Разработка модели определения тональности

Владислав Кораблинов

16.11.2019

1 Постановка задачи

В данной практической работе перед нами ставилась задача определения тональности текста. На вход подается набор текстов, для каждого из которых необходимо вывести оценку тональности от 1 до 10, где 1 - наиболее негативный, 10 - наиболее позитивный. Для обучения предлагается 20000 текстов с известными оценками. Каждый из тестовых наборов содержит 1000 текстов.

1.1 Замечания о используемых метриках качества

Основной метрикой качества в данной задаче является корень из среднеквадратичного отклонения (RMSE). Кроме того, каждое решение получает некоторый балл от 0 до 100.

Для некоторых моделей с конкретными параметрами было проведено тестирование на как минимум 5 различных наборах данных, и для каждой из них мы будем указывать минимальную, среднюю и максимальную оценку по 100-балльной шкале, а также среднее значение RMSE. Однако, ввиду того, что оценку на тестовых наборах мы получаем не локально и такие результаты тружно сохранять в удобном виде, иногда мы будем просто говорить, что некоторая модель оказался лучше другой - это означает, что она лучше проявила себя на тестовых наборах, однако точные значения не сохранялись.

Для понимания того, что такое хороший результат, приведем оценки, получаемые baseline-решением, которое всегда предсказывает 8 (так как среднее значение по тренировочному набору ≈ 7.8). Такое решение получает RMSE ≈ 2.3 и ≈ 68 баллов.

2 Общий план решения

Для решения задачи мы будем использовать регрессию, так как предсказываемая переменная имеет порядковую природу.

В целом наша модель предсказания будет состоять из

- препроцессора
- векторизатора
- обучаемой модели

Результат работы модели обрезается до диапазона 1-10, числа округляются до ближайшего целого.

3 Препроцессинг

В рамках данной задачи мы будем понимать под препроцессингом в том числе и токенизацию. Заметим сначала, что даже если использовать только обычную токенизацию по словарным символам, можно получить решение, набирающее 73.2 балла (RMSE 1.91). Тем не менее, для усиления модели хороший препроцессинг необходим.

3.1 Обработка символов

Важной особенностью нашей задачи является очевидная важность не только слов, но и символов, таких как, например, восклицательные знаки и смайлики. Для того, что понять, какие символы-/последовательности и как лучше всего обрабатывать, было проведено небольшое исследование обучающего набора. Для всех возможных последовательностей символов длины больше 1 была посчитана их частота и отброшены редко встречающиеся (< 10 раз) комбинации. Для оставшихся оказалось, что осмысленными являются восклицательный и вопросительный знаки (причем последовательности одного и того же знака могут быть достаточно длинными) и разные вариации смайликов, в том числе обычные открывающие и закрывающие скобки (больше 1 подряд). Поэтому на первой стадии препроцессинга все смайлики заменялись на две скобки соответствующего типа, а также все последовательности одного и того же символа длины больше 2 заменялись на последовательность длины 3. Таким образом, в будущий рабочий словарь помимо слов вошли токены *!!*, *!!!*, *??*, *???*, *))*, *)))*, *((*, *(((*.

3.2 Выделение токенов слов

Токены слов выделялись просто как последовательности словарных символов, это стандартное решение работает достаточно хорошо.

3.3 Морфологическая обработка

Ясно, что для для уменьшения размера словаря необходимо провести морфологическую обработку. Сначала была применена лемматизация с помощью Mystem и дальнейшая работа производилась с полученным словарем. Однако в некоторый момент было замечено следующее. После лемматиазации в словаре находились слова *банальный*, *банально*, *банальность* и *банальщина*, каждое из которых имело явно негативный вес. Однако, ясно, что семантически эти слова выражают одно и то же, а это значит, что мы имеем целых 4 коррелированных признака, которые, во-первых, рассеивают влияние смысловой категории, а во-вторых, ослабляют это влияние за счет того, что в тестовом тексте нам скорее всего встретится только одно из этих слов. Поэтому после лемматизации был применен стемминг с помощью Snowball, который значительно снизил наблюдаемую проблему, но все-таки все еще не решил ее. Инструментов, которые позволяли бы хотя бы качественно определять однокоренные слова, я не вспомнил. Также для склеивания семантически близких слов можно было бы использовать предобученные эмбеддинги слов, но это достаточно затратно по времени, поэтому не было опробовано.

3.4 Использование биграмм

Для усиления захвата смысла было также использовано извлечение биграмм. Особенно очевидна польза от него в фрагментах с употреблением частицы ‘не’, но и в целом ясно, что информация о рядом стоящих словах полезна для определения эмоциональности.

Размер итогового словаря оказался порядка 100000 токенов.

4 Векторизация

Для векторизации было решено применять простой TF-IDF. Единственным фильтром было минимальное значение DF. Путем ручного изучения документных частот слов было принято решение установить минимальную частоту 10. Это привело к сокращению размера словаря до примерно 18000 токенов.

5 Обучение

5.1 Использованые регрессоры

Для обучения были попробованы следующие регрессоры:

- простая линейная регрессия

- SGD линейная регрессия
- SVM с разными ядрами
- AdaBoost на деревьях

Отметим, что простая линейная регрессия была всюду хуже остальных регрессоров, поэтому ее результатов в дальнейшем не будет.

Также отметим, что SVM с полиномиальными ядрами и rbf куда-то не туда переобучался и на все тексты выдавал оценку 9, поэтому все результаты будут приведены для SVM с линейным ядром.

5.2 Отбор признаков

Понятно, что размер нашего словаря выглядит слишком большим для хорошей линейной регрессии, кроме того, многие слова по смыслу не должны существенно влиять на оценку тональности. Чтобы уменьшить размер словаря, были применены следующие подходы.

Простейшим способом являлось выставаление у векторизатора ограничения на размер словаря, при этом он оставлял токены с наибольшей частотой в коллекции. Эксперименты показали, что наилучший результат достигается с ограничением в 3000 токенов. Будем называть этот словарь *Vec vocab*

Далее для оценки значимости была опробована SGD линейная регрессия с L1-регуляризацией. В итоговый словарь попали токены, модуль коэффициента при которых превышал 0.01. Далее мы будем называть этот словарь *L1 vocab*.

Также для отбора токенов было опробовано применение следующей процедуры. Все тексты были разбиты на классы $T_i, i = 1..10$ по оценке на тренировочном наборе. Для каждого токена было посчитано следующее значение: $\sum_{i=1}^{10} w_i \frac{tf_i(token)}{N_i}$, где $w_i = i - 5$ при $i > 5$ и $w_i = i - 6$ при $i < 6$, $tf_i(token)$ - частота токена в текстах класса i, N_i - количество текстов в классе i. Далее токены были отсортированы по модулю полученного значения и из них выбрано 3000 лучших. Этот словарь мы будем в дальнейшем называть *TF vocab*.

5.3 Ход экспериментов

В самой простой модели не применялся препроцессинг, использовался словарь *Vec vocab*, обучение с помощью простой линейной регрессии. Как раз она дала результат 73.2 балла (RMSE 1.91).

Следующим шагом было добавление описанного препроцессинга с лемматизацией. Для обучения использовалась SGD-регрессия с L1-регуляризацией. Была протестирована модель со значением коэффициента регуляризации по умолчанию 0.0001, значением 0.001 и модель с векторизатором с минимальным значением df, равным 20. Были получены следующие результаты:

	Мин.	**Среднее**	**Макс.**
SGDR L1	73.94	75.01	76.15
SGDR L1 alpha=0.001	73.2	73.44	73.69
SGDR L1 min df=20	73.36	74.20	75.7

Видим, что увеличение коэффициента регуляризации, хотя и оставляет более значимые слова, отсекая те, которые на глаз кажутся неважными, на самом деле приводит к существенной потере качества. Минимальное значение df тоже было решено оставить равным 10.

Далее было опробовано применение SVM на словаре *L1 vocab*. Результаты получились такими:

	Мин.	**Среднее**	**Макс.**
SGDR L1	73.94	75.01	76.15
SVM + L1 vocab	73.11	74.57	76.04

Здесь SVM оказался чуть хуже.

Далее в модель были добавлены биграммы. Ожидалось, что это даст некоторый прирост, однако качество получилось хуже, чем с униграммами. Предполгая, что это вызвано увеличением размера словаря, была предпринята попытка увеличить коэффициент регуляризации в 2 раза. Тем не менее результаты получились следующими:

	Мин.	**Среднее**	**Макс.**
SGDR L1	73.94	75.01	76.15
SGDR L1 bigram	73.44	74.35	75.06
SGDR L1 bigram alpha=0.0002	73.53	74.28	75.51

Биграммы все равны не заработали. После изучения коэффициентов стало понятно, что добавление биграмм приводит к появлению множества коррелированных признаков. Для решения этой проблемы, как уже было сказано ранее, был использован стемминг после лемматизации, а для смягчения влияния биграмм было сделано следующее. Из тренировочных текстов были извлечены только биграммы, после чего на них была запущена SGD-регрессия с коэффициентом L1-регуляризации, равным 0.001 (в 10 раз больше стандартного). В результате осталось всего порядка 400 биграммных токенов, которыми был расширен словарь *L1 vocab*.

На новом словаре снова была запущена сначала сама SGD-регрессия, а потом SVM. Результаты получились следующими:

	Мин.	**Среднее**	**Макс.**
SGDR L1 lemmatization	73.94	75.01	76.15
SGDR L1 lemmatization + stemming + bigram adapted L1 vocab	73.38	74.11	75.34
SVM lemmatization + stemming + bigram adapted L1 vocab	74.0	75.38	76.18

SGD-регрессия даже с исправленным словарем проигрывает более простой версии, однако SVM на этот раз заработал очень здорово.

Далее был построен словарь *TF vocab*. Применение к нему SVM дало следующее:

	Мин.	**Среднее**	**Макс.**
SGDR L1 lemmatization	73.94	75.01	76.15
SVM lemmatization + stemming + bigram adapted L1 vocab	74.0	75.38	76.18
SVM lemmatization + stemming + TF vocab	74.02	75.11	75.51

Качество получилось все же похуже, чем у *L1 vocab*, поэтому в дальнейшем использовался именно *L1 vocab*.

Далее была попытка обучить на этом словаре ансамбль деревьев, а также взять результат SVMa, добавить его в качестве признака и обучить ансамбль деревьев на расширенных данных. Результаты получились такими:

	Мин.	**Среднее**	**Макс.**
SVM	74.0	75.38	76.18
AdaBoost	73.98	74.58	75.11
SVM + AdaBoost	73.52	74.95	75.6

Деревья сработали все-таки хуже, чем SVM, а при попытке их скрестить получилось что-то среднее. Поэтому далее хотелось улучшить результат с использованием SVM. Для этого было построено сначала 5, а потом 10 SVMов, каждый из которых обучался на бутстрапленной выборке из тренировочного набора, после чего результат считался как среднее по всем. Были получены следующие результаты:

	Мин.	**Среднее**	**Макс.**
SVM	74.0	75.38	76.18
5 SVMs	74.73	75.63	76.43
10 SVMs	74.59	75.66	76.43

Ансамбль SVMов действительно дал существенный прирост результатов, но при это увеличение их количества с 5 до 10 ситуацию не изменило.

6 Нереализованные идеи

В первую очередь стоит сказать, что скорее всего для подобной задачи должны хорошо сработать какие-нибудь глубокие нейронные сети (наверное, Transformerы, они в последнее время всюду в

NLP). Но, к сожалению, опыт обучения таких моделей отсутствовал, а времени его получать особо не было.

Много интересного в данной задаче можно придумать в препроцессинге. Например, есть признаки, такие как восклицательные знаки, слова *очень*, *заметно* и подобные, написание слов заглавными буквами, которые свидетельствуют о сильной эмоциональной окраске, но ничего не говорят о том, положительная она или отрицательная. Такие признаки плохо вписываются в линейную модель, поэтому нужно либо каким-либо образом вносить нелинейность, либо сделать на основе таких признаков отдельную модель, предсказывающую силу эмоциональной окраски, а потом смешать эту модель с обычной регрессией.

7 Выводы

По итогам проделанной работы сделаем несколько выводов:

1. Даже обычная линейная регрессия с простой предобработкой дает не самый плохой результат.

2. Задача определения тональности требует довольно-таки нетривиального препроцессинга по сравнению с многими другими классическими задачами.

3. Для обучения регрессии очень важно правильно построить словарь, с помощью которого будет производиться векторизация.