Personal Information

Name: **Ivona Birlad**

StudentID: **14020939**

Email: **ivona.birlad@student.uva.nl**

Submitted on: **19.03.2023**

GitHub: https://github.com/IvonaBirlad/TheisEDA.

Data Context

This research project will attempt to improve the results of the best performing models from Task 9 of the SemEval 2021 competition: Statement Verification and Evidence Finding with Tables (SEM-TAB-FACT), as well as extend the application of this model by exploring methods of improving performance for large tables. The task is divided into two sub-tasks: A) Table Statement Support - the identification of whether the table supports the given statement or not and B) Relevant Cell Selection - the identification of the cells in the table which provide evidence for the statement. Additionally, further experiments may be conducted to improve the performance of the model on large tables.

The data that will be used for training was made publicly available by the organizers of the SemEval Task 9 competition. Made up of 981 manually-annotated tables and 1980 automatically-generated statements for tables, the SEM-TAB-FACTS dataset contains tables scraped from scientific articles. The auto-generated table-statement pairing is prone to sounding less natural but, similar to the manually-annotated scientific tables, they are largely artifact-free.

For evaluation, the original train-development-test split will be kept. SEM-TAB-FACTS' training, validation, and test sets respectively contain 4506, 423, and 522 sentence-table pairs and are published in XML format. Under the XML format, every document consists of one or more table elements. A table element is made up of rows and statements. The rows describe the structure of the table, including the columns, and they contain the text found in each cell of the table. Additionally, some tables may also have a legend or caption to provide context for a specific section of the table.

For the external validation task, a new dataset will be used: TabFact which is made up of 117,854 manually annotated statements with regard to 16,573 Wikipedia tables. Split similarly to SEM-TAB-FACTS, TabFact's training, validation, and test sets respectively contain 92283, 12792, and 12779 sentence-table pairs.

Data Description

The dataset splits I will be working with have the following dimensions:

- main training set with Unknown label added has the shape (9012, 9)
- manually annotated training set has the shape (4506, 9)
- automatically annotated training set has the shape (179345, 9)
- development set has the shape (556, 9)
- test set task a has the shape (653, 9)
- test set task b has the shape (470, 9)

I first analyzed the entire corpus of statements and the respective labels indicating whether the statement is entailed(31%), refuted(19%), or unknown(50%). I explore the number of table-statement pairs (with an average of 9.187 statements per table) and the distribution per label. I also calculated baselines for the classification task. Since my overall goal is to develop a classifier that can correctly classify the statements as entailed, refuted, or unknown, I must outperform the following baselines:

- Random Classifier Baseline 50%
- Zero Rate Classifier Baseline 38.3%

Next I check the size of vocabulary and term frequencies for both tables and statements in the training set (here the main training set derived from the manual annotations is used). After joining all sentences in one document and all tables in another document, I tokenize and use a CountVectorizer to create a Term-Document matrix to check the size of the vocabulary.

The table corpus can be described by:

- Size of vocabulary: 71792
- Size of corpus : 981
- Average length of text in table: 443.168

For tables, the 10 most common unigrams are: ('model', 213), ('number', 186), ('time', 165), ('mean', 158), ('type', 149), ('total', 148), ('value', 139), ('year', 133), ('data', 116), ('size', 108).

The table statements can be described by:

- Size of vocabulary: 24309
- Size of corpus : 8640
- Average length of sentence: 66.259

For statements, the 10 most common unigrams are: ('model', 213), ('number', 186), ('min', 169), ('time', 165), ('yes', 160), ('mean', 158), ('type', 149), ('total', 148), ('value', 139), ('year', 133).

These results are expected since my corpus is made up of tables extracted from scientific articles.

Finally, I look more in depth at specific tables in the main training set "train_3way_df" and compare them to associated statements. To visually compare the contents of the tables to the associated statements I plot three wordclouds showing that, as expected, their content is similar for the most part, while some words seem unrelated, potentially due to the Unknown label taht represents 50% of the statement corpus.

In [261…
```
# !pip install wordcloud

import os
import numpy as np
import seaborn as sns
import pandas as pd
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer

import matplotlib.pyplot as plt
from wordcloud import WordCloud
%matplotlib inline

from utils import get_parent_directory_path
```

Data Loading

In [ ]:
```
CSV_FOLDER = get_parent_directory_path(1) + os.sep + "csv"
TSV_FOLDER =  get_parent_directory_path(1) + os.sep + "tsv"

train_3way_df = pd.read_csv(os.path.normpath(TSV_FOLDER + os.sep + 'train_3way_set.
                            sep='\t', header=0)

train_man_df = pd.read_csv(os.path.normpath(TSV_FOLDER + os.sep + 'train_man_set.ts
                            sep='\t', header=0)
train_auto_df = pd.read_csv(os.path.normpath(TSV_FOLDER + os.sep + 'train_auto_set.
                             sep='\t', header=0)
dev_df = pd.read_csv(os.path.normpath(TSV_FOLDER + os.sep + 'dev_set.tsv'),
                     sep='\t', header=0)
test_a_df = pd.read_csv(os.path.normpath(TSV_FOLDER + os.sep + 'test_a_set.tsv'),
                        sep='\t', header=0)
test_b_df = pd.read_csv(os.path.normpath(TSV_FOLDER + os.sep + 'test_b_cell.tsv'),
                        sep='\t', header=0)
```

Corpus analysis:

Make sure to add some explanation of what you are doing in your code. This will help you and whoever will read this a lot in following your steps.

In [139…
```
dfs = [train_3way_df, train_man_df,train_auto_df,dev_df,test_a_df,test_b_df]
df_name = ["main training set with Unknown label added", "manually annotated traini
```

```
for pos, df in enumerate(dfs):
    df.name = df_name[pos]
    print(f"{df.name} has the shape {df.shape}")
```

```
main training set with Unknown label added has the shape (9012, 9)
manually annotated training set has the shape (4506, 9)
automatically annotated training set has the shape (179345, 9)
development set has the shape (556, 9)
test set task a has the shape (653, 9)
test set task b has the shape (470, 9)
```

I will analyse the labeled main training set (where the 'Unknown' label was added by randomly selecting statements associated with other tables) aggregated so that each statement matches its corresponding table file name. All other sets are similarly structured.

- question is the statement
- table_file gives the file name of the corresponding table
- answer_text is the label with 0 being refuted, 1 entailed and 2 unknown

Note: I have added features annotator, position, answer_coordinates, aggreagtaion, float answer to help with Task B of evidence finding.

In [141…

```
train_3way_df.head()
```

Out[141]:

	id	annotator	position	question	table_file	answer_coordinates	answer_text	aggregat
0	1	0	0	At the same time, these networks often occur i...	train_man_1.csv	[]	1	N
1	1	0	0	For each network interaction, there is conside...	train_man_1.csv	[]	1	N
2	1	0	0	The n value is same for Hong Kong and Malaysia.	train_man_1.csv	[]	0	N
3	1	0	0	There are 9 different types country in the giv...	train_man_1.csv	[]	1	N
4	2	0	0	All the Publication titles are unique.	train_man_2.csv	[]	1	N

Since the goal of the project is to correctly classify statements as entailed, refuted (by the table) or unknown, looking at the amount of table-statement pairs may be interesting.

In [209…

```
table_question_pairs = train_3way_df.groupby('table_file')["question"].agg("count")
print(f"Average number of statements per table {table_question_pairs.mean():.3f}")
print("Other summary statistics:")
table_question_pairs.describe()
```

```
Average number of statements per table 9.187
Other summary statistics:
```

Out[209]:

```
count    981.000000
mean       9.186544
std        2.974871
min        1.000000
25%        7.000000
50%        9.000000
75%       11.000000
max       18.000000
Name: question, dtype: float64
```

Similarly, I observe the distribution per label:

In [171…

```
label_question_pairs = train_3way_df.groupby('answer_text')["question"].agg("count"
label_question_pairs["answer_text"].replace([0, 1, 2], ["Refuted", "Entailed", "Unk
```

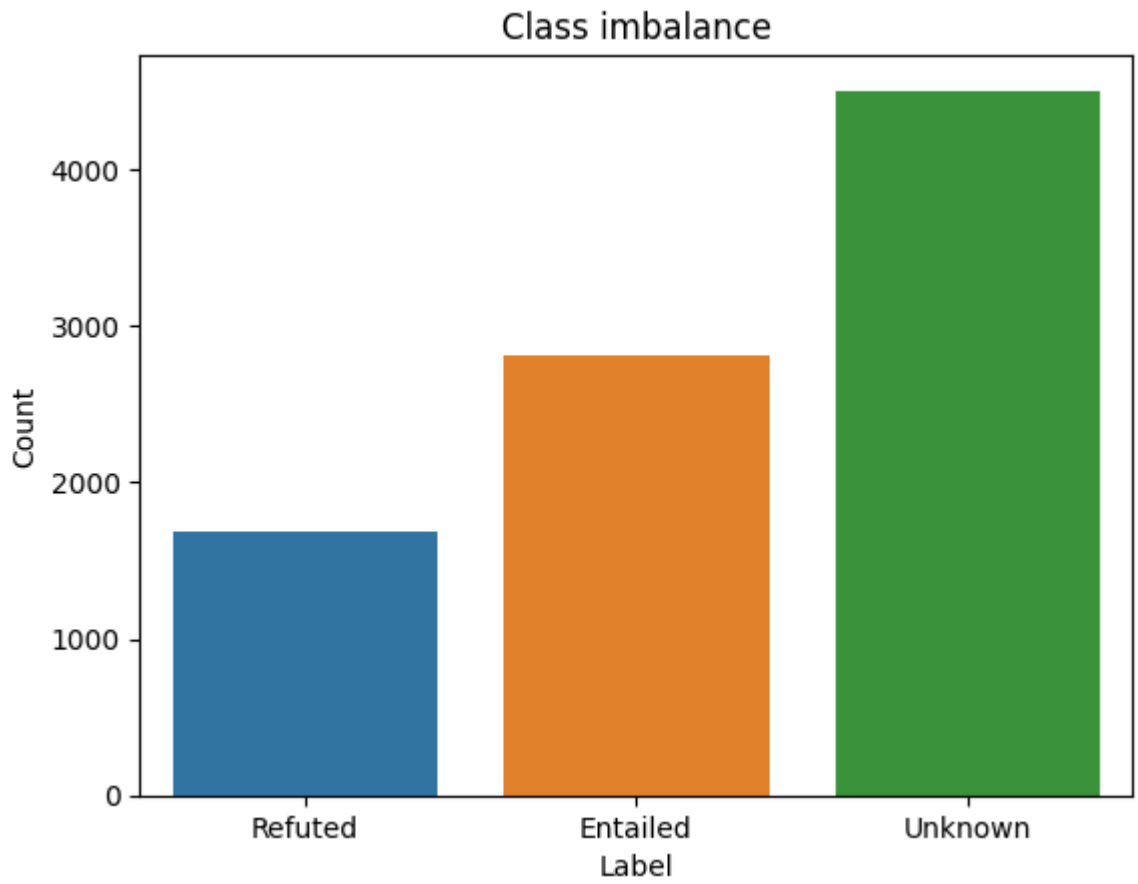
```
label_question_pairs
```

Out[171]:

	answer_text	question
0	Refuted	1688
1	Entailed	2818
2	Unknown	4506

In [265…
```
ax = sns.barplot(label_question_pairs, x= "answer_text", y= "question")
ax.set_title("Class imbalance")
ax.set_ylabel("Count")
ax.set_xlabel("Label")
```

Out[265]:
```
Text(0.5, 0, 'Label')
```



Baseline

ZeroR (or Zero Rate) Classifier Baseline- predicts only majority class, in this case Unknown class - 0.5%

Also beat random guess baseline: random classifier accuracy would be $Refuted^2 + Entailed^2 + Unknown^2$

In [233…
```
random = label_question_pairs["question"].max()/label_question_pairs["question"].su
zeror = np.power(label_question_pairs["question"]/label_question_pairs["question"].
```

```
print(f"Random Classifier Baseline {random}\nZero Rate Classifier Baseline {zeror:.
```

```
Random Classifier Baseline 0.5
Zero Rate Classifier Baseline 0.383
```

My classifier must beat the performance of both of these baselines for task A.

Analysing statements

Checking size of vocabulary, most common words

Vocabulary size for the tables in train_man

In [253…
```
all_statements = ' '.join([x.strip() for x in train_3way_df.question.tolist()])
statements_tokenized = sent_tokenize(all_statements)
```

In [255…
```
#Vocabulary size for the statements

count = CountVectorizer(
    stop_words='english',
    ngram_range=(1, 2), #consider 1grams and 2grams
)

term_doc = count.fit_transform(statements_tokenized)

vocab = count.get_feature_names_out()
print(f'Size of vocabulary: {len(vocab)}')
print(f'Size of corpus    : {len(statements_tokenized)}')
print(f'Average length of sentence: {pd.Series(statements_tokenized).apply(len).mea
print(f'Shape of Term-Document Matrix: {term_doc.shape} - {term_doc.shape[0]} Rows
```

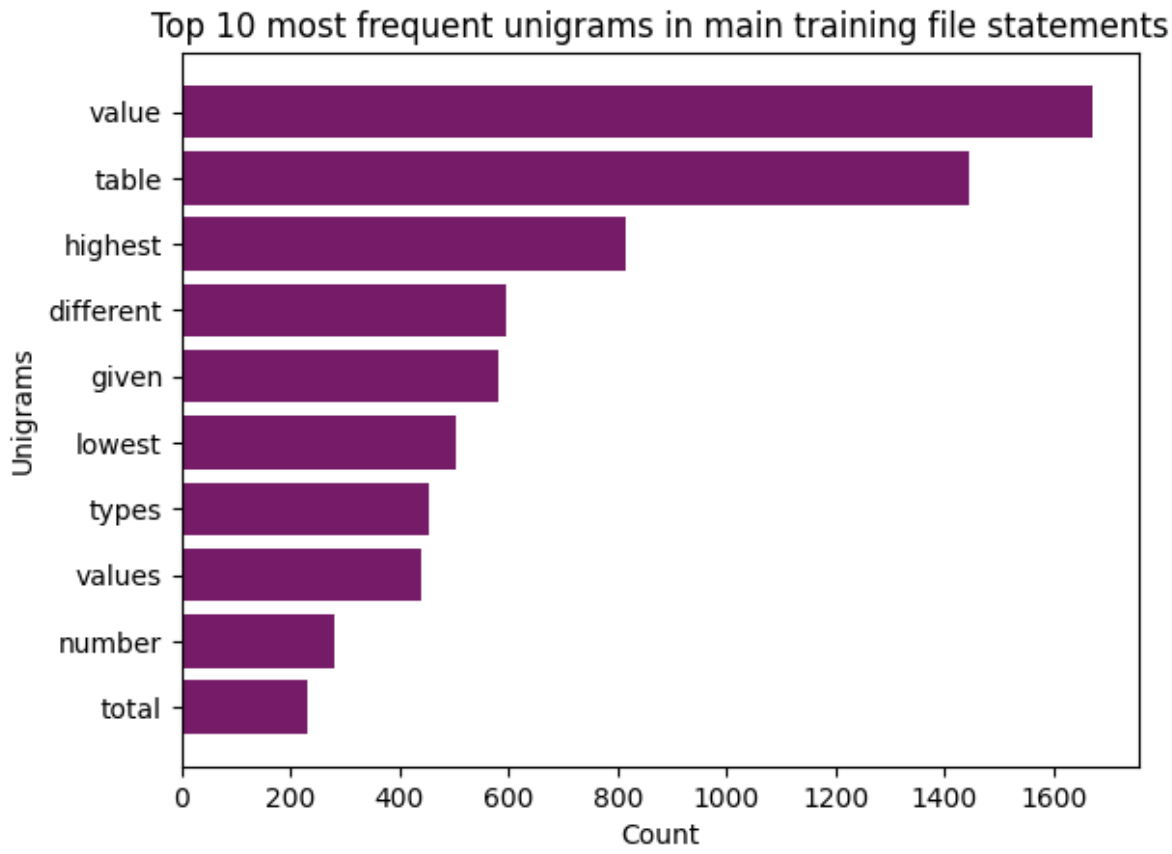
```
Size of vocabulary: 24309
Size of corpus    : 8640
Average length of sentence: 66.259
Shape of Term-Document Matrix: (8640, 24309) - 8640 Rows x 24309 Columns
```

In [400…
```
#helper functions
def get_common_words(df, n_gram_length):
    '''
    takes dataframe and n_gram_length as input
    returns 10 most common n_grams in dataframe
    '''
    if type(df) == list:
        text = df
    else:
        text = df["question"]
    stop = stopwords.words('english') + ["yes", "min"]

    vect = CountVectorizer(
      lowercase = True,
      stop_words= stop,
      ngram_range = (n_gram_length, n_gram_length),
      token_pattern = r"[a-z]{3,}",
      min_df=5,
      max_df=0.8,
```

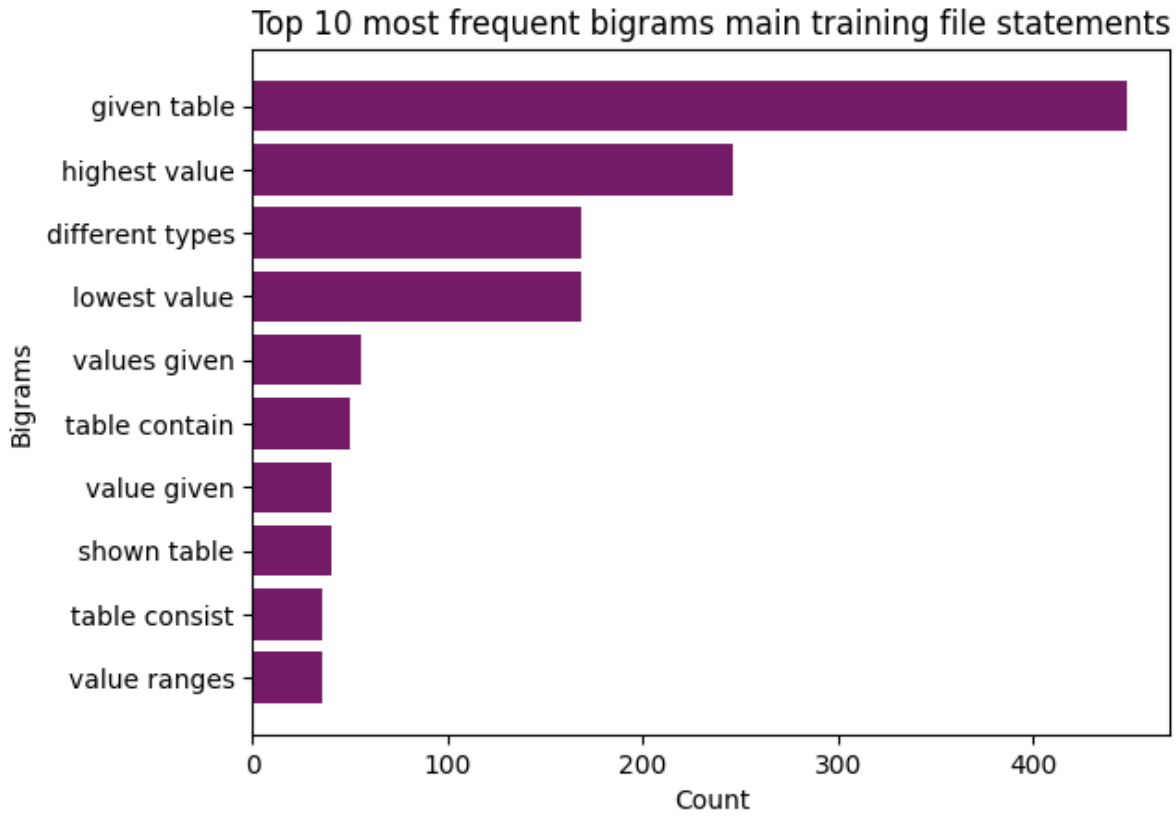
```
    )

    bow = vect.fit_transform(text)

    sum_words = bow.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vect.vocabulary_.items

    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    top_10 = words_freq[:10]

    return top_10

################################################################################

def plot_top(ngram,title, x, y):
    '''
    creates a simple barplot to display words and their frequency
    '''
    ngram = dict(ngram)
    plt.barh(range(len(ngram)), ngram.values(), color = "#761b68")
    plt.yticks(range(len(ngram)), ngram.keys())
    plt.title(title)
    plt.xlabel(x)
    plt.ylabel(y)
    plt.gca().invert_yaxis()
```

In [393…
```
unigrams = get_common_words(train_3way_df,1)
bigrams = get_common_words(train_3way_df,2)
```

In [257…
```
plot_top(unigrams, "Top 10 most frequent unigrams in main training file statements"
```



In [258… `plot_top(bigrams, "Top 10 most frequent bigrams main training file statements", "Co`



Analysing tables

Vocabulary size, term-frequencies

In [386…
```
all_tables = []
for file_name in np.unique(train_3way_df["table_file"]):
    tables = pd.read_csv(os.path.normpath(CSV_FOLDER + os.sep + file_name), header=
    flatten_df = tables.to_numpy().flatten()
    flat_table = ' '.join([x.strip() for x in flatten_df.tolist() if isinstance(x,
    all_tables.append(flat_table)
```

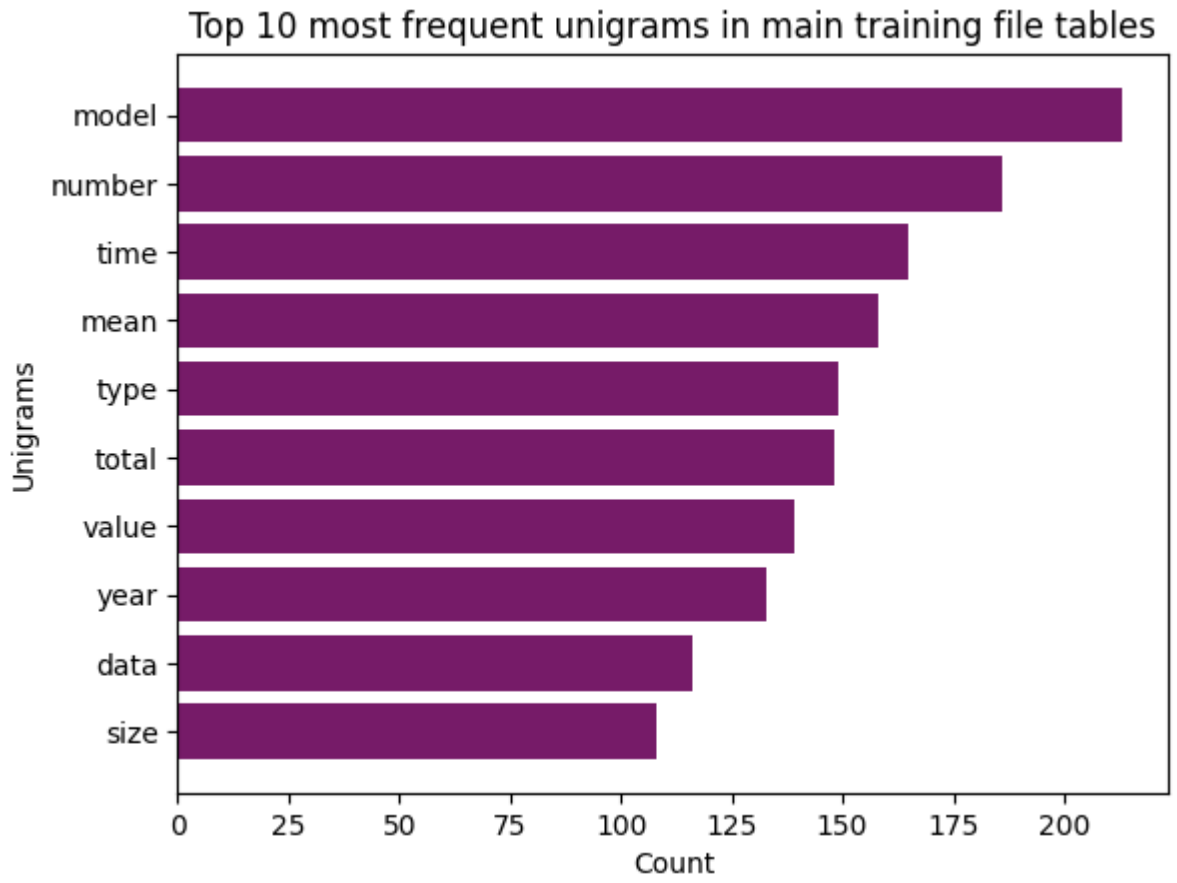
In [ ]:
```
all_tables_strip = ' '.join([x.strip() for x in all_tables])
tables_tokenized = sent_tokenize(all_tables_strip)
```

In [410…
```
#Vocabulary size for the statements

count = CountVectorizer(
    stop_words='english',
    ngram_range=(1, 2),  #consider 1grams and 2grams
)

term_doc = count.fit_transform(tables_tokenized)

vocab = count.get_feature_names_out()
print(f'Size of vocabulary: {len(vocab)}')
print(f'Size of corpus    : {len(all_tables)}')
print(f'Average length of text in table: {pd.Series(tables_tokenized).apply(len).me

print(f'Shape of Term-Document Matrix: {term_doc.shape} - {term_doc.shape[0]} Rows
```

```
Size of vocabulary: 71792
Size of corpus    : 981
Average length of text in table: 443.168
Shape of Term-Document Matrix: (1792, 71792) - 1792 Rows x 71792 Columns
```

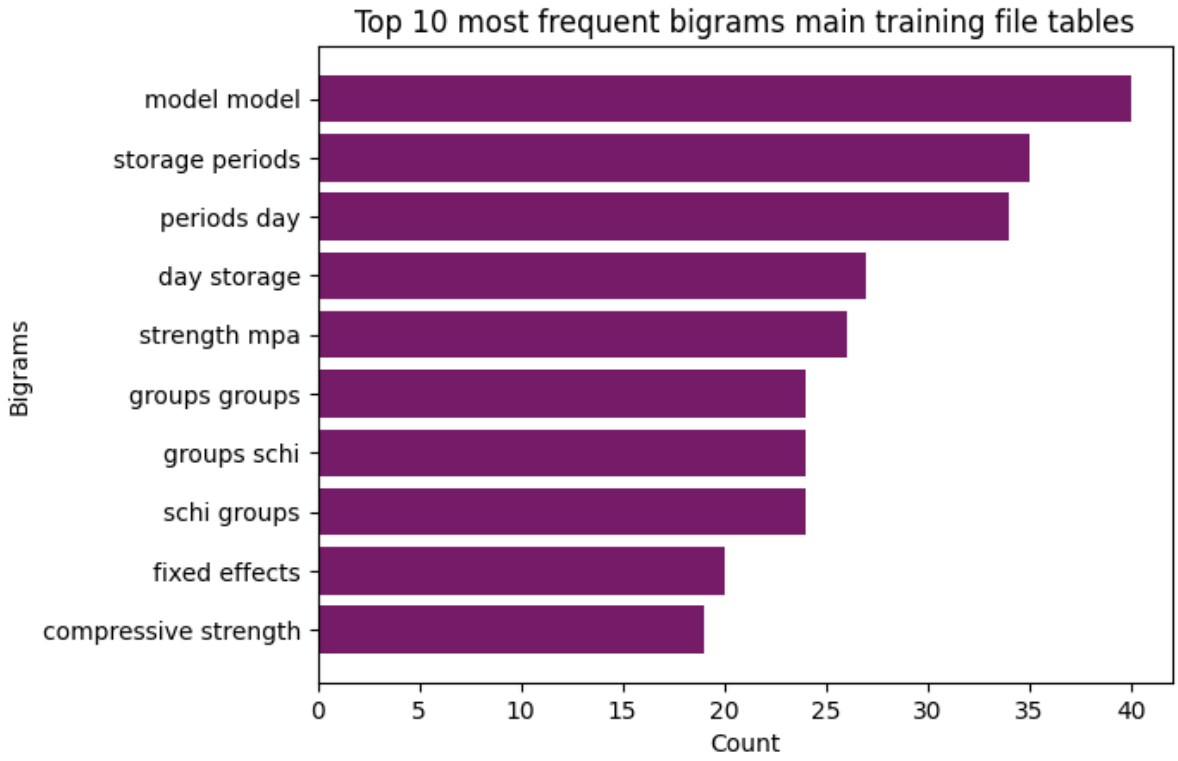
In [401…
```
unigrams_table = get_common_words(all_tables, 1)
bigrams_table = get_common_words(all_tables, 2)
```

In [406…
```
plot_top(unigrams_table, "Top 10 most frequent unigrams in main training file table
```



In [407…

```
plot_top(bigrams_table, "Top 10 most frequent bigrams main training file tables", "
```



More in-depth look at specific tables

Selecting a few tables and exploring their state and contents compared to associated statements

In [372…
```
for pos, file_name in enumerate(np.unique(train_3way_df["table_file"])[:3]):
    globals()['table%s' % pos] = pd.read_csv(os.path.normpath(CSV_FOLDER + os.sep +
```

In [373…
```
#Preview table
table1.head()
```

Out[373]:

	0	1	2	3	4
0	Communes grouped by the share of farms who mad...	Number of communes*	Amount of investments co-financed by the EU (E...	Amount of investments co-financed by the EU (E...	Share of farms who made investments co-finance...
1	0–2.25%	437	159	720	1.0
2	2.26–6.01%	433	418	3124	4.2
3	6.02–10.26%	435	617	6470	8.1
4	10.27–16.67%	435	840	11,051	13.2

In [374…
```
#Preview statements
filter1 = train_3way_df[train_3way_df["table_file"]=="train_man_1.csv"] #get all st
filter1
```

Out[374]:

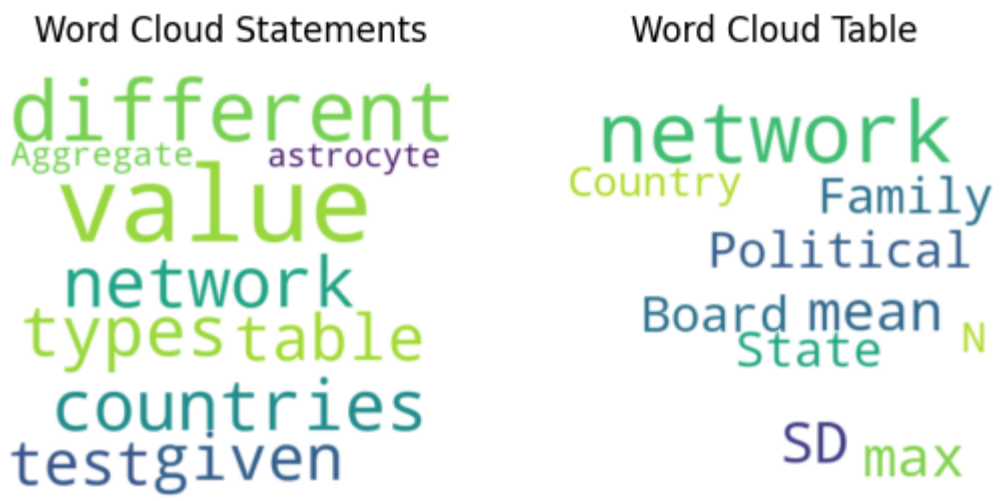
	id	annotator	position	question	table_file	answer_coordinates	answer_text	agg
0	1	0	0	At the same time, these networks often occur i...	train_man_1.csv	[]	1	
1	1	0	0	For each network interaction, there is conside...	train_man_1.csv	[]	1	
2	1	0	0	The n value is same for Hong Kong and Malaysia.	train_man_1.csv	[]	0	
3	1	0	0	There are 9 different types country in the giv...	train_man_1.csv	[]	1	
6622	1	0	0	The test Aggregate Crushing value and Aggregat...	train_man_1.csv	[]	2	
7300	1	0	0	Astrocytes NVU components has Excessively secr...	train_man_1.csv	[]	2	
7490	1	0	0	Again, the test set up having 10 mm screw and ...	train_man_1.csv	[]	2	
7988	1	0	0	Range x>60 x>110 is Picking from low location ...	train_man_1.csv	[]	2	
8326	1	0	0	They have 7 different types of countries in th...	train_man_1.csv	[]	2	
8366	1	0	0	Parametric coefficients: (Intercept) Estimate ...	train_man_1.csv	[]	2	

In [375…
```
def make_wordclouds(csv_file_name, table_name):
    filter_stmt = train_3way_df[train_3way_df["table_file"]==csv_file_name]
    statements_assoc = ' '.join([x.strip() for x in filter_stmt.question.tolist()])
    flatten_df = table_name.to_numpy().flatten()
    flat_table = ' '.join([x.strip() for x in flatten_df.tolist() if isinstance(x,
    # Creating word_cloud with statements/table text
    word_cloud_statements = WordCloud(background_color='white',
                    width=400,
                    height=400,
                    max_words=10,
                    prefer_horizontal=1.0)
    word_cloud_statements.generate_from_text(statements_assoc)
    word_cloud_table = WordCloud(background_color='white',
                    width=400,
                    height=400,
                    max_words=10,
                    prefer_horizontal=1.0)
    word_cloud_table.generate_from_text(flat_table)
    #Display the generated Word Cloud
    plt.subplot(1, 2, 1)
    plt.imshow(word_cloud_statements, interpolation='bilinear')
    plt.axis("off")
    plt.title("Word Cloud Statements")
    plt.subplot(1, 2, 2)
    plt.imshow(word_cloud_table, interpolation='bilinear')
    plt.title("Word Cloud Table")
    plt.axis("off")
    plt.show()
```

In [379…
```
np.unique(train_3way_df["table_file"])[:3] #check which tables were read
```

Out[379]:
```
array(['train_man_1.csv', 'train_man_10.csv', 'train_man_100.csv'],
      dtype=object)
```

In [376…
```
make_wordclouds("train_man_1.csv", table0)
```



In [377…
```
make_wordclouds("train_man_10.csv", table1)
```

In [378…
```
make_wordclouds("train_man_100.csv", table2)
```