

Appendix C - Code

```
1 # Appendix C1 - setup.py
2
3 # Install nltk modules
4 import nltk
5 nltk.download('punkt')
6 nltk.download('stopwords')
7
8 from src.topic_importer import TopicImporter
9 from src.document_importer import DocumentImporter
10 from src.content_importer import ContentImporter
11 import os
12
13 database_name = "klassify"
14 if os.path.exists("%s.db" % database_name):
15     os.remove("%s.db" % database_name)
16
17 # Add topics and subtopics
18 print("Importing topics and subtopics:")
19 TopicImporter().run()
20
21 # Add documents and associate them subtopics
22 print("Importing documents:")
23 DocumentImporter().run()
24
25 print("Importing documents HTML:")
26 ContentImporter().import_documents_html()
27
28 print("Importing documents data:")
29 ContentImporter().extract_documents_content()
30
```

```

1 # Appendix C2 - topic_importer.py
2
3 import requests
4 from .tables import Topic, Subtopic
5 from .db_handler import DBHandler
6
7 class TopicImporter:
8     def __init__(self):
9         self.session = DBHandler(echo=False).session
10        self.API_URL = "https://www.gov.uk/api/content"
11
12    def make_topic(self, topic_data):
13        return Topic(title=topic_data["title"],
14                     base_path=topic_data["base_path"], web_url=topic_data["web_url"],
15                     api_url=topic_data["api_url"], description=topic_data["description"])
16    def make_subtopic(self, subtopic_data):
17        return Subtopic(title=subtopic_data["title"],
18                       base_path=subtopic_data["base_path"], web_url=subtopic_data["web_url"],
19                       api_url=subtopic_data["api_url"], description=subtopic_data["description"])
20
21    def associate_topic_subtopics(self, topic, subtopics):
22        topic.subtopics = subtopics
23
24    def run(self):
25        root = requests.get(self.API_URL + "/topic").json()
26        topics_json = root["links"]["children"]
27
28        topics = []
29        print("Importing topics and subtopics", end="", flush=True)
30        for topic_json in topics_json:
31            print('.', end="", flush=True)
32            topic = self.make_topic(topic_json)
33            topics.append(topic)
34
35            topic_base_path = topic_json["base_path"]
36            topic_data = requests.get(self.API_URL + topic_base_path).json()
37            subtopics_json = topic_data["links"]["children"]
38            subtopics = []
39            for subtopic_json in subtopics_json:
40                subtopics.append(self.make_subtopic(subtopic_json))
41                self.associate_topic_subtopics(topic, subtopics)
42
43        self.session.add_all(topics)
44        self.session.add_all(subtopics)
45        self.session.commit()
46        print("\nComplete.")
47

```

```

1 # Appendix C3 - doc_importer.py
2
3 import math
4 from .tables import Subtopic, Document
5 from .db_handler import DBHandler
6 import requests
7 import sqlalchemy
8 import time
9
10 class DocumentImporter(object):
11     def __init__(self, db_name="klassify"):
12         self.ROOT_URL = "https://www.gov.uk/api/search.json?reject_specialist_sectors=_MI
SSING"
13         self.PAGE_URL = "https://www.gov.uk/api/search.json?reject_specialist_sectors=_MI
SSING&count=1000&start="
14         self.DBH = DBHandler(db_name, echo=False)
15
16     def api_response(self, url):
17         time.sleep(0.15)
18         return requests.get(url).json()
19
20     def total_documents(self, document_data):
21         self.document_count = document_data["total"]
22         return self.document_count
23
24     def pages(self, number_of_documents):
25         return math.ceil(number_of_documents / 1000)
26
27     def urls(self, number_of_pages):
28         urls = []
29         for i in range(number_of_pages):
30             item_count = i * 1000
31             url_with_pagination = self.PAGE_URL + str(item_count)
32             urls.append(url_with_pagination)
33         return urls
34
35     def associate_document_with_subtopics(self, document, subtopics):
36         # remove duplicates by converting topics to a set and then back to a list
37         subtopics = set(subtopics)
38         subtopics = list(subtopics)
39         document.subtopics = subtopics
40
41         return document
42
43     def make_document(self, document_data):
44         link = document_data["link"]
45         title = document_data["title"]
46         description = document_data["description"]
47         doc = Document(
48             web_url="https://www.gov.uk" + link,
49             description=description,
50             base_path=link,
51             title=title
52         )
53
54         return doc
55
56     def find_subtopics(self, document_data):
57         subtopics_data = document_data["specialist_sectors"]
58
59         subtopics = []
60         for subtopic_data in subtopics_data:
61             subtopic = self.DBH.session.query(Subtopic).filter_by(base_path=subtopic_data
['link']).first()
62             if subtopic: subtopics.append(subtopic)
63
64         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

```

```

1 # Appendix C4 - content_importer.py
2
3 from .db_handler import DBHandler
4 from .tables import Document
5 from bs4 import BeautifulSoup
6 import requests
7 import time
8
9 # Future implementation: Tuning features by adding Documents' to their content. Maybe with
  a multiplier.
10 class ContentImporter(object):
11     def __init__(self, db_name="klassify"):
12         self.DBH = DBHandler(db_name, echo=False)
13         self.ROOT_URL = "https://www.gov.uk"
14         self.NON_RELEVANT_PHRASES = [
15             "Skip to main content",
16             "Find out more about cookies"
17             "GOV.UK uses cookies to make the site simpler",
18             "Is there anything wrong with this page",
19             "Last updated",
20             "Other ways to apply",
21             "Before you start",
22             "Elsewhere on the web",
23             "Find out about call charges",
24             "find out more about beta services",
25             "Return to top ↑",
26             "Find out more about cookies",
27             "GOV.UK",
28             "Don't include personal or financial information",
29             "Help us improve",
30             "This file may not be suitable for users of assistive technology"
31             "If you use assistive technology and need a version of this document in a mor
e accessible format",
32             "tell us what format you need It will help us if you say what assistive techn
ology you use",
33             "Request a different format",
34             "What you were doing",
35             "What went wrong",
36             "uses cookies to make the site simpler."
37         ]
38
39     def parse_page(self, page):
40         soup = BeautifulSoup(page, 'html.parser')
41         return soup
42
43     def extract_page_content(self, page):
44         return page.text
45
46     # Iterate through each Document in the database, get their URL on the site and
47     # query it to obtain their HTML and eventually store it.
48     def import_documents_html(self):
49         documents = self.DBH.session.query(Document).all()
50
51         count = 0
52         for doc in documents:
53             if doc.html == None:
54                 time.sleep(0.75)
55                 doc.html = requests.get(doc.web_url).text
56                 self.DBH.session.commit()
57             count += 1
58             if count % 250 == 0: print("Documents processed: %d/%d" %(count, len(document
s)))
59
60     # Iterate through the Documents' HTML, parse it and store it.
61     def extract_documents_content(self):
62         documents = self.DBH.session.query(Document).all()
63

```

```

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_punctuation_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_punctuation_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

```

```

1 # Appendix C5 - feature_extractor.py
2
3 from nltk.tokenize import word_tokenize
4 from nltk.corpus import stopwords
5 from nltk.stem import PorterStemmer
6 import nltk
7
8 class FeatureExtractor():
9     def __init__(self, documents, n_features=5000):
10         self.documents = documents
11         self.stemmer = PorterStemmer()
12         self.vocabulary = self.top_words(n_features, self.freq_dist(self.make_vocabulary(
13         )))
14
15     def tokenize(self, document=None):
16         if document:
17             documents = [document]
18         else:
19             documents = self.documents
20
21         return [token for doc in documents for token in word_tokenize(doc.content)]
22
23     def process(self, vocabulary):
24         ADDITIONAL_STOP_WORDS = {'january', 'please', 'https', 'email', 'detail', 'email',
25         'send', 'if', 'december', 'october', 'kb', 'february', 'within', 'november', 'may', 'plea
26         se', '.mb', 'what', 'pdf', 'june', 'mach', 'good', 'august', 'september', 'html', 'july', '
27         beta', 'document', 'eg', 'published', 'april'}
28         stop_words = set(stopwords.words("english"))
29
30         processed_words = []
31         for word in vocabulary:
32             # select only words shorter than 20 char
33             if len(word) < 20:
34                 word = word.lower()
35                 # do not select stopwords
36                 if word not in (stop_words | ADDITIONAL_STOP_WORDS):
37                     # stem words
38                     word = self.stemmer.stem(word)
39                     # do not select words shorter than 2 characters
40                     if word.isalpha():
41                         if len(word) > 1:
42                             processed_words.append(word)
43                     else:
44                         processed_words.append(word)
45         return processed_words
46
47     def make_vocabulary(self, document=None):
48         if document:
49             vocabulary = self.tokenize(document)
50         else:
51             vocabulary = self.tokenize()
52
53         vocabulary = self.process(vocabulary)
54         return vocabulary
55
56     def bag_of_words(self, document):
57         doc_words = set(self.make_vocabulary(document))
58         bag_of_words = {}
59
60         for word in self.vocabulary:
61             bag_of_words[word] = (word in doc_words)
62
63         return bag_of_words
64
65     def freq_dist(self, vocabulary):
66         return nltk.FreqDist(vocabulary)
67
68

```

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



```

1 # Appendix C6 - doc_operator.py
2
3 from .db_handler import DBHandler
4 from .tables import Topic, Subtopic, Document
5 from .feature_extractor import FeatureExtractor
6 import random
7
8 class DocumentOperator():
9     def __init__(self, db_name="klassify", n=3, min_docs=None, max_docs=None, n_features=N
one):
10         self.DBH = DBHandler(db_name=db_name, echo=False)
11         self.topics = self.pick_random_topics(n, min_docs)
12         self.max_docs = max_docs
13         self.topic_labels = [topic.title for topic in self.topics]
14         self.docs_with_labels = self.docs_with_labels()
15         self.featuresets = []
16         self.processor = FeatureExtractor([doc for doc, cat in self.docs_with_labels], n_
features)
17
18     def pick_random_topics(self, n, min_docs):
19         topics = self.DBH.session.query(Topic).all()
20         if min_docs:
21             topics = [topic for topic in topics if len(topic.documents()) > min_docs]
22             random.shuffle(topics)
23             topics = topics[:n]
24         return topics
25
26     def find_random_doc_by_title(self, title):
27         topic = self.DBH.session.query(Topic).filter(Topic.title == title).first()
28         subtopic = random.choice(topic.subtopics)
29         return random.choice(subtopic.documents)
30
31     def random_document(self):
32         all_topics = self.DBH.session.query(Topic).all()
33         topic = random.choice(all_topics)
34         subtopic = random.choice(topic.subtopics)
35         doc = random.choice(subtopic.documents)
36         bag_of_words = self.baggify_document(doc)
37         return doc, bag_of_words
38
39     def docs_with_labels(self):
40         docs_with_filtered_labels = []
41
42         for topic in self.topics:
43             docs_with_labels = topic.documents_with_labels()
44
45             if self.max_docs:
46                 random.shuffle(docs_with_labels)
47                 docs_with_labels = docs_with_labels[:self.max_docs]
48
49             for doc, doc_labels in docs_with_labels:
50                 filtered_labels = []
51                 for label in doc_labels:
52                     # Filter out labels that are not the selected topics
53                     if label in self.topic_labels:
54                         filtered_labels.append(label)
55                 docs_with_filtered_labels.append([doc, filtered_labels])
56
57         return docs_with_filtered_labels
58
59     def build_feature_sets(self):
60         document_set_with_category = self.docs_with_labels
61         random.shuffle(document_set_with_category)
62
63         count = 0
64         for (document, category) in document_set_with_category:
65             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
```

```

1 # Appendix C7 - ovr_handler.py
2
3 from nltk import compat
4 from sklearn.naive_bayes import MultinomialNB
5 from sklearn.naive_bayes import BernoulliNB
6 from sklearn.multiclass import OneVsRestClassifier
7 from sklearn.preprocessing import MultiLabelBinarizer
8 from sklearn.feature_extraction import DictVectorizer
9 from sklearn import cross_validation
10 from sklearn.metrics import precision_score, recall_score, f1_score
11 from sklearn.cross_validation import train_test_split
12
13 class OvrHandler():
14     def __init__(self, featuresets):
15         self.mlb = MultiLabelBinarizer()
16         self.featuresets = featuresets
17         self._vectorizer = DictVectorizer(dtype=float, sparse=True)
18         self.X, self.y = self.prepare_scikit_x_and_y(self.featuresets)
19         self.classifiers = {
20             "MultinomialNB": OneVsRestClassifier(MultinomialNB()),
21             "BernoulliNB": OneVsRestClassifier(BernoulliNB()),
22         }
23
24     def prepare_scikit_x_and_y(self, labeled_featuresets):
25         X, y = list(compat.izip(*labeled_featuresets))
26         X = self._vectorizer.fit_transform(X)
27
28         set_of_labels = []
29         for label in y:
30             set_of_labels.append(set(label))
31
32         y = self.mlb.fit_transform(set_of_labels)
33
34         return X, y
35
36     def train_classifiers(self):
37         for name, clf in self.classifiers.items():
38             clf.fit(self.X, self.y)
39
40     def train_classifiers(self, X, y):
41         for name, clf in self.classifiers.items():
42             clf.fit(X, y)
43
44     def cross_validate(self):
45         results = {}
46         for name, clf in self.classifiers.items():
47             scores = cross_validation.cross_val_score(
48                 clf, self.X, self.y, cv=10
49             )
50             results[name] = {"cross score": scores.mean(), "cross variance": scores.std()
51 * 2}
52
53     def calculate_accuracy(self):
54         results = {}
55         X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, random_state=
56 0)
57
58         for name, clf in self.classifiers.items():
59             clf.fit(X_train, y_train)
60             y_pred = clf.predict(X_test)
61             proba_pos = clf.predict_proba(X_test)[:, 1]
62             precision = precision_score(y_test, y_pred, average='weighted')
63             recall = recall_score(y_test, y_pred, average='weighted')
64             f1 = f1_score(y_test, y_pred, average='weighted')
65
66             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

```

```
1 # Appendix C8 - build_and_train_classifiers.py
2
3 from src.doc_operator import DocumentOperator
4 from src.ovr_handler import OvrHandler
5 from src.measure_calculator import MeasureCalculator
6 import time
7
8 calc = MeasureCalculator()
9 start_time = time.time()
10
11 count = 1
12 while count <= 100:
13     doc_op = DocumentOperator(n=5, min_docs=400, max_docs=400, n_features=7500)
14     doc_op.build_feature_sets()
15
16     ovs = OvrHandler(doc_op.featuresets)
17
18     cross_validation_measures = ovs.cross_validate()
19     accuracy_measures = ovs.calculate_accuracy()
20
21     calc.add_measures(cross_validation_measures, accuracy_measures)
22
23     count += 1
24
25 calc.averaged_measures()
26
27 print("Total time: %0.2fs " % (time.time() - start_time))
28
```

```

1 # Appendix C9 - measure_calculator.py
2
3 class MeasureCalculator():
4     def __init__(self):
5         self.measures = {
6             "BernoulliNB": {
7                 "cross score": [],
8                 "cross variance": [],
9                 "precision": [],
10                "recall": [],
11                "f1": []
12            },
13            "MultinomialNB": {
14                "cross score": [],
15                "cross variance": [],
16                "precision": [],
17                "recall": [],
18                "f1": []
19            }
20        }
21
22    def add_measures(self, cross_validation_measures, accuracy_measures):
23        measures = self.combine_measures(cross_validation_measures, accuracy_measures)
24        for algo_type, results in measures.items():
25            for result, value in results.items():
26                self.measures[algo_type][result].append(value)
27
28    def combine_measures(self, cross_validation_measures, accuracy_measures):
29        current_measures = {}
30        current_measures["BernoulliNB"] = dict(
31            list(cross_validation_measures["BernoulliNB"].items()) +
32            list(accuracy_measures["BernoulliNB"].items())
33        )
34        current_measures["MultinomialNB"] = dict(
35            list(cross_validation_measures["MultinomialNB"].items()) +
36            list(accuracy_measures["MultinomialNB"].items())
37        )
38        return current_measures
39
40    def averaged_measures(self):
41        for algo_type, results in self.measures.items():
42            print(algo_type + ":")
43            cross_score = (sum(results["cross score"]) / len(results["cross score"]))
44            cross_precision = (sum(results["cross variance"]) / len(results["cross variance"]))
45
46            # Print out average of cross eval measure along with its variance
47            print("Cross evaluation accuracy: %1.3f (+/- %1.3f)" % (cross_score, cross_precision))
48
49            results.pop("cross score")
50            results.pop("cross variance")
51
52            for result, values in results.items():
53                # Print out averages of all remaining measures
54                print("%s: %1.3f" % (result, (sum(values) / len(values))))

```

```

1 # Appendix C10 - tables.py
2
3 from sqlalchemy import Table, Column, Integer, String, Text
4 from sqlalchemy import ForeignKey
5 from sqlalchemy.orm import relationship, backref
6 from .base import Base
7
8 class Topic(Base):
9     __tablename__ = 'topics'
10
11     id = Column(Integer, primary_key=True)
12     title = Column(String)
13     base_path = Column(String, unique=True)
14     description = Column(String)
15     web_url = Column(String)
16     api_url = Column(String)
17
18     def __repr__(self):
19         return "<Topic(title='%s', base_path='%s')>" % (self.title, self.base_path)
20
21     def documents(self):
22         documents = set()
23         for subtopic in self.subtopics:
24             for doc in subtopic.documents:
25                 documents.add(doc)
26
27         return list(documents)
28
29     def documents_with_labels(self):
30         doc_with_labels = []
31         for doc in self.documents():
32             doc_with_labels.append([doc, doc.topic_titles()])
33         return doc_with_labels
34
35 # create association table SubtopicDocument
36 subtopics_documents = Table('subtopics_documents', Base.metadata,
37     Column('subtopic_id', ForeignKey('subtopics.id'), primary_key=True),
38     Column('document_id', ForeignKey('documents.id'), primary_key=True)
39 )
40
41 class Subtopic(Base):
42     __tablename__ = 'subtopics'
43
44     id = Column(Integer, primary_key=True)
45     title = Column(String)
46     base_path = Column(String, unique=True)
47     description = Column(String)
48     web_url = Column(String)
49     api_url = Column(String)
50
51     topic_id = Column(Integer, ForeignKey('topics.id'))
52     topic = relationship("Topic", back_populates="subtopics")
53
54     documents = relationship(
55         "Document", secondary=subtopics_documents, back_populates="subtopics"
56     )
57
58     def __repr__(self):
59         return "<Subtopic(title='%s', base_path='%s')>" % (self.title, self.base_path)
60
61 # link topic to subtopics
62 Topic.subtopics = relationship(
63     "Subtopic", order_by=Subtopic.id, back_populates="topic"
64 )
65
66 class Document(Base):
67     __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, content=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

```



```
1 # Appendix C11 - base.py
2
3 from sqlalchemy.ext.declarative import declarative_base
4 Base = declarative_base()
5
```

```
1 # Appendix C12 - db_handler.py
2
3 from sqlalchemy import create_engine
4 from sqlalchemy.ext.declarative import declarative_base
5 from sqlalchemy.orm import sessionmaker
6 from .base import Base
7 import os
8
9 class DBHandler(object):
10     def __init__(self, db_name="klassify", echo=True):
11         self.db_name = db_name
12         self.db = "sqlite:///s.db" % self.db_name
13         self.engine = create_engine(self.db, echo=echo)
14         Session = sessionmaker(bind=self.engine)
15         self.session = Session()
16         Base.metadata.create_all(self.engine)
17
18     def destroy_db_if_present(self):
19         if os.path.exists("%s.db" % self.db_name):
20             print("Removing %s database" % self.db_name)
21             os.remove("%s.db" % self.db_name)
22
```

```

1 # Appendix C13 - test_content_importer.py
2
3 from klassify.src.tables import Document
4 from klassify.src.content_importer import ContentImporter
5 import os
6 import pytest
7
8 database_name = "test_klassify"
9 if os.path.exists("%s.db" % database_name):
10     os.remove("%s.db" % database_name)
11
12 DOCUMENT = Document(
13     base_path = "/intelligent-machines",
14     title = "The Intelligent Machines",
15     html = open("test/fixtures/document_page.html", 'r').read())
16 STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER = "How government works"
17 STRING_PRESENT_IN_SCRIPT_TAG = "<![CDATA["
18 STRING_PRESENT_IN_TITLE = "HM Revenue & Customs"
19
20 def setup_module(module):
21     global IMPORTER
22     IMPORTER = ContentImporter(db_name="test_klassify")
23     IMPORTER.DBH.session.add(DOCUMENT)
24     IMPORTER.DBH.session.commit()
25 def teardown_module(module):
26     IMPORTER.DBH.session.close()
27     IMPORTER.DBH.destroy_db_if_present()
28
29 def test_cleaning_methods():
30     doc = IMPORTER.DBH.session.query(Document).first()
31     page = IMPORTER.parse_page(doc.html)
32
33     assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER in page.text
34     assert STRING_PRESENT_IN_SCRIPT_TAG in page.text
35     page = IMPORTER.remove_unwanted_tags(page)
36     assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER not in page.text
37     assert STRING_PRESENT_IN_SCRIPT_TAG not in page.text
38
39     assert STRING_PRESENT_IN_TITLE in page.text
40     page = IMPORTER.get_body(page)
41     assert STRING_PRESENT_IN_TITLE not in page.text
42
43     page_content = IMPORTER.extract_page_content(page)
44     page_content = IMPORTER.remove_non_relevant_content(page_content)
45     for phrase in IMPORTER.NON_RELEVANT_PHRASES:
46         assert phrase not in page_content
47
48     assert "2016" in page_content
49     page_content = IMPORTER.remove_punctuation_and_numbers(page_content)
50     assert "2016" not in page_content
51
52 def test_extract_content_single_method():
53     doc = IMPORTER.DBH.session.query(Document).first()
54
55     assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER in doc.html
56     assert STRING_PRESENT_IN_SCRIPT_TAG in doc.html
57
58     clean_content = IMPORTER.extract_content(doc)
59
60     assert STRING_PRESENT_IN_BOTH_HEADER_AND_FOOTER not in clean_content
61     assert STRING_PRESENT_IN_SCRIPT_TAG not in clean_content
62     for phrase in IMPORTER.NON_RELEVANT_PHRASES:
63         assert phrase not in clean_content
64

```

```

1 # Appendix C14 - test_doc_operator.py
2
3 from klassify.src.doc_operator import DocumentOperator
4 from klassify.src.db_handler import DBHandler
5 from klassify.src.tables import Document, Subtopic, Topic
6 import os
7 import pytest
8
9 database_name = "test_klassify"
10 if os.path.exists("%s.db" % database_name):
11     os.remove("%s.db" % database_name)
12
13 def test_docs_with_labels():
14     document_1 = Document(title="Test title 1",
15                           base_path="/test-1",
16                           content="This is a test document - one")
17     document_2 = Document(title="Test title 2",
18                           base_path="/test-2",
19                           content="This is a test document - two")
20
21     topic_1 = Topic(
22         title='Label 1',
23         base_path='/topic/working-sea',
24         description='List of information about Topic.'
25     )
26     topic_2 = Topic(
27         title='Label 2',
28         base_path='/topic/working-sea-2',
29         description='List of information about Topic. 2'
30     )
31
32     subtopic_1 = Subtopic(
33         title='Subtopic',
34         base_path='/topic/working-sea',
35         description='List of information about Subtopic.'
36     )
37     subtopic_2 = Subtopic(
38         title='Subtopic 2',
39         base_path='/topic/working-sea-2',
40         description='List of information about Subtopic. 2'
41     )
42
43     topic_1.subtopics = [subtopic_1]
44     topic_2.subtopics = [subtopic_2]
45     subtopic_1.documents = [document_1]
46     subtopic_2.documents = [document_2]
47
48     DBH = DBHandler(db_name=database_name, echo=False)
49     session = DBH.session
50     session.add_all([topic_1, topic_2, subtopic_1, subtopic_2, document_1, document_2])
51     session.commit()
52
53     doc_op = DocumentOperator(db_name = database_name)
54
55     docs_with_labels = doc_op.docs_with_labels
56     first_set = docs_with_labels[0]
57     second_set = docs_with_labels[1]
58
59     assert [document_1.title, [topic_1.title]] in [[first_set[0].title, first_set[1]], [s
60 econd_set[0].title, second_set[1]]]
61     assert [document_2.title, [topic_2.title]] in [[first_set[0].title, first_set[1]], [s
62 econd_set[0].title, second_set[1]]]
63

```

```

1 # Appendix C15 - test_document_importer.py
2
3 from klassify.src.tables import Subtopic, Document
4 from klassify.src.document_importer import DocumentImporter
5 import json
6 import subprocess
7 import os
8 import pytest
9 import sqlalchemy
10
11 database_name = "test_klassify"
12 if os.path.exists("%s.db" % database_name):
13     os.remove("%s.db" % database_name)
14
15 with open('test/fixtures/tagged_documents.json', encoding='utf-8') as fixture_file:
16     api_response_fixture = json.loads(fixture_file.read())
17
18 DOCUMENT_DATA = api_response_fixture["results"][0]
19 SUBTOPICS = [
20     Subtopic(
21         base_path='/topic/driving-tests-and-learning-to-drive/car',
22         title='Cars'
23     ),
24     Subtopic(
25         base_path='/topic/driving-tests-and-learning-to-drive/lorry-bus',
26         title='Lorries and buses'
27     )
28 ]
29
30 def setup_module(module):
31     global IMPORTER
32     IMPORTER = DocumentImporter(db_name=database_name)
33 def teardown_module(module):
34     IMPORTER.DBH.session.close()
35     IMPORTER.DBH.destroy_db_if_present()
36
37 def test_total_documents():
38     assert IMPORTER.total_documents(api_response_fixture) == 9623
39
40 def test_pages():
41     assert IMPORTER.pages(9623) == 10
42
43 def test_urls():
44     assert IMPORTER.urls(10) == [
45         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=0',
46         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=1000',
47         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=2000',
48         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=3000',
49         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=4000',
50         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=5000',
51         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=6000',
52         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=7000',
53         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=8000',
54         'https://www.gov.uk/api/search.json?reject_specialist_sectors=_MISSING&count=1000&start=9000'
55     ]
56
57 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
```

```

1 # Appendix C16 - test_feature_extractor.py
2
3 from klassify.src.feature_extractor import FeatureExtractor
4 from klassify.src.tables import Document
5
6 initial_document_1 = Document(title="Test title 1",
7                               base_path="/test-1",
8                               content="This is a test document - one")
9 initial_document_2 = Document(title="Test title 2",
10                               base_path="/test-2",
11                               content="This is a test document - two")
12 initial_document_3 = Document(title="Test title 3",
13                               base_path="/test-3",
14                               content="This is a test document - three")
15
16 EXTRACTOR = FeatureExtractor([
17     initial_document_1,
18     initial_document_2,
19     initial_document_3,
20 ])
21
22 new_document = Document(title="Self assessment deadlines 3",
23                          base_path="/self-assessment-3",
24                          html="<strong>PAY NOW 3</strong>",
25                          content="This has a different content - four")
26
27 def test_tokenize():
28     tokenized_content = EXTRACTOR.tokenize(initial_document_1)
29     assert tokenized_content == ['This', 'is', 'a', 'test', 'document', "-", 'one']
30
31 def test_make_vocabulary():
32     # without document
33     assert EXTRACTOR.make_vocabulary() == ['test', 'one', 'test', 'two', 'test', 'three']
34     # with document
35     assert EXTRACTOR.make_vocabulary(new_document) == ['differ', 'content', 'four']
36
37 def test_bag_of_words():
38     # This is built against the vocabulary.
39     # The vocabulary is the sum of all the different terms in all the documents provided
40     # at instantiation.
41     assert EXTRACTOR.bag_of_words(initial_document_3) == {'one': False, 'test': True, 'three': True, 'two': False}
42     assert EXTRACTOR.bag_of_words(new_document) == {'one': False, 'test': False, 'three': False, 'two': False}
43
44 def test_process():
45     # What is bein discarded: Single letter words, Stop words, Long words
46     # Additionally, remaining words will be stemmed.
47     document_with_unfiltered_content = Document(title="Test", base_path="/test",
48         content=" within https .mb , a b c reallylongwordthatshouldbefilteredout cloudy regular words should be stemmed in this process")
49
50     tokenized_content = EXTRACTOR.tokenize(document_with_unfiltered_content)
51
52     assert EXTRACTOR.process(tokenized_content) == ['cloudi', 'regular', 'word', 'stem', 'process']
53

```

```

1 # Appendix C17 - test_measure_calculator.py
2
3 from klassify.src.measure_calculator import MeasureCalculator
4 from klassify.src.tables import Topic, Subtopic
5
6 first_set = {
7     "BernoulliNB": {
8         "cross score": 3, "precision": 1, "cross variance": 1
9     },
10    "MultinomialNB": {
11        "cross score": 2, "precision": 2, "cross variance": 2
12    }
13 }
14 second_set = {
15     "BernoulliNB": {"recall": 3, "f1": 1},
16     "MultinomialNB": {"recall": 2, "f1": 2}
17 }
18
19 # Groups two sets of measures by the algorithm type
20 def test_combine_measures():
21     CALC = MeasureCalculator()
22
23     assert CALC.combine_measures(first_set, second_set) == {
24         "BernoulliNB": {
25             "cross score": 3, "precision": 1, "recall": 3, "f1": 1, "cross variance": 1
26         },
27         "MultinomialNB": {
28             "cross score": 2, "precision": 2, "recall": 2, "f1": 2, "cross variance": 2
29         }
30     }
31
32 # Store sets of measures
33 def test_add_measures():
34     CALC = MeasureCalculator()
35
36     CALC.add_measures(first_set, second_set)
37
38     assert CALC.measures == {
39         "BernoulliNB": {
40             "cross score": [3], "precision": [1], "recall": [3], "f1": [1], "cross variance": [1]
41         },
42         "MultinomialNB": {
43             "cross score": [2], "precision": [2], "recall": [2], "f1": [2], "cross variance": [2]
44         }
45     }
46

```



```

1 # Appendix C18 - test_table_definition.py
2
3 from klassify.src.db_handler import DBHandler
4 from klassify.src.tables import Topic, Subtopic, Document
5 import pytest
6 import sqlalchemy
7
8 def test_db():
9     database_name = "test_klassify"
10
11     DBH = DBHandler(database_name, echo=False)
12     session = DBH.session
13     # create a topic, subtopic and document
14     test_topic = Topic(title="HMRC", base_path="/hmrc")
15     test_subtopic_1 = Subtopic(title="HMRC payments", base_path="/payments")
16     test_subtopic_2 = Subtopic(title="HMRC refunds", base_path="/refunds")
17     test_document_1 = Document(
18         title="Self assessment deadlines",
19         base_path="/self-assessment",
20         html="<strong>PAY NOW</strong>")
21     test_document_2 = Document(
22         title="Starting a business",
23         base_path="/start-business",
24         html="<strong>START NOW</strong>")
25     test_document_3 = Document(
26         title="Payment and refunds",
27         base_path="/payments-and-refunds",
28         html="<h1>payments and refunds</h1>")
29
30     # create relationships
31     test_topic.subtopics = [test_subtopic_1, test_subtopic_2]
32     test_subtopic_1.documents = [test_document_1, test_document_2]
33     test_document_3.subtopics = [test_subtopic_1, test_subtopic_2]
34
35     # add topic to session
36     session.add_all([
37         test_topic,
38         test_subtopic_1,
39         test_subtopic_2,
40         test_document_1,
41         test_document_2,
42         test_document_3
43     ])
44
45     session.commit()
46
47     # Table properties
48     assert session.query(Topic).get(test_topic.id).title == test_topic.title
49     assert session.query(Topic).get(test_topic.id).base_path == test_topic.base_path
50     assert session.query(Subtopic).get(test_subtopic_1.id).title == test_subtopic_1.title
51     assert session.query(Subtopic).get(test_subtopic_1.id).base_path == test_subtopic_1.b
52 ase_path
53     assert session.query(Document).get(test_document_1.id).title == test_document_1.title
54     assert session.query(Document).get(test_document_1.id).base_path == test_document_1.b
55 ase_path
56
57     # test relationships
58     topics_and_subtopics = session.query(Topic).get(test_topic.id).subtopics
59     subtopics_titles = [subtopic.title for subtopic in topics_and_subtopics]
60     assert test_subtopic_1.title in subtopics_titles
61     assert test_subtopic_2.title in subtopics_titles
62
63     subtopics_and_documents = session.query(Subtopic).get(test_subtopic_1.id).documents
64     documents_titles = [document.title for document in subtopics_and_documents]
65     assert test_document_1.title in documents_titles
66     assert test_document_2.title in documents_titles

```

```

66 documents_and_subtopics = session.query(Document).get(test_document_3.id).subtopics
67 subtopics_titles = [subtopic.title for subtopic in documents_and_subtopics]
68 assert test_subtopic_1.title in subtopics_titles
69 assert test_subtopic_2.title in subtopics_titles
70
71 # Test Document->Topics relation
72 doc = session.query(Document).get(test_document_1.id)
73 topic = session.query(Topic).get(test_topic.id)
74 assert topic in doc.topics()
75 assert topic.title in doc.topic_titles()
76
77 # Test Topic->Documents relation
78 doc = session.query(Document).get(test_document_1.id)
79 topic = session.query(Topic).get(test_topic.id)
80 assert doc in topic.documents()
81 assert doc, topic.title in topic.documents_with_labels()
82
83 # test unique constraint on basepath
84 clone_topic = Topic(title="Clone topic", base_path="/hmrc")
85 clone_subtopic = Subtopic(title="Clone subtopic", base_path="/refunds")
86 clone_document = Document(
87     title="Clone document",
88     base_path="/payments-and-refunds",
89     html="<h1>payments and refunds</h1>")
90 clones = [clone_topic, clone_subtopic, clone_document]
91 for clone in clones:
92     with pytest.raises(sqlalchemy.exc.IntegrityError):
93         session.rollback()
94         session.add_all([clone])
95         session.commit()
96
97 # terminate session and delete test db
98 session.close()
99 DBH.destroy_db_if_present()
100

```

```

1 # Appendix C19 - test_topic_importer.py
2
3 from klassify.src.topic_importer import TopicImporter
4 from klassify.src.tables import Topic, Subtopic
5
6 IMPORTER = TopicImporter()
7
8 def test_make_topic():
9     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'}
10
11     created_topic = IMPORTER.make_topic(topic_fixture)
12
13     expected_topic = Topic(
14         title='Working at sea',
15         base_path='/topic/working-sea',
16         web_url='https://www.gov.uk/topic/working-sea',
17         api_url='https://www.gov.uk/api/content/topic/working-sea',
18         description='List of information about Working at sea.'
19     )
20
21     assert created_topic.title == expected_topic.title
22     assert created_topic.base_path == expected_topic.base_path
23     assert created_topic.web_url == expected_topic.web_url
24     assert created_topic.api_url == expected_topic.api_url
25     assert created_topic.description == expected_topic.description
26
27 def test_make_subtopic():
28     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'}
29
30     created_subtopic = IMPORTER.make_topic(subtopic_fixture)
31
32     expected_subtopic = Subtopic(
33         title='Health and safety',
34         base_path='/topic/working-sea/health-safety',
35         web_url='https://www.gov.uk/topic/working-sea/health-safety',
36         api_url='https://www.gov.uk/api/content/topic/working-sea/health-safety',
37         description='List of information about Health and safety.'
38     )
39
40     assert created_subtopic.title == expected_subtopic.title
41     assert created_subtopic.base_path == expected_subtopic.base_path
42     assert created_subtopic.web_url == expected_subtopic.web_url
43     assert created_subtopic.api_url == expected_subtopic.api_url
44     assert created_subtopic.description == expected_subtopic.description
45
46 def test_associate_topic_subtopics():
47     topic = Topic(title="A topic title")
48     subtopic_1 = Subtopic(title="A subtopic title 1")
49     subtopic_2 = Subtopic(title="A subtopic title 2")
50
51     IMPORTER.associate_topic_subtopics(topic, [subtopic_1, subtopic_2])
52
53     assert subtopic_1.title == topic.subtopics[0].title
54     assert subtopic_2.title == topic.subtopics[1].title
55

```

```
1 # Appendix C20 - requirements.txt
2
3 pytest==2.8.7
4 SQLAlchemy==1.0.12
5 beautifulsoup4==4.4.1
6 responses==0.5.1
7 numpy==1.10.4
8 scipy==0.17.0
9 nltk==3.2
10 scikit-learn==0.17.1
11
```