**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра дискретной математики и алгоритмики**

ПРУДНИКОВ Евгений Александрович

**РАЗРАБОТКА МОДЕЛЕЙ ПРЕДУПРЕЖДЕНИЯ СБОЕВ НА**

**ПРОИЗВОДСТВЕННОЙ**

**ЛИНИИ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ**

Магистерская диссертация

Специальность 1-31 81 09 «Алгоритмы и системы обработки больших объемов информации»

Научный руководитель

Соболевская Елена Павловна

кандидат физико- математических наук,

доцент

Допущена к защите

« » 2017 г.

Зав. кафедрой дискретной математики и алгоритмики

В. М. Котов

доктор физико-математических наук, профессор

Минск, 2017

**ОГЛАВЛЕНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc483444692)

[ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ ПРЕДУПРЕЖДЕНИЯ ОШИБОК НА ПРОИЗВОДСТВЕННЫХ ЛИНИЯX 4](#_Toc483444693)

[Физические 4](#_Toc483444694)

[Программные 5](#_Toc483444695)

[ОБЗОР МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ 6](#_Toc483444696)

[Машинное обучение 6](#_Toc483444697)

[Общая постановка задачи обучения по прецедентам 6](#_Toc483444698)

[Основные стандартные типы задач 7](#_Toc483444699)

[Выводы 10](#_Toc483444700)

[СРЕДСТВА РАЗРАБОТКИ 11](#_Toc483444701)

[АНАЛИЗ ДАННЫХ, ГЕНЕРИРУЕМЫХ ДАТЧИКАМИ ПРОИЗВОДСТВЕННОЙ ЛИНИИ 13](#_Toc483444702)

[Анализ 13](#_Toc483444703)

[Выбор метрики качества модели 16](#_Toc483444704)

[ПОСТРОЕНИЕ МОДЕЛИ ПРЕДУПРЖДЕНИЯ СБОЕВ НА ПРОИЗВОДСТВЕННОЙ ЛИНИИ 20](#_Toc483444705)

[Описание алгоритма 20](#_Toc483444706)

[ЗАКЛЮЧЕНИЕ 24](#_Toc483444707)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc483444708)

[ПРИЛОЖЕНИЕ А 26](#_Toc483444709)

[ПРИЛОЖЕНИЕ Б 27](#_Toc483444710)

[ПРИЛОЖЕНИЕ В 28](#_Toc483444711)

ПОСТАНОВКА ЗАДАЧИ

Производство и производственные линии стали неотъемлемой частью современного общества. Обеспечение строгого следования стандартам качества и безопасности является одной из главных задач предприятия. Современные производственные линии снабжены большим количеством различных датчиков, которые позволяют собирать информацию о производственном процессе. Обработка этой информации и разработка, на её основе, встраиваемой модели предупреждения сбоев поможет повысить эффективность производственной линии, а также предотвратить дополнительные финансовые издержки и, следовательно, удешевить продукт для конечного пользователя.

Целью магистерской диссертации является разработка моделей предупреждения сбоев на производственной линии с использованием методов машинного обучения. Исходя из указанной цели, можно выделить частные задачи, поставленные в магистерской диссертации:

* Обработка измерительных данных датчиков производственной линии
* Разработка моделей анализа обработанных данных для оценки вероятности возникновения сбоя на производственной линии

ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ ПРЕДУПРЕЖДЕНИЯ ОШИБОК НА ПРОИЗВОДСТВЕННЫХ ЛИНИЯX

В настоящее время современные производственные линии, используемые предприятиями, обычно оснащены множеством систем предупреждения ошибок и контроля качества продукции, а также системами помощи оператору.

Условно можно разделить эти системы на два класса: физические, то есть различные датчики сбора оперативной информации, и программные, то есть программы, которые обрабатывают информацию с датчиков и предоставляют её в удобном виде, а также программы помощи операторам линии.

Физические

К физическим системам обнаружения и предотвращения ошибок можно отнести различные датчики контроля движения детали по производственной линии, анализирующие расположение и траекторию перемещения материалов по производственной линии. Данные элементы бывают нескольких типов:

*Концевые датчики* - контролируют положение рабочих органов станка. Срабатывают, когда подвижный элемент оборудования достигает определенной точки. Чаще всего устанавливают такие элементы на осях роликовых платформ и конвейерных лент. При нарушении правильного положения ленты или расположенных на ней материалов концевик информирует оператора, что в работе оборудования произошел сбой.

*Оптические линейки* – набор оптоэлектронных датчиков, с помощью которых осуществляется измерение габаритов материалов, движущихся по производственной линии. Такие линейки помогают оценить точность манипуляций, проводимых с материалом на деревообрабатывающих и режущих станках. Информация, получаемая с помощью оптической линейки, в форме аналогового сигнала передается на УЦИ и визуализируется на дисплее в виде цифр.

*Ультразвуковые датчики* – осуществляют контроль и измерение расстояния от предмета до заданного объекта. Могут применяться для определения уровня прогиба полотна, скорости работы станка, промежутков между материалами на технологической ленте. Используются в отраслях деревообработки, машиностроения, производства пищевых продуктов и на предприятиях целлюлозно-бумажной промышленности.

*Лазерные датчики (дальномеры)* – оптические устройства для определения расстояния между объектами. Работа основана на принципе измерения времени пролета луча, диапазон действия – до нескольких десятков метров. Применяют дальномеры в различных областях промышленности.

*Индуктивные датчики* – это детекторы металла. Установив такие элементы на производственной линии, можно вести подсчет изготовленных металлических деталей, осуществлять распознавание и сортировку деталей различного типа, контролировать скорость подачи материалов с одного этапа производства на другой. Применяться такие датчики могут везде, где ведется работа с элементами из металла.

Программные

К программным средствам контроля качества изделий, а также помощи оператору можно отнести различное ПО для сбора (базы данных, ERP системы), анализа и отображения информации, поступающей с датчиков, установленных на производственной линии. Примерами могут служить такие компании как Videojet с ПО CLARiTY, Omron, Balluff. Также, и собственные системы производителей.

ОБЗОР МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

В настоящее время активно используются системы, построенные на основе алгоритмов машинного обучения. Такие системы сейчас используются во многих сферах науки и техники. Учитывая задачу, а также исходные данные мы можем обратиться к алгоритмам машинного обучения для её решения.

Машинное обучение

Машинное обучение (Machine Learning) — обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Различают два типа обучения. Обучение по прецедентам, или индуктивное обучение, основано на выявлении общих закономерностей по частным эмпирическим данным. Дедуктивное обучение предполагает формализацию знаний экспертов и их перенос в компьютер в виде базы знаний. Дедуктивное обучение принято относить к области экспертных систем, поэтому термины машинное обучение и обучение по прецедентам можно считать синонимами.

Машинное обучение находится на стыке математической статистики, методов оптимизации и классических математических дисциплин, но имеет также и собственную специфику, связанную с проблемами вычислительной эффективности и переобучения. Многие методы индуктивного обучения разрабатывались как альтернатива классическим статистическим подходам. Многие методы тесно связаны с извлечением информации и интеллектуальным анализом данных (Data Mining).

Наиболее теоретические разделы машинного обучения объединены в отдельное направление, теорию вычислительного обучения (Computational Learning Theory, COLT).

Машинное обучение — не только математическая, но и практическая, инженерная дисциплина. Чистая теория, как правило, не приводит сразу к методам и алгоритмам, применимым на практике. Чтобы заставить их хорошо работать, приходится изобретать дополнительные эвристики, компенсирующие несоответствие сделанных в теории предположений условиям реальных задач. Практически ни одно исследование в машинном обучении не обходится без эксперимента на модельных или реальных данных, подтверждающего практическую работоспособность метода.

Общая постановка задачи обучения по прецедентам

Дано конечное множество прецедентов (объектов, ситуаций), по каждому из которых собраны (измерены) некоторые данные. Данные о прецеденте называют также его описанием. Совокупность всех имеющихся описаний прецедентов называется обучающей выборкой. Требуется по этим частным данным выявить общие зависимости, закономерности, взаимосвязи, присущие не только этой конкретной выборке, но вообще всем прецедентам, в том числе тем, которые ещё не наблюдались. Говорят, также о восстановлении зависимостей по эмпирическим данным — этот термин был введён в работах Вапника и Червоненкиса.

Наиболее распространённым способом описания прецедентов является признаковое описание. Фиксируется совокупность n показателей, измеряемых у всех прецедентов. Если все n показателей числовые, то признаковые описания представляют собой числовые векторы размерности n. Возможны и более сложные случаи, когда прецеденты описываются временными рядами или сигналами, изображениями, видеорядами, текстами, попарными отношениями сходства или интенсивности взаимодействия, и т. д.

Для решения задачи обучения по прецедентам в первую очередь фиксируется модель восстанавливаемой зависимости. Затем вводится функционал качества, значение которого показывает, насколько хорошо модель описывает наблюдаемые данные. Алгоритм обучения (learning algorithm) ищет такой набор параметров модели, при котором функционал качества на заданной обучающей выборке принимает оптимальное значение. Процесс настройки (fitting) модели по выборке данных в большинстве случаев сводится к применению численных методов оптимизации.

Основные стандартные типы задач

*Обучение с учителем (supervised learning)*— наиболее распространённый случай. Каждый прецедент представляет собой пару «объект, ответ». Требуется найти функциональную зависимость ответов от описаний объектов и построить алгоритм, принимающий на входе описание объекта и выдающий на выходе ответ. Функционал качества обычно определяется как средняя ошибка ответов, выданных алгоритмом, по всем объектам выборки.

* *Задача классификации (classification)* отличается тем, что множество допустимых ответов конечно. Их называют метками классов (class label). Класс — это множество всех объектов с данным значением метки.
* *Задача регрессии (regression)* отличается тем, что допустимым ответом является действительное число или числовой вектор.
* *Задача ранжирования (learning to rank)* отличается тем, что ответы надо получить сразу на множестве объектов, после чего отсортировать их по значениям ответов. Может сводиться к задачам классификации или регрессии. Часто применяется в информационном поиске и анализе текстов.
* *Задача прогнозирования (forecasting)* отличается тем, что объектами являются отрезки временных рядов, обрывающиеся в тот момент, когда требуется сделать прогноз на будущее. Для решения задач прогнозирования часто удаётся приспособить методы регрессии или классификации, причём во втором случае речь идёт скорее о задачах принятия решений.

*Обучение без учителя (unsupervised learning).* В этом случае ответы не задаются, и требуется искать зависимости между объектами.

* *Задача кластеризации (clustering)* заключается в том, чтобы сгруппировать объекты в кластеры, используя данные о попарном сходстве объектов. Функционалы качества могут определяться по-разному, например, как отношение средних межкластерных и внутрикластерных расстояний.
* *Задача поиска ассоциативных правил (association rules learning).* Исходные данные представляются в виде признаковых описаний. Требуется найти такие наборы признаков, и такие значения этих признаков, которые особенно часто (неслучайно часто) встречаются в признаковых описаниях объектов.
* *Задача фильтрации выбросов (outliers detection)* — обнаружение в обучающей выборке небольшого числа нетипичных объектов. В некоторых приложениях их поиск является самоцелью (например, обнаружение мошенничества). В других приложениях эти объекты являются следствием ошибок в данных или неточности модели, то есть шумом, мешающим настраивать модель, и должны быть удалены из выборки, см. также робастные методы и одноклассовая классификация.
* *Задача построения доверительной области (quantile estimation)* — области минимального объёма с достаточно гладкой границей, содержащей заданную долю выборки.
* *Задача сокращения размерности (dimensionality reduction)* заключается в том, чтобы по исходным признакам с помощью некоторых функций преобразования перейти к наименьшему числу новых признаков, не потеряв при этом никакой существенной информации об объектах выборки. В классе линейных преобразований наиболее известным примером является метод главных компонент.
* *Задача заполнения пропущенных значений (missing values)* — замена недостающих значений в матрице объекты–признаки их прогнозными значениями.

*Частичное обучение (semi-supervised learning)* занимает промежуточное положение между обучением с учителем и без учителя. Каждый прецедент представляет собой пару «объект, ответ», но ответы известны только на части прецедентов. Пример прикладной задачи — автоматическая рубрикация большого количества текстов при условии, что некоторые из них уже отнесены к каким-то рубрикам.

*Трансдуктивное обучение (transductive learning).* Дана конечная обучающая выборка прецедентов. Требуется по этим частным данным сделать предсказания отностительно других частных данных — тестовой выборки. В отличие от стандартной постановки, здесь не требуется выявлять общую закономерность, поскольку известно, что новых тестовых прецедентов не будет. С другой стороны, появляется возможность улучшить качество предсказаний за счёт анализа всей тестовой выборки целиком, например, путём её кластеризации. Во многих приложениях трансдуктивное обучение практически не отличается от частичного обучения.

*Обучение с подкреплением (reinforcement learning).* Роль объектов играют пары «ситуация, принятое решение», ответами являются значения функционала качества, характеризующего правильность принятых решений (реакцию среды). Как и в задачах прогнозирования, здесь существенную роль играет фактор времени. Примеры прикладных задач: формирование инвестиционных стратегий, автоматическое управление технологическими процессами, самообучение роботов, и т.д.

*Динамическое обучение (online learning)* может быть, как обучением с учителем, так и без учителя. Специфика в том, что прецеденты поступают потоком. Требуется немедленно принимать решение по каждому прецеденту и одновременно доучивать модель зависимости с учётом новых прецедентов. Как и в задачах прогнозирования, здесь существенную роль играет фактор времени.

*Активное обучение (active learning)* отличается тем, что обучаемый имеет возможность самостоятельно назначать следующий прецедент, который станет известен. См. также Планирование экспериментов.

*Метаобучение (meta-learning или learning-to-learn)* отличается тем, что прецедентами являются ранее решённые задачи обучения. Требуется определить, какие из используемых в них эвристик работают более эффективно. Конечная цель — обеспечить постоянное автоматическое совершенствование алгоритма обучения с течением времени.

* *Многозадачное обучение (multi-task learning).* Набор взаимосвязанных или схожих задач обучения решается одновременно, с помощью различных алгоритмов обучения, имеющих схожее внутренне представление. Информация о сходстве задач между собой позволяет более эффективно совершенствовать алгоритм обучения и повышать качество решения основной задачи.
* *Индуктивный перенос (inductive transfer).* Опыт решения отдельных частных задач обучения по прецедентам переносится на решение последующих частных задач обучения. Для формализации и сохранения этого опыта применяются реляционные или иерархические структуры представления знаний.

Выводы

Для решения поставленных задач, наиболее подходящей методикой является использование методов машинного обучения. Так как поставленная задача и входные данные прекрасно вписываются в общую постановку задачи для алгоритмов машинного обучения и представляет из себя стандартную задачу *классификации с обучением с учителем.*

Так, входные данные можно рассматривать как неделимые события прохождения объектов производства по конвейеру с промежуточными результатами о которых известны измерения в контрольных точках линии производства, то есть конечное множество прецедентов (объектов, ситуаций), по каждому из которых собраны (измерены) некоторые данные. И методы машинного обучения, используя эти обработанные данные, позволяют предсказать результат прохождения объектом линии или какой-то её части, что более актуально, то есть построить модель для выявления общих закономерностей, присущих не только этой конкретной выборке, но вообще всем прецедентам, в том числе тем, которые ещё не наблюдались, которые позволяют определить будет ли сбой на производственной линии.

СРЕДСТВА РАЗРАБОТКИ

Для анализа исходных данных, а также разработки последующих моделей будут использованы следующие инструменты для работы с алгоритмами машинного обучения:

* Apache Spark
* Язык программирования python

*Apache Spark*— программный каркас с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию MapReduce с дисковым хранилищем, использует специализированные примитивы для рекуррентной обработки в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач, в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Проект предоставляет программные интерфейсы для языков Java, Scala, Python, R. Состоит из ядра и нескольких расширений, таких как Spark SQL (позволяет выполнять SQL-запросы над данными), Spark Streaming (надстройка для обработки потоковых данных), Spark MLlib (набор библиотек машинного обучения), GraphX (предназначено для распределённой обработки графов). Может работать как в среде кластера Hadoop под управлением YARN, так и без компонентов ядра Hadoop, поддерживает несколько распределённых систем хранения — HDFS, OpenStack Swift, NoSQL-СУБД Cassandra, Amazon S3.

Так как исходный объем данных достаточно велик (больше 9 Гб), то для анализа дынных удобно использовать Apache Spark, так как он проектировался как система вычислений для распределённых систем и позволяет работать с большими объемами данных при ограниченных ресурсах, не загружая весь объем данных в оперативную память. Однако при дальнейшем построении модели удобно использовать алгоритмы машинного обучения из библиотеки xgboost.

Вид Spark кластера для анализа данных представлен на рисунках 1, 2, 3.

Кластер Spark состоит из приложения мастера и подконтрольных ему приложений рабочих. Мастер получает от клиента (пользователя) задачу, строит граф вычислений и распределяет его на подчиненных, при этом, балансируя нагрузку. В данном случае кластер, состоящий из мастера и одной рабочего развернут локально на одном ноутбуке, при этом приложение мастера работает на локальном ай-пи адресе, тогда как приложение рабочего использует первый доступный ай-пи адрес сети, что соответствует реальной системе, так как чаще всего приложение мастера работает на машине пользователя и связывается с рабочими кластера по сети.

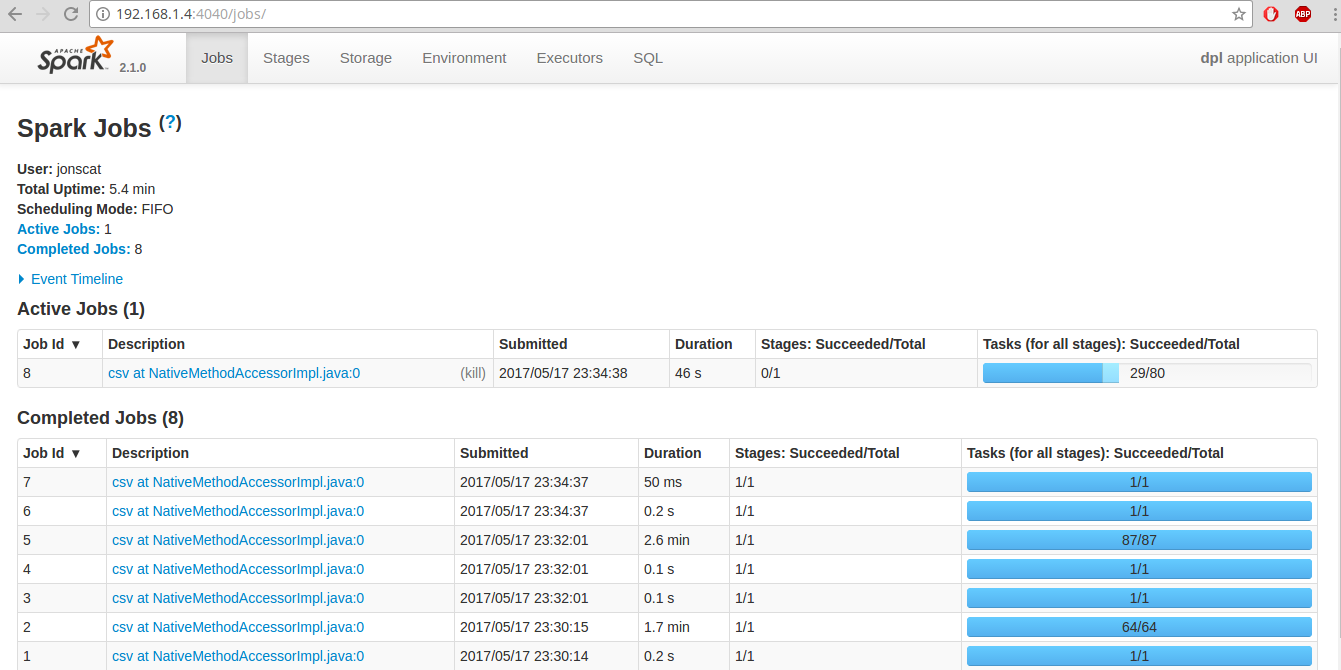


Рис 3. Пример развернутого кластера Spark. Консоль приложения. Пример выполнения задач.

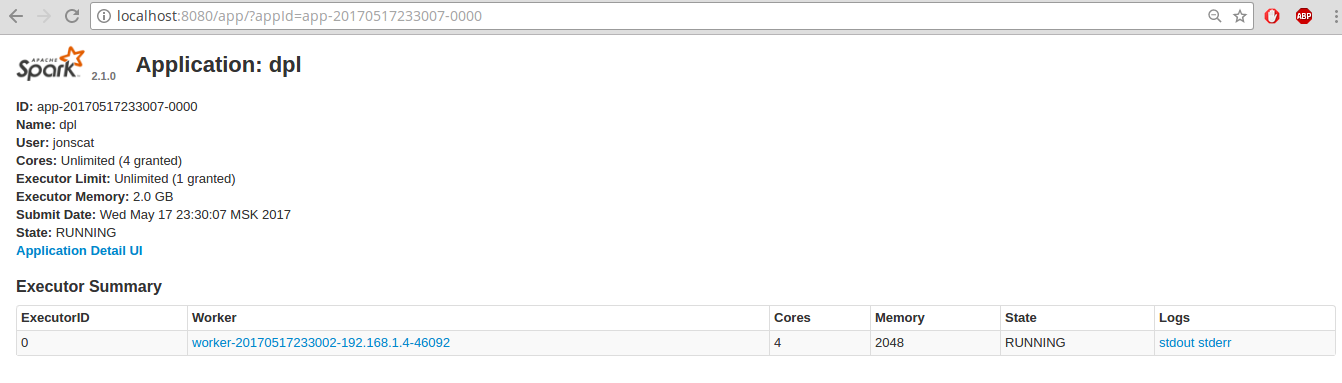


Рис 2. Пример развернутого кластера Spark. Консоль приложения.

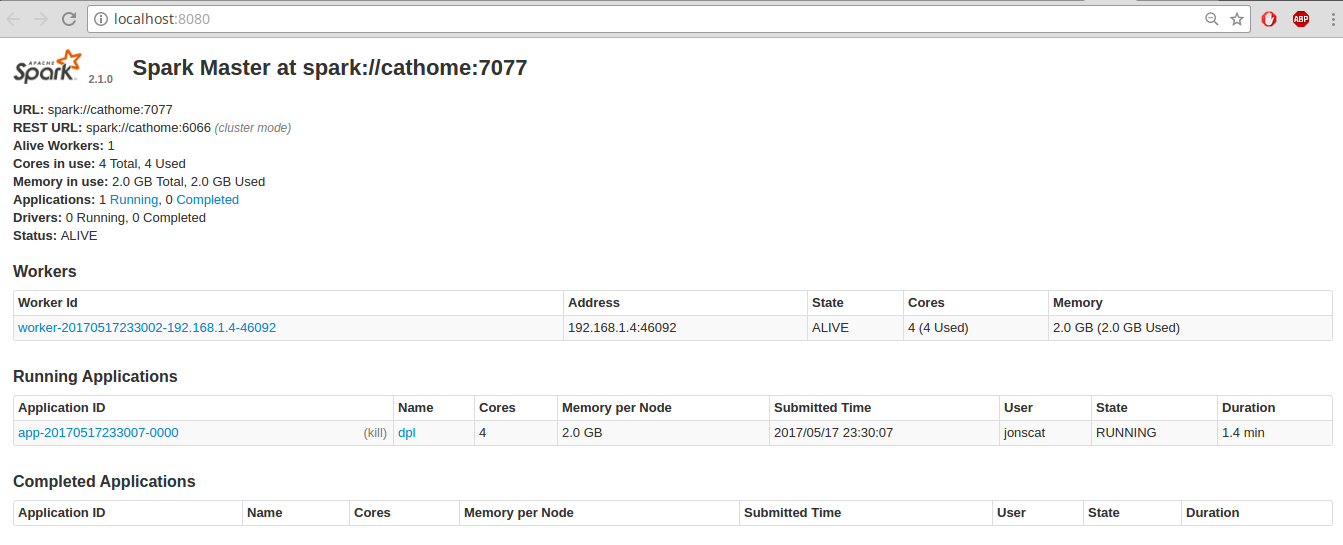


Рис 1. Пример развернутого кластера Spark. Консоль мастера.

АНАЛИЗ ДАННЫХ, ГЕНЕРИРУЕМЫХ ДАТЧИКАМИ ПРОИЗВОДСТВЕННОЙ ЛИНИИ

Исходные данные представляют из себя события прохождения объектов производства по конвейеру или его части с промежуточными результатами, о которых известны измерения в контрольных точках линии производства и представлены они в виде таблицы, строки которой, это информация движения изделия по линии или её части, а столбцы – измерения в конкретной точке производственной линии. Так как строение производственной линии, а также используемые системы и датчики являются конфиденциальной информацией то, данные анонимизированы, то есть невозможно узнать логическое значение измерений датчика и его название, однако для алгоритма эта информация не является важной, также при таком решение задачи (без привязки к конкретному оборудованию) возможен её перенос на другое производство с небольшими модификациями или без них.

Анализ

Исходные признаки разбиты на три группы: числовые, временные и категориальные. Для итоговой модели нужны все 3 группы, однако, изначально можно работать с ними и проводить анализ по отдельности.

* Числовые признаки – это некоторые измерения различных датчиков, например, датчиков скорости или положения.
* Временные признаки – это время прохождения детали участков производственной линии.
* Категориальные признаки – некоторые атрибуты прохождения детали участков производственной линии. Обычно это заранее заданный диапазон дискретных значений ({П1, П2, П3}).

Для построения хорошей модели необходимо провести анализ данных, выявить зависимости, корреляции, возможно добавить какие-то новые признаки для улучшения качества, либо удалить признаки, вносящие только шум.

Для начала необходимо посмотреть на общие статистики обрабатываемых данных, такие как количество объектов, количество признаков, проценты наполненности данных.

* Количество объектов: 1183747
* Количество признаков:
  + Числовые признаки: 970
  + Временные признаки: 1157
  + Категориальные признаки: 2141

Также необходимо посмотреть, что из себя представляют данные «внутри», то есть увидеть описание некоторой выборки данных (объектов). Возьмем выборку в 5% от основного объема объектов:

…

Следующий шаг, это узнать количество пропусков в данных, можно построить следующие диаграммы:

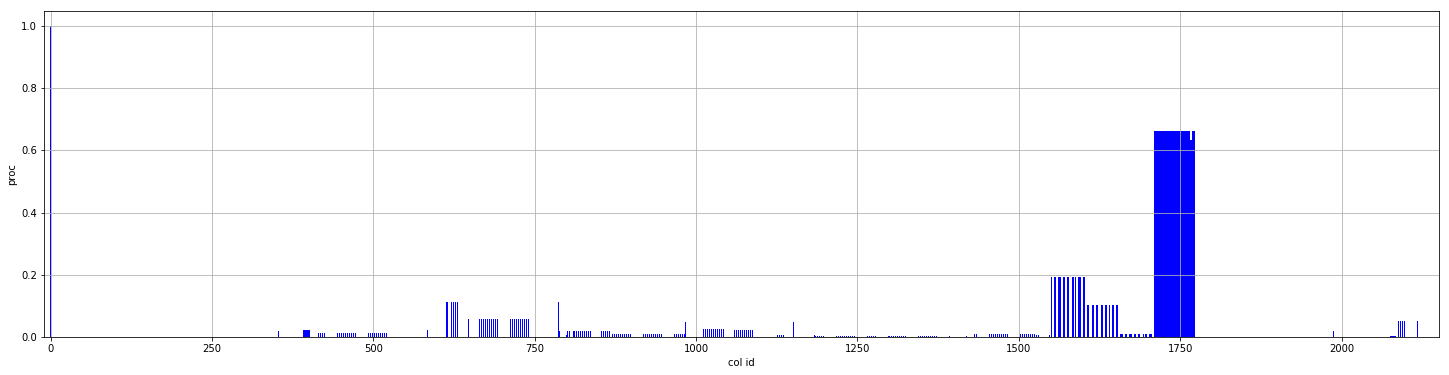


Рис 3. Распределение пропусков в категориальных признаках

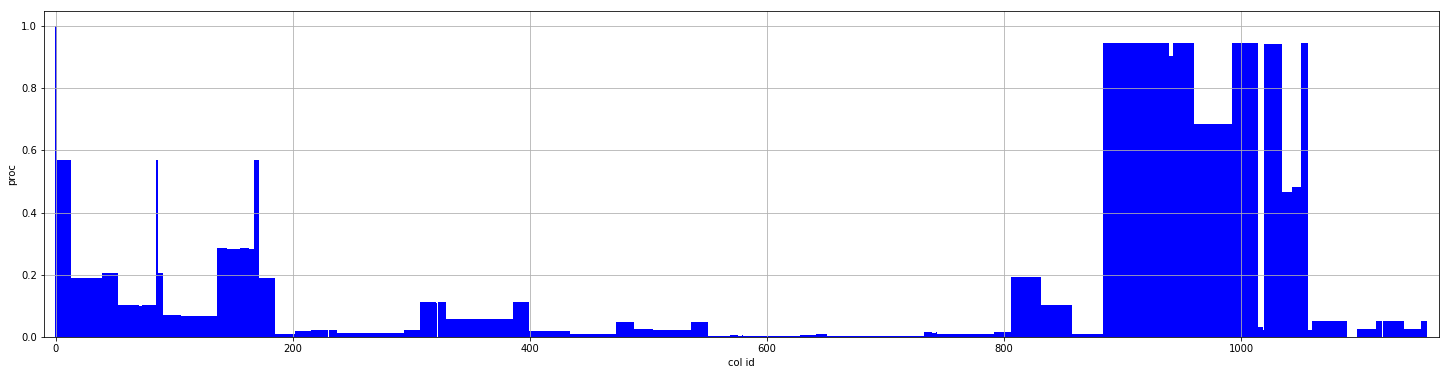


Рис 2. Распределение пропусков в временных признаках

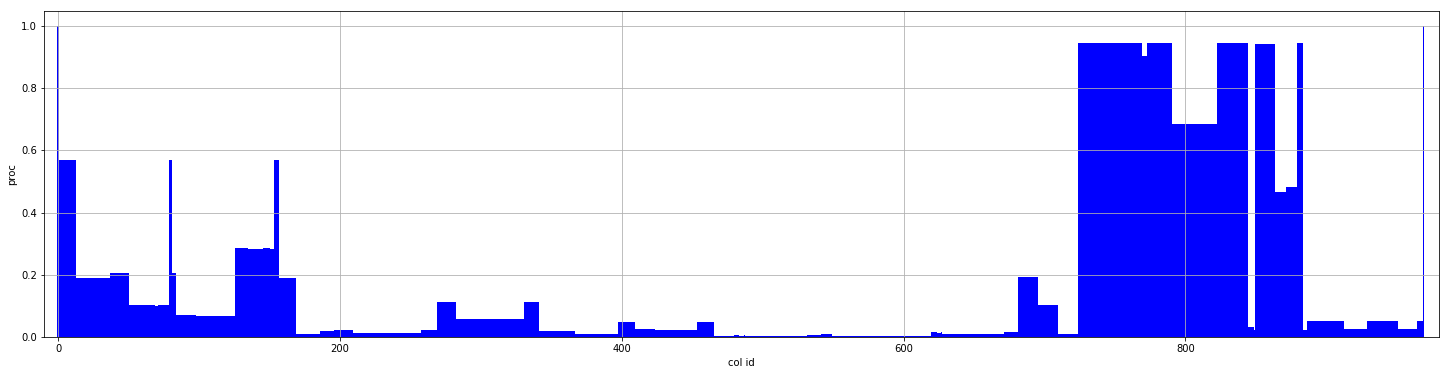
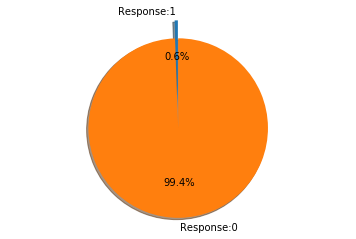


Рис 1. Распределение пропусков в числовых признаках

Так как поставленная задача – это задача бинарной классификации, то необходимо увидеть распределение объектов по классам. Данное распределение представлено на рис. 4.

Рис 4. Распределение классов целевой переменной



Из представленного графика видно, что данные сильно несбалансированные, то есть в них присутствует преобладание одного класса объектов со значением целевой переменной равным ноль. То есть большинство объектов, на линии проходят контроль качества и сбои происходят достаточно редко.

По сохраненным выборкам данных можно построить матрицу корреляций. Так как выборка берется без повторений и из случайных позиций, то такого объема хватит, чтобы увидеть основные паттерны в данных.

Матрица корреляций выборки числовых признаков (см Приложение А).

Матрица корреляций выборки временных признаков (см Приложение А).

Матрицу корреляций категориальных признаков не удалось построить из-за очень большого количества пропусков в данных.

Для алгоритмов машинного обучения сильно коррелирующие признаки –

очень нежелательное явление.

В матрице корреляций временных признаков все признаки внутри полей S#, - сильно коррелирующие, причем коэффициент корреляции ~= 1, то есть эти значения одинаковые, и как говорилось ранее, пользы для алгоритма они не несут, следовательно, их можно полностью исключить из итоговой матрицы обучения, однако есть исключение для полей с «S24», «S25».

Исходя из вышеописанного можно было бы оставить те признаки, корреляция между которыми меньше 0.8 (при корреляции больше 0.8 – переменные считаются сильно коррелирующими). Однако, как видно из получившихся матриц даже для полей с номерами «S24», «S25» значение корреляции 0.99, что очень высоко, но лучше, чем 1, исходя из этого, эти признаки можно оставить.

Так же, исходя из матрицы корреляций числовых признаков, сильно коррелирующие признаки можно исключить и оставить те, значение корреляции между которыми ~=0.

После обработки временных признаков можно посмотреть на числовые признаки. Так как самих признаков очень много, это влечет за собой некоторые проблемы, например, такие как, очень долгое обучение классификатора, а также дополнительные помехи его последующей работе из-за признаков не несущих полезной информации для алгоритма, то есть, вносящих только шум. Для решения данных проблем можно использовать методы отбора признаков. Например, метод «Informaition gain», основанный на корреляции признаков с целевой переменной.

При анализе категориальных признаков были отобраны признаки с наибольшей долей присутствия присутствия значений, так как большое количество пропусков в данных не несет полезной информации для алгоритма и часто заменяется на средние значения, медианы, ноль или очень большие значения (mean, median, 0, 999999999), однако, рассматриваемы признаки – категориальные, следовательно, замена их на среднее или медиану не несет никакого смысла (при числовом кодировании), при замене их на 0 или большое число (9999999) также не будет пользы, эти значения будут вносить только дополнительные корреляции и шум, от которых необходимо избавляться.

Используя предложенные методы были отобраны признаки, имеющие наименьшую корреляцию между собой, а также наивысшую корреляцию с целевой переменной.

Также признаки для итоговой выборки обучения были выбраны исходя из экспериментов с алгоритмом, то есть, при их наличии алгоритм показывал лучшие результаты. При работе с алгоритмами машинного обучения это часто встречающаяся практика, когда какой-то признак критичен для алгоритма и при его наличии результаты улучшаются. Например, в данной работе таким признаком является разница между значениями признака Id, а также инвертированная разница, применяемая для временных признаков.

ПОСТРОЕНИЕ МОДЕЛИ ПРЕДУПРЖДЕНИЯ СБОЕВ НА ПРОИЗВОДСТВЕННОЙ ЛИНИИ

Описание алгоритма

После анализа данных необходимо выбрать конкретный алгоритм машинного обучения для использования в модели. Для задач классификации при обучении с учителем в настоящее время самым распространенным и эффективным методом является градиентный бустинг над деревьями.

Бустинг (англ. boosting — улучшение) — это процедура последовательного построения композиции алгоритмов машинного обучения, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов. Бустинг представляет собой жадный алгоритм построения композиции алгоритмов.

В течение последних 10 лет бустинг остаётся одним из наиболее популярных методов машинного обучения, наряду с нейронными сетями и машинами опорных векторов. Основные причины — простота, универсальность, гибкость (возможность построения различных модификаций), и, главное, высокая обобщающая способность.

Бустинг над решающими деревьями считается одним из наиболее эффективных методов с точки зрения качества классификации. Во многих экспериментах наблюдалось практически неограниченное уменьшение частоты ошибок на независимой тестовой выборке по мере наращивания композиции. Более того, качество на тестовой выборке часто продолжало улучшаться даже после достижения безошибочного распознавания всей обучающей выборки. Это перевернуло существовавшие долгое время представления о том, что для повышения обобщающей способности необходимо ограничивать сложность алгоритмов. На примере бустинга стало понятно, что хорошим качеством могут обладать сколь угодно сложные композиции, если их правильно настраивать.

Впоследствии феномен бустинга получил теоретическое обоснование. Оказалось, что взвешенное голосование не увеличивает эффективную сложность алгоритма, а лишь сглаживает ответы базовых алгоритмов. Количественные оценки обобщающей способности бустинга формулируются в терминах отступа. Эффективность бустинга объясняется тем, что по мере добавления базовых алгоритмов увеличиваются отступы обучающих объектов. Причём бустинг продолжает раздвигать классы даже после достижения безошибочной классификации обучающей выборки.

Сам алгоритм можно представить в виде:

, где , где – матрица признаков, , где – целевой вектор,

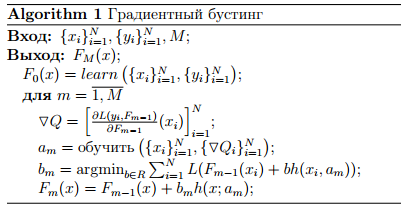


Рис. 1 Градиентный бустинг

– классификатор, – функционал ошибки, – параметры алгоритма.

Для построения модели использовалась готовая реализация алгоритма в свободной библиотеке xgboost. Библиотека xgboost представляет модифицированные, распределенные и оптимизированные алгоритмы градиентного бустинга для классификации и регрессии, а также дополнительные возможности и утилиты для настройки алгоритмов.

Запуск обучения алгоритма производился со следующими параметрами:

* objective: "binary:logistic"
* eta: 0.021
* max\_depth: 7
* colsample\_bytree: 0.82
* min\_child\_weight: 3
* base\_score: 0.005

objective – ыва

eta – выа

max\_depth – ар

colsample\_bytree – ывы

min\_child\_weight –ыв

base\_score – ыва

Поиск оптимальных параметров проводился вручную, учитывая специфику задачи и объем данных.

Так же для сравнения качества алгоритма был взят ещё один популярный и достаточно мощный алгоритм машинного обучения – RandomForest. Его реализация была взята из python библиотеки для машинного обучения sklearn.

Перед обучением модели полученная после обработки данных обучающая выборка была разделена на части в соотношении 70/30. Данные из 70% части использовались для обучения алгоритма и в них была представлена целевая переменная, тогда как данные из 30% части были использованы для проверки и оценки качества работы алгоритма, с помощью кросс-валидации.

У алгоритма бустинга из библиотеки xgboost есть возможность производить кросс-валидацию собственными средствами. Для алгоритма Random Forest были использованы дополнительные функции библиотеки sklearn.

Результаты работы алгоритмов представлены в таблицах 1 и 2.

Так же использовалось несколько экспериментов полного цикла (то есть с созданием нового классификатора) и итоговые результаты усреднялись. В таблицах 1 и 2 представлены результаты последнего эксперимента (folds = 5).

Оценка качества модели

Оценка качества и выбор метрики качества модели – очень важный аспект построения системы, основанной на методах машинного обучения.

Перед переходом к самим метрикам необходимо ввести важную концепцию для их описания в терминах ошибок классификации — confusion matrix (матрица ошибок).

Допустим, что у нас есть два класса и алгоритм, предсказывающий принадлежность каждого объекта одному из классов, тогда матрица ошибок классификации будет выглядеть следующим образом (см. Таб. 1).

|  |  |  |
| --- | --- | --- |
|  | *y* = 1 | *y* = 0 |
| *a(x)* = 1 | True Positive (TP) | False Positive (FP) |
| *a(x)* = 0 | False Negative (FN) | True Negative (TN) |

Таблица 1. Матрица ошибок.

Где *a(x)* - это ответ алгоритма на объекте, а *y* — истинная метка класса на этом объекте. Таким образом, ошибки классификации бывают двух видов: False Negative (FN) и False Positive (FP).

Наиболее очевидной мерой качества в задаче классификации является доля правильных ответов (accuracy) (см Рис. 5).

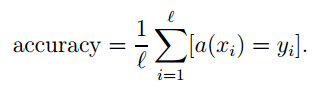


Рис. 1. Доля правильных ответов

Данная метрика, однако, имеет существенный недостаток. Если взять порог t меньше минимального значения прогноза *b(x)* на выборке или больше максимального значения, то доля правильных ответов будет равна доле положительных и отрицательных ответов соответственно. Таким образом, если в выборке 950 отрицательных и 50 положительных объектов, то при тривиальном пороге *t* = *maxi* *b(xi)* мы получим долю правильных ответов 0.95. Это означает, что доля положительных ответов сама по себе не несет никакой информации о качестве работы алгоритма *a(x)*, и вместе с ней следует анализировать соотношение классов в выборке. Также полезно вместе с долей правильных ответов вычислять базовую долю — долю правильных ответов алгоритма, всегда выдающего наиболее мощный класс.

Доля правильных ответов, выраженная через матрицу ошибок, представляется следующим образом (Рис. 2).

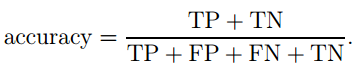


Рис. 2. Представление доли правильных ответов через матрицу ошибок

Гораздо более информативными критериями являются точность (precision) (Рис. 3) и полнота (recall) (Рис. 4).

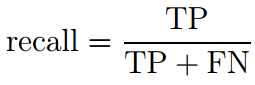


Рис. 4. Полнота

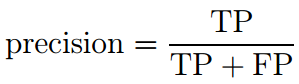


Рис. 3. Точность.

Точность показывает, какая доля объектов, выделенных классификатором как положительные, действительно является положительными. Полнота показывает, какая часть положительных объектов была выделена классификатором.

Так же важно, что точность и полнота не зависят от соотношения размеров классов. Даже если объектов положительного класса на порядки меньше, чем объектов отрицательного класса, данные показатели будут корректно отражать качество работы алгоритма. Существует несколько способов получить один критерий качества на основе точности и полноты. Один из них — *F*-мера, гармоническое среднее точности и полноты (Рис. 5).

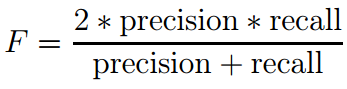


Рис. 5. F – мера.

Среднее гармоническое обладает важным свойством — оно близко к нулю, если хотя бы один из аргументов близок к нулю. Именно поэтому оно является более предпочтительным, чем среднее арифметическое (если алгоритм будет относить все объекты к положительному классу, то он будет иметь recall = 1 и precision ≪ 1, а их среднее арифметическое будет больше 1/2, что недопустимо).

Другой способ - «Matthews correlation coefficient» (Рис. 6), разработанный биохимиком Brian W. Matthews в 1975 году, часто используется, если классы сильно не сбалансированы.

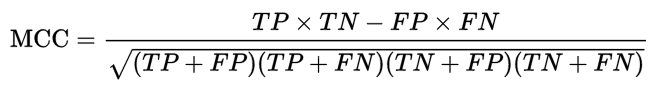


Рис. 6. Matthews correlation coefficient

Для нахождения лучшего порога для MCC, а также для оценки качества работы алгоритма, была использована функция mcc\_eval (см Приложение Б), которая может быть передана классификаторам xgboost и Random Forest для оценки для оценки качества моделей.

Так же для не сбалансированных классов очень хорошо подходит метрика «Precision-Recall кривая», на осях которой откладываются полнота и точность. Хорошая практика — это использовать несколько метрик для оценивания качества модели, так как это дает более полную картину. Если показания метрик близки, при некотором нормировании, то это говорит об адекватной оценке, если они кардинально расходятся, следовательно, либо модель показывает плохие результаты, либо метрика выбрана некорректно и не подходит для данной задачи.

Получившиеся результаты работы моделей.

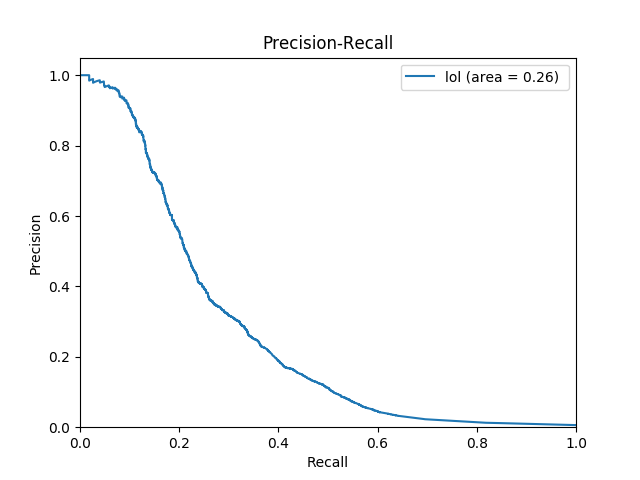


Рис 8. График Precision-Recall кривой

Получившаяся модель представлена в виде функций обработки входных данных и обученного классификатора. Предполагается, что на вход алгоритму будут представлены данные такого же формата, как и исходный формат обучающей выборки. При поступлении новых данных к ним применяются те же преобразования, что были применены к исходным данным для анализа, далее получившийся результат обработки подается на вход классификатору, который, в свою очередь, предсказывает итоговое значение целевой переменной.

Встраивание модели в систему производственной линии сильно зависит от используемого программного обеспечения и архитектуры системы и должно выполняться индивидуально с учетом вышеуказанного.

ЗАКЛЮЧЕНИЕ

В данной работе была разработана модель машинного обучения для предотвращения сбоев на производственной линии. Решены поставленные задачи: был проведен анализ данных, построены модели машинного обучения, исцеловано их качество, а также проведено сравнение результатов работы моделей при использовании различных алгоритмов машинного обучения.

Внедрение построенной модели на производстве позволит предотвратить сбои на производственной линии и, как следствие, повысить производительность производства, сократить финансовые издержки и удешевить продукт для конечного пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Davis J., Goadrich M. (2006). The Relationship Between Precision-Recall and ROC Curves. // Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA.

[2] Chawla N., Bowyer K., Hall L., Kegelmeyer W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. // Journal of Artificial Intelligence Research, Vol. 16, Pp. 321–357.

# 

# ПРИЛОЖЕНИЕ А

# ПРИЛОЖЕНИЕ Б

@jit  
**def** mcc(tp, tn, fp, fn):  
 sup = tp \* tn - fp \* fn  
 inf = (tp + fp) \* (tp + fn) \* (tn + fp) \* (tn + fn)  
 **if** inf == 0:  
 **return** 0  
 **else**:  
 **return** sup / np.sqrt(inf)  
  
  
@jit  
**def** eval\_mcc(y\_true, y\_prob, show=**False**):  
 idx = np.argsort(y\_prob)  
 y\_true\_sort = y\_true[idx]  
 n = y\_true.shape[0]  
 nump = 1.0 \* np.sum(y\_true) *# number of positive* numn = n - nump *# number of negative* tp = nump  
 tn = 0.0  
 fp = numn  
 fn = 0.0  
 best\_mcc = 0.0  
 best\_id = -1  
 mccs = np.zeros(n)  
 **for** i **in** range(n):  
 **if** y\_true\_sort[i] == 1:  
 tp -= 1.0  
 fn += 1.0  
 **else**:  
 fp -= 1.0  
 tn += 1.0  
 new\_mcc = mcc(tp, tn, fp, fn)  
 mccs[i] = new\_mcc  
 **if** new\_mcc >= best\_mcc:  
 best\_mcc = new\_mcc  
 best\_id = i  
 **if** show:  
 best\_proba = y\_prob[idx[best\_id]]  
 y\_pred = (y\_prob > best\_proba).astype(int)  
 plt.interactive(**False**)  
 plt.plot(mccs)  
 plt.show()  
 **return** best\_proba, best\_mcc, y\_pred  
 **else**:  
 **return** best\_mcc  
  
  
**def** mcc\_eval(y\_prob, dtrain):  
 **if** isinstance(dtrain, np.ndarray):  
 y\_true = dtrain  
 best\_mcc = eval\_mcc(y\_true, y\_prob)  
 **return** best\_mcc  
 **else**:  
 y\_true = dtrain.get\_label()  
 best\_mcc = eval\_mcc(y\_true, y\_prob)  
 **return 'MCC'**, best\_mcc

# ПРИЛОЖЕНИЕ В

*#Enviroment preparation***from** pyspark **import** SparkContext, SparkConf  
**from** pyspark.sql **import** SQLContext, Row  
**from** pyspark.sql.types **import** \*  
**import** pyspark.sql.functions **as** F  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** random  
**import** time **as** t  
**import** os  
**import** re  
**import** gc  
**from** subprocess **import** Popen  
**import** matplotlib.pyplot **as** plt  
*#% matplotlib inline***def** ControlSparkCluster(action):  
 **if** action == **'start'**:  
 Popen(**'./start\_cluster.sh'**, shell=**True**)  
 t.sleep(10)  
 **print 'cluster started'  
 elif** action == **'stop'**:  
 Popen(**'./stop\_cluster.sh'**, shell=**True**)  
 sc.stop()  
 t.sleep(3)  
 **print 'cluster stopped'  
 else**:  
 **raise** Exception(**'Unsupported command!'**)  
  
ControlSparkCluster(**'stop'**)  
ControlSparkCluster(**'start'**)  
*# cluster started*conf = SparkConf().setMaster(**"spark://cathome:7077"**).setAppName(**"dpl"**) \  
 .setAll(([**'spark.executor.cores'**, **'4'**]  
 , [**'spark.driver.cores'**, **'4'**]  
 , [**'spark.driver.memory'**, **'1g'**]  
 , [**'spark.driver.maxResultSize'**, **'4g'**]  
 , [**'spark.executor.memory'**, **'2g'**]  
 , [**'spark.python.worker.memory'**, **'1g'**]  
 , [**'spark.default.parallelism'**, **'4'**]  
 ))  
  
sc = SparkContext(conf=conf)  
sqlContext = SQLContext(sc)  
*#Data preparation*dir\_name = **'../../../diploma\_data'**tr\_num\_file\_name = **'train\_numeric.csv'**tr\_date\_file\_name = **'train\_date.csv'**tr\_cat\_file\_name = **'train\_categorical.csv'**tr\_num\_data\_file = **'{0}/{1}'**.format(dir\_name, tr\_num\_file\_name)  
tr\_date\_data\_file = **'{0}/{1}'**.format(dir\_name, tr\_date\_file\_name)  
tr\_cat\_data\_file = **'{0}/{1}'**.format(dir\_name, tr\_cat\_file\_name)  
  
*#%% time*df\_num = sqlContext.read.csv(tr\_num\_data\_file, sep=**','**, header=**True**, inferSchema=**True**)  
df\_date = sqlContext.read.csv(tr\_date\_data\_file, sep=**','**, header=**True**, inferSchema=**True**)  
df\_cat = sqlContext.read.csv(tr\_cat\_data\_file, sep=**','**, header=**True**, inferSchema=**True**)  
*#Data analisys / Data transformation*df\_num2 = df\_num  
df\_date2 = df\_date  
df\_cat2 = df\_cat  
  
num\_sample\_file = **'{0}/{1}'**.format(dir\_name, **'num\_sample.parquet'**)  
date\_sample\_file = **'{0}/{1}'**.format(dir\_name, **'date\_sample.parquet'**)  
cat\_sample\_file = **'{0}/{1}'**.format(dir\_name, **'cat\_sample.parquet'**)  
  
*# df\_num.sample(False, 0.02, seed=112).repartition(1024).write.parquet(num\_sample\_file)  
# df\_date.sample(False, 0.02, seed=112).repartition(1024).write.parquet(date\_sample\_file)  
# df\_cat.sample(False, 0.02, seed=112).repartition(4096).write.parquet(cat\_sample\_file)  
  
#%% time*df\_num\_smpl = sqlContext.read.parquet(num\_sample\_file)  
df\_date\_smpl = sqlContext.read.parquet(date\_sample\_file)  
  
*# CPU times: user 4 ms, sys: 4 ms, total: 8 ms  
# Wall time: 1.07 s*df\_num2.limit(10).toPandas()  
  
df\_date2.limit(10).toPandas()  
  
df\_cat2.limit(10).toPandas()  
  
**print 'numeric dataset row count: '**, df\_num2.count()  
**print 'date dataset row count: '**, df\_date2.count()  
**print 'categorical dataset row count: '**, df\_cat2.count()  
  
*#numeric dataset row count: 1183747  
#date dataset row count: 1183747  
#categorical dataset row count: 1183747  
  
# df\_num2.printSchema()  
# df\_date2.printSchema()  
# df\_cat2.printSchema()*df\_num2.describe(df\_num2.columns[1:5]).show()  
*#+-------+--------------------+--------------------+--------------------+--------------------+  
#| summary | L0\_S0\_F0 | L0\_S0\_F2 | L0\_S0\_F4 | L0\_S0\_F6 |  
#+-------+--------------------+--------------------+--------------------+--------------------+  
#| count | 673862 | 673862 | 673862 | 673862 |  
#| mean | 5.456013249000960... | 9.128575286932377E-5 | 4.005271108873819... | 1.383517693495106... |  
#| stddev | 0.08021442344615412 | 0.09279893101577752 | 0.2115196284424472 | 0.2116351123015797 |  
#| min | -0.565 | -0.616 | -0.415 | -0.416 |  
#| max | 0.278 | 0.302 | 0.585 | 0.584 |  
#+-------+--------------------+--------------------+--------------------+--------------------+*df\_date2.describe(df\_date2.columns[1:5]).show()  
*#+-------+------------------+------------------+------------------+------------------+  
#| summary | L0\_S0\_D1 | L0\_S0\_D3 | L0\_S0\_D5 | L0\_S0\_D7 |  
#+-------+------------------+------------------+------------------+------------------+  
#| count | 673862 | 673862 | 673862 | 673862 |  
#| mean | 882.2290677468101 | 882.2290677468101 | 882.2290677468101 | 882.2290677468101 |  
#| stddev | 506.72491575187627 | 506.72491575187627 | 506.72491575187627 | 506.72491575187627 |  
#| min | 0.0 | 0.0 | 0.0 | 0.0 |  
#| max | 1713.71 | 1713.71 | 1713.71 | 1713.71 |  
#+-------+------------------+------------------+------------------+------------------+*df\_cat2.describe(df\_cat2.columns[1:5]).show()  
*#+-------+---------+---------+---------+---------+  
#| summary | L0\_S1\_F25 | L0\_S1\_F27 | L0\_S1\_F29 | L0\_S1\_F31 |  
#+-------+---------+---------+---------+---------+  
#| count | 25 | 25 | 27 | 27 |  
#| mean | null | null | null | null |  
#| stddev | null | null | null | null |  
#| min | T1 | T9 | T1 | T24 |  
#| max | T1 | T9 | T1 | T9 |  
#+-------+---------+---------+---------+---------+*cols\_num = [(c.split(**'\_'**)) **for** c **in** df\_num2.columns]  
cols\_num\_df = pd.DataFrame(cols\_num[1:-1], columns=[**'line'**, **'station'**, **'feature'**])  
cols\_num\_df  
cols\_num\_df\_g = cols\_num\_df.groupby([**'line'**, **'station'**], sort=**False**).count()  
cols\_num\_df\_g  
  
cols\_date = [(c.split(**'\_'**)) **for** c **in** df\_date2.columns]  
cols\_date\_df = pd.DataFrame(cols\_date[1:], columns=[**'line'**, **'station'**, **'feature'**])  
cols\_date\_df  
cols\_date\_df\_g = cols\_date\_df.groupby([**'line'**, **'station'**], sort=**False**).count()  
cols\_date\_df\_g  
  
cols\_cat = [(c.split(**'\_'**)) **for** c **in** df\_cat2.columns]  
cols\_cat\_df = pd.DataFrame(cols\_cat[1:], columns=[**'line'**, **'station'**, **'feature'**])  
cols\_cat\_df  
cols\_cat\_df\_g = cols\_cat\_df.groupby([**'line'**, **'station'**], sort=**False**).count()  
cols\_cat\_df\_g  
  
  
**def** plt\_hist(counts\_by\_col\_d):  
 hist\_v = map(float, [v **for** k, v **in** sorted(counts\_by\_col\_d.items())])  
 elems = list(xrange(0, len(hist\_v)))  
 plt.figure(figsize=(25, 6))  
 plt.ylabel(**'proc'**)  
 plt.xlabel(**'col id'**)  
 plt.grid(**True**)  
 plt.xlim([-10, len(elems) + 10])  
 width = 1 / 1  
 *# plt.xticks(elems, row\_fileds)* plt.bar(elems, hist\_v, width, color=**"blue"**)  
  
  
**def** plt\_pipe(counts\_1\_all\_l):  
 labels = **'Response:1'**, **'Response:0'** sizes = (counts\_1\_all\_l[**'count\_1'**], counts\_1\_all\_l[**'count\_all'**])  
 explode = (0.2, 0)  
  
 fig1, ax1 = plt.subplots()  
 ax1.pie(sizes, explode=explode, labels=labels, autopct=**'%1.1f%%'**,  
 shadow=**True**, startangle=90)  
 ax1.axis(**'equal'**)  
  
  
counts\_1\_all\_l = df\_num2.select(F.sum(F.when(df\_num2[**'Response'**] == 1, 1).otherwise(0)).alias(**'count\_1'**),  
 F.count(**'\*'**).alias(**'count\_all'**)).collect()  
  
plt\_pipe(counts\_1\_all\_l[0].asDict())  
  
counts\_by\_col = df\_num2.select(\*[(F.count(c) / F.count(**"\*"**)).alias(c) **for** c **in** df\_num2.columns])  
  
counts\_by\_col.limit(1).toPandas()  
  
counts\_by\_col\_l = counts\_by\_col.collect()  
  
plt\_hist(counts\_by\_col\_l[0].asDict())  
  
col\_with\_resp = df\_num2.filter(df\_num2.Response == 1).select(  
 \*[(F.count(c) / F.count(**"\*"**)).alias(c) **for** c **in** df\_num2.columns])  
  
col\_with\_resp.limit(100).toPandas()  
  
col\_with\_resp\_l = col\_with\_resp.collect()  
  
plt\_hist(col\_with\_resp\_l[0].asDict())  
  
counts\_by\_col3 = df\_date2.select(\*[(F.count(c) / F.count(**"\*"**)).alias(c) **for** c **in** df\_date2.columns])  
  
counts\_by\_col3.limit(1).toPandas()  
  
counts\_by\_col\_l3 = counts\_by\_col3.collect()  
  
plt\_hist(counts\_by\_col\_l3[0].asDict())  
  
counts\_by\_col4 = df\_cat2.select(\*[(F.count(c) / F.count(**"\*"**)).alias(c) **for** c **in** df\_cat2.columns])  
  
counts\_by\_col\_l4 = counts\_by\_col4.collect()  
  
plt\_hist(counts\_by\_col\_l4[0].asDict())  
  
  
  
pdf\_date = df\_date\_smpl.toPandas()  
  
cols\_date = [(c.split(**'\_'**)) **for** c **in** pdf\_date.columns]  
cols\_date\_df = pd.DataFrame(cols\_date[1:], columns=[**'line'**,**'station'**,**'feature'**])  
cols\_date\_df[**'full\_name'**] = cols\_date\_df.line + **'\_'** + cols\_date\_df.station + **'\_'** + cols\_date\_df.feature  
cols\_date\_df = cols\_date\_df.drop(**'line'**,1)  
**for** i **in** list(cols\_date\_df.station.unique()):  
 station\_features = list(cols\_date\_df.loc[cols\_date\_df[**'station'**] == i].full\_name.values)  
 pdf\_date[station\_features].corr(method=**'pearson'**)  
  
*# L0\_S0\_D1 L0\_S0\_D3 L0\_S0\_D5 L0\_S0\_D7 L0\_S0\_D9 L0\_S0\_D11 L0\_S0\_D13 L0\_S0\_D15 L0\_S0\_D17 L0\_S0\_D19 L0\_S0\_D21 L0\_S0\_D23  
#L0\_S0\_D1 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D3 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D5 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D7 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D11 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D13 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D15 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D17 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D19 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D21 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S0\_D23 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#Out[79]:  
#L0\_S1\_D26 L0\_S1\_D30  
#L0\_S1\_D26 1.0 1.0  
#L0\_S1\_D30 1.0 1.0  
#Out[79]:  
#L0\_S2\_D34 L0\_S2\_D38 L0\_S2\_D42 L0\_S2\_D46 L0\_S2\_D50 L0\_S2\_D54 L0\_S2\_D58 L0\_S2\_D62 L0\_S2\_D66  
#L0\_S2\_D34 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D38 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D42 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D46 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D50 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D54 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D58 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D62 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
#L0\_S2\_D66 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0  
# ....  
  
# date df*pdf\_num = df\_mun\_smpl.toPandas()  
  
cols\_date = [(c.split(**'\_'**)) **for** c **in** pdf\_num.columns]  
cols\_date\_df = pd.DataFrame(cols\_date[1:], columns=[**'line'**,**'station'**,**'feature'**])  
cols\_date\_df[**'full\_name'**] = cols\_date\_df.line + **'\_'** + cols\_date\_df.station + **'\_'** + cols\_date\_df.feature  
cols\_date\_df = cols\_date\_df.drop(**'line'**,1)  
**for** i **in** list(cols\_date\_df.station.unique()):  
 station\_features = list(cols\_date\_df.loc[cols\_date\_df[**'station'**] == i].full\_name.values)  
 pdf\_num[station\_features].corr(method=**'pearson'**)  
  
*# num df*useful\_values = []  
cols\_date = [(c.split(**'\_'**)) **for** c **in** pdf\_num.columns]  
cols\_date\_df = pd.DataFrame(cols\_date[1:], columns=[**'line'**,**'station'**,**'feature'**])  
cols\_date\_df[**'full\_name'**] = cols\_date\_df.line + **'\_'** + cols\_date\_df.station + **'\_'** + cols\_date\_df.feature  
cols\_date\_df = cols\_date\_df.drop(**'line'**,1)  
**for** i **in** list(cols\_date\_df.station.unique()):  
 station\_features = list(cols\_date\_df.loc[cols\_date\_df[**'station'**] == i].full\_name.values)  
 corr\_df = pd.DataFrame(pdf\_num[station\_features].corr(method=**'pearson'**))  
 corr\_df  
 **try**:  
 min\_value = abs(corr\_df.loc[:, station\_features]).min(axis=1).idxmax(axis=0)  
 **except**:  
 min\_value = **None  
 if** min\_value:  
 useful\_values.append(min\_value)  
useful\_values  
  
  
*# L0\_S0\_F0 L0\_S0\_F2 L0\_S0\_F4 L0\_S0\_F6 L0\_S0\_F8 L0\_S0\_F10 L0\_S0\_F12 L0\_S0\_F14 L0\_S0\_F16 L0\_S0\_F18 L0\_S0\_F20 L0\_S0\_F22  
#L0\_S0\_F0 1.000000 0.889585 -0.024860 -0.023693 -0.021955 -0.016040 -0.778816 -0.699263 0.151366 0.206989 0.153013 0.153011  
#L0\_S0\_F2 0.889585 1.000000 -0.023703 -0.027373 -0.042980 0.012173 -0.689760 -0.778402 0.223214 0.176426 0.199401 0.199399  
#L0\_S0\_F4 -0.024860 -0.023703 1.000000 0.992738 -0.028634 0.049657 0.551105 0.552818 0.056977 0.046345 -0.013667 -0.013668  
#L0\_S0\_F6 -0.023693 -0.027373 0.992738 1.000000 -0.029508 0.049524 0.554599 0.551547 0.057959 0.044261 -0.012808 -0.012810  
#L0\_S0\_F8 -0.021955 -0.042980 -0.028634 -0.029508 1.000000 0.516884 -0.038238 -0.021610 0.042460 0.080503 -0.050112 -0.050098  
#L0\_S0\_F10 -0.016040 0.012173 0.049657 0.049524 0.516884 1.000000 0.038773 0.016819 0.220418 0.270935 -0.000579 -0.000578  
#L0\_S0\_F12 -0.778816 -0.689760 0.551105 0.554599 -0.038238 0.038773 1.000000 0.914613 -0.098344 -0.158758 -0.109484 -0.109484  
#L0\_S0\_F14 -0.699263 -0.778402 0.552818 0.551547 -0.021610 0.016819 0.914613 1.000000 -0.161912 -0.138369 -0.140083 -0.140083  
#L0\_S0\_F16 0.151366 0.223214 0.056977 0.057959 0.042460 0.220418 -0.098344 -0.161912 1.000000 0.607189 0.018838 0.018839  
#L0\_S0\_F18 0.206989 0.176426 0.046345 0.044261 0.080503 0.270935 -0.158758 -0.138369 0.607189 1.000000 0.023059 0.023059  
#L0\_S0\_F20 0.153013 0.199401 -0.013667 -0.012808 -0.050112 -0.000579 -0.109484 -0.140083 0.018838 0.023059 1.000000 1.000000  
#L0\_S0\_F22 0.153011 0.199399 -0.013668 -0.012810 -0.050098 -0.000578 -0.109484 -0.140083 0.018839 0.023059 1.000000 1.000000  
#Out[106]:  
#L0\_S1\_F24 L0\_S1\_F28  
##L0\_S1\_F24 1.00000 -0.26028  
#L0\_S1\_F28 -0.26028 1.00000  
#Out[106]:  
#L0\_S2\_F32 L0\_S2\_F36 L0\_S2\_F40 L0\_S2\_F44 L0\_S2\_F48 L0\_S2\_F52 L0\_S2\_F56 L0\_S2\_F60 L0\_S2\_F64  
#L0\_S2\_F32 1.000000 -0.336864 0.755147 -0.146511 0.294490 NaN 0.694501 0.032475 -0.114641  
#L0\_S2\_F36 -0.336864 1.000000 0.022725 0.017860 -0.005609 NaN 0.039810 -0.016953 0.017920  
#L0\_S2\_F40 0.755147 0.022725 1.000000 -0.600806 0.168338 NaN 0.595915 0.019304 -0.276755  
#L0\_S2\_F44 -0.146511 0.017860 -0.600806 1.000000 0.065475 NaN -0.102562 -0.023918 0.187910  
#L0\_S2\_F48 0.294490 -0.005609 0.168338 0.065475 1.000000 NaN 0.211209 0.058574 0.005544  
#L0\_S2\_F52 NaN NaN NaN NaN NaN NaN NaN NaN NaN  
#L0\_S2\_F56 0.694501 0.039810 0.595915 -0.102562 0.211209 NaN 1.000000 -0.584736 -0.020423  
#L0\_S2\_F60 0.032475 -0.016953 0.019304 -0.023918 0.058574 NaN -0.584736 1.000000 -0.002484  
#L0\_S2\_F64 -0.114641 0.017920 -0.276755 0.187910 0.005544 NaN -0.020423 -0.002484 1.000000*

**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.metrics **import** average\_precision\_score  
**from** sklearn.metrics **import** precision\_recall\_curve  
**from** sklearn.metrics.scorer **import** make\_scorer  
**from** sklearn.metrics **import** matthews\_corrcoef  
**from** operator **import** itemgetter  
**import** matplotlib.pyplot **as** plt  
**from** numba **import** jit  
**import** xgboost **as** xgb  
**import** pandas **as** pd  
**import** numpy **as** np  
**import** datetime  
**import** gc  
  
DIRECTORY = **'../../../diploma\_data/'**TRAIN\_FILES = [**'train\_categorical.csv'**, **'train\_date.csv'**, **'train\_numeric.csv'**]  
  
  
**def** get\_date\_features():  
 train\_file = TRAIN\_FILES[1]  
 **for** i, chunk **in** enumerate(pd.read\_csv(DIRECTORY + train\_file, chunksize=1, low\_memory=**False**)):  
 features = list(chunk.columns)  
 **break** seen = np.zeros(52)  
 rv = []  
 **for** f **in** features:  
 **if** f == **'Id' or 'S24' in** f **or 'S25' in** f:  
 rv.append(f)  
 **continue** station = int(f.split(**'\_'**)[1][1:])  
 **if** seen[station]:  
 **continue** seen[station] = 1  
 rv.append(f)  
 **return** rv  
  
  
useful\_date\_features = get\_date\_features()  
  
  
**def** get\_min\_max\_date():  
 train\_file = TRAIN\_FILES[1]  
  
 features = **None** subset = **None  
  
 for** i, chunk **in** enumerate(pd.read\_csv(DIRECTORY + train\_file, usecols=useful\_date\_features, chunksize=50000,  
 low\_memory=**False**)):  
 **print** i  
 **if** features **is None**:  
 features = list(chunk.columns)  
 features.remove(**'Id'**)  
 df\_date\_chunk = chunk[[**'Id'**]].copy()  
 df\_date\_chunk[**'mindate'**] = chunk[features].min(axis=1).values  
 df\_date\_chunk[**'maxdate'**] = chunk[features].max(axis=1).values  
  
 **if** subset **is None**:  
 subset = df\_date\_chunk.copy()  
 **else**:  
 subset = pd.concat([subset, df\_date\_chunk])  
 **del** chunk  
 gc.collect()  
  
 **return** subset  
  
  
**def** get\_min\_max\_date\_df():  
 df\_date = get\_min\_max\_date()  
  
 df\_date.sort\_values(by=[**'mindate'**, **'Id'**], inplace=**True**)  
 df\_date[**'mindate\_id\_diff'**] = df\_date.Id.diff()  
 dt\_id\_reverse = np.full\_like(df\_date.mindate\_id\_diff.values, np.nan)  
 dt\_id\_reverse[0:-1] = -df\_date.mindate\_id\_diff.values[1:]  
 df\_date[**'mindate\_id\_diff\_reverse'**] = dt\_id\_reverse  
  
 df\_date.sort\_values(by=[**'maxdate'**, **'Id'**], inplace=**True**)  
 df\_date[**'maxdate\_id\_diff'**] = df\_date.Id.diff()  
 dt\_id\_reverse = np.full\_like(df\_date.maxdate\_id\_diff.values, np.nan)  
 dt\_id\_reverse[0:-1] = -df\_date.maxdate\_id\_diff.values[1:]  
 df\_date[**'maxdate\_id\_diff\_reverse'**] = dt\_id\_reverse  
  
 **return** df\_date  
  
prepared\_df\_date = get\_min\_max\_date\_df()  
  
@jit  
**def** mcc(tp, tn, fp, fn):  
 sup = tp \* tn - fp \* fn  
 inf = (tp + fp) \* (tp + fn) \* (tn + fp) \* (tn + fn)  
 **if** inf == 0:  
 **return** 0  
 **else**:  
 **return** sup / np.sqrt(inf)  
  
  
@jit  
**def** eval\_mcc(y\_true, y\_prob, show=**False**):  
 idx = np.argsort(y\_prob)  
 y\_true\_sort = y\_true[idx]  
 n = y\_true.shape[0]  
 nump = 1.0 \* np.sum(y\_true) *# number of positive* numn = n - nump *# number of negative* tp = nump  
 tn = 0.0  
 fp = numn  
 fn = 0.0  
 best\_mcc = 0.0  
 best\_id = -1  
 mccs = np.zeros(n)  
 **for** i **in** range(n):  
 **if** y\_true\_sort[i] == 1:  
 tp -= 1.0  
 fn += 1.0  
 **else**:  
 fp -= 1.0  
 tn += 1.0  
 new\_mcc = mcc(tp, tn, fp, fn)  
 mccs[i] = new\_mcc  
 **if** new\_mcc >= best\_mcc:  
 best\_mcc = new\_mcc  
 best\_id = i  
 **if** show:  
 best\_proba = y\_prob[idx[best\_id]]  
 y\_pred = (y\_prob > best\_proba).astype(int)  
 plt.interactive(**False**)  
 plt.plot(mccs)  
 plt.show()  
 **return** best\_proba, best\_mcc, y\_pred  
 **else**:  
 **return** best\_mcc  
  
  
**def** mcc\_eval(y\_prob, dtrain):  
 **if** isinstance(dtrain, np.ndarray):  
 y\_true = dtrain  
 best\_mcc = eval\_mcc(y\_true, y\_prob)  
 **return** best\_mcc  
 **else**:  
 y\_true = dtrain.get\_label()  
 best\_mcc = eval\_mcc(y\_true, y\_prob)  
 **return 'MCC'**, best\_mcc  
  
  
**def** plot\_precision\_recall(lines, file\_name):  
 plt.clf()  
  
 **for** precision, recall, avg\_pres\_score, label **in** lines:  
 plt.plot(recall, precision, label=**'{0} (area = {1:0.2f}) '**.format(label, avg\_pres\_score))  
 plt.xlabel(**'Recall'**)  
 plt.ylabel(**'Precision'**)  
 plt.ylim([0.0, 1.05])  
 plt.xlim([0.0, 1.0])  
 plt.title(**'Precision-Recall'**)  
 plt.legend(loc=**"upper right"**)  
 *#plt.show()* plt.savefig(file\_name, format=**'png'**)  
  
  
**def** create\_feature\_map(features):  
 **with** open(**'xgb.fmap'**, **'w'**) **as** f:  
 **for** i, feature **in** enumerate(features):  
 f.write(**'{0}\t{1}\tq\n'**.format(i, feature))  
  
  
**def** get\_importance(gbm, features):  
 create\_feature\_map(features)  
 importance = gbm.get\_fscore(fmap=**'xgb.fmap'**)  
 importance = sorted(importance.items(), key=itemgetter(1), reverse=**True**)  
 **return** importance  
  
  
**def** leave\_one\_out(data1, data2, column\_name, use\_LOO=**False**):  
 grp\_outcomes = data1.groupby(column\_name)[**'Response'**].mean().reset\_index()  
 grp\_count = data1.groupby(column\_name)[**'Response'**].count().reset\_index()  
 grp\_outcomes[**'cnt'**] = grp\_count.Response  
 **if** (use\_LOO):  
 grp\_outcomes = grp\_outcomes[grp\_outcomes.cnt > 1]  
 **else**:  
 grp\_outcomes = grp\_outcomes[grp\_outcomes.cnt >= 10]  
 grp\_outcomes.drop(**'cnt'**, inplace=**True**, axis=1)  
 outcomes = data2[**'Response'**].values  
 x = pd.merge(data2[[column\_name, **'Response'**]], grp\_outcomes,  
 suffixes=(**'x\_'**, **''**),  
 how=**'left'**,  
 on=column\_name,  
 left\_index=**True**)[**'Response'**]  
 **if** (use\_LOO):  
 x = ((x \* x.shape[0]) - outcomes) / (x.shape[0] - 1)  
 **return** x.fillna(x.mean())  
  
  
**def** data\_transformation():  
 *# as most useful features with lowest correlation between themselves and max correlation with target variable* cols = [[**'Id'**,  
 **'L1\_S24\_F1559'**, **'L3\_S32\_F3851'**,**'L1\_S24\_F1827'**, **'L1\_S24\_F1582'**,  
 **'L3\_S32\_F3854'**, **'L1\_S24\_F1510'**,**'L1\_S24\_F1525'**],  
 [**'Id'**,  
 **'L3\_S30\_D3496'**, **'L3\_S30\_D3506'**,  
 **'L3\_S30\_D3501'**, **'L3\_S30\_D3516'**,  
 **'L3\_S30\_D3511'** ],  
 [**'Id'**,  
 **'L0\_S0\_F12'**, **'L0\_S1\_F24'**, **'L0\_S2\_F32'**, **'L0\_S3\_F92'**,  
 **'L0\_S4\_F104'**, **'L0\_S5\_F114'**, **'L0\_S6\_F118'**, **'L0\_S7\_F136'**,  
 **'L0\_S8\_F149'**, **'L0\_S9\_F180'**, **'L0\_S10\_F264'**,**'L0\_S11\_F306'**,  
 **'L0\_S12\_F346'**, **'L0\_S13\_F354'**, **'L0\_S14\_F366'**, **'L0\_S15\_F406'**,  
 **'L0\_S16\_F421'**, **'L0\_S17\_F431'**, **'L0\_S18\_F439'**, **'L0\_S19\_F453'**,  
 **'L0\_S20\_F466'**, **'L0\_S21\_F482'**, **'L0\_S22\_F586'**, **'L0\_S23\_F635'**,  
 **'L1\_S24\_F1421'**, **'L1\_S25\_F2677'**, **'L2\_S26\_F3106'**, **'L2\_S27\_F3214'**,  
 **'L2\_S28\_F3292'**, **'L3\_S29\_F3430'**, **'L3\_S30\_F3624'**, **'L3\_S31\_F3842'**,  
 **'L3\_S32\_F3850'**, **'L3\_S33\_F3857'**, **'L3\_S34\_F3880'**, **'L3\_S35\_F3889'**,  
 **'L3\_S36\_F3918'**, **'L3\_S37\_F3950'**, **'L3\_S38\_F3952'**, **'L3\_S39\_F3972'**,  
 **'L3\_S40\_F3982'**, **'L3\_S41\_F4026'**, **'L3\_S43\_F4090'**, **'L3\_S44\_F4112'**,  
 **'L3\_S45\_F4124'**, **'L3\_S47\_F4148'**, **'L3\_S48\_F4193'**, **'L3\_S49\_F4236'**,  
 **'L3\_S50\_F4243'**, **'L3\_S51\_F4262'**,  
 **'Response'**]]  
 train\_data = **None  
  
 for** k, f **in** enumerate(TRAIN\_FILES):  
 **print** f  
 subset = **None  
 for** i, chunk **in** enumerate(pd.read\_csv(DIRECTORY + f, usecols=cols[k], chunksize=50000, low\_memory=**False**)):  
 **print** i  
 **if** subset **is None**:  
 subset = chunk.copy()  
 **else**:  
 subset = pd.concat([subset, chunk])  
 **del** chunk  
 gc.collect()  
 **if** train\_data **is None**:  
 train\_data = subset.copy()  
 **else**:  
 train\_data = pd.merge(train\_data, subset.copy(), on=**"Id"**)  
 **del** subset  
 gc.collect()  
  
 train\_data = train\_data.merge(prepared\_df\_date, on=**'Id'**)  
  
 visible\_train\_data = train\_data[::2]  
 **print** visible\_train\_data.columns  
  
 blind\_train\_data = train\_data[1::2]  
 **print** blind\_train\_data.columns  
  
 **for** i **in** xrange(2):  
 **for** col **in** cols[i][1:]:  
 **print** col  
 blind\_train\_data.loc[:, col] = leave\_one\_out(visible\_train\_data, blind\_train\_data, col, **True**).values  
 **del** visible\_train\_data  
 gc.collect()  
 **return** blind\_train\_data  
  
  
**def** get\_train\_test\_data():  
 train\_data = data\_transformation()  
 **print 'train size:'**, train\_data.shape  
 features = list(train\_data.columns)  
 features.remove(**'Response'**)  
 features.remove(**'Id'**)  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_data[features], train\_data.Response, test\_size=0.33,  
 random\_state=42)  
 **return** features, X\_train, X\_test, y\_train, y\_test  
  
  
**def** train\_xgb\_model(features, X\_train, X\_test, y\_train, y\_test):  
 num\_rounds = 50  
 params = {  
 **'objective'**: **"binary:logistic"** , **'eta'**: 0.021  
 , **'max\_depth'**: 7  
 , **'colsample\_bytree'**: 0.82  
 , **'min\_child\_weight'**: 3  
 , **'base\_score'**: 0.005  
 , **'silent'**: **True** }  
 **print 'Fitting XGBoost'** train\_predictions = **None** X\_train = xgb.DMatrix(X\_train[features], y\_train, silent=**True**)  
 X\_test = xgb.DMatrix(X\_test[features], y\_test, silent=**True**)  
  
 folds = 5  
 **for** i **in** range(folds):  
 **print 'Fold:'**, i  
 params[**'seed'**] = i  
 watchlist = [(X\_train, **'train'**), (X\_test, **'val'**)]  
 clf = xgb.train(params, X\_train,  
 num\_boost\_round=num\_rounds,  
 evals=watchlist,  
 early\_stopping\_rounds=20,  
 feval=mcc\_eval,  
 maximize=**True** )  
 limit = clf.best\_iteration + 1  
  
 predictions = clf.predict(X\_test, ntree\_limit=limit)  
  
 best\_mcc= eval\_mcc(y\_test, predictions)  
 **print 'tree limit:'**, limit  
 **print 'mcc:'**, best\_mcc  
  
 **if** train\_predictions **is None**:  
 train\_predictions = predictions  
 **else**:  
 train\_predictions += predictions  
  
 imp = get\_importance(clf, features)  
 **print 'Importance array: '**, imp  
  
 best\_proba, best\_mcc, y\_pred = eval\_mcc(y\_test, train\_predictions / folds, **True**)  
 **print** best\_proba, best\_mcc  
 y\_pred = (train\_predictions / folds > 0.2365).astype(int)  
  
 precision, recall, threshold = precision\_recall\_curve(y\_test, y\_pred)  
 avg\_pres\_score = average\_precision\_score(y\_test, y\_pred)  
 plot\_precision\_recall([(precision, recall, avg\_pres\_score, **'pr'**)],**'pr\_xgb.png'**)  
 **print** matthews\_corrcoef(y\_test, y\_pred)  
  
  
**def** train\_rnf\_model(features, X\_train, X\_test, y\_train, y\_test):  
 params = {  
 **'max\_depth'**: 7  
 , **'min\_weight\_fraction\_leaf'**: 0  
 , **'n\_jobs'**: 3  
 , **'n\_estimators'**: 400  
 , **'max\_features'**: **None** }  
 **print 'Fitting Random Forest'** train\_predictions = **None** X\_train = X\_train.fillna(999999)  
 X\_test = X\_test.fillna(999999)  
  
 folds = 5  
 **for** i **in** range(folds):  
 **print 'Fold:'**, i  
 params[**'random\_state'**] = i  
  
 mcc\_eval\_scorer = make\_scorer(mcc\_eval, greater\_is\_better=**True**)  
  
 clf = RandomForestClassifier(\*\*params)  
  
 clf.fit(X\_train, y\_train)  
  
 **print 'score: '**, mcc\_eval\_scorer(clf, X\_train, y\_train)  
  
 **for** l, s **in** enumerate(cross\_val\_score(clf, X\_train, y\_train, scoring=mcc\_eval\_scorer, cv=5, n\_jobs=3)):  
 **print '[{0}] train-MCC:{1}'**.format(l, s)  
  
 predictions = clf.predict(X\_test)  
  
 **if** train\_predictions **is None**:  
 train\_predictions = predictions  
 **else**:  
 train\_predictions += predictions  
  
 best\_proba, best\_mcc, y\_pred = eval\_mcc(y\_test, train\_predictions / folds, **True**)  
 **print 'best\_proba, best\_mcc : '**, best\_proba, best\_mcc  
 y\_pred = (train\_predictions / folds > 0.2365).astype(int)  
 precision, recall, threshold = precision\_recall\_curve(y\_test, y\_pred)  
 avg\_pres\_score = average\_precision\_score(y\_test, y\_pred)  
 plot\_precision\_recall([(precision, recall, avg\_pres\_score, **'pr'**)], **'pr\_rnf.png'**)  
 **print** matthews\_corrcoef(y\_test, y\_pred)  
  
  
**print 'started - '** + str(datetime.datetime.now())  
  
features, X\_train, X\_test, y\_train, y\_test = get\_train\_test\_data()  
  
train\_xgb\_model(features, X\_train, X\_test, y\_train, y\_test)  
gc.collect()  
train\_rnf\_model(features, X\_train, X\_test, y\_train, y\_test)  
gc.collect()  
  
**print 'finished - '** + str(datetime.datetime.now())