Отчет по ДЗ №1

Кирщин Иван

№ 1.

Реализация лежит в ноутбуке hw1_n_gram_generation_Kirshchin.ipynb
Отсмотрев некоторое количество примеров я заметил, что шапка письма, то есть раздел со всей метаинформацией, всегда расположена в самом начале сообщения и отделена от текста письма через 2 абзаца. Таким образом сразу удалось избавиться от метаинформации.
Далее текст каждого письма нормализовывался следующим образом: удалялись все знаки препинания (так я решил сделать по той причине, что вряд ли получится генерировать разумную пунктуацию при помощи n-грамм, при этом лишняя пунктуация может повлиять на качество самих n-грамм, помимо влияния на качество генерации), все абзацы заменялись на пробелы, оставлялись только буквы английского алфавита, все буквы приводились к нижнему регистру.
По итогам обработки получился корпус текстов из приведенных к нижнему регистру текстов писем без метаинформации и знаков препинания, лишь слова в той последовательности, в которой они встречались в исходном письме.

№ 2.

Реализация лежит в tools/word_completor.py и в ноутбуке hw1_n_gram_generation_Kirshchin.ipynb
В данном задании было реализовано префиксное дерево. Сначала был реализован метод insert: для каждого нового слова мы спускаемся по дереву по существующим узлам или создавая новые идя по буквам в слове слева направо, доходя до конца слова проставляем флаг, что данный узел соответствует полноценному слову. Затем был реализован метод для нахождения всех слов, начинающихся на тот же префикс, search_prefix: в соответствии с буквами префикса спускаемся по узлам дерева, дойдя по конца префикса запускаем DFS в поддереве, где текущий узел является корнем, и сохраняем пути до всех узлов, маркированных как соответствующие полноценным словам.

№ 3.

Реализация лежит в tools/word_completor.py и в ноутбуке hw1_n_gram_generation_Kirshchin.ipynb
В данном задании был реализован класс WordCompletor, который формирует словарь и префиксное дерево, а так же умеет находить все возможные продолжения слова вместе с их вероятностями. Словарь с вероятностями формировался путем подсчета частоты слов в текстовом корпусе. Метод get_words_and_probs для префикса ищет все слова из словаря, начинающиеся на тот же префикс, и возвращает их вместе с подсчитанными на этапе инициализации вероятностями.

№ 4.

Реализация лежит в tools/n_gramm_model.py и в ноутбуке hw1_n_gram_generation_Kirshchin.ipynb
В данном задании был реализован класс для n-граммной модели. Словарь с вероятностями продолжений для каждой n-граммы формировался путем подсчета встречаемости частоты слов в текстовом корпусе после n-грамм. Метод get_next_words_and_probs для префикса из слов возвращает все слова, встречавшиеся после данного префикса, вместе с их подсчитанными на этапе инициализации вероятностями.

№ 5.

Реализация лежит в tools/text_suggestion.py и в ноутбуке hw1_n_gram_generation_Kirshchin.ipynb

В данном задании был реализован метод, возвращающий для префикса несколько вариантов продолжения фразы. Метод suggest_text устроен следующим образом: для пустого ввода не предлагается никаких продолжений; ввод нормализуется в соответствии с тем, как нормализовывались сообщения; если последним символом идет пробел, сразу переходим к предложению дальнейших слов и выбираем n_texts самых вероятных слов после имеющейся n-граммы, иначе выбираем n_texts самых вероятных продолжений префикса; затем n_words-1 раз для каждого из n_texts продолжений выбираем самое вероятное слово после текущей n-граммы и возвращаем получившийся текст.

Бонусная часть.

Реализация лежит в suggestions_app/suggestions_app.py
Демонстрация работы: ссылка
В данном задании был реализован UI для получения подсказок. Для генерации подсказок был встроен метод из предыдущего задания с n_texts=5 и n_words=3 на основе 3-грамм. В единственное окно на странице пользователь вводит запрос, и в это же время для каждого изменения строки вызывается метод set_query для обновления запроса и get_suggestions для генерации подсказок по текущему запросу, после чего подсказки на странице обновляются.