News Article Classification

August 18, 2024

1 Abstract

- In this notebook, I tried to classify news articles into five categories using various machine learning methods starting from unsupervised learning, to supervised learning, and finally the multilayer perceptron. I joined a kaggle competition to assess the test dataset results. I used the tfidf word processing algorithm to create the feature matrix.
- I optimized both the word processing method and the machine learning hyperparameters.
- From the word processing optimizations, I found that using word-grams upto pentagrams, from unigrams, and a minimum document frequency for a token, between 6 to 8, gives us the best results. Along with this, using logarithm of the term frequency gives a little better accuracy too.
- Among various machine learning methods, I found that Support Vector Classifier and Multi-layer Layer Perceptron perform better than others and obtained a test accuracy of 0.99, and, 0.98, respectively.
- I also found that unsupervised learning showed a test accuracy of upto 0.97 which can be a useful result, given the fact that labels are not used.
- Among the categories, sport was easily distinguished by the classifiers, while business, tech and, entertainment, mixed a little more in the predictions.

2 Exploratory Data Analysis

In this section, let us explore the dataset.

I will first import the necessary libraries and load the dataset.

[2]:
```
import keras
import sklearn

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, rand_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from ast import literal_eval
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.cluster import AgglomerativeClustering, KMeans
import itertools
```

```python
from scipy.sparse import csr_matrix
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from IPython.display import Image
from wordcloud import WordCloud,STOPWORDS
import inspect
from sklearn.metrics import␣
 ↪ConfusionMatrixDisplay,confusion_matrix,accuracy_score
from scipy.interpolate import griddata
import pickle
import os
import numpy as np
import time
import torch
from sklearn.svm import SVR
from IPython.display import Latex
from sklearn.gaussian_process import GaussianProcessRegressor
import keras_tuner as kt
from scikeras.wrappers import KerasRegressor
from IPython.display import Image
from skopt.space import Real, Categorical, Integer
from statsmodels.api import OLS
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.multioutput import MultiOutputRegressor
from sklearn.tree import DecisionTreeRegressor,export_graphviz
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
from sklearn.decomposition import NMF
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import KernelPCA
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import pandas as pd
from utils.colors import bcolors
from utils.pca import pca
from utils.grid_search import best_params
import utils.process_data as pr
import keras
from sklearn.pipeline import Pipeline
import inspect
from skopt import BayesSearchCV
from bayes_opt import BayesianOptimization
from bayes_opt.logger import JSONLogger
from bayes_opt.event import Events
from bayes_opt.util import load_logs
```

```
pr.set_plot_font_size()#set font size for plots
```

[3]:
```
path = '~/learn-ai-bbc'
train = pd.read_csv(path+'/BBC News Train.csv')
test = pd.read_csv(path+'/BBC News Test.csv')
```

[4]:
```
train
```

[4]:
```
      ArticleId                                               Text  \
0          1833  worldcom ex-boss launches defence lawyers defe…
1           154  german business confidence slides german busin…
2          1101  bbc poll indicates economic gloom citizens in …
3          1976  lifestyle  governs mobile choice  faster  bett…
4           917  enron bosses in $168m payout eighteen former e…
…           …                                                …
1485        857  double eviction from big brother model caprice…
1486        325  dj double act revamp chart show dj duo jk and …
1487       1590  weak dollar hits reuters revenues at media gro…
1488       1587  apple ipod family expands market apple has exp…
1489        538  santy worm makes unwelcome visit thousands of …

           Category
0          business
1          business
2          business
3              tech
4          business
…               …
1485  entertainment
1486  entertainment
1487       business
1488           tech
1489           tech

[1490 rows x 3 columns]
```

Some test examples which seem to be sport articles:

[5]:
```
print(f'{bcolors.BOLD}{test.Text[:5][0]}')
```

```
qpr keeper day heads for preston queens park rangers keeper chris day is set
to join preston on a month s loan.  day has been displaced by the arrival of
simon royce  who is in his second month on loan from charlton. qpr have also
signed italian generoso rossi. r s manager ian holloway said:  some might say it
s a risk as he can t be recalled during that month and simon royce can now be
recalled by charlton.  but i have other irons in the fire. i have had a  yes
from a couple of others should i need them.   day s rangers contract expires in
the summer. meanwhile  holloway is hoping to complete the signing of
middlesbrough defender andy davies - either permanently or again on loan -
before saturday s match at ipswich. davies impressed during a recent loan spell
at loftus road. holloway is also chasing bristol city midfielder tom doherty.
```

[6]:
```python
print(f'{bcolors.BOLD}{test.Text[:5][2]}')
```

```
d arcy injury adds to ireland woe gordon d arcy has been ruled out of the
ireland team for saturday s six nations clash against scotland in murrayfield.
like skipper brian o driscoll  d arcy failed to recover from a hamstring injury.
the side will now be led by munster lock paul o connell. shane horgan switches
from wing to centre where he will be joined by ulster s kevin maggs. girvan
dempsey comes into the team to take the right wing spot while gavin duffy is
called up to the replacements.   we gave gordon a chance but it didn t work out
said ireland coach eddie o sullivan.   in terms of the risk element  it was a
sensible precaution. he should be fine for the next game but we do not want to
tempt fate.    maggs  who will win his 67th cap  was the obvious replacement at
centre while shane horgan was always likely to be moved from the wing. the only
other change to the ireland side from last weekend s win in rome sees wasps
flanker johnny o connor replacing denis leamy. o connor will be winning his
third cap after making his debut in the victory over south africa last november.
: murphy  dempsey  horgan  maggs  hickie  o gara  stringer  corrigan  byrne
hayes  o kelly  o connell  capt  s easterby  o connor  foley.  : sheahan  horan
o callaghan  miller  g easterby  humphreys  duffy.
```

Now, let us check number of categories:

[7]:
```python
train['Category'].nunique()
```

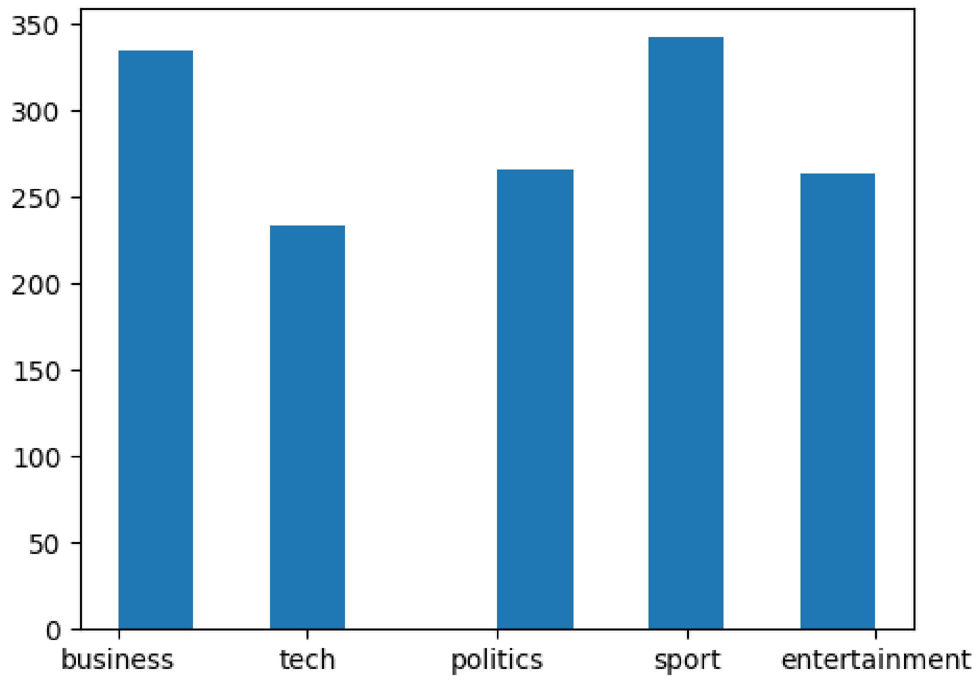
```
[7]: 5
```

Let us see the total duplicate titles and examine the deduplicated dataset through a histogram:

```
[5]: total_duplicate_titles = sum(train["Text"].duplicated())
     train = train[~train["Text"].duplicated()]
     train.describe().loc['count']
```

```
[5]: ArticleId    1440.0
     Name: count, dtype: float64
```

```
[9]: plt.hist(train['Category'])
     plt.rc('figure',figsize=[9,6])
     plt.show()
```



Let us see how the word count of the articles look like.

```
[9]: train_test=pd.concat([train.Text,test.Text])
     train_test.apply(lambda x: len(x.split())).describe()
```

```
[9]: count    2175.000000
     mean      390.551264
     std       243.143795
     min        90.000000
     25%       250.000000
     50%       337.000000
```

```
75%        479.000000
max       4492.000000
Name: Text, dtype: float64
```

Let us count the number of articles in each category;

```
[11]: f=open('traintxt.csv','w')
      f.write(str(((train.Category.to_string()))))
      cat_count=!cut -f2 -w traintxt.csv | sort| uniq -c
      cat_count
```

```
[11]: ['  335 business',
       '  263 entertainment',
       '  266 politics',
       '  342 sport',
       '  234 tech']
```

Word processing In order to categorize articles, let's use **tf-idf vectorization** text processing method. **tfidf** uses two metrics to rank importance of a given word the text corpus (collection of text objects): - **tf**, or, **term frequency**, which is the word count in the text object. It can be normalized via division of the word count value with the number of total words in the text object. - **idf**, or, **inverse document frequency**, which measures word importance which is calculated based on uniqueness of the word in the dataset.

Thus, this method requires creating a vocabulary, including every word in the dataset. Then, it calculates tf, and, idf for every text object. It assigns tf values for each word present in the text object, or article, in this case, and uses the idf values, which are fixed, for each term in the vocabulary, if the word is present in the article, or 0, if not present. The tfidf score is calculated by multiplying tf and idf values. A higher number indicates higher word importance. Thus, similar documents will have similar high ranking words. The `sklearn` library implements tfidf vectorization using the `TfidfVectorizer` class. The hyperparameters used are: - `max_df : float or int, default=1.0`; a float input means, for a word, appearing in more than (max_df*100)% of the documents will get it removed from the vocabulary. - `min_df : float or int, default=1`; when an integer, implies that words appearing in less than min_df documents will be disregarded. - `stop_words` is 'english', and this setting filters common english words from the vocabulary. - `ngram_range` specifies the range of the n-word grams that should be used as tokens. For Ex., (1,3) includes unigrams, bigrams, and, trigrams. - `sublinear_tf` cconverts term frequency, or, tf to 1+log(tf).

```
[327]: tfidf_vectorizer = TfidfVectorizer(
           max_df=1.0,
           min_df=1,
            stop_words = 'english',
           ngram_range=(1,1),
           sublinear_tf=False,)
       tfidf_train_test=tfidf_vectorizer.fit_transform(train_test)
```

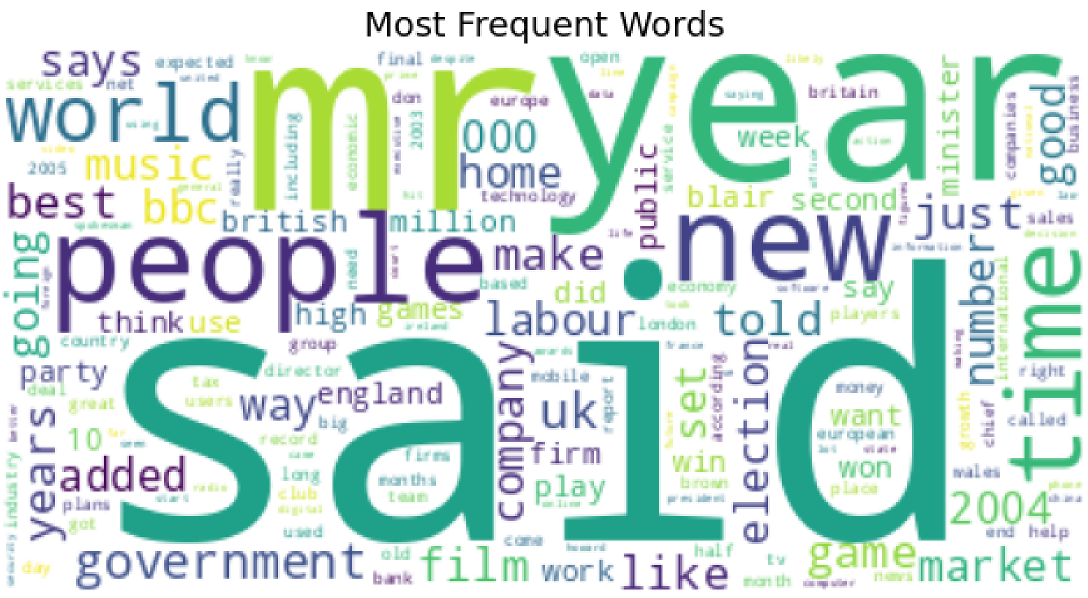
```
[328]: print('samples\tfeatures:\t',tfidf_train_test.shape)
```

```
samples features:        (2175, 29126)
```

WordCloud is a tool to see the frequencies of words. Let's try this, skipping stopwords, i.e., common english words like 'him', 'she', etc.

[339]:
```
cv=CountVectorizer(stop_words='english')
cv.fit_transform(train_test)
vocab_features_counts=dict(zip(cv.get_feature_names_out(),
                np.sum(cv.fit_transform(train_test).toarray(),axis=0)))
```

[344]:
```
wc = WordCloud(background_color = "white",max_words =300,).
  ↪generate_from_frequencies(vocab_features_counts)
plt.imshow(wc)
plt.axis("off")
plt.title('300 Most Frequent Words')
plt.show()
```



3 Unsupervised learning

I will use three unsupervised learning methods to find the best predicting model. 1. KMeans Clustering 2. Non-Negative Matrix Factorization 3. Hierarchial Clustering

We can use both the test and train data in concatenation, as we are not fitting the data using labels in unsupervised learning.

3.0.1 KMeans

Now, I will use the KMeans Algorithm on the data. We change the `random_state`, since the KMeans method gives different results every time this parameter changes. The plot below shows the variation in accuracy when the `random_state` is changed from 0 to 99.

A `permute_label` function is defined below. This function checks accuracies for all permutations of the category names to get the right clusters, as the model returns clusters as integers and we map it to the original category names to calculate accuracy. Thus, we permute the order of the category names to find the best mapping order.

[36]:
```
## Fubction to check permutations
pd.set_option('future.no_silent_downcasting', True)
def permute_label(yt,yp):

    k = list(train['Category'].unique())
    n = list(range(5))
    acc_cop = 0
    for i in itertools.permutations(k):
        d = dict(zip(n,i))
        acc = sklearn.metrics.accuracy_score(yt,pd.Series(yp).replace(d))
        if acc_cop< acc:
            acc_cop = acc
            final_label_order = d
    #print(f"accuracy : {acc_cop}")
    #print(f"final_labels : {final_label_order}")
    return acc_cop,final_label_order
```

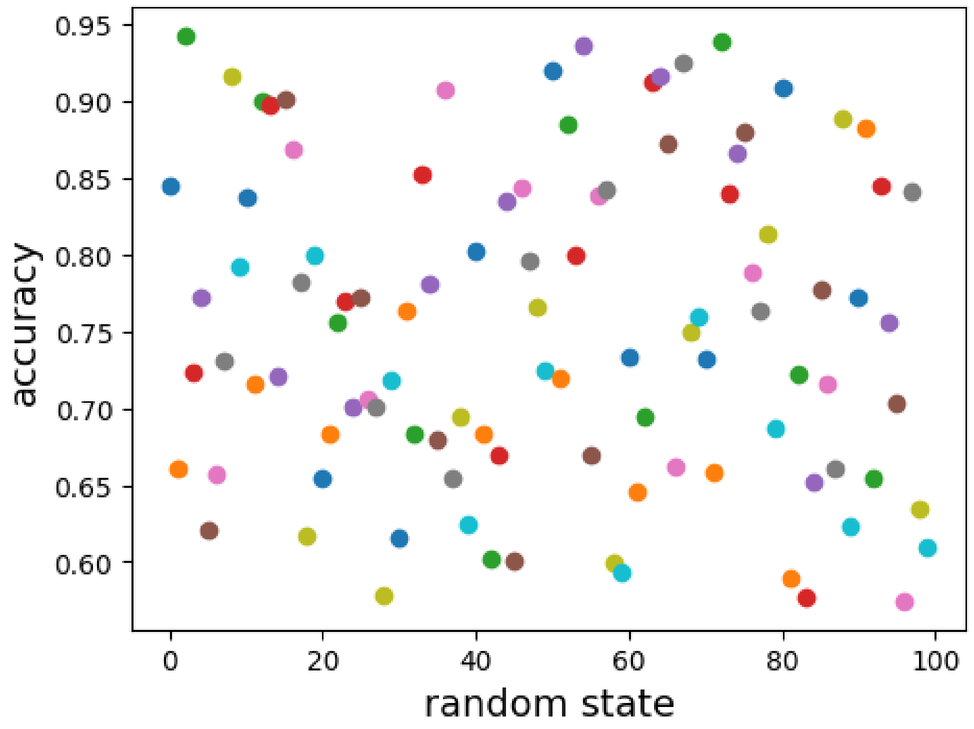
[37]:
```
#tf-idf features for KMeans
tfidf_vectorizer = TfidfVectorizer(
    max_df=1.,
    min_df=1,
    stop_words = 'english',
    analyzer='word',
    max_features=None,
    ngram_range=(1,2),
    binary=False,
    norm='l2',
    use_idf=True,
    smooth_idf=True,
    sublinear_tf=True,
)
tfidf_train_test = tfidf_vectorizer.fit_transform(train_test)
print('samples\tfeatures:\t',tfidf_train_test.shape)
```

```
samples features:        (2175, 349609)
```

[38]:
```
##MODEL KMEANS
l_kmeans = []
```

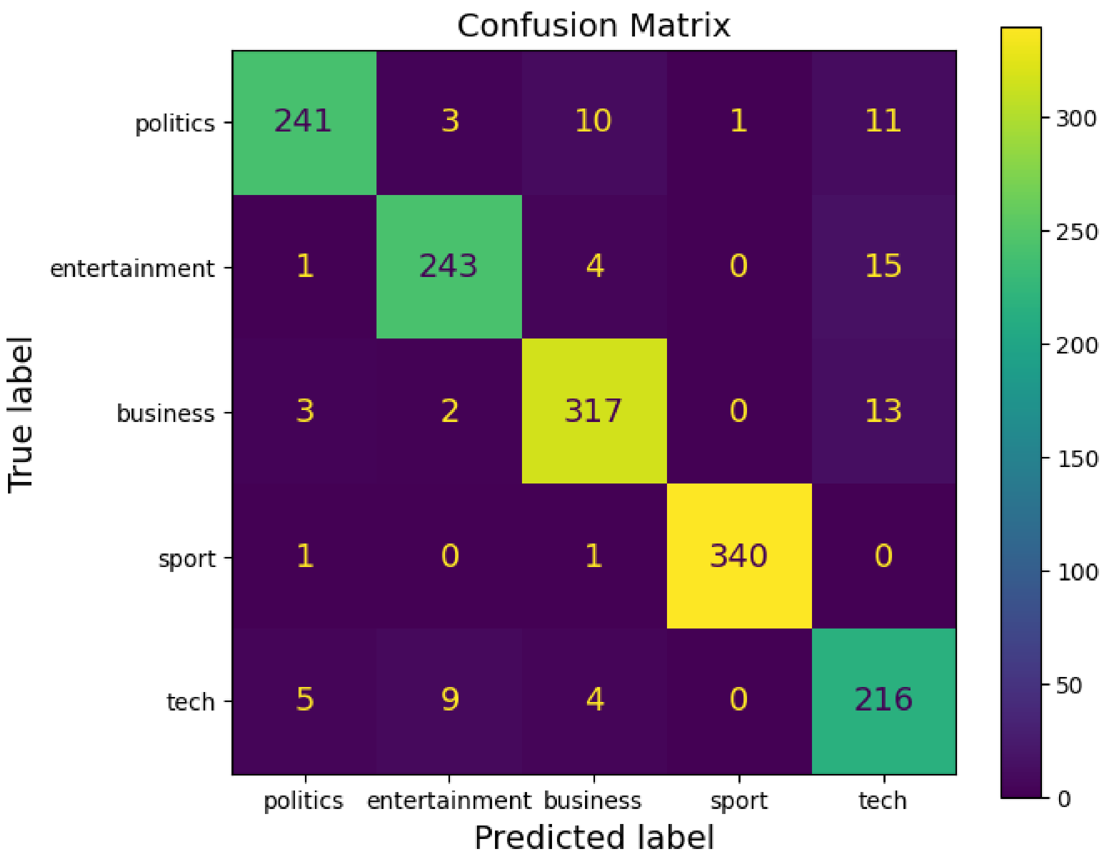
```
for i in range(100):
    model = KMeans(n_clusters = 5,random_state=i,max_iter=300)
    yp = model.fit_predict(tfidf_train_test)
    acc, final_labels = permute_label(train.Category,yp[:1440])
    #acc=sklearn.metrics.accuracy_score(tfidf_train_test.Category)
    plt.scatter(i,acc)
    l_kmeans.append([acc,final_labels,model,i])
plt.xlabel('random state')
plt.ylabel('accuracy')
pr.set_plot_font_size()
acc,label_order,model,best_random_state = sorted(l_kmeans, key = lambda x:
 ↪x[0])[-1]
print(f"accuracy",acc)
print(f"best random state",best_random_state)
plt.show()
```

```
accuracy 0.9423611111111111
best random state 2
```



[384]:
```
cat_names=[label_order[i] for i in range(5)]
train_category_int=train.Category.replace(to_replace=cat_names,
                                          value= [0,1,2,3,4]).astype(int)
```

[386]:
```
ConfusionMatrixDisplay(confusion_matrix(train_category_int,
                                                      KMeans(
                                                        ␣
  ↪n_clusters=5,random_state=best_random_state).fit_predict(tfidf_train_test)[:
  ↪1440]
                                              ,),display_labels=cat_names).plot()
plt.title('Confusion Matrix')
plt.rc('figure',figsize=[8,6])
plt.show()
```



Since, we have 349609 features, the KMeans algorithm can suffer from the curse of dimensionality, as it calculates the l2 norm which might give large values due to calculating distances in dimensions that do not matter. So, next, I fit a pipeline using KernelPCA to reduce the dimensions using Principal Component Analysis.

[387]:
```
pipe_kmeans=Pipeline([('PCA',KernelPCA(n_components=3000)),('kmeans',KMeans(n_clusters=5,rando
def fit_score_pipeline(pipeline=None):
    f1=f'acc = {sklearn.metrics.accuracy_score(train_category_int
                                              ,pipeline.fit(tfidf_train_test).
  ↪predict(
                                                tfidf_train_test[:1440]))}'
    return f1
fit_score_pipeline(pipeline=pipe_kmeans)
```

[387]:
```
'acc = 0.9423611111111111'
```

Let us time `KMeans` and `pipe_kmeans` functions.

[389]:
```
%%timeit
KMeans(n_clusters=5,random_state=best_random_state).
 ↪fit_predict(tfidf_train_test)
```

```
189 ms ± 17.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[388]:
```
%%timeit -r1
pipe_kmeans.fit_predict(tfidf_train_test)
```

```
2.63 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

PCA does not improve accuracy or fitting time, however, it does indicate that leaving features might get results similar to when we use all features. We can use the `max_features` parameter in the `tfidf_vectorizer` method to control the number of features we should use, by their rank, which is based on their tfidf score.

Next, let us optimize the `TfidfVectorizer` hyperparameters by using it in a pipeline with `KMeans`, and then using `BayesSearchCV` to do a bayesian search for best hyperparameters for the `TfidfVectorizer`. It will maximize the 'score' which is the negative of the inertia or the within-cluster sum of squares. This method takes three acquisition functions and choses the one with the best chance of improvement.

Let us now try Bayesian hyperparameter optimization for the tfidf word processing algorithm using the Upper Confidence Bound acquisition function, which finds the next parameter based on the confidence interval, given the confidence value is lesser than the upper bound. I will maximize the accuracy score.

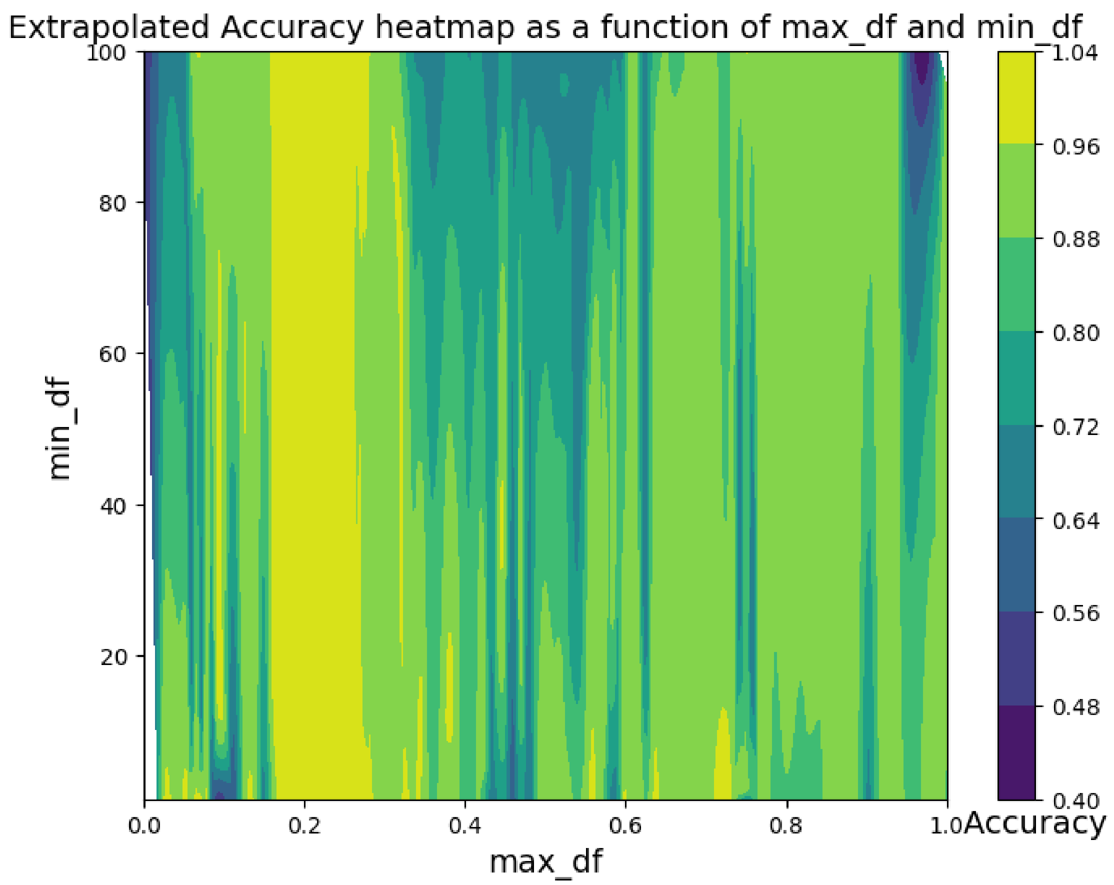
[39]:
```
l = []

pbounds = {'max_df' :(0.1,1),'min_df' : (1,100),}
## Find best parameters
def bb_function(max_df,min_df):
    tfidf_vectorizer = TfidfVectorizer(
    max_df=int(max_df*100)/100.,
    min_df=int(min_df),
    stop_words='english',
        ngram_range=(1,5),
        sublinear_tf=True
    )
    tfidf_train_test=tfidf_vectorizer.fit_transform(train_test)
    model=KMeans(n_clusters=5,random_state=best_random_state)
    yp=model.fit_predict(tfidf_train_test)
    acc, final_labels = permute_label(train.Category,yp[:1440])
    return acc
optimizer = BayesianOptimization(
    f = bb_function,pbounds = pbounds,random_state = 1)
logger = JSONLogger(path="/Users/shaurya/logs10.log.json")
optimizer.subscribe(Events.OPTIMIZATION_STEP, logger)
n_iter=200
optimizer.maximize(
init_points=int(n_iter*0.1),
n_iter=n_iter,
)
```

[40]:
```
load_logs(optimizer, logs=["/Users/shaurya/logs10.log.json"])
pd.DataFrame(optimizer.res).sort_values(by='target',ascending=False)[:5]
```

[40]:
```
       target                                                 params
68   0.968056       {'max_df': 0.1, 'min_df': 7.531736469942007}
171  0.965972  {'max_df': 0.6646005120698416, 'min_df': 17.25…
155  0.965972  {'max_df': 0.7447407356869815, 'min_df': 19.41…
173  0.965972       {'max_df': 1.0, 'min_df': 18.24530733520916}
34   0.965972      {'max_df': 1.0, 'min_df': 18.809789687239196}
```

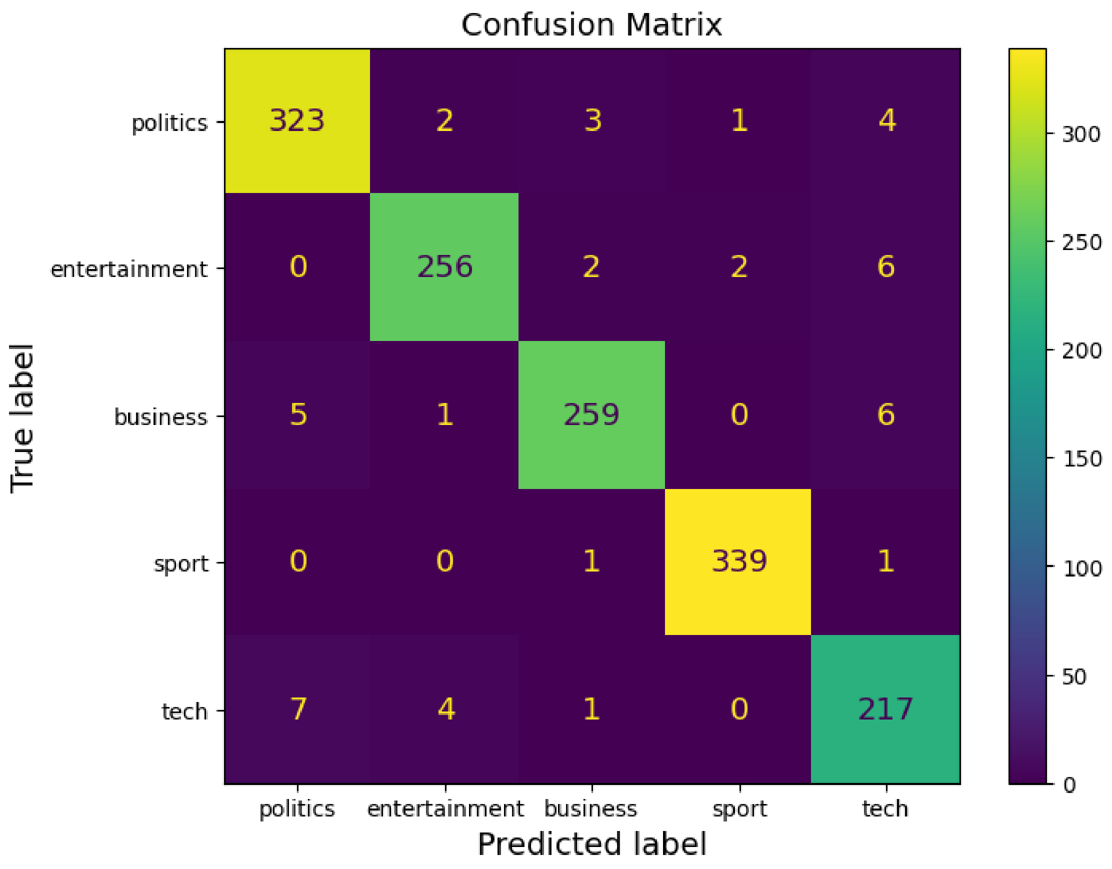
[58]:
```
max_df,min_df=optimizer.max['params']['max_df'],int(optimizer.
 ↪max['params']['min_df'])
l=[]
a = ['target','params']
b = [[item[i]for item in optimizer.res]for i in a]
l = np.array([[k for j,k in i.items()] for i in b[1]])
```

[62]:
```
x,y = np.linspace(0.1,1,1000),np.linspace(1,100,1000)
x,y = np.meshgrid(x,y)
g = griddata(points = (l[:,0],l[:,1]), values = b[0], xi = (x,y)
             ,method = 'cubic')
plt.contourf(np.linspace(0,1,1000),np.linspace(1,100,1000),g.T)
plt.xlabel('max_df')
plt.ylabel('min_df')
plt.title('Extrapolated Accuracy heatmap as a function of max_df and min_df')
cb = plt.colorbar()
cb.ax.set_xlabel('    Accuracy')
plt.rc('figure',figsize=[8,6])
plt.show()
```



[458]:
```
tfidf_vectorizer = TfidfVectorizer(
max_df=max_df,
min_df=min_df,
stop_words='english',
    ngram_range=(1,5),
    sublinear_tf=True
)
tfidf_train_test=tfidf_vectorizer.fit_transform(train_test)
model=KMeans(n_clusters=5,random_state=best_random_state)
yp=model.fit_predict(tfidf_train_test)
acc, label_order = permute_label(train.Category,yp[:1440])
yp_labeled=pd.Series(yp).replace(label_order)
ConfusionMatrixDisplay(confusion_matrix(yp_labeled[:1440],train.
 ↪Category),display_labels=cat_names).plot()
plt.title('Confusion Matrix')
plt.rc('figure',figsize=[8,6])
plt.show()
test['Category'] = pd.Series(yp[1440:]).replace(label_order)
test[['ArticleId','Category']].to_csv('Solution_kMeans.csv',index=False)
```

```
#!kaggle competitions submit -c learn-ai-bbc -f Solution_kMeans.csv -m "Message"
```



If I set `sublinear_tf` to True, it helps on the accuracy. With it the accuracy is 0.968705. Below is a result without it.

[63]:
```
tfidf_vectorizer = TfidfVectorizer(
max_df=max_df,
min_df=min_df,
stop_words='english',
    ngram_range=(1,5),
    sublinear_tf=False
)
tfidf_train_test=tfidf_vectorizer.fit_transform(train_test)
model=KMeans(n_clusters=5,random_state=best_random_state)
yp=model.fit_predict(tfidf_train_test)
acc, label_order = permute_label(train.Category,yp[:1440])
print(f'accuracy\t{acc}')
```

```
accuracy        0.9354166666666667
```

TEST SET RESULT

Solution_kMeans.csv
Complete (after deadline) · 2m ago · Message
0.96462 0.96462

```
[21]: table=[]
      table.append(['Method',"Train Accuracy", 'Test Accuracy'])
```

```
[22]: table.append(['KMeans',0.96,0.96])
```

3.0.2 Matrix Factorization

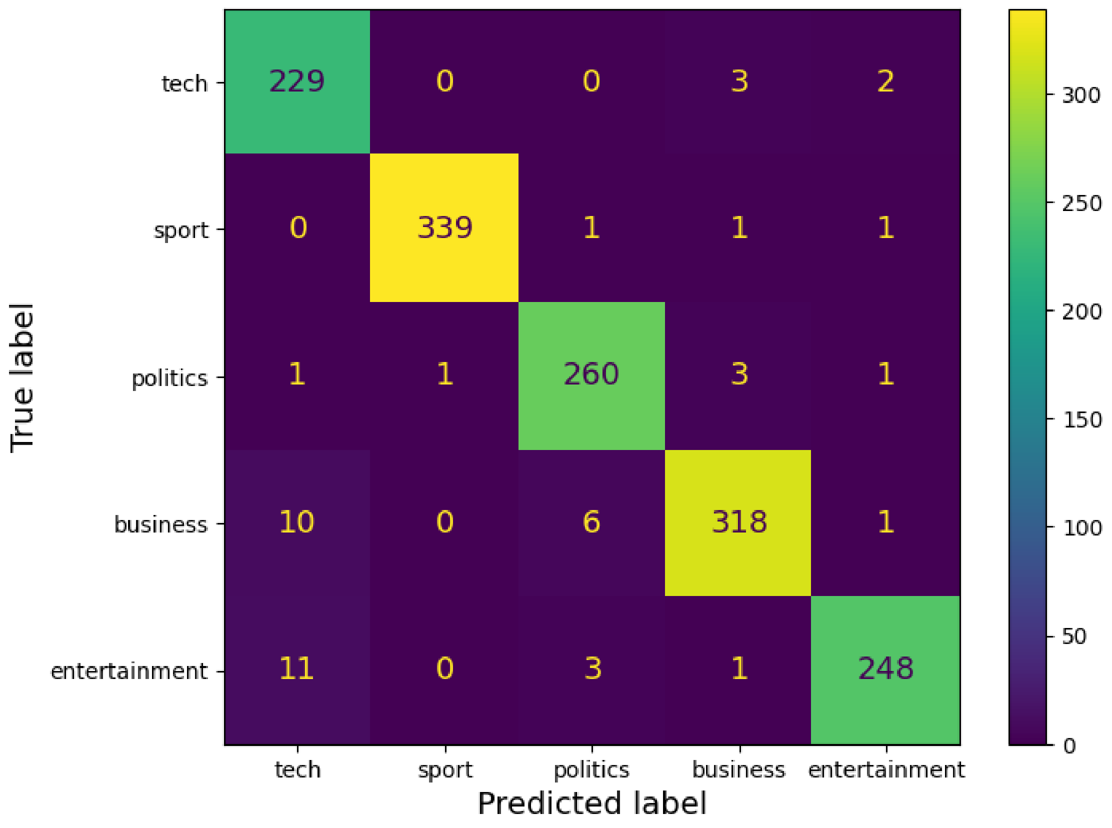
We now use the Non-Negative Matrix Factorization technique which factorizes a matrix X ~ WH. X (n×f) is broken into *d* hidden or sub-features. The reduced matrix is W (n×d), having d new features, while H (d×f) contains the weights of the original features in the new representation. We will use non-negative matrix factorization to get five components corresponding to the five categories in the dataset. These components will be classified based on their weights for each document, i.e. the weight with the highest value for each document will be the category of the document. Both train and test data will be utilized for forming the matrix X. The correct label order, i.e., the category order which matches the ground truth, will be found with the `label_permute_2` function.

```
[453]: tfidf_vectorizer = TfidfVectorizer(
           max_df=max_df,
           min_df=min_df,
            stop_words = 'english',
           ngram_range=(1,5),
           sublinear_tf=True,)
       tfidf_train_test=tfidf_vectorizer.fit_transform(train_test)
```

```
[463]: def label_permute_2(yp,Categ):
           l =[]
           for i in itertools.permutations(train["Category"].unique()):
               lookup = keras.layers.StringLookup(vocabulary = i,output_mode="one_hot",
                                                  num_oov_indices = 0,)
               label_mh = np.array([lookup(label).cpu().numpy() for label in Categ]).
        ↪squeeze()
               l.append([sklearn.metrics.accuracy_score(yp,label_mh),i])
           return sorted(l,key = lambda x : x[0])[-1]

       def nmf_opt(Train=False,input_matrix=tfidf_train_test):
           """
           Parameters:
           Train: if True, then finds the correct category order.
           input_matrix of shape (n_samples,n_features)
           Returns:
           accuracy and correct label order if Train is True, else, predictions W.
           """
           X = sklearn.decomposition.NMF(n_components=5,solver='mu',init = 'nndsvda',
                                         beta_loss="kullback-leibler",
                                                  alpha_W=0.00005,alpha_H=0.00005,
                                                  l1_ratio=1.0,random_state =4,
                                                  max_iter =500)
           W = X.fit_transform(input_matrix)
           W = np.array([W[i]==max(W[i]) for i in range(input_matrix.shape[0])]).
        ↪astype(int) ##Since every word vector is now
           #divided into its weights for each catergory
           #we decide to take the maximum strength for each
           #sample and set that to 1 and rest to zero. Hence the strongest weight
           #becomes the category.
           if Train :
               out = label_permute_2(W[:1440],train.Category.values)
           else:
               out = W
           return out
```

[455]:
```
acc, label_order = nmf_opt(Train = True)
print(f"acc\t correct label order :{acc,label_order}")
result = nmf_opt()
yp = [np.take(label_order,np.argwhere(result[i] == 1.0)[..., 0])[0] for i in␣
 ↪range(len(result))]
print(f"acc ; {accuracy_score(yp[:1440],list(train.Category))}")
plt.rc('figure',figsize=[8,6])
ConfusionMatrixDisplay(confusion_matrix(list(train.Category),yp[:1440],labels =␣
 ↪label_order ),display_labels = label_order).plot()
plt.show()
```

```
acc      correct label order :(0.9680555555555556, ('tech', 'sport', 'politics',
'business', 'entertainment'))
acc ; 0.9680555555555556
```



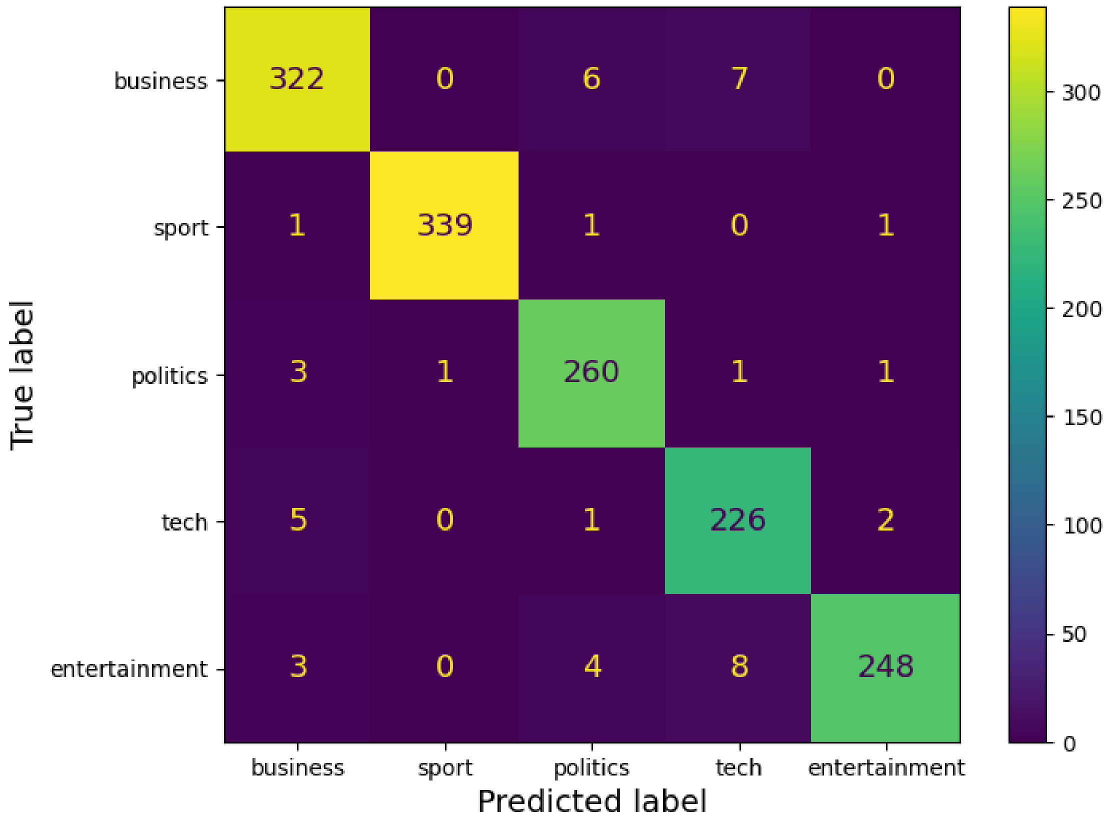
From the confusion matrix, I see that the predictions are poor on business and entertainment articles in the sense that many of them are labelled as tech. Thus I will heurestically chose a larger `max_df` value and increase `min_df` by one, and see if all confusion matrix elements are single numbers.

[469]:
```
tfidf_vectorizer = TfidfVectorizer(
    max_df=0.5,
    min_df=8,
     stop_words = 'english',
    ngram_range=(1,5),
    sublinear_tf=True,)
tfidf_train_test=tfidf_vectorizer.fit_transform(train_test)
```

[474]:
```
acc, label_order = nmf_opt(Train = True,input_matrix=tfidf_train_test)
print(f"acc\t correct label order :{acc,label_order}")
result = nmf_opt(input_matrix=tfidf_train_test)
yp = [np.take(label_order,np.argwhere(result[i] == 1.0)[..., 0])[0] for i in␣
 ↪range(len(result))]
plt.rc('figure',figsize=[8,6])
ConfusionMatrixDisplay(confusion_matrix(list(train.Category),yp[:1440],labels =␣
 ↪label_order ),display_labels = label_order).plot()
```

```
plt.show()
```

```
acc      correct label order :(0.96875, ('business', 'sport', 'politics',
'tech', 'entertainment'))
```



```
[475]: test['Category'] = yp[1440:]
       test[['ArticleId','Category']].to_csv('Solution_NMF.csv',index=False)
       !kaggle competitions submit -c learn-ai-bbc -f Solution_NMF.csv -m "Message"
```

```
100%|                         | 9.20k/9.20k [00:00<00:00, 9.49kB/s]
Successfully submitted to BBC News Classification
```

The result of the best performing model is given below.

TEST SET RESULT

Solution_NMF.csv
Complete (after deadline) · 9d ago · Message 0.96734 0.96734

```
[23]: table.append(['NMF',0.97,0.97])
```

```
[24]: pd.DataFrame(table[1:],columns=table[0][:]).set_index('Method')
```

```
[24]:         Train Accuracy  Test Accuracy
      Method
      KMeans            0.96           0.96
      NMF               0.97           0.97
```

To improve on this score, let us use another feature extraction method such as Word2Vec which uses word embeddings to convert text into word vectors. Here, the values of word vectors for the text object depend on the adjunctivity of the words to their neighbouring words. From EDA we saw that max length of an article is 3345. Thus, we initialize a word embedding as:

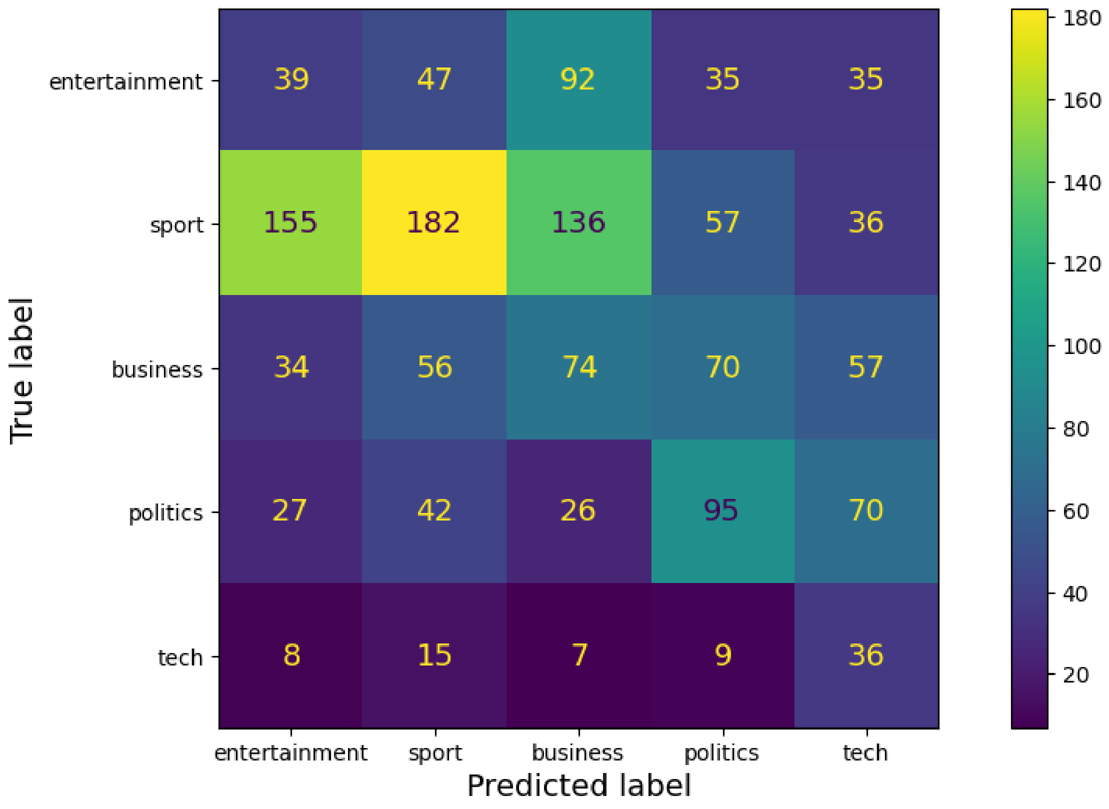
```
[484]: word2vec = keras.layers.TextVectorization(
           max_tokens=4492,
           standardize='lower_and_strip_punctuation',
           split='whitespace',
           output_mode='int',
           output_sequence_length=None,
           pad_to_max_tokens=True,
           encoding='utf-8',
       )
```

```
[485]: def remove_stopwords_word2vec_only(st):
           st = st.lower().split()
           m = [i for i in st if i not in STOPWORDS]
           return str(m)
       train_w2v = train_test.apply(lambda x: remove_stopwords_word2vec_only(x))
```

```
[ ]: word2vec_df = word2vec(train_w2v)
     acc, label_order =nmf_opt(tfidf_train_test=word2vec_df,Train=True)
     result=nmf_opt(tfidf_train_test=word2vec_df)
     yp = [np.take(label_order,np.argwhere(result[i] == 1.0)[..., 0])[0] for i in␣
      ↪range(len(result))]
```

```
[561]: ConfusionMatrixDisplay(confusion_matrix(yp[:144],list(train.Category),labels =␣
        ↪label_order ),display_labels = label_order).plot()
```

```
[561]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x36ca2d3a0>
```



[818]:
```
print(F"Train set -- accuracy, {acc}")
print(F"Label order -- :{label_order}")
```

```
Train set -- accuracy, 0.29583333333333334
Label order -- :('business', 'sport', 'politics', 'tech', 'entertainment')
```

3.0.3 Hierarchial Clustering

We will now use `AgglomerativeClustering` method from `sklearn` library to perform Hierarchial Clustering which minimizes the distance within-clusters. This technique serves important when the number of clusters is unknown and the KMeans method cannot be used.

[34]:
```
# Use tf-idf features.
tfidf_vectorizer = TfidfVectorizer(
    max_df=0.8,
    min_df=8,
    max_features=None,
     stop_words = 'english',
    ngram_range=(1,5),
    sublinear_tf=True
)
tfidf_train_test = tfidf_vectorizer.fit_transform(train_test)
```

[38]:
```
model = AgglomerativeClustering(n_clusters = 5)#, linkage = linkage, metric =␣
  ↪affinity,  )
yp = model.fit(tfidf_train_test.toarray()).labels_
ac, label_order = permute_label(train.Category,yp[:1440])
print(f'acc\t{ac}')
```

```
acc     0.9243055555555556
```

[39]:
```
res=pd.Series(yp[1440:]).replace(label_order)
```

[40]:
```
test['Category'] = res
test[['ArticleId','Category']].to_csv('Solution_Agglomerative.csv',index=False)
!kaggle competitions submit -c learn-ai-bbc -f Solution_Agglomerative.csv -m␣
  ↪"Message"
```

```
100%|                    | 9.26k/9.26k [00:01<00:00, 9.38kB/s]
Successfully submitted to BBC News Classification
```

Solution_Agglomerative.csv
Complete (after deadline) · 10s ago · Message
0.93469 0.93469

[41]:
```
table.append(['Hierarchial Clustering',0.92,0.93])
```

[43]:
```
pd.DataFrame(table[1:],columns=table[0][:]).set_index('Method').
  ↪sort_values(by='Test Accuracy',ascending=False)
```

[43]:
```
                        Train Accuracy  Test Accuracy
Method
NMF                               0.97           0.97
KMeans                            0.96           0.96
Hierarchial Clustering            0.92           0.93
```

4 Supervised Learning

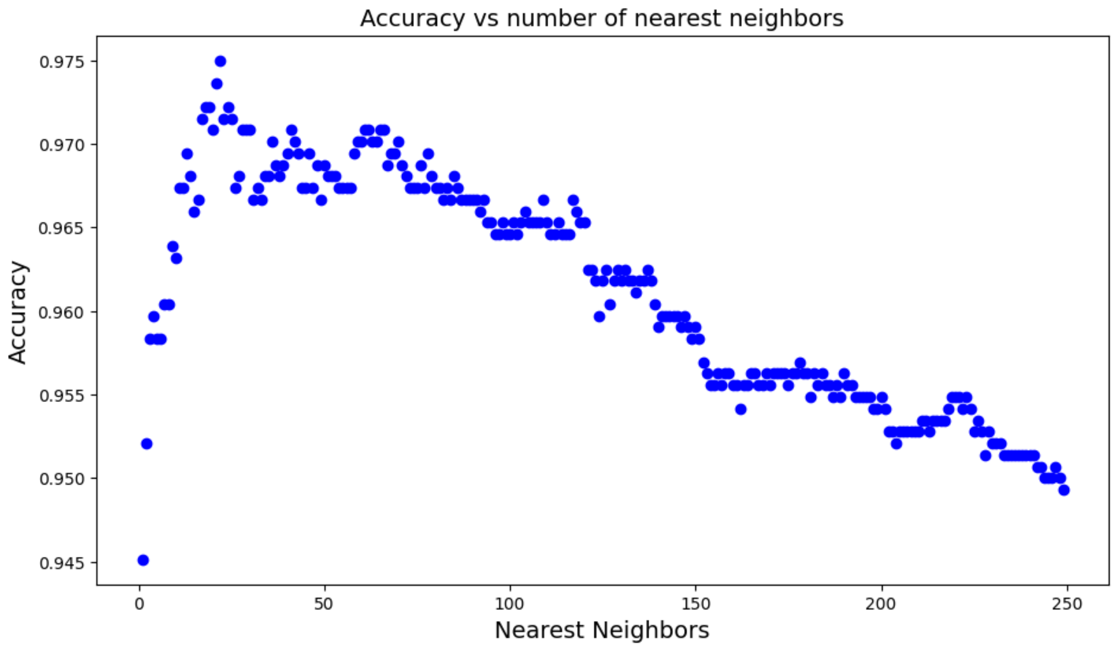
Now, let us use supervised learning methods and see how well they work compared to the unsupervised methods.

4.0.1 KNN

[45]:
```
##use KNN on the data
tfidf_vectorizer = TfidfVectorizer(
    max_df=0.5,
    min_df=8,
    stop_words="english",
    max_features = None,
    sublinear_tf=True,
    ngram_range=(1,5)
)
```

```
tfidf = tfidf_vectorizer.fit_transform(train['Text'])
tfidf_train_test = tfidf_vectorizer.fit_transform(train_test)
##Check best value of number of nearest neighbors
l_knn=[]
m_knn=[]
for i in range(1,250):
    l_knn.append([i,np.mean(cross_val_score(sklearn.neighbors.
  ↪KNeighborsClassifier(
        n_neighbors = i),tfidf,train.Category,scoring='accuracy'))])

print("neighbors\tmax_acc" , (sorted(l_knn,key =lambda x: x[1]))[-1])
nn = (sorted(l_knn,key =lambda x: x[1]))[-1][0]
##Train splitting accuracy check
plt.plot(*np.array(l_knn).T,'bo')
plt.xlabel('Nearest Neighbors')
plt.ylabel('Accuracy')
plt.title('Accuracy vs number of nearest neighbors')
y_pred = (sklearn.neighbors.KNeighborsClassifier(
        n_neighbors = nn).fit(tfidf,train.Category).predict(tfidf))
```

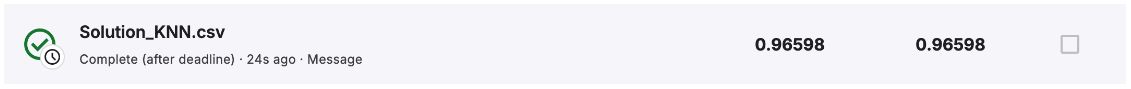
```
neighbors       max_acc [22, 0.975]
```



[50]:
```
test['Category']= sklearn.neighbors.KNeighborsClassifier(n_neighbors = nn).
  ↪fit(tfidf_train_test[:1440],train.Category).predict(tfidf_train_test[1440:])
test[['ArticleId','Category']].to_csv('Solution_KNN.csv',index=False)
!kaggle competitions submit -c learn-ai-bbc -f Solution_KNN.csv -m "Message"
```

```
100%|                    | 9.18k/9.18k [00:01<00:00, 8.91kB/s]
Successfully submitted to BBC News Classification
```

TEST SET RESULT



```
Solution_KNN.csv                                        0.96598     0.96598
Complete (after deadline) · 24s ago · Message
```

```
[51]: table.append(['KNN',0.97,0.97])
```

4.0.2 Ensemble Methods

```
[53]: rf=sklearn.ensemble.RandomForestClassifier(n_estimators=1000)
      print('Random Forest Classifier cross_val_score:')
      np.mean(cross_val_score(rf,tfidf_train_test[:1440],train.
       ↪Category,scoring='accuracy'))
```

```
Random Forest Classifier cross_val_score:
```

```
[53]: 0.9569444444444445
```

```
[54]: gbc=sklearn.ensemble.GradientBoostingClassifier(n_estimators=100)
      print('Gradient Boosting Classifier cross_val_score:')
      np.mean(cross_val_score(gbc,tfidf_train_test[:1440],train.
       ↪Category,scoring='accuracy'))
```

```
Gradient Boosting Classifier cross_val_score:
```

```
[54]: 0.9527777777777778
```

```
[55]: test['Category']= rf.fit(tfidf_train_test[:1440],train.Category).
       ↪predict(tfidf_train_test[1440:])
      test[['ArticleId','Category']].to_csv('Solution_RFC.csv',index=False)
      !kaggle competitions submit -c learn-ai-bbc -f Solution_RFC.csv -m "Message"
```

```
100%|                    | 9.23k/9.23k [00:01<00:00, 8.41kB/s]
Successfully submitted to BBC News Classification
```

```
[56]: test['Category']= gbc.fit(tfidf_train_test[:1440],train.Category).
       ↪predict(tfidf_train_test[1440:])
      test[['ArticleId','Category']].to_csv('Solution_GBC.csv',index=False)
      !kaggle competitions submit -c learn-ai-bbc -f Solution_GBC.csv -m "Message"
```

```
100%|                    | 9.23k/9.23k [00:01<00:00, 8.44kB/s]
Successfully submitted to BBC News Classification
```

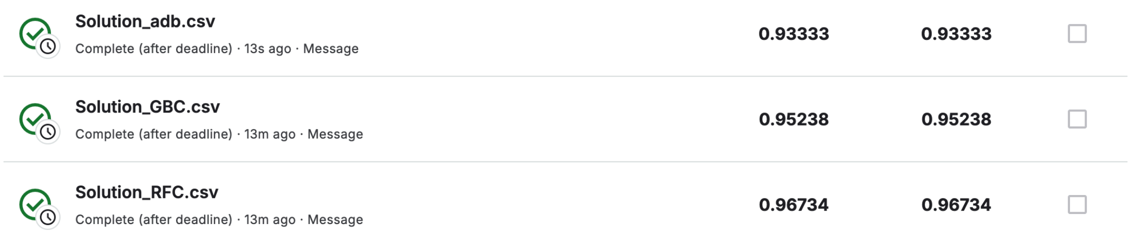
```
[59]: ##Adaboost Classifier
      from sklearn.ensemble import AdaBoostClassifier
      ada_clf=AdaBoostClassifier(DecisionTreeClassifier(random_state=0,max_depth=1,
                                                        max_leaf_nodes=2),
                                 n_estimators=2000,
                                 algorithm="SAMME",learning_rate=0.5)
      print('Adaboost Classifier cross_val_score:')
      np.mean(cross_val_score(ada_clf,tfidf_train_test[:1440],train.
       ↪Category,scoring='accuracy'))
```

```
Adaboost Classifier cross_val_score:
```

[59]:
```
0.8958333333333334
```

[61]:
```
test['Category']= ada_clf.fit(tfidf_train_test[:1440],train.Category).
 ↪predict(tfidf_train_test[1440:])
test[['ArticleId','Category']].to_csv('Solution_adb.csv',index=False)
!kaggle competitions submit -c learn-ai-bbc -f Solution_adb.csv -m "Message"
```

```
100%|                     | 9.31k/9.31k [00:01<00:00, 8.67kB/s]
Successfully submitted to BBC News Classification
```



Solution_adb.csv Complete (after deadline) · 13s ago · Message	0.93333	0.93333	☐
Solution_GBC.csv Complete (after deadline) · 13m ago · Message	0.95238	0.95238	☐
Solution_RFC.csv Complete (after deadline) · 13m ago · Message	0.96734	0.96734	☐

TEST SET RESULT

[57]:
```
table.append(['Random Forest Classifier',0.96,0.97])
table.append(['Gradient Boosting Classifier',0.95,0.95])
table.append(['Adaboost Classifier',0.90,0.93])
```

4.0.3 Support Vector Classifier

I will now use the support vector classifier, and then optimize the hyperparameters using `kernel=rbf`.

[64]:
```
from sklearn.svm import SVC
print('cross validation accuracy:')
np.mean(cross_val_score(SVC(),tfidf_train_test.toarray()[:1440]
                        ,train.Category,scoring='accuracy'))
```

```
cross validation accuracy:
```

[64]:
```
0.9736111111111111
```

[66]:
```
#With standardized inputs
pipe_SVM=Pipeline([('StandardScaler',StandardScaler()),('svm',SVC())])
print('cross validation accuracy:')
np.mean(cross_val_score(pipe_SVM,tfidf_train_test.toarray()[:1440]
                        ,train.Category,scoring='accuracy'))
```

```
cross validation accuracy:
```

```
[66]: 0.8979166666666666
```

```
[67]: for kernel in ['linear','rbf','poly','sigmoid']:
          print('cross validation accuracy:\t ',kernel,    np.
       ↪mean(cross_val_score(SVC(kernel=kernel),tfidf.toarray()
                            ,train.Category,scoring='accuracy')))
```

```
cross validation accuracy:        linear 0.975
cross validation accuracy:        rbf 0.9736111111111111
cross validation accuracy:        poly 0.7201388888888889
cross validation accuracy:        sigmoid 0.9756944444444444
```

```
[68]: grid ={'C': Real(1e-3,1e3,'log-uniform'),
             'gamma':Real(1e-1,1e1,'log-uniform')}
```

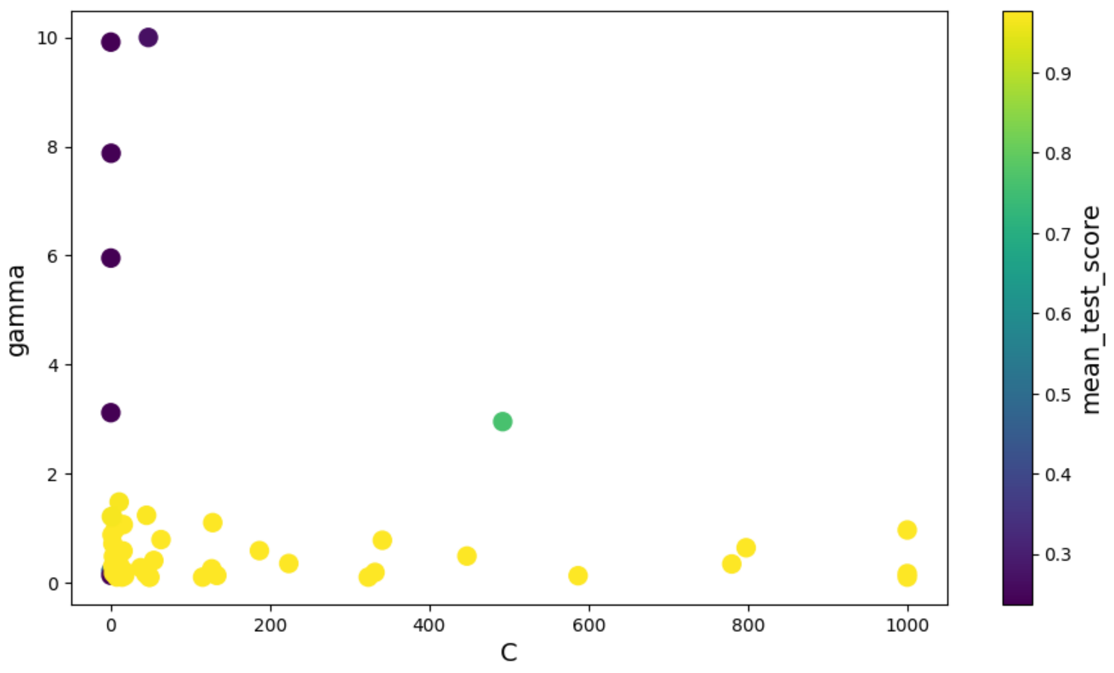
```
[ ]: opt=BayesSearchCV(SVC(),grid,n_iter=50,error_score=0.0,n_points=1,
                       return_train_score=True,verbose=20,n_jobs=10,cv=5,
                       scoring='accuracy',pre_dispatch='2*n_jobs')
     opt.fit(tfidf_train_test[:1440],train.Category)
```

```
[70]: print('best accuracy:',opt.best_score_)
      print(f'best parameters:\n{list(opt.best_params_.keys())}\n{list(opt.
       ↪best_params_.values()):}')
```

```
best accuracy: 0.9770833333333334
best parameters:
['C', 'gamma']
[63.04500782020978, 0.7883281023044573]
```

```
[71]: opt_svm=pd.DataFrame(opt.cv_results_)
      opt_svm[['param_C','param_gamma','mean_test_score']].plot(
      kind='scatter',x='param_C',y='param_gamma',c='mean_test_score',s=100,
          xlabel='C',ylabel='gamma',colorbar=True)
```

```
[71]: <Axes: xlabel='C', ylabel='gamma'>
```



The above plot indicates convergence near the origin.

```
[72]: test['Category']= opt_svm.best_.fit(tfidf_train_test[:1440],train.Category).
        ↪predict(tfidf_train_test[1440:])
      test[['ArticleId','Category']].to_csv('Solution_svm.csv',index=False)
      !kaggle competitions submit -c learn-ai-bbc -f Solution_svm.csv -m "Message"
```

```
100%|                         | 9.23k/9.23k [00:01<00:00, 9.18kB/s]
Successfully submitted to BBC News Classification
```

TEST SET RESULT

Solution_svm.csv
Complete (after deadline) · 16s ago · Message
0.98503   0.98503

```
[73]: table.append(['Support Vector Classifier',0.98,0.99])
```

```
[74]: pd.DataFrame(table[1:],columns=table[0][:]).set_index('Method').
        ↪sort_values(by='Test Accuracy',ascending=False)
```

```
[74]:                               Train Accuracy  Test Accuracy
      Method
      Support Vector Classifier              0.98           0.99
      NMF                                    0.97           0.97
      KNN                                    0.97           0.97
      Random Forest Classifier               0.96           0.97
      KMeans                                 0.96           0.96
      Gradient Boosting Classifier           0.95           0.95
```

```
Hierarchial Clustering                    0.92           0.93
Adaboost Classifier                       0.90           0.93
```

5 MultiLayer Perceptron

- A multilayer perceptron, or, neural network, consists of an input layer, hidden layers (also called Dense layers), and output layers. We also have supporting layers for normalization and sometimes other things (like dropout regularization).
- I will use the neural network, for classifying the articles.
- First, I will use a sequential neural network model using all layers as features in the input.
- Then, I will use a wide and deep model, which first splits the input into two datasets: wide and deep, based on feature importance. The deep one can go through a deep path, i.e., with more hidden layers, and the wide one, through a shallower path.
- I will make this determination based on the term frequency values across the corpus.

The dense layers use some hyperparameters such as the number of units or neurons,`activation_function`, `kernel_initializer`, etc. Let us train the neural network with default fit parameters, i.e., `learning_rate=0.001`, and `batch_size`=32, etc. Later, these will be hyperparameters, along with the number of hidden layers, the number of neurons, and, others, like `max_df` and `min_df`. The `kt.HyperModel` class can be subclassed to form a `keras-tuner` hypermodel, which can then be tuned with `keras-tuner`.

[10]:
```
##Callbacks - extra functionality to preserve, display, and, simplify results
log_dir=os.path.join(os.curdir,"my_models")
def get_path(run_name='a'):
    run_paths=time.strftime(f"{run_name}_%Y_%m_%d_%H_%M_%S")
    return os.path.join(log_dir,run_paths)
checkpoint_cb=keras.callbacks.ModelCheckpoint(get_path()+".keras"
                                              ,save_best_only=True)
callbacks=[checkpoint_cb]
#Data Processing
class shallow_nn_sequential(kt.HyperModel):
    def build(self,hp,max_n=1):
        model=keras.models.Sequential()
        #model.add(keras.Input(shape=(3191,)))
        model.add(keras.layers.BatchNormalization())
        for i in range(hp.
 ↪Int("n_hidden",min_value=0,max_value=max_n,default=0)):
            model.add(keras.layers.Dense(
                                            hp.
 ↪Int(f'neurons_layer_{str(i+1)}',min_value=0,
                                                  max_value=25,default=20,
                                                 ␣
 ↪parent_name='n_hidden',parent_values=list(range(i+1,max_n+1))),
                                            kernel_initializer=hp.
 ↪Choice('hidden_initializer',
                                                                                   ␣
 ↪['he_normal','glorot_uniform'
                                                                                   ␣
 ↪,'lecun_normal'],
                                                                                   ␣
 ↪default='glorot_uniform',
                                                                                   ␣
 ↪parent_name='n_hidden',
                                                                                   ␣
 ↪parent_values=list(range(1,max_n+1)))
                                         ,activation=
                                            hp.Choice('hidden_activation_function',
                                                                  ␣
 ↪['selu','relu','elu','softplus'],default='relu',
                                                       parent_name='n_hidden',
                                                                  ␣
 ↪parent_values=list(range(1,max_n+1)))))


            if hp.Boolean('Dropout',default=True,parent_name='n_hidden',
                          parent_values=list(range(1,max_n+1))):
                model.add(keras.layers.Dropout(hp.
 ↪Float(f"dropout_rate_layer_{str(i+1)}",
                                                           min_value=0.
 ↪0,max_value=0.99,default=0.2,
                                                           step=0.01,
                                                                       ␣
 ↪parent_name='n_hidden',
                                                                   ␣
 ↪parent_values=list(range(i+1,max_n+1)))))

            if hp.
 ↪Boolean('Batch_normalization',default=True,parent_name='n_hidden',
                          parent_values=list(range(1,max_n+1))):
                model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.
 ↪Dense(5,kernel_initializer='glorot_uniform',activation='softmax'))
                                    # hp.
 ↪Choice('output_initializer',['he_normal',
                                    #                                      ␣
 ↪'glorot_uniform'
                                    #                                        ␣
 ↪,'lecun_normal'],
                                    #                                      ␣
 ↪default='glorot_uniform'),
                                    # activation='softmax'))
```

```
        optimizer=keras.optimizers.RMSprop(learning_rate=1e-3)
        #                                    hp.
 ↪Float("learning_rate",min_value=-1e-5,
        #                                                              ␣
 ↪max_value=10.,
        #                                                              ␣
 ↪sampling='linear',default=1e-3,
        #                                                               step=0.01))
        model.compile(loss=keras.losses.
 ↪categorical_crossentropy,metrics=['accuracy'],
                      optimizer=optimizer,)

        return model

    def fit(self,hp,built_model,*args,**kwargs):
        tfidf_vectorizer = TfidfVectorizer(
            max_df=hp.Float('max_df',0.1,1.0,step=0.01,default=0.5),
            min_df=hp.Int('min_df',1,100,step=1,default=8),
            stop_words = 'english',
            ngram_range=(1,5),
            sublinear_tf=True,
        )
        tfidf_train_test = tfidf_vectorizer.fit_transform(train_test)
        lookup = keras.layers.StringLookup(vocabulary = train.Category.
 ↪unique(),output_mode="one_hot",
                                                num_oov_indices = 0,)
        train_category_int = np.array([lookup(label).cpu().numpy() for label in␣
 ↪train.Category]).squeeze()

        X_train_torch_tensor=torch.tensor(tfidf_train_test[:1440].
 ↪toarray(),device='mps',dtype=torch.float32).cpu()
        y_train_torch_tensor=torch.
 ↪tensor(train_category_int,device='mps',dtype=torch.float32).cpu()

        return built_model.
 ↪fit(X_train_torch_tensor,y_train_torch_tensor,**kwargs,shuffle=False)
                                    #batch_size=hp.
 ↪Int('batch_size',min_value=1,max_value=1612,
                                     #                ␣
 ↪sampling='log',default=32,step=2), )
hp_shallow=kt.HyperParameters()
built_model=shallow_nn_sequential().build(hp_shallow)
shallow_nn_sequential().fit(hp_shallow,built_model,epochs=20,validation_split=0.
 ↪1)
```

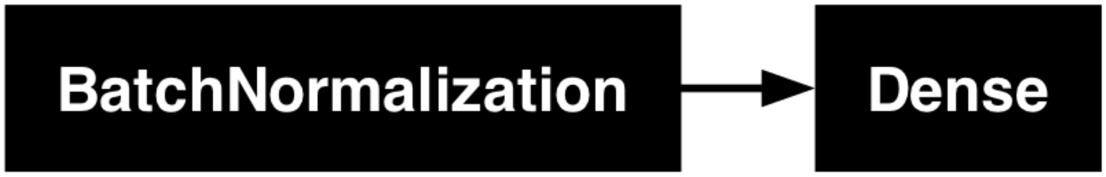
```
#print(f'{bcolors.BOLD}test mean absolute error:\t {built_model.
 ↪evaluate(X_test_torch_tensor,verbose=0):.3f} mA/cm^2')
print(f'{bcolors.ENDC}Default HyperParameters\n{hp_shallow.values}\n\n')
print(f'{bcolors.BOLD}MODEL')
keras.utils.plot_model(built_model,rankdir='LR',to_file="shallow_nn_default.
 ↪png",show_shapes=False,show_layer_names=False,show_trainable=False)
```

```
Epoch 1/20
41/41             1s 13ms/step -
accuracy: 0.7433 - loss: 0.9289 - val_accuracy: 0.6181 - val_loss: 1.4690
Epoch 2/20
41/41             0s 11ms/step -
accuracy: 0.9912 - loss: 0.1336 - val_accuracy: 0.6181 - val_loss: 1.3807
Epoch 3/20
41/41             0s 10ms/step -
accuracy: 0.9987 - loss: 0.0429 - val_accuracy: 0.6458 - val_loss: 1.2812
Epoch 4/20
41/41             0s 10ms/step -
accuracy: 1.0000 - loss: 0.0143 - val_accuracy: 0.7708 - val_loss: 1.1606
Epoch 5/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 0.0053 - val_accuracy: 0.8889 - val_loss: 1.0233
Epoch 6/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 0.0024 - val_accuracy: 0.9236 - val_loss: 0.8782
Epoch 7/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.9583 - val_loss: 0.7289
Epoch 8/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 8.9348e-04 - val_accuracy: 0.9583 - val_loss: 0.5818
Epoch 9/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 6.7420e-04 - val_accuracy: 0.9653 - val_loss: 0.4460
Epoch 10/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 5.4233e-04 - val_accuracy: 0.9653 - val_loss: 0.3306
Epoch 11/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 4.5426e-04 - val_accuracy: 0.9653 - val_loss: 0.2414
Epoch 12/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 3.9119e-04 - val_accuracy: 0.9722 - val_loss: 0.1787
Epoch 13/20
41/41             0s 8ms/step -
accuracy: 1.0000 - loss: 3.4374e-04 - val_accuracy: 0.9792 - val_loss: 0.1385
Epoch 14/20
41/41             0s 8ms/step -
```

```
accuracy: 1.0000 - loss: 3.0671e-04 - val_accuracy: 0.9792 - val_loss: 0.1150
Epoch 15/20
41/41           0s 8ms/step -
accuracy: 1.0000 - loss: 2.7698e-04 - val_accuracy: 0.9792 - val_loss: 0.1025
Epoch 16/20
41/41           0s 8ms/step -
accuracy: 1.0000 - loss: 2.5258e-04 - val_accuracy: 0.9792 - val_loss: 0.0969
Epoch 17/20
41/41           0s 8ms/step -
accuracy: 1.0000 - loss: 2.3218e-04 - val_accuracy: 0.9792 - val_loss: 0.0952
Epoch 18/20
41/41           0s 8ms/step -
accuracy: 1.0000 - loss: 2.1488e-04 - val_accuracy: 0.9792 - val_loss: 0.0955
Epoch 19/20
41/41           0s 8ms/step -
accuracy: 1.0000 - loss: 2.0000e-04 - val_accuracy: 0.9792 - val_loss: 0.0966
Epoch 20/20
41/41           0s 8ms/step -
accuracy: 1.0000 - loss: 1.8706e-04 - val_accuracy: 0.9792 - val_loss: 0.0979
Default HyperParameters
{'n_hidden': 0, 'max_df': 0.5, 'min_df': 8}
```

```
MODEL
```

[10]:



[11]:

```
shallow_nn_bo=kt.BayesianOptimization(hypermodel=shallow_nn_sequential(),
                        objective="val_accuracy",
                        max_trials=20,
                        directory="tuning_results_news_clf",
                        project_name="shallow_nn_run_11",)
shallow_nn_bo.search(verbose=1,
                     validation_split=0.1,epochs=20,)
```

```
Reloading Tuner from tuning_results_news_clf/shallow_nn_run_11/tuner0.json
```

[16]:

```
shallow_nn_bo.results_summary(3)
```

```
Results summary
Results in tuning_results_news_clf/shallow_nn_run_11
Showing 3 best trials
```

```
Objective(name="val_accuracy", direction="max")

Trial 08 summary
Hyperparameters:
n_hidden: 0
max_df: 0.13
min_df: 39
Score: 0.9861111044883728

Trial 04 summary
Hyperparameters:
n_hidden: 1
neurons_layer_1: 13
hidden_initializer: lecun_normal
hidden_activation_function: selu
Dropout: False
dropout_rate_layer_1: 0.42
Batch_normalization: False
max_df: 0.21000000000000002
min_df: 13
Score: 0.9861111044883728

Trial 02 summary
Hyperparameters:
n_hidden: 0
max_df: 0.66
min_df: 6
Score: 0.9861111044883728
```

[ ]:
```
hp_best=shallow_nn_bo.get_best_hyperparameters(3)[2]
built_model=shallow_nn_sequential().build(hp_best)
shallow_nn_sequential().fit(hp_best,built_model,epochs=20,validation_split=0.1)

tfidf_vectorizer = TfidfVectorizer(
    max_df=hp_best.values['max_df'],
    min_df=hp_best.values['min_df'],
    stop_words = 'english',
    ngram_range=(1,5),
    sublinear_tf=True,
)
tfidf_train_test = tfidf_vectorizer.fit_transform(train_test)
lookup = keras.layers.StringLookup(vocabulary = train.Category.
  ↪unique(),output_mode="one_hot",
                                                  num_oov_indices = 0,)
train_category_int = np.array([lookup(label).cpu().numpy() for label in train.
  ↪Category]).squeeze()
```

```
X_test_torch_tensor=torch.tensor(tfidf_train_test[1440:].
 ↪toarray(),device='mps',dtype=torch.float32).cpu()
result=built_model.predict(X_test_torch_tensor)
result = np.array([out==max(out) for out in result]).astype(int)
yp = [np.take(train.Category.unique(),np.argwhere(result[i] == 1.0)[..., 0])[0]␣
 ↪for i in range(len(result))]
```

[102]:
```
test['Category']= yp
test[['ArticleId','Category']].to_csv('Solution_snn.csv',index=False)
!kaggle competitions submit -c learn-ai-bbc -f Solution_snn.csv -m "Message"
```

```
100%|                         | 9.21k/9.21k [00:01<00:00, 9.40kB/s]
Successfully submitted to BBC News Classification
```

TEST SET RESULT

Solution_snn.csv
Complete (after deadline) · 1m ago · Message
0.98367 0.98367

[103]:
```
table.append(['Shallow Neural Network',0.98,0.98])
```

Now I will split the dataset into two, with higher and lower importance, using the term frequencies across the corpus. Then, I will use it in the model, which consists of the deep input going through more hidden layers, and the wide one through lesser. Then, the two get concatenated, and go through some more hidden layers, this time together. The Wide and Deep model was introduced by Heng-Tze-Cheng *et al.* in a 2016 paper (Wide and Deep).

[106]:
```
print('features\t',tfidf_train_test.shape[1])
```

```
features         11937
```

[18]:
```
tfidf_vectorizer = TfidfVectorizer(
    max_df=hp_best.values['max_df'],
    min_df=hp_best.values['min_df'],
    stop_words = 'english',
    ngram_range=(1,5),
    sublinear_tf=True,
    max_features=9000
)
tfidf_train_test = tfidf_vectorizer.fit_transform(train_test)
X_train_torch_tensor_deep=torch.tensor(tfidf_train_test[:1440].
 ↪toarray(),device='mps',dtype=torch.float32).cpu()
X_test_torch_tensor_deep=torch.tensor(tfidf_train_test[1440:].
 ↪toarray(),device='mps',dtype=torch.float32).cpu()
```

[19]:
```
tfidf_vectorizer = TfidfVectorizer(
    max_df=hp_best.values['max_df'],
    min_df=hp_best.values['min_df'],
    stop_words = 'english',
    ngram_range=(1,5),
    sublinear_tf=True,
    max_features=11937
)
tfidf_train_test = tfidf_vectorizer.fit_transform(train_test).toarray()
tfidf_train_test=np.array([sample[9000:11937] for sample in tfidf_train_test]).
  ↪squeeze()
X_train_torch_tensor_wide=torch.tensor(tfidf_train_test[:
  ↪1440],device='mps',dtype=torch.float32).cpu()
X_test_torch_tensor_wide=torch.tensor(tfidf_train_test[1440:
  ↪],device='mps',dtype=torch.float32).cpu()
```

[20]:
```
y_train_torch_tensor=torch.tensor(train_category_int,device='mps',dtype=torch.
  ↪float32).cpu()
```

[48]:
```
class nn_split(kt.HyperModel):
    """
    Builds a custom model by subclassing from the base kt.HyperModel Class.
    It overrides the kt.Hypermodel 'build' and 'fit' method by the 'build' and␣
  ↪'fit' methods definded below.
    The 'hp' parameter is an instance of the kt.HyperParameters class and␣
  ↪allows
    us to define hyperparameters in the 'build' and 'fit' method, therefore it␣
  ↪is passed to both the methods, along with self,
    which is a reference to the class itself.
    'hp' stores the names and values of the defined hyperparameters.
    """
    def build(self,hp):
        """Parameters
        hp: kt.HyperParameters instance

        Returns -> compiled model
        """
        ##Inputs
        input_D=keras.Input(shape=(9000,),name="Deep Input")
        input_W=keras.Input(shape=(2937,),name="Wide Input")

        ##Input normalization
        input_D_norm=keras.layers.BatchNormalization()(input_D)
        input_W_norm=keras.layers.BatchNormalization()(input_W)

        ##Hidden layers for deep input
        #Intialize a matrix of shape hidden_D
        hidden_D=np.empty(shape=hp.
  ↪Int("input_D+n_hidden_D",min_value=1,max_value=6,step=1,default=2),dtype=object)

        #First layer
        hidden_D[0]=input_D_norm
        ##Loop for intitializing hidden_D number of layers
        for i in range(1,len(hidden_D)):
            #Intitialize hidden_D layers
            hidden_D[i]=keras.layers.Dense(
                                                hp.
↪Int(f'neurons_D_layer_{str(i)}',min_value=1,max_value=25,default=10,step=1,
                                                        ␣
↪parent_name='input_D+n_hidden_D',parent_values=list(range(i+1,7))),
                                                kernel_initializer=hp.
↪Choice('hidden_D_initializer',['he_normal','glorot_uniform','lecun_normal'],
                                                                            ␣
↪default='glorot_uniform',
                                                                            ␣
↪parent_name='input_D+n_hidden_D',parent_values=list(range(2,7)))
                                                 ,activation=
                                                hp.
↪Choice('hidden_D_activation_function',['selu','relu','elu','softplus'],default='relu',
                                                            ␣
↪parent_name='input_D+n_hidden_D',
                                                            ␣
↪parent_values=list(range(2,7))))(hidden_D[i-1])
            #dropouts for each
            if hp.Boolean('Dropout_D',default=True):
                hidden_D[i]=keras.layers.Dropout(hp.
↪Float("dropout_rate_D",min_value=0.0,max_value=0.99,default=0.2,step=0.04,
                                                                    ␣
↪parent_name='Dropout_D',parent_values=[True]))(hidden_D[i])

            if hp.
↪Boolean('Batch_normalization_D',default=True,parent_name='input_D+n_hidden_D',parent_values
↪
                hidden_D[i]=keras.layers.BatchNormalization()(hidden_D[i])


            #hidden_D[i]=keras.layers.BatchNormalization()(hidden_D[i-1])

        #output of the hidden_D layers
        output_D=keras.layers.Dense(10,activation=hp.
↪Choice("D_output_activation",['selu','relu','elu','softplus'],default='relu'))(hidden_D[-1]

        ##Wide Inputs Processing
        #Intialize a matrix of shape hidden_W
        hidden_W=np.empty(shape=hp.
↪Int("input_W+n_hidden_W",min_value=1,max_value=6,step=1,default=1),dtype=object)
```

```
        #First Layer
        hidden_W[0]=(input_W_norm)


        ##Loop for intitializing hidden_W number of layers
        for i in range(1,len(hidden_W)):
            #Intitialize hidden_W layers
            hidden_W[i]=keras.layers.Dense(
                                            hp.
↪Int(f'neurons_W_layer_{str(i)}',min_value=1,max_value=25,default=10,step=1,
                                                    ␣
↪parent_name='input_W+n_hidden_W',parent_values=list(range(i+1,7))),
                                            kernel_initializer=hp.
↪Choice("hidden_W_initializer",['he_normal','glorot_uniform','lecun_normal'],
                                                                        ␣
↪default='glorot_uniform',
                                                                        ␣
↪parent_name='input_W+n_hidden_W',parent_values=list(range(2,7)))
                                             ,activation=
                                            hp.
↪Choice('hidden_W_activation_function',['selu','relu','elu','softplus'],default='relu',
                                                        ␣
↪parent_name='input_W+n_hidden_W',
                                                        ␣
↪parent_values=list(range(2,7))))(hidden_W[i-1])
            #dropouts for each
            if hp.Boolean('Dropout_W',default=True):
                hidden_W[i]=keras.layers.Dropout(hp.
↪Float("W_dropout_rate",min_value=0.0,max_value=0.99,default=0.2,step=0.04,
                                                                ␣
↪parent_name='Dropout_W',parent_values=[True]))(hidden_W[i])

            if hp.
↪Boolean('Batch_normalization_W',default=True,parent_name='input_W+n_hidden_W',parent_values
↪
                hidden_W[i]=keras.layers.BatchNormalization()(hidden_W[i])

            #hidden_W[i]=keras.layers.BatchNormalization()(hidden_W[i-1])
        #output of the wide input processing
        if len(hidden_W)!=1:
            output_W=keras.layers.Dense(10,activation=hp.
↪Choice("W_output_activation",['selu','relu','elu','softplus'],default='relu'))(hidden_W[-1]
        else:
            output_W=hidden_W[-1]
        #concatenation of the deep and wide outputs
        concat=keras.layers.concatenate(inputs=[output_D,output_W])
        #normalizing the concated ouput
        concat_norm=keras.layers.BatchNormalization()(concat)

        #initialize the output layer
        output=keras.layers.Dense(5,activation='softmax'
                                 )(concat_norm)

        #initialize the model
        inputs=[input_D,input_W]
        model=keras.Model(inputs=inputs,outputs=output)

        #compile the model
        optimizer=keras.optimizers.RMSprop(learning_rate=hp.
 ↪Float("learning_rate",min_value=1e-5,max_value=1,default=1e-3,step=.01))
        model.compile(loss=keras.losses.
 ↪categorical_crossentropy,metrics=['accuracy'],
                      optimizer=optimizer)

        return model

    def fit(self,hp,model,*args,**kwargs):
        """Parameters
        x: Training data Features
        y: Training data Targets
        hp: kt.HyperParameters instance
        model: compiled model

        Returns -> fitted model
        """
        return model.fit(*args, batch_size=hp.Int("batch_size",min_value=2,
                                                                     ␣
 ↪max_value=1612,sampling='log',step=2,default=32),shuffle=True,**kwargs)

X={"Deep Input":X_train_torch_tensor_deep,
   "Wide Input":X_train_torch_tensor_wide}
Y=y_train_torch_tensor

X_={"Deep Input":X_test_torch_tensor_deep,
   "Wide Input":X_test_torch_tensor_wide}
#    print(f'{bcolors.BOLD}mean absolute error\t {model.evaluate(X_,Y_)[0]:.3f}␣
 ↪mA/cm^2')
hp=kt.HyperParameters()
HyperModel=nn_split()
model=HyperModel.build(hp)
HyperModel.fit(hp,model,X,Y,epochs=20,validation_split=0.1,verbose=1)
```

```
#print(f'{bcolors.BOLD}mean absolute error\t {model.evaluate(X_,Y_):.3f} mA/
↪cm^2')
print(f'{bcolors.ENDC}Default HyperParameters\n{hp.values}')
```

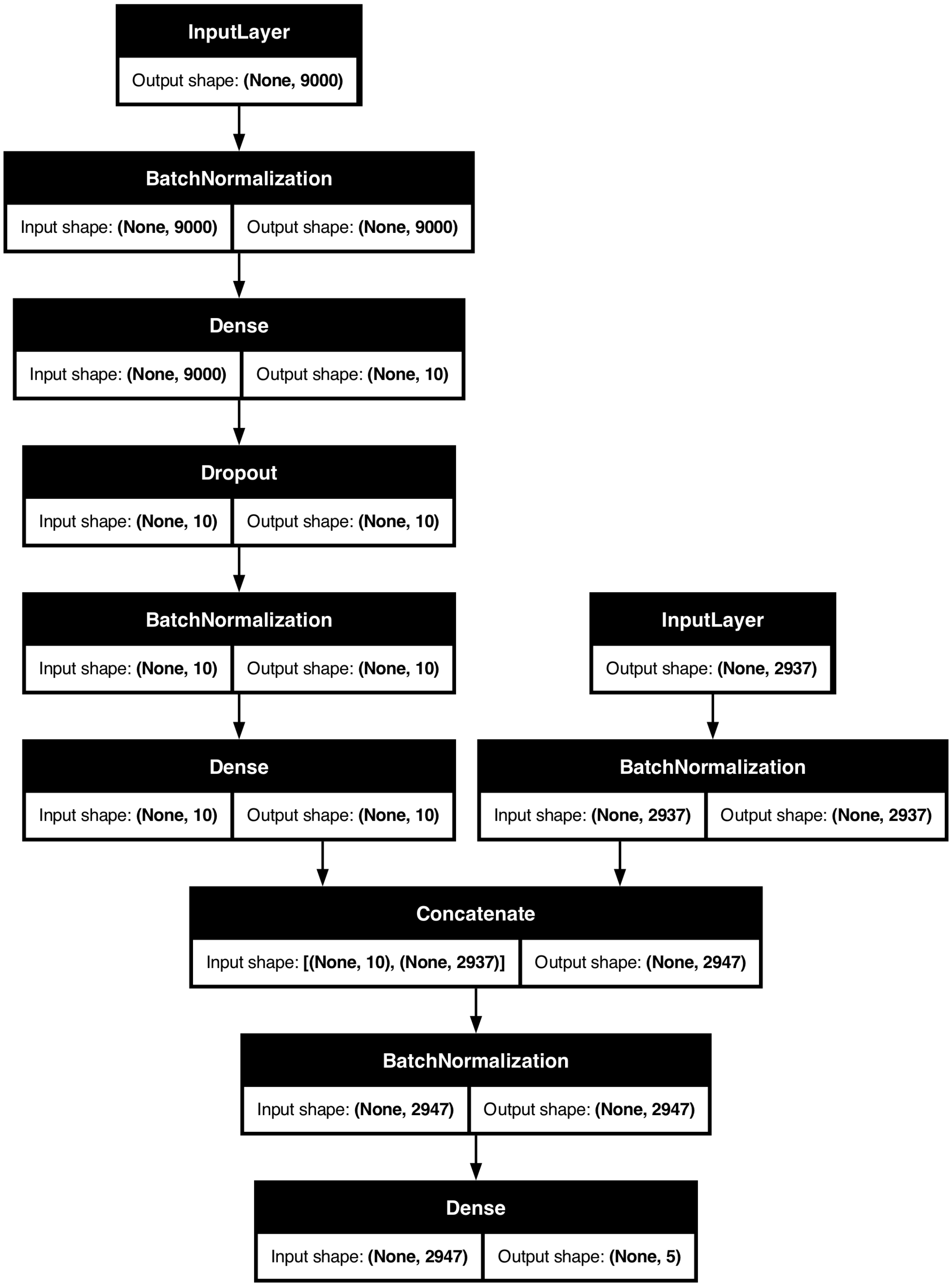
```
Epoch 1/20
41/41              2s 23ms/step -
accuracy: 0.5346 - loss: 1.2243 - val_accuracy: 0.2292 - val_loss: 1.5462
Epoch 2/20
41/41              1s 16ms/step -
accuracy: 0.9302 - loss: 0.3812 - val_accuracy: 0.5486 - val_loss: 1.4940
Epoch 3/20
41/41              1s 17ms/step -
accuracy: 0.9709 - loss: 0.1899 - val_accuracy: 0.4097 - val_loss: 1.4169
Epoch 4/20
41/41              1s 15ms/step -
accuracy: 0.9891 - loss: 0.0897 - val_accuracy: 0.5556 - val_loss: 1.3199
Epoch 5/20
41/41              1s 16ms/step -
accuracy: 0.9987 - loss: 0.0491 - val_accuracy: 0.7153 - val_loss: 1.1826
Epoch 6/20
41/41              1s 15ms/step -
accuracy: 0.9992 - loss: 0.0263 - val_accuracy: 0.7778 - val_loss: 0.9988
Epoch 7/20
41/41              1s 15ms/step -
accuracy: 0.9989 - loss: 0.0220 - val_accuracy: 0.7778 - val_loss: 0.8533
Epoch 8/20
41/41              1s 21ms/step -
accuracy: 0.9998 - loss: 0.0125 - val_accuracy: 0.8819 - val_loss: 0.6460
Epoch 9/20
41/41              1s 21ms/step -
accuracy: 0.9996 - loss: 0.0079 - val_accuracy: 0.9444 - val_loss: 0.4330
Epoch 10/20
41/41              1s 19ms/step -
accuracy: 1.0000 - loss: 0.0068 - val_accuracy: 0.9444 - val_loss: 0.3119
Epoch 11/20
41/41              1s 16ms/step -
accuracy: 0.9994 - loss: 0.0103 - val_accuracy: 0.9375 - val_loss: 0.2429
Epoch 12/20
41/41              1s 15ms/step -
accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 0.9444 - val_loss: 0.1864
Epoch 13/20
41/41              1s 18ms/step -
accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 0.9375 - val_loss: 0.1678
Epoch 14/20
41/41              1s 16ms/step -
accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 0.9306 - val_loss: 0.1566
Epoch 15/20
41/41              1s 15ms/step -
```

```
accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.9375 - val_loss: 0.1628
Epoch 16/20
41/41              1s 15ms/step -
accuracy: 0.9968 - loss: 0.0054 - val_accuracy: 0.9444 - val_loss: 0.1719
Epoch 17/20
41/41              1s 15ms/step -
accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9444 - val_loss: 0.1838
Epoch 18/20
41/41              1s 15ms/step -
accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.9583 - val_loss: 0.1937
Epoch 19/20
41/41              1s 15ms/step -
accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 0.9514 - val_loss: 0.2068
Epoch 20/20
41/41              1s 20ms/step -
accuracy: 0.9995 - loss: 0.0019 - val_accuracy: 0.9444 - val_loss: 0.2180
Default HyperParameters
{'input_D+n_hidden_D': 2, 'neurons_D_layer_1': 10, 'hidden_D_initializer':
'glorot_uniform', 'hidden_D_activation_function': 'relu', 'Dropout_D': True,
'dropout_rate_D': 0.2, 'Batch_normalization_D': True, 'D_output_activation':
'relu', 'input_W+n_hidden_W': 1, 'learning_rate': 0.001, 'batch_size': 32}
```

The figure below plots the above model.

[51]:
```python
keras.utils.plot_model(model,to_file="split_nn.
↪png",show_shapes=True,show_layer_names=False,show_trainable=False)
```

[51]:


InputLayer
Output shape: (None, 9000)
BatchNormalization
Input shape: (None, 9000)
Output shape: (None, 9000)
Dense
Input shape: (None, 9000)
Output shape: (None, 10)
Dropout
Input shape: (None, 10)
Output shape: (None, 10)
BatchNormalization
Input shape: (None, 10)
Output shape: (None, 10)
InputLayer
Output shape: (None, 2937)
Dense
Input shape: (None, 10)
Output shape: (None, 10)
BatchNormalization
Input shape: (None, 2937)
Output shape: (None, 2937)
Concatenate
Input shape: [(None, 10), (None, 2937)]
Output shape: (None, 2947)
BatchNormalization
Input shape: (None, 2947)
Output shape: (None, 2947)
Dense
Input shape: (None, 2947)
Output shape: (None, 5)

[ ]:
```
result=model.predict(X_)
result = np.array([out==max(out) for out in result]).astype(int)
yp = [np.take(train.Category.unique(),np.argwhere(result[i] == 1.0)[..., 0])[0]␣
  ↪for i in range(len(result))]
test['Category']= yp
test[['ArticleId','Category']].to_csv('Solution_ssnn.csv',index=False)
!kaggle competitions submit -c learn-ai-bbc -f Solution_ssnn.csv -m "Message"
```

Next, let us do a small bayesian search to tune this model's hyperparameters.

[27]:
```
split_nn_bo=kt.BayesianOptimization(hypermodel=nn_split(),
                        objective="val_accuracy",
                        max_trials=20,
                        directory="tuning_results_news_clf",
                        project_name="split_nn_bo_final",
                        overwrite=False    )
split_nn_bo.search(X,Y,validation_split=0.1,epochs=10)
```

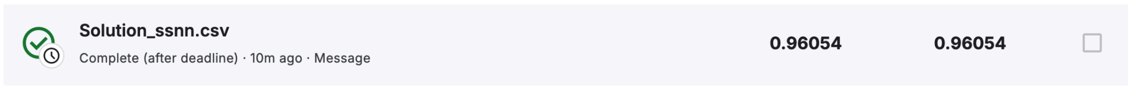
```
Trial 20 Complete [00h 04m 14s]
val_accuracy: 0.2291666716337204

Best val_accuracy So Far: 0.9513888955116272
Total elapsed time: 00h 14m 11s
```

[53]:
```
best_hp_split=split_nn_bo.get_best_hyperparameters(5)[0]
print(f'{bcolors.ENDC} Best HyperParameters\n{best_hp_split.values}')
```

```
 Best HyperParameters
{'input_D+n_hidden_D': 6, 'neurons_D_layer_1': 18, 'hidden_D_initializer':
'lecun_normal', 'hidden_D_activation_function': 'elu', 'Dropout_D': False,
'Batch_normalization_D': True, 'D_output_activation': 'selu',
'input_W+n_hidden_W': 2, 'neurons_W_layer_1': 7, 'hidden_W_initializer':
'he_normal', 'hidden_W_activation_function': 'relu', 'Dropout_W': True,
'W_dropout_rate': 0.84, 'Batch_normalization_W': True, 'W_output_activation':
'selu', 'learning_rate': 0.08001, 'batch_size': 4, 'neurons_D_layer_2': 2,
'neurons_D_layer_3': 14, 'neurons_D_layer_4': 25, 'neurons_D_layer_5': 25}
```

The above hyperparameters confirm our assumptions that wide inputs are of lower importance hence layers_D:layers_W=3 in the best found hyperparameters.

[ ]:
```
model=HyperModel.build(best_hp_split)
HyperModel.fit(best_hp_split,model,X,Y,epochs=20,validation_split=0.1,verbose=1)
result=model.predict(X_)
result = np.array([out==max(out) for out in result]).astype(int)
yp = [np.take(train.Category.unique(),np.argwhere(result[i] == 1.0)[..., 0])[0]␣
  ↪for i in range(len(result))]
test['Category']= yp
test[['ArticleId','Category']].to_csv('Solution_ssnn.csv',index=False)
!kaggle competitions submit -c learn-ai-bbc -f Solution_ssnn.csv -m "Message"
```

TEST SET RESULT 

Solution_ssnn.csv
Complete (after deadline) · 10m ago · Message
0.96054 0.96054

6 Summary of results

Below is a table of every method used and the accuracies.

```
[104]: pd.DataFrame(table[1:],columns=table[0][:]).set_index('Method').
         ↪sort_values(by='Test Accuracy',ascending=False)
```

```
[104]:                                Train Accuracy  Test Accuracy
       Method
       Support Vector Classifier                0.98           0.99
       Shallow Neural Network                   0.98           0.98
       NMF                                      0.97           0.97
       KNN                                      0.97           0.97
       Random Forest Classifier                 0.96           0.97
       KMeans                                   0.96           0.96
       Gradient Boosting Classifier             0.95           0.95
       Hierarchial Clustering                   0.92           0.93
       Adaboost Classifier                      0.90           0.93
```

References: https://scikit-learn.org/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.
glr-auto-examples-applications-plot-topics-extraction-with-nmf-lda-py