Методология экспериментальной работы с тематическими моделями в BigARTM

Мурат Апишев
great-mel@yandex.ru
MelLain@github.com

МГУ им М.В. Ломоносова

30 ноября 2016

О чём будет идти речь

- Большое количество проектов.
- Требуется много исполнителей-экспериментаторов.
- Нужно быстро обучаться. **Чему?**
- Научится пользоваться BigARTM — важно! Но для этого есть туториалы.
- Ещё важнее уметь правильно организовать свой эксперимент.

Зачем всё это

Правильная подготовка экспериментов позволяет (или может позволить) добиться:

1. улучшения получаемых результатов
2. ускорения экспериментов $\Rightarrow$ увеличения их количества $\Rightarrow$ п. 1
3. воспроизводимости экспериментов
4. удобной интерпретации и презентации результатов
5. возможности понимания и повторения эксперимента другими людьми с минимальным участием экспериментатора.

Структура изложения

1. **Данные**:
   - парсинг;
   - анализ;
   - предобработка;
2. **Моделирование в BigARTM**:
   - подготовка кода (парсинг, моделирование, журналирование);
   - построение моделей (структурные параметры, регуляризаторы и модальности);
3. **Извлечение и анализ результатов**

В качестве примера используются последние эксперименты в этничном проекте.

Почему это очень важно

«Сейчас быстро перегоню тексты в Vowpal Wabbit, и надо пробовать строить модели. . . »

Если эксперимент пилотный или тестовый — OK.

Если эксперимент в рамках проекта — так делать **НЕЛЬЗЯ**.

Невнимательное отношение к данным может привести к печальным последствиям (не только в TM, но и вообще в ML).

Для правильного построения дальнейшей работы важно:

- корректно преобразовать данные в понимаемый Вами формат.
- изучить структуру данных, статистические характеристики и проч.

Парсинг. Форматы данных

Форматы данных бывают какие угодно:

- База данных или её дамп.
- Наборы текстовых файлов или один файл с текстами.
- То же самое в json/xml/protobuf...
- Могут быть вообще сырые pdf, html...

Внутри каждого описанного типа данные могут иметь совершенно произвольный вид $\Rightarrow$ **написать универсальный парсер данных в Vowpal Wabbit невозможно.**

Задача парсинга ложится на пользователя.

Парсинг. Совет

Сразу: речь пойдёт о «живых» данных, а не об академических датасетах, вроде данных с UCI.

Два варианта:

- Структура известна: заказчик сообщил относительно полную информацию о том, как устроены данные, в каком они формате.
- Структура известна только частично или неизвестна — **так часто бывает**.

Даже в первом случае данные почти всегда не полностью удовлетворяют заявленной структуре.

Всегда на них нужно внимательно посмотреть.

Парсинг. Совет

А чем вообще можно смотреть в данные?

1. Можно прямо из Python (`with [codecs.]open(data_filename ...`). При поиске закономерностей удобно использовать *регулярные выражения* (модуль `re`)

   Помните: для стандартных форматов: json, xml, дампы баз данных — есть готовые модули для работы, не тратьте время на изобретение велосипедов.

2. В Linux/Mac OS можно пользоваться разными терминальными командами: `less`, `head`, `tail`.

3. В Windows удобно использовать графический редактор EmEditor (позволяет открывать гигантские файлы, таблицы, умеет работать с tab-ами и запятыми).

Парсинг. Пример

Заказчик выдал дамп базы данных с известной схемой.

Каждая строка — один документ.

У каждого документа есть слова, геотег, url, метка времени, возраст автора.

Кажется, что всё хорошо. Но если даже бегло посмотреть внутрь:

- сразу видно, что слова и по группам, и внутри групп разбиваются запятыми — нужно аккуратно считывать.
- данные наверняка неполные — у многих документов те или иные поля отсутствуют.

Парсинг. Пример

Начинаем делать парсинг в Python. **Важно:** если данные большие, и преобразований будет много — лучше делать их по-этапно, проверяя и сохраняя промежуточные результаты.

Данные надо поправить так, чтобы:

- все компоненты легко отделялись друг от друга (раздельные сеператоры внутри групп и между группами).
- все неполные данные были либо дополнены какими-то значениями, либо удалены из выборки.
- все ненужные данные должны быть удалены.

Если данных много, и из общей схемы парсинга выбиваются доли процента от числа документов — можно их выбросить и не тратить время.

Парсинг. Пример

В данных может быть излишняя детализация.

Например, url документа. Нет смысла возиться с каждым отдельным адресом, их много и они не повторяются.

Лучше вытащить хост и использовать его как модальность, если нужно.

`http://www.liveinternet.ru/users/ykecet/post305639491/`
`http://vk.com/wall69939430_1688`
`http://vk.com/wall-23469754_128778?reply=128805`
`http://vk.com/wall58881050_8574`

$\Rightarrow$ `liveinternet.ru`, `vk.com`.

Парсинг. Пример

Важно внутри одной модальности выдерживать общий формат.

При моделировании меток времени, все метки должны быть строго в одном формате.

При моделировании геотегов:

Москва, москва, москва_город, Москва_Столица и т.п.

— **не годится**.

Нужно строить соответствия и сливать группы разных написаний одного и того же.

Все данные лучше сразу привести в нижний регистр (если это не критично для задачи).

Анализ коллекции. Общие слова.

Итак, все данные после парсинга корректные, приведены к одному формату и понятны экспериментатору.

Следующий этап — анализ коллекции.

Необходим для понимания данных и выбора корректной стратегии обработки.

Базовые вещи, на которые всегда надо смотреть:

- Число документов в коллекции.
- Объём словаря по каждой модальности.
- Распределение длин документов, средняя длина документа.
- Суммарная длина коллекции.
- Распределение документов по специальным модальностям (метки времени, геотеги).

Анализ коллекции. Замечание.

НО!

Прежде, чем всё это делать, коллекцию надо *лемматизировать*.

Лемматизация — процесс приведения словоформы к лемме — её нормальной (словарной) форме. (Википедия)

Для английского языка не так критично, как для русского.

`санками` ⇒ `санки`
`катаясь` ⇒ `кататься`

Модули с лемматизаторами в Python: `pymorphy`, `pymystem`

Если заказчик уже использовал лемматизатор, обязательно узнавайте, какой именно!

Анализ коллекции. Лемматизация.

Пример кода на Python, выполняющего лемматизацию слова:

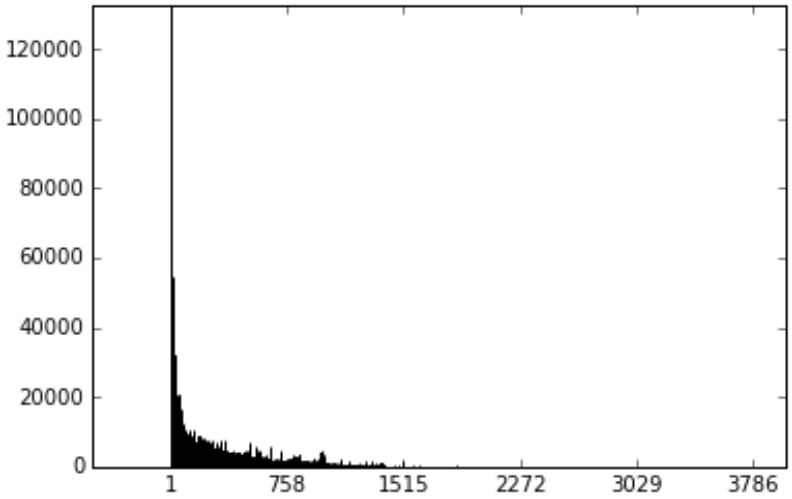
```
1  import pymystem3
2  mystem = pymystem3.Mystem()
3  lemma = mystem.lemmatize(token)[0]
```

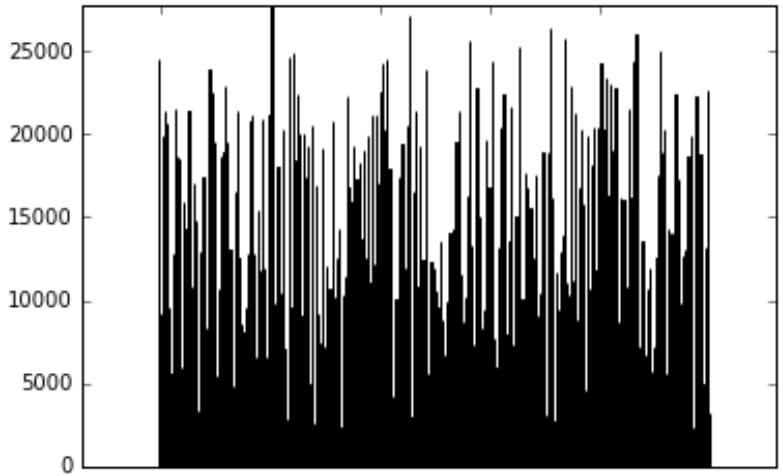
После лемматизации по коллекции можно собирать *n-граммы.*

Биграммы можно добавлять в основной словарь, разделив слова спец. символом, отсутствующим в Ваших данных:

русский_общение
украинский_родной
засылать_казачок
русский_больница

Анализ коллекции. Пример.





- Распределение длин документов — много коротких, но длинных всё равно достаточно.
- Распределение числа документов с заданной меткой времени — колоссальных разрывов нет.

Анализ коллекции. Пример.

Словарь коллекции может иметь огромные размеры — десятки миллионов.

Столько моделировать сложно и, как правило, бессмысленно.

Построив гистограмму частот слов, можно отрезать длинный низкочастотный хвост.

На слова с высокими частотами лучше смотреть глазами — как правило, это стоп-слова, и от них надо избавляться.

Помимо этого стоит выкидывать слишком коротки слова, слишком длинные, состоящие из цифр, спец. символов и т.п.

Тут тоже очень могут помочь регулярные выражения.

Анализ коллекции. Пример.

```
1  def special_match_eng(strg):
2      search=re.compile(u'^[a-z]+$').search
3      return bool(search(strg))
4
5  lower_bound = 600
6  upper_bound = 8e+5
7  min_token_len = 4
8  max_token_len = 18
9
10 filtered_dictionary = {}
11 for k, v in dictionary.iteritems():
12     if (v > lower_bound and v < upper_bound and
13             len(k) >= min_token_len and
14             len(k) <= max_token_len) and
15             special_match_eng(k):
16         filtered_dictionary[k] = v
```

Анализ коллекции. Пример.

После того, как фильтрация была завершена, надо снова пересчитать все статистики.

Если фильтрация слишком сильная или недостаточная — надо переделать.

Как только всё станет OK — можно записать итоговый файл в формате VW.

Можно сохранить как последовательный текст, так и «мешок слов» (см. документацию).

Первое может потребоваться для специализированных моделей, для подсчёта совстречаемостей и т.п.

Какой нужен код

Не будем разбирать API BigARTM.

Требуется:

- Распарсить файл VW в батчи и базовый словарь.
- Изменить словарь, если это нужно.
- Написать код для журналирования результатов экспериментов.
- Написать сам код для моделирования.

Написание кода. Пример.

```
1 def create_model(num_topics,
2                   class_ids,
3                   num_processors,
4                   num_document_passes,
5                   regularizer_strings,
6                   cache_theta=False):
```

Стоит описать такую функцию. Она создаёт модель, добавляет в неё все необходимые метрики, регуляризаторы из список и возвращает ссылку на объект ARTM.

1. меньше кода.
2. удобнее.
3. меньше шансов ошибиться и проще искать ошибку и модернизировать.

Написание кода. Пример.

```
1 def save_results(model, params, res_file,
2                  elapsed_time, regularizer_strings=[]):
3     with codecs.open(ress_file, 'w', 'utf-8') as fout:
4         fout.write(u'batch_size:{}\n'.format(batch_size))
5         ...
```

Стоит описать такую функцию. Она будет принимать на вход модель и её параметры и сохранять в файл с уникальным именем все результаты: параметры модели, топ-слова, значения метрик и т.п.

Преимущества использования функции те же.

Моделирование. Пример.

Внимательно и понятно описывайте все параметры.

```
model_name = 'artm_200_topics_geo_1_time_1_eb_10'

params = {}
params['num_processors'] = 10
params['num_topics'] = 200
params['num_collection_passes'] = 12
params['num_document_passes'] = 5

params['class_ids'] = {
                        '@default_class': 1.0,
                        '@ethnic_class': 10.0,
                        '@bigram_class': 10.0,
                        '@time_class': 1.0,
                        '@geo_class' : 1.0,
                      }
regularizer_strings = []
```

Моделирование. Пример.

Моделирование. Всё прозрачно и легко модифицируемо:

```
model = create_model(
    num_topics=params['num_topics'],
    class_ids=params['class_ids'],
    num_processors=params['num_processors'],
    num_document_passes=params['num_document_passes'],
    regularizer_strings=regularizer_strings)

time_start = time.time()
for i in xrange(params['num_collection_passes']):
    model.fit_offline(batch_vectorizer=bv)
    print 'Time:{0}, Perp:{1}'.format(
        time.time() - time_start,
        model.score_tracker['perplexity'].last_value)

save_results(model,params,'{}_res.txt'.format(model_name),
             time.time() - time_start, regularizer_strings)
```

О подборе параметров

Параметры бываюи структурные:

- Число батчей и документов в батчах;
- Число потоков-обработчиков;
- Число проходов по коллекции/документу;
- Тип алгоритма;
- Параметры алгоритма (если онлайн).

Или обычные:

- Наборы регуляризаторов и их параметров;
- Наборы модальностей и их параметров;

О подборе структурных параметров

- Число потоков обработчиков выбирается исходя из возможнойстей экспериментальной машины.
- Число батчей должно быть кратно числу потоков.
- Размер батча — не слишком маленьким (порядка тысяч документов, сотен, если документы очень большие). Но и не слишком большим.
- Тип алгоритма — оффлайн проще, онлайн – круче.
- Параметры алгоритма — чёткой методики нет, можно перебором.
- Число тем — регуляризатор отбора тем или эвристические соображения.

О подборе обычных параметров

Не надо пихать в модель сразу все регуляризаторы!

Легче добавлять по одному, оптимизируя τ. При этом надо всегда понимать, зачем именно регуляризатор добавляется в модель и как он примерно работает.

- Сглаживание/разреживание.
- Декоррелятор.
- Частичное обучение.
- Модальности.

Подбирать параметры можно с помощью grid search или random search.

Относительные коэффициенты регуляризации: `gamma=0.5` — можно перебирать τ от 0 до 1 (только Φ). **Медленнее!**

Типы результатов

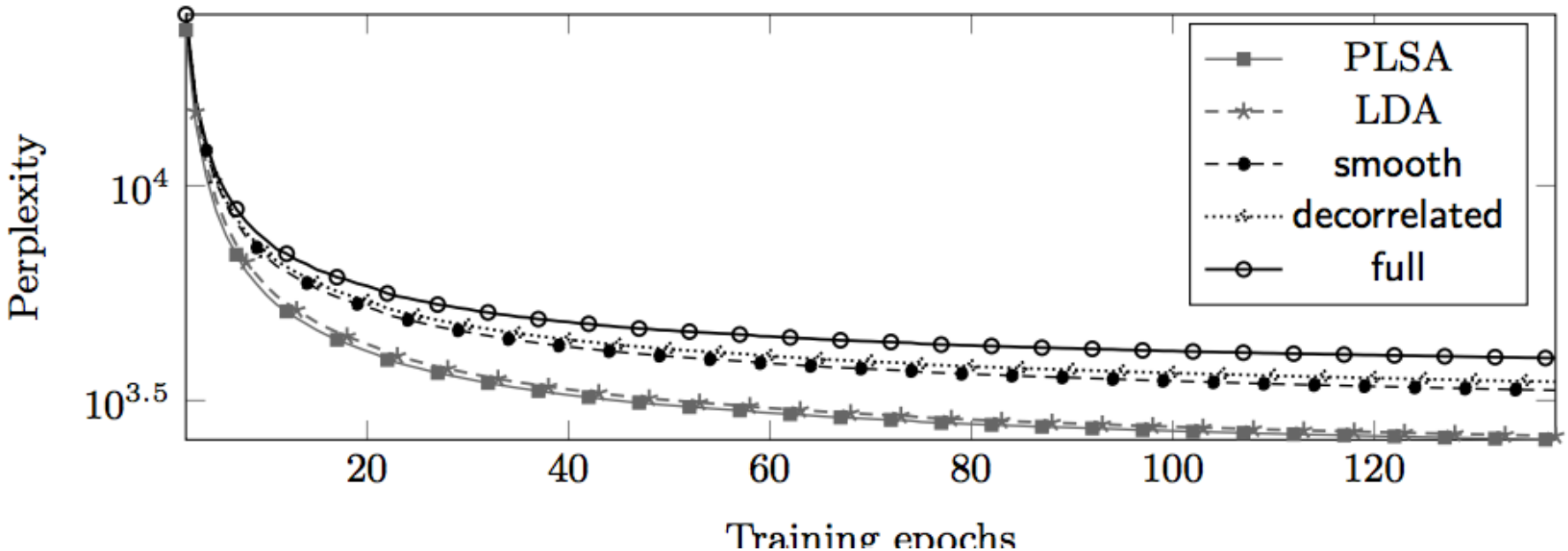
- Перплексия и другие числовые метрики.
- Топ-слова в темах.
- Документы (топ-документы надо извлекать).

Извлечение топ-документов для большой коллекции:

1. Обучили модель без сохранения Θ;
2. Идём в цикле по батчам и подаём их в `ARTM.transform()` (просим извлечь `dense_theta`).
3. Получив Θ для очередного батча, анализируем её (максимум по столбцам, например).
4. Закончив обработку, удаляем Θ для текущего батча, переходим к следующему.

Графики

Числовые метрики и понимать, и презентовать лучше всего с помощью графиков:



Топ-слова и документы придётся просматривать глазами.

Презентация результатов

Для визуализации специальных модальностей можно пользоваться разнообразными инструментами.

Геотеги — наложить на реальную карту.

Метки времени — нарисовать график изменения темы во времени.

Ещё можно строить гистограммы.

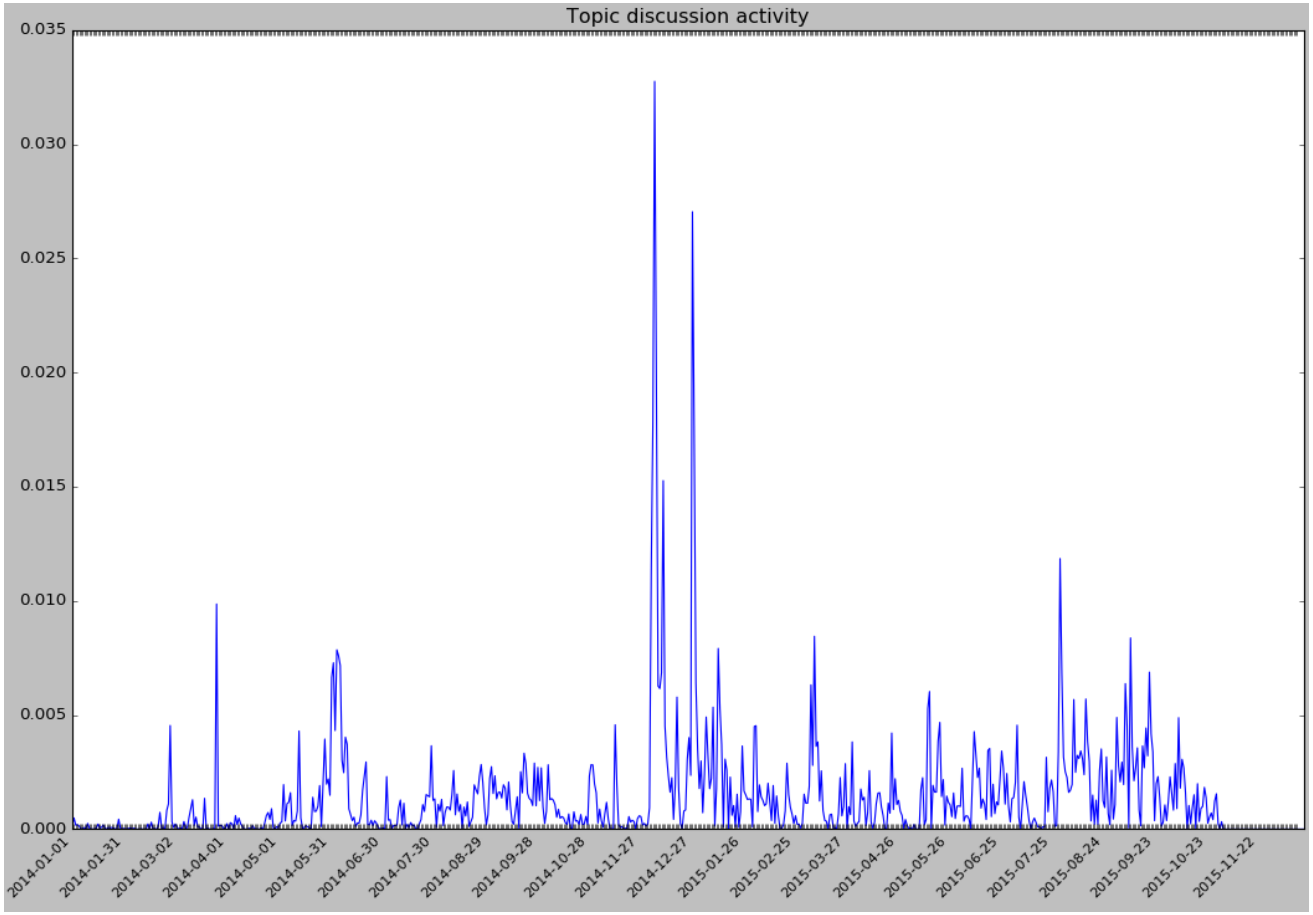
Важно: визуализация всегда нагляднее других способов (хоть и не всегда полнее).

В Python много средств визуализации. Почти всегда можно подобрать что-то с нуля за 2-3 часа.

Можно рисовать в Matlab или в LateX (tikz).

Пример графика изменения темы во времени

Это тема про войну в Чечне (пик на 20-тилетнюю годовщину):



Выводы

- Внимательно изучайте Ваши данные.
- Не ленитесь делать качественную предобработку.
- Вдвойне не ленитесь аккуратно журналировать все эксперименты.
- Сперва обоснованно подберите структурные параметры, а потом уже перебирайте регуляризаторы и модальности.
- Старайтесь наглядно демонстрировать полученные результаты, активно используйте визуализации.

Успехов!