Week 5 Tutorial: Supervised Modelling

POP77032 Quantitative Text Analysis for Social Scientists

Exercise 1: Annotation

- In this exercise, we will re-visit human annotation and the calculation of inter-coder agreement.
- We will use MIND: Microsoft News Dataset for this exercise.
- You can either download the dataset from the link on the website or from Blackboard.
- Here we will focus on the `news.tsv` file with blobs of news articles, but do check the `behaviors.tsv` file as well as it contains the actual news engagement metrics.
- In groups of 2-3 decide on a common seed for random sampling and draw a subset of 20-50 news stories.
- Individually label the selected news stories on a scale of -1, 0, 1 with -1 being negative, 0 neutral and 1 positive.
- Share your labels with others in the groups and calculate inter-coder agreement using Krippendorff's α or Cohen's κ.

```r
url <- "https://huggingface.co/datasets/yjw1029/MIND/resolve/main/MINDlarge_train.zip"
```

```r
download.file(url, destfile = "../temp/MINDlarge_train.zip")
```

```r
unzip("./temp/MINDlarge_train.zip", exdir = "../temp/MIND")
```

```r
news <- readr::read_tsv(
  "../temp/MIND/news.tsv",
  col_names = c(
    "news_id", "category", "subcategory", "title",
    "abstract", "url", "title_entities", "abstract_entities"
  )
)
```

Naive Bayes

Naive Bayes in R

- In R, we can use the `quanteda.textmodels` package to fit a Naive Bayes classifier.

```r
library("quanteda")
library("quanteda.textmodels")
```

```r
# Simulate some textual data
set.seed(123)
n_docs <- 100; n_features <- 50; n_classes <- 2
vocab <- sample(quanteda::stopwords("en"), n_features)
classes <- sample(1:n_classes, n_docs, replace = TRUE)

# Simulate texts as random draws from the vocab with
# different word frequencies for each class
txts <- NA * length(n_docs)
txts[classes == 1] <- vapply(
  1:sum(classes == 1),
  function(x) paste(
    sample(vocab, sample(10:20, 1), replace = TRUE, prob = c(rep(0.01, 25), rep(0.03, 25))),
    collapse = " "
  ), character(1)
)
txts[classes == 2] <- vapply(
  1:sum(classes == 2),
  function(x) paste(
    sample(vocab, sample(10:20, 1), replace = TRUE, prob = c(rep(0.03, 25), rep(0.01, 25))),
    collapse = " "), character(1)
)

head(txts)
```

```
[1] "other didn't won't until both be until because both ours let's them they'd more until was isn't be"
[2] "each when his too ours them herself isn't he he's isn't here's let's he over"
[3] "couldn't such he's couldn't am other until more doing he we're why"
[4] "same be so until she'd who couldn't you'll a isn't he's up here's such herself ours to no let's"
[5] "why up very so very he until up he's them let's there as too"
[6] "do no do she'll she'd when until until why each few who"
```

```r
train <- sample(1:n_docs, floor(n_docs * 0.8), replace = FALSE)
test <- setdiff(1:n_docs, train)
```

```r
# Create a document-feature matrix from the simulated texts
txts_dfm <- quanteda::dfm(quanteda::tokens(txts))
quanteda::docvars(txts_dfm, "id") <- 1:n_docs
quanteda::docvars(txts_dfm, "class") <- classes
txts_dfm_train <- quanteda::dfm_subset(txts_dfm, subset = 1:n_docs %in% train)
txts_dfm_test <- quanteda::dfm_subset(txts_dfm, subset = 1:n_docs %in% test)
```

```r
txts_nb <- quanteda.textmodels::textmodel_nb(
  txts_dfm_train,
  quanteda::docvars(txts_dfm_train, "class")
)
summary(txts_nb)
```

```
Call:
textmodel_nb.dfm(x = txts_dfm_train, y = quanteda::docvars(txts_dfm_train,
    "class"))

Class Priors:
(showing first 2 elements)
  1   2
0.5 0.5

Estimated Feature Scores:
    other  didn't   won't   until    both       be because     ours   let's
1 0.01207 0.01034 0.01207 0.01034 0.03103 0.006897 0.02586 0.008621 0.01552
2 0.03367 0.02196 0.02489 0.05124 0.01025 0.014641 0.01611 0.039531 0.02635
      them  they'd     more     was   isn't     each     when      his     too
1 0.006897 0.03448 0.036207 0.03276 0.01034 0.027586 0.029310 0.032759 0.01034
2 0.027818 0.01025 0.005857 0.01171 0.03221 0.008785 0.005857 0.007321 0.03075
  herself      he    he's   here's    over couldn't    such       am   doing
1 0.02931 0.01379 0.02241 0.006897 0.02759 0.022414 0.02586 0.022414 0.03103
2 0.01171 0.02782 0.01611 0.024890 0.01025 0.008785 0.01025 0.008785 0.01025
```

Naive Bayes can only take features into consideration that occur both in the training and test set, but we can make the features identical using `dfm_match()`

```
txts_dfm_matched <- quanteda::dfm_match(txts_dfm_test, features = quanteda::featnames(txts_dfm_train))
```

```
txts_pred_prob <- predict(txts_nb, newdata = txts_dfm_matched, type = "prob")
txts_pred_class <- predict(txts_nb, newdata = txts_dfm_matched, type = "class")
```

```
# Posterior probabilities for each class
txts_pred_prob
```

```
docs                1            2
  text2  4.725997e-02 9.527400e-01
  text7  9.999994e-01 6.077527e-07
  text8  9.999648e-01 3.517497e-05
  text13 9.997856e-01 2.144390e-04
  text15 1.492684e-01 8.507316e-01
  text18 1.330139e-06 9.999987e-01
  text22 1.717026e-03 9.982830e-01
  text23 2.032627e-03 9.979674e-01
  text25 3.922640e-04 9.996077e-01
  text31 2.435133e-06 9.999976e-01
  text36 2.812432e-01 7.187568e-01
  text39 9.999993e-01 6.911338e-07
  text57 1.814028e-03 9.981860e-01
  text67 6.649492e-05 9.999335e-01
  text70 4.873724e-02 9.512628e-01
  text73 9.984123e-01 1.587729e-03
  text78 9.998881e-01 1.118688e-04
  text80 9.999931e-01 6.888002e-06
```

```r
# Confusion matrix
confmat <- table(predicted = txts_pred_class, actual = quanteda::docvars(txts_dfm_matched, "class"))
confmat
```

```
         actual
predicted  1  2
        1  7  2
        2  0 11
```

Naive Bayes in Python

```python
import numpy as np
from nltk.corpus import stopwords

rng = np.random.default_rng(123)

n_docs = 100
n_features = 50
n_classes = 2
vocab = rng.choice(stopwords.words("english"), size = n_features, replace = False).tolist()
classes = rng.integers(1, n_classes + 1, size = n_docs)

txts = [
    " ".join(rng.choice(
      vocab, size = rng.integers(10, 21), replace = True,
      p = [0.01] * 25 + [0.03] * 25 if c == 1 else [0.03] * 25 + [0.01] * 25
      ))
    for c in classes
]

txts[:6]
```

```
["shouldn after am can hasn with don doesn't again mustn", "t m shouldn't they'd doesn down no that t were no
those those that'll if i'll they t", "no our with those they'll above we've we'd them didn't doesn't don m am
mustn they mustn't were", "we'd did about hasn with doesn't with don where down don above wouldn were just
they'd", "from couldn m mustn't by after mustn with we'll for should we'll again mustn't during", "shouldn
they'll each that m couldn doesn't did them after we'll don should am where they'll it'd"]
```

In Python we can rely on the `sklearn` library to fit a Naive Bayes classifier.

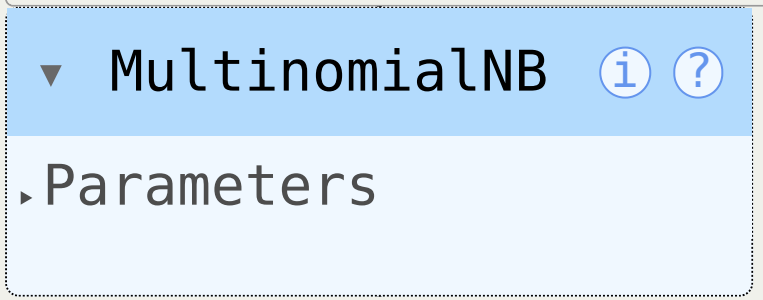
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report
```

```
# train = rng.choice(n_docs, size = int(n_docs * 0.8), replace = False)
# test = np.setdiff1d(np.arange(n_docs), train)
```

```
txts_train, txts_test, classes_train, classes_test = train_test_split(
  txts, classes, test_size = 0.2, random_state = 123
)

# Create a document-term matrix
vectorizer = CountVectorizer()
txts_dtm_train = vectorizer.fit_transform(txts_train)
txts_dtm_test = vectorizer.transform(txts_test)
```

```
# Fit a Naive Bayes classifier
nb = MultinomialNB()
nb.fit(txts_dtm_train, classes_train)
```



```
# Predict on the test set
classes_pred = nb.predict(txts_dtm_test)
```

```
# Evaluate the model
conf_mat = confusion_matrix(classes_pred, classes_test)
conf_mat
```

```
array([[13,  1],
       [ 0,  6]])
```

Exercise 2: Supervised Learning

- Let's now try fitting some supervised learning models to this dataset.
- We will try to predict the category of the news story using the title and abstract.
- We will use a simple Naive Bayes classifier for this task, but feel free to experiment with other models as well.
- First, we need to prepare the data for modelling. We will create a document-feature matrix (DFM) from the title and abstract of the news stories.
- We will also need to split the data into training and testing sets.
- Finally, we will fit the model and evaluate its performance on the test set.