```
# 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
```