Appendix A - Tables

Appendix A1 - Respective chart: B1

Multinomial NB - Labels: 5 - Number of minimum and maximum documents: 400 - Features: 3000

Sample measure: Cross Validation score - Each column from #1 to #10 is the averaged result of N measures

Number of measures	AVG #1	AVG #2	AVG #3	AVG #4	AVG #5	AVG #6	AVG #7	AVG #8	AVG #9	AVG #10	Variance (MAX	Average	Error % (Variance / Average * 100)
1	0.576	0.57	0.552	0.562	0.539	0.572	0.588	0.599	0.501	0.522	0.098	0.5581	17.56
5	0.581	0.591	0.549	0.535	0.563	0.56	0.536	0.547	0.542	0.521	0.070	0.5525	12.67
10	0.536	0.558	0.563	0.57	0.558	0.524	0.57	0.51	0.533	0.546	0.060	0.5468	10.97
25	0.529	0.553	0.55	0.564	0.558	0.518	0.557	0.558	0.557	0.547	0.046	0.5491	8.38
50	0.53	0.552	0.551	0.55	0.533	0.54	0.542	0.534	0.55	0.536	0.022	0.5418	4.06
100	0.563	0.562	0.57	0.567	0.569	0.562	0.56	0.568	0.564	0.562	0.010	0.5647	1.77

Appendix A2 - Respective chart: B2

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val variance (+/-)	Recall	Precision	F1-score	Testing time	Vocabulary	Average Score	Features available	selected features / features (%)
Multinomial NB	5	400	400	1,000	0.441	0.068	0.563	0.756	0.639	38 minutes	~ 440,000	0.600	12,305	8.13
Multinomial NB	5	400	400	2,500	0.571	0.068	0.804	0.769	0.781	40 minutes	~ 440,000	0.731	12,305	20.32
Multinomial NB	5	400	400	3,500	0.550	0.057	0.860	0.735	0.780	40 minutes	~ 440,000	0.731	12,305	28.44
Multinomial NB	5	400	400	5,000	0.605	0.058	0.853	0.767	0.804	50 minutes	~ 440,000	0.757	12,305	40.63
Multinomial NB	5	400	400	7,500	0.594	0.069	0.901	0.754	0.815	50 minutes	~ 440,000	0.766	12,305	60.95
Multinomial NB	5	400	400	10,000	0.582	0.070	0.926	0.749	0.821	0.9 hours	~ 440,000	0.770	12,305	81.27
Multinomial NB	5	400	400	12,305	0.574	0.064	0.924	0.740	0.814	1.1 hours	~ 440,000	0.763	12,305	100.00

Appendix A3 - Respective chart: B3

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val variance (+/-)	Recall	Precision	F1-score	Testing time	Vocabulary	Average Score	Features available	selected features / features (%)
Multinomial NB	5	400	/	1,000	0.469	0.037	0.694	0.701	0.676	1.20 hours	~ 1,280,000	0.635	18,616	5.37
Multinomial NB	5	400	/	2,500	0.539	0.033	0.795	0.734	0.759	1.40 hours	~ 1,280,000	0.707	18,616	13.43
Multinomial NB	5	400	/	3,500	0.544	0.038	0.801	0.737	0.724	1.5 hours	~ 1,280,000	0.702	18,616	18.80
Multinomial NB	5	400	/	7,500	0.583	0.036	0.908	0.744	0.816	1.9 hours	~ 1,280,000	0.763	18,616	40.29
Multinomial NB	5	400	/	10,000	0.593	0.035	0.923	0.750	0.825	3 hours	~ 1,280,000	0.773	18,616	53.72
Multinomial NB	5	400	/	18,616	0.590	0.034	0.937	0.748	0.830	3.6 hours	~ 1,280,000	0.776	18,616	100.00

Appendix A4 - Respective chart: B4

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val variance (+/-)	Recall	Precision	F1-score	Testing time	Vocabulary	Average Score	Features available	selected features / features (%)
Bernoulli NB	5	400	400	1,000	0.384	0.068	0.691	0.664	0.663	38 minutes	~ 440,000	0.601	12,305	8.13
Bernoulli NB	5	400	400	2,500	0.412	0.065	0.828	0.677	0.727	40 minutes	~ 440,000	0.661	12,305	20.32
Bernoulli NB	5	400	400	3,500	0.392	0.060	0.841	0.649	0.720	40 minutes	~ 440,000	0.651	12,305	28.44
Bernoulli NB	5	400	400	5,000	0.404	0.061	0.850	0.662	0.724	50 minutes	~ 440,000	0.660	12,305	40.63
Bernoulli NB	5	400	400	7,500	0.406	0.065	0.872	0.661	0.732	50 minutes	~ 440,000	0.668	12,305	60.95
Bernoulli NB	5	400	400	10,000	0.410	0.070	0.879	0.656	0.732	0.8 hours	~ 440,000	0.669	12,305	81.27
Bernoulli NB	5	400	400	12,305	0.404	0.065	0.876	0.651	0.729	1.1 hours	~ 440,000	0.665	12,305	100.00

Appendix A5 - Respective chart: B5

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val variance (+/-)	Recall	Precision	F1-score	Testing time	Vocabulary	Average Score	Features available	selected features / features (%)
Bernoulli NB	5	400	/	1,000	0.382	0.034	0.780	0.655	0.701	1.20 hours	~ 1,280,000	0.630	18,616	5.37
Bernoulli NB	5	400	/	2,500	0.427	0.035	0.817	0.662	0.728	1.40 hours	~ 1,280,000	0.659	18,616	13.43
Bernoulli NB	5	400	/	3,500	0.394	0.034	0.827	0.649	0.765	1.5 hours	~ 1,280,000	0.659	18,616	18.80
Bernoulli NB	5	400	/	7,500	0.438	0.036	0.878	0.667	0.755	1.9 hours	~ 1,280,000	0.685	18,616	40.29
Bernoulli NB	5	400	/	10,000	0.396	0.040	0.888	0.654	0.748	3 hours	~ 1,280,000	0.672	18,616	53.72
Bernoulli NB	5	400	/	18,616	0.415	0.040	0.893	0.661	0.893	3.6 hours	~ 1,280,000	0.716	18,616	100.00

Appendix A6 - Respective chart: B6

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Multinomial NB	10	200	200	1,000	0.245	0.056	0.232	0.577	0.306	40 minutes	~ 700,000	0.340	19,848	5.04
Multinomial NB	10	200	200	2,500	0.455	0.063	0.467	0.741	0.550	50 minutes	~ 700,000	0.553	19,848	12.60
Multinomial NB	10	200	200	3,500	0.473	0.060	0.522	0.761	0.597	1 hour	~ 700,000	0.588	19,848	17.63
Multinomial NB	10	200	200	7,500	0.542	0.068	0.505	0.822	0.595	1.2 hours	~ 700,000	0.616	19,848	37.79
Multinomial NB	10	200	200	10,000	0.565	0.066	0.588	0.827	0.670	1.35 hours	~ 700,000	0.663	19,848	50.38
Multinomial NB	10	200	200	15,000	0.550	0.063	0.594	0.806	0.666	1.4 hours	~ 700,000	0.654	19,848	75.57
Multinomial NB	10	200	200	19,848	0.618	0.058	0.663	0.851	0.731	1.6 hours	~ 700,000	0.716	19,848	100.00

Appendix A7 - Respective chart: B7

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Multinomial NB	10	200	/	1,000	0.356	0.032	0.507	0.586	0.498	2.20 hours	~ 1,900,000	0.487	29345	3.41
Multinomial NB	10	200	/	2,500	0.466	0.030	0.681	0.689	0.673	2.40 hours	~ 1,900,000	0.627	29345	8.52
Multinomial NB	10	200	/	3,500	0.468	0.033	0.702	0.691	0.685	2.40 hours	~ 1,900,000	0.637	29345	11.93
Multinomial NB	10	200	/	7,500	0.521	0.033	0.795	0.714	0.741	2.8 hours	~ 1,900,000	0.693	29345	25.56
Multinomial NB	10	200	/	10,000	0.489	0.034	0.843	0.693	0.747	3.10 hours	~ 1,900,000	0.693	29345	34.08
Multinomial NB	10	200	/	15,000	0.515	0.033	0.868	0.704	0.765	4.5 hours	~ 1,900,000	0.713	29345	51.12
Multinomial NB	10	200	/	20,000	0.493	0.033	0.878	0.694	0.763	5 hours	~ 1,900,000	0.707	29345	68.15

Appendix A8 - Respective chart: B8

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Bernoulli NB	10	200	200	1,000	0.210	0.054	0.258	0.410	0.287	40 minutes	~ 700,000	0.291	19841	5.04
Bernoulli NB	10	200	200	2,500	0.327	0.059	0.462	0.552	0.470	50 minutes	~ 700,000	0.453	19841	12.60
Bernoulli NB	10	200	200	3,500	0.319	0.056	0.497	0.582	0.499	1 hour	~ 700,000	0.474	19841	17.64
Bernoulli NB	10	200	200	7,500	0.311	0.068	0.461	0.580	0.476	1.2 hours	~ 700,000	0.457	19841	37.80
Bernoulli NB	10	200	200	10,000	0.309	0.059	0.494	0.569	0.491	1.35 hours	~ 700,000	0.466	19841	50.40
Bernoulli NB	10	200	200	15,000	0.308	0.060	0.489	0.577	0.498	1.4 hours	~ 700,000	0.468	19841	75.60
Bernoulli NB	10	200	200	19841	0.314	0.062	0.513	0.577	0.505	1.6 hours	~ 700,000	0.477	19841	100.00

Appendix A9 - Respective chart: B9

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Bernoulli NB	20	99	/	1,000	0.335	0.032	0.576	0.576	0.513	2.20 hours	~ 1,900,000	0.500	29345	3.41
Bernoulli NB	20	99	/	2,500	0.268	0.030	0.766	0.546	0.627	2.40 hours	~ 1,900,000	0.552	29345	8.52
Bernoulli NB	20	99	/	3,500	0.296	0.030	0.759	0.542	0.621	2.40 hours	~ 1,900,000	0.555	29345	11.93
Bernoulli NB	20	99	/	7,500	0.285	0.032	0.773	0.527	0.616	2.8 hours	~ 1,900,000	0.550	29345	25.56
Bernoulli NB	20	99	/	10,000	0.279	0.029	0.789	0.549	0.637	3.10 hours	~ 1,900,000	0.564	29345	34.08
Bernoulli NB	20	99	/	15,000	0.263	0.028	0.803	0.547	0.637	4.5 hours	~ 1,900,000	0.563	29345	51.12
Bernoulli NB	20	99	/	20,000	0.264	0.030	0.809	0.547	0.640	5 hours	~ 1,900,000	0.565	29345	68.15

Appendix A10 - Respective chart: B10

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Multinomial NB	20	99	99	3,500	0.288	0.067	0.260	0.431	0.312	50 minutes	~700,000	0.406	20,000	17.50
Multinomial NB	20	99	99	7,500	0.304	0.055	0.257	0.403	0.299	1.2 hours	~700,000	0.316	20,000	37.50
Multinomial NB	20	99	99	10,000	0.303	0.059	0.292	0.450	0.315	1.7 hours	~700,000	0.340	20,000	50.00
Multinomial NB	20	99	99	15,000	0.343	0.061	0.282	0.447	0.329	2 hours	~700,000	0.350	20,000	75.00
Multinomial NB	20	99	99	20,000	0.350	0.055	0.301	0.460	0.346	2.5 hours	~700,000	0.364	20,000	100.00

Appendix A11 - Respective chart: B11

Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Multinomial NB	20	99	/	3,500	0.375	0.032	0.504	0.590	0.518	2.4 hours	2,374,629	0.497	37198	9.41
Multinomial NB	20	99	/	7,500	0.395	0.032	0.680	0.593	0.619	3.4 hours	2,374,629	0.572	37198	20.16
Multinomial NB	20	99	/	10,000	0.429	0.030	0.742	0.606	0.655	4 hours	2,374,629	0.608	37198	26.88
Multinomial NB	20	99	/	15,000	0.457	0.033	0.715	0.637	0.661	4.5 hours	2,374,629	0.618	37198	40.32
Multinomial NB	20	99	/	20,000	0.465	0.033	0.725	0.645	0.671	6 hours	2,374,629	0.627	37198	53.77
Multinomial NB	20	99	/	30,000	0.431	0.030	0.758	0.610	0.664	8.8 hours	2,374,629	0.616	37198	80.65

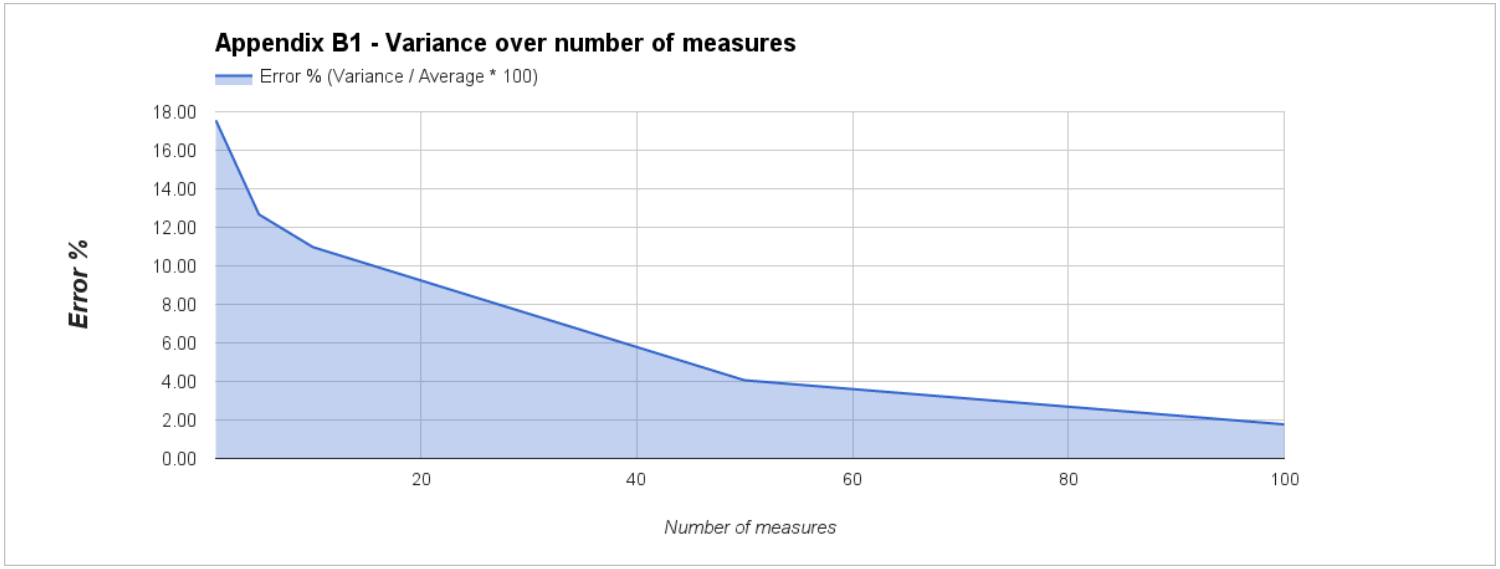
Appendix A12 - Respective chart: B12

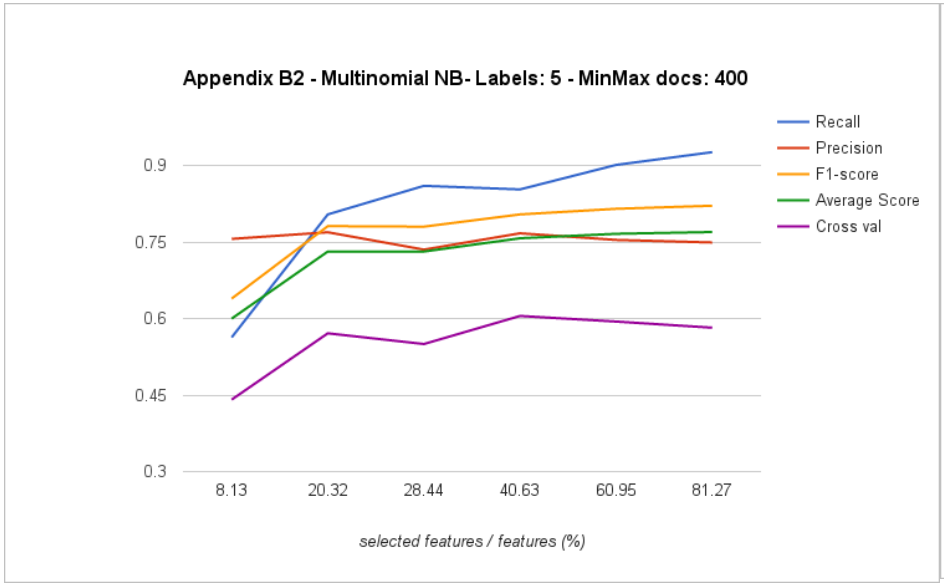
Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Bernoulli NB	20	99	99	3,500	0.106	0.038	0.125	0.158	0.133	50 minutes	~700,000	0.131	20,000	17.50
Bernoulli NB	20	99	99	7,500	0.101	0.040	0.127	0.155	0.133	1.2 hours	~700,000	0.129	20,000	37.50
Bernoulli NB	20	99	99	10,000	0.099	0.035	0.126	0.162	0.136	1.7 hours	~700,000	0.131	20,000	50.00
Bernoulli NB	20	99	99	15,000	0.113	0.037	0.135	0.154	0.137	2 hours	~700,000	0.135	20,000	75.00
Bernoulli NB	20	99	99	20000	0.102	0.038	0.123	0.153	0.129	2.5 hours	~700,000	0.127	20,000	100.00

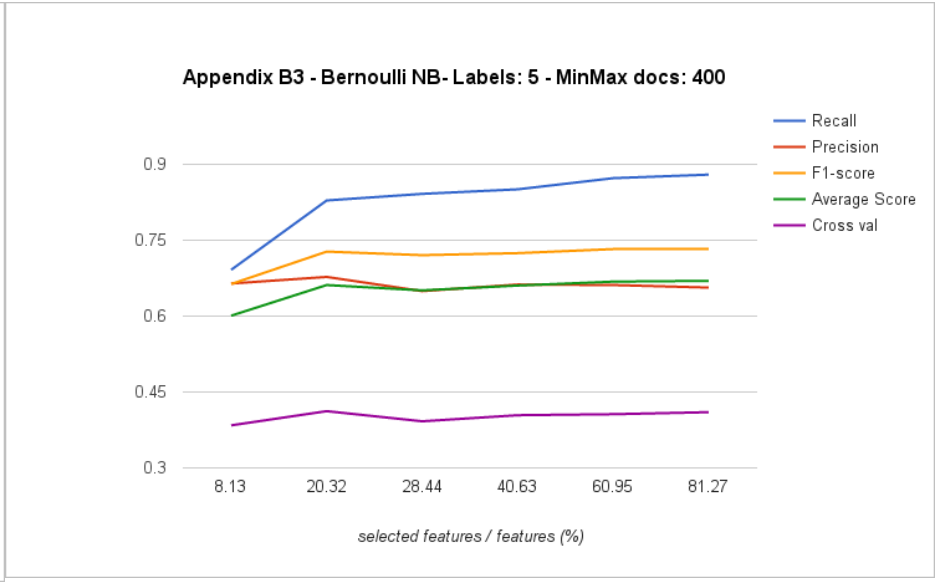
Appendix A13 - Respective chart: B13

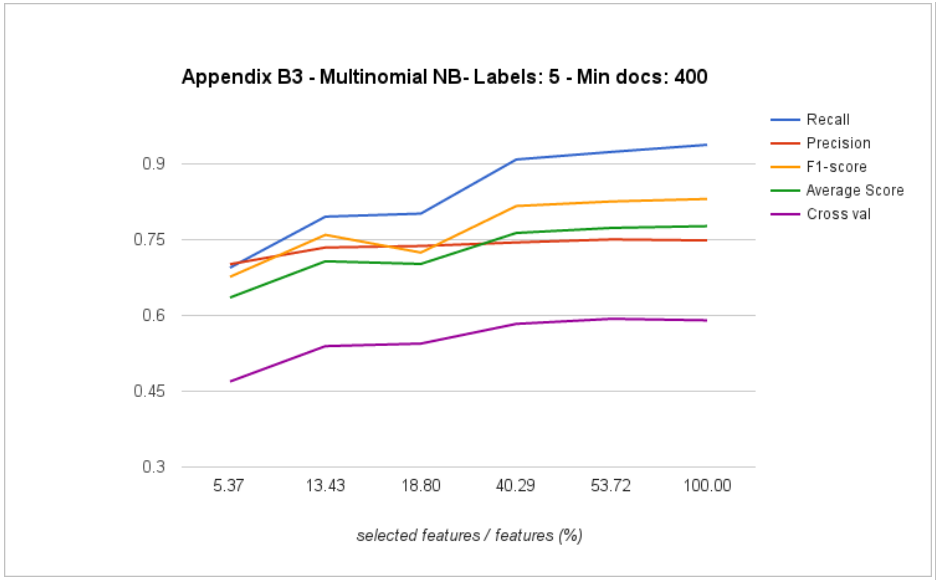
Type	Labels	Min docs	Max docs	Selected features	Cross val	Cross val precision	Recall	Precision	F1-score	Time	Vocabulary	Average Score	Features available	selected features / features (%)
Bernoulli NB	20	99	/	3,500	0.239	0.029	0.613	0.431	0.492	2.4 hours	2,374,629	0.444	37198	9.41
Bernoulli NB	20	99	/	7,500	0.206	0.026	0.683	0.435	0.521	3.4 hours	2,374,629	0.461	37198	20.16
Bernoulli NB	20	99	/	10,000	0.222	0.025	0.701	0.456	0.542	4 hours	2,374,629	0.480	37198	26.88
Bernoulli NB	20	99	/	15,000	0.205	0.025	0.693	0.443	0.532	4.5 hours	2,374,629	0.468	37198	40.32
Bernoulli NB	20	99	/	20,000	0.195	0.025	0.698	0.420	0.515	6 hours	2,374,629	0.457	37198	53.77
Bernoulli NB	20	99	/	30,000	0.195	0.027	0.698	0.437	0.528	8.8 hours	2,374,629	0.465	37198	80.65

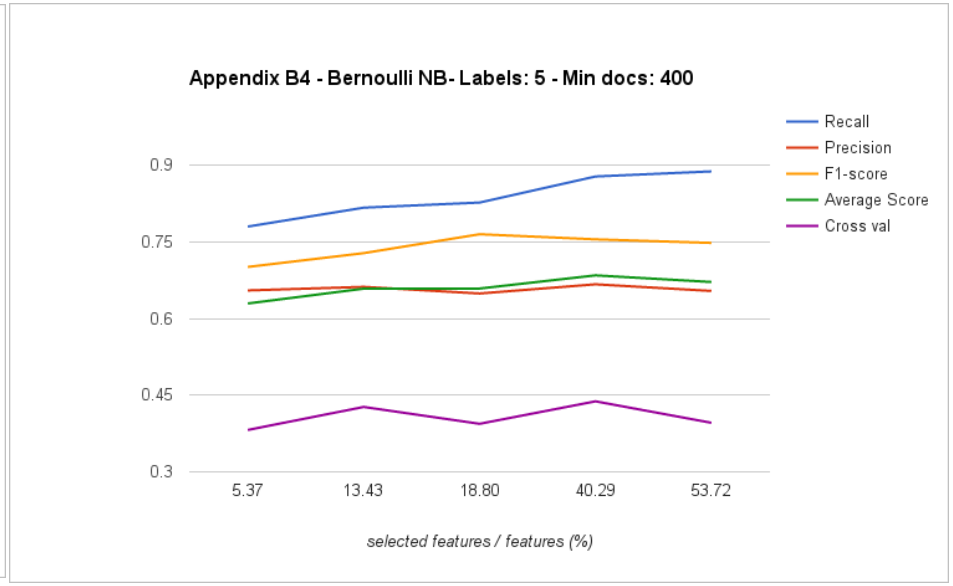
Appendix B - Charts

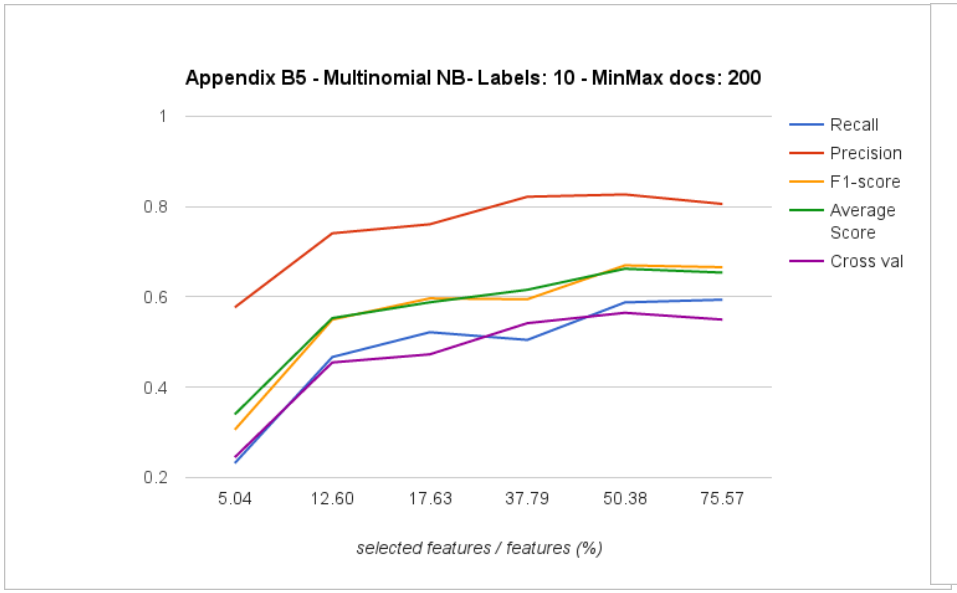


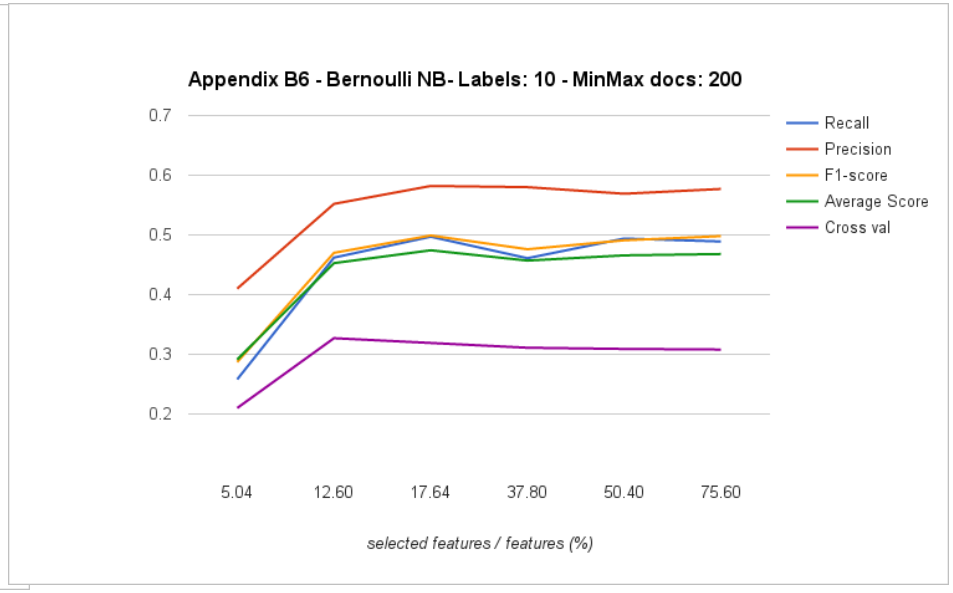


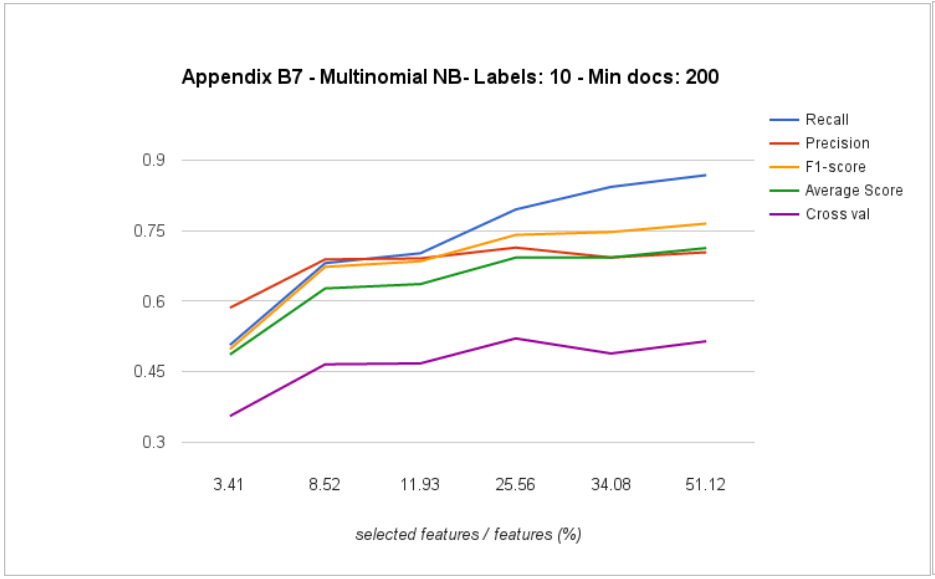


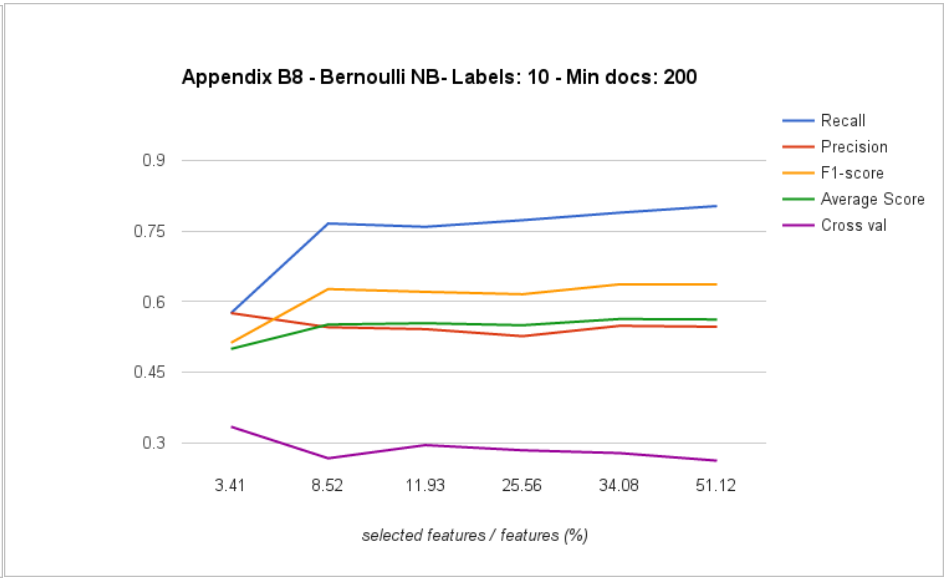

Appendix B3 - Multinomial NB- Labels: 5 - Min docs: 400
Recall
Precision
F1-score
Average Score
Cross val
0.9
0.75
0.6
0.45
0.3
5.37
13.43
18.80
40.29
53.72
100.00
selected features / features (%)

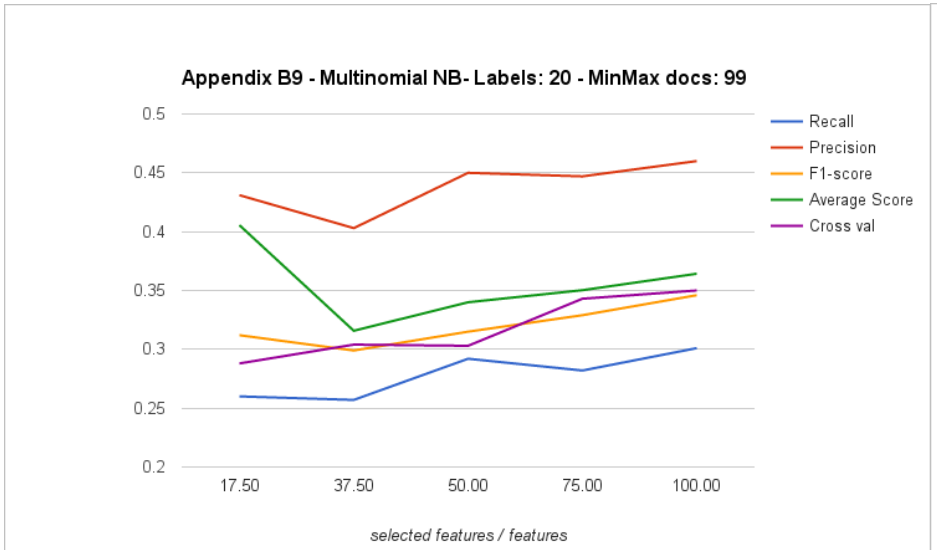

Appendix B4 - Bernoulli NB- Labels: 5 - Min docs: 400
Recall
Precision
F1-score
Average Score
Cross val
0.9
0.75
0.6
0.45
0.3
5.37
13.43
18.80
40.29
53.72
selected features / features (%)

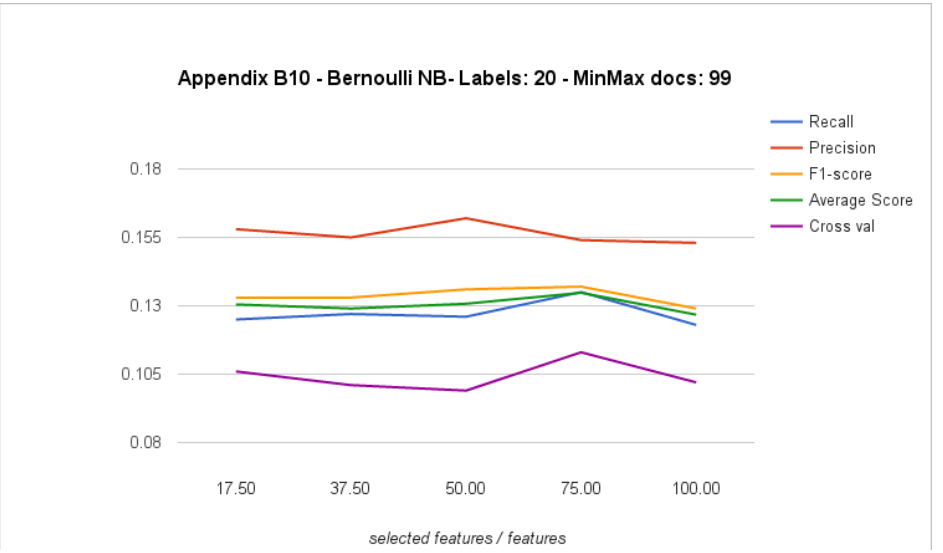

Appendix B5 - Multinomial NB- Labels: 10 - MinMax docs: 200
Recall
Precision
F1-score
Average Score
Cross val
1
0.8
0.6
0.4
0.2
5.04
12.60
17.63
37.79
50.38
75.57
selected features / features (%)

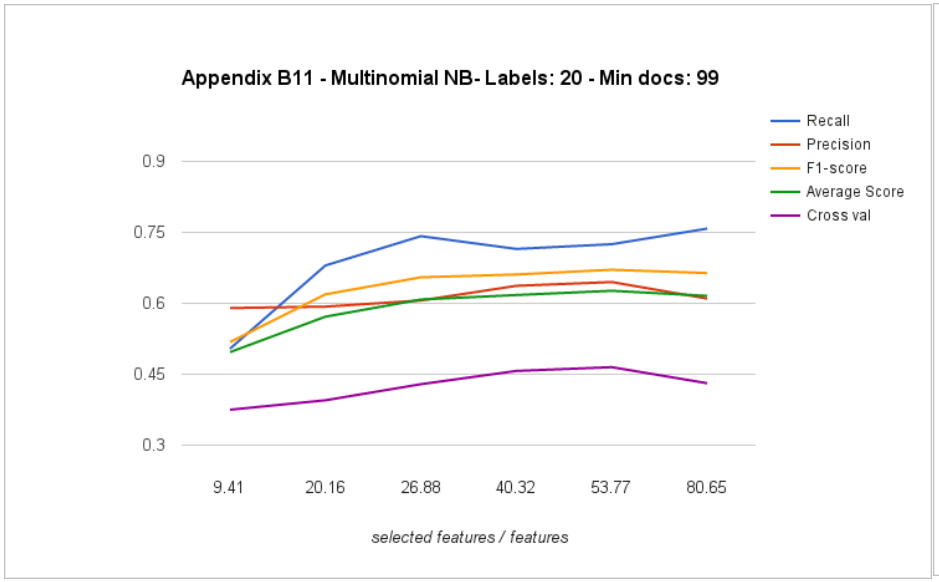

Appendix B6 - Bernoulli NB- Labels: 10 - MinMax docs: 200
Recall
Precision
F1-score
Average Score
Cross val
0.7
0.6
0.5
0.4
0.3
0.2
5.04
12.60
17.64
37.80
50.40
75.60
selected features / features (%)

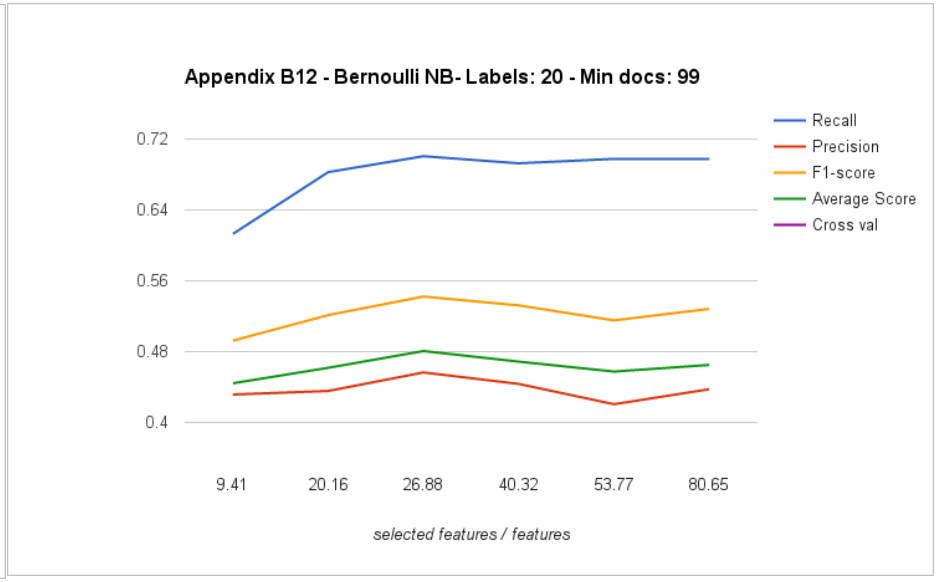

Appendix B7 - Multinomial NB- Labels: 10 - Min docs: 200
Recall
Precision
F1-score
Average Score
Cross val
0.9
0.75
0.6
0.45
0.3
3.41
8.52
11.93
25.56
34.08
51.12
selected features / features (%)


Appendix B8 - Bernoulli NB- Labels: 10 - Min docs: 200
Recall
Precision
F1-score
Average Score
Cross val
0.9
0.75
0.6
0.45
0.3
3.41
8.52
11.93
25.56
34.08
51.12
selected features / features (%)


Appendix B9 - Multinomial NB- Labels: 20 - MinMax docs: 99
Recall
Precision
F1-score
Average Score
Cross val
0.5
0.45
0.4
0.35
0.3
0.25
0.2
17.50
37.50
50.00
75.00
100.00
selected features / features


Appendix B10 - Bernoulli NB- Labels: 20 - MinMax docs: 99
Recall
Precision
F1-score
Average Score
Cross val
0.18
0.155
0.13
0.105
0.08
17.50
37.50
50.00
75.00
100.00
selected features / features


Appendix B11 - Multinomial NB- Labels: 20 - Min docs: 99
Recall
Precision
F1-score
Average Score
Cross val
0.9
0.75
0.6
0.45
0.3
9.41
20.16
26.88
40.32
53.77
80.65
selected features / features


Appendix B12 - Bernoulli NB- Labels: 20 - Min docs: 99
Recall
Precision
F1-score
Average Score
Cross val
0.72
0.64
0.56
0.48
0.4
9.41
20.16
26.88
40.32
53.77
80.65
selected features / features

Appendix C - Code

```
# Appendix C1 - setup.py

# Install nltk modules
import nltk
nltk.download('punkt')
nltk.download('stopwords')

from src.topic_importer import TopicImporter
from src.document_importer import DocumentImporter
from src.content_importer import ContentImporter
import os

database_name = "klassify"
if os.path.exists("%s.db" % database_name):
    os.remove("%s.db" % database_name)

# Add topics and subtopics
print("Importing topics and subtopics:")
TopicImporter().run()

# Add documents and associate them subtopics
print("Importing documents:")
DocumentImporter().run()

print("Importing documents HTML:")
ContentImporter().import_documents_html()

print("Importing documents data:")
ContentImporter().extract_documents_content()

```

```python
# Appendix C2 - topic_importer.py

import requests
from .tables import Topic, Subtopic
from .db_handler import DBHandler

class TopicImporter:
    def __init__(self):
        self.session = DBHandler(echo=False).session
        self.API_URL = "https://www.gov.uk/api/content"

    def make_topic(self, topic_data):
        return Topic(title=topic_data["title"],
            base_path=topic_data["base_path"], web_url=topic_data["web_url"],
            api_url=topic_data["api_url"], description=topic_data["description"])
    def make_subtopic(self, subtopic_data):
        return Subtopic(title=subtopic_data["title"],
            base_path=subtopic_data["base_path"], web_url=subtopic_data["web_url"],
            api_url=subtopic_data["api_url"], description=subtopic_data["description"])

    def associate_topic_subtopics(self, topic, subtopics):
        topic.subtopics = subtopics

    def run(self):
        root = requests.get(self.API_URL + "/topic").json()
        topics_json = root["links"]["children"]

        topics = []
        print("Importing topics and subtopics", end="", flush=True)
        for topic_json in topics_json:
            print('.', end="", flush=True)
            topic = self.make_topic(topic_json)
            topics.append(topic)

            topic_base_path = topic_json["base_path"]
            topic_data = requests.get(self.API_URL + topic_base_path).json()
            subtopics_json = topic_data["links"]["children"]
            subtopics = []
            for subtopic_json in subtopics_json:
                subtopics.append(self.make_subtopic(subtopic_json))
                self.associate_topic_subtopics(topic, subtopics)

        self.session.add_all(topics)
        self.session.add_all(subtopics)
        self.session.commit()
        print("\nComplete.")

```

```python
# Appendix C3 - doc_importer.py

import math
from .tables import Subtopic, Document
from .db_handler import DBHandler
import requests
import sqlalchemy
import time

class DocumentImporter(object):
    def __init__(self, db_name="klassify"):
        self.ROOT_URL = "https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING"
        self.PAGE_URL = "https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start="
        self.DBH = DBHandler(db_name, echo=False)

    def api_response(self, url):
        time.sleep(0.15)
        return requests.get(url).json()

    def total_documents(self, document_data):
        self.document_count = document_data["total"]
        return self.document_count

    def pages(self, number_of_documents):
        return math.ceil(number_of_documents / 1000)

    def urls(self, number_of_pages):
        urls = []
        for i in range(number_of_pages):
            item_count = i * 1000
            url_with_pagination = self.PAGE_URL + str(item_count)
            urls.append(url_with_pagination)
        return urls

    def associate_document_with_subtopics(self, document, subtopics):
        # remove duplicates by converting topics to a set and then back to a list
        subtopics = set(subtopics)
        subtopics = list(subtopics)
        document.subtopics = subtopics

        return document

    def make_document(self, document_data):
        link = document_data["link"]
        title = document_data["title"]
        description = document_data["description"]
        doc = Document(
            web_url="https://www.gov.uk" + link,
            description=description,
            base_path=link,
            title=title
        )

        return doc

    def find_subtopics(self, document_data):
        subtopics_data = document_data["specialist_sectors"]

        subtopics = []
        for subtopic_data in subtopics_data:
            subtopic = self.DBH.session.query(Subtopic).filter_by(base_path=subtopic_data['link']).first()
            if subtopic: subtopics.append(subtopic)

        return subtopics
```

```
65
66      def run(self):
67          root_data = self.api_response(self.ROOT_URL)
68          number_of_documents = self.total_documents(root_data)
69          pages = self.pages(number_of_documents)
70          urls = self.urls(pages)
71
72          count = 0
73          duplicate_documents = []
74
75          for url in urls:
76              list_of_documents =  self.api_response(url)
77              documents_data = list_of_documents['results']
78              for document_data in documents_data:
79                  document = self.make_document(document_data)
80                  subtopics = self.find_subtopics(document_data)
81                  if subtopics:
82                      self.associate_document_with_subtopics(document, subtopics)
83                  try:
84                      self.DBH.session.add(document)
85                      self.DBH.session.commit()
86                  except sqlalchemy.exc.IntegrityError:
87                      duplicate_documents.append(document.base_path)
88                      self.DBH.session.rollback()
89                  except:
90                      self.DBH.session.rollback()
91                      raise
92                  if count % 250 == 0: print("Documents processed: %d/%d" % (count, self.do
cument_count))
93                  count = count + 1
94
95          self.DBH.session.close()
96
97          print("Documents with duplicates that have been ignored: %d" % len(duplicate_docu
ments))
98
```

```python
# Appendix C4 - content_importer.py

from .db_handler import DBHandler
from .tables import Document
from bs4 import BeautifulSoup
import requests
import time

# Future implementation: Tuning features by adding Documents' to their content. Maybe with a multiplier.
class ContentImporter(object):
    def __init__(self, db_name="klassify"):
        self.DBH = DBHandler(db_name, echo=False)
        self.ROOT_URL = "https://www.gov.uk"
        self.NON_RELEVANT_PHRASES = [
            "Skip to main content",
            "Find out more about cookies"
            "GOV.UK uses cookies to make the site simpler",
            "Is there anything wrong with this page",
            "Last updated",
            "Other ways to apply",
            "Before you start",
            "Elsewhere on the web",
            "Find out about call charges",
            "find out more about beta services",
            "Return to top ↑",
            "Find out more about cookies",
            "GOV.UK",
            "Don’t include personal or financial information",
            "Help us improve",
            "This file may not be suitable for users of assistive technology"
            "If you use assistive technology and need a version of this document in a more accessible format",
            "tell us what format you need It will help us if you say what assistive technology you use",
            "Request a different format",
            "What you were doing",
            "What went wrong",
            "uses cookies to make the site simpler."
        ]

    def parse_page(self, page):
        soup = BeautifulSoup(page, 'html.parser')
        return soup

    def extract_page_content(self, page):
        return page.text

    # Iterate through each Document in the database, get their URL on the site and
    # query it to obtain their HTML and eventually store it.
    def import_documents_html(self):
        documents = self.DBH.session.query(Document).all()

        count = 0
        for doc in documents:
            if doc.html == None:
                time.sleep(0.75)
                doc.html = requests.get(doc.web_url).text
                self.DBH.session.commit()
            count += 1
            if count % 250 == 0: print("Documents processed: %d/%d" %(count, len(documents)))

    # Iterate through the Documents' HTML, parse it and store it.
    def extract_documents_content(self):
        documents = self.DBH.session.query(Document).all()

```

```
64          count = 0
65          for doc in documents:
66              doc.content = self.extract_content(doc)
67              self.DBH.session.commit()
68              count += 1
69              if count % 250 == 0: print("Documents processed: %d/%d" %(count, len(document
s)))
70
71      def extract_content(self, document):
72          page = self.parse_page(document.html)
73          page = self.remove_unwanted_tags(page)
74          page = self.get_body(page)
75
76          page_content = self.extract_page_content(page)
77          page_content = self.remove_non_relevant_content(page_content)
78          page_content = self.remove_punctuaction_and_numbers(page_content)
79          return page_content
80
81      def get_body(self, page):
82          return page.body
83
84      # Discard anything inside footer, header and scripts
85      def remove_unwanted_tags(self, page):
86          for tag in page.find_all(['footer', 'script', 'header']):
87              tag.replace_with('')
88
89          return page
90
91      def remove_non_relevant_content(self, page):
92          for phrase in self.NON_RELEVANT_PHRASES:
93              page = page.replace(phrase, "")
94          return page
95
96      def remove_punctuaction_and_numbers(self, page):
97          punctuation = [ '\\', '>', '_', '`', '{', ']', '*', '[',
98                          '^', '+', '!', '(', ':', ';', "'", "’",
99                          '<', '|', '"', '?', '=', '}', '&', '/',
100                          '$', ')', '~', '#', '%', ',' ]
101          page = ''.join(ch for ch in page if ch not in punctuation)
102          page = ''.join([i for i in page if not i.isdigit()])
103          return page
104
```

```python
# Appendix C5 - feature_extractor.py

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk

class FeatureExtractor():
    def __init__(self, documents, n_features=5000):
        self.documents = documents
        self.stemmer = PorterStemmer()
        self.vocabulary = self.top_words(n_features, self.freq_dist(self.make_vocabulary(
)))

    def tokenize(self, document=None):
        if document:
            documents = [document]
        else:
            documents = self.documents

        return [token for doc in documents for token in word_tokenize(doc.content)]

    def process(self, vocabulary):
        ADDITIONAL_STOP_WORDS = {'january', 'please', 'https', 'email', 'detail', 'email'
, 'send', 'if', 'december', 'october', 'kb', 'february', 'within', 'november', 'may', 'plea
se', '.mb', 'what', 'pdf', 'june', 'mach', 'good', 'august', 'september', 'html', 'july', '
beta', 'document', 'eg', 'published', 'april'}
        stop_words = set(stopwords.words("english"))

        processed_words = []
        for word in vocabulary:
            # select only words shorter than 20 char
            if len(word) < 20:
                word = word.lower()
                # do not select stopwords
                if word not in (stop_words | ADDITIONAL_STOP_WORDS):
                    # stem words
                    word = self.stemmer.stem(word)
                    # do not select words shorter than 2 characters
                    if word.isalpha:
                        if len(word) > 1:
                            processed_words.append(word)
                    else:
                        processed_words.append(word)
        return processed_words

    def make_vocabulary(self, document=None):
        if document:
            vocabulary = self.tokenize(document)
        else:
            vocabulary = self.tokenize()

        vocabulary = self.process(vocabulary)
        return vocabulary

    def bag_of_words(self, document):
        doc_words = set(self.make_vocabulary(document))
        bag_of_words = {}

        for word in self.vocabulary:
            bag_of_words[word] = (word in doc_words)

        return bag_of_words

    def freq_dist(self, vocabulary):
        return nltk.FreqDist(vocabulary)

```

```
64      def top_words(self, n_features, freq_dist):
65          return list(freq_dist.keys())[:n_features]
66
```

```python
# Appendix C6 - doc_operator.py

from .db_handler import DBHandler
from .tables import Topic, Subtopic, Document
from .feature_extractor import FeatureExtractor
import random

class DocumentOperator():
    def __init__(self, db_name="klassify", n=3, min_docs=None, max_docs=None, n_features=None):
        self.DBH = DBHandler(db_name=db_name, echo=False)
        self.topics = self.pick_random_topics(n, min_docs)
        self.max_docs = max_docs
        self.topic_labels = [topic.title for topic in self.topics]
        self.docs_with_labels = self.docs_with_labels()
        self.featuresets = []
        self.processor = FeatureExtractor([doc for doc, cat in self.docs_with_labels], n_features)

    def pick_random_topics(self, n, min_docs):
        topics = self.DBH.session.query(Topic).all()
        if min_docs:
            topics = [topic for topic in topics if len(topic.documents()) > min_docs]
        random.shuffle(topics)
        topics = topics[:n]
        return topics

    def find_random_doc_by_title(self, title):
        topic = self.DBH.session.query(Topic).filter(Topic.title == title).first()
        subtopic = random.choice(topic.subtopics)
        return random.choice(subtopic.documents)

    def random_document(self):
        all_topics = self.DBH.session.query(Topic).all()
        topic = random.choice(all_topics)
        subtopic = random.choice(topic.subtopics)
        doc = random.choice(subtopic.documents)
        bag_of_words = self.baggify_document(doc)
        return doc, bag_of_words

    def docs_with_labels(self):
        docs_with_filtered_labels = []

        for topic in self.topics:
            docs_with_labels = topic.documents_with_labels()

            if self.max_docs:
                random.shuffle(docs_with_labels)
                docs_with_labels = docs_with_labels[:self.max_docs]

            for doc, doc_labels in docs_with_labels:
                filtered_labels = []
                for label in doc_labels:
                    # Filter out labels that are not the selected topics
                     if label in self.topic_labels:
                         filtered_labels.append(label)
                docs_with_filtered_labels.append([doc, filtered_labels])

        return docs_with_filtered_labels

    def build_feature_sets(self):
        document_set_with_category = self.docs_with_labels
        random.shuffle(document_set_with_category)

        count = 0
        for (document, category) in document_set_with_category:
            count = count + 1
```

```
66              self.featuresets.append([self.baggify_document(document), category])
67
68      def baggify_document(self, doc):
69          return self.processor.bag_of_words(doc)
70
```

```python
# Appendix C7 - ovr_handler.py

from nltk import compat
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.feature_extraction import DictVectorizer
from sklearn import cross_validation
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.cross_validation import train_test_split

class OvrHandler():
    def __init__(self, featuresets):
        self.mlb = MultiLabelBinarizer()
        self.featuresets = featuresets
        self._vectorizer = DictVectorizer(dtype=float, sparse=True)
        self.X, self.y = self.prepare_scikit_x_and_y(self.featuresets)
        self.classifiers = {
            "MultinomialNB": OneVsRestClassifier(MultinomialNB()),
            "BernoulliNB": OneVsRestClassifier(BernoulliNB()),
        }

    def prepare_scikit_x_and_y(self, labeled_featuresets):
        X, y = list(compat.izip(*labeled_featuresets))
        X = self._vectorizer.fit_transform(X)

        set_of_labels = []
        for label in y:
            set_of_labels.append(set(label))

        y = self.mlb.fit_transform(set_of_labels)

        return X, y

    def train_classifiers(self):
        for name, clf in self.classifiers.items():
            clf.fit(self.X, self.y)

    def train_classifiers(self, X, y):
        for name, clf in self.classifiers.items():
            clf.fit(X, y)

    def cross_validate(self):
        results = {}
        for name, clf in self.classifiers.items():
            scores = cross_validation.cross_val_score(
                clf, self.X, self.y, cv=10
            )
            results[name] = {"cross score": scores.mean(), "cross variance": scores.std()
 * 2}
        return results

    def calculate_accuracy(self):
        results = {}
        X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, random_state=
0)
        for name, clf in self.classifiers.items():

            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)
            prob_pos = clf.predict_proba(X_test)[:, 1]
            precision = precision_score(y_test, y_pred, average='weighted')
            recall = recall_score(y_test, y_pred, average='weighted')
            f1 = f1_score(y_test, y_pred, average='weighted')

            results[name] = {"precision": precision, "recall": recall, "f1": f1}
```

```
66          return results
67
68      # Not used. For future implementation.
69      # Feed a document's bag of word to this method to obtain recommended classes
70      def predict_for_random(self, doc_with_bag_of_words):
71          doc, bag_of_words = doc_with_bag_of_words
72          print("Predicting for:", doc.title)
73          print("Item is labeled to:")
74          print(set(doc.topic_titles()))
75          print("====> Predictions:")
76
77          X = self._vectorizer.fit_transform(bag_of_words)
78
79          for name, clf in self.classifiers.items():
80              predicted_labels = (clf.predict(X))[0]
81              probabilities =  clf.predict_proba(X)[0]
82              named_classes = self.mlb.classes_
83
84              print("Using %s:" % name)
85
86              # If no labels are predicted for the item:
87              if not 1 in predicted_labels:
88                  print("No label suggested for item")
89                  return
90
91              for idx, label in enumerate(predicted_labels):
92                  confidence = round(float(probabilities[idx] * 100), 2)
93                  if confidence > 10:
94                      print(named_classes[idx] + " - Confidence: ", end="")
95                      print(str(confidence) + "%")
96
```

```python
# Appendix C8 - build_and_train_classifiers.py

from src.doc_operator import DocumentOperator
from src.ovr_handler import OvrHandler
from src.measure_calculator import MeasureCalculator
import time

calc = MeasureCalculator()
start_time = time.time()

count = 1
while count <= 100:
    doc_op = DocumentOperator(n=5, min_docs=400, max_docs=400, n_features=7500)
    doc_op.build_feature_sets()

    ovs = OvrHandler(doc_op.featuresets)

    cross_validation_measures = ovs.cross_validate()
    accuracy_measures = ovs.calculate_accuracy()

    calc.add_measures(cross_validation_measures, accuracy_measures)

    count += 1

calc.averaged_measures()

print("Total time: %0.2fs " % (time.time() - start_time))

```

```python
# Appendix C9 - measure_calculator.py

class MeasureCalculator():
    def __init__(self):
        self.measures = {
            "BernoulliNB": {
                "cross score": [],
                "cross variance": [],
                "precision": [],
                 "recall": [],
                 "f1": []
            },
            "MultinomialNB": {
                "cross score": [],
                "cross variance": [],
                "precision": [],
                "recall": [],
                "f1": []
            }
        }

    def add_measures(self, cross_validation_measures, accuracy_measures):
        measures = self.combine_measures(cross_validation_measures, accuracy_measures)
        for algo_type, results in measures.items():
            for result, value in results.items():
                self.measures[algo_type][result].append(value)

    def combine_measures(self, cross_validation_measures, accuracy_measures):
        current_measures = {}
        current_measures["BernoulliNB"] = dict(
            list(cross_validation_measures["BernoulliNB"].items()) +
            list(accuracy_measures["BernoulliNB"].items())
        )
        current_measures["MultinomialNB"] = dict(
            list(cross_validation_measures["MultinomialNB"].items()) +
            list(accuracy_measures["MultinomialNB"].items())
        )
        return current_measures

    def averaged_measures(self):
        for algo_type, results in self.measures.items():
            print(algo_type + ":")
            cross_score = (sum(results["cross score"]) / len(results["cross score"]))
            cross_precision = (sum(results["cross variance"]) / len(results["cross varian
ce"]))

            # Print out average of cross eval measure along with its variance
            print("Cross evaluation accuracy: %1.3f (+/- %1.3f)" % (cross_score, cross_pr
ecision))
            results.pop("cross score")
            results.pop(("cross variance"))

            for result, values in results.items():
                # Print out averages of all remaining measures
                print("%s: %1.3f" % (result, (sum(values) / len(values))))

```

```python
# Appendix C10 - tables.py

from sqlalchemy import Table, Column, Integer, String, Text
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship, backref
from .base import Base

class Topic(Base):
    __tablename__ = 'topics'

    id          = Column(Integer, primary_key=True)
    title       = Column(String)
    base_path   = Column(String, unique=True)
    description = Column(String)
    web_url     = Column(String)
    api_url     = Column(String)

    def __repr__(self):
        return "<Topic(title='%s', base_path='%s')>" % (self.title, self.base_path)

    def documents(self):
        documents = set()
        for subtopic in self.subtopics:
            for doc in subtopic.documents:
                documents.add(doc)

        return list(documents)

    def documents_with_labels(self):
        doc_with_labels = []
        for doc in self.documents():
            doc_with_labels.append([doc, doc.topic_titles()])
        return doc_with_labels

# create association table SubtopicDcoument
subtopics_documents = Table('subtopics_documents', Base.metadata,
    Column('subtopic_id', ForeignKey('subtopics.id'), primary_key=True),
    Column('document_id', ForeignKey('documents.id'), primary_key=True)
)

class Subtopic(Base):
    __tablename__ = 'subtopics'

    id          = Column(Integer, primary_key=True)
    title       = Column(String)
    base_path   = Column(String, unique=True)
    description = Column(String)
    web_url     = Column(String)
    api_url     = Column(String)

    topic_id  = Column(Integer, ForeignKey('topics.id'))
    topic     = relationship("Topic", back_populates="subtopics")

    documents = relationship(
        "Document", secondary=subtopics_documents, back_populates="subtopics"
    )

    def __repr__(self):
        return "<Subtopic(title='%s', base_path='%s')>" % (self.title, self.base_path)

# link topic to subtopics
Topic.subtopics = relationship(
    "Subtopic", order_by=Subtopic.id, back_populates="topic"
)

class Document(Base):
    __tablename__ = 'documents'
```

```
68
69      id          = Column(Integer, primary_key=True)
70      title       = Column(String)
71      base_path   = Column(String, unique=True)
72      web_url     = Column(String)
73      html        = Column(Text)
74      description = Column(Text)
75      content     = Column(Text)
76
77      subtopics = relationship(
78          'Subtopic', secondary=subtopics_documents, back_populates='documents'
79      )
80
81      def __init__(self, title, base_path, html=None, description=None, web_url=None, conte
nt=None):
82          self.title       = title
83          self.base_path   = base_path
84          self.html        = html
85          self.web_url     = web_url
86          self.description = description
87          self.content     = content
88
89      def __repr__(self):
90          return "<Document(title=%r, base_path=%r)" % (self.title, self.base_path)
91
92      def topics(self):
93          topics = set()
94
95          for subtopic in self.subtopics:
96              topics.add(subtopic.topic)
97
98          return list(topics)
99
100      def topic_titles(self):
101          return [topic.title for topic in self.topics()]
102
```

```
# Appendix C11 - base.py

from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()

```

```
# Appendix C12 - db_handler.py

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from .base import Base
import os

class DBHandler(object):
    def __init__(self, db_name="klassify", echo=True):
        self.db_name = db_name
        self.db = "sqlite:///%s.db" % self.db_name
        self.engine = create_engine(self.db, echo=echo)
        Session = sessionmaker(bind=self.engine)
        self.session = Session()
        Base.metadata.create_all(self.engine)

    def destroy_db_if_present(self):
        if os.path.exists("%s.db" % self.db_name):
            print("Removing %s database" % self.db_name)
            os.remove("%s.db" % self.db_name)

```

```
# Appendix C13 - test_content_importer.py

from klassify.src.tables import Document
from klassify.src.content_importer import ContentImporter
import os
import pytest

database_name = "test_klassify"2
if os.path.exists("%s.db" % database_name):
    os.remove("%s.db" % database_name)

DOCUMENT = Document(
    base_path = "/intelligent-machines",
    title = "The Intelligent Machines",
    html = open("test/fixtures/document_page.html", 'r').read())
STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER = "How government works"
STRING_PRESENT_IN_SCRIPT_TAG = "<![CDATA["
STRING_PRESENT_IN_TITLE = "HM Revenue & Customs"

def setup_module(module):
    global IMPORTER
    IMPORTER = ContentImporter(db_name="test_klassify")
    IMPORTER.DBH.session.add(DOCUMENT)
    IMPORTER.DBH.session.commit()
def teardown_module(module):
    IMPORTER.DBH.session.close()
    IMPORTER.DBH.destroy_db_if_present()

def test_cleaning_methods():
    doc = IMPORTER.DBH.session.query(Document).first()
    page = IMPORTER.parse_page(doc.html)

    assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER in page.text
    assert STRING_PRESENT_IN_SCRIPT_TAG in page.text
    page = IMPORTER.remove_unwanted_tags(page)
    assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER not in page.text
    assert STRING_PRESENT_IN_SCRIPT_TAG not in page.text

    assert STRING_PRESENT_IN_TITLE in page.text
    page = IMPORTER.get_body(page)
    assert STRING_PRESENT_IN_TITLE not in page.text

    page_content = IMPORTER.extract_page_content(page)
    page_content = IMPORTER.remove_non_relevant_content(page_content)
    for phrase in IMPORTER.NON_RELEVANT_PHRASES:
        assert phrase not in page_content

    assert "2016" in page_content
    page_content = IMPORTER.remove_punctuaction_and_numbers(page_content)
    assert "2016" not in page_content

def test_extract_content_single_method():
    doc = IMPORTER.DBH.session.query(Document).first()

    assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER in doc.html
    assert STRING_PRESENT_IN_SCRIPT_TAG in doc.html

    clean_content = IMPORTER.extract_content(doc)

    assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER not in clean_content
    assert STRING_PRESENT_IN_SCRIPT_TAG not in clean_content
    for phrase in IMPORTER.NON_RELEVANT_PHRASES:
        assert phrase not in clean_content

```

```
# Appendix C14 - test_doc_operator.py

from klassify.src.doc_operator import DocumentOperator
from klassify.src.db_handler import DBHandler
from klassify.src.tables import Document, Subtopic, Topic
import os
import pytest

database_name = "test_klassify"
if os.path.exists("%s.db" % database_name):
    os.remove("%s.db" % database_name)

def test_docs_with_labels():
    document_1 = Document(title="Test title 1",
                          base_path="/test-1",
                          content="This is a test document - one")
    document_2 = Document(title="Test title 2",
                          base_path="/test-2",
                          content="This is a test document - two")

    topic_1 = Topic(
        title='Label 1',
        base_path='/topic/working-sea',
        description='List of information about Topic.'
    )
    topic_2 = Topic(
        title='Label 2',
        base_path='/topic/working-sea-2',
        description='List of information about Topic. 2'
    )

    subtopic_1 = Subtopic(
        title='Subtopic',
        base_path='/topic/working-sea',
        description='List of information about Subtopic.'
    )
    subtopic_2 = Subtopic(
        title='Subtopic 2',
        base_path='/topic/working-sea-2',
        description='List of information about Subtopic. 2'
    )

    topic_1.subtopics = [subtopic_1]
    topic_2.subtopics = [subtopic_2]
    subtopic_1.documents = [document_1]
    subtopic_2.documents = [document_2]

    DBH = DBHandler(db_name=database_name, echo=False)
    session = DBH.session
    session.add_all([topic_1, topic_2, subtopic_1, subtopic_2, document_1, document_2])
    session.commit()

    doc_op = DocumentOperator(db_name = database_name)

    docs_with_labels = doc_op.docs_with_labels
    first_set = docs_with_labels[0]
    second_set = docs_with_labels[1]

    assert [document_1.title, [topic_1.title]] in [[first_set[0].title, first_set[1]], [s
econd_set[0].title, second_set[1]]]
    assert [document_2.title, [topic_2.title]] in [[first_set[0].title, first_set[1]], [s
econd_set[0].title, second_set[1]]]

```

```python
# Appendix C15 - test_document_importer.py

from klassify.src.tables import Subtopic, Document
from klassify.src.document_importer import DocumentImporter
import json
import subprocess
import os
import pytest
import sqlalchemy

database_name = "test_klassify"
if os.path.exists("%s.db" % database_name):
    os.remove("%s.db" % database_name)

with open('test/fixtures/tagged_documents.json', encoding='utf-8') as fixture_file:
    api_response_fixture = json.loads(fixture_file.read())

DOCUMENT_DATA = api_response_fixture["results"][0]
SUBTOPICS = [
    Subtopic(
        base_path='/topic/driving-tests-and-learning-to-drive/car',
        title='Cars'
    ),
    Subtopic(
        base_path='/topic/driving-tests-and-learning-to-drive/lorry-bus',
        title='Lorries and buses'
    )
]

def setup_module(module):
    global IMPORTER
    IMPORTER = DocumentImporter(db_name=database_name)
def teardown_module(module):
    IMPORTER.DBH.session.close()
    IMPORTER.DBH.destroy_db_if_present()

def test_total_documents():
    assert IMPORTER.total_documents(api_response_fixture) == 9623

def test_pages():
    assert IMPORTER.pages(9623) == 10

def test_urls():
    assert IMPORTER.urls(10) == [
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=0',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=1000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=2000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=3000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=4000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=5000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=6000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=7000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=8000',
        'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000
&start=9000'
    ]

def test_make_document():
```

```
58      made_document = IMPORTER.make_document(DOCUMENT_DATA)
59
60      assert made_document.web_url == 'https://www.gov.uk/view-driving-licence'
61      assert made_document.base_path == '/view-driving-licence'
62      assert made_document.title == 'View or share your driving licence information'
63      assert made_document.description == 'Find out what information DVLA holds about your
driving licence or create a check code to share your driving record (eg to hire a car)'
64
65  def test_associate_document_with_subtopics():
66      made_document = IMPORTER.make_document(DOCUMENT_DATA)
67      IMPORTER.associate_document_with_subtopics(made_document, SUBTOPICS)
68      assert made_document.subtopics[0] in SUBTOPICS
69      assert made_document.subtopics[1] in SUBTOPICS
70
71  def test_find_subtopics():
72      session = IMPORTER.DBH.session
73      session.add_all(SUBTOPICS)
74      session.commit()
75
76      found_topics = IMPORTER.find_subtopics(DOCUMENT_DATA)
77
78      subtopics_titles = [subtopic.title for subtopic in SUBTOPICS]
79      assert found_topics[0].title in subtopics_titles
80      assert found_topics[1].title in subtopics_titles
81
```

```
# Appendix C16 - test_feature_extractor.py

from klassify.src.feature_extractor import FeatureExtractor
from klassify.src.tables import Document

initial_document_1 = Document(title="Test title 1",
                              base_path="/test-1",
                              content="This is a test document - one")
initial_document_2 = Document(title="Test title 2",
                               base_path="/test-2",
                               content="This is a test document - two")
initial_document_3 = Document(title="Test title 3",
                               base_path="/test-3",
                               content="This is a test document - three")

EXTRACTOR = FeatureExtractor([
    initial_document_1,
    initial_document_2,
    initial_document_3,
])

new_document = Document(title="Self assessment deadlines 3",
                        base_path="/self-assessment-3",
                        html="<strong>PAY NOW 3</strong>",
                        content="This has a different content - four")

def test_tokenize():
    tokenized_content = EXTRACTOR.tokenize(initial_document_1)
    assert tokenized_content == ['This', 'is', 'a', 'test', 'document', "-", 'one']

def test_make_vocabulary():
    # without document
    assert EXTRACTOR.make_vocabulary() == ['test', 'one', 'test', 'two', 'test', 'three']
    # with document
    assert EXTRACTOR.make_vocabulary(new_document) ==  ['differ', 'content', 'four']

def test_bag_of_words():
    # This is built against the vocabulary.
    # The vocabulary is the sum of all the different terms in all the documents provided
at instantiation.
    assert EXTRACTOR.bag_of_words(initial_document_3) == {'one': False, 'test': True, 'th
ree': True, 'two': False}
    assert EXTRACTOR.bag_of_words(new_document) == {'one': False, 'test': False, 'three':
 False, 'two': False}

def test_process():
    # What is bein discarded: Single letter words, Stop words, Long words
    # Additionally, remaining words will be stemmed.
    document_with_unfiltered_content = Document(title="Test", base_path="/test",
        content=" within https .mb , a b c reallylongwordthatshouldbefilteredout cloudy r
egular words should be stemmed in this process"
    )

    tokenized_content = EXTRACTOR.tokenize(document_with_unfiltered_content)

    assert EXTRACTOR.process(tokenized_content) == ['cloudi', 'regular', 'word', 'stem',
'process']

```

```python
# Appendix C17 - test_measure_calculator.py

from klassify.src.measure_calculator import MeasureCalculator
from klassify.src.tables import Topic, Subtopic

first_set = {
    "BernoulliNB": {
        "cross score": 3, "precision": 1, "cross variance": 1
    },
    "MultinomialNB": {
        "cross score": 2, "precision": 2, "cross variance": 2
    }
}
second_set = {
    "BernoulliNB": {"recall": 3, "f1": 1},
    "MultinomialNB": {"recall": 2, "f1": 2}
}

# Groups two sets of measures by the algorithm type
def test_combine_measures():
    CALC = MeasureCalculator()

    assert CALC.combine_measures(first_set, second_set) == {
        "BernoulliNB": {
            "cross score": 3, "precision": 1, "recall": 3, "f1": 1, "cross variance": 1
        },
        "MultinomialNB": {
            "cross score": 2, "precision": 2, "recall": 2, "f1": 2, "cross variance": 2
        }
    }

# Store sets of measures
def test_add_measures():
    CALC = MeasureCalculator()

    CALC.add_measures(first_set, second_set)

    assert CALC.measures == {
        "BernoulliNB": {
            "cross score": [3], "precision": [1], "recall": [3], "f1": [1], "cross varian
ce": [1]
        },
        "MultinomialNB": {
            "cross score": [2], "precision": [2], "recall": [2], "f1": [2], "cross varian
ce": [2]
        }
    }

```

```python
# Appendix C18 - test_table_definition.py

from klassify.src.db_handler import DBHandler
from klassify.src.tables import Topic, Subtopic, Document
import pytest
import sqlalchemy

def test_db():
    database_name = "test_klassify"

    DBH = DBHandler(database_name, echo=False)
    session = DBH.session
    # create a topic, subtopic and document
    test_topic = Topic(title="HMRC", base_path="/hmrc")
    test_subtopic_1 = Subtopic(title="HMRC payments", base_path="/payments")
    test_subtopic_2 = Subtopic(title="HMRC refunds", base_path="/refunds")
    test_document_1 = Document(
        title="Self assessment deadlines",
        base_path="/self-assessment",
        html="<strong>PAY NOW</strong>")
    test_document_2 = Document(
        title="Starting a business",
        base_path="/start-business",
        html="<strong>START NOW</strong>")
    test_document_3 = Document(
        title="Payment and refunds",
        base_path="/payments-and-refunds",
        html="<h1>payments and refunds</h1>")

    # create relationships
    test_topic.subtopics    = [test_subtopic_1, test_subtopic_2]
    test_subtopic_1.documents = [test_document_1, test_document_2]
    test_document_3.subtopics = [test_subtopic_1, test_subtopic_2]

    # add topic to session
    session.add_all([
        test_topic,
        test_subtopic_1,
        test_subtopic_2,
        test_document_1,
        test_document_2,
        test_document_3
    ])

    session.commit()

    # Table properties
    assert session.query(Topic).get(test_topic.id).title == test_topic.title
    assert session.query(Topic).get(test_topic.id).base_path == test_topic.base_path
    assert session.query(Subtopic).get(test_subtopic_1.id).title == test_subtopic_1.title
    assert session.query(Subtopic).get(test_subtopic_1.id).base_path == test_subtopic_1.b
ase_path
    assert session.query(Document).get(test_document_1.id).title == test_document_1.title
    assert session.query(Document).get(test_document_1.id).base_path == test_document_1.b
ase_path

    # test relationships
    topics_and_subtopics = session.query(Topic).get(test_topic.id).subtopics
    subtopics_titles = [subtopic.title for subtopic in topics_and_subtopics]
    assert test_subtopic_1.title in subtopics_titles
    assert test_subtopic_2.title in subtopics_titles

    subtopics_and_documents = session.query(Subtopic).get(test_subtopic_1.id).documents
    documents_titles = [document.title for document in subtopics_and_documents]
    assert test_document_1.title in documents_titles
    assert test_document_2.title in documents_titles

```

```
    documents_and_subtopics = session.query(Document).get(test_document_3.id).subtopics
    subtopics_titles = [subtopic.title for subtopic in documents_and_subtopics]
    assert test_subtopic_1.title in subtopics_titles
    assert test_subtopic_2.title in subtopics_titles

    # Test Document->Topics relation
    doc = session.query(Document).get(test_document_1.id)
    topic = session.query(Topic).get(test_topic.id)
    assert topic in doc.topics()
    assert topic.title in doc.topic_titles()

    # Test Topic->Documents relation
    doc = session.query(Document).get(test_document_1.id)
    topic = session.query(Topic).get(test_topic.id)
    assert doc in topic.documents()
    assert doc, topic.title in topic.documents_with_labels()

    # test unique constraint on basepath
    clone_topic = Topic(title="Clone topic", base_path="/hmrc")
    clone_subtopic = Subtopic(title="Clone subtopic", base_path="/refunds")
    clone_document = Document(
        title="Clone document",
        base_path="/payments-and-refunds",
        html="<h1>payments and refunds</h1>")
    clones = [clone_topic, clone_subtopic, clone_document]
    for clone in clones:
        with pytest.raises(sqlalchemy.exc.IntegrityError):
            session.rollback()
            session.add_all([clone])
            session.commit()

    # terminate session and delete test db
    session.close()
    DBH.destroy_db_if_present()
```

```python
# Appendix C19 - test_topic_importer.py

from klassify.src.topic_importer import TopicImporter
from klassify.src.tables import Topic, Subtopic

IMPORTER = TopicImporter()

def test_make_topic():
    topic_fixture = {'base_path': '/topic/working-sea', 'web_url': 'https://www.gov.uk/topic/working-sea', 'content_id': '077826e8-f094', 'description': 'List of information about Working at sea.', 'title': 'Working at sea', 'api_url': 'https://www.gov.uk/api/content/topic/working-sea'}

    created_topic = IMPORTER.make_topic(topic_fixture)

    expected_topic = Topic(
        title='Working at sea',
        base_path='/topic/working-sea',
        web_url='https://www.gov.uk/topic/working-sea',
        api_url='https://www.gov.uk/api/content/topic/working-sea',
        description='List of information about Working at sea.'
    )

    assert created_topic.title == expected_topic.title
    assert created_topic.base_path == expected_topic.base_path
    assert created_topic.web_url == expected_topic.web_url
    assert created_topic.api_url == expected_topic.api_url
    assert created_topic.description == expected_topic.description

def test_make_subtopic():
    subtopic_fixture = {'content_id': '6382617d-a2c5-4651-b487-5d267dfc6662', 'locale': 'en', 'base_path': '/topic/working-sea/health-safety', 'description': 'List of information about Health and safety.', 'api_url': 'https://www.gov.uk/api/content/topic/working-sea/health-safety', 'title': 'Health and safety', 'web_url': 'https://www.gov.uk/topic/working-sea/health-safety'}

    created_subtopic = IMPORTER.make_topic(subtopic_fixture)

    expected_subtopic = Subtopic(
        title='Health and safety',
        base_path='/topic/working-sea/health-safety',
        web_url='https://www.gov.uk/topic/working-sea/health-safety',
        api_url='https://www.gov.uk/api/content/topic/working-sea/health-safety',
        description='List of information about Health and safety.'
    )

    assert created_subtopic.title == expected_subtopic.title
    assert created_subtopic.base_path == expected_subtopic.base_path
    assert created_subtopic.web_url == expected_subtopic.web_url
    assert created_subtopic.api_url == expected_subtopic.api_url
    assert created_subtopic.description == expected_subtopic.description

def test_associate_topic_subtopics():
    topic = Topic(title="A topi title")
    subtopic_1 = Subtopic(title="A subtopic title 1")
    subtopic_2 = Subtopic(title="A subtopic title 2")

    IMPORTER.associate_topic_subtopics(topic, [subtopic_1, subtopic_2])

    assert subtopic_1.title == topic.subtopics[0].title
    assert subtopic_2.title == topic.subtopics[1].title

```

```
# Appendix C20 - requirements.txt

pytest==2.8.7
SQLAlchemy==1.0.12
beautifulsoup4==4.4.1
responses==0.5.1
numpy==1.10.4
scipy==0.17.0
nltk==3.2
scikit-learn==0.17.1

```