Exploratory Data Analysis

In [ ]:

```
!wget https://raw.githubusercontent.com/miguelfzafra/Latest-News-Classifier/master/0.%20Lat
```

```
--2022-09-08 00:01:35--  https://raw.githubusercontent.com/miguelfzafra/Late
st-News-Classifier/master/0.%20Latest%20News%20Classifier/01.%20Dataset%20Cr
eation/News_dataset.csv (https://raw.githubusercontent.com/miguelfzafra/Late
st-News-Classifier/master/0.%20Latest%20News%20Classifier/01.%20Dataset%20Cr
eation/News_dataset.csv)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.1
09.133, 185.199.111.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.
109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5155701 (4.9M) [text/plain]
Saving to: ‘News_dataset.csv’

News_dataset.csv    100%[===================>]   4.92M  --.-KB/s    in 0.02s

2022-09-08 00:01:35 (233 MB/s) - ‘News_dataset.csv’ saved [5155701/5155701]
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import seaborn as sns
sns.set_style("whitegrid")
import altair as alt


# Code for hiding seaborn warnings
import warnings
warnings.filterwarnings("ignore")
```

Getting the data

In [9]:

```
df_path = "/Data Creation/News_dataset.csv"
df = pd.read_csv(df_path, sep=';',on_bad_lines='skip')
```

In [10]:

```
df.head()
```

Out[10]:

	File_Name	Content	Category	Complete_Filename
0	001.txt	Ad sales boost Time Warner profit\n\nQuarterly...	business	001.txt-business
1	002.txt	Dollar gains on Greenspan speech\n\nThe dollar...	business	002.txt-business
2	003.txt	Yukos unit buyer faces loan claim\n\nThe owner...	business	003.txt-business
3	004.txt	High fuel prices hit BA's profits\n\nBritish A...	business	004.txt-business
4	005.txt	Pernod takeover talk lifts Domecq\n\nShares in...	business	005.txt-business

##No of Articles in Each Category

1. List item
2. List item

In [11]:

```
bars = alt.Chart(df).mark_bar(size=50).encode(
    x=alt.X("Category"),
    y=alt.Y("count():Q", axis=alt.Axis(title='Number of articles')),
    tooltip=[alt.Tooltip('count()', title='Number of articles'), 'Category'],
    color='Category'

)

text = bars.mark_text(
    align='center',
    baseline='bottom',
).encode(
    text='count()'
)

(bars + text).interactive().properties(
    height=300,
    width=700,
    title = "Number of articles in each category",
)
```

Out[11]:

% of articles in each category

In [12]:

```
df['id'] = 1
df2 = pd.DataFrame(df.groupby('Category').count()['id']).reset_index()

bars = alt.Chart(df2).mark_bar(size=50).encode(
    x=alt.X('Category'),
    y=alt.Y('PercentOfTotal:Q', axis=alt.Axis(format='.0%', title='% of Articles')),
    color='Category'
).transform_window(
    TotalArticles='sum(id)',
    frame=[None, None]
).transform_calculate(
    PercentOfTotal="datum.id / datum.TotalArticles"
)

text = bars.mark_text(
    align='center',
    baseline='bottom',
    #dx=5  # Nudges text to right so it doesn't appear on top of the bar
).encode(
    text=alt.Text('PercentOfTotal:Q', format='.1%')
)

(bars + text).interactive().properties(
    height=300,
    width=700,
    title = "% of articles in each category",
)
```

Out[12]:

The classes are approximately balanced. We'll first try to train the models without oversampling/undersampling. If we see some bias in the model, we'll use these techniques.

News length by category

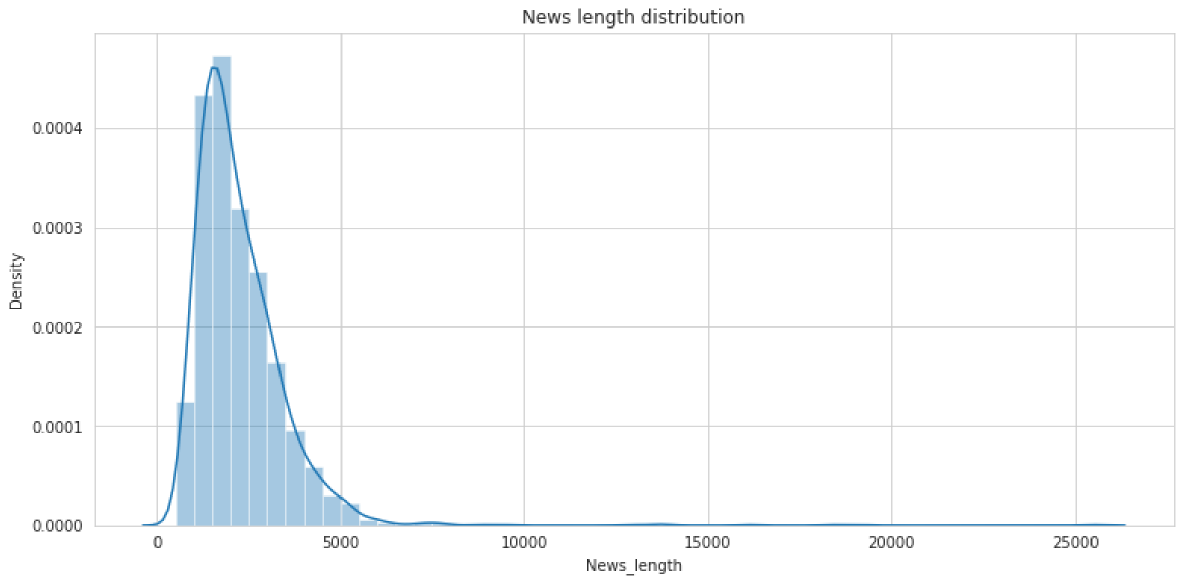
Definition of news length field. Although there are special characters in the text (`\r`, `\n`), it will be useful as an approximation.

In [13]:

```
df['News_length'] = df['Content'].str.len()
```

In [14]:

```
plt.figure(figsize=(12.8,6))
sns.distplot(df['News_length']).set_title('News length distribution');
```



In [15]:

```
df['News_length'].describe()
```

Out[15]:

```
count     2225.000000
mean      2264.790562
std       1364.305951
min        502.000000
25%       1447.000000
50%       1966.000000
75%       2803.000000
max      25484.000000
Name: News_length, dtype: float64
```

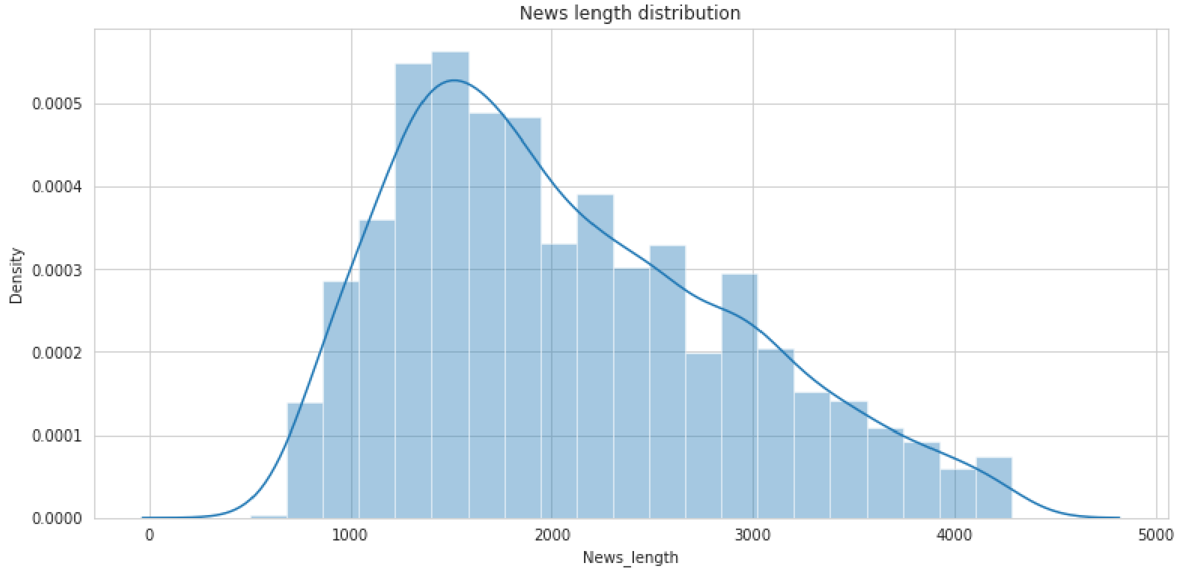
Let's remove from the 95% percentile onwards to better appreciate the histogram:

In [16]:

```
quantile_95 = df['News_length'].quantile(0.95)
df_95 = df[df['News_length'] < quantile_95]
```

In [17]:

```
plt.figure(figsize=(12.8,6))
sns.distplot(df_95['News_length']).set_title('News length distribution');
```



We can get the number of news articles with more than 10,000 characters:

In [18]:

```
df_more10k = df[df['News_length'] > 110]
len(df_more10k)
```

Out[18]:

```
2225
```

Let's see one:

In [19]:

```
df_more10k['Content'].iloc[0]
```

Out[19]:

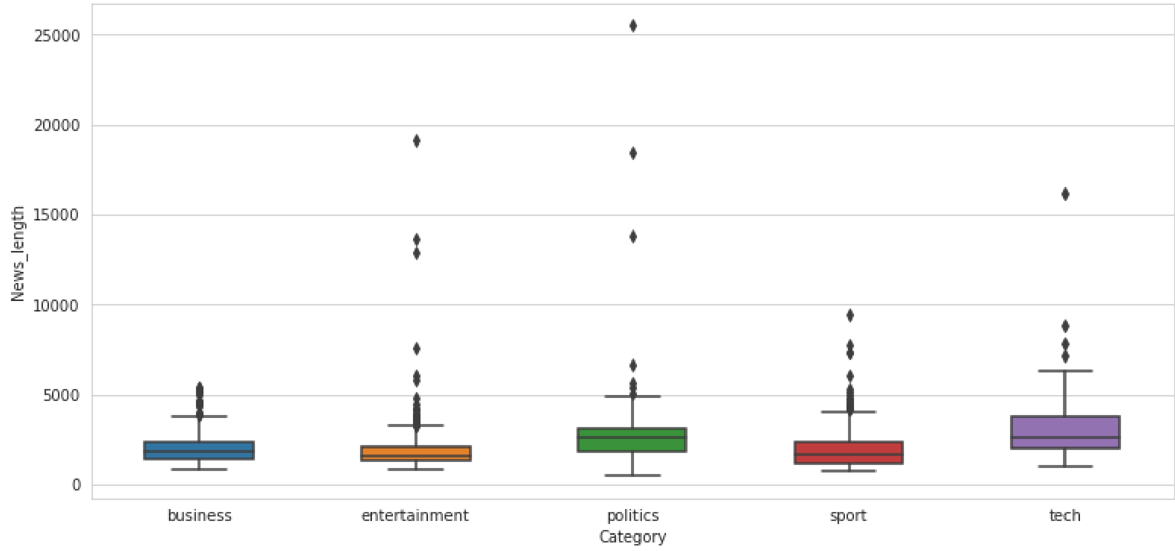
```
'Ad sales boost Time Warner profit\n\nQuarterly profits at US media giant Ti
meWarner jumped 76% to $1.13bn (Â£600m) for the three months to December, fr
om $639m year-earlier.\n\nThe firm, which is now one of the biggest investor
s in Google, benefited from sales of high-speed internet connections and hig
her advert sales. TimeWarner said fourth quarter sales rose 2% to $11.1bn fr
om $10.9bn. Its profits were buoyed by one-off gains which offset a profit d
ip at Warner Bros, and less users for AOL.\n\nTime Warner said on Friday tha
t it now owns 8% of search-engine Google. But its own internet business, AO
L, had has mixed fortunes. It lost 464,000 subscribers in the fourth quarter
profits were lower than in the preceding three quarters. However, the compan
y said AOL\'s underlying profit before exceptional items rose 8% on the back
of stronger internet advertising revenues. It hopes to increase subscribers
by offering the online service free to TimeWarner internet customers and wil
l try to sign up AOL\'s existing customers for high-speed broadband. TimeWar
ner also has to restate 2000 and 2003 results following a probe by the US Se
curities Exchange Commission (SEC), which is close to concluding.\n\nTime Wa
rner\'s fourth quarter profits were slightly better than analysts\' expectat
ions. But its film division saw profits slump 27% to $284m, helped by box-of
fice flops Alexander and Catwoman, a sharp contrast to year-earlier, when th
e third and final film in the Lord of the Rings trilogy boosted results. For
the full-year, TimeWarner posted a profit of $3.36bn, up 27% from its 2003 p
erformance, while revenues grew 6.4% to $42.09bn. "Our financial performance
was strong, meeting or exceeding all of our full-year objectives and greatly
enhancing our flexibility," chairman and chief executive Richard Parsons sai
d. For 2005, TimeWarner is projecting operating earnings growth of around
5%, and also expects higher revenue and wider profit margins.\n\nTimeWarner
is to restate its accounts as part of efforts to resolve an inquiry into AOL
by US market regulators. It has already offered to pay $300m to settle charg
es, in a deal that is under review by the SEC. The company said it was unabl
e to estimate the amount it needed to set aside for legal reserves, which it
previously set at $500m. It intends to adjust the way it accounts for a deal
with German music publisher Bertelsmann\'s purchase of a stake in AOL Europ
e, which it had reported as advertising revenue. It will now book the sale o
f its stake in AOL Europe as a loss on the value of that stake.'
```

It's just a large news article.

Let's now plot a boxplot:

In [20]:

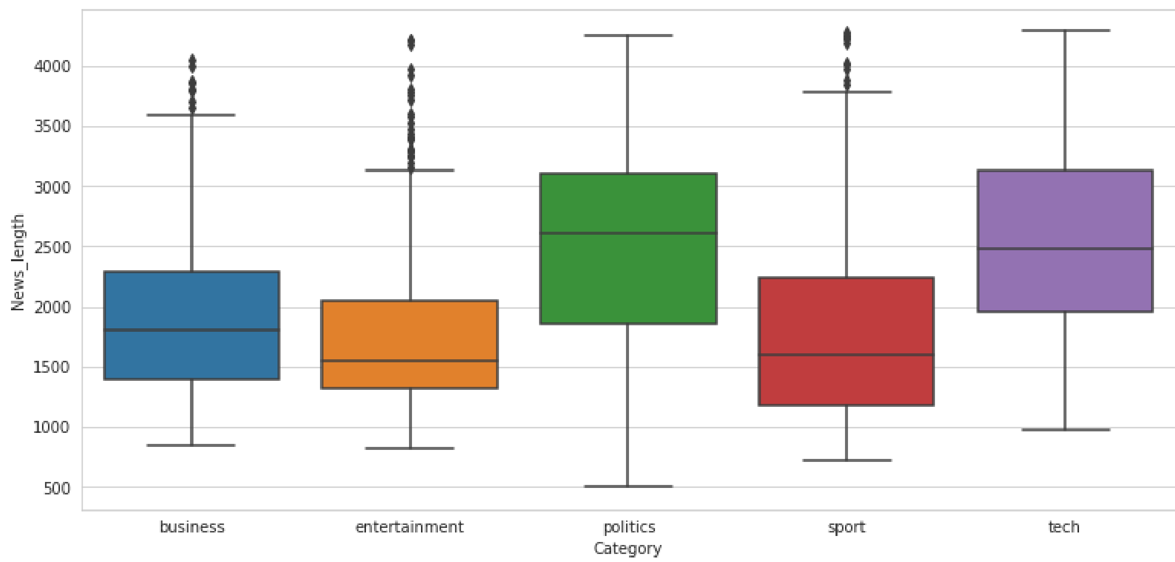
```
plt.figure(figsize=(12.8,6))
sns.boxplot(data=df, x='Category', y='News_length', width=.5);
```



Now, let's remove the larger documents for better comprehension:

In [21]:

```
plt.figure(figsize=(12.8,6))
sns.boxplot(data=df_95, x='Category', y='News_length');
```



We can see that, although the length distribution is different for every category, the difference is not too big. If we had way too different lengths between categories we would have a problem since the feature creation process may take into account counts of words. However, when creating the features with TF-IDF scoring, we will normalize the features just to avoid this.

At this point, we cannot do further Exploratory Data Analysis. We'll turn onto the **Feature Engineering** section.

We'll save the dataset:

In [22]:

```
with open('News_dataset.pickle', 'wb') as output:
    pickle.dump(df, output)
```

In [ ]:

```
import pickle
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import chi2
import numpy as np
```

In [23]:

```
path_df = "/content/News_dataset.pickle"

with open(path_df, 'rb') as data:
    df = pickle.load(data)
```

In [24]:

```
df.head()
```

Out[24]:

	File_Name	Content	Category	Complete_Filename	id	News_length
0	001.txt	Ad sales boost Time Warner profit\n\nQuarterly...	business	001.txt-business	1	2559
1	002.txt	Dollar gains on Greenspan speech\n\nThe dollar...	business	002.txt-business	1	2251
2	003.txt	Yukos unit buyer faces loan claim\n\nThe owner...	business	003.txt-business	1	1551
3	004.txt	High fuel prices hit BA's profits\n\nBritish A...	business	004.txt-business	1	2411
4	005.txt	Pernod takeover talk lifts Domecq\n\nShares in...	business	005.txt-business	1	1569

In [25]:

```
df.loc[1]['Content']
```

Out[25]:

```
'Dollar gains on Greenspan speech\n\nThe dollar has hit its highest level ag
ainst the euro in almost three months after the Federal Reserve head said th
e US trade deficit is set to stabilise.\n\nAnd Alan Greenspan highlighted th
e US government\'s willingness to curb spending and rising household savings
as factors which may help to reduce it. In late trading in New York, the dol
lar reached $1.2871 against the euro, from $1.2974 on Thursday. Market conce
rns about the deficit has hit the greenback in recent months. On Friday, Fed
eral Reserve chairman Mr Greenspan\'s speech in London ahead of the meeting
of G7 finance ministers sent the dollar higher after it had earlier tumbled
on the back of worse-than-expected US jobs data. "I think the chairman\'s ta
king a much more sanguine view on the current account deficit than he\'s tak
en for some time," said Robert Sinche, head of currency strategy at Bank of
America in New York. "He\'s taking a longer-term view, laying out a set of c
onditions under which the current account deficit can improve this year and
next."\n\nWorries about the deficit concerns about China do, however, remai
n. China\'s currency remains pegged to the dollar and the US currency\'s sha
rp falls in recent months have therefore made Chinese export prices highly c
ompetitive. But calls for a shift in Beijing\'s policy have fallen on deaf e
ars, despite recent comments in a major Chinese newspaper that the "time is
ripe" for a loosening of the peg. The G7 meeting is thought unlikely to prod
uce any meaningful movement in Chinese policy. In the meantime, the US Feder
al Reserve\'s decision on 2 February to boost interest rates by a quarter of
a point - the sixth such move in as many months - has opened up a differenti
al with European rates. The half-point window, some believe, could be enough
to keep US assets looking more attractive, and could help prop up the dolla
r. The recent falls have partly been the result of big budget deficits, as w
ell as the US\'s yawning current account gap, both of which need to be funde
d by the buying of US bonds and assets by foreign firms and governments. The
White House will announce its budget on Monday, and many commentators believ
e the deficit will remain at close to half a trillion dollars.'
```

In [26]:

```
# \r and \n
df['Content_Parsed_1'] = df['Content'].str.replace("\r", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("\n", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("    ", " ")
```

In [27]:

```
text = "Mr Greenspan\'s"
text
```

Out[27]:

```
"Mr Greenspan's"
```

In [28]:

```
# " when quoting text
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace('"', '')
```

In [29]:

```
# Lowercasing the text
df['Content_Parsed_2'] = df['Content_Parsed_1'].str.lower()
```

In [30]:

```
punctuation_signs = list("?:!.,;")
df['Content_Parsed_3'] = df['Content_Parsed_2']

for punct_sign in punctuation_signs:
    df['Content_Parsed_3'] = df['Content_Parsed_3'].str.replace(punct_sign, '')
```

In [31]:

```
df['Content_Parsed_4'] = df['Content_Parsed_3'].str.replace("'s", "")
```

In [33]:

```
import nltk
```

In [34]:

```
# Downloading punkt and wordnet from NLTK
nltk.download('omw-1.4')
nltk.download('punkt')
print("------------------------------------------------------------")
nltk.download('wordnet')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.

------------------------------------------------------------

[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Out[34]:

```
True
```

In [35]:

```
from nltk import WordNetLemmatizer
```

In [36]:

```
# Saving the lemmatizer into an object
wordnet_lemmatizer = WordNetLemmatizer()
```

In [37]:

```
nrows = len(df)
lemmatized_text_list = []

for row in range(0, nrows):

    # Create an empty list containing lemmatized words
    lemmatized_list = []

    # Save the text and its words into an object
    text = df.loc[row]['Content_Parsed_4']
    text_words = text.split(" ")

    # Iterate through every word to lemmatize
    for word in text_words:
        lemmatized_list.append(wordnet_lemmatizer.lemmatize(word, pos="v"))

    # Join the list
    lemmatized_text = " ".join(lemmatized_list)

    # Append to the list containing the texts
    lemmatized_text_list.append(lemmatized_text)
```

In [38]:

```
df['Content_Parsed_5'] = lemmatized_text_list
```

In [39]:

```
# Downloading the stop words list
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[39]:

```
True
```

In [41]:

```
from nltk.corpus import stopwords
```

In [42]:

```
# Loading the stop words in english
stop_words = list(stopwords.words('english'))
```

In [43]:

```
stop_words[0:10]
```

Out[43]:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e"]
```

In [45]:

```
import re
```

In [46]:

```
example = "me eating a meal"
word = "me"

# The regular expression is:
regex = r"\b" + word + r"\b"  # we need to build it like that to work properly

re.sub(regex, "StopWord", example)
```

Out[46]:

```
'StopWord eating a meal'
```

In [47]:

```
df['Content_Parsed_6'] = df['Content_Parsed_5']

for stop_word in stop_words:

    regex_stopword = r"\b" + stop_word + r"\b"
    df['Content_Parsed_6'] = df['Content_Parsed_6'].str.replace(regex_stopword, '')
```

In [48]:

```
df.loc[5]['Content']
```

Out[48]:

```
'Japan narrowly escapes recession\n\nJapan\'s economy teetered on the brink
of a technical recession in the three months to September, figures show.\n\n
Revised figures indicated growth of just 0.1% - and a similar-sized contract
ion in the previous quarter. On an annual basis, the data suggests annual gr
owth of just 0.2%, suggesting a much more hesitant recovery than had previou
sly been thought. A common technical definition of a recession is two succes
sive quarters of negative growth.\n\nThe government was keen to play down th
e worrying implications of the data. "I maintain the view that Japan\'s econ
omy remains in a minor adjustment phase in an upward climb, and we will moni
tor developments carefully," said economy minister Heizo Takenaka. But in th
e face of the strengthening yen making exports less competitive and indicati
ons of weakening economic conditions ahead, observers were less sanguine. "I
t\'s painting a picture of a recovery... much patchier than previously thoug
ht," said Paul Sheard, economist at Lehman Brothers in Tokyo. Improvements i
n the job market apparently have yet to feed through to domestic demand, wit
h private consumption up just 0.2% in the third quarter.'
```

In [49]:

```
df.loc[5]['Content_Parsed_1']
```

Out[49]:

```
"Japan narrowly escapes recession  Japan's economy teetered on the brink of
a technical recession in the three months to September, figures show.  Revis
ed figures indicated growth of just 0.1% - and a similar-sized contraction i
n the previous quarter. On an annual basis, the data suggests annual growth
of just 0.2%, suggesting a much more hesitant recovery than had previously b
een thought. A common technical definition of a recession is two successive
quarters of negative growth.  The government was keen to play down the worry
ing implications of the data. I maintain the view that Japan's economy remai
ns in a minor adjustment phase in an upward climb, and we will monitor devel
opments carefully, said economy minister Heizo Takenaka. But in the face of
the strengthening yen making exports less competitive and indications of wea
kening economic conditions ahead, observers were less sanguine. It's paintin
g a picture of a recovery... much patchier than previously thought, said Pau
l Sheard, economist at Lehman Brothers in Tokyo. Improvements in the job mar
ket apparently have yet to feed through to domestic demand, with private con
sumption up just 0.2% in the third quarter."
```

In [50]:

```
df.loc[5]['Content_Parsed_2']
```

Out[50]:

```
"japan narrowly escapes recession  japan's economy teetered on the brink of
a technical recession in the three months to september, figures show.  revis
ed figures indicated growth of just 0.1% - and a similar-sized contraction i
n the previous quarter. on an annual basis, the data suggests annual growth
of just 0.2%, suggesting a much more hesitant recovery than had previously b
een thought. a common technical definition of a recession is two successive
quarters of negative growth.  the government was keen to play down the worry
ing implications of the data. i maintain the view that japan's economy remai
ns in a minor adjustment phase in an upward climb, and we will monitor devel
opments carefully, said economy minister heizo takenaka. but in the face of
the strengthening yen making exports less competitive and indications of wea
kening economic conditions ahead, observers were less sanguine. it's paintin
g a picture of a recovery... much patchier than previously thought, said pau
l sheard, economist at lehman brothers in tokyo. improvements in the job mar
ket apparently have yet to feed through to domestic demand, with private con
sumption up just 0.2% in the third quarter."
```

In [51]:

```
df.loc[5]['Content_Parsed_3']
```

Out[51]:

```
"japan narrowly escapes recession  japan's economy teetered on the brink of
a technical recession in the three months to september figures show  revised
figures indicated growth of just 01% - and a similar-sized contraction in th
e previous quarter on an annual basis the data suggests annual growth of jus
t 02% suggesting a much more hesitant recovery than had previously been thou
ght a common technical definition of a recession is two successive quarters
of negative growth  the government was keen to play down the worrying implic
ations of the data i maintain the view that japan's economy remains in a min
or adjustment phase in an upward climb and we will monitor developments care
fully said economy minister heizo takenaka but in the face of the strengthen
ing yen making exports less competitive and indications of weakening economi
c conditions ahead observers were less sanguine it's painting a picture of a
recovery much patchier than previously thought said paul sheard economist at
lehman brothers in tokyo improvements in the job market apparently have yet
to feed through to domestic demand with private consumption up just 02% in t
he third quarter"
```

In [52]:

```
df.loc[5]['Content_Parsed_4']
```

Out[52]:

```
'japan narrowly escapes recession  japan economy teetered on the brink of a
technical recession in the three months to september figures show  revised f
igures indicated growth of just 01% - and a similar-sized contraction in the
previous quarter on an annual basis the data suggests annual growth of just
02% suggesting a much more hesitant recovery than had previously been though
t a common technical definition of a recession is two successive quarters of
negative growth  the government was keen to play down the worrying implicati
ons of the data i maintain the view that japan economy remains in a minor ad
justment phase in an upward climb and we will monitor developments carefully
said economy minister heizo takenaka but in the face of the strengthening ye
n making exports less competitive and indications of weakening economic cond
itions ahead observers were less sanguine it painting a picture of a recover
y much patchier than previously thought said paul sheard economist at lehman
brothers in tokyo improvements in the job market apparently have yet to feed
through to domestic demand with private consumption up just 02% in the third
quarter'
```

In [53]:

```
df.loc[5]['Content_Parsed_5']
```

Out[53]:

```
'japan narrowly escape recession  japan economy teeter on the brink of a tec
hnical recession in the three months to september figure show  revise figure
indicate growth of just 01% - and a similar-sized contraction in the previou
s quarter on an annual basis the data suggest annual growth of just 02% sugg
est a much more hesitant recovery than have previously be think a common tec
hnical definition of a recession be two successive quarter of negative growt
h  the government be keen to play down the worry implications of the data i
maintain the view that japan economy remain in a minor adjustment phase in a
n upward climb and we will monitor developments carefully say economy minist
er heizo takenaka but in the face of the strengthen yen make export less com
petitive and indications of weaken economic condition ahead observers be les
s sanguine it paint a picture of a recovery much patchier than previously th
ink say paul sheard economist at lehman brothers in tokyo improvements in th
e job market apparently have yet to fee through to domestic demand with priv
ate consumption up just 02% in the third quarter'
```

In [54]:

```
df.head(1)
```

Out[54]:

	File_Name	Content	Category	Complete_Filename	id	News_length	Content_Pars
0	001.txt	Ad sales boost Time Warner profit\n\nQuarterly...	business	001.txt-business	1	2559	Ad sales b Time Warner Quarterl

In [55]:

```
list_columns = ["File_Name", "Category", "Complete_Filename", "Content", "Content_Parsed_6"
df = df[list_columns]

df = df.rename(columns={'Content_Parsed_6': 'Content_Parsed'})
```

In [56]:

```
df.head()
```

Out[56]:

	File_Name	Category	Complete_Filename	Content	Content_Parsed
0	001.txt	business	001.txt-business	Ad sales boost Time Warner profit\n\nQuarterly...	ad sales boost time warner profit quarterly p...
1	002.txt	business	002.txt-business	Dollar gains on Greenspan speech\n\nThe dollar...	dollar gain greenspan speech dollar hit h...
2	003.txt	business	003.txt-business	Yukos unit buyer faces loan claim\n\nThe owner...	yukos unit buyer face loan claim owners emb...
3	004.txt	business	004.txt-business	High fuel prices hit BA's profits\n\nBritish A...	high fuel price hit ba profit british airways...
4	005.txt	business	005.txt-business	Pernod takeover talk lifts Domecq\n\nShares in...	pernod takeover talk lift domecq share uk dr...

In [57]:

```
category_codes = {
    'business': 0,
    'entertainment': 1,
    'politics': 2,
    'sport': 3,
    'tech': 4
}
```

In [58]:

```
# Category mapping
df['Category_Code'] = df['Category']
df = df.replace({'Category_Code':category_codes})
```

In [59]:

```
df.head()
```

Out[59]:

	File_Name	Category	Complete_Filename	Content	Content_Parsed	Category_Code
0	001.txt	business	001.txt-business	Ad sales boost Time Warner profit\n\nQuarterly...	ad sales boost time warner profit quarterly p...	(
1	002.txt	business	002.txt-business	Dollar gains on Greenspan speech\n\nThe dollar...	dollar gain greenspan speech dollar hit h...	(
2	003.txt	business	003.txt-business	Yukos unit buyer faces loan claim\n\nThe owner...	yukos unit buyer face loan claim owners emb...	(
3	004.txt	business	004.txt-business	High fuel prices hit BA's profits\n\nBritish A...	high fuel price hit ba profit british airways...	(
4	005.txt	business	005.txt-business	Pernod takeover talk lifts Domecq\n\nShares in...	pernod takeover talk lift domecq share uk dr...	(

In [62]:

```
from sklearn.model_selection import train_test_split
```

In [63]:

```
X_train, X_test, y_train, y_test = train_test_split(df['Content_Parsed'],
                                                    df['Category_Code'],
                                                    test_size=0.15,
                                                    random_state=8)
```

In [64]:

```
# Parameter election
ngram_range = (1,2)
min_df = 10
max_df = 1.
max_features = 300
```

In [66]:

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

In [67]:

```
tfidf = TfidfVectorizer(encoding='utf-8',
                        ngram_range=ngram_range,
                        stop_words=None,
                        lowercase=False,
                        max_df=max_df,
                        min_df=min_df,
                        max_features=max_features,
                        norm='l2',
                        sublinear_tf=True)

features_train = tfidf.fit_transform(X_train).toarray()
labels_train = y_train
print(features_train.shape)

features_test = tfidf.transform(X_test).toarray()
labels_test = y_test
print(features_test.shape)
```

```
(1891, 300)
(334, 300)
```

In [68]:

```
from sklearn.feature_selection import chi2
import numpy as np

for Product, category_id in sorted(category_codes.items()):
    features_chi2 = chi2(features_train, labels_train == category_id)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf.get_feature_names())[indices]
    unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
    bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
    print("# '{}' category:".format(Product))
    print("  . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-5:])))
    print("  . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-2:])))
    print("")
```

```
# 'business' category:
  . Most correlated unigrams:
. market
. price
. economy
. growth
. bank
  . Most correlated bigrams:
. last year
. year old

# 'entertainment' category:
  . Most correlated unigrams:
. tv
. music
. star
. award
. film
  . Most correlated bigrams:
. mr blair
. prime minister

# 'politics' category:
  . Most correlated unigrams:
. minister
. blair
. party
. election
. labour
  . Most correlated bigrams:
. prime minister
. mr blair

# 'sport' category:
  . Most correlated unigrams:
. win
. side
. game
. team
. match
  . Most correlated bigrams:
. say mr
. year old
```

```
# 'tech' category:
  . Most correlated unigrams:
. digital
. technology
. computer
. software
. users
  . Most correlated bigrams:
. year old
. say mr
```

In [69]:

```
bigrams
```

Out[69]:

```
['tell bbc', 'last year', 'prime minister', 'mr blair', 'year old', 'say m
r']
```

In [70]:

```
# X_train
with open('X_train.pickle', 'wb') as output:
  pickle.dump(X_train, output)

# X_test
with open('X_test.pickle', 'wb') as output:
    pickle.dump(X_test, output)

# y_train
with open('y_train.pickle', 'wb') as output:
    pickle.dump(y_train, output)

# y_test
with open('y_test.pickle', 'wb') as output:
    pickle.dump(y_test, output)

# df
with open('df.pickle', 'wb') as output:
    pickle.dump(df, output)

# features_train
with open('features_train.pickle', 'wb') as output:
    pickle.dump(features_train, output)

# labels_train
with open('labels_train.pickle', 'wb') as output:
    pickle.dump(labels_train, output)

# features_test
with open('features_test.pickle', 'wb') as output:
    pickle.dump(features_test, output)

# labels_test
with open('labels_test.pickle', 'wb') as output:
    pickle.dump(labels_test, output)

# TF-IDF object
with open('tfidf.pickle', 'wb') as output:
    pickle.dump(tfidf, output)
```

In [71]:

```
# Parameter election
ngram_range = (1,2)
min_df = 10
max_df = 1.
max_features = 300
```

In [72]:

```
tfidf = TfidfVectorizer(encoding='utf-8',
                        ngram_range=ngram_range,
                        stop_words=None,
                        lowercase=False,
                        max_df=max_df,
                        min_df=min_df,
                        max_features=max_features,
                        norm='l2',
                        sublinear_tf=True)

features_train = tfidf.fit_transform(X_train).toarray()
labels_train = y_train
print(features_train.shape)

features_test = tfidf.transform(X_test).toarray()
labels_test = y_test
print(features_test.shape)
```

```
(1891, 300)
(334, 300)
```

In [73]:

```
from sklearn.feature_selection import chi2
import numpy as np

for Product, category_id in sorted(category_codes.items()):
    features_chi2 = chi2(features_train, labels_train == category_id)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf.get_feature_names())[indices]
    unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
    bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
    print("# '{}' category:".format(Product))
    print("  . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-5:])))
    print("  . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-2:])))
    print("")
```

```
# 'business' category:
  . Most correlated unigrams:
. market
. price
. economy
. growth
. bank
  . Most correlated bigrams:
. last year
. year old

# 'entertainment' category:
  . Most correlated unigrams:
. tv
. music
. star
. award
. film
  . Most correlated bigrams:
. mr blair
. prime minister

# 'politics' category:
  . Most correlated unigrams:
. minister
. blair
. party
. election
. labour
  . Most correlated bigrams:
. prime minister
. mr blair

# 'sport' category:
  . Most correlated unigrams:
. win
. side
. game
. team
. match
  . Most correlated bigrams:
. say mr
. year old
```

```
# 'tech' category:
  . Most correlated unigrams:
. digital
. technology
. computer
. software
. users
  . Most correlated bigrams:
. year old
. say mr
```

In [74]:

```
bigrams
```

Out[74]:

```
['tell bbc', 'last year', 'prime minister', 'mr blair', 'year old', 'say m
r']
```

In [75]:

```
# X_train
with open('X_train.pickle', 'wb') as output:
  pickle.dump(X_train, output)

# X_test
with open('X_test.pickle', 'wb') as output:
    pickle.dump(X_test, output)

# y_train
with open('y_train.pickle', 'wb') as output:
    pickle.dump(y_train, output)

# y_test
with open('y_test.pickle', 'wb') as output:
    pickle.dump(y_test, output)

# df
with open('df.pickle', 'wb') as output:
    pickle.dump(df, output)

# features_train
with open('features_train.pickle', 'wb') as output:
    pickle.dump(features_train, output)

# labels_train
with open('labels_train.pickle', 'wb') as output:
    pickle.dump(labels_train, output)

# features_test
with open('features_test.pickle', 'wb') as output:
    pickle.dump(features_test, output)

# labels_test
with open('labels_test.pickle', 'wb') as output:
    pickle.dump(labels_test, output)

# TF-IDF object
with open('tfidf.pickle', 'wb') as output:
    pickle.dump(tfidf, output)
```

In [76]:

```
import pickle
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import svm
from pprint import pprint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import ShuffleSplit
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

In [77]:

```
# Dataframe
path_df = "df.pickle"
with open(path_df, 'rb') as data:
    df = pickle.load(data)

# features_train
path_features_train = "features_train.pickle"
with open(path_features_train, 'rb') as data:
    features_train = pickle.load(data)

# labels_train
path_labels_train = "labels_train.pickle"
with open(path_labels_train, 'rb') as data:
    labels_train = pickle.load(data)

# features_test
path_features_test = "features_test.pickle"
with open(path_features_test, 'rb') as data:
    features_test = pickle.load(data)

# labels_test
path_labels_test = "labels_test.pickle"
with open(path_labels_test, 'rb') as data:
    labels_test = pickle.load(data)
```

In [78]:

```
print(features_train.shape)
print(features_test.shape)
```

```
(1891, 300)
(334, 300)
```

In [79]:

```
svc_0 =svm.SVC(random_state=8)

print('Parameters currently in use:\n')
pprint(svc_0.get_params())
```

```
Parameters currently in use:

{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': 8,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

In [80]:

```
# C
C = [.0001, .001, .01]

# gamma
gamma = [.0001, .001, .01, .1, 1, 10, 100]

# degree
degree = [1, 2, 3, 4, 5]

# kernel
kernel = ['linear', 'rbf', 'poly']

# probability
probability = [True]

# Create the random grid
random_grid = {'C': C,
              'kernel': kernel,
              'gamma': gamma,
              'degree': degree,
              'probability': probability
             }

pprint(random_grid)
```

```
{'C': [0.0001, 0.001, 0.01],
 'degree': [1, 2, 3, 4, 5],
 'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
 'kernel': ['linear', 'rbf', 'poly'],
 'probability': [True]}
```

In [81]:

```
# First create the base model to tune
svc = svm.SVC(random_state=8)

# Definition of the random search
random_search = RandomizedSearchCV(estimator=svc,
                                   param_distributions=random_grid,
                                   n_iter=50,
                                   scoring='accuracy',
                                   cv=3,
                                   verbose=1,
                                   random_state=8)

# Fit the random search model
random_search.fit(features_train, labels_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[81]:

```
RandomizedSearchCV(cv=3, estimator=SVC(random_state=8), n_iter=50,
                   param_distributions={'C': [0.0001, 0.001, 0.01],
                                        'degree': [1, 2, 3, 4, 5],
                                        'gamma': [0.0001, 0.001, 0.01, 0.1,
1,
                                                  10, 100],
                                        'kernel': ['linear', 'rbf', 'poly'],
                                        'probability': [True]},
                   random_state=8, scoring='accuracy', verbose=1)
```

In [82]:

```
print("The best hyperparameters from Random Search are:")
print(random_search.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(random_search.best_score_)
```

```
The best hyperparameters from Random Search are:
{'probability': True, 'kernel': 'poly', 'gamma': 10, 'degree': 4, 'C': 0.01}

The mean accuracy of a model with these hyperparameters is:
0.9217358857612424
```

In [83]:

```
# Create the parameter grid based on the results of random search
C = [.0001, .001, .01, .1]
degree = [3, 4, 5]
gamma = [1, 10, 100]
probability = [True]

param_grid = [
  {'C': C, 'kernel':['linear'], 'probability':probability},
  {'C': C, 'kernel':['poly'], 'degree':degree, 'probability':probability},
  {'C': C, 'kernel':['rbf'], 'gamma':gamma, 'probability':probability}
]

# Create a base model
svc = svm.SVC(random_state=8)

# Manually create the splits in CV in order to be able to fix a random_state (GridSearchCV
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=svc,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1)

# Fit the grid search to the data
grid_search.fit(features_train, labels_train)
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits
```

Out[83]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=SVC(random_state=8),
             param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1], 'kernel': ['linea
r'],
                          'probability': [True]},
                         {'C': [0.0001, 0.001, 0.01, 0.1], 'degree': [3, 4,
5],
                          'kernel': ['poly'], 'probability': [True]},
                         {'C': [0.0001, 0.001, 0.01, 0.1],
                          'gamma': [1, 10, 100], 'kernel': ['rbf'],
                          'probability': [True]}],
             scoring='accuracy', verbose=1)
```

In [84]:

```
print("The best hyperparameters from Grid Search are:")
print(grid_search.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search.best_score_)
```

```
The best hyperparameters from Grid Search are:
{'C': 0.1, 'kernel': 'linear', 'probability': True}

The mean accuracy of a model with these hyperparameters is:
0.9498666666666665
```

In [85]:

```
best_svc = grid_search.best_estimator_
```

In [86]:

```
best_svc
```

Out[86]:

```
SVC(C=0.1, kernel='linear', probability=True, random_state=8)
```

In [87]:

```
best_svc.fit(features_train, labels_train)
```

Out[87]:

```
SVC(C=0.1, kernel='linear', probability=True, random_state=8)
```

In [88]:

```
svc_pred = best_svc.predict(features_test)
```

In [89]:

```
# Training accuracy
print("The training accuracy is: ")
print(accuracy_score(labels_train, best_svc.predict(features_train)))
```

```
The training accuracy is:
0.9592808038075092
```

In [90]:

```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(labels_test, svc_pred))
```

```
The test accuracy is:
0.9401197604790419
```

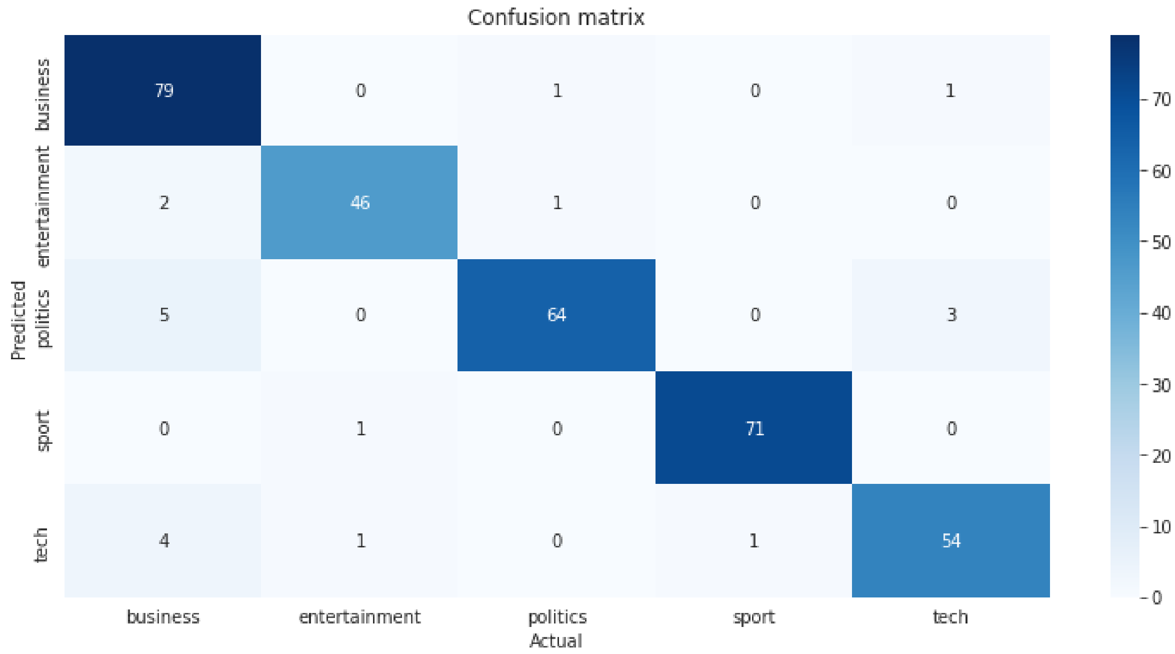
In [91]:

```
# Classification report
print("Classification report")
print(classification_report(labels_test,svc_pred))
```

```
Classification report
              precision    recall  f1-score   support

           0       0.88      0.98      0.92        81
           1       0.96      0.94      0.95        49
           2       0.97      0.89      0.93        72
           3       0.99      0.99      0.99        72
           4       0.93      0.90      0.92        60

    accuracy                           0.94       334
   macro avg       0.94      0.94      0.94       334
weighted avg       0.94      0.94      0.94       334
```

In [92]:

```
aux_df = df[['Category', 'Category_Code']].drop_duplicates().sort_values('Category_Code')
conf_matrix = confusion_matrix(labels_test, svc_pred)
plt.figure(figsize=(12.8,6))
sns.heatmap(conf_matrix,
            annot=True,
            xticklabels=aux_df['Category'].values,
            yticklabels=aux_df['Category'].values,
            cmap="Blues")
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.title('Confusion matrix')
plt.show()
```



In [93]:

```
base_model = svm.SVC(random_state = 8)
base_model.fit(features_train, labels_train)
accuracy_score(labels_test, base_model.predict(features_test))
```

Out[93]:

0.9550898203592815

In [94]:

```
best_svc.fit(features_train, labels_train)
accuracy_score(labels_test, best_svc.predict(features_test))
```

Out[94]:

0.9401197604790419

In [95]:

```
d = {
     'Model': 'SVM',
     'Training Set Accuracy': accuracy_score(labels_train, best_svc.predict(features_train)
     'Test Set Accuracy': accuracy_score(labels_test, svc_pred)
}

df_models_svc = pd.DataFrame(d, index=[0])
```

In [96]:

```
df_models_svc
```

Out[96]:

	Model	Training Set Accuracy	Test Set Accuracy
0	SVM	0.959281	0.94012

In [97]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [98]:

```
# n_estimators
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 1000, num = 5)]

# max_features
max_features = ['auto', 'sqrt']

# max_depth
max_depth = [int(x) for x in np.linspace(20, 100, num = 5)]
max_depth.append(None)

# min_samples_split
min_samples_split = [2, 5, 10]

# min_samples_leaf
min_samples_leaf = [1, 2, 4]

# bootstrap
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [20, 40, 60, 80, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000]}
```

In [99]:

```
# First create the base model to tune
rfc = RandomForestClassifier(random_state=8)

# Definition of the random search
random_search = RandomizedSearchCV(estimator=rfc,
                                   param_distributions=random_grid,
                                   n_iter=50,
                                   scoring='accuracy',
                                   cv=3,
                                   verbose=1,
                                   random_state=8)

# Fit the random search model
random_search.fit(features_train, labels_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[99]:

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=8),
                   n_iter=50,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [20, 40, 60, 80, 100,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000]},
                   random_state=8, scoring='accuracy', verbose=1)
```

In [100]:

```
print("The best hyperparameters from Random Search are:")
print(random_search.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(random_search.best_score_)
```

```
The best hyperparameters from Random Search are:
{'n_estimators': 600, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_f
eatures': 'sqrt', 'max_depth': 100, 'bootstrap': False}

The mean accuracy of a model with these hyperparameters is:
0.9434181068095491
```

In [101]:

```
# Create the parameter grid based on the results of random search
bootstrap = [False]
max_depth = [30, 40, 50]
max_features = ['sqrt']
min_samples_leaf = [1, 2, 4]
min_samples_split = [5, 10, 15]
n_estimators = [800]

param_grid = {
    'bootstrap': bootstrap,
    'max_depth': max_depth,
    'max_features': max_features,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'n_estimators': n_estimators
}

# Create a base model
rfc = RandomForestClassifier(random_state=8)

# Manually create the splits in CV in order to be able to fix a random_state (GridSearchCV
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rfc,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1)

# Fit the grid search to the data
grid_search.fit(features_train, labels_train)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
```

Out[101]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=RandomForestClassifier(random_state=8),
             param_grid={'bootstrap': [False], 'max_depth': [30, 40, 50],
                         'max_features': ['sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [5, 10, 15],
                         'n_estimators': [800]},
             scoring='accuracy', verbose=1)
```

In [104]:

```
print("The best hyperparameters from Grid Search are:")
print(grid_search.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search.best_score_)
```

```
The best hyperparameters from Grid Search are:
{'bootstrap': False, 'max_depth': 40, 'max_features': 'sqrt', 'min_samples_l
eaf': 1, 'min_samples_split': 5, 'n_estimators': 800}

The mean accuracy of a model with these hyperparameters is:
0.9450666666666668
```

In [105]:

```
best_rfc = grid_search.best_estimator_
best_rfc
```

Out[105]:

```
RandomForestClassifier(bootstrap=False, max_depth=40, max_features='sqrt',
                       min_samples_split=5, n_estimators=800, random_state=
8)
```

In [106]:

```
best_rfc.fit(features_train, labels_train)
```

Out[106]:

```
RandomForestClassifier(bootstrap=False, max_depth=40, max_features='sqrt',
                       min_samples_split=5, n_estimators=800, random_state=
8)
```

In [110]:

```
rfc_pred = best_rfc.predict(features_test)
```

In [107]:

```
# Training accuracy
print("The training accuracy is: ")
print(accuracy_score(labels_train, best_rfc.predict(features_train)))
```

```
The training accuracy is:
1.0
```

In [112]:

```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(labels_test, rfc_pred))
```

```
The test accuracy is: 
0.9281437125748503
```

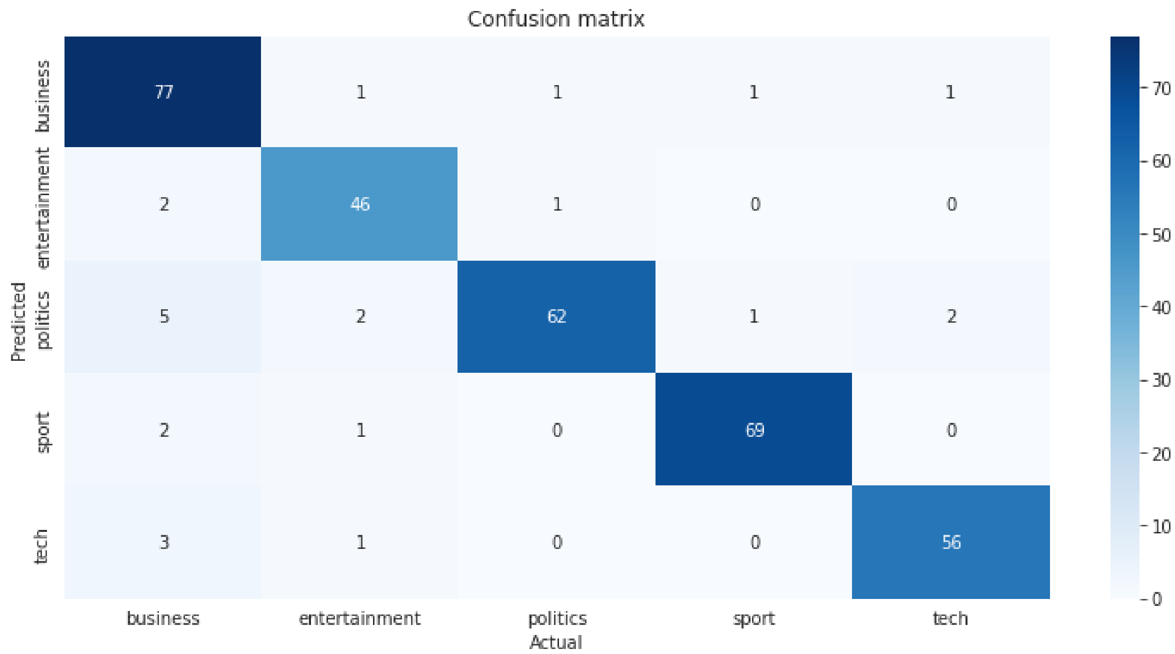
In [113]:

```
# Classification report
print("Classification report")
print(classification_report(labels_test,rfc_pred))
```

```
Classification report
              precision    recall  f1-score   support

           0       0.87      0.95      0.91        81
           1       0.90      0.94      0.92        49
           2       0.97      0.86      0.91        72
           3       0.97      0.96      0.97        72
           4       0.95      0.93      0.94        60

    accuracy                           0.93       334
   macro avg       0.93      0.93      0.93       334
weighted avg       0.93      0.93      0.93       334
```

In [114]:

```
aux_df = df[['Category', 'Category_Code']].drop_duplicates().sort_values('Category_Code')
conf_matrix = confusion_matrix(labels_test, rfc_pred)
plt.figure(figsize=(12.8,6))
sns.heatmap(conf_matrix,
            annot=True,
            xticklabels=aux_df['Category'].values,
            yticklabels=aux_df['Category'].values,
            cmap="Blues")
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.title('Confusion matrix')
plt.show()
```



In [115]:

```
base_model = RandomForestClassifier(random_state = 8)
base_model.fit(features_train, labels_train)
accuracy_score(labels_test, base_model.predict(features_test))
```

Out[115]:

0.9281437125748503

In [116]:

```
best_rfc.fit(features_train, labels_train)
accuracy_score(labels_test, best_rfc.predict(features_test))
```

Out[116]:

0.9281437125748503

In [117]:

```
d = {
     'Model': 'Random Forest',
     'Training Set Accuracy': accuracy_score(labels_train, best_rfc.predict(features_train)
     'Test Set Accuracy': accuracy_score(labels_test, rfc_pred)
}

df_models_rfc = pd.DataFrame(d, index=[0])
df_models_rfc
```

Out[117]:

	Model	Training Set Accuracy	Test Set Accuracy
0	Random Forest	1.0	0.928144

In [118]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [119]:

```
knnc_0 =KNeighborsClassifier()

print('Parameters currently in use:\n')
pprint(knnc_0.get_params())
```

```
Parameters currently in use:

{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [120]:

```
# Create the parameter grid
n_neighbors = [int(x) for x in np.linspace(start = 1, stop = 500, num = 100)]

param_grid = {'n_neighbors': n_neighbors}

# Create a base model
knnc = KNeighborsClassifier()

# Manually create the splits in CV in order to be able to fix a random_state (GridSearchCV
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=knnc,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1)

# Fit the grid search to the data
grid_search.fit(features_train, labels_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[120]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 6, 11, 16, 21, 26, 31, 36, 41, 4
6,
                                         51, 56, 61, 66, 71, 76, 81, 86, 91,
96,
                                         101, 106, 111, 116, 121, 127, 132,
137,
                                         142, 147, ...]},
             scoring='accuracy', verbose=1)
```

In [121]:

```
print("The best hyperparameters from Grid Search are:")
print(grid_search.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search.best_score_)
```

```
The best hyperparameters from Grid Search are:
{'n_neighbors': 6}

The mean accuracy of a model with these hyperparameters is:
0.9477333333333333
```

In [122]:

```
n_neighbors = [1,2,3,4,5,6,7,8,9,10,11]
param_grid = {'n_neighbors': n_neighbors}

knnc = KNeighborsClassifier()
cv_sets = ShuffleSplit(n_splits = 3, test_size = .33, random_state = 8)

grid_search = GridSearchCV(estimator=knnc,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=cv_sets,
                           verbose=1)

grid_search.fit(features_train, labels_train)
```

```
Fitting 3 folds for each of 11 candidates, totalling 33 fits
```

Out[122]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=3, random_state=8, test_size=0.33, tra
in_size=None),
             estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1
1]},
             scoring='accuracy', verbose=1)
```

In [123]:

```
print("The best hyperparameters from Grid Search are:")
print(grid_search.best_params_)
print("")
print("The mean accuracy of a model with these hyperparameters is:")
print(grid_search.best_score_)
```

```
The best hyperparameters from Grid Search are:
{'n_neighbors': 6}

The mean accuracy of a model with these hyperparameters is:
0.9477333333333333
```

In [124]:

```
best_knnc = grid_search.best_estimator_
best_knnc
```

Out[124]:

```
KNeighborsClassifier(n_neighbors=6)
```

In [125]:

```
best_knnc.fit(features_train, labels_train)
```

Out[125]:

```
KNeighborsClassifier(n_neighbors=6)
```

In [126]:

```
knnc_pred = best_knnc.predict(features_test)
```

In [127]:

```
# Training accuracy
print("The training accuracy is: ")
print(accuracy_score(labels_train, best_knnc.predict(features_train)))
```

```
The training accuracy is: 
0.9598096245372819
```

In [128]:

```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(labels_test, knnc_pred))
```

```
The test accuracy is: 
0.9281437125748503
```

In [129]:

```
# Classification report
print("Classification report")
print(classification_report(labels_test,knnc_pred))
```

```
Classification report
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        81
           1       0.93      0.88      0.91        49
           2       0.97      0.92      0.94        72
           3       0.97      0.96      0.97        72
           4       0.86      0.92      0.89        60

    accuracy                           0.93       334
   macro avg       0.93      0.92      0.93       334
weighted avg       0.93      0.93      0.93       334
```

In [130]:

```
base_model = KNeighborsClassifier()
base_model.fit(features_train, labels_train)
accuracy_score(labels_test, base_model.predict(features_test))

best_knnc.fit(features_train, labels_train)
accuracy_score(labels_test, best_knnc.predict(features_test))
```

Out[130]:

```
0.9281437125748503
```

In [131]:

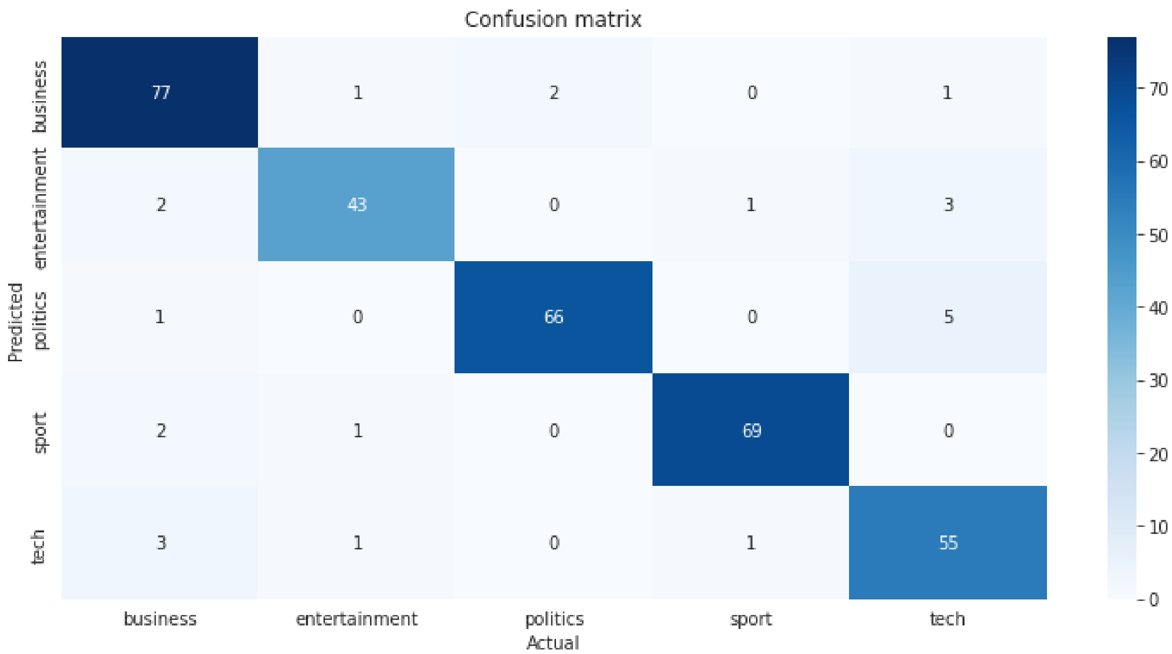
```
d = {
     'Model': 'KNN',
     'Training Set Accuracy': accuracy_score(labels_train, best_knnc.predict(features_train
     'Test Set Accuracy': accuracy_score(labels_test, knnc_pred)
}

df_models_knnc = pd.DataFrame(d, index=[0])
df_models_knnc
```

Out[131]:

	Model	Training Set Accuracy	Test Set Accuracy
0	KNN	0.95981	0.928144

In [140]:

```
aux_df = df[['Category', 'Category_Code']].drop_duplicates().sort_values('Category_Code')
conf_matrix = confusion_matrix(labels_test, knnc_pred)
plt.figure(figsize=(12.8,6))
sns.heatmap(conf_matrix,
            annot=True,
            xticklabels=aux_df['Category'].values,
            yticklabels=aux_df['Category'].values,
            cmap="Blues")
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.title('Confusion matrix')
plt.show()
```



In [132]:

```
from sklearn.naive_bayes import MultinomialNB
```

In [133]:

```
mnbc = MultinomialNB()
mnbc
```

Out[133]:

```
MultinomialNB()
```

In [134]:

```
mnbc.fit(features_train, labels_train)



mnbc_pred = mnbc.predict(features_test)
```

In [135]:

```
# Training accuracy
print("The training accuracy is: ")
print(accuracy_score(labels_train, mnbc.predict(features_train)))
```

```
The training accuracy is:
0.9539925965097832
```

In [136]:

```
# Test accuracy
print("The test accuracy is: ")
print(accuracy_score(labels_test, mnbc_pred))
```

```
The test accuracy is:
0.9341317365269461
```

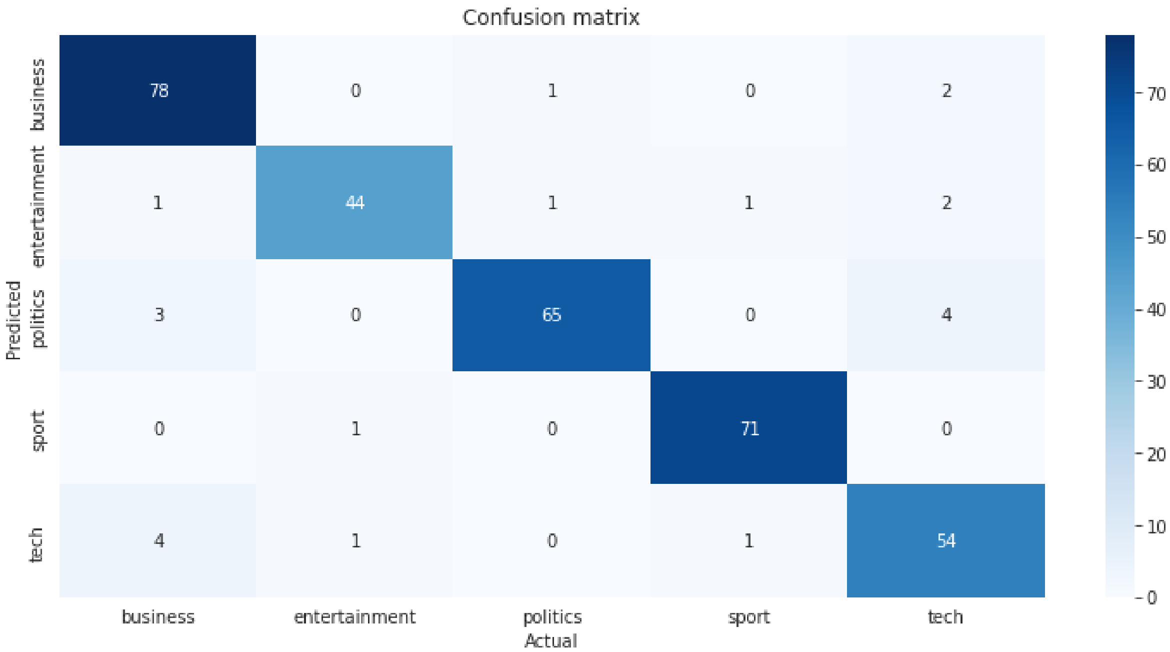
In [137]:

```
# Classification report
print("Classification report")
print(classification_report(labels_test,mnbc_pred))
```

```
Classification report
              precision    recall  f1-score   support

           0       0.91      0.96      0.93        81
           1       0.96      0.90      0.93        49
           2       0.97      0.90      0.94        72
           3       0.97      0.99      0.98        72
           4       0.87      0.90      0.89        60

    accuracy                           0.93       334
   macro avg       0.94      0.93      0.93       334
weighted avg       0.94      0.93      0.93       334
```

In [138]:

```
aux_df = df[['Category', 'Category_Code']].drop_duplicates().sort_values('Category_Code')
conf_matrix = confusion_matrix(labels_test, mnbc_pred)
plt.figure(figsize=(12.8,6))
sns.heatmap(conf_matrix,
            annot=True,
            xticklabels=aux_df['Category'].values,
            yticklabels=aux_df['Category'].values,
            cmap="Blues")
plt.ylabel('Predicted')
plt.xlabel('Actual')
plt.title('Confusion matrix')
plt.show()
```



In [139]:

```
d = {
     'Model': 'Multinomial Naïve Bayes',
     'Training Set Accuracy': accuracy_score(labels_train, mnbc.predict(features_train)),
     'Test Set Accuracy': accuracy_score(labels_test, mnbc_pred)
}

df_models_mnbc = pd.DataFrame(d, index=[0])
df_models_mnbc
```

Out[139]:

	Model	Training Set Accuracy	Test Set Accuracy
0	Multinomial Naïve Bayes	0.953993	0.934132