In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.corpus import wordnet
```

In [ ]:

```
nltk.download("wordnet")
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

Out[ ]:

```
True
```

In [ ]:

```
# https://www.kaggle.com/shivamkushwaha/bbc-full-text-document-classification
!wget -nc https://lazyprogrammer.me/course_files/nlp/bbc_text_cls.csv
```

```
--2021-11-24 05:54:58--  https://lazyprogrammer.me/course_files/nlp/bbc_text_cls.csv
Resolving lazyprogrammer.me (lazyprogrammer.me)... 104.21.23.210, 172.67.213.166, 260
6:4700:3030::ac43:d5a6, ...
Connecting to lazyprogrammer.me (lazyprogrammer.me)|104.21.23.210|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5085081 (4.8M) [text/csv]
Saving to: ‘bbc_text_cls.csv’

bbc_text_cls.csv    100%[===================>]   4.85M  19.9MB/s    in 0.2s

2021-11-24 05:54:59 (19.9 MB/s) - ‘bbc_text_cls.csv’ saved [5085081/5085081]
```

In [ ]:

```
df = pd.read_csv('bbc_text_cls.csv')
```

In [ ]:

```
df.head()
```

Out[ ]:

	text	labels
0	Ad sales boost Time Warner profit\n\nQuarterly...	business
1	Dollar gains on Greenspan speech\n\nThe dollar...	business
2	Yukos unit buyer faces loan claim\n\nThe owner...	business
3	High fuel prices hit BA's profits\n\nBritish A...	business

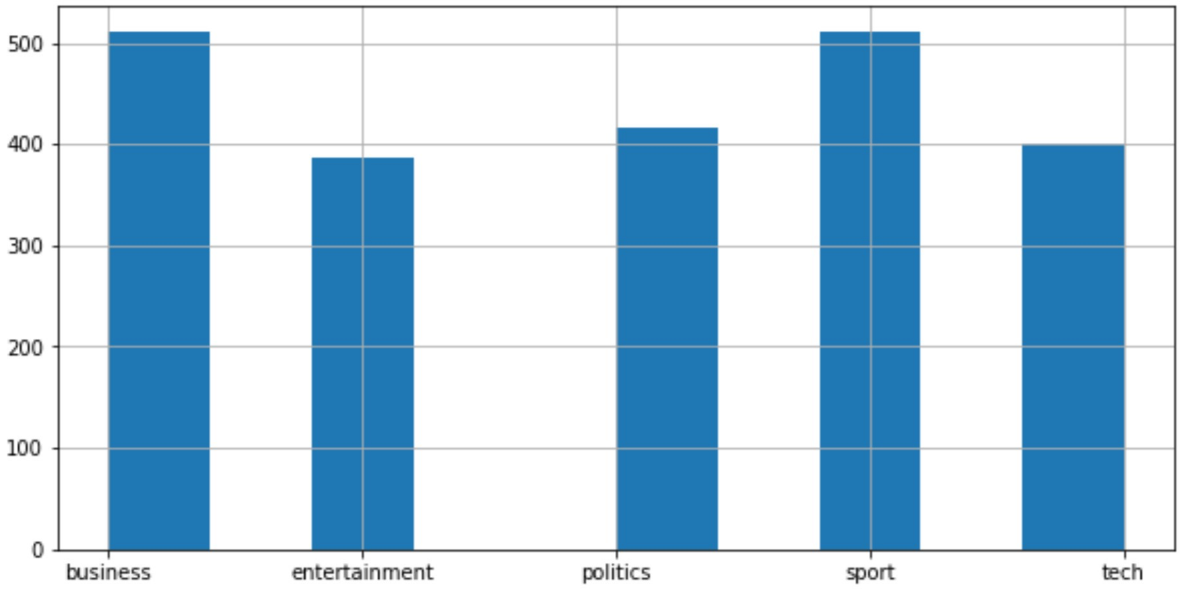
	text	labels

In [ ]:
```
len(df)
```

Out[ ]:
```
2225
```

In [ ]:
```
inputs = df['text']
labels = df['labels']
```

In [ ]:
```
labels.hist(figsize=(10, 5));
```



In [ ]:
```
inputs_train, inputs_test, Ytrain, Ytest = train_test_split(
    inputs, labels, random_state=123)
```

In [ ]:
```
vectorizer = CountVectorizer()
```

In [ ]:
```
Xtrain = vectorizer.fit_transform(inputs_train)
Xtest = vectorizer.transform(inputs_test)
```

In [ ]:
```
Xtrain
```

Out[ ]:
```
<1668x26287 sparse matrix of type '<class 'numpy.int64'>'
	with 337411 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
(Xtrain != 0).sum()
```

Out[ ]:
```
337411
```

In [ ]:

```
# what percentage of values are non-zero?
(Xtrain != 0).sum() / np.prod(Xtrain.shape)
```

Out[ ]:

```
0.007695239935415004
```

In [ ]:

```
model = MultinomialNB()
model.fit(Xtrain, Ytrain)
print("train score:", model.score(Xtrain, Ytrain))
print("test score:", model.score(Xtest, Ytest))
```

```
train score: 0.9922062350119905
test score: 0.9712746858168761
```

In [ ]:

```
# with stopwords
vectorizer = CountVectorizer(stop_words='english')
Xtrain = vectorizer.fit_transform(inputs_train)
Xtest = vectorizer.transform(inputs_test)
model = MultinomialNB()
model.fit(Xtrain, Ytrain)
print("train score:", model.score(Xtrain, Ytrain))
print("test score:", model.score(Xtest, Ytest))
```

```
train score: 0.9928057553956835
test score: 0.9766606822262118
```

In [ ]:

```
def get_wordnet_pos(treebank_tag):
  if treebank_tag.startswith('J'):
    return wordnet.ADJ
  elif treebank_tag.startswith('V'):
    return wordnet.VERB
  elif treebank_tag.startswith('N'):
    return wordnet.NOUN
  elif treebank_tag.startswith('R'):
    return wordnet.ADV
  else:
    return wordnet.NOUN
```

In [ ]:

```
class LemmaTokenizer:
  def __init__(self):
    self.wnl = WordNetLemmatizer()
  def __call__(self, doc):
    tokens = word_tokenize(doc)
    words_and_tags = nltk.pos_tag(tokens)
    return [self.wnl.lemmatize(word, pos=get_wordnet_pos(tag)) \
            for word, tag in words_and_tags]
```

In [ ]:

```
# with lemmatization
vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())
Xtrain = vectorizer.fit_transform(inputs_train)
Xtest = vectorizer.transform(inputs_test)
model = MultinomialNB()
model.fit(Xtrain, Ytrain)
print("train score:", model.score(Xtrain, Ytrain))
print("test score:", model.score(Xtest, Ytest))
```

```
train score: 0.9922062350119905
test score: 0.9676840215439856
```

In [ ]:

```
class StemTokenizer:
  def __init__(self):
    self.porter = PorterStemmer()
  def __call__(self, doc):
    tokens = word_tokenize(doc)
    return [self.porter.stem(t) for t in tokens]
```

In [ ]:

```
# with stemming
vectorizer = CountVectorizer(tokenizer=StemTokenizer())
Xtrain = vectorizer.fit_transform(inputs_train)
Xtest = vectorizer.transform(inputs_test)
model = MultinomialNB()
model.fit(Xtrain, Ytrain)
print("train score:", model.score(Xtrain, Ytrain))
print("test score:", model.score(Xtest, Ytest))
```

```
train score: 0.9892086330935251
test score: 0.9694793536804309
```

In [ ]:

```
def simple_tokenizer(s):
  return s.split()
```

In [ ]:

```
# string split tokenizer
vectorizer = CountVectorizer(tokenizer=simple_tokenizer)
Xtrain = vectorizer.fit_transform(inputs_train)
Xtest = vectorizer.transform(inputs_test)
model = MultinomialNB()
model.fit(Xtrain, Ytrain)
print("train score:", model.score(Xtrain, Ytrain))
print("test score:", model.score(Xtest, Ytest))
```

```
train score: 0.9952038369304557
test score: 0.9712746858168761
```

In [ ]:

```
# What is the vector dimensionality in each case?
# Compare them and consider why they are larger / smaller
```