.0%         73.8%
Mean ± Std Dev  96.9% ± 0.3%  64.4% ± 9.4%
```

Make Predictions using Logistic Regression Model

[36]:
```
# load best estimator
optimised_LR_model = pickle.load(open('optimised_LR_model.sav', 'rb'))

# fit model for prediction making
optimised_LR_model.fit(train_vectors, train['Label'])

# make predictions
LR_y_pred = optimised_LR_model.predict(test_vectors)
```

Assess the Precision and Recall of the Logistic Regression Model

[37]:
```
# assess results
target_names = ['Neutral', 'Positive', 'Negative']
```

```
print("LR Model")
print(classification_report(test['Label'], LR_y_pred, target_names =␣
 ↪target_names))
```

```
LR Model
              precision    recall  f1-score   support

     Neutral       0.56      0.56      0.56         9
    Positive       0.67      0.78      0.72        23
    Negative       0.86      0.67      0.75        18

    accuracy                           0.70        50
   macro avg       0.69      0.67      0.68        50
weighted avg       0.72      0.70      0.70        50
```

4.2.6 Implementing a Weighted, Multiclass Support Vector Machine Model

Support Vector Machine Model Hyperparamter Tuning

[38]:
```
# import packages
from sklearn.multiclass import OneVsRestClassifier #for handling multiclass␣
 ↪labels
from sklearn.svm import SVC # for support vector machine modelling

# define nested estimator
SVM_estimator = OneVsRestClassifier(SVC(class_weight='balanced', random_state =␣
 ↪101))

# define range of parameters to optimise the estimator
SVM_parameters = {'estimator__kernel': ['poly', 'rbf', 'sigmoid'],
                 'estimator__C': [0.1, 1, 1],
                 'estimator__gamma': [0.001, 0.01, 0.1]}

# define function for optimising the estimator estimator
def optimised_SVM_model(estimator, parameters):
    grid_search = GridSearchCV(estimator= estimator,
                               param_grid = parameters,
                               scoring = 'f1_macro',
                               cv = 10,
                               return_train_score = True,
                               verbose=True)

    grid_search.fit(train_vectors, train['Label'])

    optimised_SVM_model = grid_search.best_estimator_

    pickle.dump(optimised_SVM_model, open('optimised_SVM_model.sav', 'wb'))
```

```
    # print results to check for overfitting
    #print("params", grid_search.cv_results_["params"])
    #print("best_estimator_", optimised_SVM_model)

    data = [
    ["1",
     str(round(grid_search.cv_results_["split1_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split1_test_score"][-1]*100,1)) + '%'],
    ["2",
     str(round(grid_search.cv_results_["split2_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split2_test_score"][-1]*100,1)) + '%'],
    ["3",
     str(round(grid_search.cv_results_["split3_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split3_test_score"][-1]*100,1)) + '%'],
    ["4",
     str(round(grid_search.cv_results_["split4_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split4_test_score"][-1]*100,1)) + '%'],
    ["5",
     str(round(grid_search.cv_results_["split5_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split5_test_score"][-1]*100,1)) + '%'],
    ["Mean \u00B1 Std Dev",
     str(round((grid_search.cv_results_["mean_train_score"][-1])*100,1)) + "%␣
 ↪\u00B1 " \
      + str(round((grid_search.cv_results_["std_train_score"][-1])*100,1)) + "%",
     str(round((grid_search.cv_results_["mean_test_score"][-1])*100,1)) + "%␣
 ↪\u00B1 " \
      + str(round((grid_search.cv_results_["std_test_score"][-1])*100,1)) + "%"]]
    #print(tabulate(data, headers=["Split", "Train", "Test"]))

    with open("SVM_output.txt", "a") as SVM_output:
        print(tabulate(data, headers=["K-Fold Split", "Train", "Test"]), file =␣
 ↪SVM_output)

## run function
#optimised_SVM_model(SVM_estimator, SVM_parameters)
SVM_output = open(r'C:/Users/Imran/Desktop/Assignment_3_Repo/SVM_output.txt',␣
 ↪"r")
print("SVM Model")
print(SVM_output.read())
```

```
SVM Model
K-Fold Split    Train         Test
--------------  ------------  ------------
1               64.7%         36.7%
2               65.9%         68.3%
3               63.0%         64.0%
4               64.0%         47.8%
```

```
5                63.7%          61.8%
Mean ± Std Dev   64.9% ± 1.9%   54.6% ± 9.4%
```

Make Predictions using Support Vector Machine Model

[39]:
```
# load best estimator
optimised_SVM_model = pickle.load(open('optimised_SVM_model.sav', 'rb'))

# fit model for prediction making
optimised_SVM_model.fit(train_vectors, train['Label'])

# make predictions
SVM_y_pred = optimised_SVM_model.predict(test_vectors)
```

Assess the Precision and Recall of the Support Vector Machine Model

[40]:
```
# assess results
print("SVM Model")
print(classification_report(test['Label'], SVM_y_pred, target_names =␣
↪target_names))
```

```
SVM Model
              precision    recall  f1-score   support

     Neutral       0.45      0.56      0.50         9
    Positive       0.74      0.61      0.67        23
    Negative       0.65      0.72      0.68        18

    accuracy                           0.64        50
   macro avg       0.61      0.63      0.62        50
weighted avg       0.65      0.64      0.64        50
```

4.2.7 Implementing a Multinomial Naive Bayes Model

Multinomial Naive Bayes Model Hyperparamter Tuning

[41]:
```
# import packages
from sklearn.naive_bayes import MultinomialNB # for multinomial naive bayes␣
↪modelling

# define nested estimator
NB_estimator = OneVsRestClassifier(MultinomialNB())

# define range of parameters to optimise the estimator
NB_parameters = {'estimator__alpha': [0, 0.25, 0.5, 0.75, 1]}

# define function for optimising the estimator estimator
def optimised_NB_model(estimator, parameters):
    grid_search = GridSearchCV(estimator= estimator,
```

```
                                param_grid = parameters,
                                scoring = 'f1_macro',
                                cv = 10,
                                return_train_score = True,
                                verbose=True)

    grid_search.fit(train_vectors, train['Label'])

    optimised_NB_model = grid_search.best_estimator_

    pickle.dump(optimised_NB_model, open('optimised_NB_model.sav', 'wb'))

    # print results to check for overfitting
    #print("params", grid_search.cv_results_["params"])
    #print("best_estimator_", optimised_NB_model)

    data = [
    ["1",
     str(round(grid_search.cv_results_["split1_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split1_test_score"][-1]*100,1)) + '%'],
    ["2",
     str(round(grid_search.cv_results_["split2_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split2_test_score"][-1]*100,1)) + '%'],
    ["3",
     str(round(grid_search.cv_results_["split3_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split3_test_score"][-1]*100,1)) + '%'],
    ["4",
     str(round(grid_search.cv_results_["split4_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split4_test_score"][-1]*100,1)) + '%'],
    ["5",
     str(round(grid_search.cv_results_["split5_train_score"][-1]*100,1)) + "%",
     str(round(grid_search.cv_results_["split5_test_score"][-1]*100,1)) + '%'],
    ["Mean \u00B1 Std Dev",
     str(round((grid_search.cv_results_["mean_train_score"][-1])*100,1)) + "%␣
↪\u00B1 " \
      + str(round((grid_search.cv_results_["std_train_score"][-1])*100,1)) + "%",
     str(round((grid_search.cv_results_["mean_test_score"][-1])*100,1)) + "%␣
↪\u00B1 " \
      + str(round((grid_search.cv_results_["std_test_score"][-1])*100,1)) + "%"]]
    #print(tabulate(data, headers=["Split", "Train", "Test"]))

    with open("NB_output.txt", "a") as NB_output:
        print(tabulate(data, headers=["K-Fold Split", "Train", "Test"]), file =␣
↪NB_output)

## run function
#optimised_NB_model(NB_estimator, NB_parameters)
```

```
NB_output = open(r'C:/Users/Imran/Desktop/Assignment_3_Repo/NB_output.txt', "r")
print("NB Model")
print(NB_output.read())
```

```
NB Model
K-Fold Split    Train          Test
--------------  ------------   ------------
1               63.1%          41.0%
2               66.5%          71.7%
3               63.8%          56.5%
4               64.8%          49.1%
5               64.4%          61.5%
Mean ± Std Dev  64.7% ± 1.7%   54.0% ± 8.9%
```

Make Predictions using Multinomial Naive Bayes Model

[42]:
```
# load best estimator
optimised_NB_model = pickle.load(open('optimised_NB_model.sav', 'rb'))

# fit model for prediction making
optimised_NB_model.fit(train_vectors, train['Label'])

# make predictions
NB_y_pred = optimised_NB_model.predict(test_vectors)
```

Assess the Precision and Recall of the Multinomial Naive Bayes Model

[43]:
```
# assess results
print("NB Model")
print(classification_report(test['Label'], NB_y_pred, target_names =␣
 ↪target_names))
```

```
NB Model
              precision    recall  f1-score   support

     Neutral       1.00      0.44      0.62         9
    Positive       0.61      0.87      0.71        23
    Negative       0.69      0.50      0.58        18

    accuracy                           0.66        50
   macro avg       0.77      0.60      0.64        50
weighted avg       0.71      0.66      0.65        50
```

4.3 Analysis

4.3.1 Data Processing

[44]:
```python
# create new dataframe
data4 = data3

# transform all records using previously fitted vectorizer
data_vectors = vectorizer.transform(data4['Body_Cleaned'])

# make predictions
y_pred = optimised_NB_model.predict(data_vectors)

# change to dataframe
y_pred = pd.DataFrame(y_pred, columns = ['Pred'])

# join predictions to dataframe
data4 = pd.concat([data4, y_pred], axis=1)

# define baseline count
data4['Count'] = 1

# weight counts by label
def get_weights(i):
    if str(i) == "1":
        return 1
    elif str(i) == "2":
        return -1
    else:
        return 0
data4['Weight'] = data4['Pred'].apply(get_weights)

# calculate sentiment score from best model
data4['Sentiment_Score'] = data4.Count * data4.Weight

# make new dataset
data5 = data4

# convert the 'Date' column to datetime format
data5['Date_Transformed'] = data5['Date_Transformed'].astype('datetime64[ns]')

# subset columns of dataframe and the data into key places
data5_NSW = data5[data5['Place_Tag'] == 'NSW'][['Date_Transformed',␣
↪'Sentiment_Score']]
data5_QLD = data5[data5['Place_Tag'] == 'QLD'][['Date_Transformed',␣
↪'Sentiment_Score']]
data5_SA = data5[data5['Place_Tag'] == 'SA'][['Date_Transformed',␣
↪'Sentiment_Score']]
```

```
data5_TAS = data5[data5['Place_Tag'] == 'TAS'][['Date_Transformed',␣
 ↪'Sentiment_Score']]
data5_VIC = data5[data5['Place_Tag'] == 'VIC'][['Date_Transformed',␣
 ↪'Sentiment_Score']]
data5_WA = data5[data5['Place_Tag'] == 'WA'][['Date_Transformed',␣
 ↪'Sentiment_Score']]
data5_NT = data5[data5['Place_Tag'] == 'NT'][['Date_Transformed',␣
 ↪'Sentiment_Score']]
data5_ACT = data5[data5['Place_Tag'] == 'ACT'][['Date_Transformed',␣
 ↪'Sentiment_Score']]

# group by date in a regular format
data5_NSW = data5_NSW.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_QLD = data5_QLD.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_SA = data5_SA.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_TAS = data5_TAS.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_VIC = data5_VIC.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_WA = data5_WA.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_NT = data5_NT.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()
data5_ACT = data5_ACT.resample('d', on='Date_Transformed')[['Sentiment_Score']].
 ↪mean()

# sort dataframe by date to compute cumulative compound sentiment scores over␣
 ↪time
data5_NSW = data5_NSW.sort_values(by = 'Date_Transformed', ascending = True)
data5_QLD = data5_QLD.sort_values(by = 'Date_Transformed', ascending = True)
data5_SA = data5_SA.sort_values(by = 'Date_Transformed', ascending = True)
data5_TAS = data5_TAS.sort_values(by = 'Date_Transformed', ascending = True)
data5_VIC = data5_VIC.sort_values(by = 'Date_Transformed', ascending = True)
data5_WA = data5_WA.sort_values(by = 'Date_Transformed', ascending = True)
data5_NT = data5_NT.sort_values(by = 'Date_Transformed', ascending = True)
data5_ACT = data5_ACT.sort_values(by = 'Date_Transformed', ascending = True)

# missing value imputation
data5_NSW = data5_NSW.fillna(0.0)
data5_QLD = data5_QLD.fillna(0.0)
data5_SA = data5_SA.fillna(0.0)
data5_TAS = data5_TAS.fillna(0.0)
data5_VIC = data5_VIC.fillna(0.0)
```

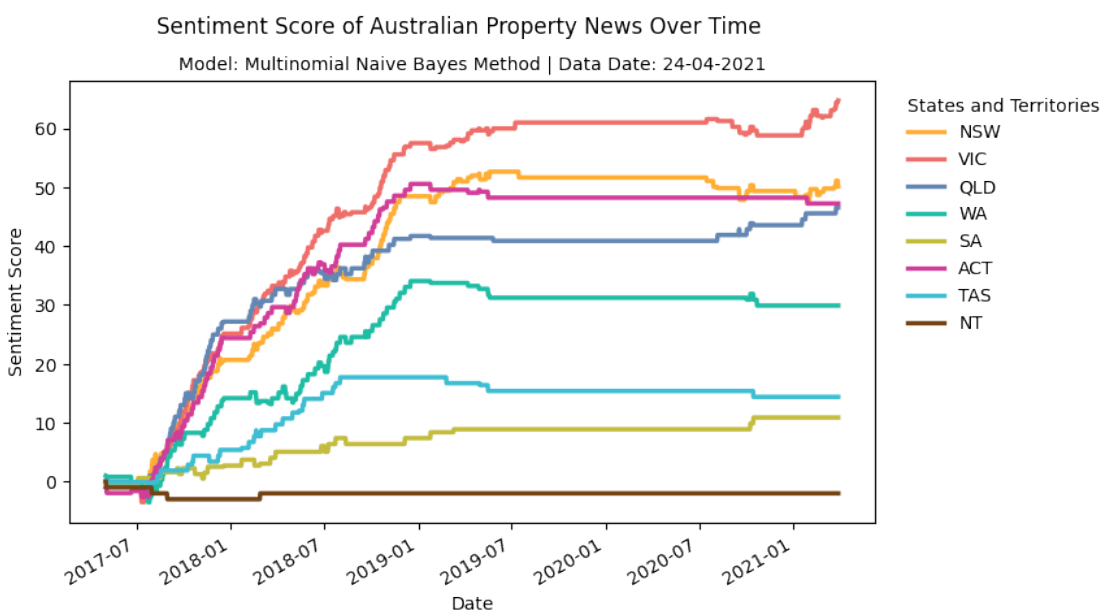
```
data5_WA = data5_WA.fillna(0.0)
data5_NT = data5_NT.fillna(0.0)
data5_ACT = data5_ACT.fillna(0.0)

# merge dataframes together
data5 = pd.concat([data5_NSW, data5_QLD, data5_SA, data5_TAS, data5_VIC,␣
↪data5_WA, data5_NT, data5_ACT], axis = 1)

# rename columns
data5.columns = ['Sentiment_Score_NSW', 'Sentiment_Score_QLD',␣
↪'Sentiment_Score_SA', 'Sentiment_Score_TAS',
                'Sentiment_Score_VIC', 'Sentiment_Score_WA',␣
↪'Sentiment_Score_NT', 'Sentiment_Score_ACT']

# missing value imputation for body sentiment after merge for missing weeks
data5 = data5.fillna(0.0)
```

4.3.2 Plot Cumulative Sums of Compound Sentiments over Time

[45]:
```
# specify start date
start_date = '2016-01-01'
# specify end date
end_date = '2021-12-01'

# define function for calculating cumulative sums
def plot_cumulative_sum(data5, start_date, end_date):

    # put in start date caps
    if pd.to_datetime(start_date) < data5.index[0]:
        start_date = data5.index[0]
    else:
        start_date = start_date

    # put in end date caps
    if pd.to_datetime(end_date) > data5.index[-1]:
        end_date = data5.index[-1]
    else:
        end_date = end_date

    # filter dataframe on dates
    data6 = data5.loc[start_date:end_date]

    # calculate cumulative sums of compound sentiment over time
    data6['Body_Cumulative_NSW'] = data6['Sentiment_Score_NSW'].cumsum()
    data6['Body_Cumulative_QLD'] = data6['Sentiment_Score_QLD'].cumsum()
    data6['Body_Cumulative_SA'] = data6['Sentiment_Score_SA'].cumsum()
    data6['Body_Cumulative_TAS'] = data6['Sentiment_Score_TAS'].cumsum()
```

```
data6['Body_Cumulative_VIC'] = data6['Sentiment_Score_VIC'].cumsum()
data6['Body_Cumulative_WA'] = data6['Sentiment_Score_WA'].cumsum()
data6['Body_Cumulative_NT'] = data6['Sentiment_Score_NT'].cumsum()
data6['Body_Cumulative_ACT'] = data6['Sentiment_Score_ACT'].cumsum()

# missing value imputation at the state level for missing weeks
data6 = data6.fillna(method='ffill')

# specify fig size and dpi
figure(figsize=(8, 5), dpi=100)

# make up some data
x = data6.index
y1 = data6['Body_Cumulative_NSW']
y2 = data6['Body_Cumulative_VIC']
y3 = data6['Body_Cumulative_QLD']
y4 = data6['Body_Cumulative_WA']
y5 = data6['Body_Cumulative_SA']
y6 = data6['Body_Cumulative_ACT']
y7 = data6['Body_Cumulative_TAS']
y8 = data6['Body_Cumulative_NT']

# plot
plt.plot(x, y1, linewidth = 2.5, color = '#ffae34', label = 'NSW')
plt.plot(x, y2, linewidth = 2.5, color = '#ef6e6a', label = 'VIC')
plt.plot(x, y3, linewidth = 2.5, color = '#6387b4', label = 'QLD')
plt.plot(x, y4, linewidth = 2.5, color = '#1fbda5', label = 'WA')
plt.plot(x, y5, linewidth = 2.5, color = '#c3bc3f', label = 'SA')
plt.plot(x, y6, linewidth = 2.5, color = '#d23d99', label = 'ACT')
plt.plot(x, y7, linewidth = 2.5, color = '#3dbed2', label = 'TAS')
plt.plot(x, y8, linewidth = 2.5, color = '#734314', label = 'NT')

# beautify the x-labels
plt.gcf().autofmt_xdate()

# set title
Data_Date = data2['Data_Date'].iloc[0]
plt.title('Model: Multinomial Naive Bayes Method | Data Date: {}'.
↪format(Data_Date), fontsize = 10)
plt.suptitle('Sentiment Score of Australian Property News Over Time',␣
↪fontsize = 12)

# control tick frequency
#plt.yticks(np.arange(0, 100, 20))

# change font size
plt.xlabel('Date', fontsize = 10)
```

```
    plt.ylabel('Sentiment Score', fontsize = 10)

    # set legend
    leg = plt.legend(loc="upper right", bbox_to_anchor = (1.3, 1), title =␣
↪'States and Territories', frameon = False)
    leg._legend_box.align = "left"

    # control tick frequency
    #plt.yticks(np.arange(0, 100, 20))

    # change font size
    plt.xlabel('Date', fontsize = 10)
    plt.ylabel('Sentiment Score', fontsize = 10)

    # show plot
    plt.show()

# call the function
plot_cumulative_sum(data5, start_date, end_date)
```



5 References

Artiles, A. (2017). Using VADER to handle sentiment analysis with social media text. Retrieved from: https://tredactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with-social-media-text.html

Brownle, J. (2020). One-vs-Rest and One-vs-One for Multi-Class Classification. Retrieved from: https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

Chahira, L., Anis, Z. & Mounir, Z. (2017). A Rule-based Named Entity Extraction Method and Syntactico-Semantic Annotation for Arabic Language. IARIA, 63-69.

Devi, G. & Kamalakkannan, S. (2020). Literature Review on Sentiment Analysis in Social Media: Open Challenges toward Applications. International Journal of Advanced Science and Technology. *Volume* - 29, 1462-1471.

Devika, M. & Ganesh, A. (2016). Sentiment Analysis: A Comparative Study On Different Approaches. Procedia Computer Science. *Volume* - 87, 44-49.

GeeksforGeeks. (2020). Rule-Based Classifier – Machine Learning. Retrieved from: https://www.geeksforgeeks.org/rule-based- classifier-machine-learning/

Heidenreich, H. (2018). Stemming? Lemmatization? What?. Retrieved from: https://towardsdatascience.com/stemming- lemmatization-what-ba782b7c0bd8

Horev, R. (2018). BERT Explained: State of the art language model for NLP. Retrieved from: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Hui, J. (2019). How to scale the BERT Training with Nvidia GPUs? Retrieved from: https://medium.com/nvidia-ai/how-to-scale-the-bert-training-with-nvidia-gpus-c1575e8eaf71#:~:text=For%20most%20of%20the%20fine,and%20later%20combine%20the%20results

Isabelle, G., Maharani, W. & Asror, I. (2018). Analysis on Opinion Mining Using Combining Lexicon-Based Method and Multinomial Naive Bayes. International Conference on Industrial Enterprise and System Engineering. *Volume* - 2, 1-6.

Jack, M. (2019). NLP: Pretrained Named Entity Recognition (NER). Retrieved from: https://medium.com/@b.terryjack/nlp-pretrained-named-entity-recognition-7caa5cd28d7b

Li, J., Sun, A., Han, J. & Li, C. (2020). A Survey on Deep Learning for Named Entity Recognition. IEEE Transactions on Knowledge and Data Engineering, 1-15.

Li, M., Yang, Q., He, F., Li, Z., Zho, P., Zhao, L. & Chen, Z. (2019). An Unsupervised Learning Approach for NER Based on Online Encyclopedia. LNCS, *Volume* - 11641.

Mahapatra, S. (2018). Why Deep Learning over Traditional Machine Learning?. Retrieved from: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063#:~:text=The%20biggest%20advantage%20Deep%20Learning,and%20hard%20core%20feature%20extraction

Molla, M. (2018). Machine Learning: Sentiment analysis of movie reviews using Logistic Regression. Retrieved from: https://itnext.io/machine-learning-sentiment-analysis-of-movie-reviews-using-logisticregression-62e9622b4532

Pajupuu, H., Altrov, R. & Pajupuu, J. (2016). Identifying Polarity in Different Text Types. Folklore (Estonia). *Volume* - 64, 125-142.

Rajasekharan, A. (2020). Unsupervised NER using BERT. Retrieved from: https://towardsdatascience.com/unsupervised-ner-using-bert-2d7af5f90b8a

Scikit Learn. (2021). sklearn.linear_model.LogisticRegression. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Scikit Learn. (2021). sklearn.svm.SVC. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Stack Exchange. (2019). How to evaluate whether model is overfitting or underfitting when using cross_val_score and GridSearchCV? Retrieved from: https://stats.stackexchange.com/questions/439485/how-to-evaluate-whether-model-is-overfitting-or-underfitting-when-using-cross-va

Tadagoppula, S. (2020). Understanding Machine Learning Algorithms — Naive Bayes. Retrieved from: https://medium.com/analytics-vidhya/understanding-machine-learning-algorithms-naive-bayes-808ed649c1ec

Thanaki, J. (2017). Python Natural Language Processing: Advanced machine learning and deep learning techniques for natural language processing. Packt.

Toral, A. (2015). Machine Learning in Natural Language Processing. Retrieved from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.696.9771&rep=rep1&type=pdf

Vilariño, J. (2020). What's New in Data Anonymization and NER? Retrieved from: https://www.acclaro.com/blog/whats-new-in-data-anonymization-and-ner/

[ ]:

```

```