**Инструкция к сдаче**

1. Настоятельно рекомендуем сдавать практическое задание в виде ссылки на личный репозиторий на github.
2. Рекомендуемый способ организации данных в репозитории: создать отдельные папки по темам и помещать в них отдельные файлы для каждой задачи с правильным расширением.

Ссылка на инструкцию по работе с git и сдачу практики:

<https://docs.google.com/document/d/1RAT_ukE39iOfbz1xa39QXae2hBUEZ4U6Fko_wFDdrsM/edit>

Ссылка на видеокурс по Git:

<https://geekbrains.ru/courses/66>

Если остались сложности с системой git, то обратитесь к преподавателю или наставнику.

## Тема “Создание признакового пространства”

Продолжим обработку данных с Твиттера.

1. Создайте мешок слов с помощью sklearn.feature\_extraction.text.CountVectorizer.fit\_transform(). Применим его к 'tweet\_stemmed' и 'tweet\_lemmatized' отдельно.

* Игнорируем слова, частота которых в документе строго превышает порог 0.9 с помощью max\_df.
* Ограничим количество слов, попадающий в мешок, с помощью max\_features = 1000.
* Исключим стоп-слова с помощью stop\_words='english'.
* Отобразим Bag-of-Words модель как DataFrame. columns необходимо извлечь с помощью CountVectorizer.get\_feature\_names().

2. Создайте мешок слов с помощью sklearn.feature\_extraction.text.TfidfVectorizer.fit\_transform(). Применим его к 'tweet\_stemmed' и 'tweet\_lemmatized' отдельно.

* Игнорируем слова, частота которых в документе строго превышает порог 0.9 с помощью max\_df.
* Ограничим количество слов, попадающий в мешок, с помощью max\_features = 1000.
* Исключим стоп-слова с помощью stop\_words='english'.
* Отобразим Bag-of-Words модель как DataFrame. columns необходимо извлечь с помощью TfidfVectorizer.get\_feature\_names().

3. Натренируем gensim.models.Word2Vec модель на наших данных.

* Тренировать будем на токенизированных твитах combine\_df['tweet\_token']
* Установим следующие параметры: size=200, window=5, min\_count=2, sg = 1, hs = 0, negative = 10, workers= 32, seed = 34.
* Используем функцию train() с параметром total\_examples равным длине combine\_df['tweet\_token'], количество epochs установим 20.

4. Давайте немного потестируем нашу модель Word2Vec и посмотрим, как она работает. Мы зададим слово positive = "dinner", и модель вытащит из корпуса наиболее похожие слова c помощью функции most\_similar. То же самое попробуем со словом "trump".

5. Из приведенных выше примеров мы видим, что наша модель word2vec хорошо справляется с поиском наиболее похожих слов для данного слова. Но как она это делает? Она изучила векторы для каждого уникального слова наших данных и использует косинусное сходство, чтобы найти наиболее похожие векторы (слова).

Давайте проверим векторное представление любого слова из нашего корпуса, например "food".

6. Поскольку наши данные содержат твиты, а не только слова, нам придется придумать способ использовать векторы слов из модели word2vec для создания векторного представления всего твита. Существует простое решение этой проблемы, мы можем просто взять среднее значение всех векторов слов, присутствующих в твите. Длина результирующего вектора будет одинаковой, то есть 200. Мы повторим тот же процесс для всех твитов в наших данных и получим их векторы. Теперь у нас есть 200 функций word2vec для наших данных.

Необходимо создать вектор для каждого твита, взяв среднее значение векторов слов, присутствующих в твите. В цикле сделать: vec += model\_w2v[word].reshape((1, size))

и поделить финальный вектор на количество слов в твите.

На выходе должен получиться wordvec\_df.shape = (49159, 200).