Правительство Российской Федерации

Федеральное государственное автономное учреждение
высшего профессионального образования

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет экономических наук
Департамент прикладной экономики

Курсовая работа

на тему:

«Предсказание и визуализация отмены заказов в компании Яндекс.Такси»

Выполнил:
Студент 3 курса
группы БЭК 131
Барабанов Иван
Дмитриевич

Научный руководитель:
Старший преподаватель департамента прикладной экономики
Демешев Борис
Борисович

Москва
2016

Содержание

Введение

Для любого сервиса, осуществляющего перевозки, вопрос оптимизации деятельности носит ключевой характер. Так, для сервиса Яндекс.Такси, являющегося одним из ключевых игроков на московском рынке такси, задача предсказания неосуществленных, или, попросту говоря, сгоревших заказов крайне актуальна, поскольку сервис действительно заинтересован в сокращении подобных заказов. Сам заказ может "сгореть"по нескольким причинам, начиная от отмены заказа самим потенциальным пассажиром либо из-за ошибки, допущенной им, заканчивая неспособностью отыскать водителя в ближайшей окрестности от места заказа.

Для разрешения этой проблемы ставится задача по созданию алгоритма, который позволит наиболее точно определить вероятность сгорания заказа. Для обучения машины будут использованы данные сервиса Яндекс.Такси, содержащие около 1800000 наблюдений с января по март 2014 года. Набор данных включает в себя точную дату, время заказа, координаты конечного пункта, время, за которое заказ был отменен, и длину предполагаемого маршрута по прямой от начальной до конечной точки. Помимо всего прочего, данные классифицированы по таким параметрам, как является ли заказ сгоревшим, найден ли водитель, а так же какого класса была выбрана машина.

Данная работа включает в себя изложение принципов работы алгоритмов, на основе которых будут выстроены предсказательные модели, описание процесса обработки имеющихся данных и извлечения переменных, способных потенциально улучшить качества моделей, а также применение и сравнение прогнозных сил использованных моделей.

Алгоритмы

Random forest

Случайный лес (random forest) есть один из наиболее эффективных алгоритмов машинного обучения, решающий задачи как классификации, так и построения регрессии. В его основе лежит переплетение принципов функционирования бэггинг-алгоритмов и решающих деревьев. Бэггинг представляет собой многократное обучение моделей на случайных выборках, сформированных посредством отбора определенного числа наблюдений с повторениями из обучающей выборки, и последующее усреднение результатов моделей с целью получения главной, итоговой. В роли моделей, которые многократно обучаются на различных выборках, выступают решающие деревья. Таким образом, каждая модель на отдельной выборке есть дерево, в совокупности - лес, случайный лес. Более формально алгоритм случайного леса **для каждого отдельного дерева** действует по следующей схеме:

- Из обучающей выборки размером n отобирается случайным образом с повторениями n наблюдений.
- Руководствуясь определенными критериями "выращивается"дерево до достижения условия остановки.

Разумеется, поскольку данные шаги едины для каждого посаженного дерева, то такая последовательность действий может повторяться и 500, и 1000 раз: все зависит от того, какое количество деревьев будет посажено в лесу. Необходимо лишь конкретизировать ряд критериев, на основе которого формируется решающее дерево.

1. Среди общего числа признаков (пусть их будет m), присутствующих в выборке , выбор разделении каждого из узлов дерева, осуществляется лишь на основе p случайно отобранных признаков, где $p < m$. Причем p признаков случайным образом выбираются абсолютно в каждом из узлов. Данная процедура осуществляются с целью избежания скоррелированности деревьев. В противном случае все деревья будут абсолютно схожи, поскольку в каждом узле у каждого дерева будет действовать идентичный критерий ответвления. В результате вместо потенциального леса будет по факту получено лишь одно решающее дерево. Кроме всего прочего, такой тип реализации разделения позволяет не доминирующим регрессорам, позволяющим спрогнозировать лишь особые, отличительные черты, выделится и стать основополагающими при разветвлении.

2. Среди отобранных случайным образом p признаков выбирается лишь один, обеспечивающий максимальное падение величины "неоднородности множества"после разветвления. В качестве данной величины может быт использовано несколько показателей. Например, энтропия, представляемая в виде:

$\varphi(p_i) = -\sum_{i=1}^{n} p_i * \ln_2(p_i)$, где p_i есть вероятность того, что наблюдение принадлежит i-ому из n классов. Таким образом, в каждом узле из p признаков выбирается такой, что получаемая взвешенная энтропия от двух ответвлений будет не просто меньше узловой, но и наименьшей среди всех остальных признаков. Из формулы видно, что если в неком узле содержатся наблюдения, принадлежащие лишь i-ому признаку, то $p_i = 1$, а за ним и $\varphi(p_i) = 0$, чего и добивается алгоритм. По идентичной логике происходит разветвление узлов на основе индекса Джини $g(p_i) = 1 - \sum_{i=1}^{n} p_i^2$: признак, обеспечивающий максимальное снижение индекса и объявляется разрешающим в конкретном узле. Если перечисленные выше критерии используются для классификации, то для решения задач предсказания количественных переменных, в качестве основного критерия выбирается минимизация суммы квадратов остатков после каждого ответвления.

В результате оценивается множество посредственных моделей - решающих деревьев. Однако их совокупное, усредненное значение обладает высокой прогнозной силой.

Support vector machine

Метод опорных векторов используется в задачах классификации, то есть для определения класса, к которому принадлежит объект. Задачу, в которой возможны лишь два класса, в свою очередь, называют бинарной. Каждое наблюдение представляет собой многомерный вектор $x_i \in \mathbb{R}^n$, содержащий определенные характеристики наблюдения.Помимо характеристик каждому наблюдению в обучающей выборке принадлежит один из элементов множества классов - Y, имеющего вид : $Y = \{1, -1\}$. Таким образом, весь имеющийся набор данных обучающей выборки можно описать, как:

$$D = \{(x_i, y_i) | x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}\}_{i=1}^{p}$$

Линейно разделимая выборка

Положим, что имеющаяся выборка является линейно сепарабельной, то есть представляется априори возможным разделить группы наблюдений, принадлежащих разным элементам множества Y посредством гиперплоскости без попадания каких-либо наблюдений в саму гиперплоскость. В результате, основой метода главных компонент является поиск оптимальной гиперплоскости, разделяющей объекты двух классов. Разумеется, между группами классов можно провести бесконечное количество гиперплоскостей, однако критерий оптимальности гарантирует нахождение лишь одной, необходимой плоскости. Согласно критерию, гиперплоскость должна быть такой, что расстояние до ближайших точек каждого из классов максимально. Это осуществляется с помощью

двух других вспомогательных, параллельных, опорных гиперплоскостей, которые фактически "очерчивают"границу, за каждой из которых объект соответствующе классифицируется. Математически представленные выше принципы можно записать следующим образом. Пусть имеется разделяющая гиперплоскость, заданная уравнением $\langle w, x\rangle - w_0 = 0$, где вектор $w = (w_1, \ldots, w_n)$ есть вектор нормали к гиперплоскости, а $\frac{w_0}{\|w\|}$ указывает на расстояние от начала координат до гиперплоскости. Ясно, что для опорных гиперплоскостей уравнение будет задаваться таким же образом за исключением того, что равенство нулю будет нарушено. Посредством нормировки, то есть домножением вектора w и w_0 на некоторую константу можно добиться того, что на каждой из опорных плоскостей выполнялось условие $\langle w, x\rangle - w_0 = y_i$. Таким образом:

$$\left[\begin{array}{l}\langle w, x\rangle - w_0 \leq -1, \text{если} y_i = -1 \\ \langle w, x\rangle - w_0 \geq 1, \text{если} y_i = 1\end{array}\right. \rightarrow y_i \cdot (\langle w, x\rangle - w_0) \geq 1$$

Представим, что x_+ наблюдение, находящееся прямо на одной из опорных гиперплоскостей, пусть той, которая отграничивает множество $y = 1$, и x_- наблюдение, по которому проходит вторая опорная гиперплоскость, ограничивающая множество $y = 1$. Тогда расстояние между ними есть $(x_+ - x_-)$, а ширина разделяющей полосы есть

$$\langle (x_+ - x_-), \frac{w}{\|w\|}\rangle = \frac{\langle x_+, w\rangle - \langle x_-, w\rangle}{\|w\|} = \frac{1 - w_0 - (-1 + w_0)}{\|w\|} = \frac{2}{\|w\|}$$

В результате SVM сводится к условной оптимизационной задаче:

$$\begin{cases}\dfrac{2}{\|w\|} \rightarrow \max \\ y_i \cdot (\langle w, x\rangle - w_0) \geq 1\end{cases} \Rightarrow \begin{cases}\|w\| \rightarrow \min \\ y_i \cdot (\langle w, x\rangle - w_0) \geq 1\end{cases}$$

Таким образом необходимо минимизировать норму вектора w при заданном ограничении.

Линейно неразделимая выборка

Теперь предположим, что выборка линейно не разделима, а значит невозможно провести такую гиперплоскость, по всей ширине которой не будут попадаться наблюдения.

1. Для решения данной задачи можно позволить алгоритму ошибаться, однако не без последствий для него. Введем определенную систему штрафов за каждое неверно классифицированное наблюдение. Для этого прибегнем к использованию так называемых "slack"переменных ξ_i. Неравенство $\frac{\xi}{\|w\|} \leq \frac{1}{\|w\|}$ означает, что наблюдение попало на "свою"сторону разделяющей гиперплоскости, а $\frac{\xi}{\|w\|} > \frac{1}{\|w\|}$ – на чужую. Таким образом, при $0 < \xi_i < 1$

наблюдение лежит внутри разделяющей гиперплоскости, а при $\xi_i > 1$ наблюдение неверно классифицировано. В результате оптимизационная задача переписывается следующим образом:

$$\begin{cases} \|w\| + C \cdot \sum\limits_{i=1} \xi_i \to \min \\ y_i \cdot (\langle w, x \rangle - w_0) \geq 1 - \xi_i \end{cases}$$

Параметр C, в свою очередь, отвечает за то, насколько серьезно воспринимаются ошибки: при достаточно малых C, близких к нулю, ошибки не оказывают практически никакого влияния на оптимизационную задачу, и наоборот – с увеличением C ошибки играют все более существенную роль, заставляя разделяющую гиперплоскость сужаться.

2. Возможно и другое решение - переход к новому пространству более высокой размерности с помощью некоторого отображения $\phi : X \to X'$. То есть если обучающая выборка не является линейно разделимой в нектором пространстве $\mathbb{R}^M$, то выбор отображения ϕ осуществляется таким образом, чтобы в новом пространстве $\mathbb{R}^N$, где $M < N$, выборка уже была линейно разделима. Разумеется, линейным решающее правило будет лишь в пространстве $\mathbb{R}^N$, при проекции обратно на $\mathbb{R}^M$ свойство линейности уже теряется. Однако если отображение ϕ переведет векторы из $\mathbb{R}^M$ в пространство $\mathbb{R}^N$, размерность которого будет крайне велико, работа алгоритма будет крайне затруднена из-за огромного количества вычислений. Исправить ситуацию помогает факт, получаемый из решения задачи условной оптимизации. Оказывается, решая задачу, алгоритм SVM использует лишь скалярное произведение векторов: $\langle x_i, x_j \rangle$, где $x_i, x_j \in \mathbb{R}^M$, а после воздействия отображения ϕ скалярное произведение превращается в $\langle \phi(x_i), \phi(x_j) \rangle$, где $\phi(x_i), \phi(x_j) \in \mathbb{R}^N$. Используя этот факт, можно найти некоторую функцию $K(x_i, x_j)$ такую, что $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$. Таким образом функция $K(x_i, x_j)$ позволит определить скалярное произведение векторов в пространстве $\mathbb{R}^N$, оставаясь при этом в пространстве $\mathbb{R}^M$, где $M < N$.Такие функции получили название ядер. Одним из наиболее широко используемых ядер является Гауссова радиальная базисная функция, имеющая вид: $K(x_i, y_j) = e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}}$.

Gradient Boosting

Принцип работы алгоритма градиентного бустинга удобно понимать, сравнивания с алгоритмом случайного леса. Как известно, random forest высаживает каждое дерево независимо от других, результаты которых затем усредняются. Однако в этом случае от каждого дерева требуется наращивание глубины, чтобы в конечном счете прийти к ответу. Вдобавок ко всему, помимо глубины деревьев приходится наращивать и их количество,что, разумеется, затягивает и утяжеляет процесс обучения машины. Boosting предлагает совершенно иной подход. Во-первых, базовые алгоритмы, коими будут выступать, например, те

же деревья, строятся не независимо друг от друга, а последовательно дополняя один одного, стремясь с каждым шагом улучшить "наработки"предыдущего алгоритма. Во-вторых, итоговый алгоритм формируется не как усредненное значение некоторого числа базовых алгоритмов, а как их линейная комбинация с определенными весами. В-третьих, бустингу вовсе не обязательно выстраивать глубоких деревьев, ровно, как и наращивать их число ввиду зависимостей базовых алгоритмов один от одного. Достаточно некоторого, небольшого числа неглубоких деревьев для разрешения задачи.
Это идея непосредственно boosting'a. Однако причем здесь градиент? Предположим в задачу алгоритма входит минимизация некоторой функции потерь $\mathbb{L}(y, \hat{y})$, где y есть истинное значение, а $\hat{y}$ - спрогнозированное. Далее, на основе выше описанной логике функционирования алгоритма бустинга, полагаем, что есть некоторая композиция N алгоритмов, дополняющих и корректирующих друг друга: $a_N(x) = \sum_{i=1}^{N} b_i(x)$. Далее, представим, что $a_{N-1}(x) = \sum_{i=1}^{N-1} b_i(x)$, то есть композиция алгоритмов на $N-1$ шаге уже построена, а значит, необходимо разработать такой алгоритм b_N, чтобы суммарная ошибка на всех этапах была минимальна:

$$\sum_{k=1}^{n} \mathbb{L}(y_i, \hat{y}) = \sum_{i=1}^{n} \mathbb{L}(y_i, \sum_{i=1}^{N-1} b_i(x_k) + b_N) \to min$$

Значение b_N, минимизирующее сумму ошибок прогноза, является антиградиент функции $\sum_{k=1}^{n} \mathbb{L}(y_i, \hat{y})$, то есть значение, направленное против направления наискорейшего роста функции - градиента. Таким образом, на каждом шаге алгоритм b_i будет подбираться таким образом, чтобы он был максимально близок к антриградиенту суммы функции потерь.
Проблема градиентного бустинга может быть скрыта в недостаточно хорошей аппроксимации некоторым базовым алгоритмом, например, решающим деревом, значения антиградиента, что приводит к "движению"алгоритма b_i на каждом шаге в неверном направлении. Для решения этой проблемы предлагается взять в расчет показатель $\eta \in (0, 1]$, отвечающий за урезания шага на пути на пути к итоговому алгоритму $a_N(x)$, теперь $a_N(X) = \sum_{n=1}^{N-1} b_n(x) + \eta \cdot b_N(x)$.

Вспомогательное: cross validation через пакет caret

Процедура кросс-валидации или так называемого скользящего контроля позволяет эмпирически, на основе обучающей выборки оценить предсказательную силу моделей, а так же оптимизировать значения определенных параметров, от которых зависит эффективность алгоритма. Разумеется, можно самостоятельно создать сетку ранжирования необходимых параметров, перебрать все возможные варианты комбинаций и оценить такое же число моделей, выбрав впоследствии оптимальную. Однако пакет caret с функцией train позволяет не

просто обучить машину, но и одновременно провести кросс-валидацию, тем самым оценив значения параметров, от которых зависит работа каждого из алгоритмов. Учитывая относительную простоту алгоритма случайного леса оценивается лишь единственный показатель - mtry, отвечающий за количество случайных регрессоров, выбираемых на каждом шаге. Для метода опорных векторов подбираются две величины: C - величина штрафа за нарушения и σ - параметр Гауссовой радиальной функции, отвечающий за величину разброса наблюдений друг от друга. Для градиентного бустинга настраиваемых параметров в разы больше. Среди них есть, например, eta - так называемая длина шага, регулирующая степень смещения в сторону построенного алгоритма, что предотвратит переобучение; max_depth - глубина дерева и т.д.

Извлечение переменных

Кроме предоставляемых показателей в изначальном наборе данных, представляется возможным извлечь еще несколько объясняющих переменных, которые смогут улучшить прогнозную силу выстраиваемых моделей.
Прежде всего, из показателя **due**, содержащего в себе информацию о дате и времени заказа в формате: YYYY-MM-DD HH:MM:SS, можно изъять несколько весьма полезных переменных. Поскольку и тренировочная, и тестовая выборки сформированы в течение одного, 2014, года, значение года как такового нас не интересует.

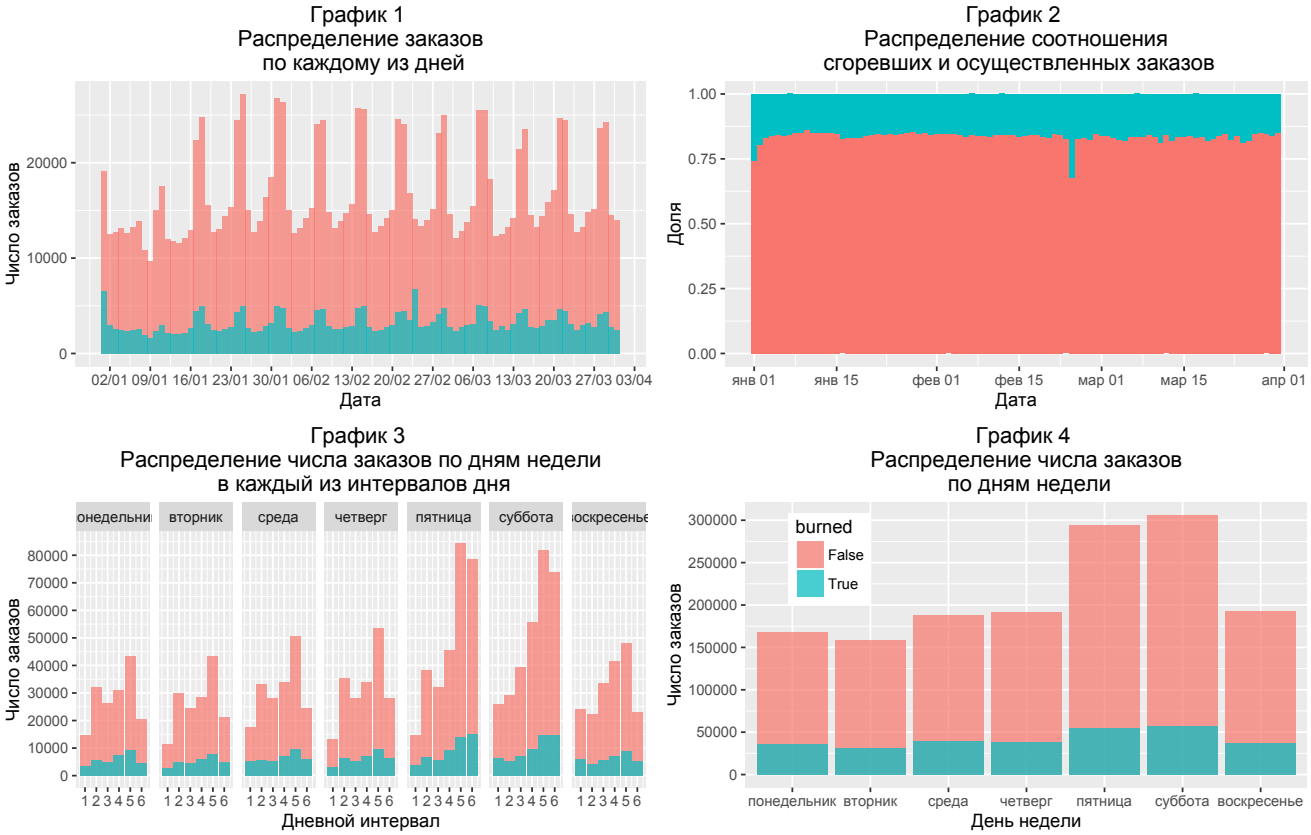
1. Из интуитивного предположения о зависимости от месяца количества заказов, а за ним и доля отменных может меняться, извлечем значение месяца для каждого заказа и поместим в переменную **month**.

2. Далее, само значение даты, если, конечно, это не праздник, в числовом выражении нам мало о чем говорит. Поэтому извлекая значение даты для каждого заказа, одновременно присваиваем ему значение дня недели, помещая эту информацию в переменную **dayofweek**. Это сделано из разумных предположений о разности концентрации заказов в будние дни, а так же в выходные – пятницу и субботу, когда население Москвы отправляется в бары и рестораны. Так же основываясь на этой логике, введем дамми-переменную **weekend**, которая принимает значение 1, если заказ был сделан в пятницу или субботу с 4 часов вечера и до полночи, и 0, если в любой другой промежуток времени.

3. Значение часа, в который был сделан заказ, также играет весомую роль. Разумеется, количество заказов поздней ночью разительно отличается от количеств заказов в утренние и вечерние часы-пик, а вместе с ним разнится и количество отмененных заказов. Таким образом, извлекая значение часа, когда был сделан заказ, присвоим ему одного из значений от 1 до 6 из созданной искусственно переменной **interval**. Что есть переменная interval? Весь дневной промежуток в 24 часа разбивается на 6 интервалов:

hour	interval
00:00 – 04:00	1
04:00 – 08:00	2
08:00 – 12:00	3
12:00 – 16:00	4
16:00 – 20:00	5
20:00 – 24:00	6

4. Отдельного внимания заслуживают праздничные дни, в которые количество заказов значительно увеличивается. С целью учета таких дней была введена дамми-переменная **holiday**, принимающая значение 1 для заказов 1 января, 14 и 23 февраля, а также 8 марта, и 0 в любые другие дни.



Итак, как видно из Графика 2, практически ежедневно соотношение долей сгоревших и осуществленных заказов держится на едином уровне, значительно изменяясь лишь в два дня: 1 января и 23 февраля. Эти даты являются праздничными днями и выделяются посредством переменной holiday. Также на Графике 1 прекрасно видно, как каждую неделю в пятницу и субботу значительно увеличивается число заказов, чему в подтверждение идет и График 4, отображающий кумулятивное число заказов по дням недели. В то время как практически во все дни сумма количества заказов равняется приблизительно 200000, в пятницу и субботу эта цифра больше в полтора раза. В конечном счет, График 3 отображает, насколько сильно разнится кумулятивное количество заказов, а вместе с ним и количество отмененных, в зависимости от заданного интервала времени дня.

Далее, обратимся к переменной **dist**, показывающей расстояние в метрах по прямой от места заказа до точки назначения. Первая проблема заключается в том, что определенная часть данных содержит значение 0 и -1, что говорит о об отсутствии информации для этой переменной. Исключить наблюдения, содержащие такие значения, невозможно по причине наличия таких же наблюдений в тестовой выборке. Также существуют крайне "подозрительные"наблюдения, для которых длина маршрута неприлично мала, например, менее 50 метров, либо крайне велика - более 200 километров. Для этого предложено на основе переменной dist сформировать новую факторную переменную **distint**, которая будет принимать значения от 0 до 5. Данная переменная является классификатором показателя dist, распределяя маршруты по "отсутствуют данные" – 0,

"крайне короткий" – 1, "короткий" – 2, "средний" – 3, "длинный" – 4, "крайне длинный" – 5. Однако для классификации необходимо определить граничные значения переменной dist.Для начала посмотрим на общее распределение данной переменной без значений 0 и -1. График строится лишь для дистанций, не превышающих 100 км, поскольку доля наблюдений, дистанция в которых превышает 70 км, крайне незначительно относительно общего числа наблюдений, их размещение на графике не изменит общей картины, а лишь ухудшит общий вид графика.

Теперь в поисках примерной границы для коротких дистанций посмотрим на вероятность отказа в каждом из промежутков для маршрутов до 2 км. Для этого отберем заказы, длина маршрута которых не превышает, 2 км и разделим на 40 отрезков по 500 метров. Для каждого отрезка посчитаем свою вероятность сгорания заказа. Как видно на Графике 1, для отрезков под номерами меньше 10-го достаточно велика вероятность отказа, а так же динамика изменения крайне резка. После 10-го отрезка, вероятность продолжает снижаться, однако не столь круто. Именно поэтому границей, разделяющей "крайне короткие"и "короткие"дистанции, устанавливается 500 метров.

Далее, проделаем аналогичные, по сути, операции только для установления границы между короткими и средними дистанциями. Для этого отберем заказы, длина маршрута которых находится в пределах от 2 до 50 км, и разделим на 48 отрезков по 1000 метров. Для каждого отрезка посчитаем свою вероятность сгорания заказа. Как видно на Графике 2, для отрезков под номерами меньше 8-11 вероятность воспроизводит тренд предыдущего графика - она снижается, после чего меняет свое направление на рост. Поэтому границу в 10 км устанавливаем как показатель разграничивающий "короткие"и "средние"дистанции. Так же на этом графике заметно, как начиная с 32 интервала, соответствующего дистанции приблизительно в 35 км, вероятность начинает зигзагообразно скакать, что говорит об изменении в структуре данных. Как видно из Графика 1, начиная от 35 километров количество наблюдений крайне мало по сравнению с предыдущими интервалами. А значит, в каждый из интервалов длиною в 1 км уже будет попадать не примерно одинаковое и меньшее число наблюдений, что заставляет вероятность скакать. Таким образом 35 км устанавливаем как верхнюю границу "средней"дистанции.

Отследить последнюю границу удастся, отобрав заказы, длина маршрута которых лежит от 35 до 200 км, с шагом в 15 км. Согласно Графику 4, начиная с интервала 4, соответствующего длине в 100 км, вероятность начинает резко увеличиваться. Поэтому, начиная со 100 км, любой заказ, дистанция в котором превышает данный порог, будет считаться крайне длинной. По факту, это полностью подчиняется обыденной логике: такси обычно не заказывают на столь большие расстояния.

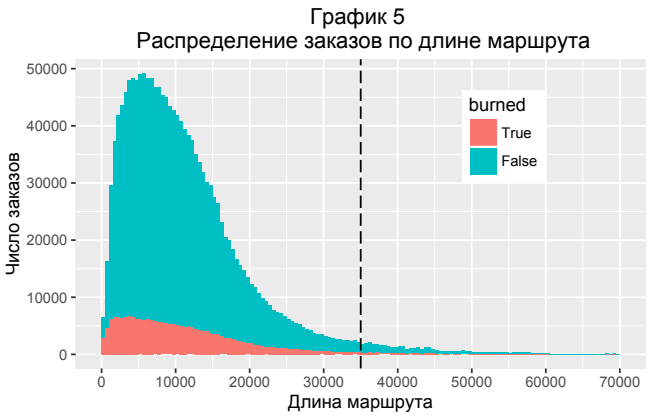


График 5
Распределение заказов по длине маршрута

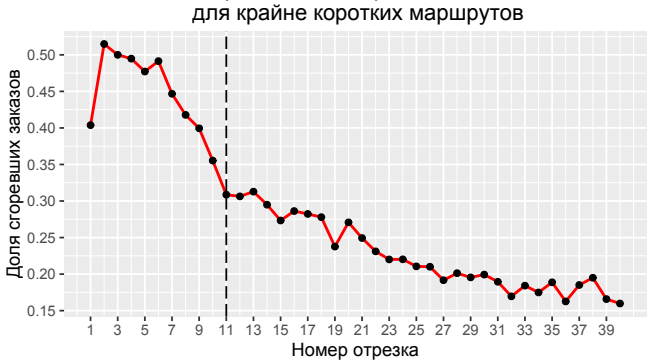


График 6
Распределение сгоревших заказов
для крайне коротких маршрутов

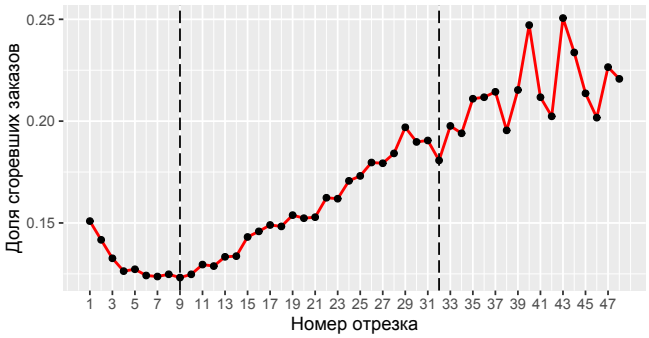


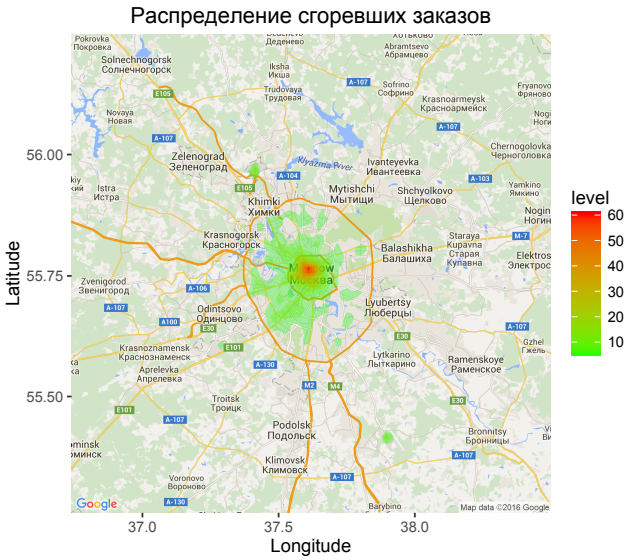
График 7
Распределение сгоревших заказов
для коротко-средних маршрутов

График 8
Распределение сгоревших заказов
для средне-длинных маршрутов

dist	distint
0 или -1	0
<500 m	1
500 m <...<3 km	2
3 km <...<35 km	3
35 km <...<100 km	4
>100 km	5

Кроме всего прочего, существуют переменные **lat** и **lon**, отвечающие за географические координаты места, куда был сделан заказ. Из всех, около 1800000, заказов примерно 300 тысяч сгорели.Таким образом, отобразив на карте сгоревшие заказы, можно обнаружить локальные области, в которых велика концентарция отмененных заказов.

Как видно на карте Москвы, львиная доля отмененных заказов приходится на западную часть столицы, с очагом в районе северо-запада Садового кольца. К сожалению, локализовать область с самой большой концентрацией отмененных заказов не удастся, поскольку несгоревшие заказы распределены еще плотнее и многочисленнее. Однако на левой карте прекрасно выделяются главные аэропорты Москвы: Шереметьево, Внуково и Домодедово, в которых также достаточно высока концентрация отмененных заказов. Ровно как на правой карте выделяется отдельные локации такие, как Белорусский и Казанский вокзалы, район Красного октября с прилегающими к нему клубами, а так же Новый Арбат. Введем дамми-переменную **location**.Для каждой локации очертим квадрат, при попадании координат заказа в который, location = 1.



Результаты моделей и выводы

Ввиду слабых мощностей рабочего компьютера для обучения машины использовались далеко не весь набор данных. Случайным образом было отобрано около 36 тысяч наблюдений, которые уже затем были разделены на обучающую и тестовую выборки в отношении $4:1$. Как и было рассказано выше, тестировались три алгоритма: случайный лес, метод опорных векторов и градиентный бустинг. Ниже представлена матрица ошибок, а так же использованная метрика точности - $Accuracy = \dfrac{[F,F]+[T,T]}{[F,F]+[T,T]+[F,T]+[T,F]}$, то есть указывающая на долю верно предсказанных значений во всем объеме тестовой выборки.

Confusion matrix

Random forest		
	False	True
False	5911	90
True	935	237

SVM		
	False	True
False	5909	92
True	930	242

XGBoost		
	False	True
False	5905	96
True	920	252

Accuracy

Random forest	SVM	XGBoost
0.8571	0.8579	0.8584

По таблице видно, что каждый алгоритм не уловил порядка 900 отмененных заказов, классифицируя их неверно как состоявшиеся.

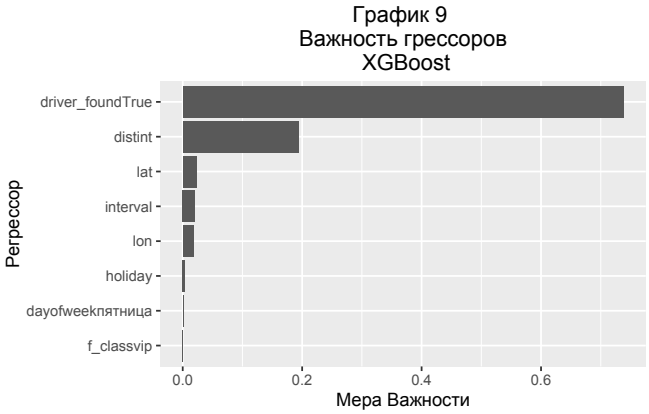


График 9
Важность грессоров
XGBoost

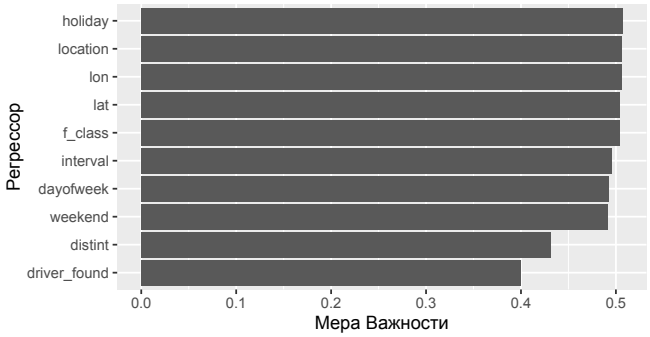


График 10
Важность грессоров
SVM

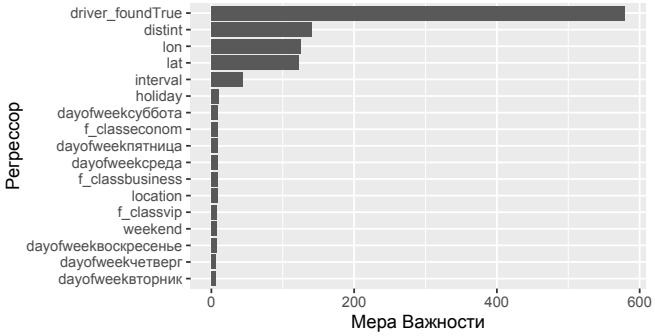
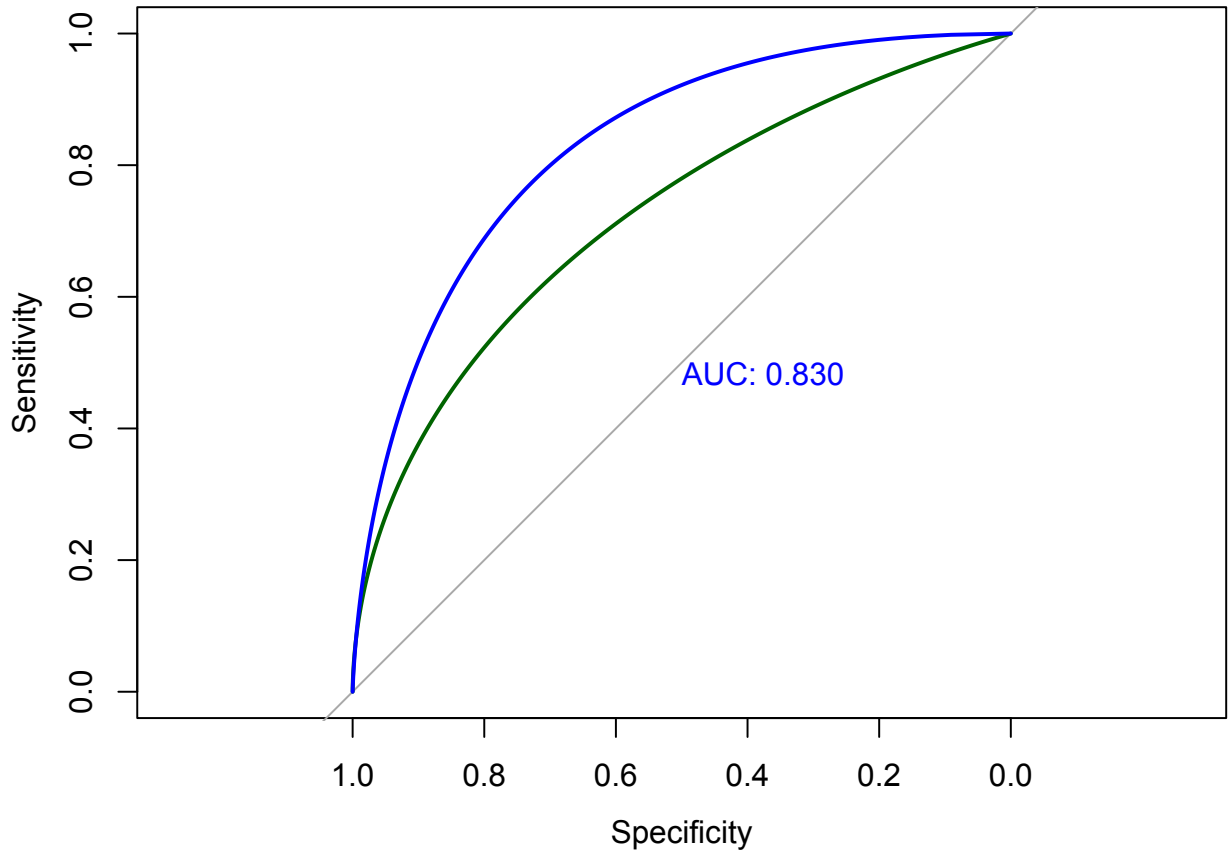


График 11
Важность регрессоров
Random Forest



Итак, как видно из графиков 9-11, отображающих степень важности регрессоров в каждом из алгоритмов, лидером является переменная, отвечающая за факт обнаружения водителя. На самом деле, это вполне ожидаемо, поскольку

даже из логических соображений данная переменная кажется крайне универсальным инструментом для предсказания отмены любого заказа. Чего не скажешь о другим регрессорах, которые, скорее, концентрируются на локальных характеристиках отдельного заказа. Крайне весомой также оказалась введенная переменная distint, которая является, скорее, индикатором, насколько реалистична длина задаваемого маршрута. Для случайного леса также оказались важны координаты заказа, а также то, какому интервалу времени принадлежит заказ.

Последний график отображает предсказательную силу моделей посредством построения ROC-кривой и сравнением AUC-метрики, отвечающей за площадь под ROC-кривой. К сожалению, по техническим причинам не удалось отобразить кривую для SVM. Однако случайный лес, отображенный синим цветом, показал себя значительно, на удивление, лучше, чем xgboost, отмеченный зеленым цветом. $AUC_{rf} = 0.83 > AUC_{xgb} = 0.72$.

Таким образом, было получено несколько неплохих моделей, способных спрогнозировать определенным образом статус заказа. Возможно, каждую из моделей можно улучшить путем извлечения дополнительных объясняющих переменных. Также, возможно, общую силу моделей можно было бы повысить, если бы в статистике компании Яндекс.Такси фиксировались в момент заказа и другие факторы, например, погодные условия или ориентировочная стоимость поездки.