**Как мужики машину думать учили**

Собрались как-то в Лесных Далях старожилы мысли оценочной – мастера ценовой канцелярии, и стали размышлять, как бы им узнать, почём народ торгует землёй русской, и за какую землицу какие деньги просит. Посидели они, покумекали, и решили, что надобно изначально создать чудо-юдо-паука машинного, который ползал бы по сети компьютерной, Интернетом именуемой, которой уже вся Земля опутана, и собирал бы искусы торговые, объявлениями о продаже ныне называемые.

Сказано – сделано. Подрядили на работу умельцев мозговитых, и создали чудо-юдо-паука машинного, по сайтам ползучего. Стал он ползать, и собирать искусы торговые, объявлениями называемые.

Настало время смотреть, почём же землицей русской народ приторговывает, и тут выяснилось, что объявления те торговые есть не что иное, как жижа словесная, именуемая ныне неструктурированной. Чего только не пишет народ про землю русскую, благо ещё, что не матерно. А старожилам мысли оценочной надобно знать, какие на той земле есть комфорты современные, коммуникациями именуемые. И вот стали они вновь думать, да лбы свои тереть: как бы им разобрать жижу словесную, да разуметь о коммуникациях, которые либо уже есть на наделе земельном, либо находятся около. И нарешали они создать машину самодумающую, чтобы сама читала жижу словесную, а потом сообщала, что же там на земле с комфортами современными, коммуникациями именуемыми.

Сказано – сделано. Озадачили работой умельцев мозговитых, а те и говорят, мол, коли есть уже жижа словесная, DataSet’ом по науке именуемая, то чтобы машину самодумающую сделать, надо самим как следует подумать, и пройти три больших испытания: поставить нам задачу, после чего мы создадим действующую обучающуюся модель, которую впоследствии должны будем «вживить» в машину, после чего она и станет самодумающей. Или на современном языке: постановка задачи – DataSet – создание модели – продакшн (техническая реализация).

То, что нужно сделать с жижей словесной, относится к задаче классификации, то бишь, определению состояния каждой из коммуникаций исходя из пяти придуманных нами классов: «есть»; «нет»; «есть возможность подключения»; «нет возможности подключения» и «информация отсутствует». А из коммуникаций мы решили изучать (пока) лишь три: холодную водицу; электричество, чаще именуемое в народе светом; и газ подземный, для обогрева и приготовления еды используемый.

И начали создавать машину самодумающую. А работа та созидательная состояла из нескольких частей.

Часть первая.

Обмозговали наш DataSet, обдумали задачи, и начали кумекать, как же нам создать эту машину самодумающую? Уж чего-чего, а мыслей и идей не счесть у народа русского, но ведь и ассигнования на проект не безграничные, и надобно было сопоставлять полёт мыслей русских с возможностями по их воплощению.

Часть вторая.

Здесь мы обдумывали, а действительно ли нам нужна машина думающая, или достаточно будет небольшой «дрессировки» на специальных наборах слов или их частей, называемых регулярными выражениями. Например, если в словесной жиже встречается выражение «на участке есть свет» - то это значило бы, что электричество на участке есть. Поначалу думали, что всё очень легко получится, достаточно лишь подобрать правильные регулярные выражения. Но… словесная жижа – жижа и есть, тем более на «великом и могучем» сотворённая. Вышепомянутый пример может быть и таким: «газ на участке есть свет пока не проведен». Да-да, вот так, без запятых. И здесь тоже встречается выражение «на участке есть свет», но есть ли на земле электричество? Или вот ещё пример наглядный: «без комуникаций но все рядом свет прям на участке».

1. **БЕЗ** комуникаций – сразу хочется сказать, что ничего нет.

2. Ко**м**уникаций – орфографическая ошибка.

3. **БЕЗ**… **НО** всё рядом – смысловая конструкция.

4. Без… но **ВСЁ** рядом.

5. Без… но всё **РЯДОМ**.

6. «Свет» – разговорный синоним, а бывают ещё эл-во, э-во, электрифицирован, «КТП в 100 метрах», 15кВт, техусловия и много ещё чего.

7. «Прям» – оборот «великого и могучего».

8. «свет прям на участке» - никак нельзя сказать, что света нет.

А можно ли всё-таки построить машину на регулярных выражениях? Можно всё, но каковы будут быстродействие (с учётом огромного количества регулярных выражений), а, главное, какая будет точность? Мозговитые умельцы предсказывали примерно 25 долей из ста, а метрики пробных запусков были значительно ниже.

В итоге, решили учить машину думать.

Часть третья.

На данном этапе работ следовало принять решение о том, использовать ли готовую машину (а такие тоже есть), или создавать свою. С одной стороны, известно стремление русского человека к халяве. «По щучьему велению, по моему хотению» можно использовать и здесь. Но проблема в том, что хотение – наше, а веление-то «щучье». Что, а, главное, каким образом будет повелевать «щука» - можно до конца и не узнать. Именно к такому заключению пришли умны молодцы, и решили, что «своя рубашка ближе к телу». Понятное дело, что без чужеродных заимствований, именуемых библиотеками и фреймворками, обойтись будет сложно, но здесь необходимо исходить из разумной целесообразности: баланса между ассигнованиями на проект и глубиной собственных разработок. При этом собственные разработки хороши тем, что позволяют при должной модульности проекта «пощупать» различные алгоритмы, и оценить их в деле. Именно этим путём и решено было идти.

Часть четвёртая.

На чём всё это делать? Куда «вживлять» искусственный мозг? Здесь многое зависит от финального «продакшна» - совокупности технической, программной и пользовательской составляющих. Если самодумающая машина ни к чему не привязана (эдакая «Голова профессора Доуэля»), и является «вещью в себе», предназначенной для того, чтобы «кушать» какую-то неструктурированную информацию, и «выплёвывать» требуемые классы/категории – это одно. А когда искусственный мозг является часть большого организма – это уже другое. В нашем случае именно так есть. Чудо-юдо-паук ведь уже есть, хоть и «безголовый», и необходимо было пришить «мозг» к «пауку». Мало того, паук получился весьма аккуратным, и то, что натаскал к себе в «амбар» (который именуют обычно базой данных), аккуратно разложил по полочкам, и надо было научить «мозг» ещё и по этим полочкам лазить, а потом и назад складывать так же аккуратно.

Принципы технической и финансовой целесообразности проекта привели к отказу от платных баз данных в пользу бесплатной PostgreSQL. Немного подробнее остановимся на чудо-юдо-пауке. Он написан на C#, поскольку объединяет в себе множество различных библиотек (от управления браузером до оптического распознавания текста), имеет распределённую структуру с множеством пользователей, ролей и сущностей (с возможностью одновременной работы), программу установки, механизм обновления и ещё много чего. Использование C# предполагает использование .NET Framework от Microsoft, что в свою очередь предполагает использование в качестве операционной серверной системы Microsoft Windows Server. Но выход в свет нового .NET Core, умеющего работать на Linux, позволил и здесь отказаться от платных решений. Плюс в том, что это бесплатно. Минус в том, что на этом стеке нужно уметь работать. Но, благо, наши мозговитые умельцы действительно оказались умельцы.

Вот так устроен наш чудо-юдо-паук. Теперь же стояла задача породить искусственный интеллект – нашу самодумающую машину. Использование в пауке .NET Core и C# подталкивало к использованию F#, и мы начали «щупать» этого «зверя». Функциональный язык F# оказался весьма неплох, но у него выявилось два основных недостатка: небольшой объём решений на кириллице, и не столь развитые сообщества, именуемые также community (когда на форумах и хабах можно найти ответы на интересующие вопросы). С языком R всё оказалось гораздо интересней как в части поддержки кириллицы, так и в части community. После обсуждений было решено двигаться дальше в связке .NET Core, C# и R. Однако, в дальнейшем возникла необходимость частично использовать Python. Не потому, что C# и R не справлялись с поставленными задачами, а потому что наш искусственный мозг должен работать как с пауком, так и «лазить по полкам в амбаре» по просьбам юзеров (т.е. работать с базой данных), а «лазилка» по «амбару» (web-интерфейс) написана на Django+Python. Мало того, предусмотрен функционал, который позволяет «скормить» нашему мозгу информацию из внешних источников, а на выходе получить структурированную информацию. В общем, для решения именно нашей задачи оптимальным оказалось использование связки .NET Core, C#, R и чуть-чуть Python.

Изначально предполагалась явно выраженная модульная структура, причём как в макро-, так и в микро-масштабах. Это означает, что можно заменить как самодумающую машину целиком (расположенную на отдельном виртуальном сервере), так и её компоненты, чем мы активно пользовались, тестируя различные алгоритмы, библиотеки и фреймворки, тем самым подбирая оптимальные варианты для решения конкретных прикладных задач.

Часть пятая.

Кто, как и в каких условиях создавал нашу машину.

Изначально были попытки работать с вольнонаёмными мозговитыми умельцами, или в переводе на новояз: работать на аутсорсинге с фрилансерами. Плохо это, или хорошо? Не плохо, и не хорошо! Для каждой из задач есть своё решение. Для решения нашей задачи вольнонаёмники подходили в меньшей степени. Объясню почему. Во-первых, вольнонаёмным для работы нужно постоянно отправлять депеши, именуемые нынче техническими заданиями. Нет, само по себе техническое задание – это не плохо, это даже очень хорошо, ибо позволяет чётко структурировать все процессы и аспекты выполнения проекта. Но для столь творческого проекта предвидеть и спроектировать сразу всё практически невозможно, и по ходу выполнения проекта нередко возникают уточняющие ТЗ. А частенько доходит до того, что намного проще забежать к мозговитым умельцам в их палаты творческие, и на пальцах, да вприсядку за пару минут объяснить «чё да как», чем сидеть полдня в роли дятла клавиатурного. И подобных моментов бывает очень много, а «на пальцах, да вприсядку» через Ентернет аль телефон, конечно, тоже можно, но, увы, не столь эффективно. Во-вторых, когда проект выходит на стадию работы с конечными пользователя, периодические возникают экстренные нештатные ситуации, требующие различного (а порой не достаточно определённого) времени на их решение. И в этом плане человек из своего коллектива подходит для решения подобных задач, зачастую, лучше. В общем, для оперативного решения повседневных рабочих вопросов в подобных проектах эффективнее набирать мозговитых умельцев себе в дружину на постоянное место службы.

Что касается технической стороны вопроса, то ничего сверхъестественного здесь не требуется. И чудо-юдо-паук, и «абмар» для хранения найденного, и машина самодумающая располагаются на одном физическом сервере, хотя и на разных виртуальных серверах. Физический сервер вполне заурядный: Intel Xeon E5 2680 v2 и 64Гб, правда дисковый «амбар» довольно большой, поскольку хранить приходится не только жижу словесную, но и визуальное подтверждение этой самой жижи, в народе именуемое скринами.

Обучение моделей производится на локальном компьютере.

Часть шестая.

Ну вот мы и подобрались к результатам нашей задачи – созданию самой машины. Как уже упоминалось выше, данная задача – это задача классификации, т.е. каждое объявление о продаже земли русской необходимо отнести к одному из придуманных нами классов в части информации о коммуникациях. Если этих классов два, то задача называется бинарной классификацией, например, есть или нет на участке коммуникация. Но бинарная классификация вряд ли подойдёт для решения нашей задачи, поскольку в класс «есть» попадут участки, на которых уже есть коммуникации, и участки с возможностью подключения, а в класс «нет» попадут участки, по которым в объявлениях просто нет никакой информации. Если классов более двух, то это многоклассовая (мультиклассовая) классификация. И вот это уже наша тема.

Суть процесса заключается в том, что обрабатывая словесную жижу объявлений при помощи словаря (токенизотора – набора токенов – объектов, создающихся из лексем) и какого-то алгоритма, мы классифицируем наш объект, т.е. относим его к одному из определённых нами классов.

Первая подзадача. Токенизатор. Его мы создавали сами, поскольку от его количественных и качественных характеристик существенно зависит итоговый результат. Здесь важно работать на стыке трёх основных предметов: программирования, русского языка (во всём его многообразии) и практики оценочной деятельности (чтобы не всё подряд добавлять в токенизатор, тем самым не перегружая его избыточной информацией).

Вторая подзадача. Обучающая выборка. Чтобы наша машина была самодумающая, мы её сначала должны обучить на конкретных примерах. Это называется обучение с учителем. Другими словами, мы должны показать машине реальный набор словесной жижи (объявлений), и при этом сообщить какая жижа к какому классу относится, чтобы машина «поняла» что к чему. Если ещё проще, то мы даём машине прочитать объявление, а после этого сообщаем ей о состоянии коммуникаций на данном участке (в нашем случае это: «есть»; «нет»; «есть возможность подключения»; «нет возможности подключения» и «информация отсутствует»). Здесь также желательно наличие практики оценки, которое позволяет более точно классифицировать ту или иную смысловую структуру текста объявления (словесной жижи) с точки зрения дальнейшей оценки объекта (в первую очередь для разделения классов «есть» и «есть возможность подключения»).

Третья подзадача. Выбор алгоритма. Их много. Очень не хочется обсуждать и осуждать какие-то отдельно взятые алгоритмы. Изначально задуманная модульная структура нашей машины позволяет «пощупать» в деле, и затем «прикрутить на продакшн» различные алгоритмы, чем мы не преминули воспользоваться, ибо теория – это одно, а прикладные задачи – другое. Пробовали мы на словесной жиже выращивать «решающие деревья». Плоховато растут ☺ И плоды какие-то не ярко выраженные. С той же обучающей выборкой оказалось намного эффективней вытаскивать из словесной жижи коммуникации при помощи «невода» (рекуррентной нейронной сети - RNN). Если в цифрах, то на одних и тех же токенизаторе и обучающей выборке деревья показали точность порядка 30-40%, а RNN - порядка 75-85%. В общем, решили пока остановиться на алгоритме глубокого машинного обучения LSTM (RNN) из фреймворка Keras, как справляющемся с поставленной задачей, и имеющем потенциал для роста точности.

Четвёртая подзадача. Ресурсозатратность. Ещё один плюс собственной разработки – более глубокая возможность оптимизации процессов на продакшне в отношении серверных ресурсов. В нашем случае мы имеем три обученные модели (вода, свет, газ) и токенизатор (лексер). Изначально наша самодумающая машина «думала» по очереди: вода, свет, газ. Последующая оптимизация позволила загружать одновременно все три модели, и работать параллельно.

Пятая и последняя подзадача. Финальная реализация.

Создать токенизатор, выбрать алгоритм, и обучить модель, т.е. создать нашу самодумающую машину – это лишь одна из частей общей задачи. «Прикрутить» её на продакшн – не менее сложная задача. Поначалу хотели, чтобы наши искусственные мозги постоянно были включены, независимо от того, ползает чудо-юдо-паук по сети, или отдыхает, но потом решили, что это нецелесообразно, и сделали нашу самодумающую машину вызываемой юзерами: по нашему велению, по юзера хотению, а ну-ка жижа словесная, превратись в информацию о коммуникациях. Так оказалось и практичней, и удобней, и позволило предусмотреть возможность внешней загрузки информации для машинной обработки.

В качестве заключения.

Сложно ли создавать подобные машины в прикладных целях? Если рассматривать в комплексе, то, пожалуй, да. Приходится решать множество комплексных задач, привлекая высококлассных специалистов. Но, думается, что в ближайшее время появится большее количество готовых решений, а их адаптация для прикладных задач упростится. И немного о трудностях и преимуществах самостоятельного создания продуктов, использующих машинное обучение при работе с кириллическими текстами.

Трудности:

* Необходимость привлечения недешёвых специалистов;
* Небольшой выбор готовых решений на кириллице, а те, которые есть, ещё довольно «сырые». В итоге подготовка собственного качественного токенизатора – одна из сложных ключевых задач;
* Необходимость решать множество задач (подчас взаимоисключающих).

Преимущества:

* Токенизатор, оптимизированный под конкретные задачи;
* Глубокие возможности доработки, настройки и оптимизации, и как следствие повышение точности и быстродействия;
* Гибкие функциональные возможности;
* Возможность смены алгоритмов.