# Introduction

In this assignment, you are going to build a classifier for named entities from the Groningen Meaning Bank corpus. Named entity recognition (NER) takes noun phrases from a text and identifies whether they are persons, organizations, and so on. You will be using the Groningen Meaning Bank named entity corpus available on mltgpu at [/scratch/lt2222-v21-resources/GMB_dataset.txt](/scratch/lt2222-v21-resources/GMB_dataset.txt). In this version of the task, you will assume we know *that* something is a named entity, and instead use multi-class classification to identify its type. So you will be doing named entity classification but *not* recognition.

The data looks like this:

```
3996	182.0	Nicole	NNP	B-per
3997	182.0	Ritchie	NNP	I-per
3998	182.0	is	VBZ	O
3999	182.0	pregnant	JJ	O
4000	182.0	.	.	O
4001	183.0	Speaking	VBG	O
4002	183.0	to	TO	O
4003	183.0	ABC	NNP	B-org
4004	183.0	News	NNP	I-org
4005	183.0	interviewer	NN	O
4006	183.0	Dianne	NNP	B-per
4007	183.0	Sawyer	NNP	I-per
4008	183.0	,	,	O
4009	183.0	the	DT	O
4010	183.0	25-year-old	JJ	O
4011	183.0	co-star	NN	O
4012	183.0	of	IN	O
4013	183.0	TV	NN	O
4014	183.0	's	POS	O
4015	183.0	The	DT	B-art
4016	183.0	Simple	NNP	I-art
4017	183.0	Life	NNP	I-art
4018	183.0	said	VBD	O
4019	183.0	she	PRP	O
4020	183.0	is	VBZ	O
4021	183.0	almost	RB	O
4022	183.0	four	CD	O
4023	183.0	months	NNS	O
4024	183.0	along	IN	O
4025	183.0	in	IN	O
```

```
4026    183.0   her     PRP$    O
4027    183.0   pregnancy       NN      O
4028    183.0   .       .       O
```

The first column is the line number. The second column is a sentence number (for some reason given as a float; ignore it). The third column is the word. The fourth column is a part of speech (POS) tag in Penn Treebank format. The last column contains the named entity annotation.

The annotation works like this. Every `O` just means that the row does not represent a named entity. `B-xyx` means the first word in a named entity with type `xyx`. `I-xyz` means the second and later words of an `xyz` entity, if there are any. That means that every time there's a `B` or an `I`, there's a named entity.

The entity types in the corpus are `art`, `eve`, `geo`, `gpe`, `nat`, `org`, `per`, and `tim`

Your task is the following.

1. To preprocess the text (lowercase and lemmatize; punctuation can be preserved as it gets its own rows).
2. To create instances from every from every identified named entity in the text with the type of the NE as the class, and a surrounding context of five words on either side as the features.
3. To generate vectors and split the instances into training and testing datasets at random.
4. To train a support vector machine (via `sklearn.svm.LinearSVC`) for classifying the NERs.
5. To evaluate the performance of the classifier.

You will do this by modifying a separate file containing functions that will be called from this notebook as a module. You can modify this notebook for testing purposes but please only submit the original. You will document everything in Markdown in README.md and submit a GitHub repository URL.

This assignment is due on **Tuesday, 2021 March 9 at 23:59**. It has **25 points** and **7 bonus points**.

```
!pip install lazypredict #try to use lazypredict classifier for the first time (optio
!pip install scikit-plot

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
True
```

```
import importlib
import a2
```

```
import a2
import pandas as pd

import numpy as np
import scikitplot as skplt

from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import plot_confusion_matrix,confusion_matrix

import nltk

import matplotlib.pyplot as plt

import lazypredict
from lazypredict.Supervised import LazyClassifier
```

## Part 1 - preprocessing (3 points)

See step 1 above. The data is coming to you as an unused file handle object. You can return the data in any indexable form you like. You can also choose to remove infrequent or uninformative words to reduce the size of the feature space. (Document this in README.md.)

```
gmbfile = open('GMB_dataset.txt', "r")
inputdata = a2.preprocess(gmbfile)
inputdata[20:40]
```

| | Sentence # | Word | POS | Tag | Tag_prefix | Tag_entity | Word_seq |
|---|---|---|---|---|---|---|---|
| **36** | 2.0 | banner | NNS | O | O | None | 8 |
| **39** | 2.0 | slogan | NNS | O | O | None | 9 |
| **41** | 2.0 | """" | `` | O | O | None | 10 |
| **42** | 2.0 | bush | NNP | B-per | B | per | 11 |
| **43** | 2.0 | number | NN | O | O | None | 12 |
| **44** | 2.0 | one | CD | O | O | None | 13 |
| **45** | 2.0 | terrorist | NN | O | O | None | 14 |
| **46** | 2.0 | """" | `` | O | O | None | 15 |

```
inputdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43977 entries, 0 to 66160
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Sentence #  43977 non-null  object
 1   Word        43977 non-null  object
 2   POS         43977 non-null  object
 3   Tag         43977 non-null  object
 4   Tag_prefix  43977 non-null  object
 5   Tag_entity  9736 non-null   object
 6   Word_seq    43977 non-null  int64
dtypes: int64(1), object(6)
memory usage: 2.7+ MB
```

## Part 2 - Creating instances (7 points)

Do step 2 above. You will create a collection of Instance objects. Remember to consider the case where the NE is at the beginning of a sentence or at the end, or close to either (you can create a special start token for that). You can also start counting from before the `B` end of the NE mention and after the last `I` of the NE mention. That means that the instances should include things before and after the named entity mention, but not the named entity text itself.

```
n=5

instances = a2.create_instances(inputdata,n=n,skip_ne=False)
instances[20:40]
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **geo** | percent | world | smoker | thought | live | india | alone | S |
| **geo** | world | smoker | thought | live | china | alone | S5 | S |
| **gpe** | S1 | S2 | S3 | S4 | S5 | military | launched | offensiv |
| **geo** | S5 | pakistani | military | launched | offensive | hunt | taliban | insurge |
| **org** | military | launched | offensive | orakzai | hunt | insurgent | S5 | S |
| **nat** | die | cancer | soon | greater | death | tuberculosis | malaria | combine |
| **tim** | may | rise | 27 | million | year | 17 | million | peop |
| **org** | force | tried | suppress | report | abuse | prisoner | S5 | S |
| **org** | S3 | S4 | S5 | document | released | civil | liberty | unic |
| **org** | S4 | S5 | document | released | american | liberty | union | tuesda |
| **org** | S5 | document | released | american | civil | union | tuesday | sa |
| **org** | document | released | american | civil | liberty | tuesday | say | sta |
| **tim** | released | american | civil | liberty | union | say | staff | memb |
| **org** | union | tuesday | say | staff | member | defense | intelligence | agenc |
| **org** | tuesday | say | staff | member | pentagon | intelligence | agency | d |
| **org** | say | staff | member | pentagon | defense | agency | dia | witnesse |
| **org** | staff | member | pentagon | defense | intelligence | dia | witnessed | sever |
| **org** | member | pentagon | defense | intelligence | agency | witnessed | several | incide |
| **org** | S5 | document | also | included | complaint | personnel | monitored | speci |

```
encode_strings = ["S{}".format(i) for i in range(1, n + 1)]
instances = instances.replace(to_replace=encode_strings,value="NA")
instances[20:40]
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **geo** | percent | world | smoker | thought | live | india | alone | N |
| **geo** | world | smoker | thought | live | china | alone | NA | N |
| **gpe** | NA | NA | NA | NA | NA | military | launched | offensiv |
| **geo** | NA | pakistani | military | launched | offensive | hunt | taliban | insurge |
| **org** | military | launched | offensive | orakzai | hunt | insurgent | NA | N |
| **nat** | die | cancer | soon | greater | death | tuberculosis | malaria | combine |
| **tim** | may | rise | 27 | million | year | 17 | million | peop |
| **org** | force | tried | suppress | report | abuse | prisoner | NA | N |
| **org** | NA | NA | NA | document | released | civil | liberty | unic |
| **org** | NA | NA | document | released | american | liberty | union | tuesda |
| **org** | NA | document | released | american | civil | union | tuesday | sa |
| **org** | document | released | american | civil | liberty | tuesday | say | sta |
| **org** | staff | member | pentagon | defense | intelligence | dia | witnessed | sever |

## Part 3 - Creating the table and splitting (10 points)

Here you're going to write the functions that create a data table with "document" vectors representing each instance and split the table into training and testing sets and random with an 80%/20% train/test split.

```
importlib.reload(a2)

bigdf = a2.create_table(instances,method='tfidf')
bigdf[20:40]
```

| | _class_ | 10 | 100 | 103 | 10th | 11 | 110 | 119 | 11th | 12 | 120 | 123 | 12th | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | geo | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 21 | geo | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 22 | gpe | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 23 | geo | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 24 | org | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 25 | nat | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 26 | tim | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 27 | org | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 28 | org | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 29 | org | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 30 | org | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 31 | org | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

```
train_X, train_y, test_X, test_y = a2.ttsplit(bigdf)
```

```
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(7184, 4085) (7184,) (1796, 4085) (1796,)
```

```
len(test_y) / (len(test_y) + len(train_y))
```

```
0.2
```

```
len(test_X) / (len(test_X) + len(train_X))
```

```
0.2
```

```
test_y[0]
```

```
'per'
```

## Part 4 - Training the model (0 points)

This part you won't do yourself.

```
model = LinearSVC()
model.fit(train_X, train_y)
```

```
train_predictions = model.predict(train_X)
test_predictions = model.predict(test_X)
```

```
train_predictions
```

```
array(['tim', 'geo', 'gpe', ..., 'org', 'per', 'org'], dtype=object)
```

```
train_y
```

```
array(['gpe', 'geo', 'gpe', ..., 'org', 'per', 'org'], dtype=object)
```

```
test_predictions
```

```
array(['per', 'per', 'per', ..., 'org', 'geo', 'per'], dtype=object)
```

```
test_y
```

```
array(['per', 'org', 'gpe', ..., 'org', 'geo', 'per'], dtype=object)
```

```
model = LazyClassifier(verbose=1,ignore_warnings=True, custom_metric=None)
model.fit(train_X,test_X, train_y,test_y)
```

```
  3%|▎         | 1/29 [01:09<32:25, 69.49s/it]{'Model': 'AdaBoostClassifier', '.
  7%|▋         | 2/29 [03:01<42:31, 94.51s/it]{'Model': 'BaggingClassifier', 'A
 10%|█         | 3/29 [03:04<22:49, 52.66s/it]{'Model': 'BernoulliNB', 'Accuracy
 14%|█▍        | 4/29 [44:18<7:00:16, 1008.66s/it]{'Model': 'CalibratedClassifi
 21%|██        | 6/29 [45:23<2:51:32, 447.50s/it]{'Model': 'DecisionTreeClassifi
 24%|██▍       | 7/29 [45:25<1:50:43, 301.98s/it]{'Model': 'DummyClassifier', '.
 28%|██▊       | 8/29 [45:30<1:12:29, 207.14s/it]{'Model': 'ExtraTreeClassifier
 31%|███       | 9/29 [48:12<1:04:24, 193.20s/it]{'Model': 'ExtraTreesClassifier
 34%|███▍      | 10/29 [48:16<42:39, 134.70s/it] {'Model': 'GaussianNB', 'Accur
 38%|███▊      | 11/29 [51:03<43:25, 144.73s/it]{'Model': 'KNeighborsClassifier
 41%|████▏     | 12/29 [51:21<30:06, 106.24s/it]{'Model': 'LabelPropagation', '.
 45%|████▍     | 13/29 [51:40<21:16, 79.80s/it] {'Model': 'LabelSpreading', 'Ac
 48%|████▊     | 14/29 [54:56<28:42, 114.80s/it]{'Model': 'LinearDiscriminantAn
 52%|█████▏    | 15/29 [1:04:29<58:58, 252.78s/it]{'Model': 'LinearSVC', 'Accur
 55%|█████▌    | 16/29 [1:04:57<40:09, 185.36s/it]{'Model': 'LogisticRegression'
 59%|█████▊    | 17/29 [1:05:01<26:06, 130.54s/it]{'Model': 'NearestCentroid',
 66%|██████▌   | 19/29 [1:05:55<13:20, 80.07s/it]{'Model': 'PassiveAggressiveCla
 69%|██████▉   | 20/29 [1:06:15<09:17, 61.89s/it]{'Model': 'Perceptron', 'Accur
 72%|███████▏  | 21/29 [1:06:58<07:29, 56.21s/it]{'Model': 'QuadraticDiscrimina
 76%|███████▌  | 22/29 [1:07:53<06:31, 55.91s/it]{'Model': 'RandomForestClassifi
 79%|███████▉  | 23/29 [1:08:08<04:21, 43.61s/it]{'Model': 'RidgeClassifier', '.
 83%|████████▎ | 24/29 [1:11:39<07:49, 93.84s/it]{'Model': 'RidgeClassifierCV',
 86%|████████▌ | 25/29 [1:13:08<06:09, 92.44s/it]{'Model': 'SGDClassifier', 'Acc
 90%|████████▉ | 26/29 [1:31:52<20:05, 401.98s/it]{'Model': 'SVC', 'Accuracy':
 97%|█████████▋| 28/29 [1:49:25<07:39, 459.36s/it]{'Model': 'XGBClassifier', 'A
100%|██████████| 29/29 [1:49:52<00:00, 227.33s/it]{'Model': 'LGBMClassifier', 'A

(                              Accuracy  ...  Time Taken
 Model                                  ...
```

```
ExtraTreesClassifier                0.55  ...      162.54
RandomForestClassifier              0.52  ...       55.21
RidgeClassifierCV                   0.45  ...      211.01
NearestCentroid                     0.46  ...        3.07
LinearDiscriminantAnalysis          0.43  ...      195.70
LogisticRegression                  0.46  ...       28.77
RidgeClassifier                     0.44  ...       14.92
ExtraTreeClassifier                 0.42  ...        4.07
GaussianNB                          0.36  ...        3.71
Perceptron                          0.44  ...       19.54
BaggingClassifier                   0.46  ...      112.03
PassiveAggressiveClassifier         0.44  ...       52.21
XGBClassifier                       0.46  ...     1052.57
DecisionTreeClassifier              0.42  ...       62.49
LGBMClassifier                      0.46  ...       27.40
SGDClassifier                       0.48  ...       89.15
LinearSVC                           0.39  ...      572.54
KNeighborsClassifier                0.37  ...      167.48
SVC                                 0.48  ...     1124.14
BernoulliNB                         0.49  ...        2.86
CalibratedClassifierCV              0.44  ...     2474.19
QuadraticDiscriminantAnalysis       0.13  ...       42.95
LabelSpreading                      0.01  ...       18.95
LabelPropagation                    0.01  ...       18.19
AdaBoostClassifier                  0.28  ...       69.49
DummyClassifier                     0.20  ...        2.39

[26 rows x 5 columns],
                               Accuracy  ...  Time Taken
Model                                    ...
```

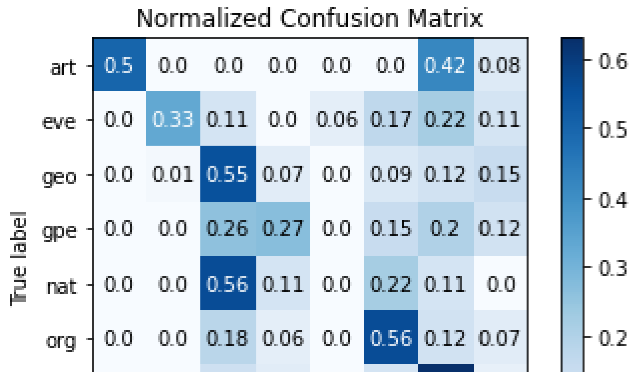
## Part 5 - Evaluation (5 points)

Investigate for yourself what a "confusion matrix". Then implement a function that takes the data and produces a confusion matrix in any readable form that allows us to compare the performance of the model by class.

```
model.score(train_X,train_y),model.score(test_X,test_y)
```

```
(0.9001948775055679, 0.5005567928730512)
```

```
#Method 1
importlib.reload(a2)
a2.confusion_matrix(test_y, test_predictions)
```

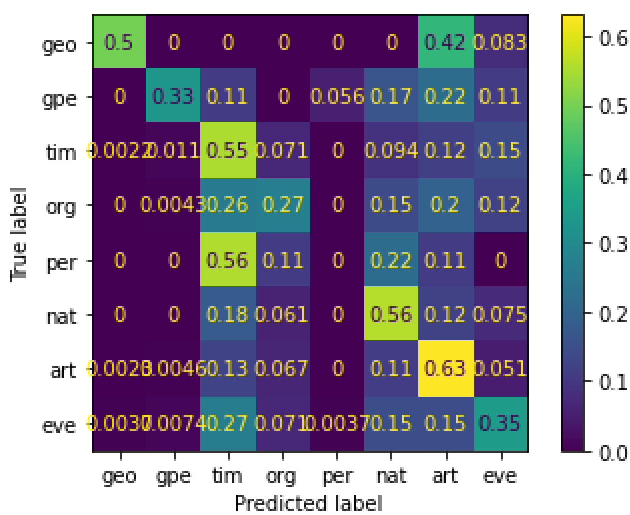


```
#Method 2
np.set_printoptions(precision=2)
fig = plt.figure(figsize=(8,8))
class_names = bigdf['_class_'].unique()

disp = plot_confusion_matrix(model,test_X, test_y,
                                 display_labels=class_names,
                                 normalize='true')

confusion_matrix(test_y,test_predictions)
```

```
array([[  6,   0,   0,   0,   0,   0,   5,   1],
       [  0,   6,   2,   0,   1,   3,   4,   2],
       [  1,   5, 249,  32,   0,  42,  54,  66],
       [  0,   1,  60,  63,   0,  34,  46,  29],
       [  0,   0,   5,   1,   0,   2,   1,   0],
       [  0,   0,  68,  23,   0, 210,  46,  28],
       [  1,   2,  56,  29,   0,  49, 272,  22],
       [  1,   2,  72,  19,   1,  41,  40,  93]])
<Figure size 576x576 with 0 Axes>
```



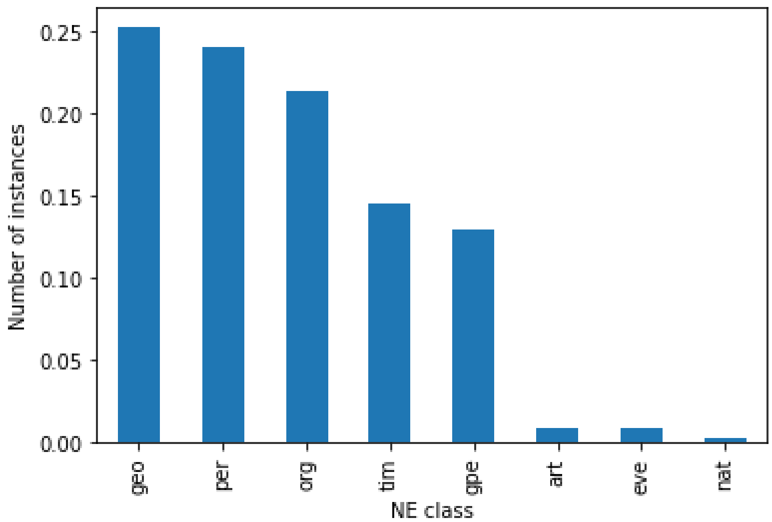
Examine the matrix and describe your observations in README.md. In particular, what do you notice about the predictions on the training data compared to those on the test data.

## Bonus Part A - Error analysis (2 points)

Look at the weakest-performing classes in the confusion matrix (or any, if they all perform poorly to the same extent). Find some examples in the test data on which the classifier classified incorrectly for those classes. What do you think is the reason why those are hard? Consider linguistic factors and statistical factors, if applicable. Write your answer in README.md.

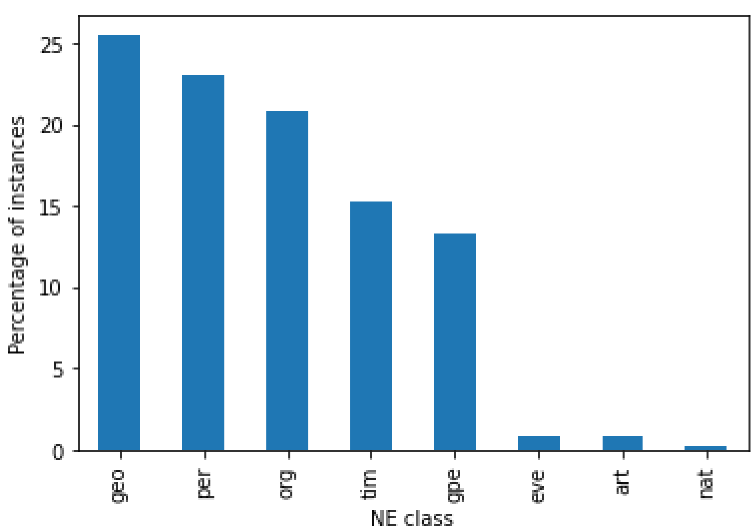
```
inputdata['Tag_entity'].value_counts(normalize=True).plot.bar()
plt.xlabel('NE class')
plt.ylabel('Number of instances')
```

```
Text(0, 0.5, 'Number of instances')
```




```
counts = bigdf['_class_'].value_counts(normalize=True)*100
counts.plot.bar()
plt.xlabel('NE class')
plt.ylabel('Percentage of instances')
```

```
Text(0, 0.5, 'Percentage of instances')
```



```
counts

    geo   25.51
    per   23.02
    org   20.87
    tim   15.24
    gpe   13.32
    eve    0.87
    art    0.86
    nat    0.31
    Name: _class_, dtype: float64
```

## Bonus Part B - Expanding the feature space (7 points)

Run the entire process above, but incorporate part-of-speech tag information into the feature vectors. It's your choice as to how to do this, but document it in README.md. Your new process should run from the single call below:

```
a2.bonusb('/scratch/lt2222-v21-resources/GMB_dataset.txt')
```