```
import csv
import os
import argparse
import json
import hnswlib
from sentence_transformers import SentenceTransformer

class hnswlibIndex:
  def __init__(self, file_path, model_name):
    self.file_path = file_path
    self.model = SentenceTransformer(model_name)
    self.org_dict = {}  # contains Org ID as key and all the data as value
    self.subject_dict = {} # contains subject_id as a key which contain another dictionary of subject name and a list of Org IDs with that subject
    self.index = None

  def load_csv(self):
    """ Load a CSV file into a list of dictionaries.
    Ideally we should use SQL query to fetch data straight from the SAcommunity database?
    This CSV file was provided by the web development team:
    2 type of dictionaries are created:
    1. org_dict: with Org ID as key and all the data as value
    2. subject_dict: with subject_id as a key which contain another dictionary of subject name and a list of Org IDs as value"""

    if os.path.exists(self.file_path) is False:
      raise FileNotFoundError(f"File not found: {self.file_path}")

    with open(self.file_path, mode='r', encoding='utf-8') as f:
      reader = csv.DictReader(f)  # Automatically creates dictionaries
      # Convert reader object to list of dictionaries
      data_list = list(reader)

    # Only include the columns we need, add more columns if needed
    keys_to_keep = ['Org ID', 'Org Name', 'AKA', 'Services', 'Subjects', 'Local Community dir', 'Primary Category']
    data_list = [{key: org[key] for key in keys_to_keep} for org in data_list]

    # add the Org ID key to each dictionary
    self.org_dict = {org['Org ID']: org for org in data_list}

    # Create another dictionary called subject_dict which will have subject ids as keys with the subject name and a list of Org IDs as values
    subject_dict = {}
```

```
for org in data_list:
    # Ensure 'Subjects' field is not empty before processing
    subjects = org['Subjects']
    if subjects:  # Proceed only if there are subjects
        subjects = subjects.split(';')

        for subject in subjects:
            subject = subject.strip()  # Clean extra spaces

            # Skip empty subjects
            if not subject:
                continue

            # If the subject is already in the dictionary, add the Org ID
            if subject not in subject_dict:
                # Use set to avoid duplicate Org IDs
                subject_dict[subject] = set()
            subject_dict[subject].add(int(org['Org ID']))

    # Add the Local Community dir and Primary Category as subjects
    local_subject = org['Local Community dir']
    primary_subject = org['Primary Category']
    if local_subject:
        if local_subject not in subject_dict:
            subject_dict[local_subject] = set()
        subject_dict[local_subject].add(int(org['Org ID']))
    if primary_subject:
        if primary_subject not in subject_dict:
            subject_dict[primary_subject] = set()
        subject_dict[primary_subject].add(int(org['Org ID']))

# convert the set() back to list before saving to JSON because JSON does not support set
for subject, org_ids in subject_dict.items():
    subject_dict[subject] = list(org_ids)

# Convert the subject_dict to a dictionary with index as key
for idx, (subject, data) in enumerate(subject_dict.items(), start=1):
    self.subject_dict[f'{idx}'] = {
        'Subject': subject,
        'Org_ids': data
    }

os.makedirs("database", exist_ok=True)
# Save the dictionaries as JSON files
# convert dictionary to list of dictionaries
```

```
    org_dict_path = "database/organisation_dict.json"
    subject_dict_path = "database/subject_dict.json"

    with open(org_dict_path, 'w') as f:
      json.dump(self.org_dict, f)

    with open(subject_dict_path, 'w') as f:
      json.dump(self.subject_dict, f)

    # Free up memory
    del data_list
    del subject_dict

  def create_index(self):
    """ Create a HNSW index using the SentenceTransformer model
    The index is saved on disk as hnsw_index.bin
    The SentenceTransformer model is saved on disk as embedding_model
    """
    # Create a HNSW index
    dim = self.model.get_sentence_embedding_dimension()
    self.index = hnswlib.Index(space='cosine', dim=dim)
    self.index.init_index(max_elements=20000,
                ef_construction=200, M=16)
    self.index.set_ef(50)

    # Prepare text & labels in a single step (more efficient)
    texts = [
      f"Organisation Name: {org['Org Name']}\nAKA: {org['AKA']}\nServices: {org['Services']}" for org in
self.org_dict.values()]
    texts += [data['Subject'] for data in self.subject_dict.values()]
    labels = [org['Org ID'] for org in self.org_dict.values(
    )] + list(self.subject_dict.keys())  # Use Org ID and Subject ID as labels

    # Batch encode texts
    print("Encoding texts")
    embeddings = self.model.encode(
      texts, batch_size=64, show_progress_bar=True)

    # Add all embeddings to the index at once
    print("Adding items to index")
    self.index.add_items(embeddings, labels)

    # Paths to save files
    hnsw_index_path = "hnsw_index.bin"
    model_save_path = "embedding_model"
```

```
        # Save the HNSW index and embeddings model on disk
        self.index.save_index("database/" + hnsw_index_path)
        self.model.save("database/" + model_save_path)

        # free up memory
        del texts
        del labels
        del embeddings

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--model_name',
                        type=str,
                        default='all-MiniLM-L6-v2',
                        help='model name')
    parser.add_argument('--input_file',
                        type=str,
                        help='input file name')
    args = parser.parse_args()

    print(f"Model Name: {args.model_name}")
    print(f"Input File: {args.input_file}")

    # process and create index
    index = hnswlibIndex(args.input_file, args.model_name)
    index.load_csv()
    index.create_index()
    print("Index created successfully")


if __name__ == "__main__":
    main()
```