Conclusion

Chin Man KWAN

January 10, 2022

Contents

Chapter 1

examiner: Aragorn II Elessar # only required if PhD; otherwise delete or comment out

Placeholder

1.1 Metadata

1.2 Front Matter

1.3 Bibliography

1.4 PDF Formatting

1.5 Output Options

Chapter 2

Introduction

Placeholder

Chapter 3

Literature Review

Placeholder

3.1 Theories

3.1.1 What is national identity?

3.1.2 Impact of conceptions of national identity on attitudes towards immigrants

3.1.3 Chineseness in Hong Kong

3.2 Empirics

3.2.1 How do different political camps in Hong Kong perceive national identity?

The pro-Beijing camp

The pro-Democracy camp (or pan-democrats)

The localist camp

3.2.2 Asylum seekers in Hong Kong

Hong Kong's policy regime

Responses by political camps

Chapter 4

Methods

Placeholder

4.1 Hypotheses

4.2 Data collection

4.3 Modelling

4.4 Operationalisation

4.4.1 Main independent variable

4.4.2 Dependent variable

4.4.3 Control variables

```
library(reticulate)
```

```
import warnings

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import jieba

import zhon.hanzi

import re
```

```
from itertools import chain


# Setting options for display and random seed

np.random.seed(1)

pd.set_option('display.max_columns', None)

sns.set_theme(context='paper')

warnings.filterwarnings(category=FutureWarning, action = 'ignore')
```

```
news_df = pd.read_csv("Coding/asylum_seekers_articles_final.csv")

num_of_articles = news_df.shape[0]

num_of_newspapers = news_df.Newspaper.nunique()
```

Chapter 4

Results

After searching for articles which contain at least one of the keywords and are relevant to asylum seekers residing in Hong Kong, there were in total 557 articles published in 2019 by 16 newspapers. In this section, I will first explore the data set preliminarily, and then move onto sentiment analysis with machine learning models to find out whether the political camp of media outlets is associated with the polarity of the news articles towards asylum seekers.

4.1 Exploratory data analysis (EDA)

4.1.1 How does the number of news articles vary by political camps and month?

Starting with the number of articles by media outlets as shown in the left plot of figure 4.1, consistent with the study by Ng, Choi, and Chan (2019), Oriental Daily News continues to be the media outlet covering the most frequently on asylum seekers with 384 (or 68.94%) articles throughout 2019. By contrast, the second-most frequent publisher *Sing Tao Daily* only had 45 entries (or 8.08%) of the total number of articles published. Each of the other newspaper outlets only constituted to a small portion of news articles about non-refoulement claimants in 2019. Therefore, the issue of asylum seekers in Hong Kong still appeared to be the most salient for Oriental Daily News by 2019, evidenced by its unmatched volume of articles related to this topic vis-a-vis other media outlets.

On a higher level of political stance, the right plot of figure 4.1 indicates that largely due to the huge volume of articles by Oriental Daily News, the pro-Beijing camp dominated the coverage of asylum seekers in Hong Kong in 2019. Meanwhile, both neutral and pro-democracy newspaper outlets published similar amounts of articles throughout 2019, and both camps constituted to small proportions of the share of articles during the year. Even if we omitted the sheer volume of articles published by

Oriental Daily News, the pro-Beijing media would still have 173 articles published altogether which was still considerably more than the quantity of articles by neutral and pro-democracy media outlets.

```
fig, axes = plt.subplots(1,2)


# Plotting by-outlet amount of articles
articles_outlet = news_df.value_counts('Newspaper')
sns.barplot(x=articles_outlet, y=articles_outlet.index, ax=axes[0])
axes[0].set(xlabel='Number of articles (in base-10 log scale)', ylabel='', title='By newspaper', xscal
for idx, value in enumerate(articles_outlet):
    axes[0].text(value + 5, idx + 0.2, value)


# Plotting by-camp amount of articles
sns.countplot(x='Political_camp', data=news_df, ax=axes[1])
axes[1].set(xlabel='', ylabel='Number of articles', title='By political camp')
count_by_political_camp = news_df['Political_camp'].value_counts(sort=False)
for idx, value in enumerate(count_by_political_camp):
  axes[1].text(idx - 0.1, value + 5, value)
axes[1].set_xticklabels(['Pro-Beijing', 'Neutral', 'Pro-democracy'], rotation=60)
plt.tight_layout()
plt.show()
```

```
plt.clf()
```

Lastly, it will also be intriguing to see how the number of articles might vary by month in 2019. As noted before, the anti-extradition law protest lasted mostly from June to November when numerous large-scale clashes between protesters and the police occurred. From figure 4.2, it appears that coincidentally, there were the fewest amounts of articles about asylum seekers published between August and November when some of the most intense clashes (notably the *siege* of the Hong Kong Polytechnic University in November 2019) took place. Although investigating whether the number of news articles

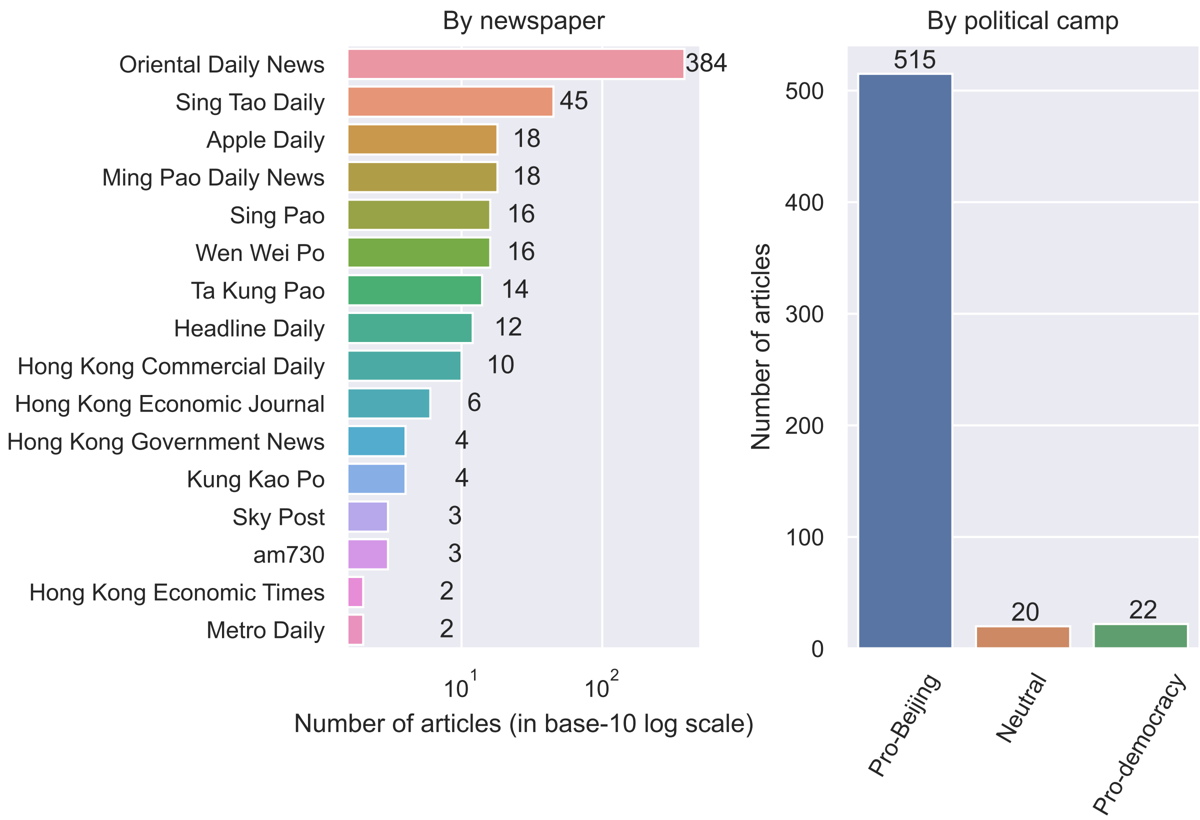


Figure 4.1: News articles on asylum seekers in 2019 by news outlet (left) and political camp (right)

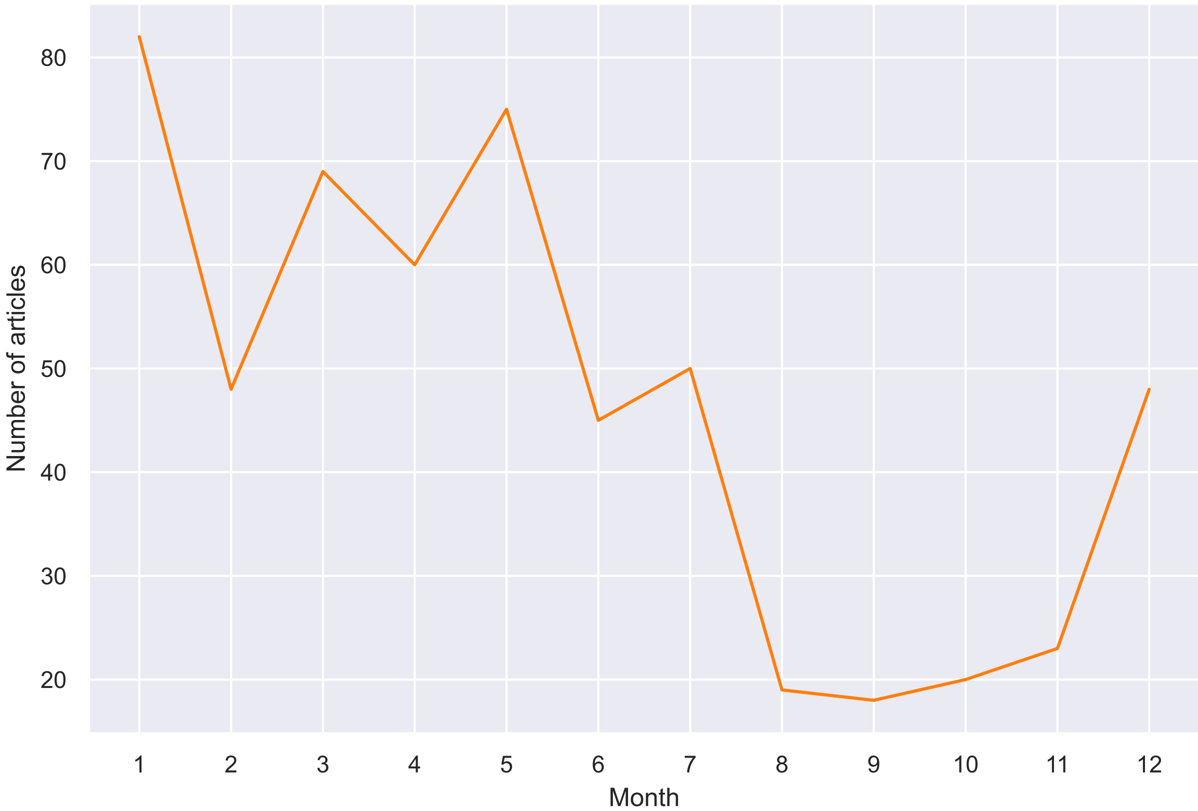


Figure 4.2: Temporal patterns of the publication of news articles about asylum seekers in Hong Kong in 2019

about asylum seekers may be correlated with that about the anti-extradition law protest would be out of the scope of this paper, this could be a research question to be pursued in another occasion.

```
articles_by_month = news_df.Month.value_counts(sort=False)
ax = sns.lineplot(x=articles_by_month.index, y=articles_by_month, color='tab:orange')
ax.set(xlabel='Month', ylabel='Number of articles', xticks=np.arange(1,13))
plt.tight_layout()
plt.show()
```

```
plt.clf()
```

In short, the majority of news articles about non-refoulement claimants in Hong Kong in 2019 were published by pro-Beijing media outlets, of which a huge proportion was from Oriental Daily

News. Moreover, the number of articles by month was the lowest from August to November when the anti-extradition law witnessed some of the most large-scale and intense clashes.

4.1.2 Polarities of the news articles

According to table 4.1, the polarity of the news articles about asylum seekers in Hong Kong in 2019 tilted towards negative, since only around 4.3% and 23.5% of articles respectively depicted asylum seekers positively and neutrally. The fact that the sentiment of the news articles in 2019 was skewed towards negativity implies that I will need to take class imbalance into account for modelling later. Political-camp-wise, pro-Beijing media outlets had over 70% of its articles depicting asylum seekers in Hong Kong in negative lights, whereas neutral and pro-democracy media outlets had their reportage evenly spread between neutral and positive articles (albeit they altogether constituted to only a small proportion of the total number of articles in 2019). While H_1 shall be tested formally with machine learning models after including other control variables later, preliminary evidence suggests that the polarities of the news articles vary with the political camp that the outlets belong to.

```
sentiment_camp = pd.crosstab(news_df.Political_camp, news_df.Sentiment, margins=True)
sentiment_camp.columns = ["Negative", "Neutral", "Positive", "All"]
```

```
knitr::kable(py$sentiment_camp, digits = 4, caption="Polarities of the news articles on asylum seekers
```

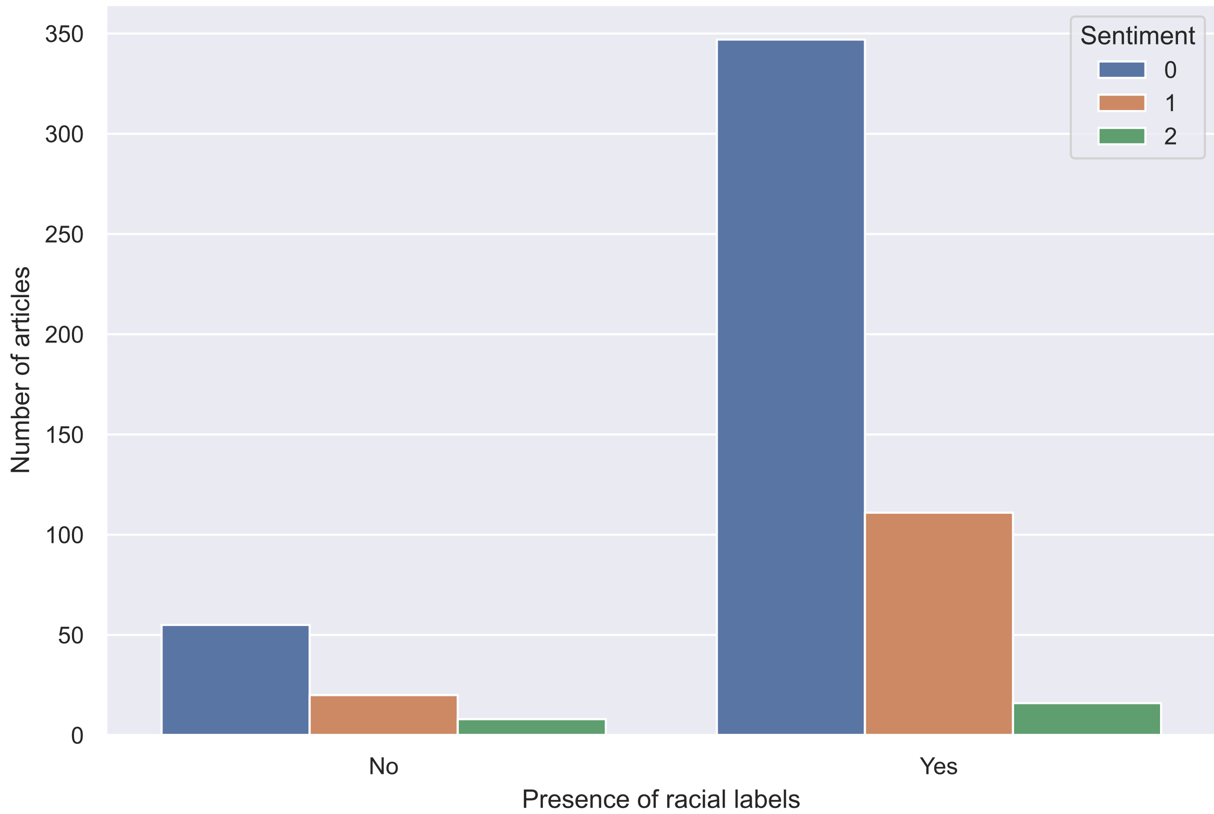
Table 4.1: Polarities of the news articles on asylum seekers in Hong Kong in 2019

	Negative	Neutral	Positive	All
Neutral	0	12	8	20
Pro-Beijing	402	108	5	515
Pro-democracy	0	11	11	22
All	402	131	24	557

4.1.3 Presence of racial labels

Given the majority of asylum seekers in Hong Kong being non-ethnic Chinese, it will also be worth glimpsing whether the presence of racial labels for describing asylum seekers is associated with the sentiment of the news articles. Judging from figure 4.3 preliminarily, however, it appears that the patterns of the polarities are quite similar whether news articles contain racial labels or not, namely, most of the articles framed non-refoulement claimants negatively, and then some reported on events about this group of population neutrally, and finally only a small amount of articles were favourable towards asylum seekers residing in the city. In any case, the machine learning model can add the presence of racial labels as a control variable to test this potential association more formally later.

```
ax = sns.countplot(x="Racial_label", hue="Sentiment", data=news_df)
ax.set(xlabel="Presence of racial labels", ylabel="Number of articles")
ax.set_xticklabels(["No", "Yes"])
plt.show()
```




```
plt.clf()
```

4.1.4 Character lengths of news articles and titles

Lastly, let's look at the distribution of the character lengths of the titles and main texts of the news articles. According to figure 4.4 and table 4.2, it appears that both the title and main text lengths have right-skewed distributions. In other words, while most of the news articles on asylum seekers in Hong Kong in 2019 had relatively short titles and/or main texts, a few of them were considerably more verbose than the rest of the articles.

```
fig, axes = plt.subplots(1, 2)

# Plotting distribution of title word count
sns.histplot(x='Title_length', data=news_df, ax=axes[0], color='tab:blue', alpha=0.5)
axes[0].set(xlabel='Word count', title='Article title')
mean_title_length = news_df.Title_length.mean()
axes[0].axvline(mean_title_length, alpha=0.5, linestyle = '-.', c='black', label='Mean of title length
axes[0].legend()

# Plotting distribution of article word count
sns.histplot(x='Raw_article_length', data=news_df, ax=axes[1], color='tab:orange', alpha=0.5)
axes[1].set(xlabel='Word count (in thousands)', title='Raw article text')
mean_article_length = news_df.Raw_article_length.mean()
axes[1].axvline(mean_article_length, alpha=0.5, linestyle = '--', c='black', label='Mean of main text
axes[1].legend()

# Global setup
plt.tight_layout()
plt.show()
```

```
plt.clf()
```

```
article_length_summary = news_df[['Title_length', 'Raw_article_length']].describe()
```

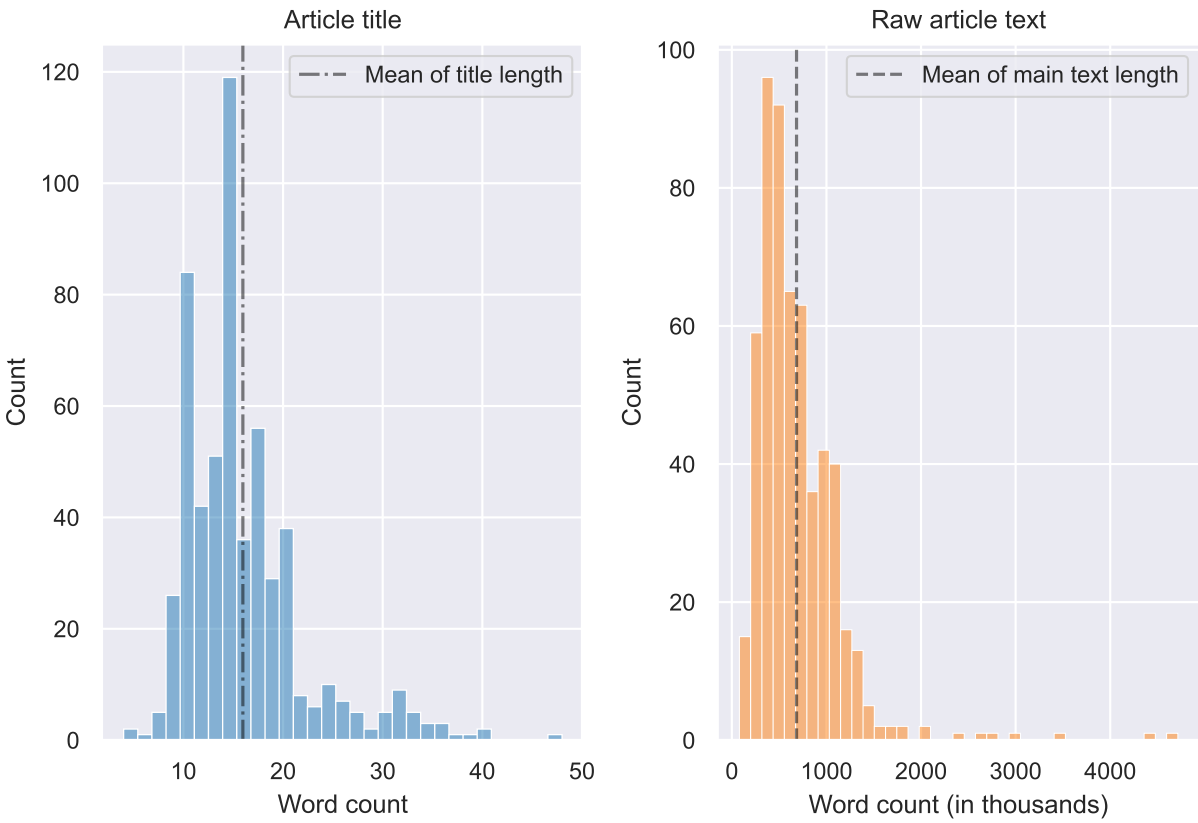


Figure 4.3: Distributions of the word counts of the articles' titles (left) and main texts (right)

```
knitr::kable(py$article_length_summary, col.names = c("Title", "Raw main text"), caption="Summary stat
```

Table 4.2: Summary statistics of the word counts of the news articles' titles and main texts

	Title	Raw main text
count	557.000000	557.0000
mean	15.965889	683.9264
std	5.993154	453.9607
min	4.000000	80.0000
25%	12.000000	404.0000
50%	15.000000	581.0000
75%	18.000000	893.0000
max	48.000000	4715.0000

4.2 Sentiment analysis

4.2.1 Preprocessing

After making sense of the dataset with EDA, it is time to build the sentiment analysis model to see whether the political affiliation of news media outlets is associated with the polarities of the news articles after controlling for other variables. But first there are some preprocessing steps to be done so that the data are transformed into suitable formats as inputs for machine learning models. For starters, columns of the metadata should be excluded for being the inputs of the models. Note that I have also removed the `Newspaper` column since H_1 is more interested in whether newspaper outlets of the pro-Beijing camp *as a whole* may hold more negative attitudes towards asylum seekers in Hong Kong vis-a-vis media outlets with other political stances. The removed metadata columns are: `Index`, `Date`, `Category`, `Page_number` and `Newspaper`.

```
metadata_columns = ["Index", "Date", "Category", "Page_number", "Newspaper"]
news_df.drop(columns=metadata_columns, inplace=True)
```

Furthermore, I have binned `Month` into four even split yearly quarters (`Quarter`) to reduce the dimensionality of the dataset. A further note on the categorical features is that they will need to be transformed via one-hot encoding, meaning that each of them will be transformed into `n` variables, with `n` being the number of the original distinct values. Meanwhile, it would also be better to standardise the numerical features (i.e. other than `Political_camp` and `Quarter`) by centering theirs means at 0 for better model convergence, but the standardiser should only be fitted on the training set after splitting the data into the training and validation sets in order to avoid data leakage (the same is also true for creating the TF-IDF matrix). In any case, sklearn's Pipeline class allows me to perform both standardisation and one-hot encoding conveniently later on.

```
# Making pro-Beijing become the reference category
news_df["Political_camp"] = pd.Categorical(news_df["Political_camp"], categories=['Pro-Beijing', 'Neut

# Binning the months into four quarters
def quarter(x):
  if x <= 3:
    return "Q1"
  elif x <= 6:
    return "Q2"
  elif x <= 9:
    return "Q3"
  else:
    return "Q4"
news_df["Quarter"] = pd.Categorical(news_df.Month.apply(quarter))
news_df.drop(columns="Month", inplace=True)

# One-hot encoding
news_df = pd.get_dummies(news_df, columns=["Political_camp", "Quarter"])
```

```
from sklearn.model_selection import train_test_split
X = news_df.drop(columns="Sentiment")
y = news_df.Sentiment
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=1)
```

The next step is to transform both the titles and main texts of the articles into a TF-IDF term-document matrix. Apart from joining the Title and Text columns together as the complete Article, I will also add additional words into the dictionary and remove stop words as well as punctuation for better tokenisation so that the `NMF` model can better discover the latent topics.

```
# Train set
X_train["Article"] = X_train.Title.str.cat(news_df.Text, sep=" ")
X_train.drop(columns=["Text", "Title"], inplace=True)

# Test set
X_test["Article"] = X_test.Title.str.cat(news_df.Text, sep=" ")
X_test.drop(columns=["Text", "Title"], inplace=True)
```

```
def read_text(path):
    with open(path, 'r', encoding='utf-8') as file:
        text = file.readlines()
        text = [word.replace('\n', '') for word in text]
        return text
```

```
stop_words_cantonese = read_text('Coding/text_cleaning/cantonese_stopwords.txt')
punctuations = [punc for punc in zhon.hanzi.punctuation]
stop_words_full = list(set(word for word in chain(stop_words_cantonese, punctuations)))
```

To avoid data leakage as mentioned before, I will only fit the `TfidfVectorizer` and the `NMF` models on the train set (i.e. `X_train`) and then use the fitted instances to transform both the train and test sets. Note that `NMF` is used here to reduce the dimensionality of the dataset to prevent overfitting because a typical TF-IDF matrix usually has thousands of columns (i.e. tokens) as features which will very likely cause the models to overfit on the current dataset with only around 550 observations in total if the term-document matrix is directly used as inputs. According to Stevens et al. (2012) (p.953), the matrix denoted as H which captures the weight of each topic (as columns) in each document (as rows) of the corpus can help summarise the information of the articles in terms of which topic(s) they primarily focus on.

I set the number of latent topics (`n_components`) as 10 for the `NMF` model, and this is decided based on figure 4.5 which plots the reconstruction error measuring the difference of the values between the original TF-IDF matrix and the reconstructed version after NMF. Although there are certainly other valid choices of the number of latent topics to be discovered by NMF, `10` appears to be a reasonable choice as a compromise between finding out a wide variety of topics in the corpus and not fitting too much into the noise of the data.

```
# tokenizer
def tokenize_zh(doc):
  return jieba.cut(doc)


# Preprocessor
def preprocessor_zh(doc):
  regex_punctuation = r"[\d+\s+\n\t]|[\s+\.\!\/_,$%^*(+\"\']+|[+~@#%&*]+|[]+|[]"
  return re.sub(regex_punctuation, "", doc)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF


# Creating the tfidf matrix from the training set
tfidf_vec = TfidfVectorizer(min_df = 0.02,  # Each token must appear in at least 2% of the documents
                            preprocessor=preprocessor_zh,
                            tokenizer=tokenize_zh,
                            stop_words=stop_words_full)
X_train_tfidf = tfidf_vec.fit_transform(X_train.Article)


# Plotting the reconstruction error according to the number of latent topics
```

```
## C:\Users\kenji\ANACON~1\envs\py_397\lib\site-packages\sklearn\feature_extraction\text.py:396: UserWa
##   warnings.warn(
```

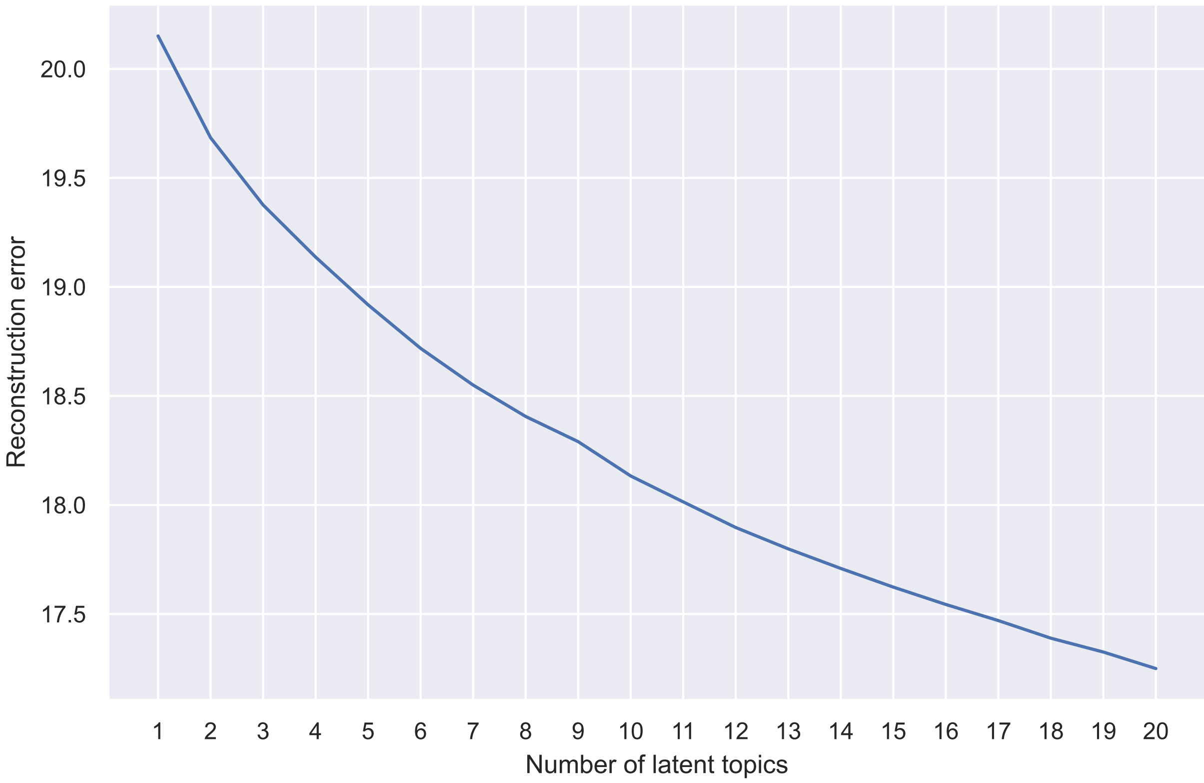


Figure 4.4: Elbow plot of the reconstruction error of NMF as a function of the number of pre-specified latent topics

```
reconstruct_error = []
for i in range(1, 21):
  nmf = NMF(n_components=i, max_iter=500, random_state=1)
  articles_nmf = nmf.fit(X_train_tfidf)
  reconstruct_error.append(nmf.reconstruction_err_)


ax = sns.lineplot(x=np.arange(1, 21), y=reconstruct_error)
ax.set(xlabel="Number of latent topics", ylabel="Reconstruction error", xticks=np.arange(1, 21))
plt.show()
```

```
plt.clf()


# Let's set n_components as 10 for nmf
```

```
nmf_10 = NMF(n_components=10, max_iter=500, random_state=1)
X_train_nmf = nmf_10.fit_transform(X_train_tfidf)
```

In order to make the latent topics generated by the NMF model be named more intuitive, I will inspect the 30 most prominent words of each latent topic and then summarise each topic in a few words[1]. Overall, the ten topics generated by NMF are more or less semantically coherent and can be summed up concisely. Finally, I will transform the validation set's articles with the fitted instance of the TF-IDF and NMF models with the training set.

```
# Naming the latent topics more precisely
topics_list = ["Crimes", "Non-refoulement policy", "Illegal labours", "Illegal gambling", "Drugs", "Il

# Concatenating the NMF DataFrame for the training set
X_train_nmf_df = pd.DataFrame(X_train_nmf, index=X_train.index, columns=topics_list)
X_train_final = pd.concat([X_train, X_train_nmf_df], axis=1)
X_train_final.drop(columns="Article", inplace=True)

# Concatenating the NMF DataFrame for the validation set
X_test_tfidf = tfidf_vec.transform(X_test.Article)
X_test_nmf = nmf_10.transform(X_test_tfidf)
X_test_nmf_df = pd.DataFrame(X_test_nmf, index=X_test.index, columns=topics_list)
X_test_final = pd.concat([X_test, X_test_nmf_df], axis=1)
X_test_final.drop(columns="Article", inplace=True)
```

4.2.2 Training the model

After the above preprocessing steps, it is time to train a model that adequately predicts the relations between the features and the sentiment of the articles before finding out the importance of the political camp as the main independent variable. To facilitate the decision of which model to use and model tuning, I will first run some baseline models with the default hyper-parameters, except that I have

[1]A list of the 30 most prominent words in each of the topic and the code to generate it are available in the appendix.

adjusted the weights of each class in the dependent variable due to class imbalance and also tweaked the `n_estimators` of the XGBoost model to 15 instead of the default value of 200 to prevent the model from being too big[2]. Moreover, tree-based models (i.e. random forest and xgboost) do not necessarily need to have the numerical features standardised, and thus only the categorical columns need to be one-hot encoded. The baseline models will be compared based on their performance on log loss in 5-fold cross validation and on the testing set.

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import log_loss

# Defining the kfold strategy
five_fold_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

# Utility function for evaluating the model's performance in cross validation and test set in terms of
def evaluate_model(model, model_name: str, cv=five_fold_cv, X_train=X_train_final, X_test=X_test_final
  cv_log_loss = np.mean(-cross_val_score(model, X_train, y_train, cv=cv, scoring="neg_log_loss"))
  y_pred_proba = model.predict_proba(X_test)
  test_log_loss = log_loss(y_test, y_pred_proba)
  return {"5-fold cv log loss": cv_log_loss, "Test set log loss": test_log_loss}
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Separating the columns for respective preprocessing steps
numeric_columns = [col for col in X_train_final.columns if X_train_final[col].dtype in ["int64", "floa

# Preprocessor for linear models
stand_preprocessor = ColumnTransformer([("standardiser", StandardScaler(), numeric_columns)],
```

[2] For the complete documentation of the default parameters of the models used in this thesis, refer to the websites of scikit-learn and XGBoost Documentation — xgboost 1.5.1 documentation.

```
                                          remainder='passthrough')
_ = stand_preprocessor.fit(X_train_final)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.utils.class_weight import compute_sample_weight
import xgboost as xgb

# Logistic regression pipeline
log_reg_baseline = Pipeline([("preprocess", stand_preprocessor),
                             ("log_reg", LogisticRegression(random_state=1,
                                                            class_weight="balanced"))])
_ = log_reg_baseline.fit(X_train_final, y_train)
log_reg_base_result = evaluate_model(log_reg_baseline, "baseline logistic regression")

# SVM pipeline
svm_baseline = Pipeline([("preprocess", stand_preprocessor),
                         ("svm", SVC(probability=True, class_weight="balanced", random_state=1))])
_ = svm_baseline.fit(X_train_final, y_train)
svm_base_result = evaluate_model(svm_baseline, "baseline support vector machine")

# Random forest pipeline
rf_baseline = RandomForestClassifier(class_weight="balanced", random_state=1, criterion="entropy")
_ = rf_baseline.fit(X_train_final, y_train)
rf_base_result = evaluate_model(rf_baseline, "baseline random forest")


# xgboost pipeline
```

```
xgb_sample_weight = compute_sample_weight(class_weight="balanced", y=y_train)
xgboost_baseline = xgb.XGBClassifier(n_estimators=15,
                                     objective="multi:softmax",
                                     eval_metric="mlogloss",
                                     random_state=1,
                                     use_label_encoder=False)
_ = xgboost_baseline.fit(X_train_final,
                         y_train,
                         sample_weight=xgb_sample_weight)
xgb_base_result = evaluate_model(xgboost_baseline, "baseline XGBoost")

# Creating the DataFrame of the baseline results
baseline_log_loss_df = pd.DataFrame([log_reg_base_result, svm_base_result, rf_base_result, xgb_base_re
```

```
knitr::kable(py$baseline_log_loss_df, caption = "Log loss on 5-fold cv and test set for the 4 baseline
```

Table 4.3: Log loss on 5-fold cv and test set for the 4 baseline models

	5-fold cv log loss	Test set log loss
Logistic regression	0.4648889	0.3495389
SVM	0.4416793	0.3371468
Random forest	0.3805292	0.3466180
XGBoost classifier	0.3863108	0.2859553

Table 4.3 contains the performance of the log loss on both the 5-fold cross validation and test set log loss scores for the four baseline models. Judging by the performance of log loss on the test set, it seems that the XGBoost model performs the best out of all candidates. I will then perform some hyperparameter tuning of the XGBoost model to see if there could be any improvements of its performance[3]. Surprisingly, the tuned XGBoost model has a higher log loss on the test data than the pre-tuned one. If we look at the per-class f1 score in tables 4.4 and 4.5, however, we can see that the tuned XGBoost model performs considerably better in the f1 score for predicting whether a news article's polarity is positive or not (i.e., class 2). Therefore, I will use the tuned model as the basis of

[3]The code for tuning the model can be found in the appendix.

interpreting the impact of the features on predicting the sentiments of the news articles about asylum seekers in Hong Kong in 2019.

```
# Loading the tuned model
import pickle
xgboost_tuned = pickle.load(open("xgb_clf.pkl", "rb"))


# Creating the DataFrames for f1 score of both baseline and tuned models
from sklearn.metrics import classification_report
xgb_base_class_report = pd.DataFrame(classification_report(y_test, xgboost_baseline.predict(X_test_fin
xgb_tuned_class_report = pd.DataFrame(classification_report(y_test, xgboost_tuned.predict(X_test_final
```

```
knitr::kable(py$xgb_base_class_report, digits=4, caption="Classification report of the baseline XGBoos
```

Table 4.4: Classification report of the baseline XGBoost model

	precision	recall	f1-score	support
0	0.8953	0.9506	0.9222	81.000
1	0.7826	0.6923	0.7347	26.000
2	1.0000	0.6000	0.7500	5.000
accuracy	0.8750	0.8750	0.8750	0.875
macro avg	0.8927	0.7476	0.8023	112.000
weighted avg	0.8738	0.8750	0.8710	112.000

```
knitr::kable(py$xgb_tuned_class_report, digits=4, caption="Classification report of the tuned XGBoost
```

Table 4.5: Classification report of the tuned XGBoost model

	precision	recall	f1-score	support
0	0.9259	0.9259	0.9259	81.0000
1	0.7407	0.7692	0.7547	26.0000
2	1.0000	0.8000	0.8889	5.0000
accuracy	0.8839	0.8839	0.8839	0.8839
macro avg	0.8889	0.8317	0.8565	112.0000
weighted avg	0.8862	0.8839	0.8845	112.0000

4.2.3 Is the pro-Beijing camp more likely to portray asylum seekers in 2019 more negatively than other outlets?

We can use SHAP values (Lundberg 2022) which evaluates how much impact each feature has on the model prediction between taking certain values and its baseline value. The higher the SHAP values of a feature, the higher its impact of the model's prediction. According to figure 4.6, we can see that on the level of the whole XGBoost model, whether a media belongs to the pro-Beijing camp or not (`Political_camp_Pro-Beijing`) is the fourth most important features in predicting the sentiment of a news article, only being lower than the magnitudes of three themes of the news articles about asylum seekers (i.e., public security, murder and crimes). Specifically, pro-Beijing affiliation has the second largest magnitude in affecting the prediction of whether a news article has a negative narrative against non-refoulement claimants, as one can see the pink bar representing the impact of a feature on predicting class `0` in the model is the second longest for the `Political_camp_Pro-Beijing` feature (only behind that of the theme `Crimes` in the articles). Let's zoom into the SHAP values plot for predicting class `0`.

```
# Setting up the shap values
import shap
xgb_explainer = shap.TreeExplainer(xgboost_tuned)
xgb_shap_values = xgb_explainer.shap_values(X_test_final)


# Defining the plotting function
```

```
## ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

```
def plot_shap_values(class_label=None, **kwargs):
  if class_label == None:
    shap.summary_plot(xgb_shap_values, X_test_final, **kwargs)
    plt.clf()
  else:
    shap.summary_plot(xgb_shap_values[class_label], X_test_final, **kwargs)
```

```
    plt.clf()
```

```
plot_shap_values()
```

The beeswarm plot in figure 4.7 zooms into the importance of each feature in predicting whether a news article on asylum seekers has a negative (coded as `0`) polarity, and features appearing at the top of the y-axis are deemed more important than those at the lower end of the y-axis. In a SHAP value beeswarm plot, dots in red mean the value of a feature is high (or present in case of a binary feature, e.g. one-hot-encoded columns), whereas those in blue mean the value of a feature is low (or absent in the case of a binary feature). As one can see, not only is the affiliation to pro-Beijing camp the second most determinant feature in predicting whether a news article has negative polarity, but such affiliation will also increase the probability of a news articles portraying asylum seekers in negative lights.

```
plot_shap_values(0)
```

If we inquire further in figures 4.8 and 4.9 about the impact of the features on the predictions of neutral and positive articles respectively, then some interesting observations arise. Firstly, pro-Beijing affiliation is not a very prominent feature in affecting the prediction of whether a news article is simply reporting on news related to asylum seekers objectively without much added sentiment and interpretation by the journalists, as figure 4.8 shows that the `Political_camp_Pro-Beijing` only occupies the middle layer of the y-axis and does not have a significant magnitude in affecting the prediction. By contrast, pro-Beijing affiliation of a media outlet once again is the second most important feature for the prediction of whether an article depicts asylum seekers in Hong Kong favourably. In particular, pro-Beijing media outlets are less likely to have positive reportage on non-refoulement claimants vis-a-vis media outlets from other camps.

```
plot_shap_values(1)
```

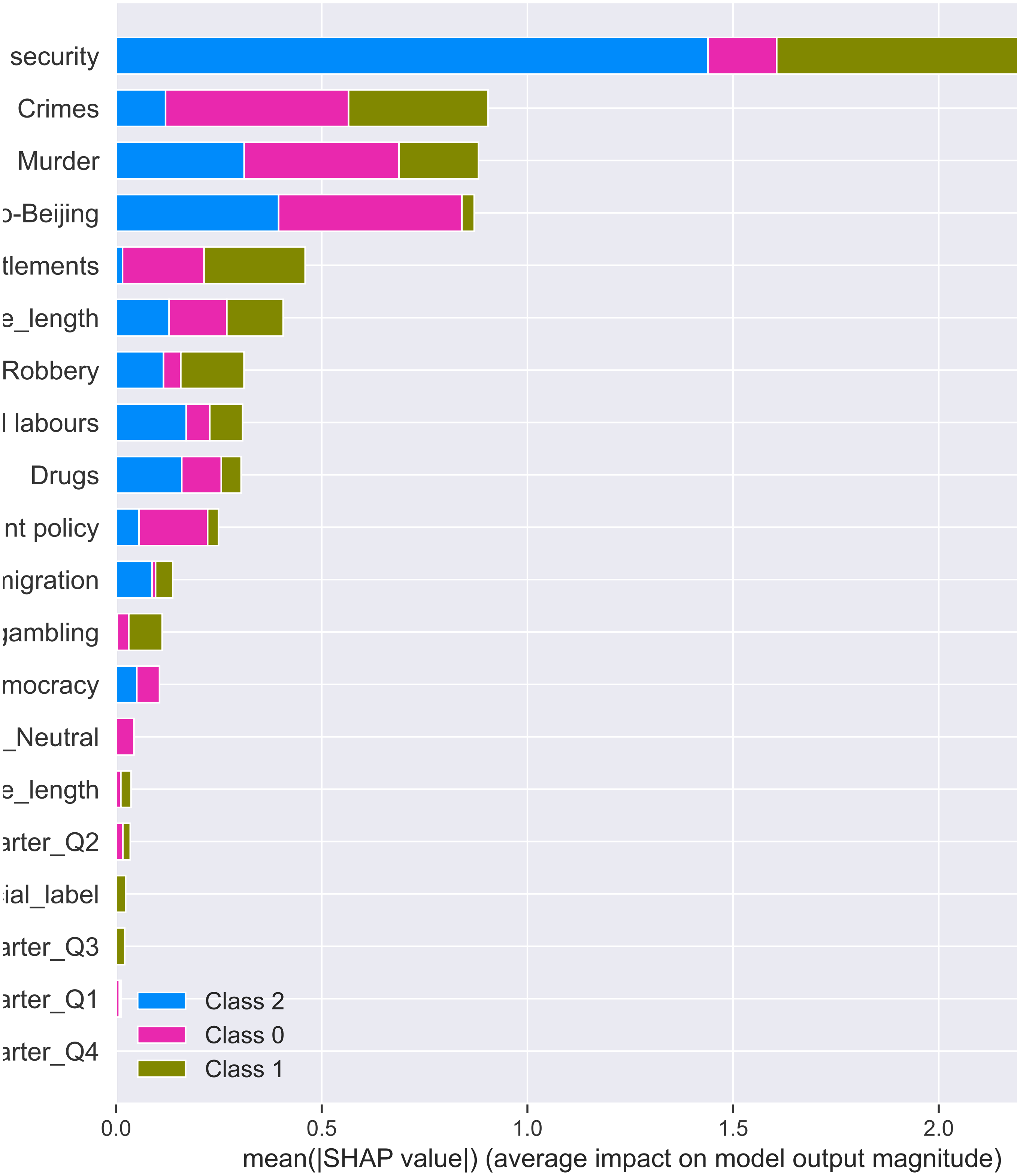
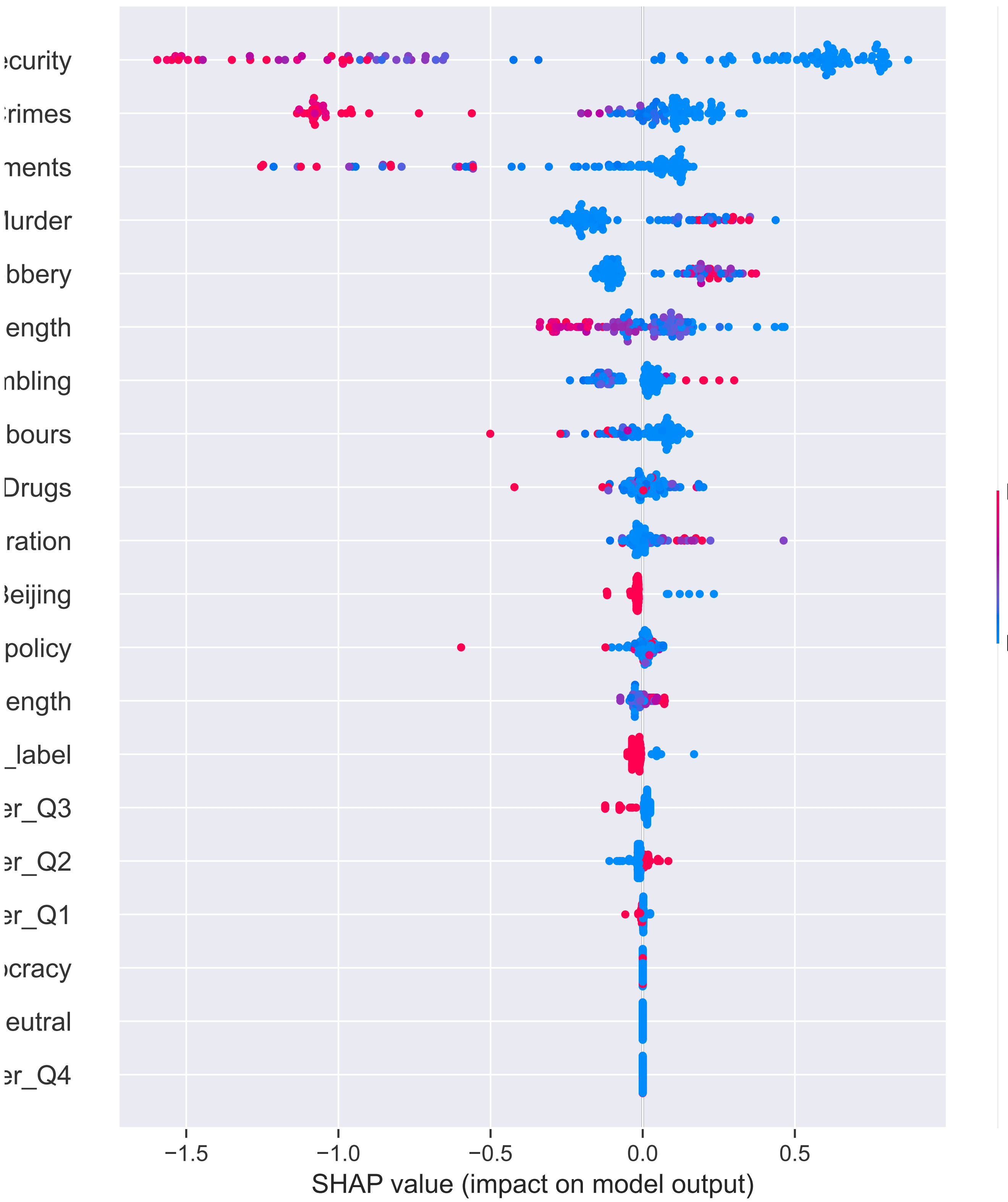
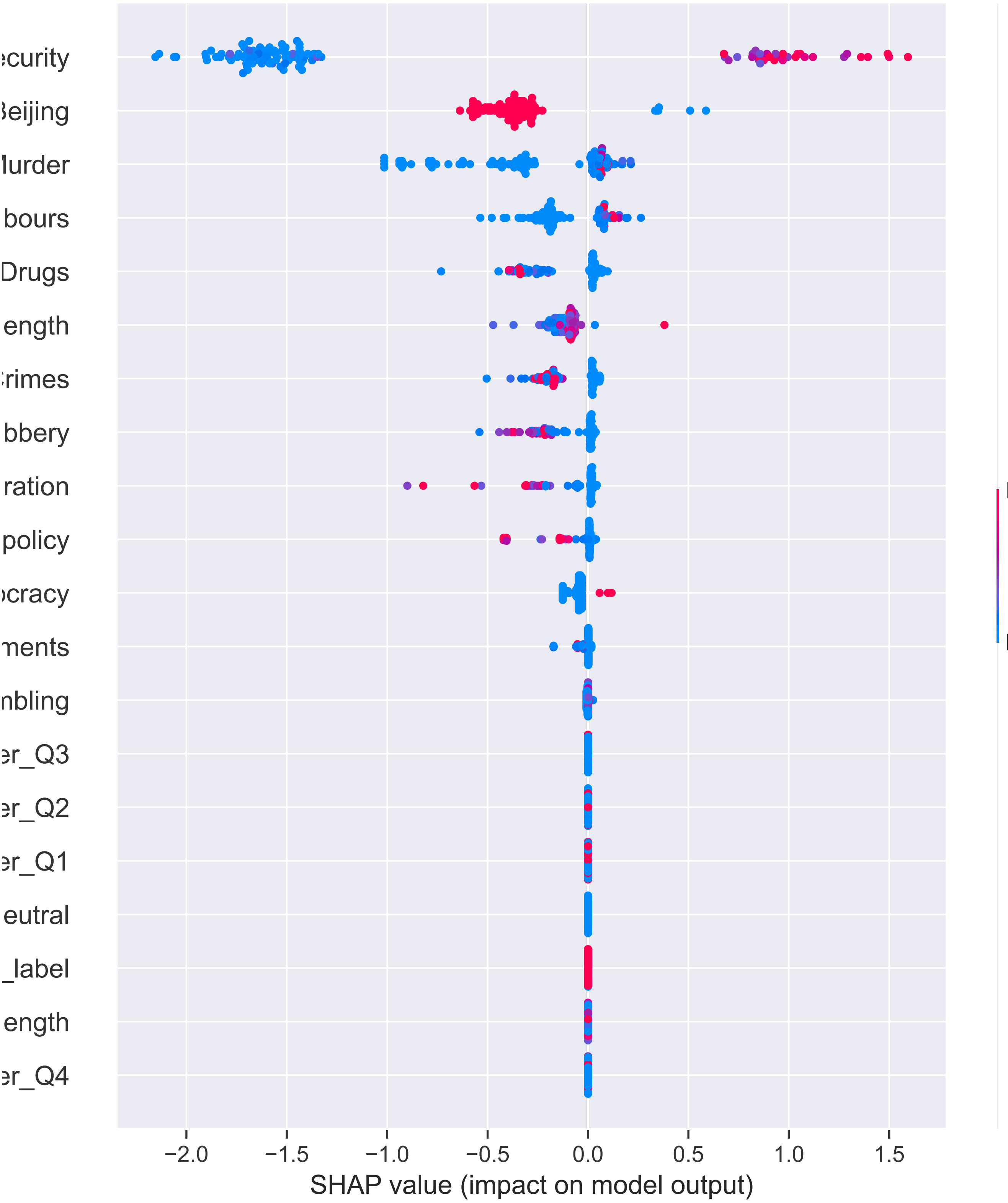


Figure 4.5: The overall SHAP values of the features in the model's predictions and their impact by class

Figure 4.6: The SHAP values of the features in the prediction of whether an article has a negative polarity


ecurity
rimes
ments
lurder
bbery
ength
nbling
bours
Drugs
ration
Beijing
policy
ength
_label
er_Q3
er_Q2
er_Q1
ocracy
eutral
er_Q4
−1.5
−1.0
−0.5
0.0
0.5
SHAP value (impact on model output)

```
plot_shap_values(2)
```


ecurity
Beijing
Murder
bours
Drugs
ength
rimes
bbery
ration
policy
ocracy
ments
nbling
er_Q3
er_Q2
er_Q1
eutral
_label
ength
er_Q4
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
SHAP value (impact on model output)

Based on the SHAP values of the features as shown in figures 4.6-4.9, we can therefore conclude that H_1 is supported by the evidence produced from the above sentiment analysis on the articles about asylum seekers in Hong Kong in 2019. Specifically, pro-Beijing Media outlets are more likely to frame asylum seekers negatively by using incorrect terms such as “fake refuges” to refer to this group of population or criticising them as social ills, and these media outlets are quite unlikely to have favourable reportage on non-refoulement claimants.

Here are also some notable observations about other features in the dataset. For starters,

Chapter 5

Conclusion

In conclusion, it is found that at least in 2019, the political camp of media outlets was associated with their attitudes towards asylum seekers in Hong Kong. Specifically,

5.1 How might the instigation of the National Security Law affect the public discourse on asylum seekers in Hong Kong?

Just a year after the anti-extradition law protest had started and once again mobilised a huge section of Hong Kong's society to oppose , the HKSAR Government promulgated the National Security Law in July 2020 which

As mentioned before, the opposition's political influence has been greatly crippled since the promulgation of the National Security Law in July 2020. With the conclusion of the recent 2021 Legislative Council election after an overhaul of the electoral system which essentially gatekeeps candidacy only to the "patriots" (Lau and Yam 2021), Hong Kong's legislature, once a political venue where opposition parties could access political resources and advocate alternative policy discourses,

Even in the media

Alternative media such as Apple Daily and Stand News were also forced to shut down in 2021 due to the