Целью данной работы было познакомиться с базовыми алгоритмами машинного обучения, с реализацией машинного обучения в библиотеке Spark ML, и получить навыки разработки программного обеспечения для анализа данных с использованием pyspark.

Провести анализ датасета, используя алгоритмы машинного обучения:

* Градиентный бустинг для решения задачи регрессии;
* Логистическую регрессию для решения задачи классификации.

Инструмент машинного обучения Apache Spark, библиотека MLlib стандартизирует API-интерфейсы для ML-алгоритмов, чтобы упростить объединение нескольких алгоритмов в один конвейер или рабочий процесс. Это реализовано с помощью следующих специальных структур данных и методов:

1) DataFrame – ML API, который использует DataFrame из Spark SQL в качестве датасета для машинного обучения, который может содержать различные типы данных, включая специфические для Machine Learning.

2) Преобразователь (Transformer) – алгоритм, который может преобразовывать один DataFrame в другой. Например, ML-модель – это трансформер, который преобразует DataFrame с предикторами в DataFrame с прогнозами. Трансформер можно представить в виде абстракции, которая включает преобразователи фичей и обученные модели. Технически Transformer реализует метод transform(), который преобразует один DataFrame в другой путем добавления одного или нескольких столбцов. К примеру, VectorAssembler является преобразователем, поскольку он принимает входный датафрейм и возвращает преобразованный с новым столбцом, который является векторным представлением всех функций.

3) Оценщик (Estimator) – алгоритм, который можно разместить в DataFrame для создания преобразователя. Например, алгоритм обучения – это оценщик, который обучается на DataFrame и создает ML-модель. Можно сказать, что оценщик — это высокоуровневая абстракция алгоритма обучения, который возвращает модель (преобразователь). Она, в свою очередь, преобразует датафрейм в соответствии с параметрами, которые исследуются на этапе подгонки (fitting) или обучения. Технически каждый оценщик реализует метод fit(), принимающий DataFrame и создающий ML-модель, которая имеет метод transform(). Например, алгоритм обучения LogisticRegression, является оценщиком, который возвращает преобразователь LogisticRegresionModel после исследования параметров данных.

4) Конвейер (Pipeline), который связывает несколько преобразователей и оценщиков в единый рабочий процесс машинного обучения. Apache Spark предоставляет класс, который формируется путем объединения различных этапов конвейера, т.е. Estimator’ов и Transformer’ов, выполняемых последовательно. В классе конвейера есть метод fit(), который запускает весь рабочий процесс. Он возвращает модель PipelineModel, которая имеет точно такое же количество этапов, что и конвейер, за исключением того, что все этапы оценщика заменяются соответствующим преобразователем, полученным во время выполнения. Эта модель конвейера может быть сериализована для повторного использования без затрат на настройку или обучение. Во время выполнения каждый этап вызывается последовательно, в зависимости от его типа (преобразователь или оценщик) вызываются соответствующие методы fit() или transform().

5) Параметр (Parameter) для задания настроечных параметров у преобразователей и оценщиков через общий API. Каждый экземпляр преобразователя или оценщика имеет уникальный идентификатор, который полезен при указании параметров.

Введём в датасет дополнительные числовые признаки, взяв за основу рейтинги графств, районов и городов, упорядоченные по возрастанию средней цены. Добавим в каждую строку номера, соответствующие её графству, району и городу из рейтинга.

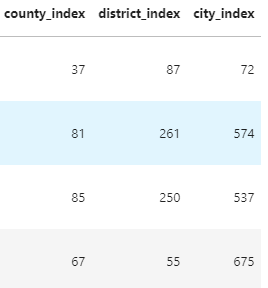


Рисунок 1 – Добавленные колонки

Разделим датасет на тренировочную и тестовую части в соотношении 0,95 к 0,05 воспользовавшись функцией randomSplit из библиотеки PySpark.

Начнем с решения задачи бинарной классификации. Для неё будем использовать алгоритм логистической регрессии.

Логистическая регрессия — это метод классификации, используемый для прогнозирования значения категориальной зависимой переменной на основе связи с одной или несколькими независимыми переменными, которые предположительно имеют логистическое распределение. Если зависимое значение имеет только два возможных значения (успех или неудача), логистическая регрессия будет двоичной. Если зависимая переменная имеет более двух возможных значений (группа крови по результатам диагностического теста), логистическая регрессия будет полиномиальной.

В библиотеке PySpark логистическая регрессия представлена классом LogisticRegression из модуля MLlib.

Объявим pipeline для приведения исходных данных к виду, подходящему в качестве входных данных для логистической регрессии и осуществления обучения модели логистической регрессии и предсказания с её помощью.

lr = LogisticRegression(featuresCol="features", labelCol="new", weightCol="weight", maxIter=10, regParam=0.3)

classification\_pipeline = Pipeline(stages = [

VectorAssembler(inputCols = ["Price", "county\_index", "district\_index", "city\_index"], outputCol="numFeatures"),

MinMaxScaler(inputCol = "numFeatures", outputCol="normFeatures"),

VectorAssembler(inputCols=["additional\_entry", "freehold", "terraced", "semi\_detached", "detached", "flats", "other", "normFeatures"], outputCol="features"),

WeightSetter(inputCol = "new", outputCol = "weight"),

lr

])

Проведём обучение модели на тренировочной части, используя созданный pipeline, после чего получим предсказания для тестовых данных используя обученную модель.

classification\_model = classification\_pipeline.fit(train)

classification\_prediction = classification\_model.transform(test)

Посмотрим на полученные предсказания.

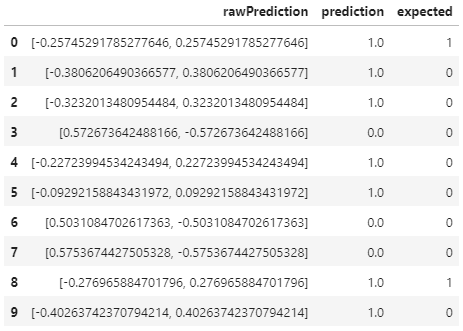


Рисунок 2 – Предсказания обученной модели

ROC-кривая — графический инструмент оценки точности моделей бинарной классификации. Она позволяет найти оптимальный баланс между чувствительностью и специфичностью модели, который соответствует точке ROC-кривой, наиболее близкой к координате (0,1), в которой чувствительность и специфичность равны 1, когда ложноположительные и ложноотрицательные классификации отсутствуют.

В случае идеальной модели график ROC-кривой проходит через точку (0,1), и площадь под ним максимальна и равна 1. По мере снижения точности модели (т.е. росте числа ложноположительных и ложноотрицательных классификаций) кривизна ROC-кривой уменьшается, при этом уменьшается и AUC.

Когда AUC=0, такой классификатор всегда распознает положительный пример как отрицательный, т.е. вероятность ошибки составляет 100%. На практике в зависимости от значения AUC эффективность модели классифицируется следующим образом:

• 0,8 ≤ AUC ≤ 1,0 — модель работает превосходно;

• 0,6 ≤ AUC < 0,8 — модель работает хорошо;

• 0,5 < AUC < 0,6 — модель работает удовлетворительно;

• AUC≤0,5 — модель не работает.

Проведём оценку точности модели по метрике ROC AUC используя класс BinaryClassificationEvaluator из библиотеки PySpark.

classification\_evaluator = BinaryClassificationEvaluator(labelCol="new", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

classification\_evaluator.evaluate(classification\_prediction)

Для данной модели значение ROC AUC составило 0.6739. Это означает, модель далека от идеала, но даёт лучший результат, чем идеальная константная модель.

Построим матрицу несоответствия для более точной оценки модели. Для этого определим количество истинно-положительных, истинно отрицательных, ложно-положительных и ложно-отрицательных предсказаний. Вычислим значение точности (Precision) как отношение числа истинно-положительных предсказаний к общему числу положительных предсказаний (TP/(TP+FP)); значение отзыва (Recall), как отношение числа истинно-положительных предсказаний к общему числу положительных меток (TP/(TP+FN)); значение оценки F1, как отношение удвоенного произведения точности и отзыва к их сумме (F1=(2\*Recall\*Precision)/(Recall+Precision).).

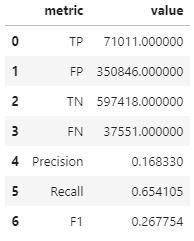


Рисунок 3 – Матрица несоответствия

По результатам построенной матрице несоответствия можно сказать, что модель предсказывает больше половины реально положительных значений, но делает много ложно-положительных предсказаний.

Займемся подбором оптимальных параметров регрессии. Для этого воспользуемся методом кросс-валидации. Используем ParamGridBuilder для задания подбираемых параметром регрессии: regParam (коэффициент регуляризации), maxIter (максимальное число итераций), threshold (пороговое значение).

Произведём подбор параметров и обучение модели используя ранее созданный pipeline и класс CrossValidator.

paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.3, 0.1]).addGrid(lr.maxIter, [10, 5]).addGrid(lr.threshold, [0.4, 0.3]).build()

cv = CrossValidator(estimator=classification\_pipeline, evaluator=BinaryClassificationEvaluator(labelCol="new"), estimatorParamMaps=paramGrid, numFolds=2)

model = cv.fit(train)

newPrediction = model.transform(test)

Получим оценку точности новой модели. Для данной модели значение ROC AUC составило 0.6749, что ненамного больше соответствующей оценки предыдущей модели. Построим матрицу несоответствия для более точной оценки модели.

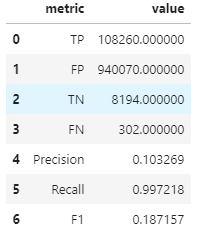


Рисунок 4 - Матрица несоответствия новой модели

По низким значениям TN и FN можно понять, что модель практически не предсказывает значения, равные 0, а значит не подходит для адекватной классификации, и исходный параметр лучше подходили для этой цели.

Перейдём к решению задачи регрессии с использованием метода градиентного бустинга на основе деревьев решений.

Градиентный бустинг — метод машинного обучения, который создает решающую модель прогнозирования в виде ансамбля слабых моделей прогнозирования, обычно деревьев решений. Он строит модель поэтапно, позволяя оптимизировать произвольную дифференцируемую функцию потерь.

Также как и в предыдущей задаче составим pipeline:

regression\_pipeline = Pipeline(stages = [

VectorAssembler(inputCols = ["additional\_entry", "freehold", "new", "terraced", "semi\_detached", "detached", "flats", "other"], outputCol="catFeatures"),

VectorIndexer(inputCol = "catFeatures", outputCol = "idxCatFeatures"),

VectorAssembler(inputCols = ["county\_index", "district\_index", "city\_index"], outputCol="numFeatures"),

MinMaxScaler(inputCol = "numFeatures", outputCol="normFeatures"),

VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"], outputCol="features"),

GBTRegressor(featuresCol="features", labelCol = "Price")

])

Финальным этапом pipeline является класс GBTRegressor, реализующий метод градиентного бустинга на основе деревьев решений.

Проведём обучение модели используя pipeline и получим предсказания для тестовой части датасета.

regression\_pipline\_model = regression\_pipeline.fit(train)

regression\_prediction = regression\_pipline\_model.transform(test)

Посмотрим на получившиеся предсказания.

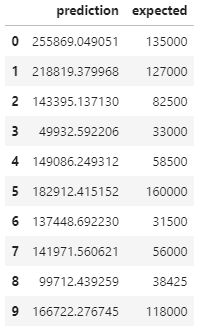


Рисунок 5 – Предсказание модели градиентного бустинга

Проведём оценку полученной модели по метрики r2 используя RegressionEvaluator.

regression\_evaluator = RegressionEvaluator(labelCol="true\_price", predictionCol="prediction", metricName="r2")

regression\_evaluator.evaluate(regression\_prediction)

Для данной модели значение r2 составило 0.3164. Это означает, что модель описывает 32% диспрерсии предсказываемой величины.

Выводы: по результатам проведённого анализа можно сделать вывод, что данные модели далеки от идеальных, но дают лучшие результаты чем идеальная константная модель.