

Математические методы анализа текстов

Классификация текстов

Мурат Апишев (mel-lain@yandex.ru)

Октябрь, 2020

Постановка задачи классификации

- ▶ Классификация — одна из наиболее часто решаемых задач в NLP
- ▶ Многие задачи (разметка последовательности, анализ тональности) могут быть сведены к классификации
- ▶ Данные:
 - ▶ $d \in D$ — множество документов (объектов)
 - ▶ $c \in C$ — множество меток классов
- ▶ Типы задач:
 - ▶ Бинарная классификация: $|C| = 2, \quad \forall d \in D \leftrightarrow c \in C$
 - ▶ Многоклассовая классификация [multiclass]:
 $|C| = K, K > 2, \quad \forall d \in D \leftrightarrow c \in C$
 - ▶ Многотемная классификация [multi-label]:
 $|C| = K, K > 2, \quad \forall d \in D \leftrightarrow \tilde{C} \subseteq C$

Метрики качества классификации

- ▶ Доля правильных ответов (**accuracy**): $acc = \#(correct) / \#(total)$
- ▶ Чаще всего используются точность (**precision**) и полнота (**recall**):

$$Pr = \frac{tp}{tp + fp}, \quad R = \frac{tp}{tp + fn}$$

$$F_1 = \frac{2}{\frac{1}{Pr} + \frac{1}{R}} = \frac{2 \cdot Pr \cdot R}{Pr + R}$$

		Верный ответ	
		+1	-1
Ответ модели	+1	tp	fp
	-1	fn	tn

- ▶ В многоклассовом случае используется **микро-усреднение** (нивелируется влияние маленьких классов):

$$Pr_{micro} = \frac{\sum tp_i}{\sum tp_i + \sum fp_i}, \quad R_{micro} = \frac{\sum tp_i}{\sum tp_i + \sum fn_i}$$

- ▶ Или **макро-усреднение** (все классы учитываются одинаково):

$$Pr_{macro} = \sum Pr_i / |C|, \quad R_{macro} = \sum R_i / |C|$$

Предобработка текстов

- ▶ Пусть дана коллекция текстовых документов D
- ▶ Текст представляет собой одну строку и алфавитных и неалфавитных символов
- ▶ Обработать его в таком виде неудобно, сперва нужно выделить числовые признаки
- ▶ Для этого данные надо привести к удобному виду и нормализовать
- ▶ Базовые шаги предобработки:
 1. токенизация
 2. приведение к нижнему регистру
 3. удаление стоп-слов
 4. удаление пунктуации
 5. фильтрация слов по частоте/длине/регулярному выражению
 6. лемматизация или стемминг

Базовые признаковые описания документов

- ▶ Обычно в ML данные представляют собой матрицу «объекты-признаки»
- ▶ Для N текстов тоже нужно как-то получить такую матрицу
- ▶ «Мешок слов» — признаков по числу уникальных слов, значение — число вхождений слова в документ:

Номер текста	Вхождений «абрикос»	...	Вхождений «яблоко»
1	0	...	23
...
N	2	...	0

- ▶ Счётчик можно заменить на значение TF-IDF:

$$v_{wd} = tf_{wd} \times \log \frac{N}{df_w}$$

- ▶ tf_{wd} — доля слова w в словах документа d
- ▶ df_w — число документов, содержащих w

Модель для классификации

- ▶ Для классификации текстов используются многие модели ML:
 - ▶ Наивный байесовский классификатор
 - ▶ Линейные модели
 - ▶ Композиции решающих деревьев
 - ▶ Полносвязные нейросети
 - ▶ Свёрточные нейросети
 - ▶ Рекуррентные нейросети
- ▶ У всех подходов есть свои преимущества и недостатки, которые обсудим в дальнейшем

Базовая модель — логистическая регрессия

- ▶ Лог-регрессия — линейная модель с логистической функцией потерь

$$\log(1 + \exp(-y\langle x, w \rangle))$$

- ▶ $x \in \mathbb{R}^n$ — вектор признаков объекта
- ▶ $w \in \mathbb{R}^n$ — вектор весов линейной модели
- ▶ $y \in \{+1, -1\}$ — метка класса объекта
- ▶ При обучении может использоваться регуляризация:
 - ▶ L_1 (Lasso)
 - ▶ L_2 (Ridge)
 - ▶ $L_1 + L_2$ (ElasticNet)
- ▶ Выход модели определяется с помощью сигмоиды по формуле

$$p(c = +1|x; w) = \frac{1}{1 + \exp(-\langle x, w \rangle)}$$

На что стоит обращать внимание

- ▶ Порог-бинаризации:
 - ▶ Лог-регрессия возвращает вероятность отнесения к классу, по ней решаем, к какому классу окончательно отнести объект
 - ▶ Для этого выставляется порог бинаризации (в sklearn метод `predict` по-умолчанию использует значение 0.5)
 - ▶ Порог лучше подбирать на валидации (в sklearn — используя метод `predict_proba`)
- ▶ Кросс-валидация: лог-регрессия обучается быстро, можно использовать кросс-валидацию для более честной оценки качества модели
- ▶ Регуляризация: часто помогает, коэффициент лучше подбирать

Многоклассовая классификация

- ▶ Лог-регрессия обобщается на многоклассовый случай заменой сигмоиды на функцию **softmax**:

$$p(c = c_i | x; W) = \frac{\exp(\langle x, w_i \rangle)}{\sum_{j=1}^m \exp(\langle x, w_j \rangle)}$$

- ▶ c_i — метка класса
 - ▶ m — число классов
 - ▶ $W \in \mathbb{R}^{m \times n}$ — набор весов m линейных моделей
 - ▶ w_i — i -я строка матрицы W
- ▶ Любой бинарный классификатор можно использовать для многоклассового случая с помощью одной из стратегий:
 - ▶ **One-vs-One**: обучаем по классификатору на каждую пару классов
 - ▶ **One-vs-Rest**: обучаем по классификатору на каждый класс

Особенности лог-регрессии

▶ Преимущества:

- ▶ Быстро и просто обучать
- ▶ Ответ интерпретируется как вероятность (преимущество перед другими моделями general ML)
- ▶ Веса модели имеют интерпретацию
- ▶ Легко реализовать вывод самостоятельно, имея готовую модель и векторизатор для данных

▶ Недостатки:

- ▶ Мало параметров в модели — может не выучить сложные закономерности в данных

Признаковые описания документов

- ▶ Описание «мешка слов» имеет ряд проблем:
 1. Не учитывается порядок слов
 2. Векторы имеют большую размерность и сильно разрежены, нет явного учёта семантики
 3. Учитывается только информация о словах
- ▶ В зависимости от используемой модели, документ можно представлять как одним вектором, так и набором векторов
- ▶ Для случая одного вектора описанные проблемы решаются так:
 1. Учесть локальный контекст с помощью коллокаций
 2. Использовать сжатые векторные представления
 3. Учесть дополнительную информацию (синтаксис, мета-информацию документа, ...)

Коллокации

- ▶ N-граммы — устойчивые последовательности из N слов, идущих подряд («машина опорных векторов»)
- ▶ Коллокация — устойчивое сочетание слов, не обязательно идущих подряд («Он сломал своему противнику руку»)
- ▶ Часто коллокациями бывают именованные сущности (но не всегда)
- ▶ Методы получения N-грамм:
 - ▶ на основе частот встречаемости ([sklearn](#), [nltk](#))
 - ▶ на основе морфологических шаблонов ([Томита](#), [YARGY-парсер](#))
 - ▶ с помощью ассоциации и статистических критериев на основе частот совместных встречаемостей ([texttt nltk](#), [TopMine](#))
 - ▶ иные подходы ([RAKE](#), [TextRank](#))

<https://yandex.ru/dev/tomita>

<https://github.com/natasha/yargy>

El-Kishky, Ahmed, et al. Scalable topical phrase mining from text corpora, 2014

Rose, Engel, et al. Automatic keyword extraction from individual documents, 2010

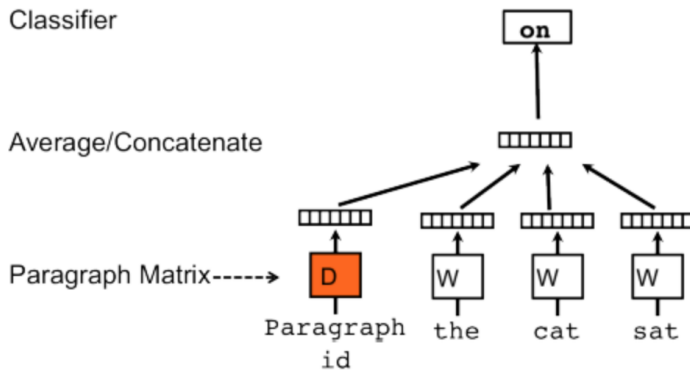
Beliga, Mestrovic, et al. An Overview of Graph-Based Keyword Extraction Methods and Approaches, 2015

Сжатые векторы для документов

- ▶ Размерность векторов «мешка слов» можно сократить, применив **Truncated SVD-разложение**
- ▶ Простое и эффективное решение — усреднить векторы слов документа (**word2vec**, **GloVe**, **FastText**)
- ▶ Ещё лучше может сработать взвешенная сумма векторов, в качестве весов можно взять, например, значения TF-IDF
- ▶ Можно напрямую обучать векторы документов, как общие, так и заточенные под задачу классификации
- ▶ Пример метода обучения общих векторов — **doc2vec (paragraph2vec)**
- ▶ Он представлен двумя моделями:
 - ▶ Distributed Memory
 - ▶ Distributed Bag of Words

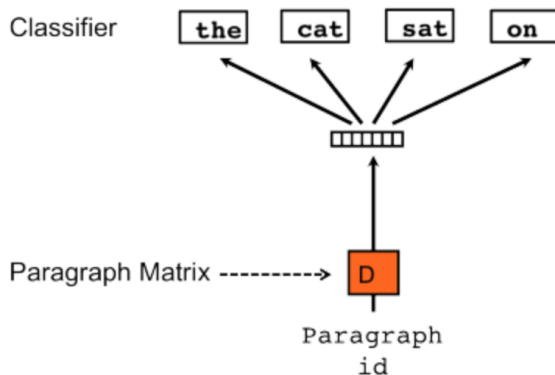
Модель Distributed Memory

- ▶ За основу берётся модель CBOW word2vec
- ▶ При обучении проходим окном по словам документов
- ▶ Кроме слов кодируем и учитываем при обучении документы (параграфы)
- ▶ В каждом окне добавляем на вход $|D|$ -мерный one-hot вектор документа



Модель Distributed Bag of Words

- ▶ За основу берётся модель [Skip-gram word2vec](#)
- ▶ При обучении проходим окном по словам документов
- ▶ Модель обучается предсказывать случайное слово из случайного окна по one-hot вектору документа, в котором оказалось окно



Документ как набор векторов

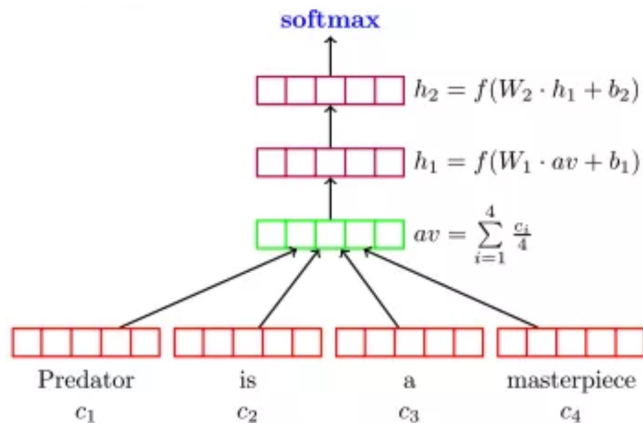
- ▶ Многие модели допускают обработку документа в виде набора векторов (упорядоченного или нет)
- ▶ В таком случае представление документа в виде одного вектора формируется внутри модели
- ▶ В нейросетевых моделях признаки и модели тесно взаимосвязаны
- ▶ Обычно документ представляется в виде векторов его **структурных компонентов**:
 - ▶ символов
 - ▶ символьных N-грамм и слов
 - ▶ словарных N-грамм
- ▶ Входные векторы также могут быть как предобученными, так и настраиваемыми при обучении

Входные признаки

- ▶ Входные векторы также могут быть как предобученными, так и настраиваемыми при обучении
- ▶ Входные векторы слов могут содержать различные признаки:
 - ▶ Семантические (вектор word2vec)
 - ▶ Грамматические и морфологические (номер части речи)
 - ▶ Позиционные и синтаксические (вектор значений sin от позиции)
- ▶ Модели условно можно разделить по возможностям учёта контекста (работы с векторами как с последовательностью):
 - ▶ Не учитывают контекст (полносвязные сети)
 - ▶ Учитывают локальный контекст (свёрточные сети)
 - ▶ Учитывают глобальный контекст (рекуррентные сети, Transformer)

Модель Deep Averaging Network

- ▶ Вход — векторы структурных компонентов текста
- ▶ Векторы усредняются и проходят через 2-4 полносвязных слоя
- ▶ Выход — **Softmax** по числу классов от выхода последнего слоя

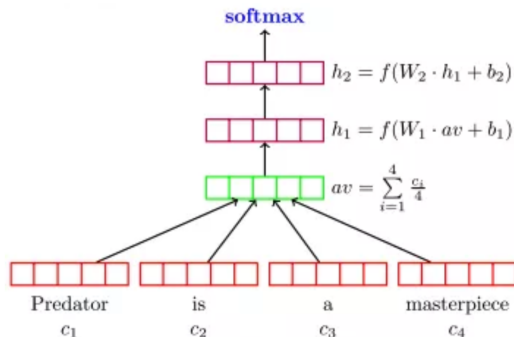


Обучение Deep Averaging Network

- ▶ Для промежуточных полносвязных слоёв полезна **регуляризация**, стандартная схема:

1. Выходы слоя
2. Batch-нормализация
3. Нелинейная функция активации (например, ReLU)
4. Dropout-регуляризация

- ▶ Для DAN дополнительно используют **Word Dropout**: из входа случайно удаляются несколько векторов



Модель FastText

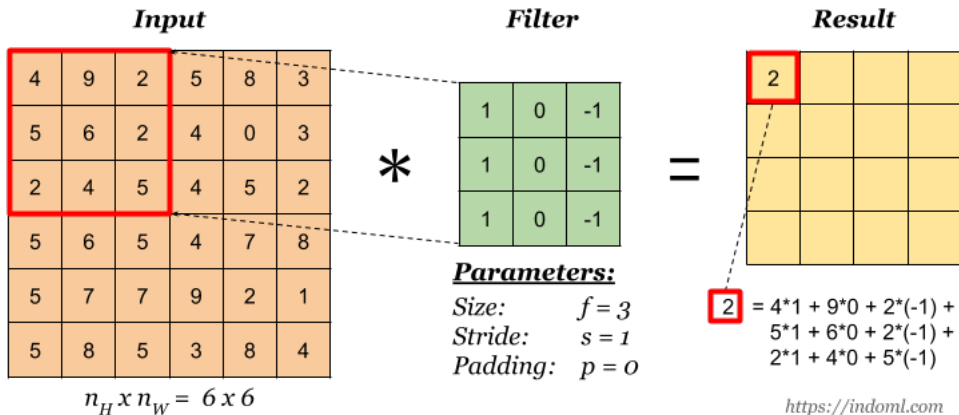
- ▶ FastText — знакомая модель обучения векторных представлений слов
- ▶ Модель FastText структурно идентична word2vec: двуслойная полносвязная нейросеть (без нелинейностей до Softmax)
- ▶ Одноимённая библиотека работает эффективно и параллельно на CPU с использованием hashing trick
- ▶ Эта библиотека имеет режим обучения с учителем
- ▶ Вход — one-hot векторы слов и символьных/словарных N-грамм
- ▶ На первом слое обучаются векторные представления компонентов входа
- ▶ На втором настраивается полносвязный слой для многоклассового линейного классификатора
- ▶ Выход — Softmax от выхода второго слоя

Convolutional Neural Network

- ▶ CNN лежат в основе многих современных методов компьютерного зрения
- ▶ Оказалось, что они хорошо подходят для ряда задач обработки текстов
- ▶ В задаче классификации CNN позволяют учитывать локальный контекст слов без явного использования словарных N-грамм
- ▶ При сопоставимом уровне качества они обучаются существенно быстрее рекуррентных нейронных сетей
- ▶ Компоненты CNN:
 - ▶ Входной слой — содержит векторные представления признаков
 - ▶ Свёрточный слой — использует входные векторы для извлечения локальных признаков
 - ▶ Слой субдискретизации (пулинг) — извлекает глобальные признаки с наибольшим сигналом
 - ▶ Полносвязный слой — элемент классификатора на выходе сети

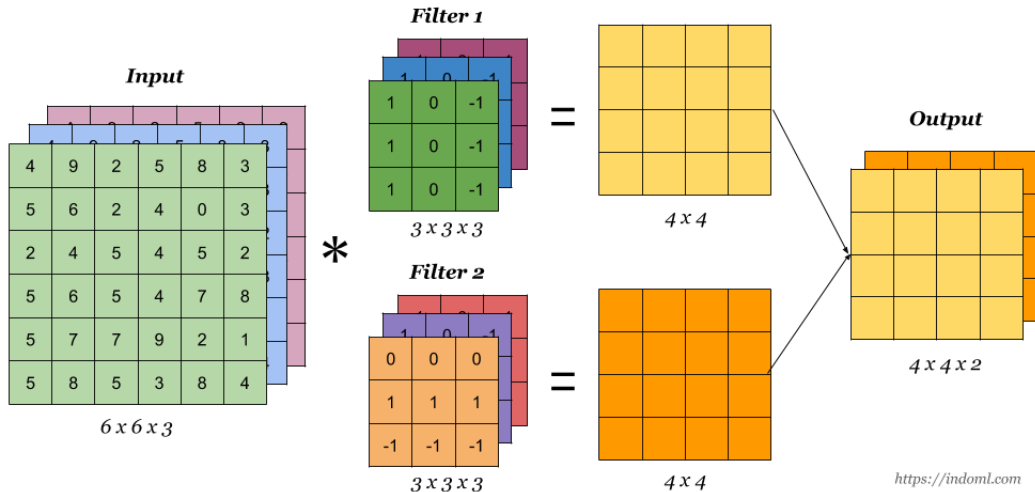
Свёрточный слой

- ▶ **Вход** — числовые матрицы (обычно 2-х или 3-хмерные)
- ▶ Слой параметризуется набором ядер свёрток (фильтров)
- ▶ **Фильтр** — небольшая матрица, которая применяется последовательно к различным частям входа



Feature Map

- ▶ Выходом фильтра является матрица — **карта признаков**
- ▶ Свёрточный слой возвращает набор карт признаков (по числу фильтров)

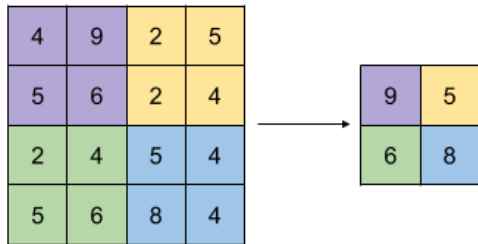


<https://indoml.com>

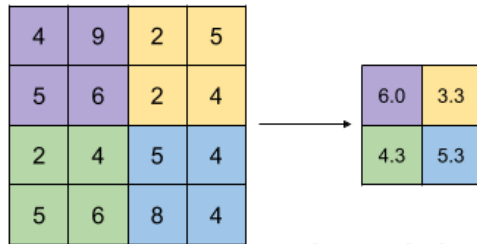
Слой пулинга

- ▶ Для того, чтобы сделать сеть более устойчивой к изменению входа набор близких признаков агрегируется в один
- ▶ Применяются различные виды пулинга, для картинок наиболее популярны усреднение и **max-пулинг**

Max Pooling



Avg Pooling



<https://indoml.com>

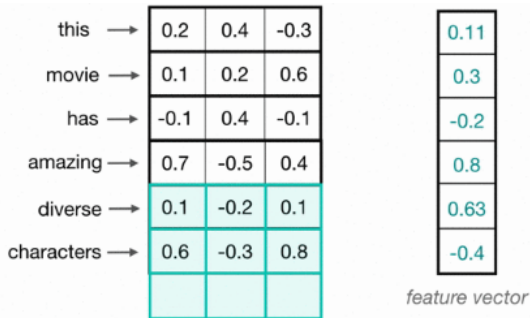
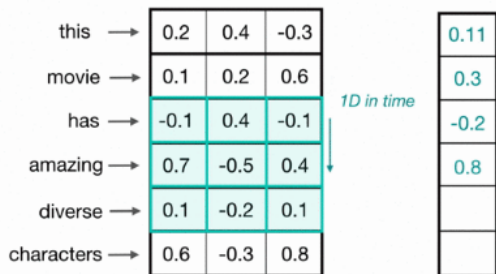
Свёрточные сети для текстов

- ▶ Обычно строятся неглубокие модели с одним свёрточным слоём и одним слоем пулинга + несколько полносвязных слоёв
- ▶ **Вход** — матрица конкатенированных векторных представлений слов
- ▶ Документы могут иметь разную длину — нужен **паддинг**
- ▶ Вектор паддинга должен быть таким, чтобы пулинг не выбрал его значения

I
like
this
movie
very
much
!

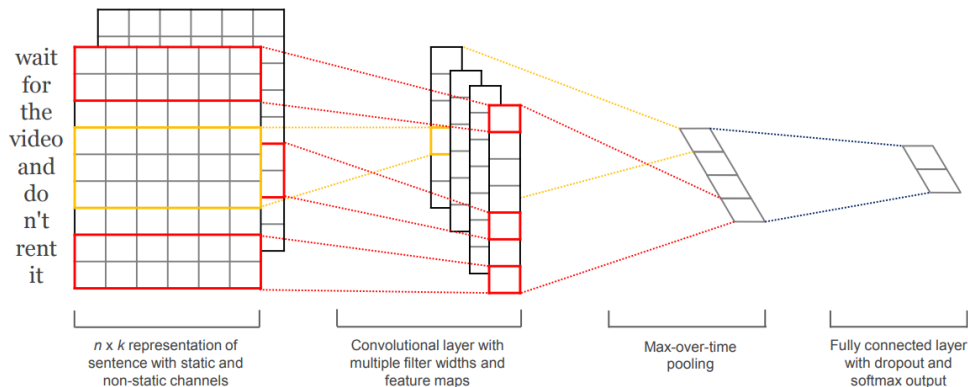
Свёрточные сети для текстов

- ▶ Для текстов обычно используются **одномерные фильтры**
- ▶ Первая размерность определяет количество слов, обрабатываемых фильтром одновременно
- ▶ Вторая фиксирована и равна размерности векторных представлений



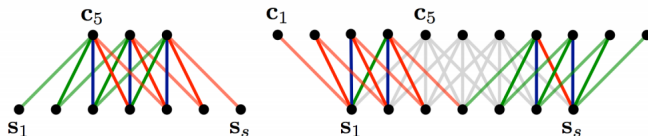
Фильтры и пулинг

- ▶ Фильтры могут иметь разные размеры, чтобы учитывать контексты различной длины
- ▶ Карты признаков для документа будут иметь разные размерности
- ▶ Пулинг агрегирует их в один вектор фиксированной длины
- ▶ **max-over-time pooling** выбирает из каждой карты наибольшее значение



Важные детали

- ▶ Свёртки могут быть узкими (**narrow**) или широкими (**wide**), в зависимости от наличия zero-падинга
- ▶ Выбор между ними существенен, когда размер фильтра сопоставим с размером входной матрицы



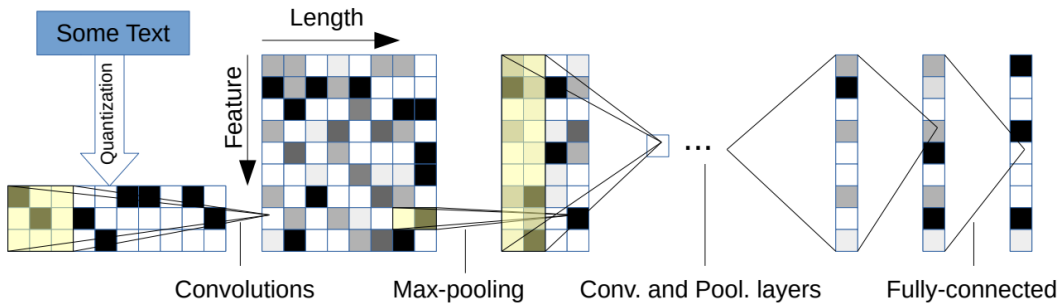
- ▶ Вместо **max-over-time pooling** можно использовать **k-max pooling**, выбирающий k наибольших значений из каждой карты
- ▶ Можно вытаскивать число элементов, пропорциональное длине карты
- ▶ Есть более сложные виды пулинга, например, **attentive pooling**, в котором агрегация карт параметризуется небольшой нейросетью

Входной слой сети

- ▶ Векторные представления на первом слое могут быть как предобученными, так и обучаемыми с нуля вместе с сетью
- ▶ Предобученные векторы тоже можно фиксировать, а можно дообучать
- ▶ Можно иметь более одного входного канала, например, два из векторов word2vec, один фиксированный, один — дообучаемый
- ▶ В этом случае каждый фильтр применяется к обоим входам, результаты можно суммировать
- ▶ Вместо векторов слов можно использовать на входе one-hot векторы символов текста из фиксированного набора
- ▶ Такой подход работает хорошо только на больших выборках

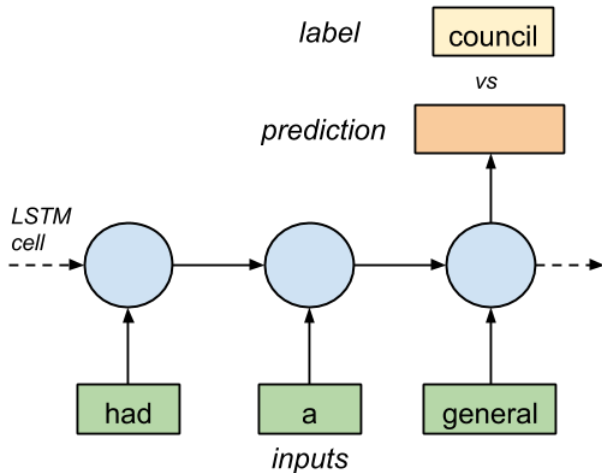
Многослойные CNN для текстов

- ▶ Можно работать с входной матрицей как с изображением
- ▶ Тогда можно применять обычные свёртки/пулинги и обучать многослойные архитектуры
- ▶ Такой подход хорошо работает, когда на вход сети подаются one-hot векторы символов



RNN для классификации

- ▶ Рекуррентные нейросети хорошо подходят для обработки текстов как последовательностей, даже длинных (**LSTM**, **GRU**)
- ▶ Хорошо настроенная многослойная **RNN** может очень хорошо решать сложные задачи классификации
- ▶ **Пример:** соревнование по анализу токсичности комментариев на Kaggle



RNN для классификации

- ▶ Хорошо настроенная рекуррентная сеть может очень хорошо решать задачу классификации
- ▶ На практике их редко используют для классификации текстов
 - ▶ RNN требуют много данных для обучения
 - ▶ Обучение занимает много времени
 - ▶ Параметры сети могут занимать много памяти
 - ▶ Сеть в режиме предсказания может работать медленно
- ▶ Большинство задач классификации текстов достаточно простые, чтобы хорошо решаться линейными моделями и DAN
- ▶ В более сложных случаях используются CNN
 - ▶ Они тоже до определённой степени учитывают контекст
 - ▶ Скорость работы и число параметров сильно меньше, чем у глубоких RNN

Transfer learning в классификации

- ▶ Одна из наиболее сильных и популярных моделей для обработки последовательностей — **Transformer** и модели на его основе
- ▶ Кодировщик **Transformer** (например, **BERT**) можно использовать в качестве генератора качественных признаков, на основе которых можно обучить простую, но сильную модель классификации
- ▶ Если базовая сеть-кодировщик хорошо предобучена, то
 - ▶ дообучать классификатор сильно проще
 - ▶ можно решать сложные задачи классификации, имея небольшие размеченные выборки
- ▶ Модель получается тяжеловесной и медленной, но эффективность можно повышать с помощью дистиллированных моделей (**DistilBERT**, **TinyBERT**)

Практический кейс I

Классификация обращений в службу поддержки на 100+ классов

- ▶ Сырых данных много, разметить можно только небольшую часть
- ▶ Большое число классов, многие похожи, список классов часто меняется
- ▶ На всех сырых данных предобучается BERT
- ▶ На его выходах обучается полносвязная сеть
- ▶ Если набор классов изменился, достаточно переобучить голову сети

Варианты использования BERT для классификации:

- ▶ Брать выходной вектор для токена CLS
- ▶ Брать векторы для CLS с разных уровней и применять пулинг или складывать с обучаемыми весами
- ▶ Применять аналогичную операцию для выходных векторов всех токенов

Практический кейс II

Бинарная классификация медицинских карт

- ▶ Каждая карта — набор записей, созданных автоматической системой или вбитых врачом
- ▶ Выборка сильно несбалансированная
- ▶ Записей в карте очень много, всю карту в BERT/RNN не поместить
- ▶ Простые модели работают плохо, важны вариации числовых значений в N-граммах («уровень белка 19.89», «уровень белка 21.2»)
- ▶ Варианты решения:
 - ▶ Кодировать записи с помощью BERT, выходные эмбединги подавать в CNN со свёртками разного размера
 - ▶ Кодировать слова с помощью FastText и тоже подавать в CNN
- ▶ И BERT, и FastText предобучаются на большом корпусе размеченных медицинских карт

Этапы решения задачи машинного обучения

- ▶ Выбор метрики качества
- ▶ Формирование требований к данным
- ▶ Сбор данных
- ▶ Предобработка данных
- ▶ Формирование признакового пространства
- ▶ Выбор класса моделей для апробации
- ▶ Настройка моделей
- ▶ Релиз модели
- ▶ Поддержка модели

Решим продуктовую задачу классификации

Контекст:

делаем сервис по агрегации и рекомендации новостного контента

Неформальная постановка задачи:

борьба с «чернушными» новостями

Формальная постановка задачи:

нужно придумать

Метод решения:

нужно придумать

Метрика:

нужно придумать

Внедрение:

открытый вопрос

Формализация постановки

- ▶ Сперва необходимо понять, что представляет собой решаемая задача, и декомпозировать её
- ▶ В рассматриваемом случае задача очевидно делится на две части:
 - ▶ обучение модели классификатора «чёрного контента»
(техническая подзадача)
 - ▶ подбор стратегии его использования в рамках сервиса
(продуктовая подзадача)
- ▶ Важно формализовать описание выявляемого контента
- ▶ Первый очевидный вариант:
 - ▶ тексты, описывающие убийства, пытки и т.п.
 - ▶ тексты про бытовые конфликты, оскорбления
 - ▶ скандальные новости, низкосортный «жёлтый» контент
- ▶ Формулировки могут (и будут) меняться в процессе решения

Метрика качества решения

- ▶ Начинать работу нужно с определения метрики, по которой решение будет приниматься
- ▶ Метрики для классификатора: F1-мера, точность/полнота, ROC-AUC
- ▶ Бизнесу, как правило, эти числа неинтересны
- ▶ Важно не только качество классификатора, но и то, как он используется в продакшене
- ▶ Метрики, по которым будет определяться стратегия показа выявленного контента, и будет целевой для всей системы
- ▶ Считаются такие метрики обычно в А/Б-тесте, примеры:
 - ▶ CTR (click-through rate)
 - ▶ Доля жалоб на контент
 - ▶ Клики на рекламу
 - ▶ ...
- ▶ Эти метрики можно считать для разных когорт пользователей

Промежуточный итог I

Что получили:

- ▶ задача разделена на две последовательных этапа, есть общее видение решения
- ▶ понятно, как измерять качество на каждом этапе
- ▶ понятно, что нужно начинать с обучения классификатора текстов
- ▶ дано первое определение целевого контента

Что делать дальше:

- ▶ Искать подходящие данные для обучения/валидации
- ▶ определиться с предобработкой и признаками
- ▶ выбрать модели, обучить и провалидировать их
- ▶ убедиться, что результат получается хорошим (как по метрике, так и с т.з. здравого смысла)

Где брать сырые данные

- ▶ В отличие от учебных задач, в задачах из реальной жизни **данных** обычно нет
- ▶ Часто нет не только размеченных данных, но и сырых текстов
- ▶ В последнем случае данные:
 - ▶ **покупаются** (если есть, у кого)
 - ▶ скачиваются через **официальный API** (если он есть)
 - ▶ скачиваются с помощью **веб-краулинга** (если нет запрета и защиты)
- ▶ Если подходящих данных не существует или их нельзя получить, можно их сгенерировать
- ▶ Для этого нужно понимать, как они устроены, как и на основе чего их можно генерировать
- ▶ Можно использовать ассессоров или **краудсорсинг** (Яндекс.Толока, Amazon Mechanical Turk)

Разметка данных

- ▶ В рассматриваемом кейсе **сырые данные есть** (тексты статей)
- ▶ Они уже полезны для предобучения, но для классификации в общем случае нужна разметка
- ▶ Иногда её можно получить относительно просто, например, взять **тексты, на которые жаловались пользователи**
- ▶ В общем случае данные нужно **изучить и разметить руками** какую-то часть (это в принципе полезно)
- ▶ Можно определить, какими словами/фразами/регулярными выражениями характеризуются «чернушные тексты» и пополнить выборку с помощью правил и словарей
- ▶ Если данных много или размечать их нужно регулярно, лучше опять использовать **краудсорсинг**
- ▶ Разберём работу с ним на примере **Толоки**

- ▶ Сервис-маркетплейс, объединяющий
 - ▶ сообщество людей, делающих разметку за деньги
 - ▶ их потенциальных заказчиков
- ▶ Предоставляет интерфейсы для решения многих задач разметки:
 - ▶ Категоризация
 - ▶ Попарное сравнение
 - ▶ Сегментация изображений
 - ▶ Генерация текста
 - ▶ Разметка последовательности
 - ▶ ...
- ▶ Даёт возможности отбора размечающих, контроля качества разметки, отложенной оплаты и т.д.
- ▶ Имеет API, можно писать обёртки для автоматизации процессов загрузки данных, выгрузки и валидации результатов

Этапы работы с Толокой

- ▶ **Написание инструкции**

Мы хотим показать текст и попросить выбрать класс

- ▶ **Подготовка визуального шаблона**

- ▶ **Настройки фильтров пользователей и контроля качества**

Выбираем наиболее качественных пользователей и ставим ограничения на количество ошибок и скорость разметки

- ▶ **Генерация и загрузка заданий в виде пулов**

Собираем множество текстов на разметку, добавляем немного правильных ответов для онлайн-контроля разметки

- ▶ **Запуск разметки, валидация результата, оплата**

- ▶ **Выгрузка и использование результатов**

Написание инструкции

- ▶ При написании инструкции нужно исходить из двух посылов:
 - ▶ на платформе есть люди, которые будут добросовестно решать задачу
 - ▶ они не специалисты ни в техническом, ни в продуктивном смыслах
- ▶ Поэтому задача должна быть поставлена
 - ▶ однозначно
 - ▶ всеобъемлюще
 - ▶ по возможности — кратко

- ▶ Как стоит писать:

В задании показывается текст, нужно определить, «чернушный» он или нет. Текст «чернушный», если он ...

- ▶ Как НЕ стоит писать:
















В данном задании Вам предстоит решать задачу классификации новостей. Классификация бинарная, то есть на два класса, один из которых назовём условно «чернушным»...

Подготовка визуального шаблона

Шаблоны

Шаблоны позволяют сформировать и запустить задания, исходя из ваших потребностей. Вы можете использовать шаблон как есть или адаптировать его под ваши входные данные и формат получаемых ответов.

Классификация

 <input type="radio"/>  <input type="radio"/> 	Оценка видео Позволяет просмотреть видео и выбрать один из возможных вариантов. В шаблоне видео-плеер и несколько радио-кнопок. <input type="button" value="Выбрать"/> <input type="button" value="Предпросмотр"/>
 <input checked="" type="checkbox"/>  <input type="radio"/>  <input type="radio"/> 	Категоризация изображений Подходит для классификации изображений и проставления тегов. В шаблоне изображение и радиокнопки. <input type="button" value="Выбрать"/> <input type="button" value="Предпросмотр"/>
 <input checked="" type="checkbox"/>  <input type="radio"/> 	Модерация контента Позволяет классифицировать текстовые комментарии. В шаблоне текст и радиокнопки. <input type="button" value="Выбрать"/> <input type="button" value="Предпросмотр"/>
 <input checked="" type="checkbox"/>  <input type="radio"/> 	Категоризация изображений с заголовком на странице В шаблоне шапка с общей информацией для всей страницы с заданиями, картинка и радиокнопки для каждого задания. <input type="button" value="Выбрать"/> <input type="button" value="Предпросмотр"/>
	Категоризация текста с дополнительными опциями
	Категоризация поисковых запросов В шаблоне текст, кнопки для поиска в интернете и несколько

Настройка фильтров и контроля

- ▶ Ограничить качество пользователей
- ▶ Определить навыки пользователей (например, знание языка)
- ▶ Установить значение **перекрытия** для каждого задания (обычно 3-5)
- ▶ Ограничить число или долю ошибок относительно перекрытия
- ▶ Ограничить число или долю ошибок в капче (соответственно, выставить частоту капчи)
- ▶ Ограничить число или долю ошибок в проверочных заданиях
- ▶ Установить верхние и нижние границы времени на задание, банить за слишком быстрые ответы
- ▶ Установить стоимость выполнения заданий

Создание и загрузка пула

- ▶ Пул представляет собой набор заданий для разметки внутри проекта
- ▶ Для его формирования нужно создать tsv-файл, в котором будут содержаться необходимые данные
- ▶ В рассматриваемом случае это могут быть колонки с текстом, его идентификатором и правильным ответом
- ▶ Ответ для большинства заданий будет пустым
- ▶ Задания, у которых он проставлен, будут считаться контрольными
- ▶ После загрузки пула с учётом перекрытия будут сформированы итоговые страницы с заданиями и подсчитаны статистики

Запуск разметки и мониторинг

ЗАДАНИЯ ПУЛА (Пример загрузочного файла (tsv, кодировка UTF-8))

Загрузить

файлы

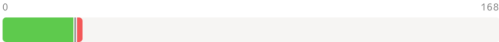
Предпросмотр

56 страниц заданий	0 обучающее задание
221 заданий	0 контрольное задание

16 %

Сделано 27, принято 24

Проверить задания



СТАТИСТИКА ПУЛА

<div>38сек</div> <div>Среднее время выполнения задания</div>		<div>—</div> <div>Приблизительное время завершения</div>		<div>0,72 (+ 0,14)</div> <div>Израсходованные средства (+ надбавка)</div>		<div>\$ 4,98 (+ 1,00)</div> <div>Приблизительный бюджет (+ надбавка)</div>					
<div>116 чел.</div> <div>Активные пользователи, которым доступны задания</div>		<div>14 чел.</div> <div>Заинтересованные пользователи</div>		<div>6 чел.</div> <div>Взявшие задания пользователи</div>		<div>4,50</div> <div>Выполненные задания на одного пользователя</div>		<div>9 шт.</div> <div>Просроченные задания</div>		<div>1 шт.</div> <div>Пропущенные задания</div>	

Отбор документов, активное обучение

- ▶ Разметка требует денег и времени, а контента много, весь не разметить
- ▶ Нужно определить, какие документы нужно разметить в первую очередь
- ▶ Это особенно важно, когда есть сильный дисбаланс классов

Что можно сделать:

- ▶ Собираем максимально быстро и просто (по словарям и регулярным выражениям) базовую выборку
- ▶ Обучаем на ней простую модель с вероятностным ответом, например, лог-регрессию
- ▶ Получаем с её помощью ответы для неразмеченных документов, отправляем в разметку те, в которых модель **слабо уверена**
- ▶ Переобучаем модель, повторяем процедуру итеративно

Аугментация текстовых данных

- ▶ Часто данных для обучения не хватает и получить новые полноценной разметкой сложно или дорого
- ▶ Частный случай этой проблем — дисбаланс классов
- ▶ Желательно сформировать обучающую выборку так, чтобы классы были представлены в примерно равных пропорциях

Что можно сделать:

- ▶ Для борьбы с дисбалансом в простейшем случае делать перевзвешивание, увеличивая веса объектов меньшего класса
- ▶ Использовать методы аугментации текстов
- ▶ Использовать методы аугментации векторов текстов

Методы аугментации текстов

- ▶ Есть стандартный набор методик «размножения» текстов:
 - ▶ заменять часть значимых (не стоп-слов) слов в тексте на один из их синонимов
 - ▶ случайно вставлять в разные позиции в тексте синонимы значимых слов
 - ▶ случайно переставлять в тексте пары слов (если модель работает с последовательным текстом)
 - ▶ случайно удалять из текста часть слов (как в Word Dropout)
- ▶ При подборе словаря важно учитывать лингвистическую область, в которой решается задача
- ▶ Если данные генерируются пользователями, то нельзя игнорировать сленг, жаргонизмы, нецензурную лексику и т.п.

Методы аугментации векторов

- ▶ Если данные любого типа погружены в векторное пространство, то генерировать новые можно напрямую в этом же пространстве
- ▶ **Пример:** пространство векторов-сумм word2vec слов документа
- ▶ Очевидно, что тексты для новых векторов получены не будут, не факт, что они в принципе существуют
- ▶ Один из возможных вариантов — алгоритм **SMOTE**:
 - ▶ выбираем произвольный объект одного класса
 - ▶ ищем с помощью kNN его соседей из этого же класса
 - ▶ выбираем из них случайного соседа
 - ▶ выбираем случайную точку между этой парой объектов
 - ▶ добавляем её в выборку как объект этого класса
- ▶ Реализация **SMOTE** (и его модификации **ASMO**) для Python есть в библиотеке **imbalanced-learn**

Промежуточный итог II

Что получили:

- ▶ научились генерировать нужные данные
- ▶ данные подготовлены, сформировано подходящее признаковое описание
- ▶ классификатор обучен в Jupyter-ноутбуке, на валидационной выборке он показывает неплохое качество

Что делать дальше:

- ▶ интегрировать классификатор в продуктовую систему, протестировать его в продакшене
- ▶ определиться со стратегией его применения, подсчитать нужные бизнес-метрики, оценить эффект
- ▶ автоматизировать поддержку и мониторинг решения

Детали релиза решения

- ▶ **Интеграция** — внедрение решения в существующую инфраструктуру:
 - ▶ Например, у компании уже может быть система для оффлайн-обучения моделей и выкатки их в продакшн
 - ▶ Тогда весь написанный код нужно перенести в него + добавить туда нужные библиотеки, если их не хватает
- ▶ **Мониторинг** — качество работы модели нужно измерять постоянно в онлайн-режиме, сигнализируя о проблемах
- ▶ **Поддержка** — почти все сделанные выше шаги будут повторяться:
 - ▶ Разметка очередной порции данных
 - ▶ Дообучение классификатора
 - ▶ Валидация его качества
 - ▶ Выкатка новой версии модели в продакшн
 - ▶ Настройка гиперпараметров (например, порог бинаризации для лог-регрессии)

Стратегия использования и оценки решения

- ▶ Система запущена, сама отправляет данные на разметку, дообучается, мониторит себя и выкатывает в продакшн
- ▶ Хотим использовать результаты для управления потоком показов
- ▶ **Какие гипотезы можно пробовать проверять:**
 - ▶ полностью исключить «чернуху» из потока
 - ▶ заменять такой контент на семантически близкий, но менее отталкивающий
 - ▶ выделить пользователей, которые склонны к «чёрным» текстам, и показывать их только им
 - ▶ этим же пользователям показывать больше подобного контента
- ▶ Принимаемся по бизнесовой метрике, например, **CTR** в **A/Б-тесте**
- ▶ В одной из групп улучшение получилось статистически значимым
- ▶ **Успех! Решение работает, приносит пользу, работаем дальше**

Прошло два месяца...

- ▶ Уже неделю показы стабильно падают, нужно разбираться
- ▶ Мониторинг говорит, что классификатор «чернушного» контента ведёт себя необычно
- ▶ В последние две недели он выдаёт всё больше положительных ответов
- ▶ Нужно заглянуть в тексты, которые он отмечает положительными, и проверить состояние модели
- ▶ Анализ показывает, что с моделью всё в порядке
- ▶ А вот в контенте стало появляться много исторических текстов про войну, в которых много негативных сигналов, на которых срабатывает классификатор

Поддержка решения

- ▶ Смотрим в календарь — середина апреля, к 9 мая стали писать больше лонгридов про BOB
- ▶ Расследование завершено, необходимо исправлять ситуацию

ML в индустрии — процесс циклический:

- ▶ Возвращаемся на этап сбора данных
- ▶ Теми же методами формируем выборку для классификатора исторических текстов
- ▶ С его помощью корректируем выборку для классификатора «чернушного» контента
- ▶ Уточняем инструкцию в Толоке
- ▶ Прогоняем весь пайплайн — **проблема решена!**

Итоги занятия

- ▶ Задача классификация — наиболее распространённая в NLP
- ▶ В зависимости от сложности применяются различные модели ML и DL
- ▶ Признаки могут быть как счётчиками, так и обучаемыми векторами
- ▶ Лог-регрессия — простой и сильный бейзлайн
- ▶ Стандартные сильные модели — FastText и CNN
- ▶ Рекуррентные модели, несмотря на высокое качество, применяются реже
- ▶ Предобученные модели на основе архитектуры Transformer используются для transfer learning в ситуациях недостатка размеченных данных и большого числа сложно отделимых классов
- ▶ Критическое значение имеют способ получения обучающих данных, их качество и методы предобработки
- ▶ Важно не только обучить модель, но также внедрить её и обеспечить поддержку и контроль качества