Appendix C - Code

```python
# 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=_MI
SSING"
        self.PAGE_URL = "https://www.gov.uk/api/search.json?reject_specialist_sectors=_MI
SSING&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

    def run(self):
        root_data = self.api_response(self.ROOT_URL)
        number_of_documents = self.total_documents(root_data)
        pages = self.pages(number_of_documents)
        urls = self.urls(pages)

        count = 0
        duplicate_documents = []

        for url in urls:
            list_of_documents =  self.api_response(url)
            documents_data = list_of_documents['results']
            for document_data in documents_data:
                document = self.make_document(document_data)
                subtopics = self.find_subtopics(document_data)
                if subtopics:
                    self.associate_document_with_subtopics(document, subtopics)
                try:
                    self.DBH.session.add(document)
                    self.DBH.session.commit()
                except sqlalchemy.exc.IntegrityError:
                    duplicate_documents.append(document.base_path)
                    self.DBH.session.rollback()
                except:
                    self.DBH.session.rollback()
                    raise
                if count % 250 == 0: print("Documents processed: %d/%d" % (count, self.document_count))
                count = count + 1

        self.DBH.session.close()

        print("Documents with duplicates that have been ignored: %d" % len(duplicate_documents))

```

```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()

        count = 0
        for doc in documents:
            doc.content = self.extract_content(doc)
            self.DBH.session.commit()
            count += 1
            if count % 250 == 0: print("Documents processed: %d/%d" %(count, len(documents)))

    def extract_content(self, document):
        page = self.parse_page(document.html)
        page = self.remove_unwanted_tags(page)
        page = self.get_body(page)

        page_content = self.extract_page_content(page)
        page_content = self.remove_non_relevant_content(page_content)
        page_content = self.remove_punctuaction_and_numbers(page_content)
        return page_content

    def get_body(self, page):
        return page.body

    # Discard anything inside footer, header and scripts
    def remove_unwanted_tags(self, page):
        for tag in page.find_all(['footer', 'script', 'header']):
            tag.replace_with('')

        return page

    def remove_non_relevant_content(self, page):
        for phrase in self.NON_RELEVANT_PHRASES:
            page = page.replace(phrase, "")
        return page

    def remove_punctuaction_and_numbers(self, page):
        punctuation = [ '\\', '>', '_', '`', '{', ']', '*', '[',
                        '^', '+', '!', '(', ':', ';', "'", "’",
                        '<', '|', '"', '?', '=', '}', '&', '/',
                         '$', ')', '~', '#', '%', ',' ]
         page = ''.join(ch for ch in page if ch not in punctuation)
         page = ''.join([i for i in page if not i.isdigit()])
         return page
```

```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', 'please', '.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)

    def top_words(self, n_features, freq_dist):
        return list(freq_dist.keys())[:n_features]

```

```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
            self.featuresets.append([self.baggify_document(document), category])

    def baggify_document(self, doc):
        return self.processor.bag_of_words(doc)

```

```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}
        return results

    # Not used. For future implementation.
    # Feed a document's bag of word to this method to obtain recommended classes
    def predict_for_random(self, doc_with_bag_of_words):
        doc, bag_of_words = doc_with_bag_of_words
        print("Predicting for:", doc.title)
        print("Item is labeled to:")
        print(set(doc.topic_titles()))
        print("====> Predictions:")

        X = self._vectorizer.fit_transform(bag_of_words)

        for name, clf in self.classifiers.items():
            predicted_labels = (clf.predict(X))[0]
            probabilities =  clf.predict_proba(X)[0]
            named_classes = self.mlb.classes_

            print("Using %s:" % name)

            # If no labels are predicted for the item:
            if not 1 in predicted_labels:
                print("No label suggested for item")
                return

            for idx, label in enumerate(predicted_labels):
                confidence = round(float(probabilities[idx] * 100), 2)
                if confidence > 10:
                    print(named_classes[idx] + " - Confidence: ", end="")
                    print(str(confidence) + "%")
```

```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 variance"]))

            # Print out average of cross eval measure along with its variance
            print("Cross evaluation accuracy: %1.3f (+/- %1.3f)" % (cross_score, cross_precision))
            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'

    id          = Column(Integer, primary_key=True)
    title       = Column(String)
    base_path   = Column(String, unique=True)
    web_url     = Column(String)
    html        = Column(Text)
    description = Column(Text)
    content     = Column(Text)

    subtopics = relationship(
        'Subtopic', secondary=subtopics_documents, back_populates='documents'
    )

    def __init__(self, title, base_path, html=None, description=None, web_url=None, content=None):
        self.title       = title
        self.base_path   = base_path
        self.html        = html
        self.web_url     = web_url
        self.description = description
        self.content     = content

    def __repr__(self):
        return "<Document(title=%r, base_path=%r)" % (self.title, self.base_path)

    def topics(self):
        topics = set()

        for subtopic in self.subtopics:
            topics.add(subtopic.topic)

        return list(topics)

    def topic_titles(self):
        return [topic.title for topic in self.topics()]

```

```python
# Appendix C11 - base.py

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

```

```python
# 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)

```

```python
# 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"
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_punctuation_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

```

```python
# 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():
    made_document = IMPORTER.make_document(DOCUMENT_DATA)

    assert made_document.web_url == 'https://www.gov.uk/view-driving-licence'
    assert made_document.base_path == '/view-driving-licence'
    assert made_document.title == 'View or share your driving licence information'
    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)'

def test_associate_document_with_subtopics():
    made_document = IMPORTER.make_document(DOCUMENT_DATA)
    IMPORTER.associate_document_with_subtopics(made_document, SUBTOPICS)
    assert made_document.subtopics[0] in SUBTOPICS
    assert made_document.subtopics[1] in SUBTOPICS

def test_find_subtopics():
    session = IMPORTER.DBH.session
    session.add_all(SUBTOPICS)
    session.commit()

    found_topics = IMPORTER.find_subtopics(DOCUMENT_DATA)

    subtopics_titles = [subtopic.title for subtopic in SUBTOPICS]
    assert found_topics[0].title in subtopics_titles
    assert found_topics[1].title in subtopics_titles

```

```python
# 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, 'three': 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 regular 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

```