Санкт-Петербургский государственный университет

Кафедра технологии программирования

Хажоян Арсен Владимирович

Выпускная квалификационная работа бакалавра

Предсказание ссылок в социальных сетях

Направление 010400
Прикладная математика и информатика

Заведующий кафедрой:
кандидат физ.-мат. наук, доцент Сергеев С. Л.

Научный руководитель:
ст. преп. Мишенин А. Н.

Санкт-Петербург
2016

Содержание

Введение

Социальные сети являются популярным способом взаимодействия между пользователями и группами пользователей. Анализ социальных сетей имеет широкое применение в ряде дисциплин и приложений. Из распространённых приложений можно отметить такие, как анализ поведения пользователя, бизнес-аналитика и даже правоохранительные мероприятия (например, выявление скрытых преступных организаций).

Интерес к этой области в последнее время растёт, так как за последние десять лет социальные сети стали неотъемлемой частью нашей жизни. Почти у каждого человека есть аккаунт хотя бы в одной, а часто и в трёх–четырёх социальных сетях. Компании всё чаще используют анализ социальных сетей, чтобы предсказать поведение пользователя или построить качественную рекомендательную систему.

Во всех вышеперечисленных приложениях анализа социальных сетей так или иначе фигурирует предсказание ссылок. В этой работе мы проведём обзор некоторых методов и попробуем разные подходы к решению задачи предсказания ссылок в социальном графе.

Постановка задачи

Общая задача предсказания ссылок в социальном графе ставится следующим образом: дан социальный граф $G(V, E)$, в котором вершины обозначают различные сущности – пользователей социальной сети, группы, ссылки на другие ресурсы, разнообразный медиа-контент (видео- и аудиозаписи, фотографии). Рёбра $e = (u, v)$ обозначают какую–либо форму взаимодействия между этими сущностями в определенный момент времени t. Например, взаимодействия вида "пользователи A и B добавили друг друга в друзья в ДД.ММ.ГГ" или "пользователь C посетил сайт `example.com` неделю назад". Задача состоит в том, чтобы предсказать взаимодействия между сущностями в заданный момент времени, в том числе и будущем времени.

Мы рассмотрим более узкую задачу. В нашем случае вершины графа представляют собой пользователей, а рёбра — взаимодействия вида "пользователи A и B состоят в друзьях". Заметим, что момент времени, в котором это взаимодействие было совершено, в постановке задачи не фигурирует. Задача состоит в том, чтобы предсказать наличие связей между пользователями, связи между которыми были скрыты.

Обзор литературы

В ходе проведения работы очень полезной оказалась обзорная статья о проблеме предсказания ссылок [6]. Из неё были подчерпнуты основные идеи, рассмотренные в работе.

Много деталей использованных методов машинного обучения я узнал из книги The Elements of Statistical Learning [5] и из лекций К. В. Воронцова на сайте `machinelearning.ru` [13]. Эти источники очень помогли разобраться в используемых классификаторах. Помимо этого, помог раздел сайта `kaggle.com`, посвященный метрикам [7]. Так же оказалась полезной статья, посвящённая подбору гиперпараметров модели машинного обучения [2].

В процессе применения методов часто приходилось обращаться к документациям SciPy, sklearn и XGBoost. Так же при реализации персонализированного PageRank оказалась полезной статья Википедии [12].

Глава 1. Данные и их обработка

Данные для экспериментов были получены из проходившего в 2016 году SNAHackathon [9].

1.1. Граф

Вершинами графа являются пользователи социальной сети. Рёбра в графе представляют собой взаимодействие между пользователями вида "пользователи A и B являются друзьями". Кроме того, представлен "характер" этой дружбы, например, являются ли пользователи просто друзьями, или родственниками, или коллегами, и так далее.

В этих данных, помимо самого графа, дана информация о самих пользователях:

- дата создания аккаунта;
- дата рождения пользователя;
- пол пользователя;
- страна и город, указанные в профиле;
- локация, откуда пользователь чаще заходит в социальную сеть;

1.2. Вид данных

Граф был представлен в виде разреженной матрицы, партиционированной на 16 файлов, сжатых утилитой gzip. В каждом из этих файлов построчно дана информация о связях пользователей:

ID_пользователя1 {(ID_друга1,маска1), (ID_друга2,маска2),...}

где маска — битовая маска, кодирующая информацию о виде связи.

Всего пользователей в графе около одного миллиона. Данные о пользователях представлены в виде CSV файла, партиционированного аналогичным образом. Его строки выглядят следующим образом:

userId create_date birth_date gender ID_country ID_Location loginRegion

1.3. Обработка данных

Использованные для обработки данных инструменты: Python, NumPy, SciPy, pandas.

Часть социального графа (8 из 16 файлов) была обработана и загружена в память в виде разреженной матрицы `csr_matrix` из модуля `scipy.sparse`. В SciPy есть несколько видов разреженных матриц [10]. Нам больше подходит именно `csr_matrix`, потому что этот вид разреженных матриц обладает высокой скоростью матричного умножения и позволяет быстро работать со строками матрицы.

Обработка состояла из первичного прохода по этим файлам с целью создать структуру данных, отображающую id пользователей во множество $\{0..N\}$, где N – количество пользователей, информация о которых содержится в этих 8 файлах. Сделано это для того, чтобы обрезать рёбра, не ведущие в $\{0..N\}$ и таким образом снизить размерность матрицы. Далее, во время основного прохода по файлам, заполнялась разреженная матрица.

Затем случайным образом были скрыты 5% рёбер в графе.

После того, как граф загружен в память и у нас есть отображение из "старых" идентификаторов в "новые", загрузим данные о пользователях в формат pandas dataframe.

1.4. Анализ и выбор метрики

Перед нами стоит задача обучения с учителем, а именно — задача бинарной классификации. Классы в данном случае обозначают отсутствие/наличие связи между двумя пользователями — 0 и 1 соответственно. Посмотрим на рисунок 1. Видно, что у большинства пользователей количество друзей близко к нулю.

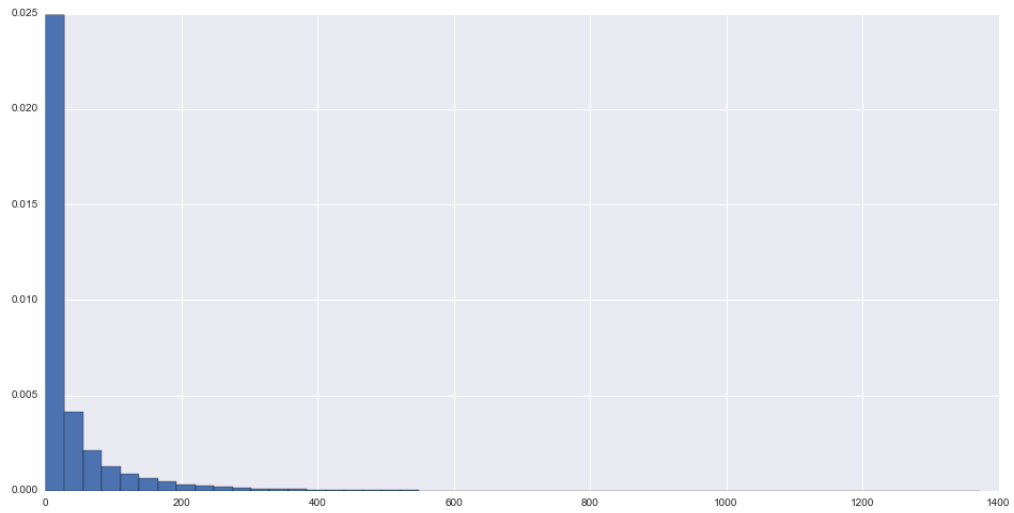

Рис. 1: Гистограмма количества друзей у пользователей

Таблица 1: Виды ответов бинарного классификатора

	Правильный ответ: 0	Правильный ответ: 1
Предсказание: 0	TN (true negative)	FN (false negative)
Предсказание: 1	FP (false positive)	TP (true positive)

Среднее количество друзей — 43.78 в полном графе и 41.5 в графе со скрытыми рёбрами.

Так как классы несбалансированы, нужно правильно выбрать метрику. Довольно интуитивная и популярная метрика — accuracy (отношение количества правильных ответов к количеству всех ответов):

$$A = \frac{TN + TP}{TN + FN + TP + FP}$$

Эта метрика в случае несбалансированных классов будет давать нерелевантный результат. Если, к примеру, классификатор всегда будет давать ответ 0, то значение accuracy в таком случае будет близким к $\frac{N_0}{N_0+N_1} = \frac{N_0}{N}$ (отношение количества элементов класса 0 к размеру тестового множества). Этот недостаток метрики так же известен, как accuracy paradox [11].

Так как наше тестовое множество ожидается несбалансированным, правильнее будет использовать метрику, нечувствительную к несбалансированным классам, например, AUC–ROC — площадь под ROC-кривой. Кривая представляет собой зависимость True Positive Rate ($TPR = \frac{TP}{TP+FN}$) от False Positive Rate ($FPR = \frac{FP}{FP+TN}$). Пример ROC-кривой можно увидеть на графике 2.

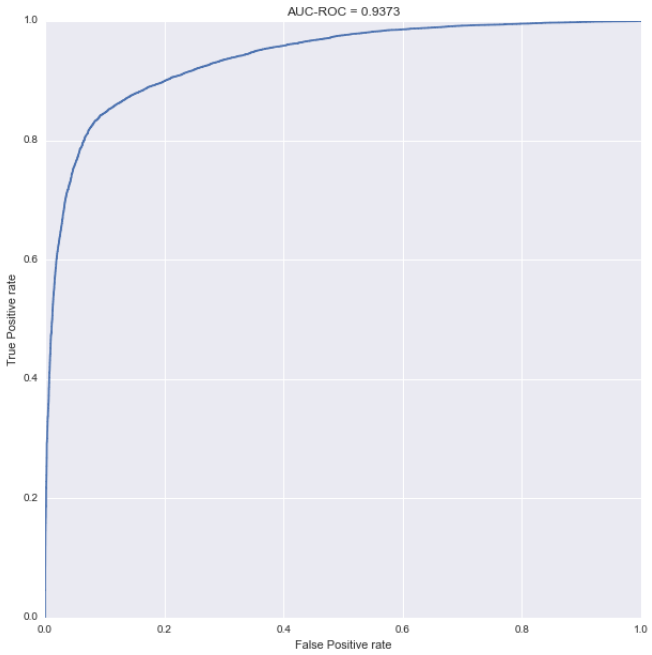

Рис. 2: ROC-кривая

Глава 2. Построение обучающего и тестового множества

Итак, поставлена задача бинарной классификации с метрикой AUC–ROC. Данные представлены в виде разреженной матрицы, содержащей информацию о связях между пользователями, и структурой данных pandas dataframe, содержащей информацию о пользователях. Нужно построить обучающее и тестовое множества.

2.1. Предпосчёт количества общих друзей

Многие подходы к решению этой задачи так или иначе используют количество общих друзей. Рассмотрим пример социального графа на рисунке 3. Представим его в виде таблицы:

0	1	1	1
1	0	0	1
1	0	0	0
1	1	0	0

Заметим, что матрица связей F, соответствующая этой таблице, симметрична, т.е. $F = F^T$.

Рассмотрим пользователей 2 и 1. У них один общий друг — пользователь 0. Несложно заметить, что это можно вычислить скалярным произведением:

$$(1, 0, 0, 1)(1, 0, 0, 0)^T = 1$$

Очевидно, всю матрицу количества общих друзей C можно посчитать следующим образом:

$$C = FF^T = F^2$$

Для разреженной матрицы `scipy.sparse.csr_matrix` размера $[500000 \times 500000]$ эта операция на моём компьютере занимает около двух

минут.

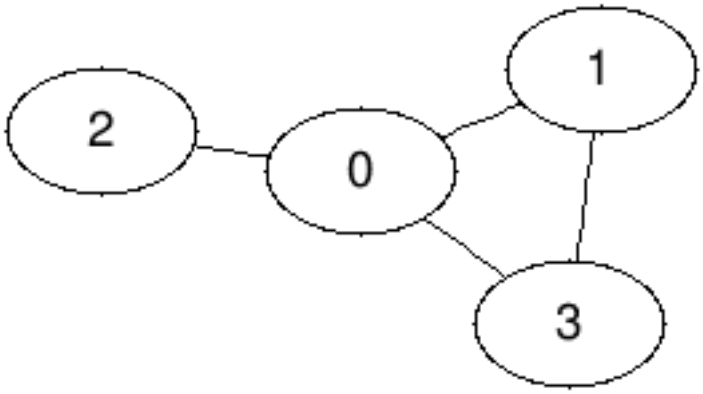

Рис. 3: Пример социального графа

2.2. Обучающее и тестовое множество

Из описанных выше матриц и pandas dataframe нужно сформировать матрицу признаков X и вектор ответов y.

Зафиксируем пользователя u и сформируем обучающее множество на основе его связей с остальными пользователями. Для него вектором ответов y будет соответствующая строка в матрице F. Сформируем матрицу X из следующих признаков:

- количество общих друзей двух пользователей;
- разница в возрасте;
- факт совпадения или различия полов;
- факт совпадения или различия города;
- факт совпадения или различия страны;
- факт совпадения или различия локации, из которой пользователь заходит чаще всего;

Количества общих друзей — просто строка в матрице C. Разница в возрасте получается вычитанием из соответствующего столбца

dataframe значения возраста пользователя u: $|\text{age} - \text{age}_u|$. Все остальные признаки получаются сравнением остальных столбцов dataframe с соответствующим значением информации о пользователе u и приведением результата к числовому типу.

Получили матрицу признаков X_u и вектор ответов y_u с информацией о пользователе u. Далее таким же образом можно собрать информацию о других пользователях и соединить получившиеся результаты в одну матрицу X и один вектор y.

Вспомним, что классы несбалансированы. Чтобы классификатор одинаково хорошо обучался на обоих классах, можно сформировать сбалансированную выборку, то есть такую, в которой количество элементов класса 0 и количество элементов класса 1 были примерно равны. Этого можно добиться, например, следующим образом: в каждой паре (X_i, y_i) случайным образом удалить столько элементов класса 0, чтобы оставшиеся элементы оказались сбалансированы.

Итак, выберем случайно несколько пользователей и построим матрицу X и вектор ответов y. Затем выделим строки, касающиеся скрытых связей, и вынесем их в тестовое множество. Добавим в тестовое множество некоторое количество элементов из класса 0. Оставшиеся части X и y будет обучающим множеством.

Глава 3. Подбор гиперпараметров

У многих методов, которые будут использованы, есть набор гиперпараметров, которые бывают непрерывные, целочисленные и категориальные. Для максимального качества модели эти параметры необходимо подобрать.

3.1. Полный перебор

Самый популярный для этого метод — перебор по сетке. Для каждого гиперпараметра задаётся некий набор значений, и затем перебираются все комбинации этих значений для разных гиперпараметров. Для фиксированной комбинации совершается k-fold кросс-валидация, и мы получаем k значений качества для каждой комбинации. Таким образом, можно посчитать не только среднее значение качества для комбинации, но и дисперсию.

Часто этот способ сопряжен с вычислительными затратами, так как приходится перебрать 5-10 (иногда и 50) комбинаций и выбрать среди них лучшую. Кроме того, для каждой комбинации нужно совершить обучение и предсказание k раз.

Для полного перебора будет использован класс `GridSearchCV` из пакета scikit–learn.

3.2. Случайный перебор

Другой способ для этого — случайный перебор [2]. Для каждого гиперпараметра задаётся вероятностное распределение, из которого будут сэмплироваться значения. Для случайного перебора будет использован класс `RandomSearchCV` из пакета scikit-learn.

Запустим несколько раз случайный перебор и полный перебор с разным количеством комбинаций и посмотрим на качество на графике 4. Можно сделать вывод, что с небольшим количеством комбинаций случайный поиск в среднем будет выгоднее. Таким образом, в большинстве

экспериментов будет оптимальнее использовать случайный перебор, а после выбора самой производительной модели лучше совершить полный перебор по большой сетке параметров.

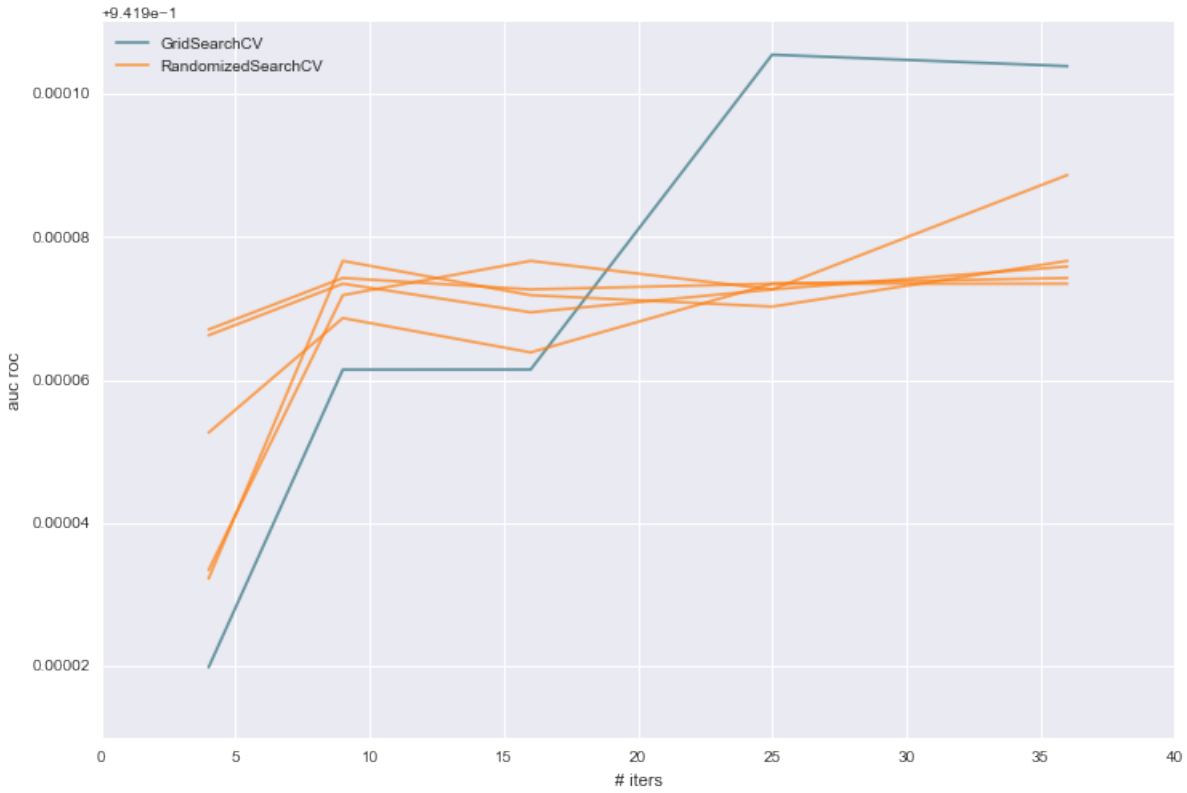

Рис. 4: Сравнение полного и случайного переборов гиперпараметров

Глава 4. Методы машинного обучения

Рассмотрим несколько методов машинного обучения, которые будут использованы в дальнейших экспериментах.

4.1. Метод k ближайших соседей

Метод k ближайших соседей (kNN) — один из простейших методов машинного обучения, относится к так называемым метрическим алгоритмам классификации, основанным на вычислении метрик между элементами [5].

В своей простейшей реализации метод ищет ближайшие k объектов и относит классифицируемый объект к тому классу, которому принадлежит большинство ближайших к нему объектов обучающей выборки. Существует вариация этого метода — метод k взвешенных ближайших соседей. В этой вариации при подсчёте элементов различных классов среди соседей этим элементам присваиваются веса, обратно пропорциональные расстоянию от классифицируемого элемента.

В общем виде формула ответа этого классификатора выглядит так:

$$a(u) = \arg\max_{y \in Y} \sum_{i=1}^{m} \big[y(x_{i;u}) = y\big] w(i,u),$$

где $x_{i;u}$ обозначает i-ый ближайший сосед элемента u. $w(i,u)$ представляет собой весовую функцию. В случае обычного метода k ближайших соседей эту функцию можно задать таким образом:

$$w(i,u) = [i \leq k]$$

В случае метода k взвешенных ближайших соседей:

$$w(i,u) = [i \leq k] q^i; \quad q \in (0,1)$$

В качестве реализации используем класс `KNeighborsClassifier` из пакета scikit–learn. Важные гиперпараметры модели:

Название	Описание
`n_neighbors`	Параметр k — количество ближайших соседей, учитывающихся при выборе класса
`metric`	Метрика, используемая для определения ближайших соседей. Часто используется метрика Минковского $\rho(x, y) = (\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p)^{1/p}$ с разными p.
`weight`	Взвешенный или обычный метод ближайших соседей. Можно задать свою весовую функцию.

4.2. Метод опорных векторов

Метод опорных векторов (Support Vector Machine, SVM, машина опорных векторов) — линейный классификатор с l^2 регуляризатором и функцией потерь hinge loss. Метод опорных векторов решает следующую задачу [3]:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$

при ограничениях вида $y_i \left(w^T \phi(x_i) + b\right) \geq 1 - \xi_i$, $i = 1, \ldots, N$. Разделяющая функция будет иметь вид

$$f(x) = w \cdot \phi(x) + b, \quad w = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i),$$

Таким образом, необходимо вычислить скалярные произведения. Рассмотрим ядро — функцию $K(x, y) = \phi(x) \cdot \phi(y)$. В случае линейного ядра $K(x, y) = x \cdot y$ метод ищет линейную разделяющую поверхность, максимизирующую ширину разделяющей полосы между классами. Другие популярные ядра: полиномиальное — $K(x, y) = (x \cdot y + 1)^d$ и гауссово (RBF) — $K(x, y) = \exp(-\gamma|x - y|^2), \gamma > 0$. Особым образом выбирая $\phi(x)$, можно изменять вид разделяющей поверхности, делая отображение в пространство большей размерности. На графике 5 можно увидеть разделяющие поверхности для разных популярных ядер.

В качестве реализации используем класс `SVC` из пакета scikit–learn.

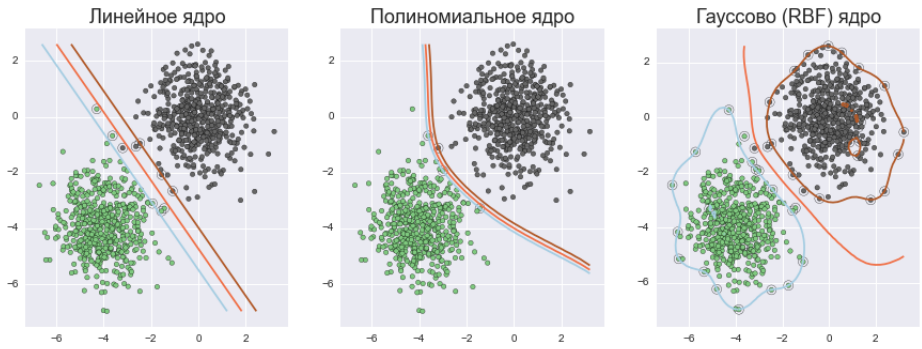

Рис. 5: Виды разделяющих поверхностей для разных ядер SVM

Важные гиперпараметры модели:

Название	Описание
`C`	Коэффициент C, влияет на регуляризацию и ширину разделяющей полосы.
`kernel`	Ядровая функция. Влияет на форму разделяющей поверхности. Часто используемые ядра: линейное, RBF, полиномиальное.
`gamma`	Параметр γ RBF ядра
`degree`	Степень полиномиального ядра

4.3. Случайный лес

Случайный лес (random forest) — метод машинного обучения, основанный на использовании ансамбля деревьев принятия решений [5].

Дерево принятия решений относится к так называемым логическим методам машинного обучения. Оно представляет собой дерево, в узлах которого лежат логические функции, принимающие значения некоторых из признаков. В листьях дерева лежат значения целевой функции — в случае бинарной классификации это 0 и 1.

Random Forest строит несколько решающих деревьев на *случайном подпространстве* обучающего множества. Для предсказания вычисляется среднее из предсказаний всех обученных решающих деревьев. Из интересных особенностей метода можно отметить:

- случайный лес нечувствителен к монотонным преобразованиям признаков;
- с помощью него можно оценивать значимость признаков;
- метод не очень сильно зависит от подбора гиперпараметров;

Часто метод используют в качестве базового решения — обычно случайный лес показывает хороший результат, не требуя при этом тщательного подбора гиперпараметров и предобработки данных.

В качестве реализации используем класс `RandomForestClassifier` из пакета scikit–learn. Важные гиперпараметры модели:

Название	Описание
`n_estimators`	Количество деревьев принятия решений.
`max_depth`	Максимальная глубина деревьев принятия решений

4.4. Градиентный бустинг деревьев решений

Градиентный бустинг, как и случайный лес, работает с композицией деревьев [5]. В этом случае ансамбль строится последовательно так, что каждая следующая модель улучшает качество уже построенного ансамбля. Новые деревья добавляются в ансамбль путём жадной минимизации эмпирического риска, заданного дифференцируемой функцией потерь.

В последнее время градиентный бустинг решающих деревьев стал одним из самых популярных методов обучения с учителем. В частности, метод популярен в области обучения ранжированию и используется для ранжирования результатов веб-поиска.

В качестве реализации используем пакет XGBoost. Важные гиперпараметры модели:

Название	Описание
`n_estimators`	Количество деревьев принятия решений.
`max_depth`	Максимальная глубина деревьев принятия решений

Глава 5. Эксперименты

5.1. Базовые признаки

Попробуем перечисленные выше модели на выборке, содержащей следующие признаки:

- количество общих друзей двух пользователей;
- разница в возрасте;
- факт совпадения или различия полов;
- факт совпадения или различия страны;
- факт совпадения или различия города;
- факт совпадения или различия локации, из которой пользователь заходит чаще всего;

Обучающее множество построим длиной 300000, тестовое — 10000.

Метод	Гиперпараметры	AUC–ROC
Random Forest	`n_estimators = 34`, `max_depth = 4`	0.8704
XGBoost	`n_estimators = 45`, `max_depth = 4`	0.9145
SVM	`kernel = linear`, `C` $\approx$ `1e6`	0.5486
kNN	`metric = manhattan`, `n_neighbors = 175`, `weights = distance`	0.5564

Значение `manhattan` у параметра `metric` у метода ближайших соседей значит метрику Минковского с $p = 1$. `weights = distance` значит, что использовался метод взвешенных ближайших соседей.

Лучший результат показали методы, основанные на решающих деревьях.

5.2. Логарифмирование

Рассмотрим распределение признака ”количество общих друзей”:

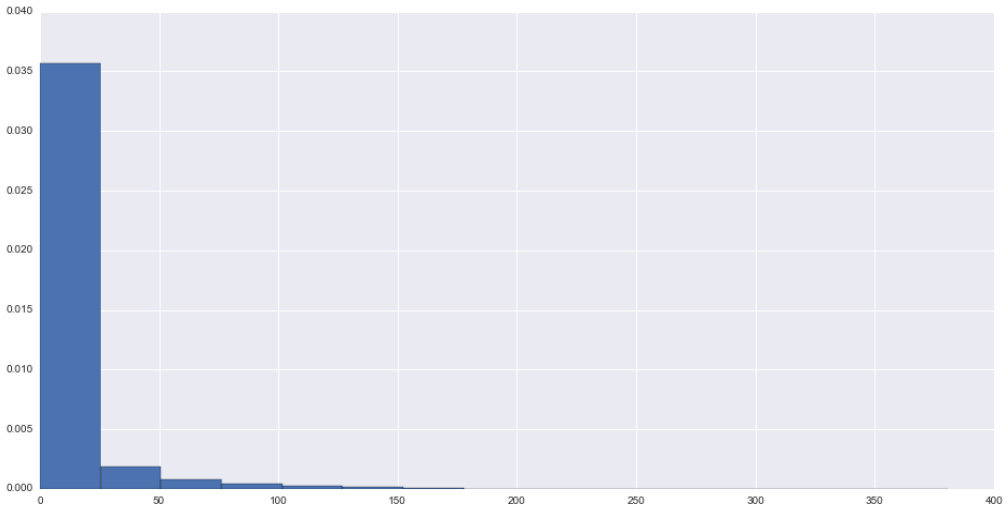

Рис. 6: Гистограмма признака ”количество общих друзей”

Распределение негладкое. В таких случаях помогает сглаживание функцией $\log(x+1)$. Результат применения этой функции:

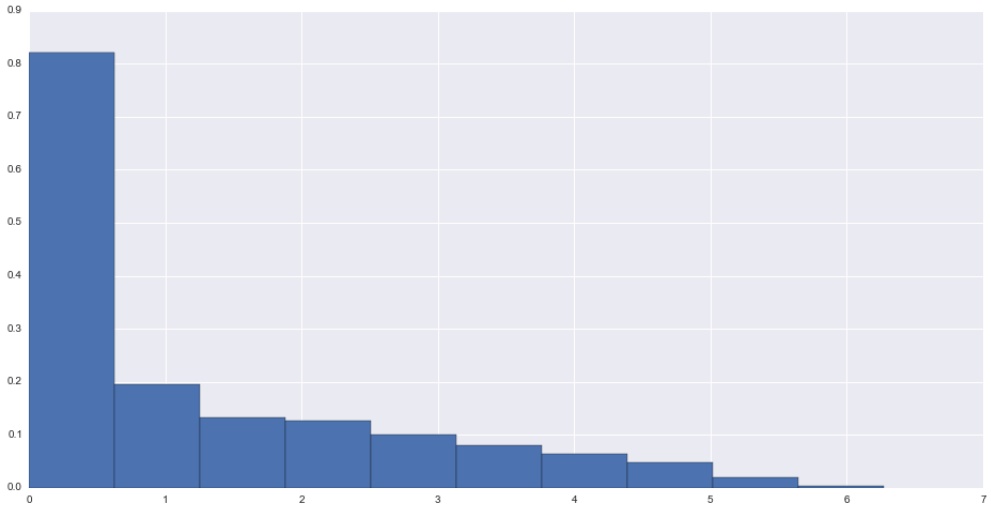

Рис. 7: Логарифм признака ”количество общих друзей”

Результат обучения на измененной выборке:

Метод	Гиперпараметры	AUC–ROC
Random Forest	`n_estimators = 28`, `max_depth = 4`	0.8732
XGBoost	`n_estimators = 30`, `max_depth = 4`	0.9139
SVM	`kernel = linear`, `C` $\approx$ `1e7`	0.909
kNN	`metric = euclidean`, `n_neighbors = 86`, `weights = uniform`	0.9112

Этот приём сильно улучшил результат методов SVM и kNN.

5.3. Масштабирование

Бывает полезно отмасштабировать матрицу признаков X, например, разделив её элементы на разность максимального и минимального элементов. Обычно это улучшает результат метрических алгоритмов.

Результат обучения на отлогарифмированной и затем отмасштабированной выборке:

kNN	`metric = euclidean`, `n_neighbors = 193`, `weights = uniform`	0.9104

Видно, что результат не улучшился. Так как значения всех признаков лежат в $[0, 1]$, метрический алгоритм считает изменение любого признака одинаково важным. Однако это далеко не так, и мы можем воспользоваться алгоритмами, основанными на решающих деревьях, чтобы отмасштабировать выборку в соответствии с важностью признаков. Умножим значения каждого признака на оценку важности, полученную с помощью алгоритма XGBoost. Результат kNN (остальные методы оказались нечувствительны к масштабированию):

kNN	`metric = euclidean,`
	`n_neighbors = 160,`
	`weights = uniform`
		0.9117

Получили небольшой прирост качества.

5.4. Коэффициент Жаккара

Ранее мы использовали количество общих друзей. Рассмотрим две ситуации:

- два пользователя, имеющие по 10 друзей каждый, имеют 2 общих друга;
- два пользователя, имеющие по 1000 друзей каждый, имеют 2 общих друга;

С точки зрения построенных ранее классификаторов, вероятности связей между пользователями этих пар равны (при прочих равных). Скорее всего, это не так, и вероятность связи во второй паре ниже.

Существует метод для измерения схожести множеств — коэффициент Жаккара:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, \quad 0 \leq J(A,B) \leq 1.$$

Коэффициент Жаккара учитывает размеры множеств и его значения лежат в $[0,1]$. Этот метод часто используется в области предсказания ссылок [6].

Для двух пар из примера выше значения коэффициента Жаккара будут 0.1 и 0.001 соответственно.

Вычислить этот признак просто, имея количества общих друзей пар пользователей:

$$J(u,v) = \frac{C_{uv}}{\sum_{i=1}^{N}[F_{ui} + F_{iv}]},$$

где C — матрица количества общих друзей, и F — матрица связей:

$$F_{ij} = \begin{cases} 1, & \text{пользователи } i \text{ и } j \text{ являются друзьями} \\ 0, & \text{пользователи } i \text{ и } j \text{ не являются друзьями} \end{cases}$$

Заменим признак "количество общих друзей" на коэффициент Жаккара, применим функцию $\log(x+1)$, отмасштабируем и обучим методы:

Метод	Гиперпараметры	AUC–ROC
Random Forest	`n_estimators = 16`, `max_depth = 4`	0.8971
XGBoost	`n_estimators = 25`, `max_depth = 4`	0.9113
SVM	`kernel = linear`, `C` $\approx$ `1e7`	**0.9122**
kNN	`metric = euclidean`, `n_neighbors = 86`, `weights = distance`	0.9058

Этот приём улучшил результат SVM. Результаты остальных методов не улучшились.

5.5. Коэффициент Адамик-Адара

Помимо количества общих друзей и коэффициента Жаккара, в области предсказания ссылок так же популярен коэффициент Адамик-Адара [1]:

$$A(u,v) = \sum_{w \in common(u,v)} \frac{1}{\log|w|},$$

где $common(u,v)$ обозначает множество общих друзей пользователей u и v.

Для вычисления коэффициента Адамик-Адара предпосчитаем вектор количества друзей пользователей f:

$$f_j = \sum_{i=1}^{N} F_{ij}$$

Введём матрицу общих друзей пользователя u:

$$M_{ij}^{(u)} = \begin{cases} 1, & i \in common(u, j) \\ 0, & i \notin common(u, j) \end{cases}$$

Вычислить её можно таким образом:

$$M^{(u)} = F \cdot \mathrm{diag}(F_u),$$

т.е. умножив матрицу связей на диагональную матрицу, диагональ которой состоит из элементов u-ой строчки матрицы связей. Далее рассмотрим следующую матрицу:

$$\hat{M}_{ij}^{(u)} = \begin{cases} f_w, & i \in common(u, j) \\ 0, & i \notin common(u, j) \end{cases}$$

Вычислим её следующим образом:

$$\hat{M}^{(u)} = M^{(u)} \cdot \mathrm{diag}(f) = F \cdot \mathrm{diag}(F_u) \cdot \mathrm{diag}(f)$$

Осталось применить к ненулевым элементам матрицы $\hat{M}$ операцию $\frac{1}{\log(x)}$ и просуммировать все строки. Получим вектор коэффициентов Адамик-Адара для пользователя u.

Заменим признак "количество общих друзей" на коэффициент Адамик-Адара, применим функцию $\log(x+1)$, отмасштабируем и обучим методы:

Метод	Гиперпараметры	AUC–ROC
Random Forest	`n_estimators = 31`, `max_depth = 4`	0.9002
XGBoost	`n_estimators = 27`, `max_depth = 4`	**0.9174**
SVM	`kernel = linear`, `C` $\approx$ `1e7`	**0.9163**
kNN	`metric = euclidean`, `n_neighbors = 97`, `weights = distance`	0.9104

Немного улучшились результаты SVM и XGBoost.

5.6. Персонализированный PageRank

Попробуем учесть структуру графа.

PageRank — метод, использованный для ранжирования поисковой выдачи в Google [4]. Он оценивает важность вершины на основе ссылок в графе.

Метод моделирует пользователя, который случайный образом переходит по исходящим ссылкам в графе. На каждом шаге с вероятностью α пользователь может не пройти по ссылке, а перепрыгнуть на любую случайную вершину. То же самое происходит, если у вершины нет исходящих ссылок. Метод оценивает частоту попадания в каждую вершину.

Обозначим значение PageRank вершины v как $PR(v)$. $PR(v)$ определяется рекурсивно и зависит от количества вершин, имеющих ссылки на v, а так же от значений PageRank этих вершин. Если на вершину v ссылается много вершин с большим PageRank, то и $PR(v)$ будет высоким.

Персонализированный PageRank [6] отличается от оригинального таким образом, что вместо прыжка на случайную вершину, пользователь всегда попадает в заранее заданную вершину. Таким образом,

вычисляется оценка «важности» остальных вершин по отношению к вершине v.

Существуют несколько методов, позволяющие приблизительно вычислить PageRank. Один из популярных методов является Power Method [8]. Воспользуемся им и напишем код, работающий с разреженными матрицами SciPy. Персонализированный PageRank для разреженной матрицы размером $[500000 \times 500000]$ на моём компьютере вычисляется около двух секунд.

Заменим признак "количество общих друзей" на значение персонализированного PageRank, применим функцию $\log(x+1)$, отмасштабируем и обучим методы:

Метод	Гиперпараметры	AUC–ROC
Random Forest	`n_estimators = 32`, `max_depth = 4`	0.6204
XGBoost	`n_estimators = 35`, `max_depth = 4`	0.6842
SVM	`kernel = linear`, `C` $\approx$ `1e7`	0.5674
kNN	`metric = euclidean`, `n_neighbors = 397`, `weights = distance`	0.5753

Можно заметить, что этот приём сработал не так хорошо, как остальные.

5.7. Новые признаки

Добавим признаки "количество друзей пользователя", "отношение количества «близких» друзей к количеству всех друзей" и "отношение количества «обычных» друзей к количеству «близких» друзей". Попробуем этот приём с разными комбинациями признаков "количество общих друзей", "коэффициент Жаккара", "коэффициент Адамик-Адара",

”персонализированный PageRank”. Выберем лучшие комбинации признаков для каждого метода. Результат:

Метод	Гиперпараметры	AUC–ROC
Random Forest (`jaccard`)	`n_estimators = 38`, `max_depth = 5`	0.9221
XGBoost (`common-friends + adamic-adar`)	`n_estimators = 17`, `max_depth = 4`	**0.9421**
SVM (`jaccard + adamic-adar`)	`kernel = linear`, `C` $\approx$ `1e7`	**0.9402**
kNN (`common-friends + adamic-adar`)	`metric = euclidean`, `n_neighbors = 216`, `weights = distance`	0.9282

Этот подход дал очень хороший результат. С помощью случайного леса оценим важность признаков в обучающем множестве, содержащем признаки ”количество общих друзей”, ”коэффициент Жаккара” и ”коэффициент Адамик-Адара”:

Признак	Важность
Коэффициент Жаккара	37%
Коэффициент Адамик-Адара	33%
Количество общих друзей	17%
Количество друзей пользователя	8.6%
Отношение количества «обычных» друзей к количеству «близких» друзей	2.8%
Отношение количества «близких» друзей к количеству всех друзей	$< 1\%$
Модуль разницы в возрасте	$< 1\%$
Совпадение/различие города	$< 1\%$
Совпадение/различие локации, из которой пользователи заходят чаще всего	$< 1\%$
Совпадение/различие пола	$< 1\%$
Совпадение/различие страны	$< 1\%$

Новые признаки заняли места, следующие сразу за признаками, основанными на схожести вершин. XGBoost и SVM показали очень хорошие результаты — более 0.94.

5.8. Оптимизация гиперпараметров

Улучшим результаты XGBoost и SVM более тщательным подбором гиперпараметров — полным перебором по сетке.

Для XGBoost рассмотрим следующую сетку параметров:

Гиперпараметр	Значения
`n_estimators`	`5, 6, ..., 29, 30, 40, ..., 140, 150, 200`
`max_depth`	`2, 3, ..., 10`

Для SVM рассмотрим следующую сетку параметров:

Гиперпараметр	Значения
`kernel`	RBF, линейное, полиномиальное
`C`	`1e-9, 1e-8, ..., 1e8, 1e9`
`degree`	`1e-4, 1e-3, ..., 1e3, 1e4`
`gamma`	`2, 3, ..., 10`

Результаты:

Метод	Гиперпараметры	AUC–ROC
XGBoost	`n_estimators = 13`, `max_depth = 4`	0.9442
SVM	`kernel = linear`, `C` $\approx$ `1e7`	0.9425

Итак, лучший результат показал градиентный бустинг деревьев принятия решений — AUC–ROC = 0.9442.

Выводы

К рассмотренным данным были применены основные методы предсказания ссылок, использующие меры схожести вершин (количество общих друзей, коэффициент Жаккара, коэффициент Адамик-Адара), а так же метод, основанный на структуре социального графа — персонализированный PageRank. Так же были применены техники, позволяющие использовать информацию о виде связи.

В процессе работы был написан программный код, работающий с разреженными матрицами и позволяющий быстро вычислять все необходимые показатели на матрицах большого размера. В экспериментах рассматривалась матрица размера $[500000 \times 500000]$, и все нужные коэффициенты (Жаккара, Адамик-Адара, персонализированный PageRank) для одного пользователя вычислялись в пределах 5 секунд. Этот код можно будет использовать в дальнейшей работе.

На основе проведённых экспериментов можно сделать вывод, что информация о видах связи между пользователями играет довольно большую роль и может быть успешно скомбинирована с существующими метода предсказания ссылок в социальных сетях.

Заключение

В работе были рассмотрены различные известные методы предсказания ссылок в графе наряду с несколькими популярными алгоритмами машинного обучения. Так же была определённым образом задействована информация о виде связей, что показало хороший результат на рассмотренных данных.

В дальнейшем планируется использовать другие методы, использующие информацию о структуре графа, а так же различные вероятностные модели. Кроме того, планируется рассмотреть некоторые другие наборы данных.

Список литературы

[1] Adamic Lada, Adar Eytan. Friends and Neighbors on the Web // Social Networks. — 2001. — Vol. 25. — P. 211–230.

[2] Bergstra James, Bengio Yoshua. Random Search for Hyper-parameter Optimization // J. Mach. Learn. Res. — 2012. — . — Vol. 13. — P. 281–305.

[3] Cortes Corinna, Vapnik Vladimir. Support-vector networks // Machine Learning. — 1995. — Vol. 20, no. 3. — P. 273–297. — URL: http://dx.doi.org/10.1007/BF00994018.

[4] L. Page, S. Brin, R. Motwani, T. Winograd // Proceedings of the 7th International World Wide Web Conference. — Brisbane, Australia. — P. 161–172.

[5] Hastie Trevor, Tibshirani Robert, Friedman Jerome. — New York, NY, USA.

[6] Liben-Nowell David, Kleinberg Jon. The Link-prediction Problem for Social Networks // J. Am. Soc. Inf. Sci. Technol. — 2007. — . — Vol. 58, no. 7. — P. 1019–1031.

[7] Metrics. — URL: https://www.kaggle.com/wiki/Metrics.

[8] PageRank Computation and the Structure of the Web: Experiments and Algorithms / Arvind Arasu, Jasmine Novak, John Tomlin, Andrew Tomkins. — 2002.

[9] SNAHackathon. — URL: http://www.snahackathon.org/.

[10] Sparse Matrices // SciPy.org. — URL: http://docs.scipy.org/doc/scipy/reference/sparse.html.

[11] Wikipedia. Accuracy paradox // Википедия, свободная энциклопедия. — URL: https://en.wikipedia.org/wiki/Accuracy_paradox.

[12] Wikipedia. PageRank // Википедия, свободная энциклопедия. — URL: https://en.wikipedia.org/wiki/PageRank.

[13] Машинное обучение (курс лекций, К.В.Воронцов). — URL: http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_(курс_лекций,%2C_К.В.Воронцов).