Assignment 1. Document Embeddings in Numpy

10 марта 2021 г.

1 Введение

В этом задании вам предстоит реализовать на Numpy модель DV-ngram (Document Vector by predicting ngrams[1]), позволяющую строить низкомерные векторные представления документов (document embeddings), и использовать их вместо bag-of-ngrams векторов в качестве входного представления текста для классификатора тональности отзывов о фильмах.

2 Word2Vec SGNS – напоминание

В данном разделе приводятся краткие сведения об одной из моделей семейства word2vec, на базе которой строится модель DV-ngram, которую необходимо реализовать. Рассмотрим одну из моделей семейства word2vec – Skip-Gram Negative Sampling (SGNS), позволяющую строить низкомерные векторные представления слов (word embeddings). Для этого обучается бинарный классификатор, получающий на вход пару слов w, c (центральное слово и слово из контекста[1]) и возвращающий оценку вероятности того, что эти слова могут встретиться на расстоянии не более K в реальных текстах (K – гиперпараметр, называемый шириной окна). Далее обозначим через w центральное слово, $C(w)$ – набор слов в контексте слова w, и $c \in C(w)$. Наша задача оценить $P((w,c) \in D|(w,c))$, где D множество реальных текстов. В качестве классификатора возьмём

$$P(I_D = 1|(w,c);\theta) = \sigma(< v_w, v'_c >),$$

где v_w, v'_c – обучаемые векторы слов. $I_D = 1$ индикатор того, что пара встретилась в реальных текстах D. Векторы для центрального слова и слова из контекста берутся из разных матриц, что делает модель более адекватной. Так, редко когда одно и тоже слово встречается через небольшой интервал, поэтому вероятность P((собака, собака)) должна быть низкой. Этого, очевидно, трудно достичь, если мы используем один и тот же вектор. В целом, выбор модели обусловлен следующим: во-первых, мы хотим каждому слову сопоставить вектор; во-вторых, чем чаще слова встречаются вместе, тем большее скалярное произведение должны иметь их вектора. Может показаться, что все это не нужно и достаточно для каждой пары слов проверить, существует ли текст в котором они стоят рядом или нет: если есть, то $P(I_D = 1|(w,c);\theta) = 1$, иначе $P(I_D = 1|(w,c);\theta) = 0$. Однако, доступа ко всем возможным текстам у нас нет, поэтому модели предстоит научиться обобщать и записывать информацию о сочетаемости слов в их вектора. Далее под D подразумевается наш датасет. Конечно, чтобы обучать наш классификатор нам нужны ещё и отрицательные примеры, т.е примеры пар слов, которые вряд ли будут стоять вместе в реальных текстах. Обозначим через D' множество таких пар. Построим его следующим образом: для каждого положительного примера (w,c) из D возьмём k (гиперпараметр) отрицательных примеров (w,c'), где c' – случайные слова из некоторого распределения $P'(w)$. В качестве $P'(w)$ авторы работы предлагают брать $P'(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$, где $U(w)$ частота слова w в корпусе, Z – нормирующая константа. Степень $3/4$ уменьшает вероятность сэмплировать частотные, обычно не определяющие смысл слова, контексты (например, предлоги) и увеличивает вероятность сэмплировать более редкие, но содержательные контексты.

[1] По аналогии с другими моделями семейства word2vec одно из этих слов называется target/center word, а другое – context word, хотя в данной модели всё симметрично.

Итого наша оптимизационная задача выглядит так:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log(P(I_D = 1|(w,c);\theta)) + \sum_{(w,c')\in D'} \log(1 - P(I_D = 1|(w,c');\theta))$$
$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log\ \sigma(< v_w, v'_c >) + \sum_{(w,c')\in D'} \log(1 - \sigma(< v_w, v'_{c'} >))$$
$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log\ \sigma(< v_w, v'_c >) + \sum_{(w,c')\in D'} \log\ \sigma(- < v_w, v'_{c'} >)$$

Если вы не уверены, что полностью осознали, как и почему устроен SGNS, настоятельно рекомендуем краткое и доступное описание (Goldberg and Levy[2]).

3 DV-ngram

Возьмём представленную выше модель, но только будем не по паре слов учиться определять вероятность того, что они стоят рядом, а по паре (документ, n-грамма) учиться определять, содержится ли приведённая n-грамма в документе или нет. Данная задача решается тривиально, если в качестве входного представления документа используется bag-of-ngrams, однако модель DV-ngram использует вместо этого обучаемое низкомерное (размерности 500) представление документа (document embedding), инициализируемое в начале обучения случайными значениями. Модели придётся научиться сжимать информацию о том, какие из миллиона возможных n-грамм есть в документе, в сравнительно небольшой вектор. Например, n-граммам, появление которых коррелирует друг с другом, можно выдать близкие векторы (таким образом, высокую вероятность встретиться в неком документе будут получать не отдельные n-граммы, а сразу группы n-грамм, часто встречающихся вместе), чтобы независимые n-граммы могли получить ортогональные векторы.

Полученные таким способом векторы документов будем использовать вместо bag-of-ngrams представления в качестве входа логистической регрессии для классификации текстов IMDB. Мы рекомендуем воспользоваться классом LogisticRegression из библиотеки scikit-learn (популярная python библиотека, в которой реализованы многие алгоритмы машинного обучения). В этом случае достаточно подобрать силу L2-регуляризации (гиперпараметр C), для оптимизации будут использоваться методы второго порядка, не требующие подбора learning rate.

4 Теоретическая часть

1. Пусть, вы обучаетесь алгоритмом SGD и на данном шаге из датасета вы выбрали слово w и документ d. Тогда у вас есть один вектор слова v_w и один вектор документа v_d, а также метка y_i, которая равна 1 в случае, если слово w содержится в документе d, и 0, если не содержится. **Вопрос:**
   (a) Выпишите функцию потерь для этой пары в схеме skip-gram negative sampling.
   (b) Найдите градиент по весам v_w в терминах матричных операций.
   (c) Найдите градиент по весам v_d в терминах матричных операций.
2. Прочитайте практическую часть и вернитесь к выполнению задания. Кратко, задача состоит в том чтобы:
   (a) Завести 2 матрицы: одна для эмбедингов слов, вторая для эмбедингов документов.
   (b) Выбрать из первой матрицы batch_size эмбедингов слов и batch_size эмбедингов соответствующих документам.
   (c) Посчитать скалярные произведения векторов слов и соответствующих векторов документов.
   (d) Навесить функцию потерь на логиты.
   (e) Просуммировать потери на каждом примере в общую потерю. Это и будет ваш loss.

   Вопрос:

(a) Выпишите размерности матриц векторов слов и векторов документов.

(b) Выпишите размерности матриц после взятия подмножеств векторов отвечающих словам и документам из одного батча.

(c) Выпишите размерность вектора скалярных произведений, вектора потерь.

3. Рассмотрим случай, когда эмбединги слов обучаются по схеме word2vec skip-gram negative sampling, но отрицательные примеры не используются (то есть всем примерам из датасета соответствуют единичные метки). **Вопрос:** Как вам кажется, что будет происходить с векторами во время обучения?

4. Предположим, что мы хотим решать задачу классификации текстов на основе их эмбедингов. При этом у нас есть три корпуса данных: обучающая, валидационная и тестовая выборка. Сколько моделей DV-ngram нужно построить для получения эмбедингов? Можно ли использовать модель DV-ngram в режиме online (когда постоянно приходят новые тестовые данные)?

5 Практическая часть

1. Сделайте предобработку документов по аналогии с предыдущими заданиями.

2. Замените каждый документ на набор униграмм, биграмм и триграмм, которые в нём встречаются (каждая n-грамма входит в набор столько раз, сколько раз встречается в тексте).

3. Создайте словарь из индексов n-грамм и посчитайте кол-во вхождений в корпус каждой n-граммы. Сделайте отсечку по частоте встречаемости в корпусе, так чтобы осталось около миллиона элементов.

4. Возведите количество вхождений в корпусе каждой n-граммы в степень $3/4$. А затем нормализуйте так, чтобы получились вероятности.

5. Все документы представьте в виде двух массивов данных одинакового размера: в первый массив сохраните все индексы n-грамм для всех документов, во второй массив — индексы документов для соответствующих n-грамм в первом массиве. Перед генерацией батчей необходимо перемешать все n-граммы и все документы для лучшего обучения модели (получить перестановку индексов можно с помощью функции $numpy.random.permutation$, а затем применить ее к массиву n-грамм и массиву индексов документов).

6. Напишите функцию $batch\_generator(words\_idxs, docs\_idxs, probs, nb = 5, batch\_size = 100)$, $words\_idxs$ - массив индексов n-грамм в корпусе, $docs\_idxs$ – массив индексов документов для соответсвующих n-грамм , nb – количество негативных батчей на один позитивный, $probs$ – вероятности соответствующие каждому слову в словаре, $batch\_size$ – размер батча. Данная функция должна возвращать генератор батчей для обучения модели. Один батч данных – это 3 массива размера $batch\_size$. Первый массив состоит из индексов n-грамм (индексы в словаре). Второй массив состоит из индексов документов соответствующих n-граммам из первого массива. Третий состоит из меток 0 и 1, 1 означает что i-ая n-грамма из первого массива взята из i-ого документа из второго массива, а 0 – что это случайная n-грамма. Батчи должны генерироваться следующим образом:

   (a) Генерируем один позитивный батч – то есть выбираем из длинного массива очередные $batch\_size$ n-грамм и $batch\_size$ документов, которых содержатся соответствующие n-граммы.

   (b) После этого генерируем 5 батчей с нулевой меткой – сэмплируем случайные n-граммы из словаря пропорционально их вероятностям, а документы оставляем те же самые, меняем метки с 1 на 0. Сэмплирование можно производить с помощью функции $numpy.random.choice$, которая принимает на вход массив, элементы из которого будут сэмплироваться, а также вероятности соответствующие каждому элементу.

(c) Если сэмплировать негативные примеры на каждом шаге, то это может занять намного больше времени, чем если сделать это в самом начале (для этого надо просэмплировать $nb * len(words\_idxs)$ случайных n-грамм).

7. Далее создадим класс Doc2Vec, который в конструкторе на вход будет принимать размер словаря, количество документов и размер эмбедингов(по умолчанию **500**). В конструкторе создадим две матрицы: $word\_embs$ - матрица эмбедингов всех n-грамм из словаря, и doc_embs - матрица эмбедингов всех документов. Теперь созданные матрицы эмбедингов нужно инициализировать случайными значениями(по умолчанию значениями **от -0.001 до 0.001**).

8. Затем напишите метод $train(words\_idxs, docs\_idxs, labels, lr)$ класса Doc2Vec, который принимает на вход один батч из train выборки, и обновляет веса эмбеддингов слов и документов. Опираясь на формулы из теоретической части, необходимо посчитать градиенты для каждого примера из батча независимо, а затем обновить веса для каждого примера независимо. Важно заметить, что мы не последовательно считаем градиенты для каждого примера, а сразу фиксируем точку в которой считаем градиенты, а применяем их уже последовательно.

9. Напишите функцию, которая будет принимать эмбединги и метки документов из train и dev выборок, обучать логистическую регрессию на обучающей и возвращать ее точность на обучающей и валидационной выборках. Рекомендуем воспользоваться классом LogisticRegression из sklearn. Перед обучением необходимо осуществлять автоматический подбор веса L2-регуляризации с помощью 10-фолдовой кросс-валидации на обучающей выборке (см. класс GridSearchCV из sklearn) – в процессе обучения DV-ngram оптимальное значение C может меняться. Убедитесь, что выбранный C находится не на границе перебора.

10. Напишите цикл обучения модели по эпохам. На каждой эпохе обучаем модель на всей выборке, затем обучаем логистическую регрессию на эмбедингах документов. **Вопрос.** Нарисуйте графики того, как изменяется значение оценочной функции DV-ngram и точность логистической регрессии на train/dev выборках в процессе обучения. Что можно сказать про переобучение/недообучение модели?

11. **Вопрос**. Опишите, какие гиперпараметры модели и каким образом подбирались? Какая получилась итоговая точность на train/dev/test выборках? Как зависит точность от значения гиперпараметров?

12. **Вопрос**. Какие трудности возникли при выполнения задания?

6 Исследовательская часть

1. Скорость обучения и качество финальной модели может зависеть от того, каким образом составляются батчи. Сравните следующие варианты с реализованным ранее: 1) в каждом батче – позитивные и негативные примеры для одних и тех же документов 2) в каждом батче – абсолютно случайные позитивные и негативные примеры. Приведите график с кривыми обучения для каждого варианта. Для каждого варианта может быть свой оптимальный learning rate, желательно нарисовать графики для нескольких learning rate и убедиться, что оптимальный находится не на границе перебора.

2. Известно, что комбинирование bag-of-ngrams векторов документов и их векторов, построенных моделью DV-ngram, позволяет существенно улучшить качество классификации. Попробуйте сконкатенировать эти вектора и обучить логистическую регрессию. Сравните, как ведут себя при комбинировании разные варианты bag-of-ngrams: бинарный, небинарный, с NB-весами.

3. Насэмплируйте достаточно большое количество пар:

   - векторов документов
   - векторов слов

- вектор слова, вектор документа

Для каждого из 3 случаев постройте гистограмму скалярных произведений. Заметили ли вы что-нибудь необычное?

4. Какие небольшие изменения надо внести в ваш код, чтобы можно было обучать эмбединги слов по схеме w2v skip-gram negative sampling?

5. После внесения изменений из пункта 4, допустим, что у нас есть датасет D:

[Учить NLP достаточно интересно;
Я стараюсь тратить всё свободное время, чтобы учить NLP;
Мама не обязана мыть раму в свой выходной, она тоже хочет заниматься NLP!]

Насэмплируем на каждое слово по 2 слова для создания отрицательных примеров. Предположим, что у нас насэмплировались следующие отрицательные пары (w, c') (датасет D'):

[(не, в),
(учить, NLP),
(мыть, NLP)
(учить, стараюсь)
(учить, интересно)
(тратить, выходной)
(учить, раму)]

Остальные пары вам надо досэмплировать самостоятельно. Обучите w2v с шириной окна $K = 1$ и размерностью векторов равной 500. К чему будет стремится $P((\text{учить}, \text{NLP}) \in D|(\text{учить}, \text{NLP}))$ в процессе обучения? Приведите экспериментальное и теоретическое обоснование.

6. Если Вы задумаетесь, то модель w2v skip-gram negative sampling оптимизирует следующую функцию потерь:

$$\sum_{(i,c)\in D} \big(log(\sigma(\hat{v}_c^T v_i)) + kE_{c'\sim P(w)} log(\sigma(-\hat{v}_{c'}^T v_i))\big),$$

где i – некоторое слово из словаря, c – некоторое слово из контекста слова i, $P(w)$ – некоторое распределение для сэмплирования негативных пар. Будем считать, что сэмплируем мы пропорционально частоте появления слова в датасете. Пусть k - количество негативных примеров, которое сэмплируется в процессе обучения, v_c, v_i – векторы слов, n – количество слов в датасете, n_c – количество раз, которое слово встретилось в датасете, $n_{c,i}$ – количество раз, которое пара слов (c, i) встретилась в датасете на расстоянии не более выбранной ширины окна.

(a) Замените $\sum_{(i,c)\in D}$ на двойную сумму $\sum_c \sum_i$. В этом вам поможет введеная ранее величина $n_{c,i}$.

(b) Разбейте на 2 суммы то, что получилось на предыдущем шаге: с матожиданием и без. Можно заметить, что сумма с матожиданием не зависит от с. Преобразуйте эту сумму, используя то, что $\sum_c n_{i,c} = n_i$.

(c) $E_{c'\sim P(w)} log(\sigma(-\hat{v}_{c'}^T v_i)$ – матожидание с.в. с дискретным распределением. Распишите его по определению и подставьте в вашу формулу.

(d) Рассмотрим некоторую конкретную пару слов $\hat{i}, \hat{c}$. Выберете из суммы слагаемые, которые соответствуют именно этой паре и замените $\hat{v}_{\hat{c}}^T \hat{v}_{\hat{i}}$ на x. Прооптимизируйте по x. Получилось ли у вас, что $x = \hat{v}_{\hat{c}}^T \hat{v}_{\hat{i}} = \log(\frac{n_{c,i} n}{n_c n_i}) - \log k$? Похоже ли полученное выражение на pointwise mutual information? Что будет, если применить это к модели doc2vec?

(e) Допустим, что у вас есть небольшой датасет и вы посчитали матрицу $\log(\frac{n_{i,c}n}{n_c n_i}) - \log k$ для всех пар слов. На что было бы логично заменить $-\infty$ в матрице? Какое матричное разложение и как можно было бы использовать для обучения векторов слов и векторов контекстов? Как, используя выбранное вами матричное разложение, строить эмбединги определенной размерности? Почему именно так?

7. Вопросы аналогичные пункту 5, но теперь в начале каждой эпохи датасет D' сэмплируется заново с вероятностями пропорциональными частоте слов.

Список литературы

[1] Document Vector by predicting ngrams https://arxiv.org/abs/1512.08183

[2] Goldberg and Levy word2vec explained https://arxiv.org/abs/1402.3722