Математическая модель предсказания результатов рассылки

Постановка задачи машинного обучения

Наша задача как компании – доставлять наиболее релевантные предложения в наиболее привлекательной для людей форме. Для этого необходимо, чтобы подписчика заинтересовало предложение в рассылке. Мы хотим улучшить её качество: повысить процент открытий письма, повысить процент переходов по ссылке. В отличие от рекламы, например в ВК, мы не можем протестировать некоторые гипотезы на небольшом бюджете в 100 рублей. Разосланную рассылку не отменить. Поэтому появляется необходимость количественной оценки качества письма, построения некоторой прогнозной модели. Вручную очень сложно оценить нелинейные, комплексные взаимосвязи на большом количестве данных. На помощь приходят методы машинного обучения.

Перед нами стоит задача обучения с учителем (supervised learning). У нас есть обучающая выборка (датасет с результатами старых рассылок) с целевой переменной (результат отправки). Его мы делим на два набора данных: тренировочный и тестовый. Необходимо выбрать подходящую модель, обучить её на тренировочном наборе, затем проверить на тестовом. Если результат по выбранной метрике устроит нас, то можно использовать обученную модель для предсказаний результатов следующих рассылок.

Описание датасета, признаков

В представленном наборе данных имеем 18336 наблюдение и 23 признака.

Имеется следующий набор признаков: ID рассылки, Клиент, Тема, Тип предложения, Текст рассылки, Картинка в шапке, Email, Телефон, Результат отправки, ВУЗ, Имя профиля, Специальности, Уровень английского, Программа обучения, Пол, Год выпуска, Город, Факультет, utm_source, personal_data, utm_medium, utm_campaign, utm_content.

Результат отправки является нашей целевой переменной, в предоставленном датасете имеет следующие возможные принимаемые значения:

```
1 df['Результат отправки'].value_counts(dropna=False)
```

```
ok_delivered                17597
ok_read                       647
NaN                            68
ok_link_visited                10
err_delivery_failed             6
skip_dup_unreachable            3
ok_unsubscribed                 2
skip_dup_temp_unreachable       1
Зарегистрировался               1
err_mailbox_full                1
Name: Результат отправки, dtype: int64
```

Как видно на картинке, имеется дисбаланс классов: подавляющее большинство писем имеет статус «доставлено», чуть меньшее количество прочитано, и ещё меньшее количество переходов. Остальные варианты нам не важны и их можно отбросить (возможно кроме ok_unsubscribed – но там слишком мало наблюдений). Кроме того, так же есть пропущенные значения, наблюдения с ними так же придется отбросить.

Таким образом, перед нами стоит задача мульти классовой классификации с тремя несбалансированными классами.

Оставшиеся признаки можно разделить на 2 основные группы:

1. Признаки, относящиеся к письму: ID рассылки, Клиент, Тема, Тип предложения, Текст рассылки, Картинка в шапке
2. Признаки, относящиеся к пользователю: Email, Телефон, ВУЗ, Имя профиля, Специальности, Уровень английского, Программа обучения, Пол, Год выпуска, Город, Факультет.

Так же имеются дополнительные признаки, которые в датасете не заполнены: utm_source, personal_data, utm_medium, utm_campaign, utm_content.

Сразу можно отбросить признаки, которые не дадут модели значимой информации: ID рассылки (не важно, к какой рассылке принадлежит письмо, важен его контент), Email или телефон (необходимо выбрать что-то одно для идентификации пользователя, т.к. эти признаки имеют четкую корреляцию), имя профиля (по имени человека с научной точки зрения выводов сделать не получится).

```
ID рассылки             0.000000
Клиент                  0.000000
Тема                    0.000000
Тип предложения         0.000000
Текст рассылки          0.000000
Картинка в шапке        0.000000
Email                   0.000000
Телефон                70.337042
Результат отправки      0.370855
ВУЗ                    38.907068
Имя профиля            24.001963
Специальности          45.517016
Уровень английского    49.258290
Программа обучения     75.578098
Пол                    70.211606
Год выпуска            26.663394
Город                  27.268761
Факультет              38.678010
utm_source             99.121946
personal_data          99.683682
utm_medium             99.438264
utm_campaign           99.438264
utm_content            99.487347
dtype: float64
```

На картине изображён % пропущенных значений по признакам ко всему датасету. Например, номер телефона неизвестен у 70% пользователей, а ВУЗ только у 38,9%.

Можно сделать вывод, что лучше использовать email в качестве идентификатора пользователя, так как он известен у всех, в отличие от номера телефона.

Тем не менее, имеем, что в большинстве признаков, относящихся к пользователям, большое количество пропущенных данных. К сожалению, такие признаки как **программа обучения, пол, уровень английского, специальности** скорее всего придётся выкинуть. Так же выкинем **телефон** и **utm-метки**. Так же выбросим признаки, упомянутые раннее (**имя профиля, id рассылки**).

Получились следующие признаки с определенным процентом пропущенных данных:

```
Клиент                0.000000
Тема                  0.000000
Тип предложения       0.000000
Текст рассылки        0.000000
Картинка в шапке      0.000000
Email                 0.000000
Результат отправки    0.370855
ВУЗ                  38.907068
Год выпуска          26.663394
Город                27.268761
Факультет            38.678010
dtype: float64
```

Таким образом, осталось всего лишь 11 признаков. Тем не менее, мы сможем достать множество признаков из текста, так что это не проблема.

Теперь поступим следующим образом: сделаем копию нашего датасета с оставшимися 11 признаками и выкинем из неё все строки с пропущенными значениями.

```
:  1 df_new_no_na = df_new.dropna().copy()
   2 df_new_no_na.shape
   3 # df_new_no_na.isnull().sum()/len(df_ne
```

```
:  (5369, 11)
```

Таким образом, остаётся лишь 5369 наблюдений из исходных 18336 или 29% от исходных. Это говорит о том, что несмотря на то, что максимально кол-во пропущенных значений по признаку 39%, пропущенные данные по разным признакам встречаются в разных наблюдениях. Это достаточно существенная проблема.

Тем не менее, можно попробовать обработать пропущенные значения, например заполнить модой – наиболее часто встречающимся значением.

В другой части датасета отбросим все признаки с NA, т.е., по сути, оставим признаки, содержащие информацию только о письме.

Затем попробуем построить модели на обеих копиях датасета.

Обработка признаков

Мы можем повлиять на содержание письма: тему, текст рассылки и картинку в шапке. Поэтому встаёт задача учесть именно их. Картинку обрабатывать сложно, и по опыту знакомого middle DS-разработчика в Яндексе, она зачастую не даёт важную информацию в подобных задачах. Остаётся текст и тема. Для обработки текста существует различное множество методов, которые сводятся в превращение его в количественную информацию. Основные подходы: tf-idf, w2v, эмбеддинги. Последний подход достаточно сложный, но позволяет добыть достаточно большое количество дополнительных признаков (сотни). Но так как у нас признаков относительно много, отбросим его и попробуем более простые подходы, связанные с текстом.

Обработка текста

Основная проблема заключается в том, что текст — это набор объектов переменной длины, состоящих из букв. Хочется преобразовать этот набор символов в фиксированное количество признаков. Для этого можно использовать подход под названием **мешок слов**. В нём текст кодируется в виде количества вхождений каждого слова из словаря в документ. Этот метод хорошо себя зарекомендовал, и часто используется в задачах машинного обучения.

Но перед этим, необходимо **предобработать** текст: разбить текст на отдельные слова («токенизация»), привести слова к начальной форме (чтобы кот и кошка не считались разными словами) и убрать стоп-слова (предлоги, союзы и т.д.), которые не несут информации.

Предобработка текста

Токенизация — это разбиение непрерывной строки на отдельные «слова». Токенизация состоит из нескольких этапов. В первую очередь текст приводится к **нижнему** регистру. При этом, опять же, можно потерять часть информации. Например, «ООО» может являться сокращением (общество с ограниченной ответственностью), а «ооо» — просто выражением эмоций.

Следующий этап — это **замена всех знаков препинания** и прочих символов на пробелы. И снова на этом этапе возникает много нюансов. Как было упомянуто выше, при наличии сложных составных слов (например, «красно-чёрный») заменять в них дефис на пробел не очень разумно: может потеряться смысл слова. Кроме того, удаление знаков препинания приводит к удалению смайликов. Они могут играть большую роль при анализе твитов или записей в соц.сетях, например, для определения эмоционального окраса текста.

После этого каждое слово, отделённое пробелом, объявляется **отдельным токеном**. Здесь тонкость заключается в том, что некоторые наборы слов должны рассматриваться как одно. Например, названия городов («Нижний Новгород»), или сокращения («к.т.н.», кандидат технических наук, это полезный термин при рассмотрении трёх букв вместе, но по отдельности они никакой информации не несут).

Тем не менее, вряд ли у нас возникнут такие сложности в наших текстах.

Следующий этап после токенизации — это нормализация слов в тексте, то есть приведение каждого слова к его начальной форме. Например, слово «машинное» необходимо привести к слову «машинный», «шёл» — «идти». Форма слова не всегда несёт в себе полезную информацию, при этом в задачах, где данных не очень много, хотелось бы использовать как можно меньшее число признаков. Их количество напрямую зависит от количества различных слов, поэтому было бы здорово сократить их число. Приведение слов к нормальной форме позволяет это сделать. Существует два основных подхода к нормализации: **стемминг** и **лемматизация**.

Стемминг — это самый простой метод нормализации. Его суть состоит в том, что слова «стригутся»: по некоторым правилам от каждого слова отрезается его окончание. К сожалению, данный подход не всегда работает. Например, некоторые слова при изменении формы меняются целиком: «был», «есть», «будет». Понятно, что из этих трёх слов невозможно получить одно, отрезая буквы.

По описанной выше причине чаще пользуются более сложным подходом, **лемматизацией**. В нём используется словарь, в котором уже записано большое

количество слов и их форм. В первую очередь слово проверяется по словарю. Если оно там есть, то понятно, к какой форме его приводить. Иначе по определённому алгоритму выводится способ изменения данного слова, на основании него делаются выводы о начальной форме. Этот подход работает лучше, и он подходит для новых неизвестных слов. Однако лемматизация сложнее, поэтому работает гораздо **медленнее**, чем стемминг.

Извлечение признаков из предобработанного текста

Как уже отмечалось ранее, для текстов очень неплохо работает подход под названием **мешок слов**. В этом методе текст рассматривается как неупорядоченный набор слов. Используя такой подход, имеет смысл обращать внимание на два нюанса. Первый — это **стоп-слова**. Это популярные слова, встречающиеся в каждом тексте (например, предлоги или союзы) и не несущие в себе никакой информации, а только засоряющие признаки. Их стоит удалять ещё при предобработке, например, на этапе токенизации. Кроме того, имеет смысл удалять **редкие слова**. Если какое-то слово входит только в 1 или 2 текста, то, скорее всего, не получится его значимо учесть в модели и оценить, какой вклад оно вносит в целевую переменную.

TF-IDF (text frequency - inverse document frequency) — это подход к формированию вектора признаков. Считается, что если слово часто встречается в тексте, и оно не является стоп-словом, то, скорее всего, оно важно. Но есть и вторая тонкость. Если слово встречается в других документах реже, чем в данном, то и в этом случае, скорее всего, оно важно для текста. По этому слову можно отличить этот текст от остальных. Если учесть описанные выше соображения, в результате получится подход TF-IDF.

Значение признака для слова **w** и текста **x** вычисляется по следующей формуле:

$$\mathrm{TF-IDF(x,w)} = n_{dw} \ln \frac{l}{n_w}$$

n_{dw} - доля вхождений слова w в документ d. Если слово часто встречается в тексте, то оно важно для него, и значение признака будет выше. Второй множитель называется IDF (inverse document frequency, обратная документная частота), это отношение l, общего количества документов, к n_w , числу документов в выборке, в которых слово w встречается хотя бы раз. Если это отношение велико, слово редко встречается в других документах, значение признака будет увеличиваться. Если слово встречается в каждом тексте, то значение признака будет нулевым $\ln \frac{l}{l} = \ln 1 = 0$.

Метод **«мешок слов»,** о котором шла речь ранее**, никак не учитывает порядок слов** в документе, а лишь подсчитывает, сколько раз каждое слово встретилось в тексте.

Этот подход не очень хорош. Пусть в тексте есть слова «нравится» и «не нравится», у них противоположный смысл. «Мешок слов» никак не поможет уловить эти смыслы: будет только информация о том, что в тексте есть слово «нравится» и частица «не». Это не несёт много информации.

Более того, если в тексте учитывать не только слова, но и словосочетания, то признаковое пространство становится более обширным. При этом появляется возможность находить более сложные закономерности, используя простые модели (аналогично при добавлении новых признаков линейные модели могут находить более сложные разделяющие поверхности).

Учёт порядка слов

Использование n-грамм — это самый простой способ учитывать порядок слов. По сути n-грамма — это последовательность из n идущих подряд слов в тексте. Для предложения «Наборы подряд идущих токенов» существуют следующие n-граммы:

- униграммы(n = 1): наборы, подряд, идущих, токенов;
- биграммы (n = 2): наборы подряд, подряд идущих, идущих токенов;
- триграммы (n = 3): наборы подряд идущих, подряд идущих токенов.

После того как в тексте найдены все требующиеся n-граммы, используются те же подходы, что и для мешка слов: подсчитываются счётчики, вычисляются TF-IDF.

Чем больше n, до которого будут найдены n-граммы, тем больше признаков получится, признаковое пространство будет более богатым. При этом n — это гиперпараметр, и его увеличение может привести к переобучению. Например, если n — это длина текста, то у каждого документа будет уникальный признак, и алгоритм сможет переучиться под обучающую выборку.

Буквенные n-граммы

N-граммы можно использовать не только на словах. В качестве токенов можно рассматривать отдельные символы в предложении. После нахождения таких токенов для них можно также вычислять n-граммы (буквенные n-граммы). В качестве признаков, как и раньше, выступают TF-IDF.

У этого подхода есть много преимуществ. Он позволяет учитывать смайлы или известные слова в незнакомых формах. Часто буквенные n-граммы используют вместе с n-граммами по словам.

Skip-граммы — это чуть более расширенный подход к использованию n-грамм. Более точно они называются k-skip-n-граммы, это наборы из n токенов, причём расстояние между соседними должно составлять не более k токенов.

Обучение моделей на тексте

Как уже было сказано ранее, прежде чем начать обучать модель, необходимо подготовить текст: сформировать выборку и вычислить признаки. Как правило, для этого сначала из текста удаляются слишком редкие или популярные слова (стоп-слова). Затем вычисляются признаки. Это могут быть n-граммы, skip-граммы, на которых вычисляются TF-IDF.

Как правило, размерность полученного пространства признаков оказывается очень **большой**. При использовании униграмм **количество** признаков может составлять $10^3 - 10^4$, если добавить биграммы и триграммы, то при использовании большого корпуса текстов количество признаков может достигать $10^6 - 10^7$. Поэтому можно пробовать отбирать признаки, находить только те униграммы, биграммы и т.д., которые **коррелируют** с целевой переменной. Кроме того, можно понижать размерность. Например, использовать **метод главных компонент** для перехода в пространство меньшей размерности, в котором всё ещё будет сохраняться хорошее признаковое описание.

С большим количеством признаков есть некоторые проблемы: желательно, чтобы наблюдений было не меньше числа признаков, а кроме того, не все модели хорошо работают в таком случае.

Выбор модели

Скорее всего, размерность признакового пространства не получится радикально уменьшить, и признаков будет очень много. Нужно понять, как в такой ситуации будут работать различные методы машинного обучения.

В **случайном лесе** используются деревья большой глубины, они строятся до тех пор, пока в каждом листе не окажется очень мало объектов. Если признаков очень много, то этот подход работает не очень хорошо: деревья будут очень глубокими, на их построение будет уходить слишком много времени, и можно просто не дождаться, когда лес достроится до нужного размера.

При использовании **градиентного бустинга** решающие деревья, наоборот, имеют очень небольшую глубину. Но это тоже проблема. Каждое дерево может учесть лишь небольшое подмножество признаков, в то время как зачастую ответ зависит от комбинации большого количества слов в документе. Поэтому для хорошей работы градиентного бустинга нужно будет использовать очень много деревьев, но даже в этом случае нет гарантии, что полученное качество будет приемлемым.

Байесовские методы обычно хорошо себя показывают при работе на текстах. Однако чаще всего на текстовых данных используются **линейные модели**. Они хорошо масштабируются, могут работать с большим количеством признаков, на очень больших выборках.

Word2vec

Тем не менее, описанное выше применимо к подходу, основанному на мешке слов. Существует так же и другой популярный подход.

Второй подход — это word2vec, обучение представлений слов. В нём для получения представления слова используется его контекст, то есть слова, рядом с которыми оно часто встречается. В этом случае появляется некоторый порядок. Однако признаковое описание текста получается усреднением признаковых описаний входящих в него слов, и порядок слов снова никак не учитывается.

Слова «идти» и «шагать» — это синонимы, они имеют похожий смысл. Однако для компьютера это просто строки, причём не очень похожие: у них разная длина, разные буквы. Просто по этим двум строкам компьютер не может сказать, что они имеют одинаковый смысл.

При анализе текстов хочется уметь оценивать, насколько похожи различные пары слов. Это можно сделать, используя текстовые данные. Оказывается, слова со схожим смыслом часто встречаются рядом с одними и теми же словами. Другими словами, похожие слова имеют одинаковые контексты. На этом принципе основан метод word2vec.

Итак, требуется описать каждое слово w с помощью вектора $\vec{w}$ размерности d:

$$w \to \vec{w} \in \mathbb{R}^d$$

При этом хочется иметь следующие свойства:

- **Компактные** представления слов, то есть величина d должна быть не очень большой.
- **Похожие** слова должны иметь **близкие** векторы (например, по евклидовой или косинусной метрике).
- К числовым векторам можно применять **арифметические** операции: складывать, вычитать, умножать на коэффициент. Хотелось бы, чтобы эти операции имели смысл.

Можно показать, что полученные представления будут обладать всеми вышеперечисленными свойствами.

При поиске вектора $\vec{w}$ работа будет происходить с вероятностями

$$p(w_i|w_j) = \frac{e^{\langle w_i, w_j \rangle}}{\sum_w e^{\langle w_i, w \rangle}}$$

Эта функция показывает, насколько вероятно встретить слово w_i в контексте слова w_j , то есть рядом.

Векторные представления слов будут настраиваться так, чтобы вероятности встретить слова, находящиеся в **одном контексте**, были высокими. В функционал для каждого слова входят вероятности встретить его вместе с k словами до и после него:

$$\sum_{i=1}^{n} \sum_{j=-k}^{k} \log p(w_{i+j}|w_i) \rightarrow max$$

Оптимизировать этот функционал можно с помощью стохастического градиентного спуска.

Если обучиться на большом корпусе текстов, то окажется, что векторы представлений имеют много интересных свойств. В частности, если слова похожи по смыслу, то соответствующие им векторы близки по косинусной метрике.

Что касается арифметических операций, то наблюдается следующее:

$$king - man + woman \approx queen$$

$$Moscow - Russia + England \approx London$$

Более того, оказывается, такой подход можно использовать для перевода слов. Если обучить векторы одновременно на корпусе английских и испанских текстов, то

$$one - uno + four \approx quatro$$

Этот метод проигрывает методам, специально разработанным для машинного перевода, тем не менее с его помощью удаётся получать интересные результаты.

Наконец, от представлений отдельных слов можно попытаться перейти **к представлениям текстов**. Оно получается **усреднением** векторов всех слов, входящих в документ. С полученным признаковым описанием целого текста также можно работать.

Word2vec и обучение с учителем

Проблема мешка слов состоит в том, что получается пространство признаков слишком **большой** размерности (десятки и сотни тысяч признаков). Если же использовать подход word2vec, где размер вектора равен 100, то получится признаковое описание текста, размер которого также равен 100.

При таком количестве признаков становится возможным использовать любые методы машинного обучения (например, случайные леса и градиентный бустинг), а не только линейные модели.

Таким образом, используя данный подход, можно значительно **сжимать** представления текстов, и обучать **сложные** методы машинного обучения.

Рекуррентные нейронные сети — это ещё один способ работать с текстами, но тем не менее, отбросим его и остановимся на первых двух описанных выше.

Обработка оставшихся признаков

Вспомним, что мы имеем 2 датасета: датасет с 10 признаками, но 5369 наблюдениями и датасет с 5 признаками, но с 18336 наблюдениями.

Все оставшиеся признаки являются категориальными, т.е. принимают ограниченный набор значений: Клиент, Тип предложения, Email, ВУЗ, Год выпуска, Город, Факультет.

Для кодирования категориальных переменных существуют два подхода:

- **OneHotEncoder –** переменная кодируется как набор новых признаков, каждый из которых равен 0 или 1, что означает принадлежность тому или иному значению признака. Пример: имеем переменную «животное». Набор его возможных значений: кот, собака, жираф, слон. Таким образом, будет создано 4 новых признака, отвечающих за принадлежность определенному типу животного.
- **OrdinalEncoder –** каждое возможное значение признака кодируется как набор чисел, заданных по порядку от 0 до k, где k – число возможных значений.

Первый подход необходимо использовать, когда категориальная переменная принимает далекие друг от друга значения, например, кошка или собака. **Второй** подход подразумевает, что категориальные переменные можно упорядочить. Например, уровень образование: начальная школа < средняя школа < старшая школа < бакалавриат < магистратура < аспирантура.

Все наши категориальные переменные принимают значения, которые нельзя логически упорядочить. Будем использовать OneHotEncoder для всех признаков, кроме Email. Причина проста: слишком много уникальных значений. Для каждого необходимо будет создавать новый признак, что приведёт к их переизбытку. Поэтому можно пожертвовать некоторой логикой (вряд ли можно упорядочить почты) ради возможности использования этого признака.

Обработка данных

Тем не менее, перед кодированием необходимо так же внимательно посмотреть на данные и обработать их. Например, один и тот же город записан несколькими способами:

```
Москва      13318
москва         11
МОСКВА          2
```

```
Экономика                                  327
Менеджмент                                 244
Экономический                              139
Экономический факультет                     99
Международные отношения                     95
```

Или одно и то же направление разными способами.

```
НИУ ВШЭ
1348
Финансовый университет (бывш. ВЗФЭИ ВФ)
750
МГУ им. М.В. Ломоносова
656
РАНХиГС при Президенте РФ (бывш. АНХ при Правительстве РФ)
476
РЭУ им. Г. В. Плеханова
463

...
СибГМУ
1
Уральский федеральный университет
1
федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный лингвистически
й университет»       1
ЛГУ имени А.С.Пушкина
1
БФУ им. И. Канта
1
Name: ВУЗ, Length: 975, dtype: int64
```

Так же есть более часто встречающиеся ВУЗы и менее. Чтобы не плодить признаки, можно добавить редкие учебные заведения в отдельную группу.

Выбор метрики

Так как присутствует дисбаланс классов, лучше использовать balanced accuracy или f1-score

Итоговый план

1. Обработать оставшиеся признаки
2. Обработать текст: токенизация -> нормализация (лемматизация/стеминг) -> word2vec или TF-IDF -> текст в виде TF-IDF засунуть в линейную модель (логистическую регрессию), текст в виде word2vec засунуть в бустинг или случайный лес.
3. Обработать оставшиеся признаки (объединить, отбросить, закодировать)
4. На оставшихся признаках попробовать случайный лес или бустинг.
5. Результат моделей на текстах и оставшихся признаках засунуть в ансамбль (либо сразу запустить модель и на текстах, и на других признаках)
6. Посмотреть результат по выбранным метрикам