

Spark ML: extending with PravdaML

Социальная сеть
Одноклассники

2020

Дмитрий Бугайченко

Pravda ML: мотивация



PravdaML

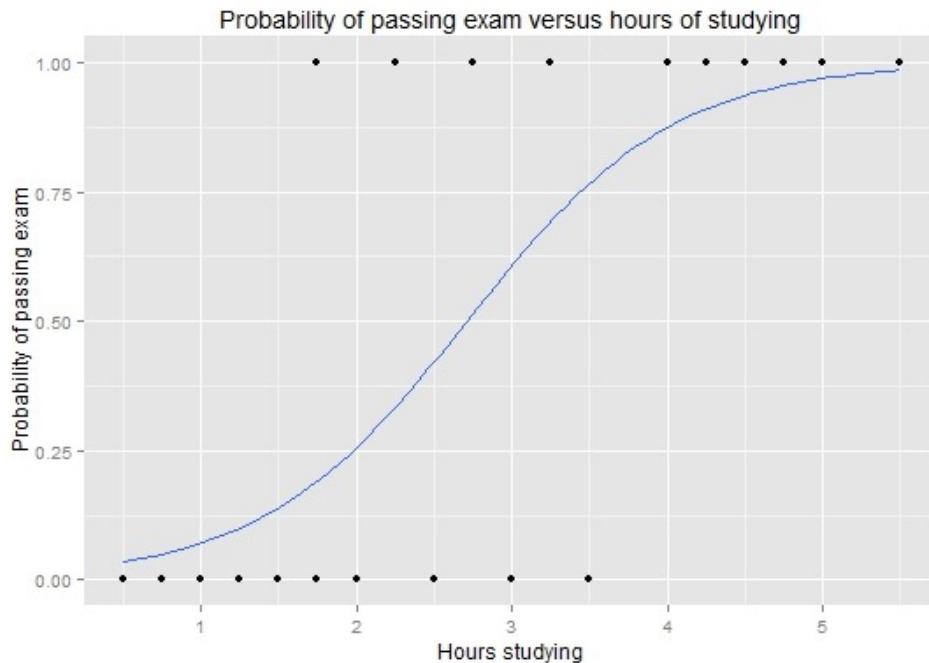
- Проект с открытым кодом:
<https://github.com/odnoklassniki/pravda-ml>
- Расширяет Spark ML с целью
 - Гибкой организации потока данных
 - Лучше утилизацией ресурсов
 - Масштабирования и операционализации ML



Пример задачи

- Прогноз оттока
- Данные: 10M пользователей, 200+ признаков
- Результат: удержание пользователей 😊
- Метод:
 - Найти неочевидные зависимости
 - Построить модель
 - Проанализировать веса
 - Использовать инсайты для принятия решений
 - Profit!

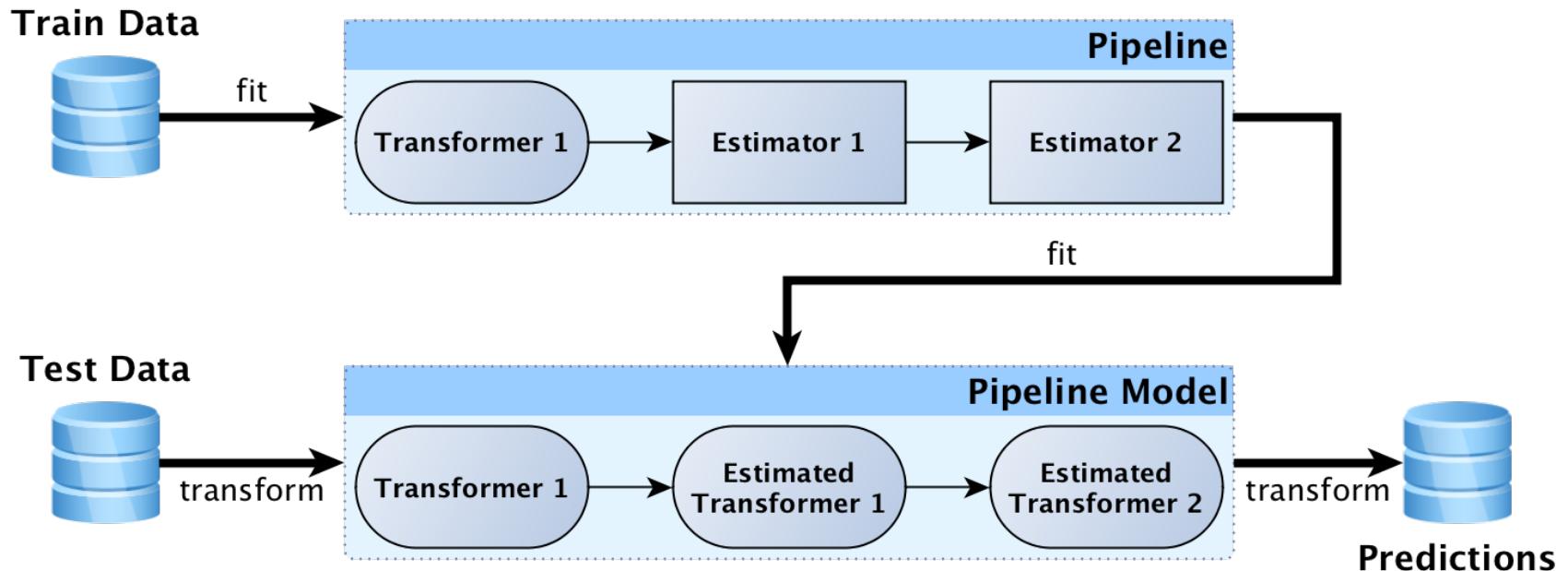
Модель: логистическая регрессия



$$\frac{1}{1 + e^{-(a \cdot x + b)}}$$



Spark ML Pipelines





Первый конвейер

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new LogisticRegressionLBFGS()  
)
```

```
val model = pipeline.fit(data)
```

```
val predictions = model.transform(data)
```



Первый конвейер: Извлекаем признаки

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new LogisticRegressionLBFGS()  
)
```

```
val model = pipeline.fit(data)
```

```
val predictions = model.transform(data)
```



Первый конвейер: Векторизуем

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new LogisticRegressionLBFGS()  
)
```

```
val model = pipeline.fit(data)
```

```
val predictions = model.transform(data)
```



Первый конвейер: Задаем модель

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new LogisticRegressionLBFGS()  
)
```

```
val model = pipeline.fit(data)
```

```
val predictions = model.transform(data)
```



Первый конвейер: Тренируем

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new LogisticRegressionLBFGS()  
)
```

```
val model = pipeline.fit(data)
```

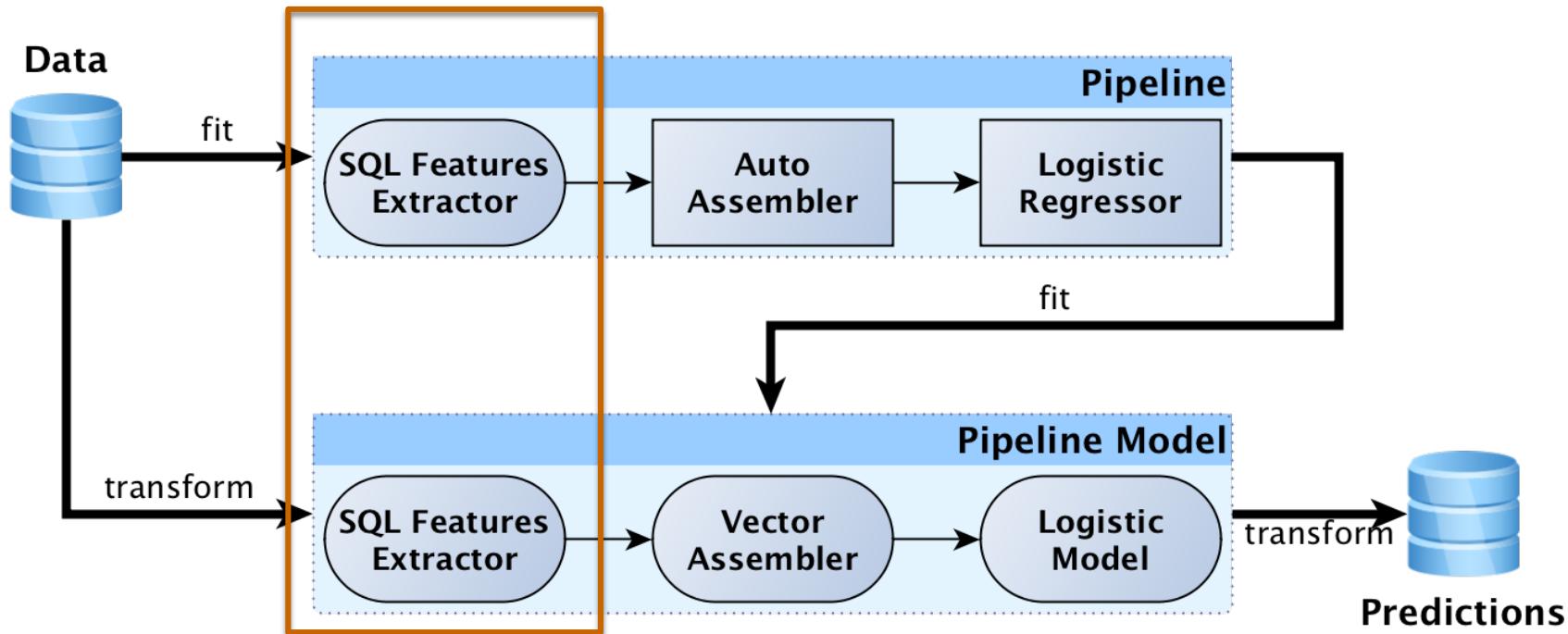
```
val predictions = model.transform(data)
```



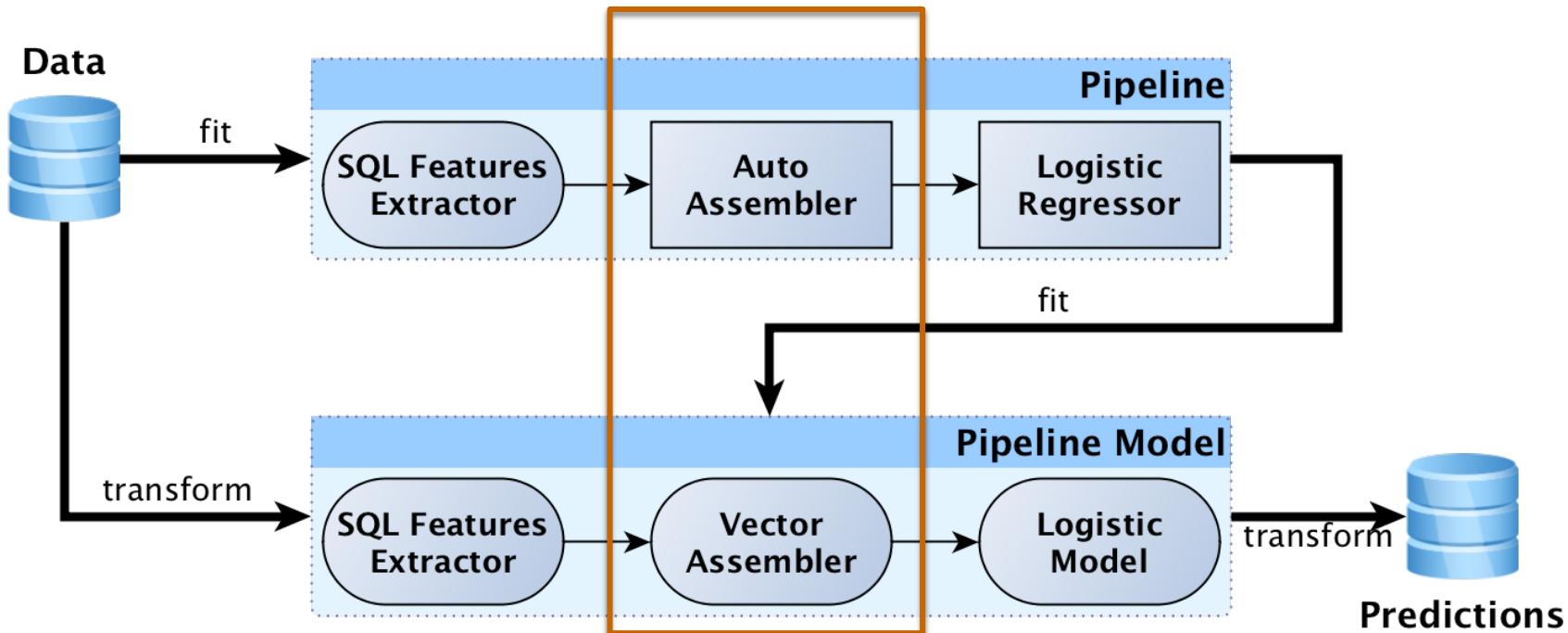
Первый конвейер: Получаем прогноз

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new LogisticRegressionLBFGS()  
)  
  
val model = pipeline.fit(data)  
  
val predictions = model.transform(data)
```

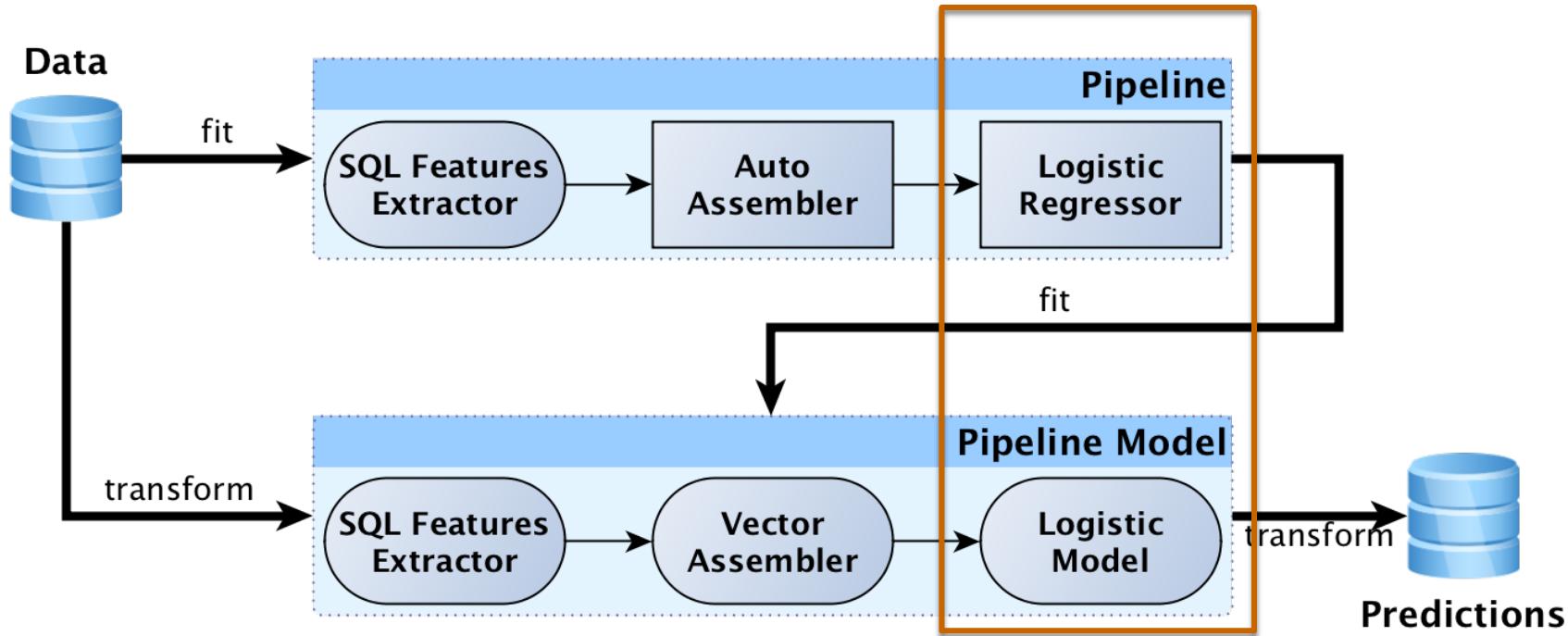
Первый конвейер: Извлекаем признаки



Первый конвейер: Векторизуем



Первый конвейер: Тренируем модель





Изучаем модель

- Нам нужны не сам прогнозы
- Нам нужны инсайты о том, что влияет на отток
- Давайте взглянем на веса



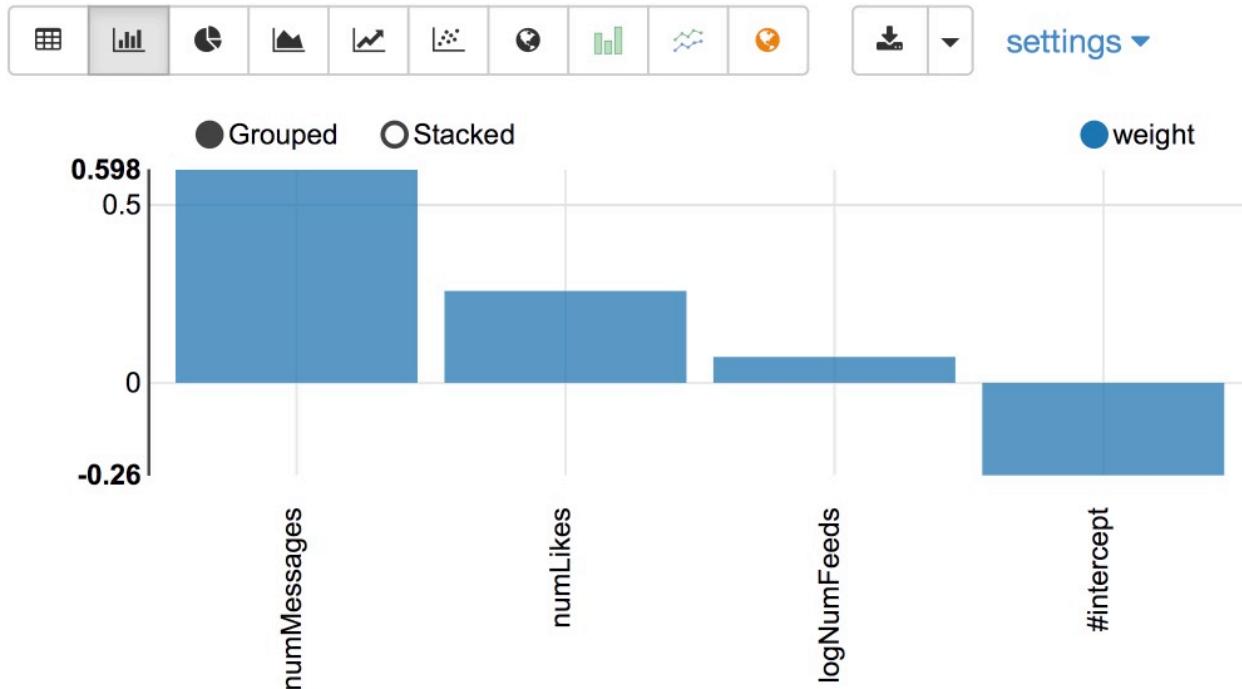
Изучаем модель

- Нам нужны не сам прогнозы
- Нам нужны инсайты о том, что влияет на отток
- Давайте взглянем на веса

```
model.write.save("/churn/simplestModel")
```

```
val weights = sqlc.read.parquet(  
    "/churn/simplestModel/stages/*/weights/")  
  
z.show(weights)
```

Изучаем модель





But wait!

Spark ML does NOT work
this way!

Изучаем модель

- Нам нужны не сам прогнозы
- Нам нужны инсайты о том, что влияет на отток
- Давайте взглянем на веса

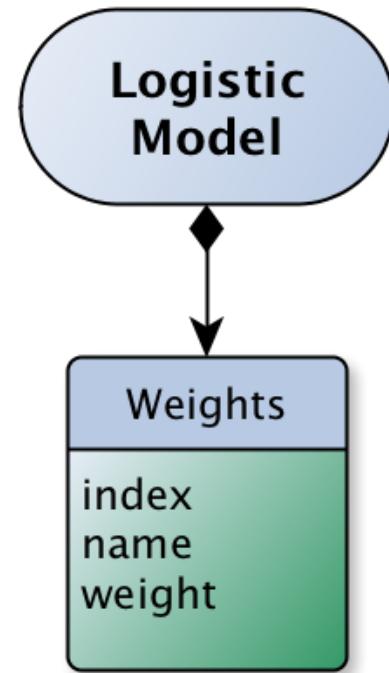
```
model.write.save("/churn/simplestModel")
```

```
val weights = sqlc.read.parquet(  
    "/churn/simplestModel/stages/*/weights/")
```

```
z.show(weights)
```

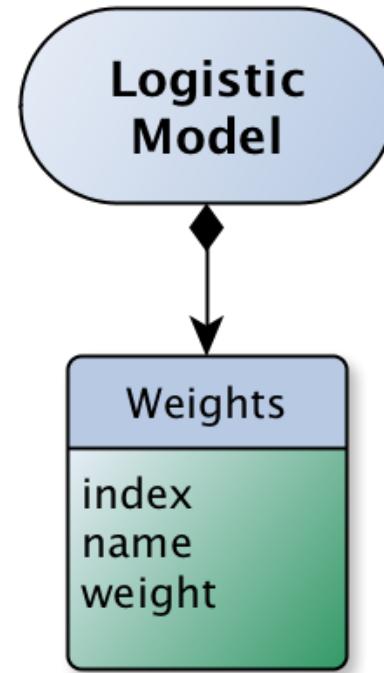
PravdaML: Model Summary

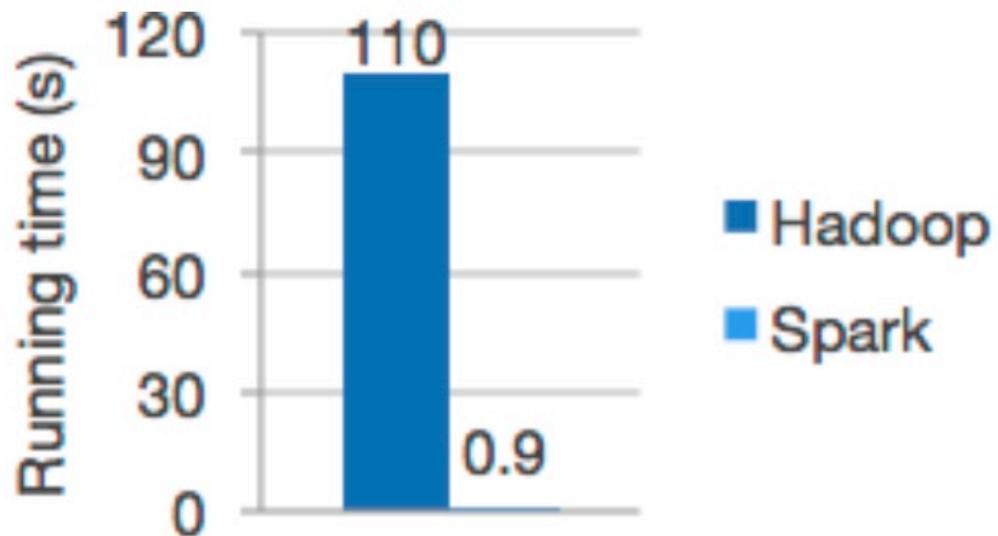
- Коллекция именованных DataFrame
- Добавляется PravdaML-эстиматорами
- Сохраняется в паркетный файл



PravdaML: Model Summary

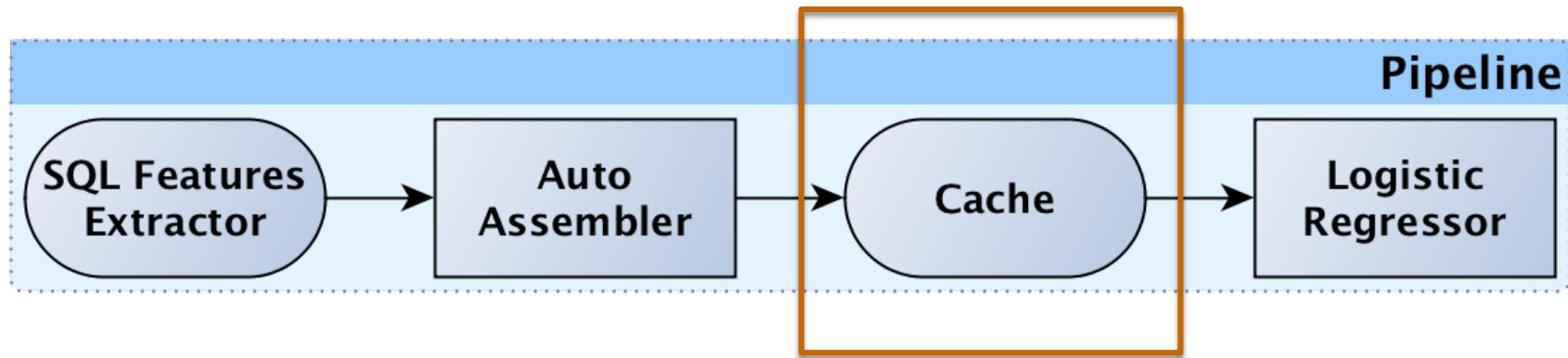
- Коллекция именованных DataFrame
- Добавляется PravdaML-эстиматорами
- Сохраняется в паркетный файл
- Любой Spark ML эстиматор можно завернуть в PravdaML



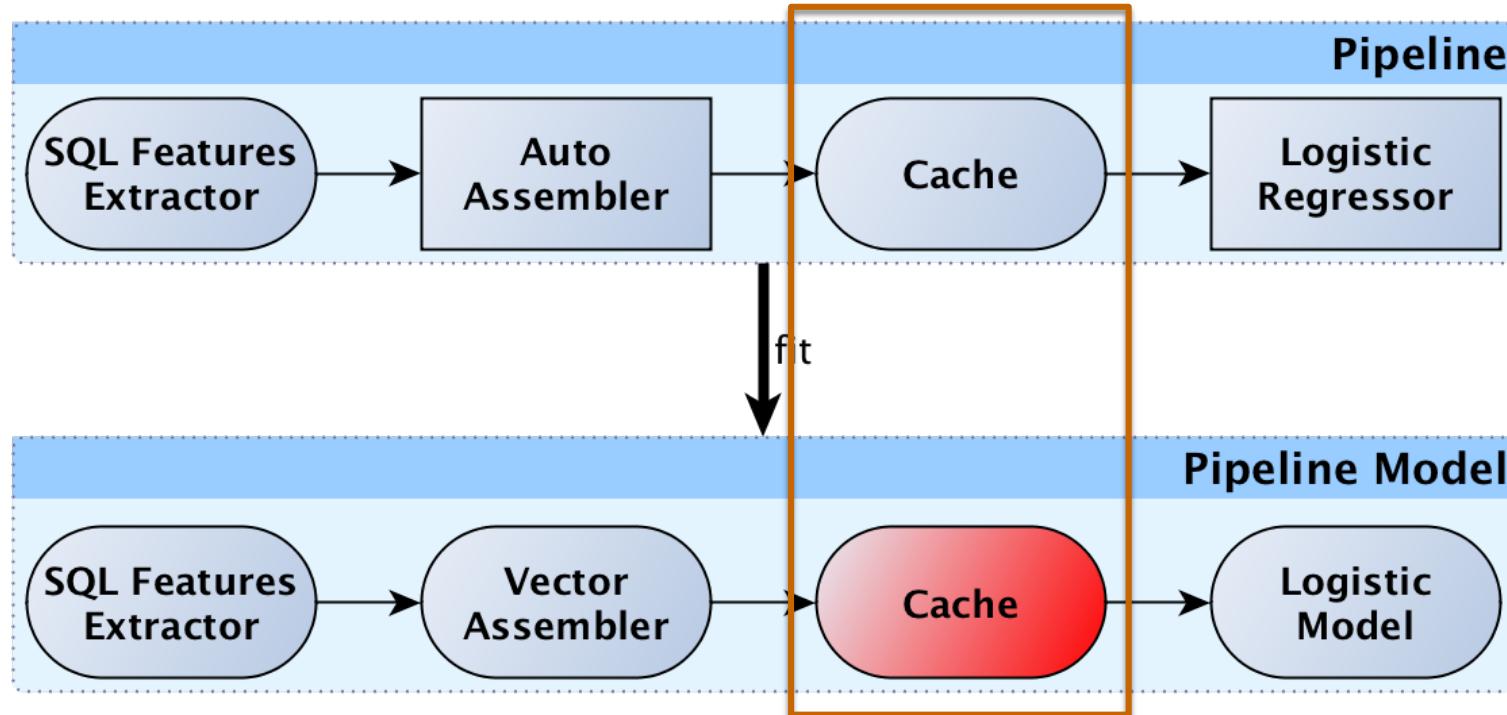


Logistic regression in Hadoop and Spark

Кеширование: простое решение



Кеширование: простое плохое решение

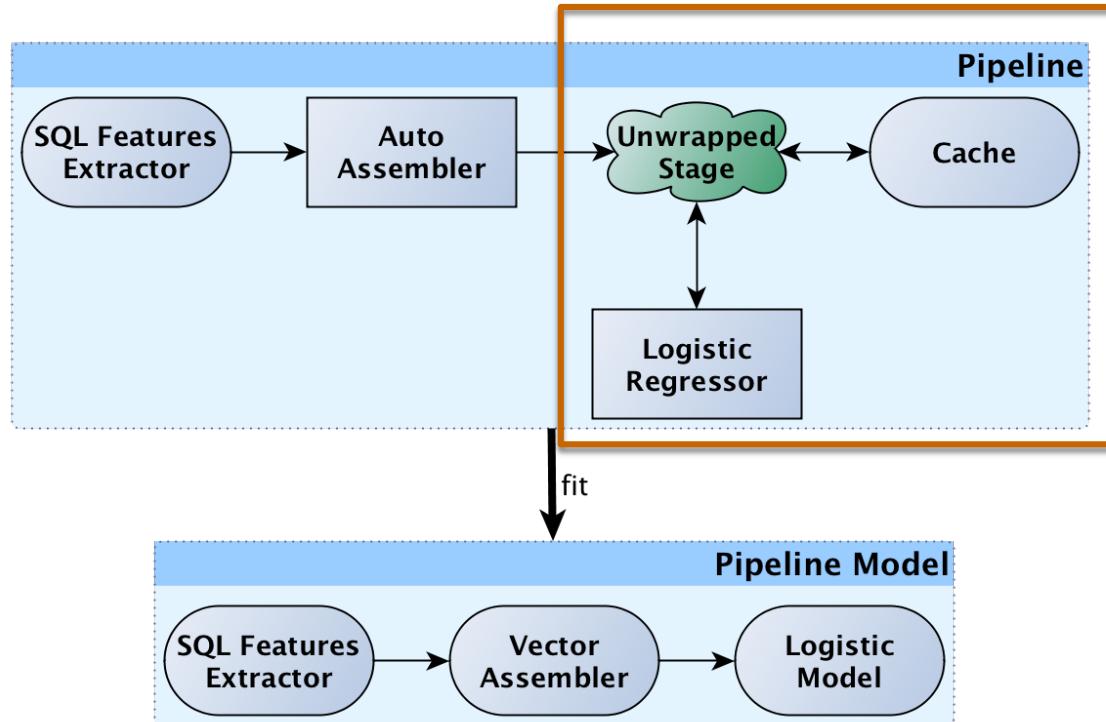




Много подобных ситуаций

- Кеширование
- Партиционирование
- Сохранение временных результатов
- Сэмплирование
- Обрезка по времени
- ...

PravdaML: Ниндзя-этап

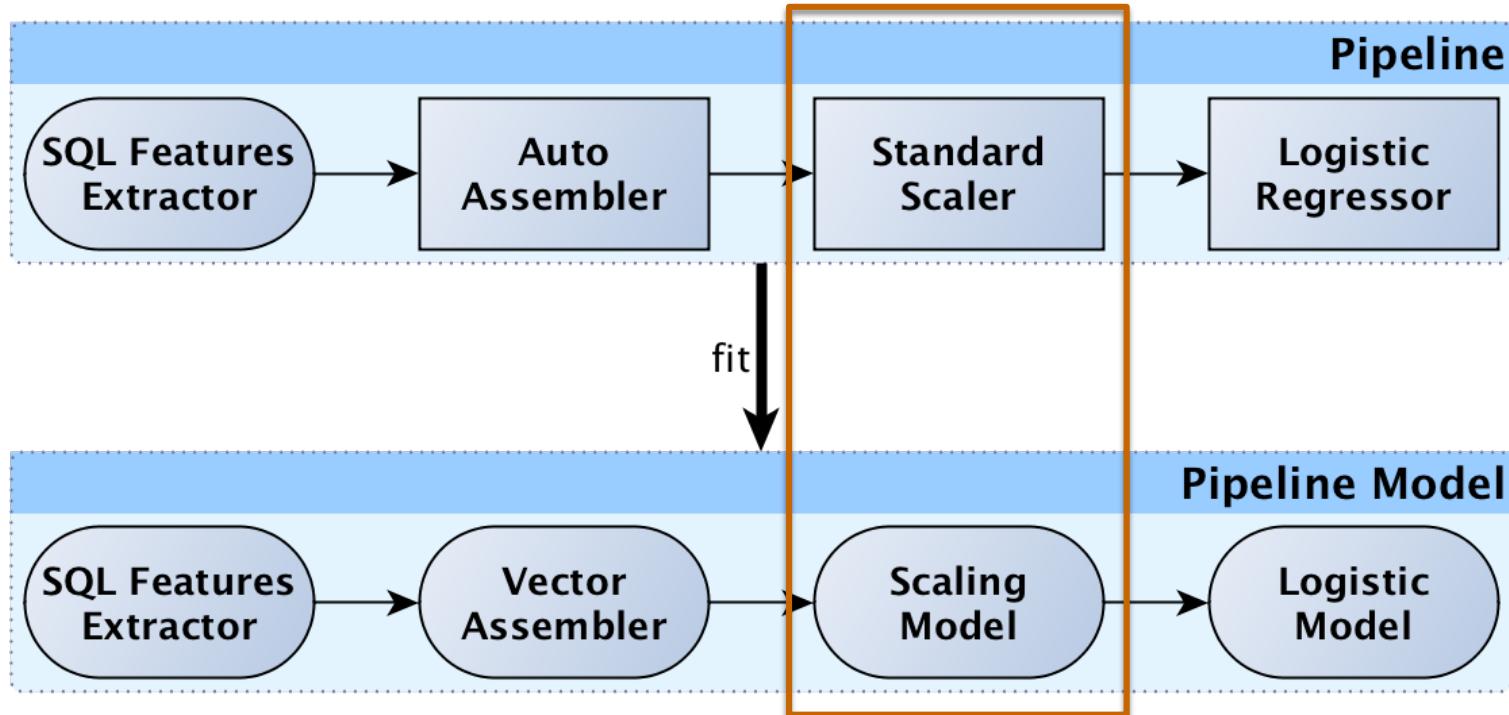




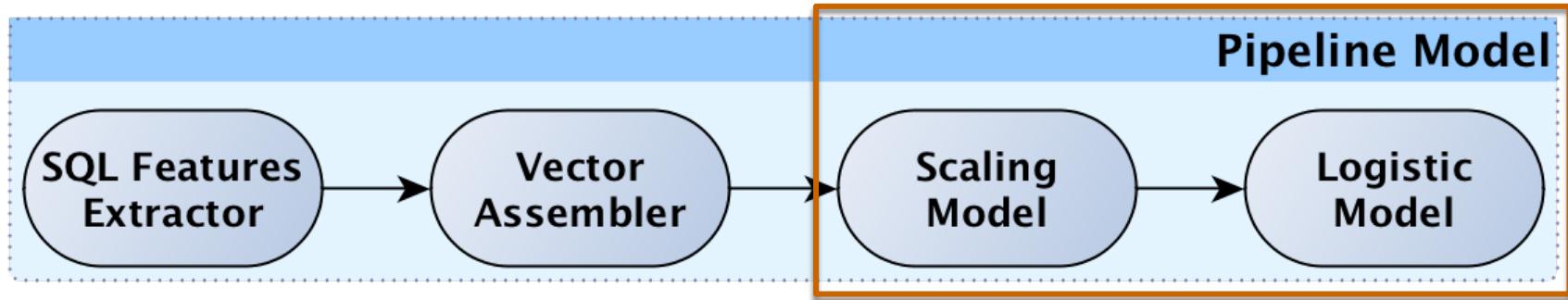
PravdaML: Ниндзя-этап

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    UnwrappedStage.cache(  
        new LogisticRegressionLBFGS()))  
)
```

Забытый этап: нормализация

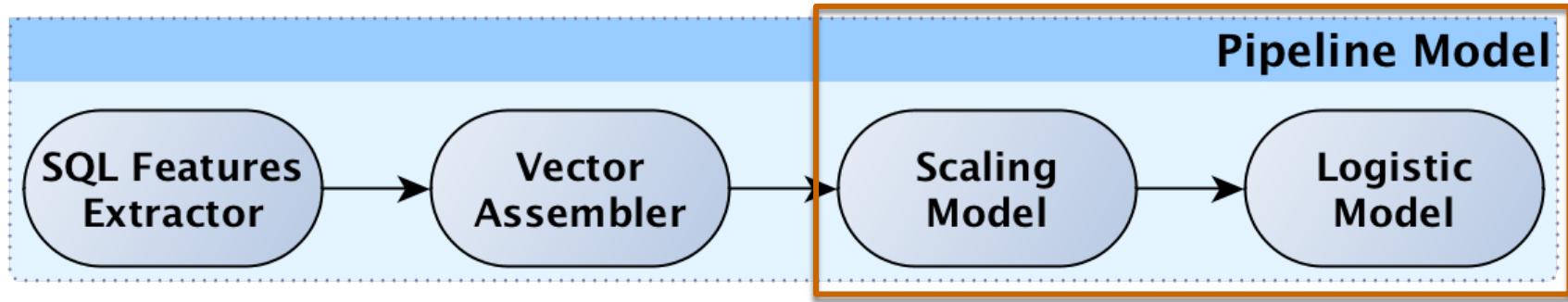


Нормализация Spark ML



- Вычисляем две модели
- Эксплуатируем две модели
- Следим чтобы не разошлись две модели

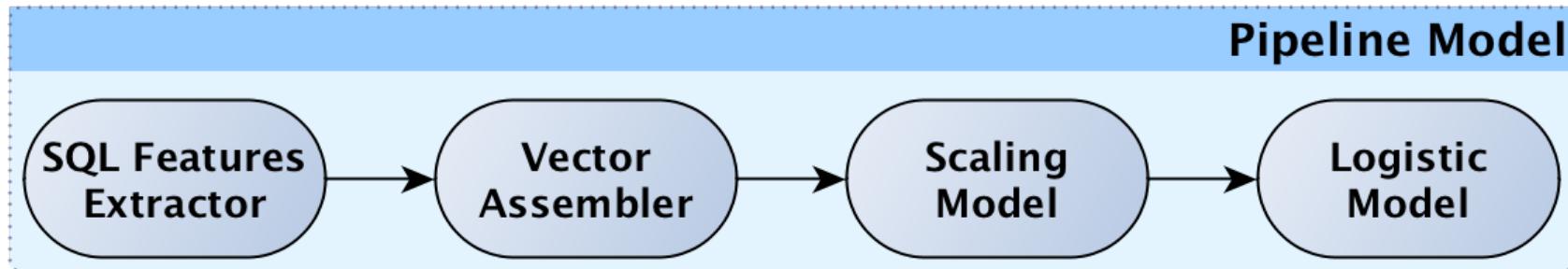
Нормализация в Spark ML



- Вычисляем две модели
- Эксплуатируем две модели
- Следим чтобы не разошлись две модели

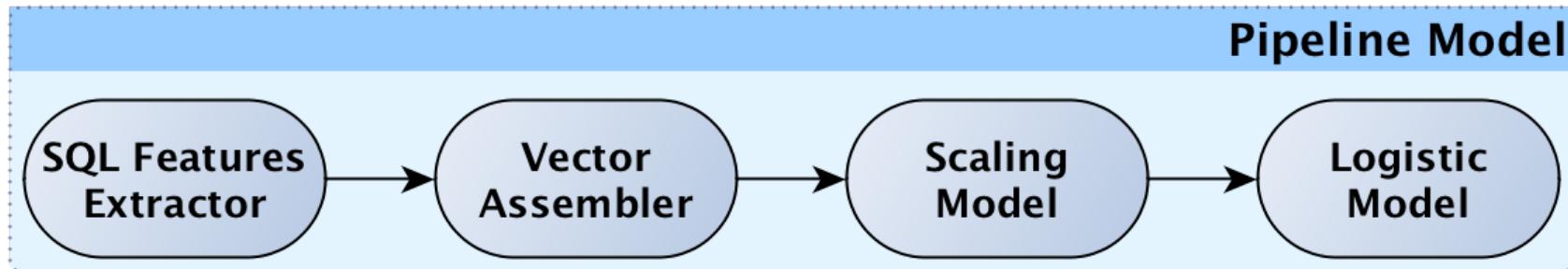


Нормализация в Spark ML



$$\frac{x_i - \bar{x}_i}{sd(x_i)}$$

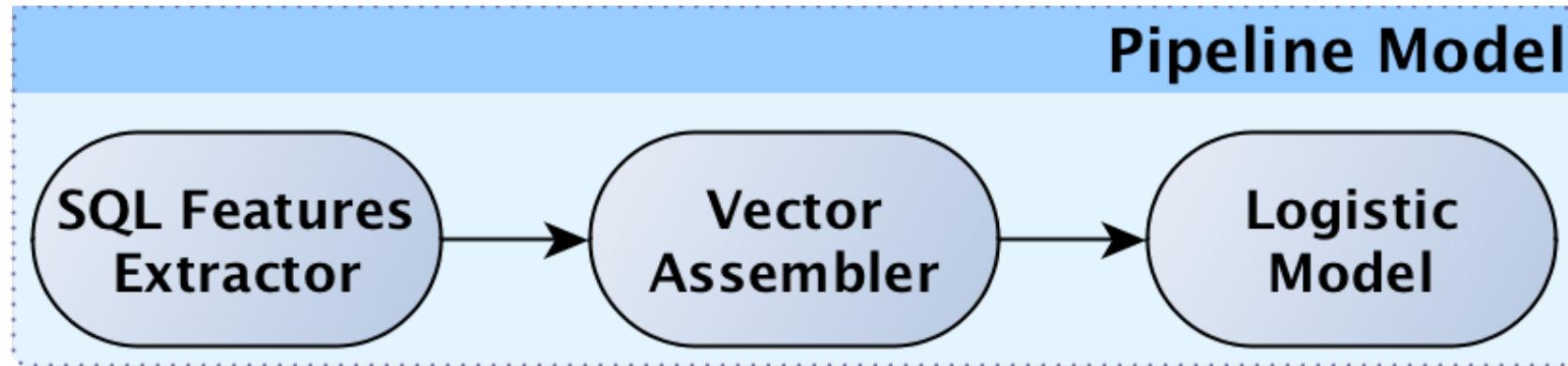
Нормализация в Spark ML



$$\frac{x_i - \bar{x}_i}{sd(x_i)}$$

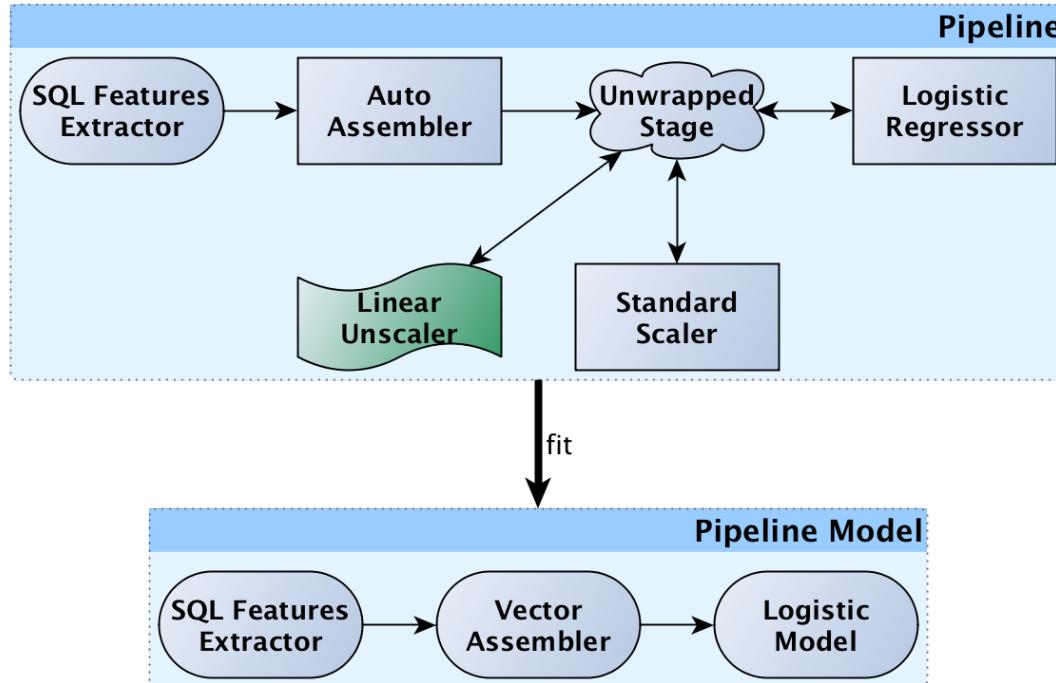
$$\frac{1}{1 + e^{-(a \cdot x + b)}}$$

Инлайнинг нормализации в PravdaML



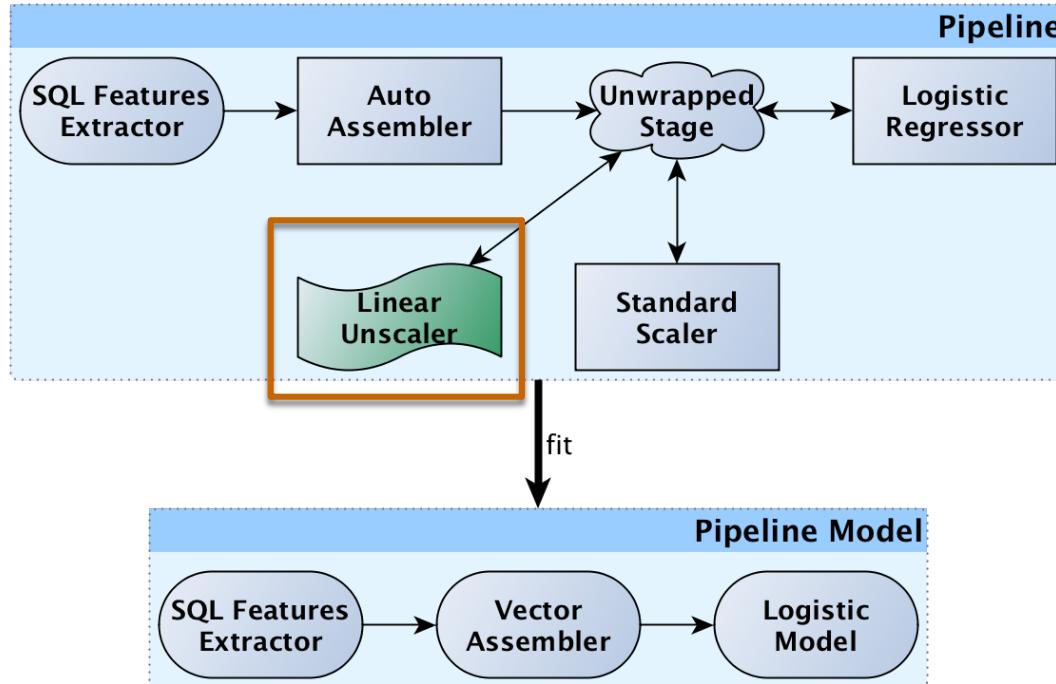
$$a'_i = \frac{a_i - \bar{x}_i}{sd(x_i)}; b' = b - a' \bullet \bar{x}$$

PravdaML: Model Transformer

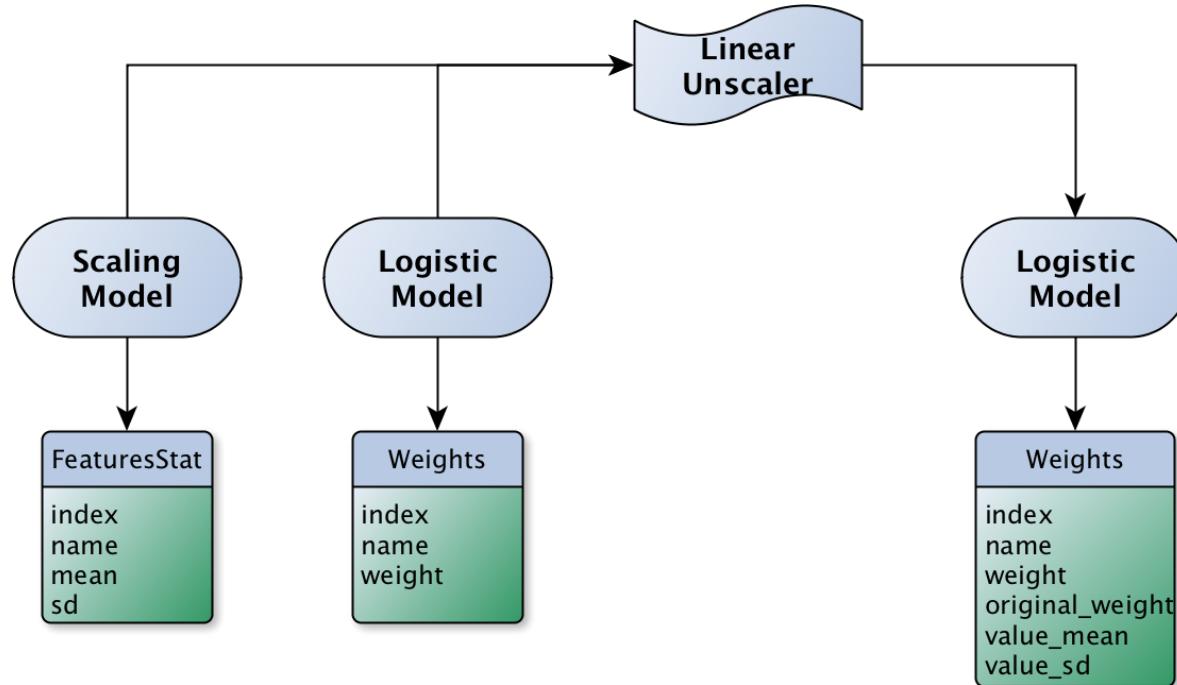




PravdaML: Model Transformer



Инлайнинг нормализации в PravdaML





Инлайнинг нормализации в PravdaML

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    Scaler.scale(UnwrappedStage.cache(  
        new LogisticRegressionLBFGS())))  
)
```



Примеры Model Transformer

- Масштабирование/нормировка
- Добавление intercept
- Отбор признаков
- Сохранение модели

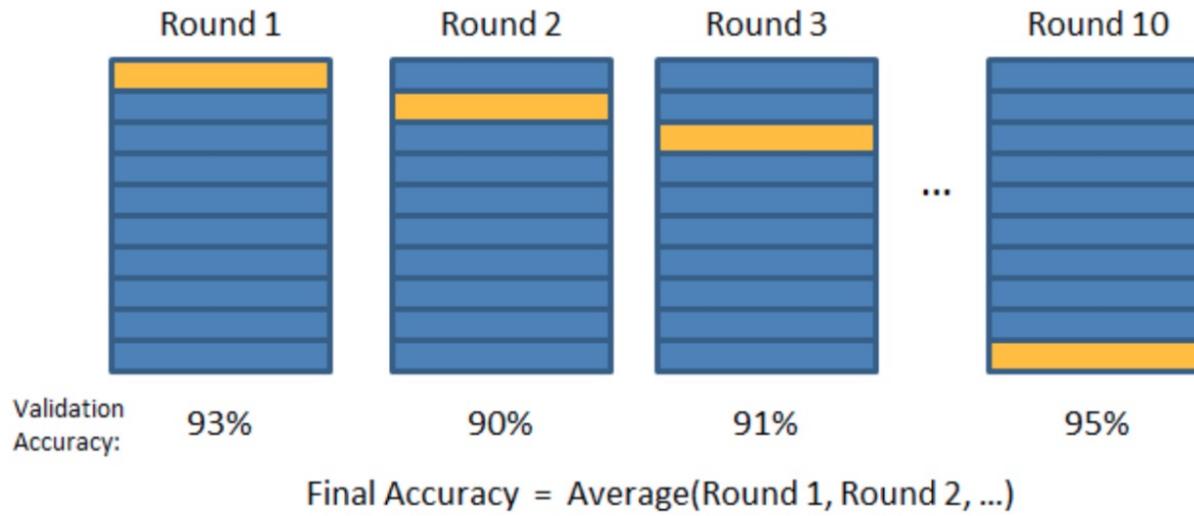


Примеры Model Transformer

- Масштабирование/нормировка
- Добавление intercept
- Отбор признаков
- Сохранение модели
- Оценка качества

Кросс-валидация

Validation Set
Training Set





Spark ML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new CrossValidator()  
        .setEstimator(Scaler.scale(new LogisticRegressionLBFGS()))  
        .setEvaluator(new BinaryClassificationEvaluator())  
        .setNumFolds(10)  
        .setEstimatorParamMaps(Array(new ParamMap())))  
))
```



Spark ML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new CrossValidator()  
        .setEstimator(Scaler.scale(new LogisticRegressionLBFGS()))  
        .setEvaluator(new BinaryClassificationEvaluator())  
        .setNumFolds(10)  
        .setEstimatorParamMaps(Array(new ParamMap()))  
))
```



Spark ML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new CrossValidator()  
        .setEstimator(Scaler.scale(new LogisticRegressionLBFGS()))  
        .setEvaluator(new BinaryClassificationEvaluator())  
        .setNumFolds(10)  
        .setEstimatorParamMaps(Array(new ParamMap())))  
))
```



Spark ML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new CrossValidator()  
        .setEstimator(Scaler.scale(new LogisticRegressionLBFGS()))  
        .setEvaluator(new BinaryClassificationEvaluator())  
        .setNumFolds(10)  
        .setEstimatorParamMaps(Array(new ParamMap()))  
))
```



Spark ML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new CrossValidator()  
        .setEstimator(Scaler.scale(new LogisticRegressionLBFGS()))  
        .setEvaluator(new BinaryClassificationEvaluator())  
        .setNumFolds(10)  
        .setEstimatorParamMaps(Array(new ParamMap()))  
))
```



Spark ML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    new CrossValidator()  
        .setEstimator(Scaler.scale(new LogisticRegressionLBFGS()))  
        .setEvaluator(new BinaryClassificationEvaluator())  
        .setNumFolds(10)  
        .setEstimatorParamMaps(Array(new ParamMap())))  
))
```



Spark ML: Кросс-валидация

```
@Since("1.2.0")
class CrossValidatorModel private[ml] (
    @Since("1.4.0") override val uid: String,
    @Since("1.2.0") val bestModel: Model[_],
    @Since("1.5.0") val avgMetrics: Array[Double])
  extends Model[CrossValidatorModel] with CrossValidatorParams with MLWritable {
  /**
   * A Python-friendly auxiliary constructor.
   */
  private[ml] def this(uid: String, bestModel: Model[_], avgMetrics: JList[Double]) = {
    this(uid, bestModel, avgMetrics.asList.toArray)
  }

  private var _subModels: Option[Array[Array[Model[_]]]] = None
```



Spark ML: Кросс-валидация

```
@Since("1.2.0")
class CrossValidatorModel private[ml] (
    @Since("1.4.0") override val uid: String,
    @Since("1.2.0") val bestModel: Model[_],
    @Since("1.5.0") val avgMetrics: Array[Double])
    extends Model[CrossValidatorModel] with CrossValidatorParams with MLWritable {
    /**
     * A Python-friendly auxiliary constructor.
     */
    private[ml] def this(uid: String, bestModel: Model[_], avgMetrics: JList[Double]) = {
        this(uid, bestModel, avgMetrics.asList.toArray)
    }

    private var _subModels: Option[Array[Array[Model[_]]]] = None
```



PravdaML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    Scaler.scale(UnwrappedStage.cache)  
        Evaluator.crossValidate(  
            estimator = new LogisticRegressionLBFGS(),  
            evaluator = new BinaryClassificationEvaluator()))))
```



PravdaML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    Scaler.scale(UnwrappedStage.cache)  
    Evaluator.crossValidate(  
        estimator = new LogisticRegressionLBFGS(),  
        evaluator = new BinaryClassificationEvaluator()))))
```



PravdaML: Кросс-валидация

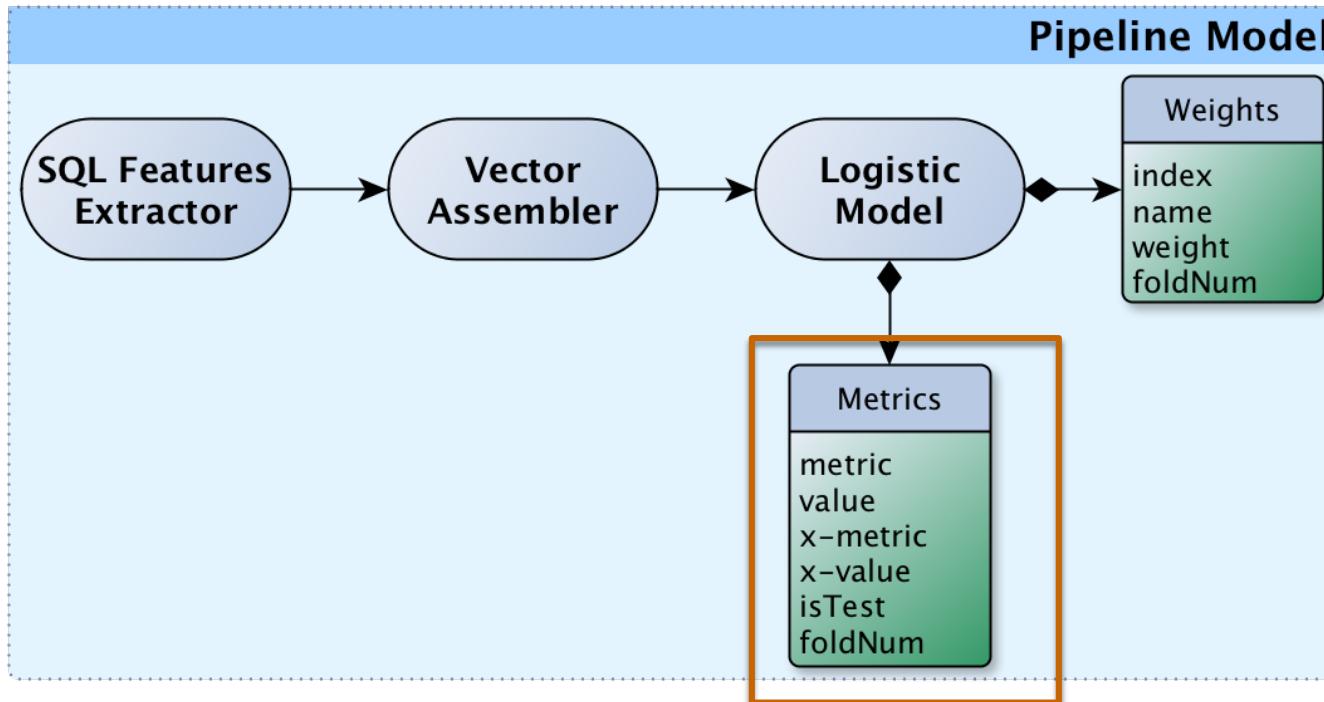
```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    Scaler.scale(UnwrappedStage.cache)  
        Evaluator.crossValidate(  
            estimator = new LogisticRegressionLBFGS(),  
            evaluator = new BinaryClassificationEvaluator()))))
```



PravdaML: Кросс-валидация

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    Scaler.scale(UnwrappedStage.cache)  
    Evaluator.crossValidate(  
        estimator = new LogisticRegressionLBFGS(),  
        evaluator = new BinaryClassificationEvaluator()))  
))
```

PravdaML: Кросс-валидация



PravdaML: Кросс-валидация

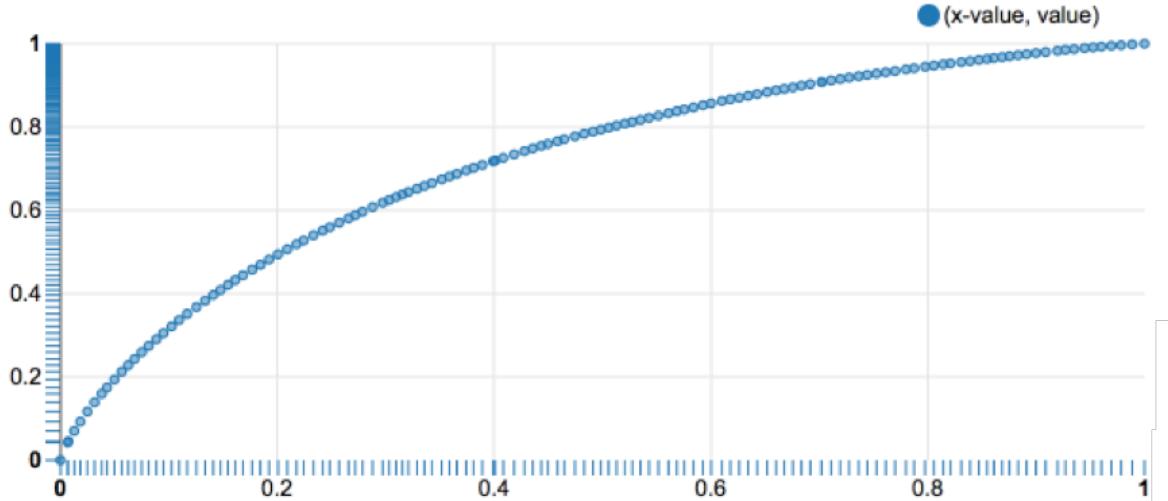
ROC

FINISHED

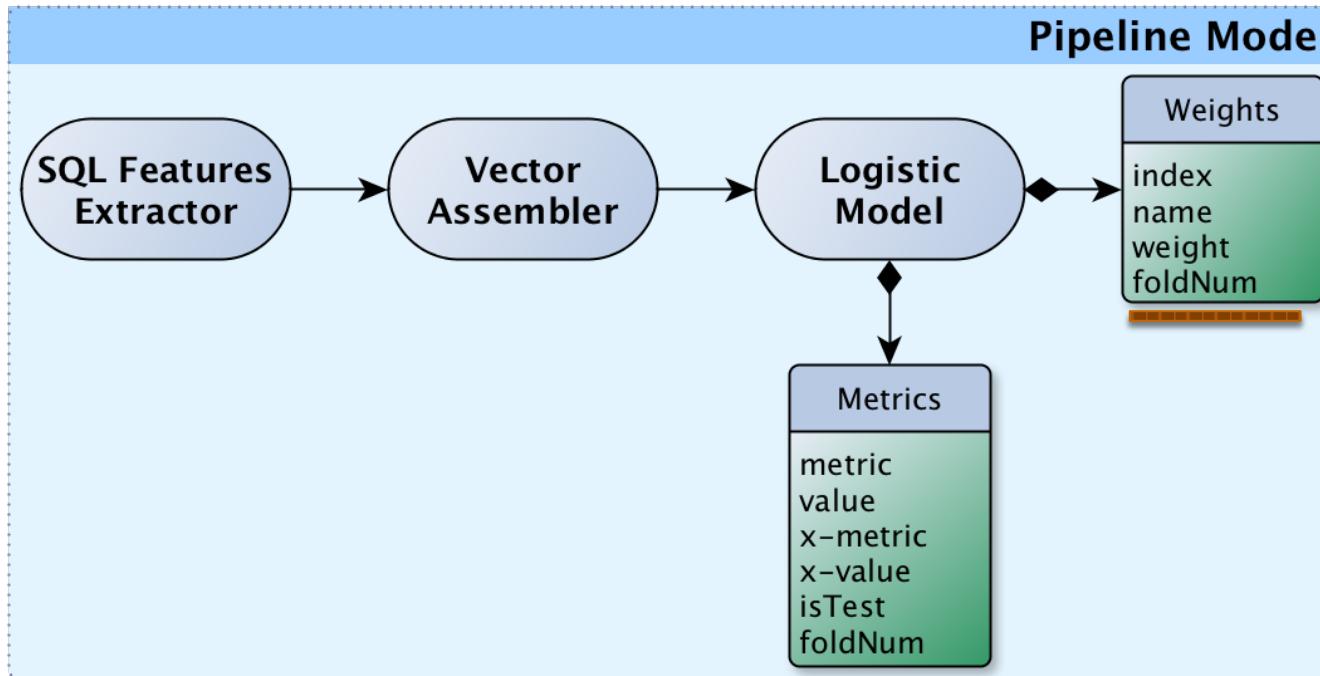
```
z.show(metrics.where("metric = 'tp_rate' AND isTest").orderBy("value"))
```



settings ▾



PravdaML: Кросс-валидация





Кросс-валидация: Spark ML vs. PravdaML

Spark ML

- Оценка модели = скаляр
- Ориентировано на подбор гиперпараметров
- Большая часть информации теряется

PravdaML

- Оценка модели = DataFrame
- Многоцелевое использование
- Большая часть информации сохраняется



PravdaML: Оценка моделей

- Бинарная классификация
 - ROC, RP-curve, F1-plot
- Партиционированное ранжирование
 - NDCG, AUC,...
- Пост-обработка
 - Собираем статистику по метрикам
- Режимы работы
 - Train/test
 - Cross-validation



Пара мыслей по кросс-валидации

- 10+1 моделей
- Нет общего изменяемого состояния
- 90% пересечения по входным данным

Пара мыслей по кросс-валидации

- 10+1 моделей
- Нет общего изменяемого состояния
- 90% пересечения по входным данным
- Не более 100 ядер полезно включаются в тренировку регрессии
- На кластере 10000+ ядер



Spark ML CrossValidator

```
val splits = MLUtils.kFold(dataset.toDF.rdd, $(numFolds), $(seed))
splits.zipWithIndex.foreach { case ((training, validation), splitIndex) =>
    val trainingDataset = sparkSession.createDataFrame(training, schema).cache()
    val validationDataset = sparkSession.createDataFrame(validation, schema).cache()
```

- Последовательно по фолду
- Каждый фолд кешируется отдельно и безусловно

Spark ML CrossValidator

```
val splits = MLUtils.kFold(dataset.toDF.rdd, $(numFolds), $(seed))
splits.zipWithIndex.foreach { case ((training, validation), splitIndex) =>
    val trainingDataset = sparkSession.createDataFrame(training, schema).cache()
    val validationDataset = sparkSession.createDataFrame(validation, schema).cache()
```

- Последовательно по фолду
- Каждый фолд кешируется отдельно и безусловно



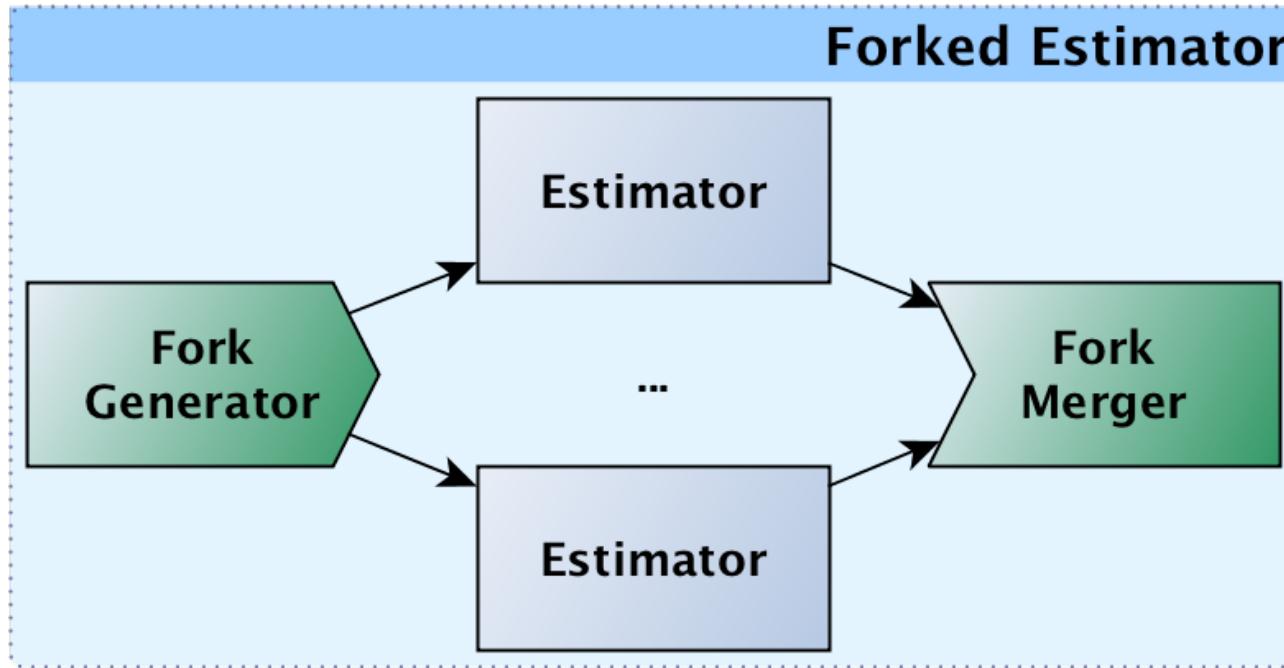
PravdaML: CrossValidator

```
val pipeline = new Pipeline().setStages(Array(  
    new ColumnsExtractor()  
        .withColumns("label", "numMessages", "numLikes")  
        .withExpressions("logNumFeeds" -> "LOG(numFeeds + 1)"),  
    new AutoAssembler().setColumnsToExclude("label"),  
    Scaler.scale(UnwrappedStage.cache(  
        Evaluator.crossValidate(  
            estimator = new LogisticRegressionLBFGS(),  
            evaluator = new BinaryClassificationEvaluator(),  
            numThreads = 11))))
```

- Кешируются разделяемые данные
- Фолды обрабатываются параллельно



PravdaML: Forked Estimator





PravdaML: Forked Estimator

- Сегментация моделей
- Многоклассовая классификация
- Отбор признаков
- Кросс валидация



PravdaML: Forked Estimator

- Сегментация моделей
- Много-классовая классификация
- Отбор признаков
- Кросс валидация
- Подбор гиперпараметров



PravdaML: Дополнительные блоки

- Работа с языком:
 - Language detection
 - Language aware analyzer
 - URL eliminator
 - Vectorizer
 - N-gram extractor
 - Trending term detection
 - LDA (work in progress)
- Статистика
 - Vector stat collector
 - Extended online summarizer
 - TopKTransformer
- Алгоритмы
 - Matrix LBFGS
 - DSRVGD
 - CRR

Подбор гиперпараметров

Data Science



Программистам
мало платят...

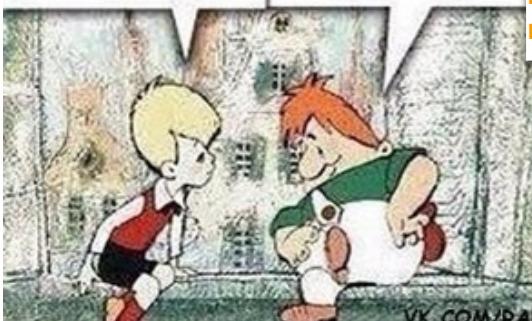


А ты становись дата
саентистом, как я!



Зачем?

Делать AI
за 300 К/с!



Ты же просто переби-
раешь параметры по-
ка валидация не даст
нужный результат!





Data Science

Просто перебираешь параметры пока кросс-валидация не
даст нужный результат...



Data Science

Просто перебираешь параметры пока кросс-валидация не даст нужный результат...

- Алгоритм подбирает параметры модели



Data Science

Просто перебираешь параметры пока кросс-валидация не даст нужный результат...

- Алгоритм подбирает параметры модели
- Дата саентист подбирает алгоритмы и параметры алгоритмов (гиперпараметры)



Data Science

Просто перебираешь параметры пока кросс-валидация не даст нужный результат...

- Алгоритм подбирает параметры модели
- Дата саентист подбирает алгоритмы и параметры алгоритмов (гиперпараметры)
- **AutoML** подбирает алгоритмы и гиперпараметры **вместо** дата саентиста



Параметры vs. Гиперпараметры

- Очень много (миллионы)
- Известна зависимость ответа модели от значений параметров
- Быстро проверить насколько удачны
- Сложно подбирать распределенно
- Очень мало (десятки)
- Как отреагирует результат на изменение неизвестно
- Проверять качество долго
- Поиск распределяется практически линейно



Параметры vs. Гиперпараметры

- Очень много (миллионы)
- Известна зависимость ответа модели от значений параметров
- Быстро проверить насколько удачны
- Сложно подбирать распределенно
- Очень мало (десятки)
- Как отреагирует результат на изменение неизвестно
- Проверять качество долго
- Поиск распределяется практически линейно

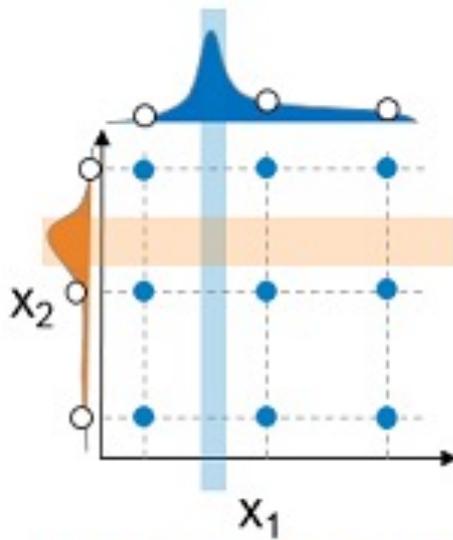


Ограничения SparkML

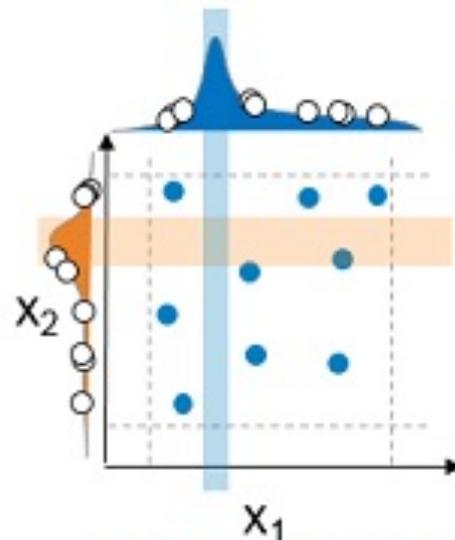
- Один способ подбора гиперпараметров - GridSearch
- Теряется большая часть истории
- Монолитная связка валидации и поиска



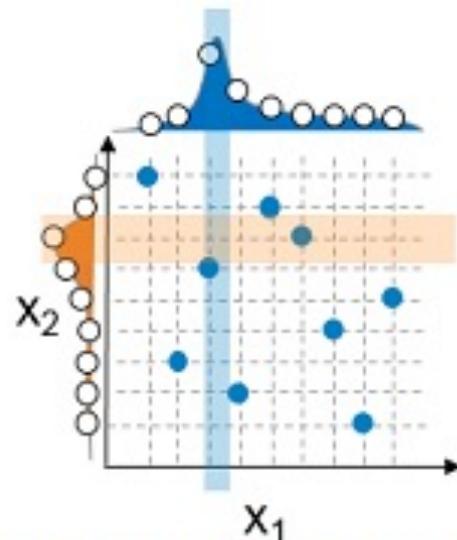
Grid search



Standard Grid Search



Random Search



Random Latin Hypercube



Ограничения SparkML

- Один способ подбора гиперпараметров
- Теряется большая часть истории
- Монолитная связка валидации и поиска
- **Падает в процессе на сложных моделях и приходится начинать с нуля**



PravdaML

- Вся информация о метриках, весах, параметрах и т.д. сохраняется вместе с моделью в паркете
- Параллельное вычисление блоков с восстановлением после падений
- Выделенные абстракции для тест/трэйн разбиения, оценки качества, поиска гиперпараметров
- Гибкое управление параллелизмом
- Прокаченные распределённые МЛ алгоритмы



PravdaML in action

```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.regParam, new DoubleRangeDomain( lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain( lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```



PravdaML in action

```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.regParam, new DoubleRangeDomain( lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain( lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```



PravdaML in action

```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.regParam, new DoubleRangeDomain( lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain( lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```



PravdaML in action

```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.regParam, new DoubleRangeDomain( lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain( lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```



PravdaML in action

```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.reqParam, new DoubleRangeDomain(lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain(lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```



PravdaML in action

```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.regParam, new DoubleRangeDomain( lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain( lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```



PravdaML in action

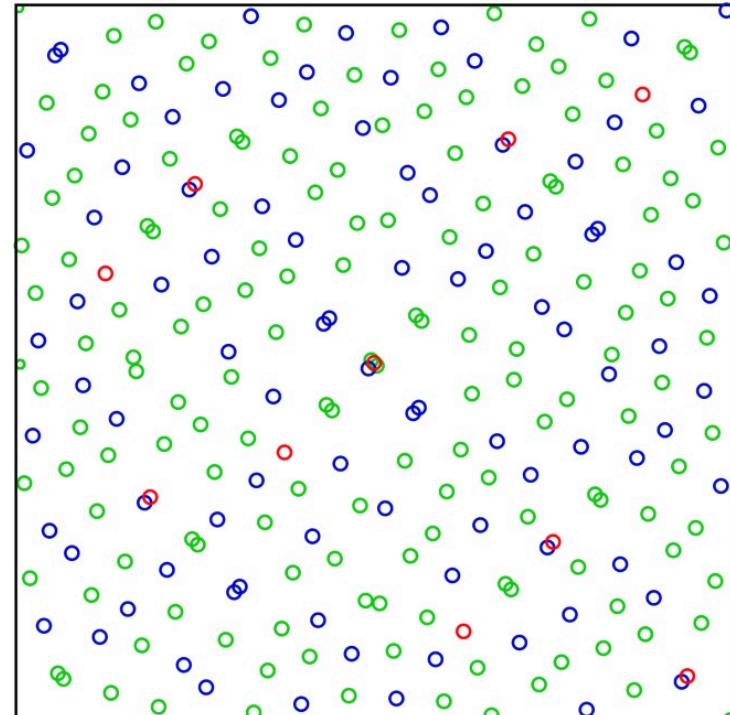
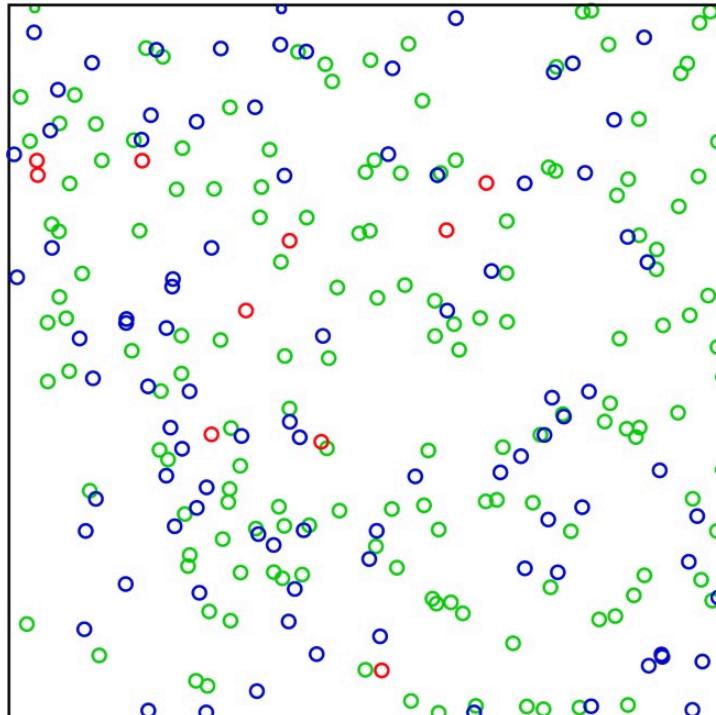
```
val model = new LogisticRegressionLBFGS()

val evaluator = Evaluator.crossValidate(
    model,
    new TrainTestEvaluator(new BinaryClassificationEvaluator()),
    numFolds = 5, numThreads = 5)

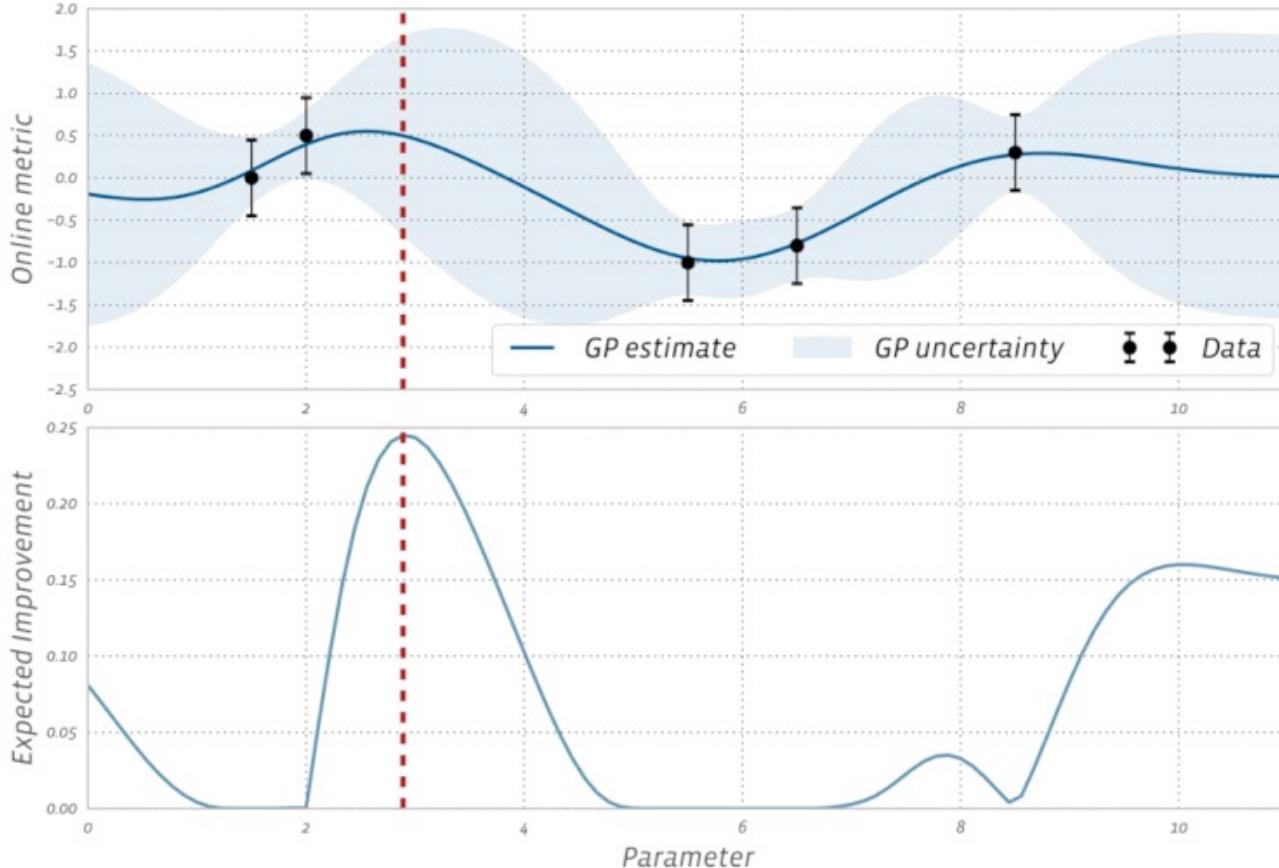
val estimator = new StochasticHyperopt(evaluator)
    .setParamDomains(
        ParamDomainPair(model.regParam, new DoubleRangeDomain( lower = 0, upper = 2)),
        ParamDomainPair(model.elasticNetParam, new DoubleRangeDomain( lower = 0, upper = 0.5)))
    .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")
    .setNumThreads(10)

val result = estimator.fit(data)
```

Случайный поиск: Sobol Sequence



Поиск с аппроксимацией Гауссовским процессом





Особенности реализации

- На базе LinkedIn Photon-ML:
<https://github.com/linkedin/photon-ml>
- Теоретическая база «Practical Bayesian Optimization of Machine Learning Algorithms»:
<https://arxiv.org/pdf/1206.2944.pdf>
- Интеграл по гиперпараметрам Гауссовского процесса
- Можно передать «прайор» от прошлых запусков
- Интеграл по «фантазиям» в процессе оценки (work in progress)



Поиск по группам параметров

```
val optimizer = new StochasticHyperopt(evaluated)
  .setSearchMode(BayesianParamOptimizer.GAUSSIAN_PROCESS)
  .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")

val gridSearch = new GridSearch(evaluated)
  .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")

val estimator = new GroupedSearch(Seq(
  "regParam" -> optimizer.copy(ParamMap(optimizer.paramDomains -> Seq(
    ParamDomainPair(nested.regParam, new DoubleRangeDomain( lower = 0, upper = 2)))),
  "elasticNet" -> gridSearch.copy(ParamMap(gridSearch.estimatorParamMaps -> new StableOrderParamGridBuilder()
    .addGrid(nested.elasticNetParam, Array(0.0,0.25,0.5))
    .build())))
))
```



Поиск по группам параметров

```
val optimizer = new StochasticHyperopt(evaluated)
  .setSearchMode(BayesianParamOptimizer.GAUSSIAN_PROCESS)
  .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")

val gridSearch = new GridSearch(evaluated)
  .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")

val estimator = new GroupedSearch(Seq(
  "regParam" -> optimizer.copy(ParamMap(optimizer.paramDomains -> Seq(
    ParamDomainPair(nested.regParam, new DoubleRangeDomain( lower = 0, upper = 2)))),
  "elasticNet" -> gridSearch.copy(ParamMap(gridSearch.estimatorParamMaps -> new StableOrderParamGridBuilder()
    .addGrid(nested.elasticNetParam, Array(0.0,0.25,0.5))
    .build())))
))
```

Поиск по группам параметров

```
val optimizer = new StochasticHyperopt(evaluated)
  .setSearchMode(BayesianParamOptimizer.GAUSSIAN_PROCESS)
  .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")

val gridSearch = new GridSearch(evaluated)
  .setMetricsExpression("SELECT AVG(value) FROM __THIS__ WHERE metric = 'auc' AND isTest")

val estimator = new GroupedSearch(Seq(
  "regParam" -> optimizer.copy(ParamMap(optimizer.paramDomains -> Seq(
    ParamDomainPair(nested.regParam, new DoubleRangeDomain( lower = 0, upper = 2)))),
  "elasticNet" -> gridSearch.copy(ParamMap(gridSearch.estimatorParamMaps -> new StableOrderParamGridBuilder()
    .addGrid(nested.elasticNetParam, Array(0.0,0.25,0.5))
    .build())))
))
```

Немного о граблях





Падения после долгого обучения

- Причины падения: GC, пертурбации на кластере, баги, ...
- Повторять долгие расчеты заново обидно
- Хочется продолжать с места падения



Падения после долгого обучения

- Причины падения: GC, пертурбации на кластере, баги, ...
- Повторять долгие расчеты заново обидно
- Хочется продолжать с места падения
- **PravdaML: setPathForTempModels**



Зубодробительные планы

- Конвейер подготовки может быть длинным
- Накручивание операций приводит к существенному усложнению плана
- С определенного момента оптимизатору сносит крышу



Зубодробительные планы

- Конвейер подготовки может быть длинным
- Накручивание операций приводит к существенному усложнению плана
- С определенного момента оптимизатору сносит крышу
- PravdaML: persistToTemp



Букет проблем «все в одном»

- Разные оптимальные конфигурации на разных этапах
- Cleaner Thread затыкается при высоком параллелизме
- При работе с нативным МЛ под капотом иногда невозможно в одном процессе учить несколько моделей

OK







Из Спарка можно
запускать Спарк!



PravdaML: ForkedSparkEstimator

```
val forkedPipeline = new Pipeline().setStages(Array(  
    new VectorAssembler()  
    .setInputCols(Array("first", "second"))  
    .setOutputCol("features"),  
    new ForkedSparkEstimator[LinearRegressionModel, LinearRegressionSGD](new LinearRegressionSGD())  
    .setTempPath("tmp/forkedModels")  
    .setMaster("yarn")  
    .setDeployMode("cluster")  
    .setConfOverrides(  
        "spark.dynamicAllocation.enabled" -> "false")  
    .setSubmitArgs(  
        "--num-executors", "4")  
)
```



PravdaML: ForkedSparkEstimator

```
val forkedPipeline = new Pipeline().setStages(Array(  
    new VectorAssembler()  
    .setInputCols(Array("first", "second"))  
    .setOutputCol("features"),  
    new ForkedSparkEstimator[LinearRegressionModel, LinearRegressionSGD](new LinearRegressionSGD())  
    .setTempPath("tmp/forkedModels")  
    .setMaster("yarn")  
    .setDeployMode("cluster")  
    .setConfOverrides(  
        "spark.dynamicAllocation.enabled" -> "false")  
    .setSubmitArgs(  
        "--num-executors", "4")  
)
```



PravdaML: ForkedSparkEstimator

```
val forkedPipeline = new Pipeline().setStages(Array(  
    new VectorAssembler()  
    .setInputCols(Array("first", "second"))  
    .setOutputCol("features"),  
    new ForkedSparkEstimator[LinearRegressionModel, LinearRegressionSGD](new LinearRegressionSGD())  
        .setTempPath("tmp/forkedModels")  
        .setMaster("yarn")  
        .setDeployMode("cluster")  
        .setConfOverrides(  
            "spark.dynamicAllocation.enabled" -> "false")  
    .setSubmitArgs(  
        "--num-executors", "4")  
)
```



PravdaML: ForkedSparkEstimator

```
val forkedPipeline = new Pipeline().setStages(Array(  
    new VectorAssembler()  
    .setInputCols(Array("first", "second"))  
    .setOutputCol("features"),  
    new ForkedSparkEstimator[LinearRegressionModel, LinearRegressionSGD](new LinearRegressionSGD())  
        .setTempPath("tmp/forkedModels")  
        .setMaster("yarn")  
        .setDeployMode("cluster")  
        .setConfOverrides(  
            "spark.dynamicAllocation.enabled" -> "false")  
    .setSubmitArgs(  
        "--num-executors", "4")  
)
```



PravdaML: ForkedSparkEstimator

```
val forkedPipeline = new Pipeline().setStages(Array(  
    new VectorAssembler()  
    .setInputCols(Array("first", "second"))  
    .setOutputCol("features"),  
    new ForkedSparkEstimator[LinearRegressionModel, LinearRegressionSGD](new LinearRegressionSGD())  
        .setTempPath("tmp/forkedModels")  
        .setMaster("yarn")  
        .setDeployMode("cluster")  
        .setConfOverrides(  
            "spark.dynamicAllocation.enabled" -> "false")  
    .setSubmitArgs(  
        "--num-executors", "4")  
)
```



PravdaML: ForkedSparkEstimator

```
val forkedPipeline = new Pipeline().setStages(Array(  
    new VectorAssembler()  
    .setInputCols(Array("first", "second"))  
    .setOutputCol("features"),  
    new ForkedSparkEstimator[LinearRegressionModel, LinearRegressionSGD](new LinearRegressionSGD())  
    .setTempPath("tmp/forkedModels")  
    .setMaster("yarn")  
    .setDeployMode("cluster")  
    .setConfOverrides(  
        "spark.dynamicAllocation.enabled" -> "false")  
    .setSubmitArgs(  
        "--num-executors", "4")  
)
```



Join the movement!

Подключить либу

```
libraryDependencies +=  
  "ru.odnoklassniki" %%  
  "pravda-ml" %  
  "0.6.1" withSources()
```

Скачать сорцы

<https://github.com/odnoklassniki/pravda-ml>