DMT2023_HW3

May 19, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Aur, Marina Iuliana, 1809715

Balestrucci, Sophia, 1713638

0.2 Homework 3

The homework consists of two parts:

1. Dimensionality Reduction

and

2. Supervised Learning

   Ensure that the notebook can be faithfully reproduced by anyone (hint: pseudo random number generation).

   If you need to set a random seed, set it to `160`.

1 Part 1

In this part of the homework, you have to deal with Dimensionality Reduction.

[ ]:
```
#REMOVE_OUTPUT#
!pip install --upgrade --no-cache-dir gdown
from bs4 import BeautifulSoup
#YOUR CODE STARTS HERE#
import pandas as pd
import numpy as np
from gensim import corpora
from gensim.models import LsiModel
from gensim.models.coherencemodel import CoherenceModel
import nltk
nltk.download('stopwords')
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import string
import matplotlib.pyplot as plt
import re
from numpy import count_nonzero
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

1.1 Part 1.1

The data you need to process comes from the book *Le Morte D'Arthur* by Thomas Malory.

You have to carry out Topic Modeling on book chapters.

The goal is to achieve a topic division within the following limits:

- The total computation may not exceed 10 minutes (starting from Part 1.1.5; Parts 1.1.1 to 1.1.4 are not considered for time calculation)
- The division into topics must be the "best one"

1.1.1 1.1.1

Download the data from the Drive link (code already provided).

[ ]:
```
#REMOVE_OUTPUT#
!gdown 1zHgvidy9FvhZvE68SOmXWkoF-hHMpiUL
!gdown 1VjpTkFcbfaLIi4TXVafokW9e_bvGnfut
```

1.1.2 1.1.2

Parse the HTML. **Part** of code already provided: follow the comments to complete the code.

[46]:
```
with open('The Project Gutenberg eBook of Le Morte D’Arthur, Volume I (of II),␣
↪by Thomas Malory.html') as fp:
    vol1 = BeautifulSoup(fp, 'html.parser')
with open('The Project Gutenberg eBook of Le Morte D’Arthur, Volume II (of II),␣
↪by Thomas Malory.html') as fp:
    vol2 = BeautifulSoup(fp, 'html.parser')

def clean_text(txt):
    words_to_put_space_before = [".",",",";",":","’","'"]
    words_to_lowercase =␣
↪["First","How","Some","Yet","Of","A","The","What","Fifth"]

    app = txt.replace("\n"," ")
    for word in words_to_put_space_before:
        app = app.replace(word," "+word)
    for word in words_to_lowercase:
        app = app.replace(word+" ",word.lower()+" ")
    return app.strip()

def parse_html(soup):
    titles = []
    texts = []
    for chapter in soup.find_all("h3"):
        chapter_title = chapter.text
        if "CHAPTER" in chapter_title:
            chapter_title = clean_text("".join(chapter_title.split(".")[1:]))
            titles.append(chapter_title)

            chapter_text = [p.text for p in chapter.findNextSiblings("p")]
            chapter_text = clean_text(" ".join(chapter_text))
            texts.append(chapter_text)
    return titles, texts
```

[47]:
```
#YOUR CODE STARTS HERE#
#Extract all the chapters' titles and texts from the two volumes
vol1_titles, vol1_texts = parse_html(vol1)
```

```
vol2_titles, vol2_texts = parse_html(vol2)







#Transform the list into a pandas DataFrame.
d1 = {'Title' : vol1_titles, 'Text' : vol1_texts}
df_vol1 = pd.DataFrame(d1)
d2 = {'Title' : vol2_titles, 'Text' : vol2_texts}
df_vol2 = pd.DataFrame(d2)
df_book = pd.concat([df_vol1, df_vol2], ignore_index = True)
df_book['docno'] = [str(i) for i in range(len(df_book))]

#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[48]:

```
#YOUR CODE STARTS HERE#
print("The first 8 rows of DataFrame 'df_book':")
df_book.head(8)





#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
The first 8 rows of DataFrame 'df_book':
```

[48]:

```
                                              Title  \
0  first , how Uther Pendragon sent for the duke …
1  how Uther Pendragon made war on the duke of Co…
2     of the birth of King Arthur and of his nurture
3               of the death of King Uther Pendragon
4  how Arthur was chosen king , and of wonders an…
5  how King Arthur pulled out the sword divers times
6  how King Arthur was crowned , and how he made …
7  how King Arthur held in Wales , at a Pentecost…

                                               Text docno
0  It befell in the days of Uther Pendragon , whe…     0
1  Then Ulfius was glad , and rode on more than a…     1
2  Then Queen Igraine waxed daily greater and gre…     2
3  Then within two years King Uther fell sick of …     3
4  Then stood the realm in great jeopardy long wh…     4
```

5	Now assay , said Sir Ector unto Sir Kay . And …	5
6	And at the feast of Pentecost all manner of me…	6
7	Then the king removed into Wales , and let cry…	7

1.1.3 1.1.3

Extract character's names from the **titles** only. **Part** of code already provided: follow the comments to complete the code.

[49]:
```
all_characters = set()
def extract_character_names_from_string(string_to_parse):
    special_tokens = ["of","the","le","a","de"]

    remember = ""
    last_is_special_token = False

    tokens = string_to_parse.split(" ")
    characters_found = set()
    for i,word in enumerate(tokens):
        if word[0].isupper() or (remember!="" and word in special_tokens):
            #word = word.replace("'s","").replace("’s","")
            last_is_special_token = False
            if remember!="":
                if word in special_tokens:
                    last_is_special_token = True
                remember = remember+" "+word
            else: remember = word
        else:
            if remember!="":
                if last_is_special_token:
                    for tok in special_tokens:
                        remember = remember.replace(" "+tok,"")
                characters_found.add(remember)
            remember = ""
            last_is_special_token = False
    return characters_found

#all_characters = set([x for x in all_characters if x[-2:]!="'s"])
```

[50]:
```
#YOUR CODE STARTS HERE#
#Extract all characters' names
for title in df_book['Title']:
 all_characters.update(extract_character_names_from_string(title))
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

[51]:
```
#YOUR CODE STARTS HERE#
knights = []
print("The names of all the knights:\n")
for name in all_characters:
  if 'sir' in name.lower() and name not in knights:
    knights.append(name)
    print(name)

#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
The names of all the knights:

Sir Epinogris
Sir Bors
Sir Lavaine
Sir Galahalt
Sir Lamorak de Galis
Sir Galihodin
Sir Accolon of Gaul
Sir Dinadan
Sir Blamore
Sir Persant
Sir Belliance
Sir Ector
Sir Pelleas
Sir Berluse
Sir Kay
Sir Bliant
Sir Accolon
Sir Persant of Inde
Sir Carados
Sir Bedivere
Sir Nabon
Sir Frol
Sir Meliagaunce
Sir Uwaine
Sir Archade
Sir Tor
Sir Lanceor
Sir Meliagrance
Sir Uriens
Sir Launcelot
Sir Brian
```

Sir Mador
Sir Elias
Sir Malgrin
Sir Lionel
Sir Urre
Sir Percivale
Sir Sadok
Sir Turquine
Sir Palomides
Sir Breuse Saunce Pité
Sir Safere
Sir Pedivere
Sir Tristram de Liones
Sir Dagonet
Sir Agravaine
Sir Aglovale
Sir Anguish
Sir Gareth
Sir Breunor
Sir Galahad
Sir Marhaus
Sir Colgrevance
Sir Sagramore le Desirous
Sir Mordred
Sir Pervivale
Sir Amant
Sir Suppinabiles
Sir Alisander
Sir Bleoberis
Sir Gaheris
Sir Lamorak
Sir Lancelot
Sir Segwarides
Sir Tristram
Sir Gawaine
Sir Beaumains

1.1.4 1.1.4

Preprocess the data

Consider only the titles

Each document must be a list of terms

Discard documents that have less than 10 (non-unique) words before the preprocessing (split by whitespace, ignore punctuation)

After preprocessing, each document must be represented by at least 5 tokens

- Several preprocessing options are possible

[52]:
```
#YOUR CODE STARTS HERE#

def preprocess_data(doc_, token_min_length=1):
    en_stop = set(stopwords.words('english')) # create English stop words list
    p_stemmer = PorterStemmer() # create p_stemmer of class PorterStemmer

    processed_tokenized_texts = []

    for text in doc_: # loop through document list
        lowercase_text = text.lower()
        pattern = re.compile(r'[^a-z]+')
        cleaned_text = pattern.sub(' ', lowercase_text).strip() # clean text:␣
↪replace pattern with space
        tokenized_text = cleaned_text.split(" ") # divide text in tokens
        stopped_tokens = [token for token in tokenized_text if not token in␣
↪en_stop] # remove stop words from tokens
        if token_min_length>1:
          meaningful_tokens = [token for token in stopped_tokens if len(token)␣
↪>= token_min_length] # remove very small words, length < 3
        else:
          meaningful_tokens = stopped_tokens
        stemmed_tokens = [p_stemmer.stem(token) for token in meaningful_tokens]␣
↪# stem tokens
        processed_tokenized_texts.append(stemmed_tokens) # add tokens to list
    return processed_tokenized_texts

# Create the list of documents considering only the title
docs=[]
for index, row in df_book.iterrows():
  new_string = row['Title'].translate(str.maketrans('', '', string.punctuation))
  unique=len(set(new_string.split(" ")))

  # Check if there are more then 10 unique words
  if unique>=10:
    docs.append(row['Title'])
```

```
# Create the clean doc
clean_docs=preprocess_data(docs, token_min_length=3)




#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[53]:
```
#YOUR CODE STARTS HERE#
for i in range(len(clean_docs)):
  if 'bediver' in clean_docs[i]:
    print(clean_docs[i])





#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
['sir', 'bediver', 'found', 'morrow', 'dead', 'hermitag', 'abod', 'hermit']
```

1.1.5 1.1.5

Build a dictionary of the terms in the documents.

[54]:
```
#YOUR CODE STARTS HERE#

# Creating the term dictionary of our courpus, where every unique term is␣
↪assigned an index
dictionary = corpora.Dictionary(clean_docs)











#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[55]:
```
#YOUR CODE STARTS HERE#
print("The 5 most common terms: ")
dictionary.most_common(n=5)





#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
The 5 most common terms:
```

[55]:
```
[('sir', 589),
 ('king', 173),
 ('launcelot', 148),
 ('tristram', 130),
 ('knight', 127)]
```

1.1.6 1.1.6

Perform a document-term encoding of the dataset.

- Several encodings are possible

[56]:
```
#YOUR CODE STARTS HERE#

# Converting list of documents (corpus) into Document Term Matrix using␣
↪dictionary
doc_term_matrix = [dictionary.doc2bow(doc) for doc in clean_docs]













#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[83]:
```
#YOUR CODE STARTS HERE#
total_ele = sum(len(row) for row in doc_term_matrix)
sparsity = 1.0 - (count_nonzero(doc_term_matrix) / total_ele)
print("Sparsity matrix: {:.2%}".format(sparsity))




#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
Sparsity matrix: 88.71%
```

1.1.7 1.1.7

Perform Latent Semantic Analysis for at least 5 different numbers of topics.

[58]:
```
#YOUR CODE STARTS HERE#
number_of_topics = 5
lsa_model = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word =␣
↪dictionary) # Train model















#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

1.1.8 1.1.8

For each of the calculations above, calculate a measure of the “goodness” of the division into topics.

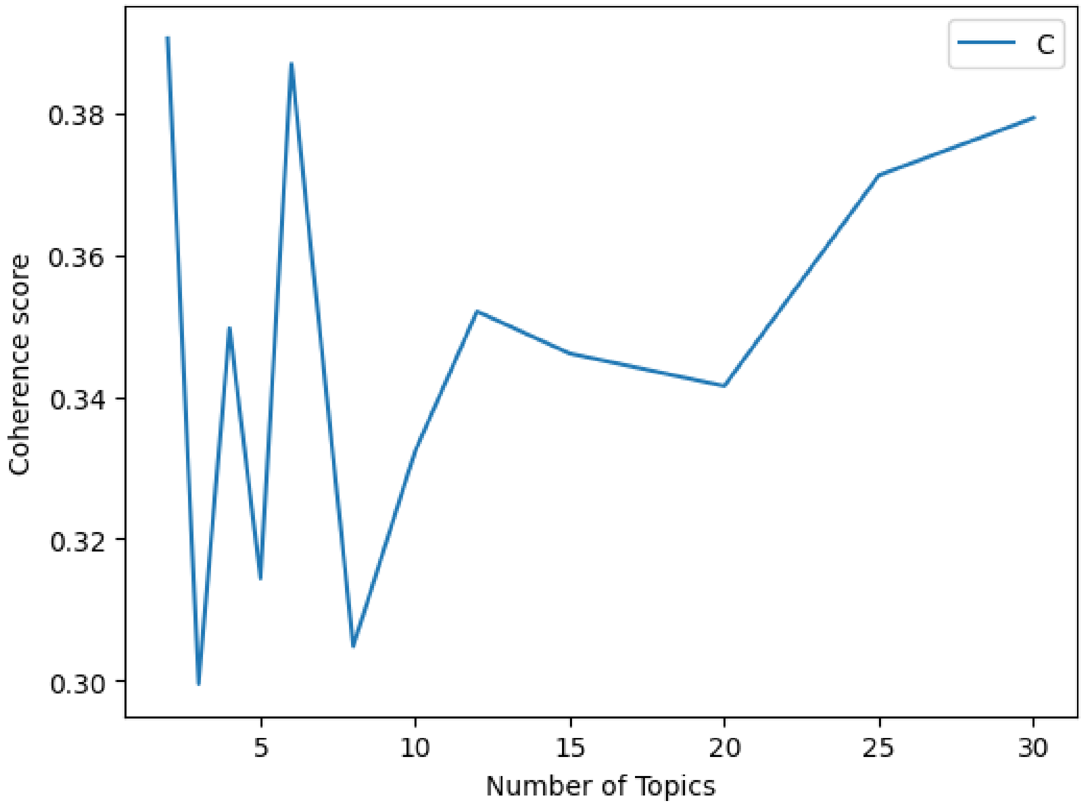
[59]:
```
#YOUR CODE STARTS HERE#
coherence_values = []

possible_numbers_of_topics = [2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 25, 30]

for number_of_topics in possible_numbers_of_topics:
  print("LSA for #topics:",number_of_topics)
  lsa_model = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word =␣
↪dictionary, random_seed = 160) # seed for reproducibility

  coherence_model = CoherenceModel(model=lsa_model, texts=clean_docs,␣
↪dictionary=dictionary, coherence='c_v')
  coherence_values.append(coherence_model.get_coherence())
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

```
LSA for #topics: 2
LSA for #topics: 3
LSA for #topics: 4
LSA for #topics: 5
LSA for #topics: 6
LSA for #topics: 8
LSA for #topics: 10
LSA for #topics: 12
LSA for #topics: 15
LSA for #topics: 20
LSA for #topics: 25
LSA for #topics: 30
```

[60]:

```
#YOUR CODE STARTS HERE#
plt.plot(possible_numbers_of_topics, coherence_values)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence score")
plt.legend(("Coherence values"), loc='best')
plt.show()
















#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```



———-YOUR TEXT STARTS HERE———-

As we can see after 10 topics till 30 we have an hight coherence score for every number of topics choosen: this suggests to us that the topics generated within this range are very similar to each other.

Therefore, it is reasonable to consider that the optimal number of topics can be choosen in a **range between 5 and 8**.

1.1.9 1.1.9

Print the 10 most important words for the 5 most important topics.

[61]:
```
#YOUR CODE STARTS HERE#
for topic_i,words_and_importance in lsa_model.print_topics(num_topics=5,␣
 ↪num_words=10):
  print("TOPIC:",topic_i)
  for app in words_and_importance.split(" + "):
    value,token = app.split("*")
    value = float(value)
    token = str(token.replace('"',""))
    print("\t",value,token)
  print()










#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

```
TOPIC: 0
         0.909 sir
         0.205 tristram
         0.199 launcelot
         0.138 king
         0.095 palomid
         0.09 knight
         0.074 arthur
         0.073 came
         0.062 gawain
         0.06 fought

TOPIC: 1
         0.78 king
         0.491 arthur
         -0.196 sir
         0.168 knight
         0.125 came
         0.123 mark
         0.082 made
         -0.053 tristram
         0.046 great
```

```
         0.044 merlin

TOPIC: 2
         0.528 knight
         -0.519 tristram
         0.468 launcelot
         -0.195 king
         0.149 queen
         -0.148 palomid
         0.137 came
         -0.123 isoud
         -0.091 beal
         -0.089 mark

TOPIC: 3
         0.674 knight
         0.431 tristram
         -0.378 launcelot
         0.179 fought
         0.136 ladi
         -0.116 king
         -0.106 arthur
         -0.102 sir
         0.093 slew
         0.082 two

TOPIC: 4
         0.599 launcelot
         0.483 tristram
         -0.247 came
         0.204 queen
         -0.2 sir
         -0.181 gawain
         0.165 made
         -0.137 met
         0.126 isoud
         -0.117 ladi
```

———-YOUR TEXT STARTS HERE———-

We selected the 5 most important topics using the method **print_topic(num_topics = n)** that return the most n significant topics.

———-YOUR TEXT STARTS HERE———-

Topics 1 empahasizes the focus on the king Arthur and his relationship with the knights. The other topics primarily revolves around the figure of the knights or some knight in particular (Sir Tristram and Sir Launcelot.).

1.2 Part 1.2

1.2.1 1.2.1

Suppose you have a dataset with N samples and M features.

You only have B units of memory available on your storage medium.

Assume further that each feature occupies a constant number b of memory units and that this cannot be changed (e.g. you cannot change the precision of floats).

Assuming that the entire dataset cannot fit on your storage medium, how would you accommodate all N samples while retaining as much information about your data as possible?

Use at most 3 sentences.

———-YOUR TEXT STARTS HERE———-

We can utilize dimensionality reduction techniques such as **Principal Component Analysis (PCA)** to project the data into a *lower-dimensional space* while retaining *as much information as possible*.

By applying PCA, the dimensionality of the dataset is reduced from the original number of features to the number of selected principal components.

The **number of components** depends on the available memory B and the memory required for one feature: they are selected based on the amount of variance the number of components explain in the original dataset.

2 Part 2

In this part, your goal is to obtain the best classification on a dataset according to a metric specified in each section.

[ ]:
```
#REMOVE_OUTPUT#
#YOUR CODE STARTS HERE#
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import time
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier

#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

2.1 Part 2.1

In this part, you will perform a tf-idf encoding of the data, and then train a classifier, optimising its hyper-parameters.

In the various steps, we will slowly prepare a pipeline to perform a hyper-parameter optimisation; try to prepare the required objects with this target in mind.

The goal is to maximise the accuracy on the test set.

2.1.1 2.1.1

Prepare the dataset for Supervised Learning.

> It should be a Pandas DataFrame with two fields: `Text`, `Label`.
>
> The `Text` column must contain the text of a chapter
>
> The `Label` column must contain a value of 0 or 1

- The `Label` is 0 if the chapter is in Book 1
- The `Label` is 1 if the chapter is in Book 2

[63]:
```
#YOUR CODE STARTS HERE#
d1 = {'Text' : vol1_texts, 'Label' : ['0' for _ in range(len(vol1_titles))]}
df_vol1 = pd.DataFrame(d1)
d2 = {'Text' : vol2_texts, 'Label' : ['1' for _ in range(len(vol2_titles))]}
df_vol2 = pd.DataFrame(d2)
df_book2 = pd.concat([df_vol1, df_vol2], ignore_index = True)
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

[64]:
```
#YOUR CODE STARTS HERE#
print("2 rows of the dataset with Label 0: ")
display(df_book2[df_book2.Label == '0'].head(2))
print("")
print("2 rows of the dataset with Label 1: ")
display(df_book2[df_book2.Label == '1'].head(2))









#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

```
2 rows of the dataset with Label 0:

                                                Text Label
0  It befell in the days of Uther Pendragon , whe…     0
1  Then Ulfius was glad , and rode on more than a…     0


2 rows of the dataset with Label 1:
```

	Text	Label
238	And if so be ye can descrive what ye bear , ye…	1
239	So Sir Tristram alighted off his horse because…	1

2.1.2 2.1.2

Divide the dataset into training (68%), validation (17%) and test set (15%).

[65]:
```python
#YOUR CODE STARTS HERE#
X = df_book2['Text']
y = df_book2['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15,
 ↪random_state = 160) # seed for reproducibility
# 0.85 represents the remaining 85% after the initial test split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size =
 ↪0.17/0.85, random_state = 160) # seed for reproducibility










#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

[66]:
```python
#YOUR CODE STARTS HERE#
print("The percentage of samples with negative labels in training set: ",
 ↪(list(y_train).count('0')/len(list(y_train)))*100, "%")
print("The percentage of samples with negative labels in validation set: ",
 ↪(list(y_val).count('0')/len(list(y_val)))*100, "%")
print("The percentage of samples with negative labels in test set: ",
 ↪(list(y_test).count('0')/len(list(y_test)))*100, "%")




#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
The percentage of samples with negative labels in training set:
48.97360703812317 %
The percentage of samples with negative labels in validation set:
47.674418604651166 %
The percentage of samples with negative labels in test set:  39.473684210526315
%
```

2.1.3 2.1.3

Create an object that performs a tf-idf transformation on the data. The transformation must **NOT** lowercase character names.

Create a dictionary containing configurations for the tf-idf vectorizer. Each hyper-parameter should have exactly **3 values**.

[67]:
```python
#YOUR CODE STARTS HERE#

# Initialize the vectorizer
vectorizer = TfidfVectorizer(lowercase = False) # the transformation must NOT␣
 ↪lowercase character names


# Define configurations for the tf-idf vectorizer
tfidf_configs = {
    'vect__max_features' : [None, 150, 300],  # maximum number of features to␣
 ↪include in the vocabulary
    'vect__ngram_range' : [(1, 1), (1, 2), (1, 3)],  # range of n-gram lengths␣
 ↪to consider
}
















#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

2.1.4 2.1.4

Choose a maximum of 2 classification algorithms (from those seen during the course) and prepare objects containing them.

For each of the selected classification algorithms, prepare a hyper-parameter configuration.

Each configuration must vary **at least 4 different hyper-parameters**.

If a parameter is itself composed of several parameters (if it is a dictionary, for example), each of these must vary at least 4 different hyper-parameters.

[68]:
```
#YOUR CODE STARTS HERE#

# Initialize the classification algorithm
rf_clf = RandomForestClassifier()
dt_clf = DecisionTreeClassifier()


# Define configurations for the classification algorithms
rf_configs = {
    'clf__n_estimators': [100, 150, 250],  # number of decision trees in the␣
 ↪forest
    'clf__max_depth': [5, 10, 15],  # maximum depth of the individual trees
    'clf__min_samples_split': [5, 10, 15],  # minimum number of samples␣
 ↪required to split an internal node
    'clf__min_samples_leaf': [5, 10]  # minimum number of samples required to␣
 ↪be at a leaf node

}

dt_configs = {
    'clf__criterion': ['gini', 'entropy', 'log_loss'],  # function to measure␣
 ↪the quality of a split
    'clf__max_depth' : [5, 10, 15],  # maximum depth of the tree
    'clf__min_samples_split': [5, 10, 15], # minimum number of samples required␣
 ↪to split an internal node
    'clf__min_samples_leaf': [5, 10]  # minimum number of samples required to␣
 ↪be at a leaf node
}






#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

2.1.5 2.1.5

For each of the classification algorithms selected in step 2.1.4, perform a 5-fold Cross-Validation on the validation set, combining the configurations of the vectorizer defined in step 2.1.3 and those of the classifier being used defined in step 2.1.4.

Perform the best hyper-parameter optimisation you can afford in **LESS than 15 minutes**.

If you are using two classifications algorithms, the maximum total optimisation time is **INSTEAD** 30 minutes.

[69]:

```
#YOUR CODE STARTS HERE#

# Start time
start_time = time.time()

# Initialize the pipelines
pipeline_rf = Pipeline([
        ('vect', vectorizer),
        ('clf', rf_clf)
        ])

pipeline_dt = Pipeline([
        ('vect', vectorizer),
        ('clf', dt_clf)
        ])

# Define the parameters to optimize for each classification algorithm
params_rf = {**tfidf_configs, **rf_configs}
params_dt = {**tfidf_configs, **dt_configs}


# Optimization with GridSearchCV on the validation set
grid_search_rf = GridSearchCV(pipeline_rf, params_rf,
                           scoring = metrics.make_scorer(metrics.
 ↪matthews_corrcoef),

 ↪            cv = 5, n_jobs = -1, verbose = 10)

grid_search_rf.fit(X_val, y_val)

grid_search_dt = GridSearchCV(pipeline_dt, params_dt,
                           scoring = metrics.make_scorer(metrics.
 ↪matthews_corrcoef),

 ↪            cv = 5, n_jobs = -1, verbose = 10)

grid_search_dt.fit(X_val, y_val)

# End time
end_time = time.time()


#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Fitting 5 folds for each of 486 candidates, totalling 2430 fits
Fitting 5 folds for each of 486 candidates, totalling 2430 fits
```

[70]:

```
#YOUR CODE STARTS HERE#
print("Total time taken: %s minutes" % ((end_time - start_time)/60))







#YOUR CODE ENDS HERE#
#THIS IS LINE 10#
```

```
Total time taken: 18.073807577292126 minutes
```

2.1.6 2.1.6

For each of the optimisations run in step 2.1.5:

Select the 5 best configurations and print them.

[75]:
```
#YOUR CODE STARTS HERE#

# Convert the results in a dataframe
df_params_rf = pd.DataFrame(grid_search_rf.cv_results_)
df_params_dt = pd.DataFrame(grid_search_dt.cv_results_)

# Select the 5 best configurations in terms of accuracy and print them
best_5_config_rf = df_params_rf.sort_values(by = 'mean_test_score', ascending =␣
 ↪False).head(5).reset_index()
best_5_config_dt = df_params_dt.sort_values(by = 'mean_test_score', ascending =␣
 ↪False).head(5).reset_index()

print("The 5 best configurations of Random Forest Classifier: \n ")
display(best_5_config_rf[['params', 'mean_test_score']])
print("")
print("")
print("The 5 best configurations of Decision Tree Classifier: \n ")
display(best_5_config_dt[['params', 'mean_test_score']])


#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

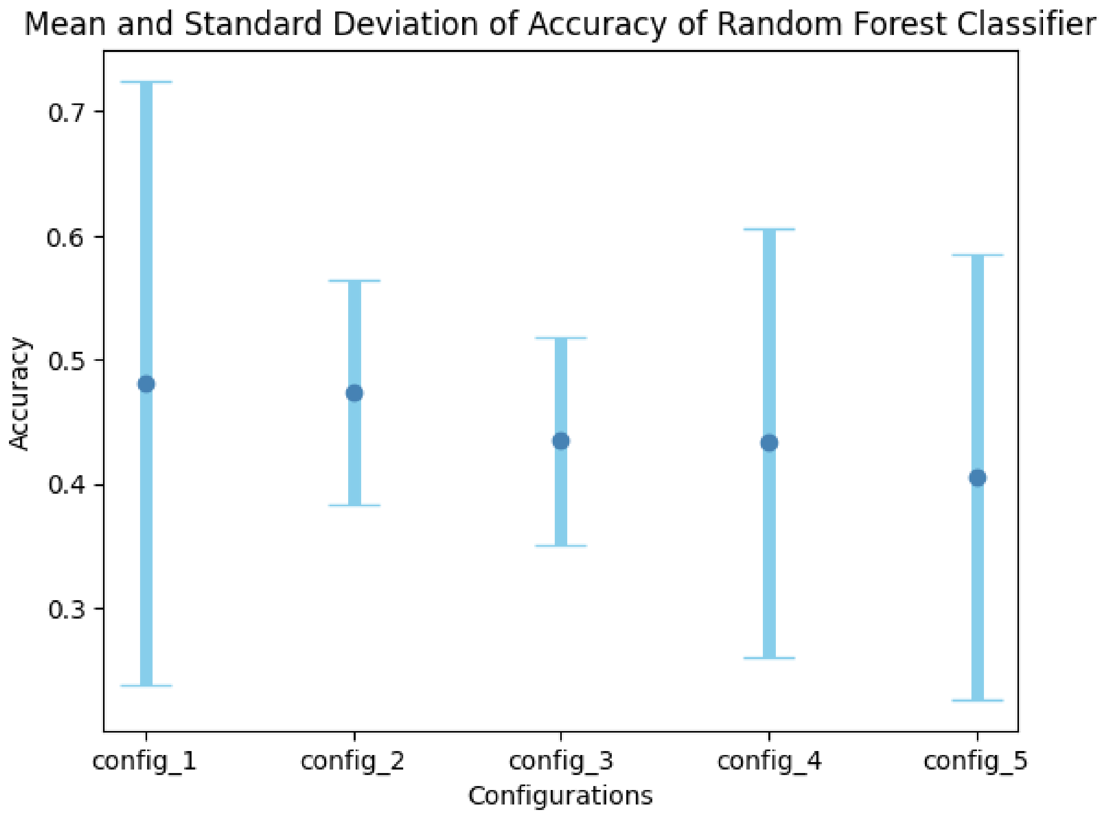
```
The 5 best configurations of Random Forest Classifier:


                                              params  mean_test_score
0  {'clf__max_depth': 5, 'clf__min_samples_leaf':…         0.480861
1  {'clf__max_depth': 15, 'clf__min_samples_leaf'…         0.473734
2  {'clf__max_depth': 10, 'clf__min_samples_leaf'…         0.434569
3  {'clf__max_depth': 5, 'clf__min_samples_leaf':…         0.432934
4  {'clf__max_depth': 5, 'clf__min_samples_leaf':…         0.405307



The 5 best configurations of Decision Tree Classifier:


                                              params  mean_test_score
0  {'clf__criterion': 'entropy', 'clf__max_depth'…         0.467220
1  {'clf__criterion': 'gini', 'clf__max_depth': 5…         0.437081
2  {'clf__criterion': 'gini', 'clf__max_depth': 5…         0.437081
3  {'clf__criterion': 'gini', 'clf__max_depth': 1…         0.429634
4  {'clf__criterion': 'log_loss', 'clf__max_depth…         0.419721
```

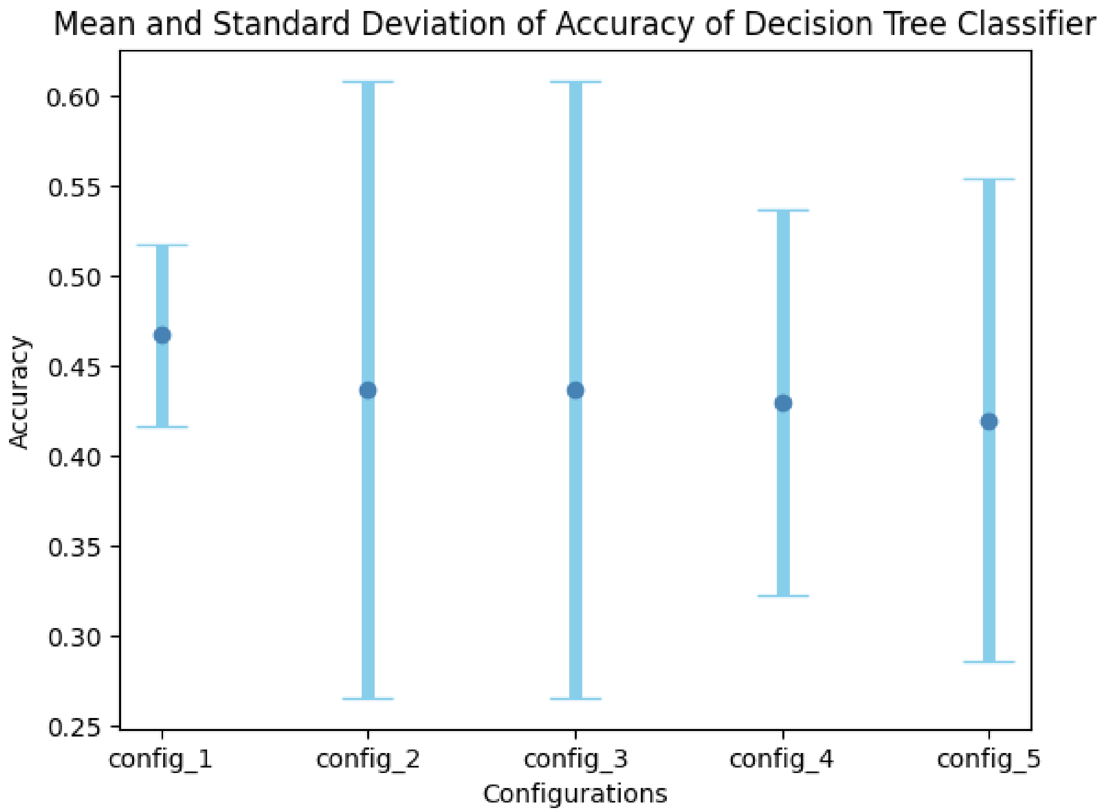
2.1.7 2.1.6

For each of the optimisations run in step 2.1.5:

> Produce a plot with mean and standard deviation of the accuracy calculated on the test set (**of each fold**) for the 5 configuration selected in step 2.1.6.

[76]:
```
#YOUR CODE STARTS HERE#
# Select mean_test_score of configurations
mean_scores = [best_5_config_rf['mean_test_score'],
 ↪best_5_config_dt['mean_test_score']]

# Select std_test_score of configurations
std_scores = [best_5_config_rf['std_test_score'],
 ↪best_5_config_dt['std_test_score']]

# List of configurations and algorithms
configurations = ['config_1', 'config_2', 'config_3', 'config_4', 'config_5']
algorithms = ['Random Forest Classifier', 'Decision Tree Classifier']

for i in range(2):
  plt.errorbar(configurations, mean_scores[i], yerr = std_scores[i], fmt = 'o',
 ↪color = 'steelblue',
            ecolor = 'skyblue', elinewidth = 5, capsize=10)
  plt.xlabel('Configurations')
  plt.ylabel('Accuracy')
  plt.title('Mean and Standard Deviation of Accuracy of %s' % (algorithms[i]))
  plt.show()
#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```


Mean and Standard Deviation of Accuracy of Random Forest Classifier
0.7
0.6
0.5
0.4
0.3
Accuracy
config_1
config_2
config_3
config_4
config_5
Configurations



———-YOUR TEXT STARTS HERE———-

In the graph, each dot represents the mean accuracy and the length of the bars indicates the variability of accuracy (standard deviation) for each configuration.

Since our goal is to **maximize accuracy** in the test phase, the best configuration is the one with the highest average accuracy, while also taking into account the standard deviation because a high standard deviation indicates unstable accuracy (by having better results, we may also encounter a greater number of unfavorable outcomes).

The configuration that satisfies these conditions is **configuration 2 of the Random Forest** classifier achieves higher accuracy with good stability, making it the chosen configuration for our model. The other options are not a good choice due to their high variability.

2.1.8 2.1.8

For each of the optimisations, obtain a classifier using the parameters you selected in step 2.1.6.

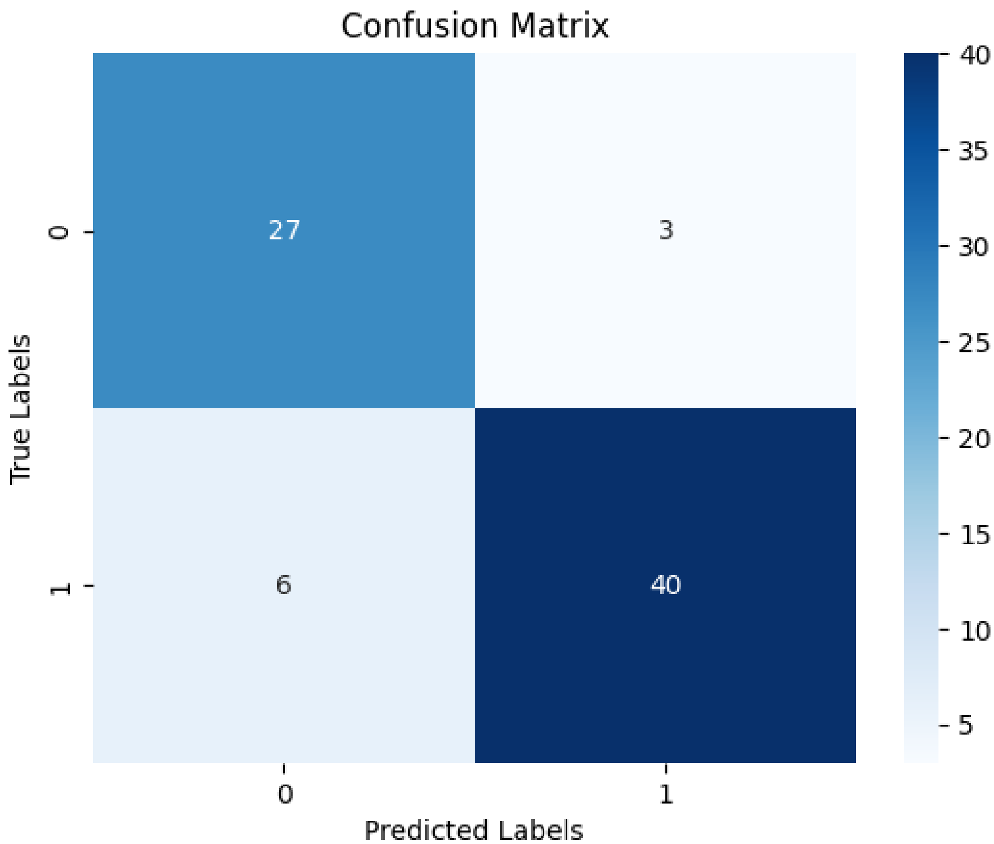
[77]:
```
#YOUR CODE STARTS HERE#

# Select best parameters
params = best_5_config_rf['params'].loc[1]  # configuration 2 of RF
print("Best parameters: \n")
for key, value in params.items():
    print(key, ':', value)

# Initialize the classifier with best parameters
pipeline = Pipeline([
        ('vect', TfidfVectorizer(lowercase = False)),
        ('clf', RandomForestClassifier(random_state = 160)), # seed for␣
 ↪reproducibility
        ])

pipeline.set_params(**params)

# Train and test the model
pipeline.fit(X_train, y_train)
pred_y_test = pipeline.predict(X_test)










#YOUR CODE ENDS HERE#
#THIS IS LINE 30#
```

```
Best parameters:

clf__max_depth : 15
clf__min_samples_leaf : 5
clf__min_samples_split : 10
clf__n_estimators : 250
vect__max_features : 300
vect__ngram_range : (1, 1)
```

[78]:
```
#YOUR CODE STARTS HERE#
print("Accuracy: ", metrics.accuracy_score(y_test, pred_y_test))
```

```
# Create the confusion matrix
cm = metrics.confusion_matrix(y_test, pred_y_test)
sns.heatmap(cm, annot = True, cmap = "Blues", fmt = "d")

# Plot the confusion matrix
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

#YOUR CODE ENDS HERE#
#THIS IS LINE 15#
```

```
Accuracy:  0.881578947368421
```



2.2 Part 2.2

2.2.1 2.2.1

You have a training set containing N documents. There are M_1 unique terms within the dataset.

The test dataset will have M_2 unique terms within it. However, we know that only a small amount of these will be in common with the training dataset.

What precautions could we use to preprocess the data?

What could we change at test time and which of the classification algorithms seen in class would best suit the change?

Use at most 4 sentences.

———-YOUR TEXT STARTS HERE———-

During the pre-processing, we have to ensure that M_2 (the unique terms found in the test dataset) match M_1 (the unique terms found in the training dataset) in order to prevent **inconsistencies** caused by encountering unseen terms during the test time.

To archive it, we can configure the **'vocabulary' parameter of the Tf-Idf Vectorizer to match** M_1: in this way, in the feature extraction process we will ignore any additional terms.

During the test time, we have to apply the same pre-processing steps employed during the training phase because any change can bring to **anomalies in the prediction outcomes**.

Anyway, one classification algorithm that can handle differences between the training and test datasets is **Naive Bayes** due to its *feature independence assumption*: in fact, this classifier manage to generalize well also unseen features in the test dataset.