ایمپورت کردن کتاب خانه های لازم:

```
import pandas as pd
from collections import Counter
from nltk import ngrams, pos_tag
from hazm import sent_tokenize, word_tokenize, stopwords_list, Normalizer
from nltk.probability import LaplaceProbDist, SimpleGoodTuringProbDist, FreqDist
```

مرحله صفر:

باز کردن فایل و تبدیل ستون کامنت به str

```
# Step 0: Open the Excel file
file_name = "digikala_comment.xlsx"
data_file = pd.read_excel(file_name)
data_file['comment'] = data_file['comment'].astype(str)
```

مرحله ۱-۱:

توکن کردن کامنت ها:

این مرحله با استفاده از hazm انجام شده است.

```
# Step 1: Preprocessing
# Step 1-1: Tokenize comments into sentences
print('step 1-1')
data_file['comment_sentences'] = data_file['comment'].apply(sent_tokenize)
```

مرحله ۱-۲:

حذف فضاهای خالی اضافی با استفاده از strip

```
# Step 1-2: Remove extra spaces
print('step 1-2')
data_file['comment_sentences'] = data_file['comment_sentences'].apply(lambda sentences: [sentence.strip() for sentence in sentences])
```

مرحله ۱-۳:

حذف استاپ ورد، علایم نگارشی و سایر موارد اضافی:

```
# Step 1-3: Tokenize, remove stop words, non-alphabetic characters, HTML tags, emojis, and normalize text
print('step 1-3')
stopwords = set(stopwords_list())
normalizer = Normalizer()
data_file['tokenized_comment'] = data_file['comment_sentences'].apply(lambda sentences: [
    [normalizer.normalize(word.replace('\u200c', ' ')) for word in word_tokenize(sentence) if word.isalpha() and word not in stopwords]
    for sentence in sentences
])
```

مرحله ۱-۴:

تبدیل اعداد به آدرس ها به جایگزین گفته شده:

```
# Step 1-4: Replace numbers with <NUM> and URLs with <URL>
print('step 1-4')
data_file['tokenized_comment'] = data_file['tokenized_comment'].apply(lambda sentences: [
    ['<NUM>' if word.isdigit() else '<URL>' if word.startswith(('http', 'www')) else word for word in sentence]
    for sentence in sentences
])
```

مراحل ۲-۱ ۲-۲ ۲-۳

ساخت n-gram:

```
# Step 2: Create language models
# Step 2-1: Create n-grams
print('step 2-1')
Comment Code
def get_ngrams(data, n):
    ngram_list = [ngrams(sentence, n) for sentence in data]
    ngrams_flat = [ng for sent in ngram_list for ng in sent]
    return ngrams_flat

unigrams = get_ngrams(data_file['tokenized_comment'], 1)
bigrams = get_ngrams(data_file['tokenized_comment'], 2)
trigrams = get_ngrams(data_file['tokenized_comment'], 3)
```

چاپ موارد مربوط به این مرحله:

```
# Print each n-gram count
print('Unigram Count:', len(unigrams))
print('Bigram Count:', len(bigrams))
print('Trigram Count:', len(trigrams))

# Print each n-gram
print('Sample Unigrams:', unigrams[:10])
print('Sample Bigrams:', bigrams[:10])
print('Sample Trigrams:', trigrams[:10])

# Step 2-2: Convert lists to tuples
flat_unigrams = [tuple(word) for gram in unigrams for word in gram]
flat_bigrams = [tuple(word) for gram in bigrams for word in gram]
flat_trigrams = [tuple(word) for gram in trigrams for word in gram]

freq_unigrams = FreqDist(flat_unigrams)
freq_bigrams = FreqDist(flat_bigrams)
freq_trigrams = FreqDist(flat_trigrams)
```

نتیجه چاپ:

روش Laplace smoothing برای n-gram ها به خوبی عمل نمی کند زیرا در این روش به هر کلمه یک تعداد ثابت از تعداد تکرارها اضافه می شود تا از احتمال صفر برای کلماتی که در داده ها دیده نشده اند، جلوگیری شود. این اضافه کردن تعداد ثابت به تمام کلمات ممکن است باعث افزایش تعداد تکرارها و در نتیجه افزایش احتمالات شود. این امر می تواند منجر به افزایش دقت نا همگن در تخمین احتمالات شود.

به طور مثال، اگر یک کلمه در داده ها دیده نشده باشد، روش Laplace smoothing باعث افزایش احتمال آن کلمه می شود، حتی اگر دیگر کلمات در متن با آن هماهنگ نباشند. این مشکل به خصوص برای n-gram هایی با تعداد زیادی از واژگان ممکن خودش را نشان می دهد.

محاسبه perplexity:

```python
import math
Comment Code
def calculate_perplexity(model, test_data):
    N = sum(len(sentence) for sentence in test_data)
    cross_entropy = -sum(math.log2(model.prob(word)) for sentence in test_data for word in sentence) / N
    perplexity = 2 ** cross_entropy
    return perplexity

test_sentences = [
    "بوی تند ولی خوشبو داره",
    "بلوتوثش کار نمی‌کنه حالا تا بدستم رسیده باید برش گردونم",
    "بلند گوهاش بیس بالا و صدای زیادی بمی‌داره که بعد از مدتی باعث خسته شدن مغز آدم می‌شه",
    "لطفاً کالای مورد نظر رو در پیشنهاد ویژه قرار بدید"
]

for i, sentence in enumerate(test_sentences, 1):
    laplace_unigram = LaplaceProbDist(freq_unigrams, bins=len(freq_unigrams))
    perplexity_unigram = calculate_perplexity(laplace_unigram, [sentence])

    laplace_bigram = LaplaceProbDist(freq_bigrams, bins=len(freq_bigrams))
    perplexity_bigram = calculate_perplexity(laplace_bigram, [sentence])

    laplace_trigram = LaplaceProbDist(freq_trigrams, bins=len(freq_trigrams))
    perplexity_trigram = calculate_perplexity(laplace_trigram, [sentence])

    print(f"Sentence {i} - Laplace Unigram Perplexity: {perplexity_unigram}")
    print(f"Sentence {i} - Laplace Bigram Perplexity: {perplexity_bigram}")
    print(f"Sentence {i} - Laplace Trigram Perplexity: {perplexity_trigram}")
    print('-' * 50)
```

گزارش این مرحله:

```
Sentence 1 - Laplace Unigram Perplexity: 82120.99999999996
Sentence 1 - Laplace Bigram Perplexity: 63562.999999999796
Sentence 1 - Laplace Trigram Perplexity: 56783.00000000005
------------------------------------------------
Sentence 2 - Laplace Unigram Perplexity: 82120.99999999873
Sentence 2 - Laplace Bigram Perplexity: 63563.000000000735
Sentence 2 - Laplace Trigram Perplexity: 56783.00000000054
------------------------------------------------
Sentence 3 - Laplace Unigram Perplexity: 82120.99999999975
Sentence 3 - Laplace Bigram Perplexity: 63563.00000000112
Sentence 3 - Laplace Trigram Perplexity: 56783.000000000684
------------------------------------------------
Sentence 4 - Laplace Unigram Perplexity: 82120.99999999894
Sentence 4 - Laplace Bigram Perplexity: 63563.000000000495
Sentence 4 - Laplace Trigram Perplexity: 56783.00000000054
------------------------------------------------
```

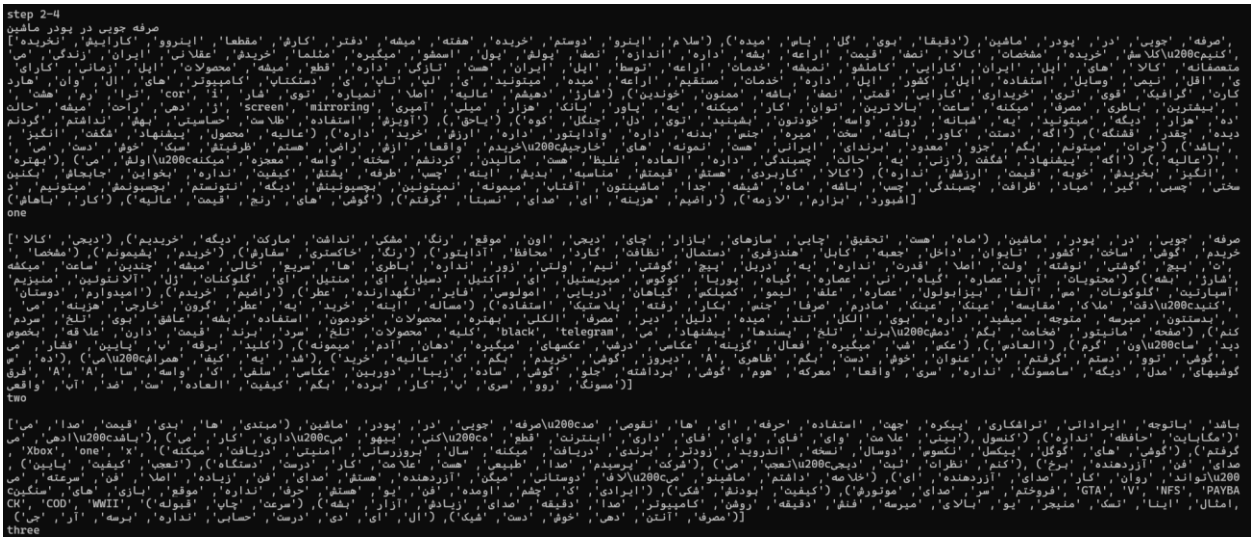
مرحله ۲-۴:

```python
print('step 2-4')
Comment Code
def predict_next_words(model, input_sequence, length=15):
    for i in range(length):
        next_word = model.generate()
        input_sequence.append(next_word)

    return input_sequence

for sentence in ["یک تن ماهی خوب", "رنگ قرمز کفش", "گوشی سامسونگ", "یکی از چراغهای وضعیت", "صرفه جویی در پودر ماشین"]:
    print(sentence)
    print(predict_next_words(laplace_unigram, sentence.split()))
    print('one \n')
    print(predict_next_words(laplace_bigram, sentence.split()))
    print('two \n')
    print(predict_next_words(laplace_trigram, sentence.split()))
    print('three \n')
```

خروجی این قسمت یک مقدار طولانی بود قسمتی از خروجی نمایش داده شده:



مرحله ۲-۵

```python
# Step 5-2: Calculate perplexity for generated sentences
print('step 5-2')
generated_sentences = [
    predict_next_words(laplace_unigram, "صرفه جویی در پودر ماشین".split()),
    predict_next_words(laplace_bigram, "یکی از چراغهای وضعیت".split()),
    predict_next_words(laplace_trigram, "گوشی سامسونگ".split()),
    predict_next_words(laplace_unigram, "رنگ قرمز کفش".split()),
    predict_next_words(laplace_trigram, "یک تن ماهی خوب".split())
]

perplexity_generated_unigram = calculate_perplexity(laplace_unigram, generated_sentences)
perplexity_generated_bigram = calculate_perplexity(laplace_bigram, generated_sentences)
perplexity_generated_trigram = calculate_perplexity(laplace_trigram, generated_sentences)

print(f"Perplexity of Generated Unigram Sentences: {perplexity_generated_unigram}")
print(f"Perplexity of Generated Bigram Sentences: {perplexity_generated_bigram}")
print(f"Perplexity of Generated Trigram Sentences: {perplexity_generated_trigram}")
```

خروجی این مرحله:

```
Perplexity of Generated Unigram Sentences: 38398.744332762778
Perplexity of Generated Bigram Sentences: 26728.870551571043
Perplexity of Generated Trigram Sentences: 24723.66982356164
```

مرحله ۳-۱:

```python
# Step 3: POS Tagging
# Step 3-1: Perform POS tagging on the preprocessed data
print('step 3-1')
data_file['pos_tags'] = data_file['tokenized_comment'].apply(lambda sentences: [pos_tag(sentence) for sentence in sentences])

with open('pos.txt', 'w', encoding='utf-8') as file:
    for sentence_tags in data_file['pos_tags']:
        for tags in sentence_tags:
            sentence = ' '.join([f"{word} --> {tag}" for word, tag in tags])
            file.write(sentence + '\n')
        file.write('\n')
```

خروجی کار(فقط بخشی از خروجی در اسکرین شات است، فایل خروجی نیز ارسال شده):

```
سلام --> JJ نظرم --> NNP یگم --> NNP می‌خواستم --> NNP موضوع --> NNP اشاره --> NNP نظراتی --> NNP کالاها --> NNP ثبت --> NNP می‌کنیم --> NNP هستن --> NNP ردم
بهتره --> JJ ک --> NNP کارشناسی --> NNP الکی --> NNP ندیم --> NNP NN
سال --> JJ پاور --> NNP بانک --> NNP دارم --> NNP نفر --> NNP اشنا --> NNP خریدم --> NNP پیشنهاد --> NNP خریدم --> NNP واقعا --> NNP عالیه --> NNP گذشت --> NNP
همزمان --> JJ پاور --> NNP adata --> NN همسرم --> NNP خریدم --> NNP اونم --> NNP دروغه --> NNP افت --> NNP کمتر --> NNP بازدهی --> NNP قارژ --> NNP داره --> NNP
تجربه --> JJ اطرافیانم --> NNP ک --> NNP شیائومی --> NNP کالایی --> NNP ک --> NNP می‌خرید --> NNP اصل --> NNP باشه --> NNP فیک --> NNP داره --> NNP واقعا --> NNP
ببخشید --> NN طولانی --> NN

بوی --> JJ تند --> JJ خوشبو --> NNP داره --> NNP ماندگاریش --> NNP خوبه --> NNP چقدر --> NNP میگذره --> NNP خوشبوتر --> NNP میشه --> NNP --> NN

عمر --> JJ مفید --> NNP ماشین --> NNP هست --> NNP سال --> NNP دیگه --> NNP پیاطری --> NNP کار --> NNP نمیکنه --> NNP گیر --> NNP میکنه --> NNP نحوی --> NNP --> NNP
```

مرحله ۳-۲:

تعداد رخداد هر تگ:

```python
# Step 3-2: Count occurrences of each POS tag
print('step 3-2')
pos_tags_flat = [tag[1] for sentence in data_file['pos_tags'].sum() for tag in sentence]
pos_tags_count = Counter(pos_tags_flat)
print(f"3-2: POS Tags Count: {pos_tags_count}")
```

خروجی:

```
step 3-2
3-2: POS Tags Count: Counter({'NNP': 426441, 'NN': 47401, 'JJ': 38203, 'FW': 684, 'VBZ': 402, 'VBD': 332, 'DT': 181, 'VBP': 117, 'VB': 110, 'NNS': 86, 'IN': 65, 'MD': 57, 'CC': 53, 'RB':
51, 'CD': 43, 'VBG': 39, 'PRP': 13, 'JJR': 12, 'RP': 10, 'TO': 8, 'VBN': 7, 'PRP$': 7, 'JJS': 2, 'SYM': 1, 'NNPS': 1, "''": 1, 'RBR': 1})
```

مرحله ۳-۳:

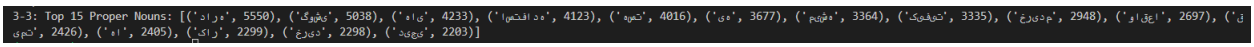
```python
# Step 3-3: Extract and count proper nouns (NNP)
print('step 3-3')
proper_nouns = [word[0] for sentence in data_file['pos_tags'].sum() for word in sentence if word[1] == 'NNP']
proper_nouns_count = Counter(proper_nouns)

print(f"3-3: Top 15 Proper Nouns: {proper_nouns_count.most_common(15)}")
```

خروجی:

```
3-3: Top 15 Proper Nouns: [ ,'داره')5550 ,'گوفی') ,(5038 ,'های') ,(4233 ,'استفاده') ,(4123 ,'هست') ,(4016 ,'یه') ,(3677 ,'میشه') ,(3364 ,'کیفیت') ,(3335 ,'خریدم') ,(2948 ,'واقعا') ,(2697(
, , )2426 ,'ها') ,(2405 ,'کار') ,(2299 ,'خرید') ,(2298 ,'دیجی') ,(2203(]
(env_hw_2) PS D:\sku\nlp\2>
```

چون کلمات فارسی هستن یکم خوانش پایین اومده این تو کنسول vscode



سوال ۱

- مرحله یک: تولید کاندید ها (Candidate Generation): در این مرحله، با استفاده از مدل های مختلف، یک لیست از کوئری های پیشنهادی برای یک پیشوند (prefix) خاص تولید می شود.

- مرحله دو: رتبه بندی کاندید ها (Candidate Ranking): کاندید ها بر اساس اهمیت و ارتباط با کوئری ورودی رتبه بندی می شوند. این مرحله با استفاده از مدل های یادگیری عمیق انجام می شود.

سوال ۲

- حافظه محدود: توانایی مدل در به خاطر سپاری کاندید ها و اطلاعات مرتبط با کاربر.

- کیفیت بالای پیش بینی: توانایی مدل در پیش بینی کاندید های با کیفیت بالا برای ورودی های مختلف.

- کارآیی و سرعت: اجرای سریع و کارآیی بالا در محیط های واقعی.

- بهبود بازیابی (Recall): افزایش تعداد کوئری های مناسب در لیست پیشنهادی.

- بهبود دقت (Precision): افزایش ارتباط کاندیداها با نیاز واقعی کاربر.

- مدیریت بازه زمانی (Latency): افزایش سرعت و کاهش تاخیر در تولید پیشنهاد ها.

- مدیریت منابع (Resource Management): بهره گیری بهینه از منابع محاسباتی برای اجرای بهتر سیستم.

سوال ۳

- توانایی در مدل سازی ارتباطات پیچیده: مدل های زبان عصبی قابلیت مدل سازی ارتباطات پیچیده بین کلمات را دارا هستند.

- قابلیت یادگیری از داده: مدل های زبان عصبی با توجه به حجم بالای داده ها، قابلیت یادگیری و به روز رسانی بهتری دارند.

- انعطاف پذیری در تطبیق با داده جدید: مدل های زبان عصبی قابلیت تطبیق با داده های جدید را دارا هستند و می توانند به تغییرات در زبان و استفاده کنندگان پاسخ دهند.

- مدل سازی همه جانبه (End-to-End Modeling): قابلیت یادگیری ارتباطات پیچیده درون کاندیداها و کوئری ورودی.

- قابلیت افزودن ویژگی های اضافی (Incorporating Additional Features): قابلیت اضافه کردن ویژگی های شخصی سازی مانند شناسه کاربر در مدل.

- مدیریت همسانی دنباله (Sequence Coherence Modeling): توانایی مدل سازی همسانی دنباله های کلمات.

سوال ۴

رویکرد MCG یا Maximum Context Generation که در این مقاله مطرح شده است، در فاز تولید کاندید ها بر اساس الگوریتم گریدی (Greedy Matching) بر روی کلمات پیشوندی ورودی عمل می کند. این الگوریتم از آخرین چندین کلمه پیشوندی به صورت گریدی (از طولانی ترین به کوتاه ترین) برای پیدا کردن مطلوب ترین کاندید ها برای تکمیل خودکار کوئری استفاده می کند.

سوال ۵

Unnormalized Language Model (LSTMEmb): یک مدل زبان عصبی بر اساس شبکه عصبی با ساختار LSTM برای محاسبه امتیاز رتبه بندی.

Frequency-Based Models (MPC, LWG, MCG): مدل های مبتنی بر فراوانی که بر اساس تکرار و فراوانی واژگان محاسبه می شوند.

Hybrid Models (NN+Frequency): مدل های ترکیبی که از اطلاعات فراوانی و مدل های عصبی برای رتبه بندی استفاده می کنند.

Convolutional Latent Semantic Model (CLSM): یک مدل عصبی مبتنی بر شبکه های عصبی کانولوشنی برای محاسبه امتیاز رتبه بندی.