Машинное обучения датасета, а конкретно решение задачи регресии, было выполнена с помощью алгоритма LinearRegression, входящего в состав библиотеки Spark ML.

Spark ML состоит из двух библиотек:

1. spark.ml – это библиотека машинного обучения, основанная на DataFrame API;
2. sparkmllib – на RDD API.

Начиная с версии 2.0 основной библиотекой является spark.ml, но библиотека spark.mllib содержит типы данных, используемые в библиотеке spark.ml

Масштабируемая библиотека машинного обучения (MLlib) Apache Spark предоставляет возможности моделирования для распределенной среды. Пакет Spark spark.ml — это набор API высокого уровня, созданных на основе кадров данных. Эти API позволяют создавать и настраивать практические конвейеры машинного обучения. Машинное обучение Spark использует этот API на основе кадров данных библиотеки машинного обучения, а не более старый API конвейера на основе RDD.

Конвейер машинного обучения — это полный рабочий процесс, объединяющий несколько алгоритмов машинного обучения.

Непрерывная цепочка связанных работ дает следующие преимущества в промышленном Machine Learning:

1. чистый код за счет автоматизации процедур подготовки данных – выборка, очистка, генерация предикторов (фичей, от англ. feature) и пр.;
2. сокращение ошибок благодаря отработанной последовательности шагов, не получится пропустить или неправильно выполнить какой-то этап;
3. простота развертывания в production – обычно преобразовать ML-модель от прототипа к масштабируемому и надежному решению для промышленной эксплуатации достаточно сложно, однако конвейеры помогут и здесь, облегчая тестирование и прочие MLOps-процедуры;
4. дополнительная проверка ML-модели – можно применить перекрестную проверку (кросс-валидацию) и другие методы к этапам конвейера, пробуя различные параметры. Это ускоряет оптимизацию алгоритма и выбор наилучших конфигурационных настроек.

Обработка и анализ данных могут включать в себя множество обязательных шагов, требующих определенной последовательности алгоритмов. Конвейеры определяют этапы и порядок процесса машинного обучения. В конвейер могут входить следующие процедуры подготовки данных к машинному обучению и, собственно, само ML-моделирование:

* формирование выборки;
* устранение пропусков;
* преобразование категориальных значений в номинальные и числовые;
* нормализация диапазона значений для каждого измерения;
* наконец, непосредственно ML-моделирование, где обучается алгоритм машинного обучения.

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.

Загружаем датасет, полученный в 1 главе (рисунок 1).

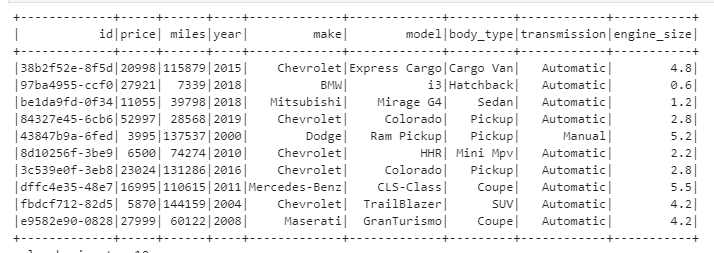


Рисунок 1 – датасет из главы 1

Целевым признаком для нас будет цена – именно её модель и будет предсказывать.

Далее, разбиваем датасет на две части – часть, на которой будем обучать модель (train data) и часть для проверки модели (test data). Разделим датасет по стандартному распределению – 70% данных для обучения и 30% для проверки (рисунок 2).



Рисунок 2 – разделение данных на обучения и проверки

Затем создадим конвейер, который и будет выполнять все необходимые нам действия. Конвейер состоит из ряда этапов, которые обычно подготавливают DataFrame данных для моделирования, а затем обучают прогнозную модель. В данном случае конвейер будет иметь семь этапов (6 преобразователей и 1 оценщик):

1. StringIndexer – преобразователь, который преобразует строковые значения в индексы для категориальных признаков.
2. VectorAssembler - преобразователь, объединяющий категориальные признаки в один вектор.
3. VectorIndexer - преобразователь, который создает индексы для вектора категориальных признаков.
4. VectorAssembler - преобразователь, который создает вектор непрерывных числовых признаков.
5. MinMaxScaler - преобразователь, нормализующий непрерывные числовые признаков.
6. VectorAssembler - преобразователь, который создает вектор категориальных и непрерывных признаков.
7. LinearRegression – оценщик, который обучает модель классификации.

Для LinearRegression определены следующие гиперпараметры:

* maxIter: 10 (максимальное количество итераций)
* regParam: 0.3 (параметр регуляризации)
* elasticNetParam: 0.8 (параметр микширования ElasticNet)

ElasticNet — это популярный тип регуляризованной линейной регрессии, который сочетает в себе два популярных штрафа, а именно штрафные функции L1 и L2.



Рисунок 3 – код создания конвейера

Запустим метод конвейера fit() чтобы обучить модель, передав в функцию подготовленные данные для обучения.

Далее преобразуем тестовые данные со всеми этапами и обученной моделью в конвейере с помощью метода transform(), чтобы сгенерировать метки прогноза.

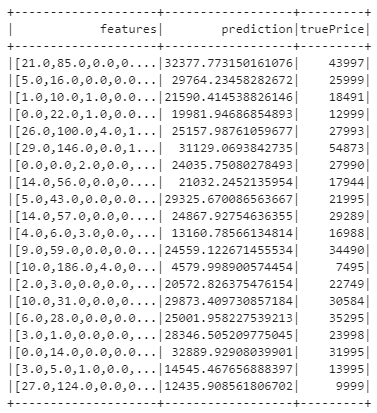


Рисунок 4 – метки прогноза цены

Для оценки качества построенной модели воспользуемся классом pyspark.ml.evaluation RegressionEvaluator. С помощью данного класса рассчитаем 4 метрики: RMSE, MSE, R2, MAE.

Средняя квадратичная ошибка (англ. Mean Squared Error, MSE)



Рисунок 5 - MSE

MSE применяется в ситуациях, когда нам надо подчеркнуть большие ошибки и выбрать модель, которая дает меньше больших ошибок прогноза. Грубые ошибки становятся заметнее за счет того, что ошибку прогноза мы возводим в квадрат. И модель, которая дает нам меньшее значение среднеквадратической ошибки, можно сказать, что что у этой модели меньше грубых ошибок.

Корень из средней квадратичной ошибки (англ. Root Mean Squared Error, RMSE)

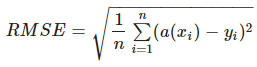


Рисунок 6 - RMSE

Средняя абсолютная ошибка (англ. Mean Absolute Error, MAE)



Рисунок 7 - MAE

Среднеквадратичный функционал сильнее штрафует за большие отклонения по сравнению со среднеабсолютным, и поэтому более чувствителен к выбросам. При использовании любого из этих двух функционалов может быть полезно проанализировать, какие объекты вносят наибольший вклад в общую ошибку — не исключено, что на этих объектах была допущена ошибка при вычислении признаков или целевой величины.

Среднеквадратичная ошибка подходит для сравнения двух моделей или для контроля качества во время обучения, но не позволяет сделать выводов о том, на сколько хорошо данная модель решает задачу. Например, MSE = 10 является очень плохим показателем, если целевая переменная принимает значения от 0 до 1, и очень хорошим, если целевая переменная лежит в интервале (10000, 100000). В таких ситуациях вместо среднеквадратичной ошибки полезно использовать коэффициент детерминации — R2.

Коэффициент детерминации (R2)

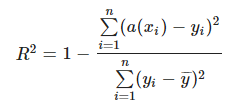


Рисунок 8 - R2

Коэффициент детерминации измеряет долю дисперсии, объясненную моделью, в общей дисперсии целевой переменной. Фактически, данная мера качества — это нормированная среднеквадратичная ошибка. Если она близка к единице, то модель хорошо объясняет данные, если же она близка к нулю, то прогнозы сопоставимы по качеству с константным предсказанием.

Получившиеся метрики модели представлены на рисунке.

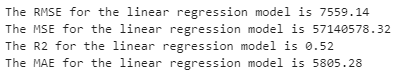


Рисунок 9 – метрики получившийся модели

Коэффициент детерминации дает однозначный вывод, что модель недостаточно хорошо предсказывает цены на подержанные автомобили. Это можно объяснить недостаточным количеством признаков, которые показывают состояния автомобиля. Фактически, текущий датасет имеет только один признак, оценивающий состояние автомобиля – количество пройденных миль. Этого, как показывают данные на рисунке, недостаточно для предсказания цены.

Проблема низких показателей может так же заключаться в неправильном подборе гиперпараметров. Чтобы попытаться найти наиболее эффективные параметры, мы можем использовать класс CrossValidator для оценки каждой комбинации параметров, определенных в ParameterGrid, разделенных на наборы данных для обучения и проверки.

CrossValidator выполняет следующие действия.

1. Разбивает данные на несколько секций.
2. Модель обучается на каждой из секций с помощью указанного алгоритма оценки машинного обучения в наборе данных для обучения.
3. Эффективность каждой модели оценивается с помощью указанного Evaluator на тестовом наборе данных.
4. Для всех моделей возвращается сама модель, а также ее метрики.

Определим следующие наборы параметров:

1. regParam – [0.2, 0.5, 1], elasticNetParam - [0, 0.5, 1], solver - ['auto', 'normal', 'l-bfgs']
2. regParam - [0.1, 0.4, 0.9], maxIter - [10, 20, 30], solver – ['auto', 'normal', 'l-bfgs']
3. maxIter - [25, 50, 75], regParam - [0.3, 0.4, 0.5], elasticNetParam - [0, 0.5, 1]

В таблице приведены используемые в ParameterGrid параметры и полученные метрики.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наборы параметров | RMSE | MSE | R2 | MAE |
| 1 | 5805.28 | 57140590.78 | 0.52 | 5805.28 |
| 2 | 5805.28 | 57140580.58 | 0.52 | 5805.28 |
| 3 | 5805.28 | 57140574.14 | 0.52 | 5805.28 |

Из таблицы можно сделать однозначный вывод, что подбор гиперпараметров не принёс никакого результата, что подтверждает вышеобозначенные выводы о качестве модели.

Решение задачи бинарной классификации было выполнено с помощью алгоритма RandomForest. Так же будет использован датасет из 1 главы работы.

В датасете нет явного бинарного признака. При рассмотрении можно выделить признак transmission, так как он оперирует двумя значениями – Manual и Automatic. Но данный признак слишком однозначно коррелирует с производителем и моделью автомобиля – вероятность что одна модель оснащается как ручной, так и автоматической коробкой передач мала, особенно если рассматривать рынок США сегмента. Поэтому в качестве признака был взят средний объём двигателя, взятый из 1 главы работы.

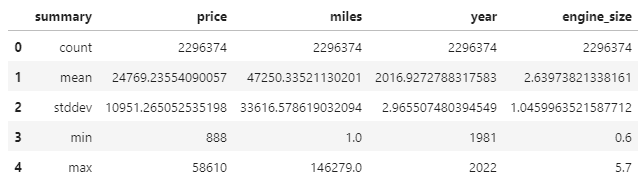


Рисунок 10 – статистика числовых признаков датасета

К DataFrame был добавлен признак label, который формировался следующим образом – если значение engine\_size >= 2.6, то значение label устанавливалось в 1, в противоположном случае – 0 (рисунок 11).

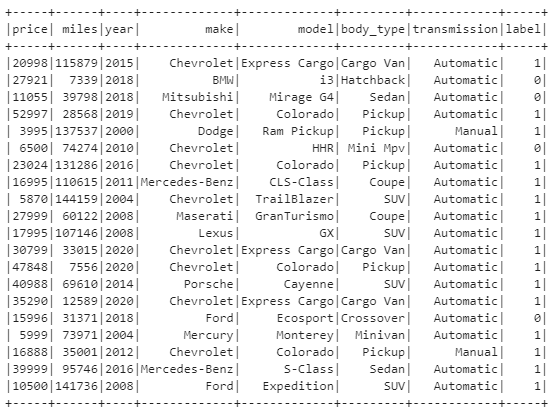


Рисунок 11 – преобразованный датасет с новым признаком

При разделении данных и подготовки конвейера проделывается те же действия, что и в решении задачи регресии. Отличием будет использование оценщика RandomForestClassifier. Для него будут определены определим следующие гиперпараметры:

* numTrees=10 (количество деревьев)
* maxDepth=2 (максимальная глубина дерева решений)
* maxBins=900 (Максимальное количество узлов)



Рисунок 12 – код создания конвейера

Метки для задачи бинарной классификации будут выглядеть таким образом (рисунок 13)

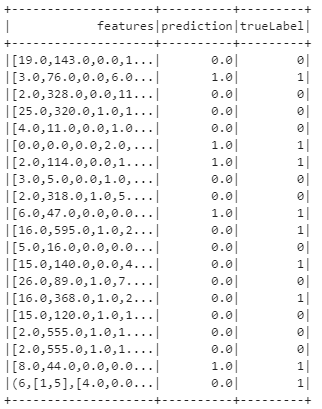


Рисунок 13 – метки прогноза

Для оценки качества моделей воспользуемся матрицей ошибок. Матрица ошибок – это метрика производительности классифицирующей модели. Матрица представлена таблицей с 4 различными комбинациями прогнозируемых и фактических значений. Прогнозируемые значения описываются как положительные и отрицательные, а фактические – как истинные и ложные

|  |  |  |
| --- | --- | --- |
| Прогноз | Реальность | |
| + | - |
| + | True Positive (истинно-положительное решение, TP): прогноз совпал с реальностью, результат положительный произошел, как и было предсказано | False Positive (ложноположительное решение, FP): ошибка 1-го рода, модель предсказала положительный результат, а на самом деле он отрицательный |
| - | False Negative (ложноотрицательное решение, FN): ошибка 2-го рода –модель предсказала отрицательный результат, но на самом деле он положительный | True Negative (истинно-отрицательное решение, TN): результат отрицательный, прогноз совпал с реальностью |

С математической точки зрения оценить точность модели бинарной классификации можно с помощью следующих метрик:

1. Точность (Precision) – сколько всего результатов было предсказано верно: Precision=TP/(TP+FP);
2. Отзыв (recall)– сколько истинных результатов было предсказано верно: Recall =TP/(TP+FN);
3. F-мера, которая позволяет сравнить 2 модели, одновременно оценив полноту и точность: F1=(2\*Recall\*Precision)/(Recall+Precision). Здесь используется среднее гармоническое вместо среднего арифметического, сглаживая расчеты за счет исключения экстремальных значений.

Рассчитаем данные метрики для модели (рисунок 14)

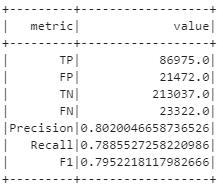


Рисунок 14 – метрики модели

F1 метрика равняется 0.79, что является хорошим результатом. Другой способ оценить производительность модели классификации — измерить площадь под кривой ROC (receiver operating characteristic или «кривая ошибок») для 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.

Библиотека spark.ml включает класс BinaryClassificationEvaluator, который можно использовать для вычисления значений площади под кривой ROC.

Рассчитанное значение равняется 0.909421202881171, что является очень хорошим результатом.

Используем кросс-валидацию, чтобы подобрать более подходящие гиперпараметры для достижения наиболее точных оценок.

Процесс подбора гиперпараметров идентичен механизму, описанному в решении задачи регрессии. Определим следующие наборы параметров:

1. numTrees – [10,15,20], maxDepth - [1, 2, 4], maxBins - [1000, 1200, 1400];
2. numTrees – [1,2,5], maxDepth - [1, 2, 3], maxBins - [900, 1000, 1000];
3. numTrees – [5,10,12], maxDepth - [2, 2, 3], maxBins - [875, 885, 900].

В таблице приведены используемые в ParameterGrid параметры и полученные метрики.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наборы параметров | Precision | Recall | F1 | AUC ROC |
| 1 | 0.804 | 0.827 | 0.815 | 0.866 |
| 2 | 0.808 | 0.786 | 0.794 | 0.848 |
| 3 | 0.813 | 0.798 | 0.806 | 0.856 |

При первом наборе гиперпараметров модель демонстрирует лучшие значения всех ключевых метрик. Это подтверждает обозначенные выше выводы, что модель хорошо справляется с бинарной классификацией автомобилей по объёму двигателя.

Выводы: в этой главе были рассмотрены и решены задачи регрессии и бинарной классификации. В качестве данных был использован датасет из первой главы, содержащий информацию о продаваемых автомобилях в США. Машинное обучение было выполнено с помощью библиотеки Spark ML. В задаче регрессии необходимо было предсказать цену на автомобили с помощью алгоритма LinearRegression, модель справилась не лучшим образом - из-за того, что в датасете отсутствуют признаки, однозначно характеризующие состояние автомобиля. Кросс-валидация не смогла найти гиперпараметры для улучшения модели. Итоговый коэффициент детерминации составил 0.52

Для задачи бинарной классификации был использован алгоритм RandomForest, а также был создан новый бинарный признак – объём двигателя равен или более 2.6. Изначальные гиперпараметры, а также гиперпараметры, подобранные с помощью кросс валидации демонстрируют что модель хорошо справилась с задачей – значение метрики F1 находилась на уровне 0.8, а AUC ROC – 0.8. Лучшие значения метрик – 0.8158 и 0.8663 соответственно.