Hidden Markov Model for Generating news based on collated Machine Learning articles

Author: Ajay Tom George
Student ID: 21251222

Introduction

- Create Transition matrix and generate news articles from the given data

How to run

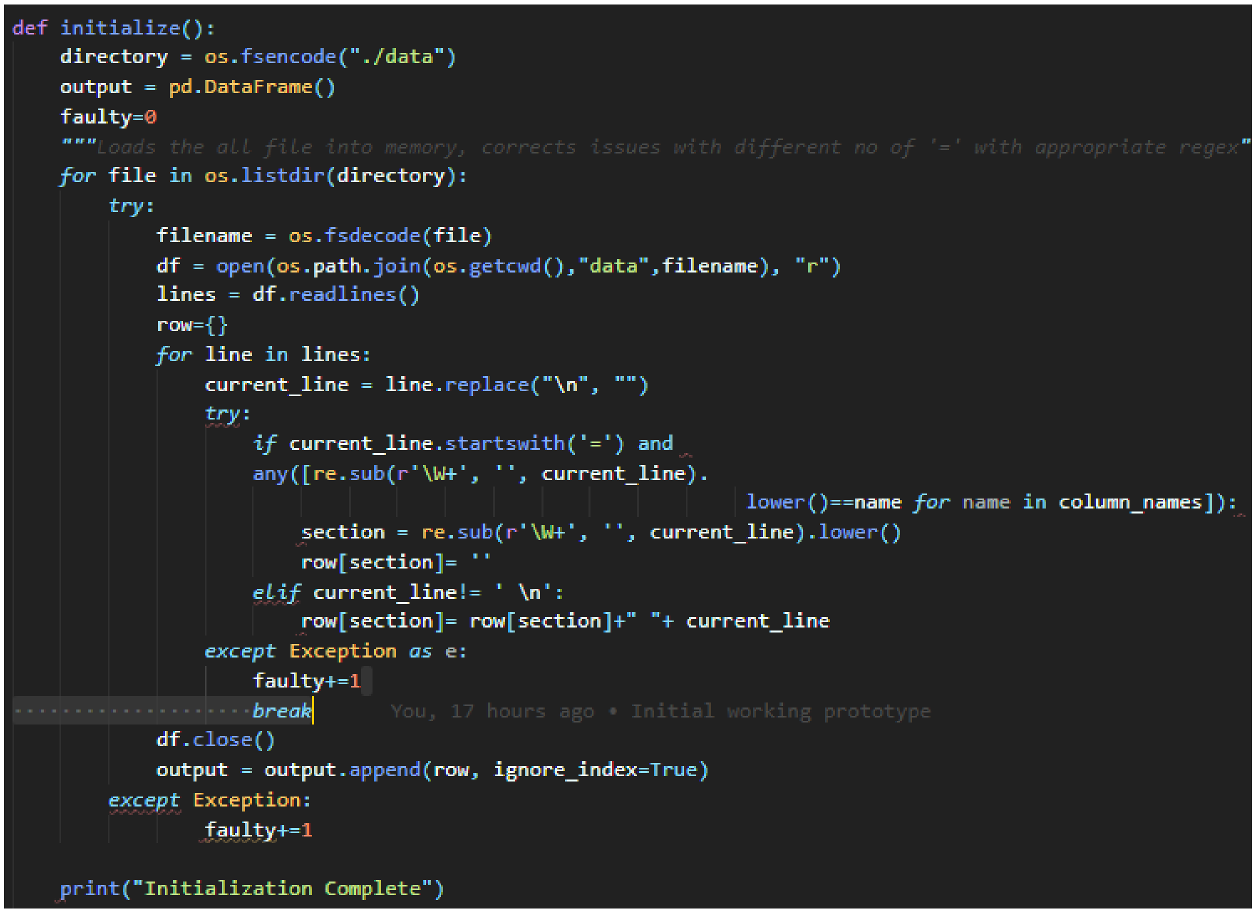
- Install libraries
    - *pip install pandas numpy nltk*
- Place ./data folder
- Run *python hmm_news.py*

Code explanation

Main Libraries

- **Nltk** for word processing

Stage 1 - Preprocessing - 1 Loading



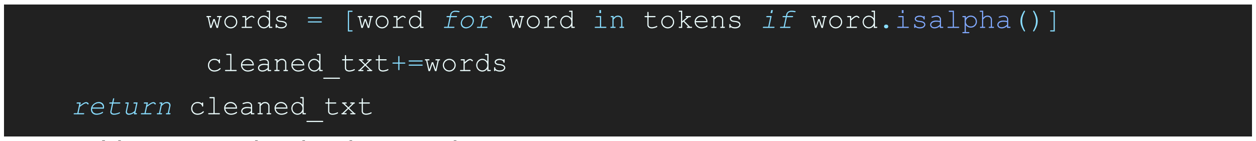
```python
def initialize():
    directory = os.fsencode("./data")
    output = pd.DataFrame()
    faulty=0
    """Loads the all file into memory, corrects issues with different no of '=' with appropriate regex"
    for file in os.listdir(directory):
        try:
            filename = os.fsdecode(file)
            df = open(os.path.join(os.getcwd(),"data",filename), "r")
            lines = df.readlines()
            row={}
            for line in lines:
                current_line = line.replace("\n", "")
                try:
                    if current_line.startswith('=') and 
                    any([re.sub(r'\W+', '', current_line).
                                                          lower()==name for name in column_names]):
                        section = re.sub(r'\W+', '', current_line).lower()
                        row[section]= ''
                    elif current_line!= ' \n':
                        row[section]= row[section]+" "+ current_line
                except Exception as e:
                    faulty+=1
                    break      You, 17 hours ago • Initial working prototype
            df.close()
            output = output.append(row, ignore_index=True)
        except Exception:
            faulty+=1

    print("Initialization Complete")
```

- Files are iterated one by one
- Add each section to dictionary
- If there is any pattern like ==*..source*, we will add it to section
- The regex handle the different number of =,
- “\n” are also replaced
- While with improper codecs are ignored

Stage 2 .Preprocessing - 2 Cleaning

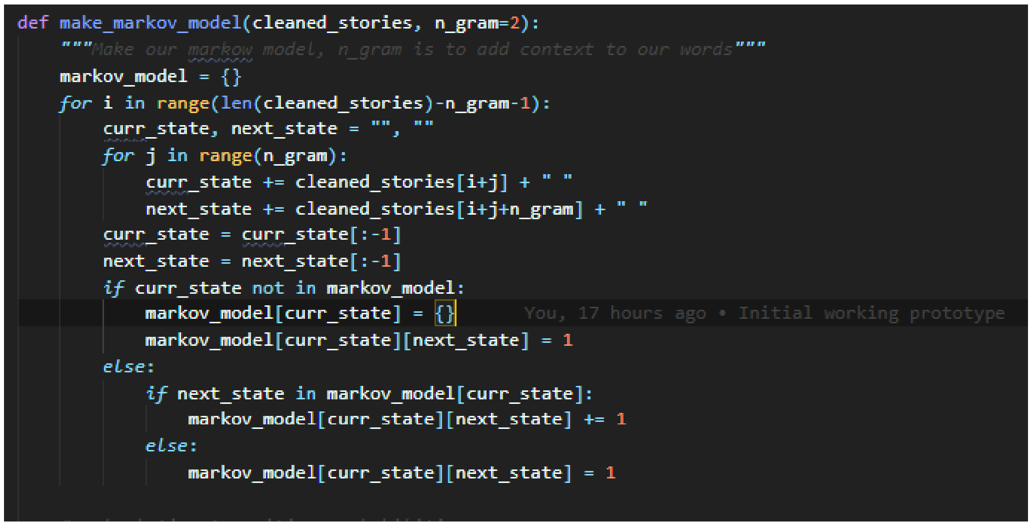
```python
def clean(txt):
    '''Further Clean the data of any special symbols and tokenise it'''
    cleaned_txt = []
    for line in txt:
        if isinstance(line, str):
            line = line.lower()
            line = re.sub(r"[,.\"\'!@#$%^&*(){}?/;`~:<>+=-\\]", "", line)
            tokens = word_tokenize(line)
```



```
            words = [word for word in tokens if word.isalpha()]
            cleaned_txt+=words
    return cleaned_txt
```

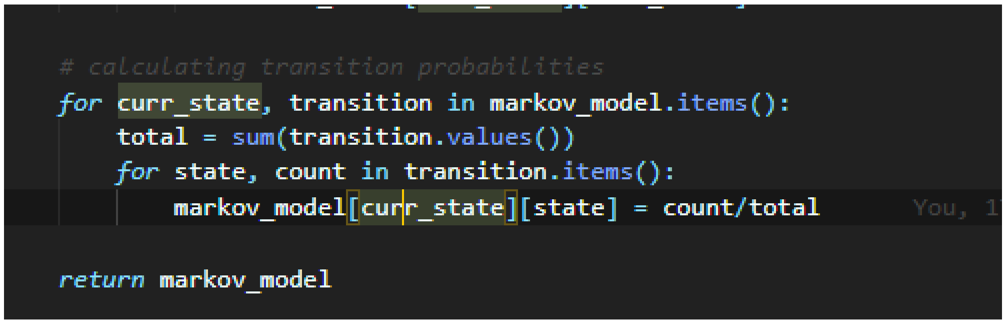
- Lines are tokenized to words
- Words are removed of any unwanted symbols
- Converting all letters to lowercase
- Only alphabets are retained

Stage 3. Creating Markov Model



```
def make_markov_model(cleaned_stories, n_gram=2):
    """Make our markow model, n_gram is to add context to our words"""
    markov_model = {}
    for i in range(len(cleaned_stories)-n_gram-1):
        curr_state, next_state = "", ""
        for j in range(n_gram):
            curr_state += cleaned_stories[i+j] + " "
            next_state += cleaned_stories[i+j+n_gram] + " "
        curr_state = curr_state[:-1]
        next_state = next_state[:-1]
        if curr_state not in markov_model:
            markov_model[curr_state] = {}        You, 17 hours ago • Initial working prototype
            markov_model[curr_state][next_state] = 1
        else:
            if next_state in markov_model[curr_state]:
                markov_model[curr_state][next_state] += 1
            else:
                markov_model[curr_state][next_state] = 1
```

- We will consider all words as states
- To give context, we will consider two words as 1 state
- These data are stored in a nested dictionary

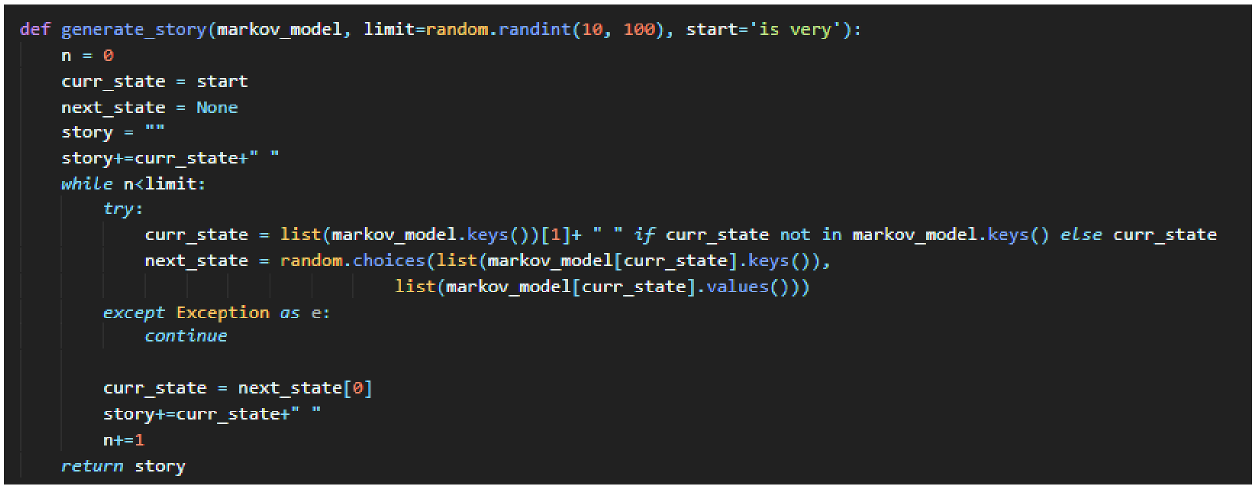


```
    # calculating transition probabilities
    for curr_state, transition in markov_model.items():
        total = sum(transition.values())
        for state, count in transition.items():
            markov_model[curr_state][state] = count/total        You, 1

    return markov_model
```

-
- We will calculate transition probabilities count/total, ie no of times paths / sum of all paths

Stage 4: Generate the news story

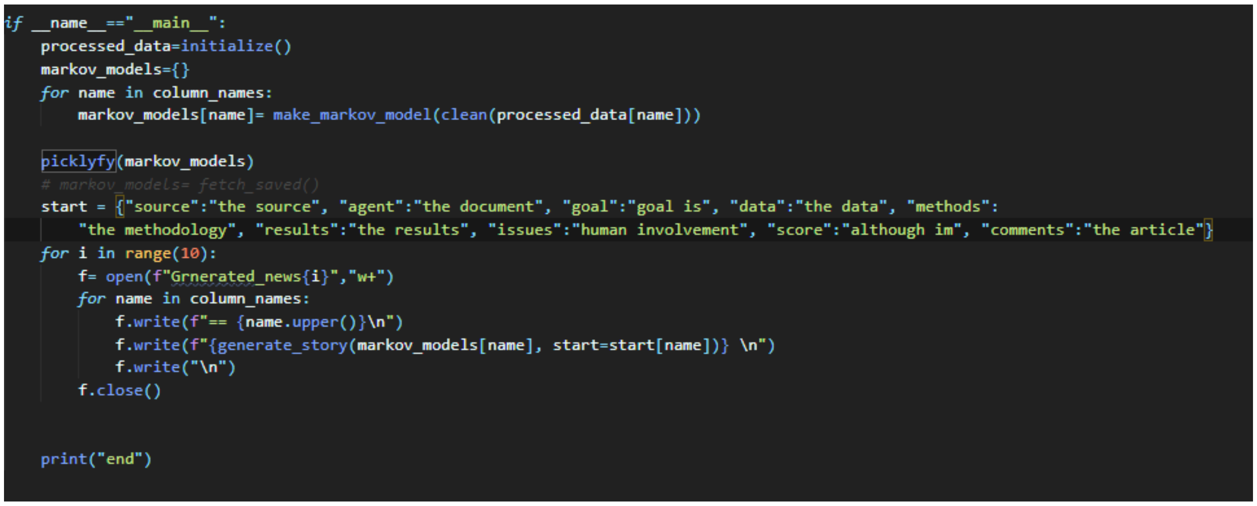


```python
def generate_story(markov_model, limit=random.randint(10, 100), start='is very'):
    n = 0
    curr_state = start
    next_state = None
    story = ""
    story+=curr_state+" "
    while n<limit:
        try:
            curr_state = list(markov_model.keys())[1]+ " " if curr_state not in markov_model.keys() else curr_state
            next_state = random.choices(list(markov_model[curr_state].keys()),
                                        list(markov_model[curr_state].values()))
        except Exception as e:
            continue

        curr_state = next_state[0]
        story+=curr_state+" "
        n+=1
    return story
```

- He we assign random start state and iterate through states based on limits(no of word pairs)

Stage 5: Main stage



```python
if __name__=="__main__":
    processed_data=initialize()
    markov_models={}
    for name in column_names:
        markov_models[name]= make_markov_model(clean(processed_data[name]))

    picklyfy(markov_models)
    # markov_models= fetch_saved()
    start = {"source":"the source", "agent":"the document", "goal":"goal is", "data":"the data", "methods":
        "the methodology", "results":"the results", "issues":"human involvement", "score":"although im", "comments":"the article"}
    for i in range(10):
        f= open(f"Grnerated_news{i}","w+")
        for name in column_names:
            f.write(f"== {name.upper()}\n")
            f.write(f"{generate_story(markov_models[name], start=start[name])} \n")
            f.write("\n")
        f.close()


    print("end")
```

- We initialize and clean data first
- Iterating through each dataframe column section like ***source, agent, comment*** and passing it to the markov model
- We save the models with pickle for future use
- To generate stories withuse the function generate story which starts from a random state and randomly generates next states based on transition probabilities from the previous step