Санкт-Петербургский государственный университет

Прикладная математика и информатика
Исследование операций и принятие решений в задачах оптимизации, управления и экономики

Мирошниченко Никита

Самообучающиеся системы и их приложения

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Романовский И. В.

Рецензент:
доцент Машарский С. М.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Computer Science
Operation Research and Decision Making in Optimisation, Control and Economics Problems

Nikita Miroshnichenko

Self-learning systems and their applications

Graduation Thesis

Scientific supervisor:
Dr. Sci., Professor Joseph V. Romanovsky

Reviewer:
Associate Professor Sergey M. Masharsky

Saint-Petersburg
2016

Оглавление

Введение

Мы живем в век информации: нас окружает огромное количество данных, которые создаются ежесекундно, просто кликом по ссылке. Таким образом, возникает важная задача анализа данных. Машинное обучение занимается этой дисциплиной.

Машинное обучение — обширный подраздел искусственного интеллекта, математическая дисциплина, использующая разделы математической статистики, численных методов оптимизации, теории вероятностей, дискретного анализа, и извлекающая знания из данных. Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость, то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. [2]

Эта область заинтересовала меня уже давно, так как имеет большое приложение в реальной жизни и позволяет делать многие вещи проще. Поэтому я решил изучить эту тему более детально. К действию меня побудил конкурс, который проводила компания «Билайн». Были предложены обезличенные данные по пятидесяти тысячам пользователей, состоящие из идентификатора и шестидесяти переменных, которые необходимо было связать с возрастной группой пользователей и проверить эту зависимость на тестовой выборке. На мой взгляд, довольно интересная задача, для решения которой нужно изучить основные направления машинного обучения и существующие инструменты. В данной работе будет предпринята попытка решить эту задачу и сравнить результаты с конкурсными.

1. Основные понятия

Задано множество *объектов* X, множество *допустимых ответов* Y, и существует *целевая функция* $y^* : X \to Y$, значения которой $y_i = y^*(x_i)$ известны только на конечном подмножестве объектов $\{x_1 \dots x_l\} \subset X$. Пары «объект–ответ» (x_i, y_i) называются *прецедентами*. Совокупность пар $X^l = (x_i, y_i)_{i=1}^l$ называется *обучающей выборкой*.

Задача *обучения по прецедентам* заключается в том, чтобы по выборке X^l *восстановить зависимость* y^*, то есть построить *решающую функцию* $a : X \to Y$, которая приближала бы целевую функцию $y^*(x)$, причём не только на объектах обучающей выборки, но и на всём множестве X.

Решающая функция a должна допускать эффективную компьютерную реализацию, по этой причине будем называть её *алгоритмом*.

Также необходимо уточнить тип решаемой задачи. В зависимости от природы множества допустимых ответов Y задачи обучения по прецедентам делятся на следующие типы.

Если $Y = \{1, \dots, M\}$, то это задача *классификации* на M непересекающихся классов. В этом случае всё множество объектов X разбивается на классы $K_y = x \in X : y^*(x) = y$, и алгоритм $a(x)$ должен давать ответ на вопрос "какому классу принадлежит x?".

Если $Y = R$, то это задача *восстановления регрессии*, то есть в качестве ответа нужно предъявить функцию.

Метод обучения — это отображение $\mu : (X \times Y)^l \to A$, которое произвольной конечной выборке $X^l = (x_i, y_i)_{i=1}^l$ ставит в соответствие некоторый алгоритм $a \in A$. Говорят, что метод μ *строит* алгоритм a по выборке X^l. [3]

Встает проблема оценки построенного алгоритма. Для этого введем следующее понятие.

Функция потерь — это неотрицательная функция $L(a, x)$, характеризующая величину ошибки алгоритма a на объекте x. Если $L(a, x) = 0$, то ответ $a(x)$ называется *корректным*.

Функционал качества алгоритма a на выборке X_l:

$$Q\left(a, X^l\right) = \frac{1}{l}\sum_{i=1}^{l} L\left(a, x_i\right).$$

Основной задачей является минимизация функционала качества алгоритма для выборки, то есть в заданной модели необходимо найти такой алгоритм , что фукционал Q принимает на нем свое минимальное значение для заданной обучающей выборки X^l:

$$\mu\left(X^l\right) = \operatorname*{argmin}_{a\in A} Q\left(a, X^l\right).$$

Однако необходимо понимать, что существует риск *переобучения* на тренировочной выборке — качество работы алгоритма на объектах, не вошедших в нее, может оказаться существенно хуже. Построенный алгоритм может быть слишком сильно привязан к обучающей выборке, поэтому для успешной работы метод должен уметь обобщать данные.

Теперь, когда мы ввели необходимые определения, рассмотрим некоторые методы машинного обучения.

2. Метрические методы классификации

При рассмотрении таких сложных объектов, как фотографии, временные ряды, подписи, мы обнаруживаем, что сравнивать их между собой выгоднее и проще, чем изобретать признаки и сравнивать признаковые описания. Действительно, если мера сходства подобрана удачно, то зачастую оказывается, что схожим объектам очень часто соответствуют схожие объекты. Если формализовать это высказывание: классы образуют компактно локализованные подмножества. Это предположение принято называть *гипотезой компактности.* Для того, чтобы формализовать понятие «сходства» вводится функция расстояния в пространстве объектов X. Методы обучения, основанные на анализе сходства объектов, будем называть *метрическими*

Пусть на множестве объектов X задана функция расстояния $\rho : X \times X \to [0, \infty)$. Существует целевая зависимость $y^* : X \to Y$, значения которой известны только на объектах обучающей выборки $X^l = (x_i, y_i)_{i=1}^l, y_i = y^*(x_i)$. Множество классов Y конечно. Требуется построить алгоритм классификации $a : X \to Y$, аппроксимирующий целевую зависимость $y^*(x)$ на всём множестве X.

Если мы выберем любой объект из X и расположим элементы обучающей выборки в порядке возрастания до этого элемента, то получим перенумерованную выборку: $y_u^{(1)}$ - i-тый сосед объекта u.

Метрический алгоритм классификации с обучающей выборкой X^l относит объект u к тому классу $y \in Y$, для которого суммарный вес ближайших обучающих объектов $\Gamma_y\left(u, X^l\right)$ максимален:

$$a\left(u; X^l\right) = \operatorname*{argmax}_{y \in Y} \Gamma_y\left(u, X^l\right); \;\; \Gamma_y\left(u, X^l\right) = \sum_{i=1}^{l} [y_u^{(i)} = y] w(i, u),$$

где весовая функция $w(i, u)$ оценивает степень важности i-го соседа для классификации объекта u. Функция $\Gamma_y\left(u, X^l\right)$ называется *оценкой близости* объекта u к классу y.

Метрический классификатор определён с точностью до весовой функции $w(i, u)$. Для того, чтобы соответствовать гипотезе компатности,

необходимо выбирать неотрицательную и возрастающую по i функцию.

Параметрами такого алгоритма является сама обучающая выборка и весовая функция. Но если настройка обучающей выборки больше относится к вопросу обработки поступающей информации, то весовая функция позволяет менять сам алгоритм. Выбирая весовую функцию, можно получать различные метрические классификаторы.

2.1. Метод ближайшего соседа.

Алгоритм ближайшего соседа крайне прост: он находит ближайший к классифицируемому объект из обучающей выборки, смотрит на его класс и относит классифицируемый объект к этому классу. Если формализовать это высказывание:

$$w\left(i,u\right)=[i=1];\ a\left(u,X^{l}\right)=y_{u}^{(1)},$$

где $u \in X$ — классифицируемый объект, а $y_u^{(1)}$ - класс ближайшего к нему объекта.

Достоинства этого алгоритма простота и объяснимость выводов, а недостатками являются неустойчивость к погрешностям, низкое качеством классификации, отсутствие гибкости, то есть при построении алгоритма не учитываются достаточно большое количество информации. Некоторым решением может являться алгоритм k ближайших соседей.

2.2. Алгоритм k ближайших соседей.

Ранее учитывался только один сосед — ближайший, рассмотрим алгоритм, который использует k ближайших соседей.

$$w\left(i,u\right)=[i\leq k];\ a\left(u;X^{l},k\right)=\underset{y\in Y}{\operatorname{argmax}}\sum_{i=1}^{k}[y_{u}^{(i)}=y].$$

Предыдущий алгоритм является частным случаем этого при $k = 1$, и так же неустойчив к погрешностям. Однако, не стоит стремиться к максимально возможному увеличению k, так как в данном случае ал-

горитм будет вырождаться в константу. Появляется проблема выбора константы k, для решения которой предлагается использовать критерий скользящего контроля с *исключением объекта по одному*: для каждого объекта проверяется правильно ли он классифицируется по своим k ближайшим соседям (разумеется нужно исключить сам объект).

$$LOO\left(k, X^l\right) = \sum_{i=1}^{l} [a\left(x_i; X^l \backslash \{x_i\}, k\right) \neq y_i] \to \min_k .$$

Необходимо отметить, что существует еще один вариант этого метода, когда в каждом классе выбирается k ближайших к u объектов, и объект u относится к тому классу, для которого среднее расстояние до k ближайших соседей минимально.

Также существует проблема достижения максимума на нескольких классах. Решением данной проблемы, является введение строго убывающей последовательности вещественных весов w_i, задающих вклад i-го соседа в классификацию. Стоит брать нелинейную последовательность, в противном случае неоднозначности все же могут возникнуть.

Рассмотренные выше алгоритмы имеют неустранимые недостатки: необходимость хранить всю обучающую выборку целиком, сравнение классифицируемого объект в лучшем случае с $O\left(lnl\right)$ соседями, а также небольшой набор параметров. Следующий алгоритм предлагает устранить некоторые из них.

2.3. Метод парзеновского окна.

Будем изменять функцию веса так, чтобы она зависела от расстояния до соседа, а не от его ранга. Для этого введем *функцию ядра* $K\left(z\right)$, невозрастающую на $[0, \infty)$. Положив $w\left(i, u\right) = K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{h}\right)$, получим алгоритм

$$a\left(u; X^l, h\right) = \underset{y \in Y}{\operatorname{argmax}} \sum_{i=1}^{l} [y_u^{(i)} = y] K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{h}\right).$$

Параметр h — *ширина окна*, роль которого соответствует роли количества соседей — k. "Окно" — сферическая окрестность объекта u радиуса h, при попадании в которое обучающий объект повышает шанс на попадание u в свой класс.Скользящий контроль помогает задавать параметр h так, чтобы окно не было слишком узким, что приводит к неустойчивой классификации, или слишком широким, так как в этом случае алгоритм вырождается в константу.

Фиксация окна не всегда оправдана, так как в задачах, может оказаться, большое количество соседей в одних окрестностях, а других — вообще ни одного. В данных задачах можно применять *переменную ширину окна*. Необходимо использовать функцию $K(z)$, положительную на отрезке $[0,1]$ и равную нулю вне его. Определить h как наибольшее число, при котором ровно k ближайших соседей объекта u получают ненулевые веса: $h(u) = \rho\left(u, x_u^{(k+1)}\right)$. Тогда алгоритм принимает вид:

$$a\left(u; X^l, k\right) = \underset{y \in Y}{\operatorname{argmax}} \sum_{i=1}^{k} [y_u^{(i)} = y] K\left(\frac{\rho\left(u, x_u^{(i)}\right)}{\rho\left(u, x_u^{(k+1)}\right)}\right).$$

2.4. Метод потенциальных функций

К методу "окон" можно использовать другой подход: рассматривать окрестности объектов обучающей выборки — при попадании u в окрестность x_i, шанс попадания в класс y_i повышается.

$$a\left(u; X^l\right) = \underset{y \in Y}{\operatorname{argmax}} \sum_{i=1}^{l} [y_i = y] \gamma_i K\left(\frac{\rho(u, x_i)}{h_i}\right), \quad \gamma_i > 0, h_i > 0.$$

Алгоритм имеет достаточно богатый набор из $2l$ параметров γ_i, h_i. Один из методов настройки представлен в следующем алгоритме:

Вход: X^l — обучающая выборка;

Выход: Коэффициенты $\gamma_i, i = 1, \ldots, l$;

1: инициализация: $\gamma_i = 0; i = 1, \ldots, l$;

2: повторять

3: выбрать объект $x_i \in X^l$;

4: если $a(x_i) \neq y_i$, то

5: $\gamma_i := \gamma_i + 1$;

6: пока число ошибок на выборке не окажется достаточно мало.

Данный алгоритм находит только веса γ_i, предполагая, что радиусы потенциалов h_i и ядро K выбраны заранее. Идея алгоритма заключается в следующем, если обучающий объект x_i классифицируется неверно, то потенциал класса y_i недостаточен в точке x_i, и вес γ_i увеличивается на единицу. Выбор объектов на шаге 3 лучше осуществлять в случайном порядке.

Представленный алгоритм достаточно эффективен, при поступлении обучающей выборки потоком. Однако он имеет некоторые недостатки: медленная сходимость, зависимость результата от порядка предъявления объектов, грубая настройка параметров с шагом в единицу. И как результат алгоритм обладает недостаточно высоким качеством классификации.

2.5. Отбор эталонных объектов.

В приведенных выше алгоритмах одним из важных параметров является сама обучающаяся выборка. Необходимо упомянуть о некоторых методах, которые позволяют улучшить ее качество, выделить типичных представителей класса. Если рассматриваемый объект близок к такому типичному представителю — *эталону*, то скорее всего он сам принадлежит к этому классу. Но как найти такие объекты? Для этого введем *функцию отступа*, которая позволит оценивать степень типичности объекта.

Отступом объекта $x_i \in X^l$ относительно алгоритма классификации, имеющего вид $a(u) = \operatorname{argmax}_{y \in Y} \Gamma_y(u)$, называется величина

$$M(x_i) = \Gamma_{y_i}(x_i) - \max_{y \in Y \setminus y_i} \Gamma_y(x_i).$$

Отступ — это показатель *степени типичности объекта*. Отрицательным он может быть только в случае, когда алгоритм допускает

ошибку на данном объекте. В зависимости от значения отступа обучающие объекты условно делятся на типы.

Эталонные объекты имеют большой положительный отступ, плотно окружены объектами своего класса и являются наиболее типичными его представителями.

Неинформативные объекты также имеют положительный отступ. Изъятие этих объектов из выборки (при условии, что эталонные объекты остаются) не влияет на качество классификации.

Пограничные объекты имеют отступ близкий к нулю. Классификация таких объектов неустойчива: малые изменения метрики или состава обучающей выборки могут изменять их классификацию.

Ошибочные объекты имеют отрицательные отступы и классифицируются неверно. Возможной причиной может быть неадекватность алгоритмической модели, в частности, неудачная конструкция метрики ρ.

Шумовые объекты или выбросы — объекты с большими отрицательными отступами. Они плотно окружены объектами чужих классов и классифицируются неверно.

Рассматривая распределение значений отступов, можно сделать некоторые выводы о выборке в общем. При большом количестве отрицательных отступов велика вероятность того, что гипотеза компактности не выполняется и применение метрических алгоритмов в данной задаче нецелесообразно по отношению к выбранной метрике. Большое количество пограничных объектов показывает, что надежность данной классификации недостаточна.

2.6. Применение метрических методов классификации

В качестве релизации метода k-ближайших соседей был выбран класс KNeighborsClassifier библиотеки sclearn для языка Python. Признаки были отранжированы по корреляции с классом объекта. Для данного метода была использованп переменная, обладавшая наибольшей корре-

ляцией с классом объекта. Количество рассматриваемых соседей было фиксировано — 200. Получившаяся точность пресказания равна 72.287%. На первом этапе при обучении были использованы все переменные, однако качество предсказания оставляло желать лучшего. При изменении учитываемого количества соседей точность оставалась в пределах 40%. На втором этапе с помощью метода $get\_fscore()$ класса XGBclassifier (библиотека xgboost) переменные были отранжированы по влиянию на класс объекта, и при последовательном исключении переменных с наименьшей важностью точность повышалась. На третьем этапе осталась только самая важная переменная, для которой было подобрано оптимальное значение количества соседей. Изменение метрики со стандартной давало только ухудшение результатов. Код для получения наилучшей точности приведен в Приложении 1.

3. Линейные классификаторы

Пусть X — пространство объектов; $Y = \{-1, 1\}$ — множество допустимых ответов; объекты описываются n числовыми признаками $f_j : X \to R,\ j = 1, \ldots, n$. Вектор $x = (x_1, \ldots, x_n) \in R^n$, где $x_j = f_j(x)$, называется признаковым описанием объекта x. Рассмотрим линейный классификатор:

$$a(x, w) = \operatorname{sign}\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right) = \operatorname{sign}\langle x, w\rangle.$$

Уравнение $\langle x, w\rangle = 0$ задаёт гиперплоскость, разделяющую классы в пространстве R^n. Если вектор x находится по одну сторону гиперплоскости, то объект x относится к классу $+1$, иначе — к классу -1. [1]

Параметр w_0 может быть опущен, в связи с тем, что среди признаков есть константа, $f_j(x) \equiv -1$, и тогда роль свободного коэффициента w_0 играет параметр w_j.

В случае произвольного числа классов *линейный классификатор* определяется выражением

$$a(x, w) = \arg\max_{y \in Y} \sum_{j=0}^{n} w_{y_j} f_j(x) = \arg\max_{y \in Y} \langle x, w_y\rangle,$$

где каждому классу соотвествует свой вектор весов $w_y = (w_{y_0}, w_{y_1}, \ldots, w_{y_n})$.

Обучение линейного классификатора заключается в том, чтобы по заданной обучающей выборке $X^m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ построить алгоритм $a : X \to Y$ указанного вида, минимизирующий фунционал эмпирического риска:

$$Q(w) = \sum_{i=1}^{m} \big[a(x_i, w) \neq y_i\big] \to \min_w.$$

Методы обучения линейных классификаторов различаются подходами к решению данной оптимизационной задачи.

Для удобства введем понятие отступа для произвольного обучаю-

щего объекта $x_i \in X^m$ в случае произвольного числа классов:

$$M(x_i) = \langle x_i, w_{y_i} \rangle - \max_{y \in Y,\, y \neq y_i} \langle x_i, w_y \rangle.$$

Отступ определяет "степень погруженности" объекта в своей класс, чем он меньше, тем больше вероятность, что на этом объекте допускается ошибка классификации. В том случае, когда отступ принимает отрицательное значение, алгоритм допускает ошибку на этом объекте. Если формализовать данное высказывание:

$$Q(w) = \sum_{i=1}^{m} \big[M(x_i) < 0 \big].$$

Минимизация функционала $Q(w)$ по вектору весов — задача нахождения максимальной совместной подсистемы в системе неравенств. Однако она NP-полная, и количество решений может быть очень большим, поиск всех точных решений зачастую не представляет практического интереса. Нас устроит решение достаточно близкое к точному. Самым частым упрощением задачи является замена пороговой функции потерь на ее аппроксимацию:

$$\big[M < 0 \big] \leq L(M),$$

где $L : R \to R_+$ — непрерывная или гладкая функция, как правило, невозрастающая.

Соотвественно, минимизируется не сам функционал эмпирического риска, а его верхняя оценка:

$$Q(w) \leq \tilde{Q}(w) = \sum_{i=1}^{m} L\big(M(x_i)\big).$$

Такие непрерывные аппроксимации позволяют применять известные численные методы оптимизации для настройки весов.

К оценке стоит добавить штрафное слагаемое, которое запретит слишком большое значение нормы вектора весов.

$$Q(w) \leq \tilde{Q}(w) = \sum_{i=1}^{m} L\big(M(x_i)\big) + \gamma ||w||^p \to \min_w.$$

Данное штрафное слагаемое-регуляризатор снижает риск переобучения и повышает устойчивость вектора весов по отношению к малым изменениям обучающей выборки. Этот параметр подбирается исходя из априорных соображений, либо по скользящему контролю.

3.1. Метод стохастического градиента

Существует оптимизационная задача

$$Q(w) = \sum_{i=1}^{l} L(a(x_i, w), y_i) \to \min_w,$$

где $L(a, y)$ — заданная функция потерь. Применим для минимизации $Q(w)$ метод градиентного спуска. Сначала выбираем некоторое начальное приближение для вектора весов w, затем запускаем итерационный процесс, на каждом шаге которого вектор w изменяется в направлении наиболее быстрого убывания функционала Q. Это направление противоположно вектору градиента $Q'(w) = \left(\frac{\delta Q(w)}{\delta w_j}\right)_{j=1}^{n}$:

$$w := w - \eta Q'(w),$$

где $\eta > 0$ – величина шага в направлении антиградиента, называемая также темпом обучения. Предполагая, что функция потерь L дифференцируема, распишем градиент:

$$w := w - \eta \sum_{i=1}^{l} L'(\langle w, x_i \rangle y_i) x_i y_i.$$

Каждый прецедент $(x_i, , y_i)$ вносит аддитивный вклад в изменение вектора w, но вектор w изменяется только после перебора всех l объектов. Сходимость итерационного процесса можно улучшить, если выбирать прецеденты (x_i, y_i) по одному, для каждого делать градиентный шаг и сразу обновлять вектор весов:

$$w := w - \eta \sum_{i=1}^{l} L'_a(\langle w, x_i \rangle y_i) x_i y_i.$$

В методе стохастического градиента прецеденты перебираются в случайном порядке, если же объекты предъявлять в некотором фиксированном порядке, процесс может зациклиться или разойтись

Алгоритм стохастического градиента.

Вход: X^l — обучающая выборка; η — темп обучения; λ — параметр сглаживания функционала Q;

Выход: Вектор весов w;

1: инициализировать веса w_j $j = 0, \ldots, n$;

2: инициализировать текущую оценку функционала:

$$Q := \sum_{i=1}^{l} L\left(a\left(x_i, w\right),\, y_i\right);$$

3: повторять пока значение Q не стабилизируется и/или веса w не перестанут изменяться

4: выбрать объект x_i из X^l (например, случайным образом);

5: вычислить выходное значение алгоритма $a\left(x_i, w\right)$ и ошибку:

$$\varepsilon_i := L\left(a\left(x_i, w\right),\, y_i\right);$$

6: сделать шаг градиентного спуска:

$$w := w \,-\, \eta L'_a\left(a\left(x_i, w\right),\, y_i\right)\varphi'\left(< w, x_i >\right)x_i;$$

7: оценить значение функционала:

$$Q := \left(1 \,-\, \lambda\right)Q \,+\, \lambda\varepsilon_i;$$

Одним из параметров данного метода является *порядок предъявления объектов*. Существует несколько определенных вариантов: выбирать объекты случайным образом. Выбирать случайно, но попеременно из разных классов, что позволяет сильнее изменять вектор весов. Еще один вариант: предъявлять объекты с вероятностью обратно пропорциональной величине ошибки на объекте, что, однако, приводит к чуствительности к шумам.

Инициализация вектора весов — также один из параметров. Наи-

более распространено — заполнение нулями, что не всегда является лучшим подходом.

$$w_j := rand\left(-\frac{1}{n}, \frac{1}{n}\right),$$

где n — размерность пространства признаков. Этот подход существенно более удачен, чем предыдущий, при условии нормализации признакового описания.

Недостатками данного алгоритма являются: *отсутствие сходимости* или *слишком медленная сходимость*. Возможно *"застревание"* на локальных минимумах, так как функционал Q многоэкстремален. Для устранения данного недостатка используются техника встряхивания коэффициентов: при стабилизации функционала производятся случайные модификации вектора w в большой окрестности текущего значения, далее процесс запускается для модифицированного вектора.

3.2. Применение линейных классификаторов.

Для реализации линейного классфикатора был испоьлзован класс SGDclassifier из библиотеки sklearn для языка python. В котором в качестве метода минимизации эмпирического риска используется метод стохастического градиента, а в качестве функции потерь $L\left(M\left(x_i\right)\right) = \log_2\left(1 + e^{-M(x_i)}\right)$. В выборке все отсутствующие значения были заменены на средние по столбцу. Получившаяся точность предсказания на тестовой выборке равняется: 31.51%. Код для применения линейного классификатора приведен в Приложении 2.

4. Линейные композиции. Бустинг

При решении задач классификации зачастую оказывается, что ни один из используемых алгоритмов не обеспечивает нужной нам точности предсказания. Одним из выходов может быть построение композиции этих алгоритмов для компенсации проблем каждого из них. В качестве примера композиции можно привести простое или взвешенное голосование.

4.1. Композиция алгоритмов

Композицией K алгоритмов $a_k(x) = C(b_k(x)), k = 1, \ldots, K$ называется суперпозиция алгоритмических операторов $b_k : X \to R$ (иногда называющихся базовыми алгоритмами), корректирующей операции $F : R^K \to R$ и решающего правила $C : R \to Y$:

$$a(x) = C(F(b_1(x), \ldots, b_K(x))), x \in X.$$

То есть алгоритм классификации имеет следующю структуру: сначала $b(x)$ вычисляет некую оценку попадания объекта в тот или иной класс, далее с помощью решающего правила алгоритм переводит их в конечный результат — номер класса. С помощью пространства оценок R расширяется множество допустимых корректирующих операций, так как при определении F как отображения $Y^t \to Y$ возникает проблема выбора "хорошего" F, и количество оптимальных F мало. При комбинировании ответов алгоритмических операторов операция использует оценки принадлежности объекта к классам, которые более точны. Например, можно использовать линейную комбинацию и настраивать для каждого базового алгоритма свой коэффициент. [4]

4.2. Градиентный бустинг для произвольной функции потерь

Рассмотрим обобщенный случай бустинга в задачах классификации, где корректирующая функция является линейной комбинацией базовых алгоритмов, что является взвешенным голосованием. Пусть $Y = 1, \ldots, M$, $R = R^M$, базовые алгоритмы возвращают вектора оценок принадлежности объекта к тому или иному классу. Введем алгоритмическую композицию как:

$$a(x) = C(F(b_1(x), \ldots, b_K(x))) = \underset{y \in Y}{\operatorname{argmax}} \sum_{k=1}^{K} a_k b_{k,y}(x), x \in X.$$

Определим функционал качества получившегося алгоритма как число ошибок, допускаемых на обучающей выборке:

$$Q_K = \sum_{j=1}^{l} \left(\underset{y \in Y}{\operatorname{argmax}} \sum_{k=1}^{K} a_k b_{k,y}(x_j) \neq y_j \right)$$

Очевидно, что наша задача заключается в минимазации данного функционала Q_K. Для упрощения введем эвристику: пороговая функция потерь функционала качества заменяется непрерывно дифференцируемой оценкой сверху $L(M)$. Данная оценка является одним из варьируемых параметров.

$$Q_K \leq Q_K^{'} = \sum_{i=1}^{l} L \left(\sum_{k=1}^{K} a_k b_k(x_i), y_i \right)$$

В целях упрощения задачи минимизации введем следующую эвристику: при добавлении k-того слагаемого, оптимизируются только k-тый базовый алгоритм и коэффициент при нем, все ранее введенные слагаемые остаются фиксированными. С помощью такой тактики оптимизируется набор базовых алгоритмов под наши нужды, то есть при обучении следующего алгоритма повышается вес объектов, на которых совершалась ошибка классификации. Таким образом, учитывают-

ся недостатки прежних базовых алгоритмов. Получим:

$$Q\left(a,b;X^l\right)=\sum_{i=1}^{l}L\left(\sum_{k=1}^{K-1}a_kb_k\left(x_i\right)+ab\left(x_i\right),y_i\right)\to\min_{a,b}$$

Введем обозначения:

$f_{K-1}=\left(f_{K-1,i}\right)_{i=1}^{l}=\left(\sum_{k=1}^{K-1}a_kb_k\left(x_i\right)\right)_{i=1}^{l}$ — текущее приближение,

$f_K=\left(f_{K,i}\right)_{i=1}^{l}=\left(\sum_{k=1}^{K-1}a_kb_k\left(x_i\right)+ab\left(x_i\right)\right)_{i=1}^{l}$ — следующее приближение.

Для минимизации $Q\left(f\right)$ используем градиентный метод, первоначально не обращая внимания на тот факт, что f_K имеет непроизвольные координаты. Получив результат, далее будем аппроксимируем его с помощью a и b. Положим начальное приближение:

$$f_0:=0,$$

$$f_{K,i}:=f_{K-1,i}-ag_i,\ i=1,\ldots,l;$$

$g_i=L^{'}\left(f_{K-1,i},y_i\right)$ — компоненты вектора градиента, a — градиентный шаг

Зная вектор-градиент, аппроксимируем его базовым алгоритмом b_K так, чтобы $\left(b_k\left(x_i\right)\right)_{i=1}^{l}$ приближал вектор $\left(-g_i\right)_{i=1}^{l}$:

$$b_K:=\operatorname*{argmax}_b\sum_{i=1}^{l}\left(b\left(x_i\right)+g_i\right)^2.$$

Данный шаг отражает главную идею бустинга — последовательного построения композиции алгоритмов, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих. С помощью градиентного шага минимизируем функционал и в результате получаем новый базовый алгоритм. Выкладки можно записать в формальный алгоритм:

Вход: обучающая выборка X^l; параметр K;

Выход: базовые алгоритмы и их веса $a_kb_k,\ k=1,\ldots,K$;

1: инициализация: $f_i:=0,\ i=1,\ldots,l$;

2: для всех $k = 1, \ldots, K$:

3: найти базовый алгоритм, приближающий градиент;

$$b_K := \underset{b}{\operatorname{argmin}} \sum_{i=1}^{l} \left(b\left(x_i\right) + L^{'}\left(f_i, y_i\right) \right)^2;$$

4: решить задачу одномерной минимизации:

$$a_k := \underset{a>0}{\operatorname{argmin}} \sum_{i=1}^{l} L^{'}\left(f_i + a b_k\left(x_i\right), y_i\right)^2;$$

5: обновить значения композиции на объектах выборки:

$$f_i := f_i + a_k b_k\left(x_i\right);\ i = 1, \ldots, l.$$

4.3. Применение градиентного бустинга

Для реализации был использован класс XGBclassifier из библиотеки xgboost для языка python. В данном классе в качестве базовых алгоритмов используются решающие деревья, а в качестве функции потерь $L\left(M\left(x_i\right)\right) = \log_2\left(1 + e^{-M(x_i)}\right)$, количество базовых алгоритмов — решающих деревьев — было ограничено 1900, с глубиной не более 10.

В выборке все отсутствующие значения были заменены на средние по столбцу. Получившаяся точность предсказания на тестовой выборке равно 76.53%.

Основными инструментами для настройки алгоритма бустинга являются количество используемых базовых алгоритмов и шаг градиентного метода. Варьируя данные параметры, на первом этапе был найден локальный максимум точности по количеству базовых алгоритмов, на втором этапе был выбран шаг градиентного метода для данного количества. Код приведен в Приложении 3.

Заключение

В работе были рассмотрены и применены три вида классификаторов: метрический, линейный и бустинговый. Каждый из них обладает своими преимуществами и недостатками.

Метрические классификаторы просты в применении, однако не обладают достаточной гибкостью, неустойчивы к шумам и выбросам в исходных данных. Для устранения недостатков — необходимо дополнительно работать с выборкой и использовать лишние ресурсы.

Линейные классификаторы довольно гибки. Можно самостоятельно подобрать способы нахождения вектора весов, замену функции потерь. Однако существует проблема сходимости , и неустойчивости к шумам и выбросам. Одним из решений является применение различных эвристик для улучшения сходимости методов нахождения вектора весов.

Бустинг позволяет объединить слабые классификаторы в один сильный. Так как базовые классифкаторы выступают в роли черного ящика, что является несомненным преимуществом. Объединение базовых алгоритмов позволяет устранить недостатки каждого алгоритма.

Таким образом, заданная цель классификации объекта, обозначенная в работе, была решена. Самый лучший результат показала реализация алгоритма бустинга XGBclassifier из библиотеки xgboost для языка python. При сравнении результатов победителя конкурса компании "Билайн" с результатами, полученными в данной работе, точность предсказания улучшена на 0.14%.

Список литературы

[1] Wikipedia. Линейный классификатор. — 2008. — URL: http://goo.gl/cyVON9.

[2] Wikipedia. Машинное обучение. — 2016. — URL: https://goo.gl/vraQGr.

[3] Воронцов К. В. Лекции по алгоритмическим компощициям. — 2012. — URL: http://goo.gl/dtF1OI.

[4] Воронцов К. В. Математические методы обучения по прецедентам. — 2014. — URL: http://goo.gl/7eOltj.

5. Приложения

В приложениях содержится код для метрического, линейного классификаторов и градиентного бустинга.

5.1. Приложение 1: метрический классификатор

```
import xgboost as xgb
import numpy as np
import pandas as pd
from sklearn import ensemble

def toInt(x):
    if (type(x) ==str):
        return int(x, 16)
    else:
        return x
#чтение
train_df = pd.read_csv('train.csv', header=0)
test_df = pd.read_csv('test.csv', header=0)
ans_df = pd.read_csv('ans.csv', header=0)
train_X = pd.DataFrame();
test_X= pd.DataFrame();
#приведение к удобному виду
train_Y = train_df['y']
test_Y = ans_df['y']
predictors = [x for x in train_df.columns if x not in ['y']]
for x in range(0, len(predictors)):
    train_X[predictors[x]] = train_df[predictors[x]].apply(toInt)
    test_X[predictors[x]] = test_df[predictors[x]].apply(toInt)
#обработка отсутствующих значений
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
```

```
imp.fit(train_X)
train_X = pd.DataFrame(imp.transform(train_X))
test_X = pd.DataFrame(imp.transform(test_X))
#вычисление важности переменных
XGBclass = xgb.XGBClassifier(max_depth=10, n_estimators=800,
                                              learning_rate=0.03)
XGBclass.fit(train_X, train_Y)
bst = XGBclass.booster()
imps = bst.get_fscore()
feat_imp = pd.Series(imps).sort_values(ascending=False)
#применение knn на самой важной переменной
test_x = test_X[int(feat_imp.axes[0][0])].reshape(-1, 1)
train_x = train_X[int(feat_imp.axes[0][0])].reshape(-1, 1)
neigh = KNeighborsClassifier(n_neighbors=250)
neigh.fit(train_x, train_Y)
knn = neigh.predict(test_x)
print ('Accuracy=%f' % ((1 - (sum( int(knn[i]) != test_Y[i]
        for i in range(len(test_Y))) / float(len(test_Y)) ))*100))
```

5.2. Приложение 2: линейный классификатор

```
import pandas as pd
from sklearn.preprocessing import Imputer
from sklearn import linear_model
import xgboost as xgb
#чтение
train_df = pd.read_csv('train.csv', header=0)
test_df = pd.read_csv('test.csv', header=0)
ans_df = pd.read_csv('ans.csv', header=0)
train_X = pd.DataFrame();
test_X= pd.DataFrame();
#приведение к удобному виду
def toInt(x):
```

```
    if (type(x) ==str):
        return int(x, 16)
    else:
        return x
train_Y = train_df['y']
test_Y = ans_df['y']
predictors = [x for x in train_df.columns if x not in ['y']]
for x in range(0, len(predictors)):
    train_X[predictors[x]] = train_df[predictors[x]].apply(toInt)
    test_X[predictors[x]] = test_df[predictors[x]].apply(toInt)
#обработка отсутствующих значений
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit(train_X)
train_X = pd.DataFrame(imp.transform(train_X))
test_X = pd.DataFrame(imp.transform(test_X))
#применение линейного классификатора
SGD = linear_model.SGDClassifier()
SGD.fit(train_X, train_Y)
SGDtest = SGD.predict(test_x)
print ('Accuracy=%f' % (1 - (sum( int(SGDtest[i]) != test_Y[i]
        for i in range(len(test_Y))) / float(len(test_Y)) ))*100)
```

5.3. Приложение 3: градиентный бустинг

```
import xgboost as xgb
import numpy as np
import pandas as pd
from sklearn import ensemble

def toInt(x):
    if (type(x) ==str):
```

```
        return int(x, 16)
    else:
        return x
#чтение
train_df = pd.read_csv('train.csv', header=0)
test_df = pd.read_csv('test.csv', header=0)
ans_df = pd.read_csv('ans.csv', header=0)
train_X = pd.DataFrame();
test_X= pd.DataFrame();
#приведение к удобному виду
train_Y = train_df['y']
test_Y = ans_df['y']
predictors = [x for x in train_df.columns if x not in ['y']]
for x in range(0, len(predictors)):
    train_X[predictors[x]] = train_df[predictors[x]].apply(toInt)
    test_X[predictors[x]] = test_df[predictors[x]].apply(toInt)
#обработка отсутствующих значений
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit(train_X)
train_X = pd.DataFrame(imp.transform(train_X))
test_X = pd.DataFrame(imp.transform(test_X))
#применение градиентного бустинга
XGBclass = xgb.XGBClassifier(max_depth=10, n_estimators=1900,
                             subsample=0.5, learning_rate=0.01)
XGBclass.fit(train_X, train_Y)
XGBtest = XGBclass.predict(test_X)
print ('Accuracy=%f' % ((1 - (sum( int(XGBtest[i]) != test_Y[i]
        for i in range(len(test_Y))) / float(len(test_Y)) ))*100))
```