Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _Информатика и системы управления_____

КАФЕДРА ____Системы обработки информации и управления_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Выбор модели обучения на примере рынка недвижимости

Студент ____ИУ5-34М_____ _____ **Д.С. Гудилин**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **Ю.Е.Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ___ИУ5__
(Индекс)
_____ **В.И. Терехов**
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме _ Выбор модели обучения на примере рынка недвижимости _____
_____
_____

Студент группы ____ИУ5-34М____

___Гудилин Дмитрий Сергеевич_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

___учебная_____

Источник тематики (кафедра, предприятие, НИР)
__кафедра____

График выполнения НИР: 25% к _4_ нед., 50% к _8_ нед., 75% к _12_ нед., 100% к _17 нед.

Техническое задание: *построить классифицирующую модель машинного обучения*

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 19 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « _1_ » ___октября___ 2022 г.

Руководитель НИР _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Студент _____ **Д.С. Гудилин**
(Подпись, дата) (И.О.Фамилия)

Введение

Построение исследования практически невозможно без выбора правильной модели обучения. Процесс выбора модели может напрямую влиять на результат исследования и порой занимает большую часть времени.

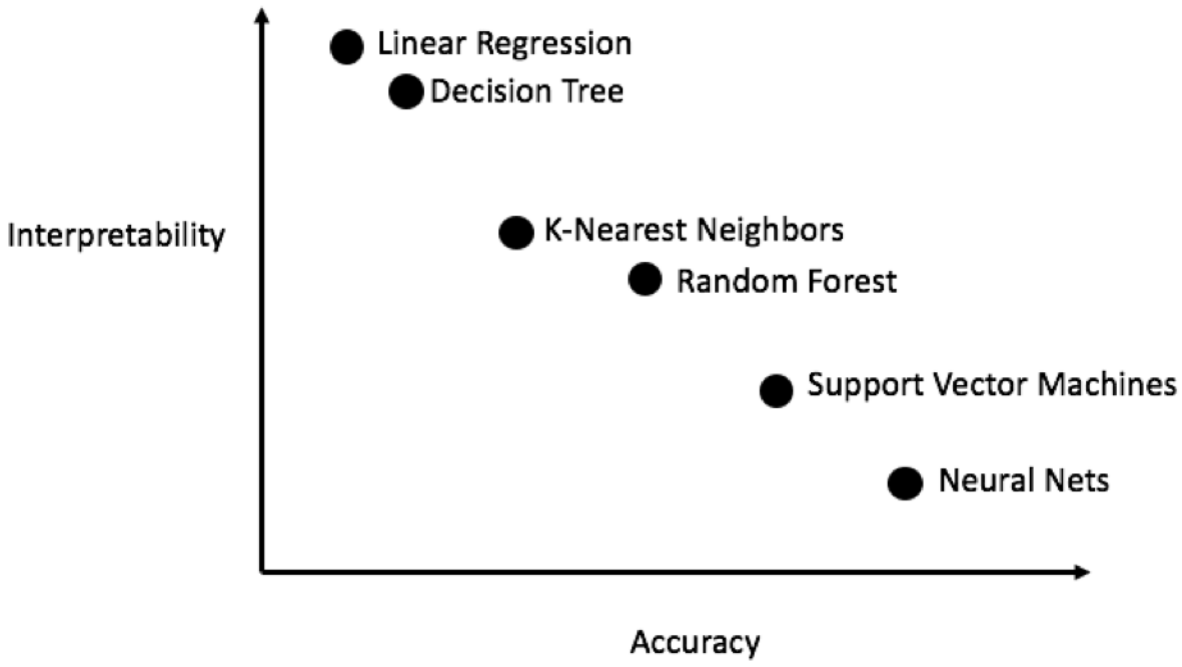
В данном исследовании я рассмотрю популярные модели машинного обучения на примере рынка недвижимости.

Оглавление

1. Оценка и выбор модели

Машинное обучение по большей части основывается на эмпирических, а не теоретических результатах, и практически невозможно заранее понять, какая модель окажется точнее.

Обычно рекомендуется начинать с простых, интерпретируемых моделей, таких как линейная регрессия, и если результаты будут неудовлетворительными, то переходить к более сложным, но обычно более точным методам. На этом графике показана взаимосвязь точности и интерпретируемости некоторых алгоритмов:



Будут оцениваться пять моделей разной степени сложности:

- Линейная регрессия.
- Метод k-ближайших соседей.
- «Случайный лес».
- Градиентный бустинг.

- Метод опорных векторов.

Рассмотрится не теоретический аппарат этих моделей, а их реализация.

Хотя при очистке данных отбрасываются колонки, в которых не хватает больше половины значений, у нас ещё отсутствует немало значений. Модели машинного обучения не могут работать с отсутствующими данными, поэтому нам нужно их заполнить.

Сначала считаю данные:

```
import pandas as pd
import numpy as np
# Read in data into dataframes
train_features = pd.read_csv('data/training_features.csv')
test_features = pd.read_csv('data/testing_features.csv')
train_labels = pd.read_csv('data/training_labels.csv')
test_labels = pd.read_csv('data/testing_labels.csv')
Training Feature Size:  (6622, 64)
Testing Feature Size:   (2839, 64)
Training Labels Size:   (6622, 1)
Testing Labels Size:    (2839, 1)
```

Каждое NaN-значение — это отсутствующая запись в данных. Заполнять их можно по-разному, воспользуюсь достаточно простым методом медианного заполнения (median imputation), который заменяет отсутствующие данные средним значениями по соответствующим колонкам.

В нижеприведённом коде создам Scikit-Learn-объект Imputer с медианной стратегией. Затем обучю его на обучающих данных (с помощью imputer.fit), и применю для заполнения отсутствующих значений в обучающем и тестовом наборах (с помощью imputer.transform). То есть записи, которых не хватает в *тестовых данных*, будут заполняться соответствующим медианным значением из *обучающих данных*.

Делаю заполнение и не обучаю модель на данных как есть, чтобы избежать проблемы с утечкой тестовых данных, когда информация из тестового датасета переходит в обучающий.

```
# Create an imputer object with a median filling strategy
imputer = Imputer(strategy='median')
# Train on the training features
imputer.fit(train_features)
# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
Missing values in training features:  0
Missing values in testing features:   0
```

Теперь все значения заполнены, пропусков нет.

1. Масштабирование признаков

Масштабированием называется общий процесс изменения диапазона признака. Это необходимый шаг, потому что признаки измеряются в разных единицах, а значит покрывают разные диапазоны. Это сильно искажает результаты таких алгоритмов, как метод опорных векторов и метод k-ближайших соседей, которые учитывают расстояния между измерениями. А масштабирование позволяет этого избежать. И хотя методы вроде линейной регрессии и «случайного леса» не требует масштабирования признаков, лучше не пренебрегать этим этапом при сравнении нескольких алгоритмов.

Масштабирование проводится с помощью приведения каждого признака к диапазону от 0 до 1. Беру все значения признака, выбираем минимальное и делю его на разницу между максимальным и минимальным (диапазон). Такой способ масштабирования часто называют нормализацией, а другой основной способ — стандартизацией.

Этот процесс легко реализовать вручную, поэтому воспользуюсь

объектом MinMaxScaler из Scikit-Learn. Код для этого метода идентичен коду для заполнения отсутствующих значений, только вместо вставки применяется масштабирование.

```
# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))
# Fit on the training data
scaler.fit(X)
# Transform both the training and testing data
X = scaler.transform(X)
X_test = scaler.transform(X_test)
```

Теперь у каждого признака минимальное значение равно 0, а максимальное 1. Заполнение отсутствующих значений и масштабирование признаков — эти два этапа нужны почти в любом процессе машинного обучения.

2. Реализация модели машинного обучения

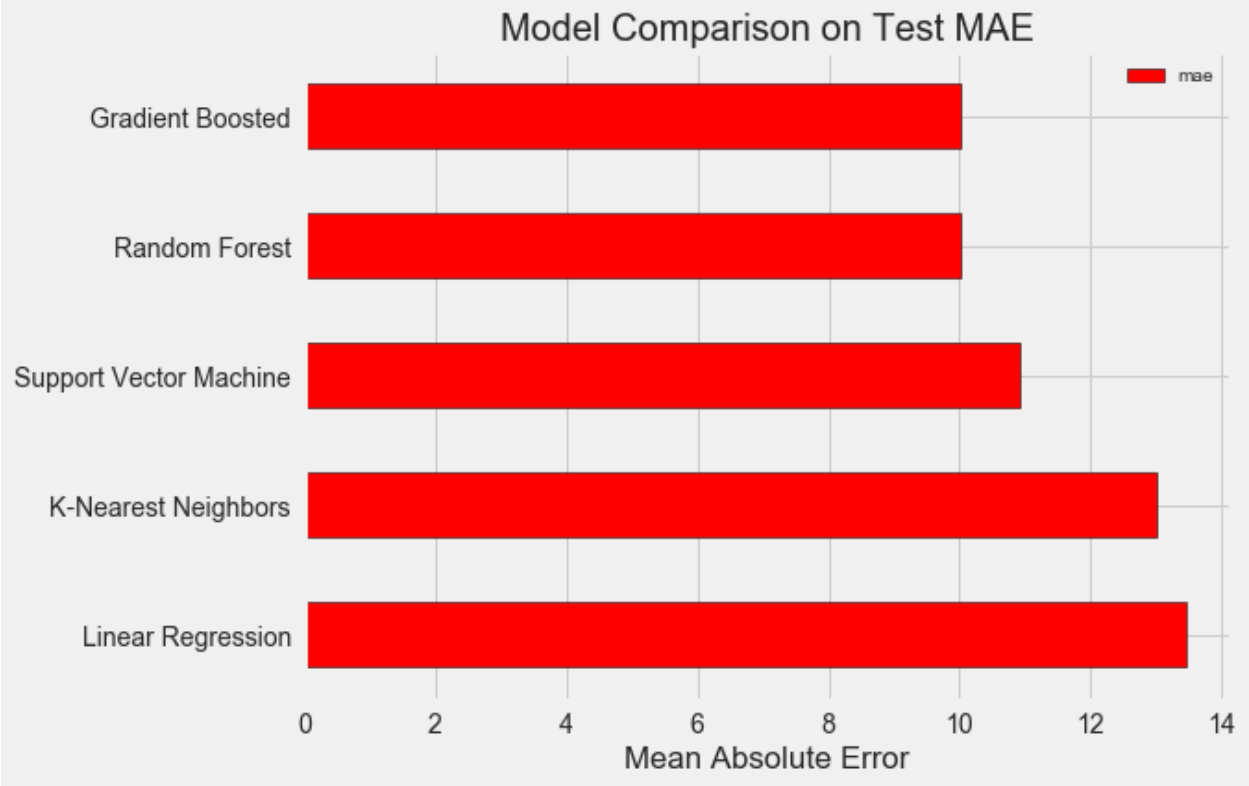
После всех подготовительных работ процесс создания, обучения и прогона моделей относительно прост. Будет использоваться в Python библиотека Scikit-Learn, прекрасно документированная и с продуманным синтаксисом построения моделей.

Иллюстрировать процесс создания, обучения (.fit) и тестирования (.predict) буду с помощью градиентного бустинга:

```
from sklearn.ensemble import GradientBoostingRegressor

# Create the model
gradient_boosted = GradientBoostingRegressor()

# Fit the model on the training data
gradient_boosted.fit(X, y)

# Make predictions on the test data
predictions = gradient_boosted.predict(X_test)
```

```
# Evaluate the model
mae = np.mean(abs(predictions - y_test))

print('Gradient Boosted Performance on the test set: MAE = %0.4f' % mae)

Gradient Boosted Performance on the test set: MAE = 10.0132
```

Всего по одной строке кода на создание, обучение и тестирование. Для построения других моделей воспользуюсь тем же синтаксисом, меняя только название алгоритма.



Чтобы объективно оценивать модели, с помощью медианного значения цели вычислилю базовый уровень. Полученное значение равно 24,5. А полученные результаты оказались значительно лучше, так что задачу можно решить с помощью машинного обучения.

В моем случае градиентный бустинг (MAE = 10,013) оказался чуть лучше

«случайного леса» (10,014 MAE). Хотя эти результаты нельзя считать абсолютно честными, потому что для гиперпараметров по большей части используются значения по умолчанию. Эффективность моделей сильно зависит от этих настроек, особенно в методе опорных векторов. Тем не менее на основании этих результатов выберу градиентный бустинг и стану его оптимизировать.

3. Гиперпараметрическая оптимизация модели

После выбора модели можно оптимизировать её под решаемую задачу, настраивая гиперпараметры.

- Гиперпараметры модели можно считать настройками алгоритма, которые задаются до начала его обучения. Например, гиперпараметром является количество деревьев в «случайном лесе», или количество соседей в методе k-ближайших соседей.
- Параметры модели — то, что она узнаёт в ходе обучения, например, веса в линейной регрессии.

  Управляя гиперпараметром, можно повлиять на результаты работы модели, меняя баланс между её недообучением и переобучением. Недообучением называется ситуация, когда модель недостаточно сложна (у неё слишком мало степеней свободы) для изучения соответствия признаков и цели. У недообученной модели высокое смещение (bias), которое можно скорректировать посредством усложнения модели.

  Переобучением называется ситуация, когда модель по сути запоминает учебные данные. У переобученной модели высокая дисперсия (variance), которую можно скорректировать с помощью ограничения сложности модели посредством регуляризации. Как недообученная, так и переобученная модель не сможет хорошо обобщить

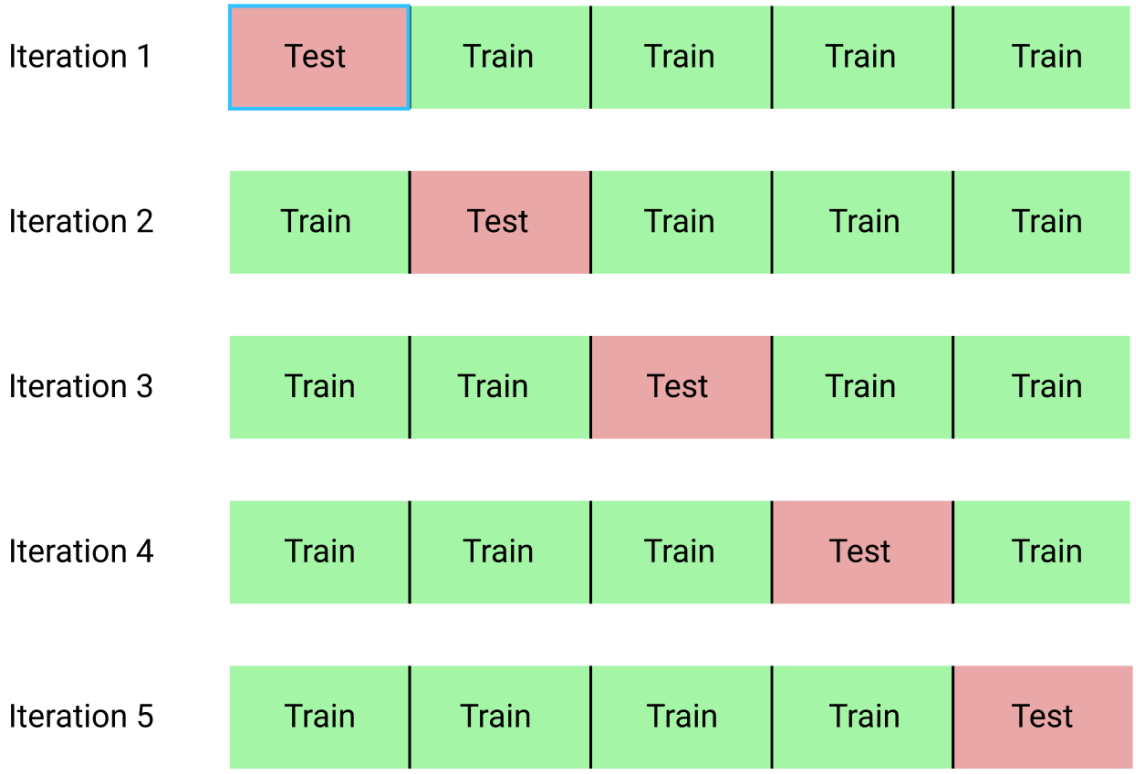
тестовые данные.

Трудность выбора правильных гиперпараметров заключается в том, что для каждой задачи будет уникальный оптимальный набор. Поэтому единственный способ выбрать наилучшие настройки — попробовать разные комбинации на новом датасете. К счастью, в Scikit-Learn есть ряд методов, позволяющих эффективно оценивать гиперпараметры. Более того, в проектах вроде TPOT делаются попытки оптимизировать поиск гиперпараметров с помощью таких подходов, как генетическое программирование.

Случайный поиск с перекрёстной проверкой

- Случайный поиск — методика выбора гиперпараметров. Определяется сетку, а потом из неё случайно выбираются различные комбинации, в отличие от сеточного поиска (grid search), при котором последовательно пробуется каждая комбинация.
- Перекрёстной проверкой называется способ оценки выбранной комбинации гиперпараметров. Вместо разделения данных на обучающий и тестовый наборы, что уменьшает количество доступных для обучения данных, воспользуюсь k-блочной перекрёстной проверкой (K-Fold Cross Validation). Для этого разделю обучающие данные на k блоков, а затем прогоню итеративный процесс, в ходе которого сначала обучу модель на k-1 блоках, а затем сравню результат при обучении на k-ом блоке. Буду повторять процесс k раз, и в конце получу среднее значение ошибки для каждой итерации. Это и будет финальная оценка.

Вот наглядная иллюстрация k-блочной перекрёстной проверки при k = 5:



Весь процесс случайного поиска с перекрёстной проверкой выглядит так:

1. Задаём сетку гиперпараметров.
2. Случайно выбираем комбинацию гиперпараметров.
3. Создаём модель с использованием этой комбинации.
4. Оцениваем результат работы модели с помощью k-блочной перекрёстной проверки.
5. Решаем, какие гиперпараметры дают лучший результат.

Будет использоваться регрессионная модель на основе градиентного бустинга. Это сборный метод, то есть модель состоит из многочисленных «слабых учеников» (weak learners), в данном случае из отдельных деревьев решений (decision trees). Если в пакетных алгоритмах вроде «случайного леса» ученики обучаются параллельно, а затем методом голосования выбирается результат прогнозирования, то в boosting-алгоритмах вроде

градиентного бустинга ученики обучаются последовательно, и каждый из них «сосредотачивается» на ошибках, сделанных предшественниками.

В последние годы boosting-алгоритмы стали популярны и часто побеждают на соревнованиях по машинному обучению. Градиентный бустинг — одна из реализаций, в которой для минимизации стоимости функции применяется градиентный спуск (Gradient Descent). Реализация градиентного бустинга в Scikit-Learn считается не такой эффективной, как в других библиотеках, например, в XGBoost, но она неплохо работает на маленьких датасетах и выдаёт достаточно точные прогнозы.

В регрессии с помощью градиентного бустинга есть много гиперпараметров, которые нужно настраивать. Буду оптимизировать:

- loss: минимизация функции потерь;
- n_estimators: количество используемых слабых деревьев решений (decision trees);
- max_depth: максимальная глубина каждого дерева решений;
- min_samples_leaf: минимальное количество примеров, которые должны быть в «листовом» (leaf) узле дерева решений;
- min_samples_split: минимальное количество примеров, которые нужны для разделения узла дерева решений;
- max_features: максимальное количество признаков, которые используются для разделения узлов.

В этом коде создается сетка из гиперпараметров, затем создаётся объект RandomizedSearchCV и ищется с помощью 4-блочной перекрёстной проверки по 25 разным комбинациям гиперпараметров:

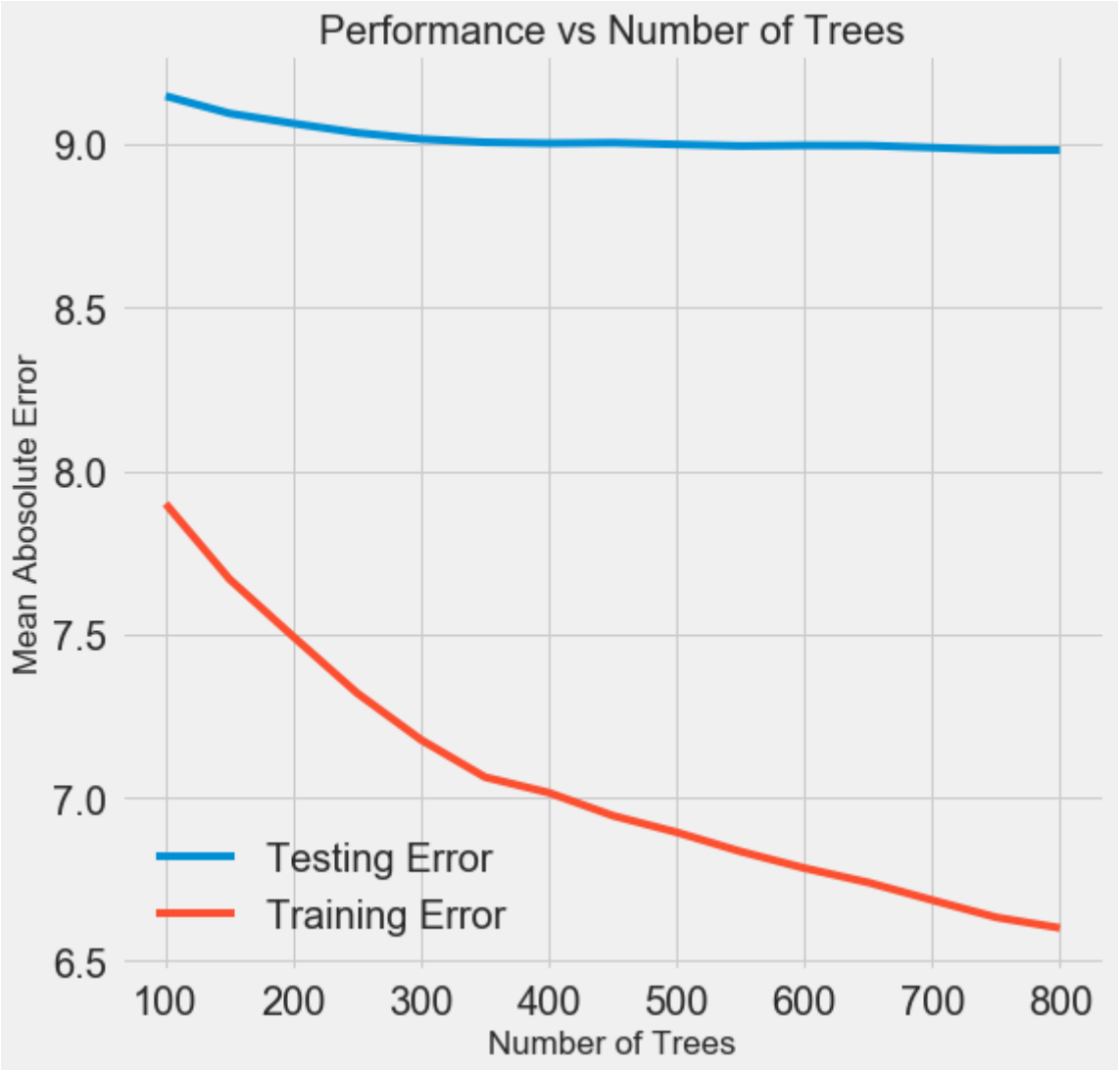
```
# Loss function to be optimized
```

```
loss = ['ls', 'lad', 'huber']

# Number of trees used in the boosting process
n_estimators = [100, 500, 900, 1100, 1500]

# Maximum depth of each tree
max_depth = [2, 3, 5, 10, 15]

# Minimum number of samples per leaf
min_samples_leaf = [1, 2, 4, 6, 8]

# Minimum number of samples to split a node
min_samples_split = [2, 4, 6, 10]

# Maximum number of features to consider for making splits
max_features = ['auto', 'sqrt', 'log2', None]

# Define the grid of hyperparameters to search
hyperparameter_grid = {'loss': loss,
    'n_estimators': n_estimators,
    'max_depth': max_depth,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'max_features': max_features}

# Create the model to use for hyperparameter tuning
model = GradientBoostingRegressor(random_state = 42)

# Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=model,
    param_distributions=hyperparameter_grid,
    cv=4, n_iter=25,
    scoring = 'neg_mean_absolute_error',
    n_jobs = -1, verbose = 1,
    return_train_score = True,
    random_state=42)

# Fit on the training data
random_cv.fit(X, y) After performing the search, we can inspect the RandomizedSearchCV object to find the best
model:
```

```
# Find the best combination of settings
random_cv.best_estimator_
GradientBoostingRegressor(loss='lad', max_depth=5,
    max_features=None,
    min_samples_leaf=6,
    min_samples_split=6,
    n_estimators=500)
```

Эти результаты можно использовать для сеточного поиска, выбирая для сетки параметры, которые близки к этим оптимальным значениям. Но дальнейшая настройка вряд ли существенно улучшит модель. Есть общее правило: грамотное конструирование признаков окажет на точность модели куда большее влияние, чем самая дорогая гиперпараметрическая настройка. Это закон убывания доходности применительно к машинному обучению: конструирование признаков даёт наивысшую отдачу, а гиперпараметрическая настройка приносит лишь скромную выгоду.

Для изменения количества оценщиков (estimator) (деревьев решений) с сохранением значений других гиперпараметров можно поставить один эксперимент, который продемонстрирует роль этой настройки. Вот что получилось в результате:



С ростом количества используемых моделью деревьев снижается уровень ошибок в ходе обучения и тестирования. Но ошибки при обучении снижаются куда быстрее, и в результате модель переобучается: показывает отличные результаты на обучающих данных, но на тестовых работает хуже.

На тестовых данных точность всегда снижается (ведь модель видит правильные ответы для учебного датасета), но существенное падение говорит о переобучении. Решить эту проблему можно с помощью увеличения объёма обучающих данных или уменьшения сложности модели с помощью гиперпараметров.

Для финальной модели возьму 800 оценщиков, потому что это даст

самый низкий уровень ошибки при перекрёстной проверке.

4. Оценка с помощью тестовых данных

Модель никоим образом не получала доступ к тестовым данным в ходе обучения. Поэтому точность при работе с тестовыми данными мы можем использовать в роли индикатора качества модели, когда её допустят к реальным задачам.

Скормлю модели тестовые данные и вычислю ошибку. Вот сравнение результатов алгоритма градиентного бустинга по умолчанию и настроенной модели:

```
# Make predictions on the test set using default and final model
default_pred = default_model.predict(X_test)
final_pred = final_model.predict(X_test)
Default model performance on the test set: MAE = 10.0118.
Final model performance on the test set: MAE = 9.0446.
```

Гиперпараметрическая настройка помогла улучшить точность модели примерно на 10 %. В зависимости от ситуации это может быть очень значительное улучшение, но требующее немало времени.

Сравнить длительность обучения обеих моделей можно с помощью команды `%timeit` в Jupyter Notebooks. Сначала измерю длительность работы модели по умолчанию:

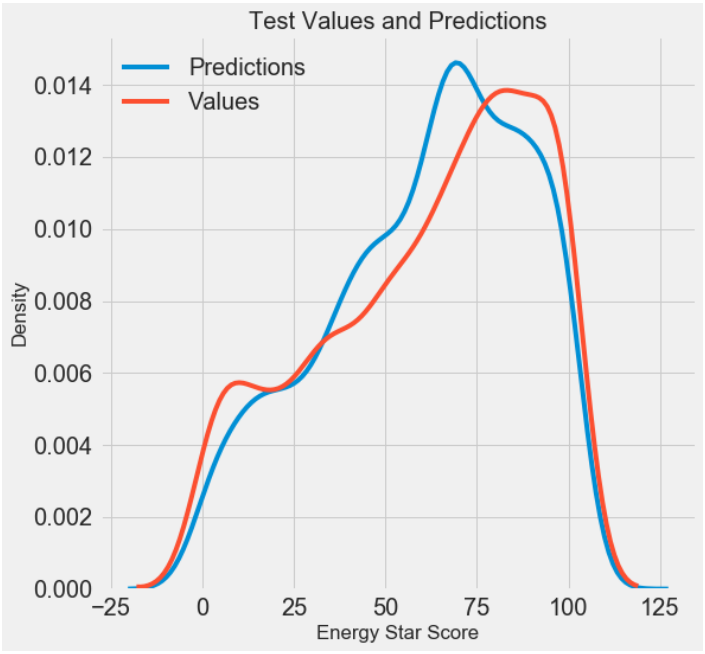
```
%%timeit -n 1 -r 5
default_model.fit(X, y)
1.09 s ± 153 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

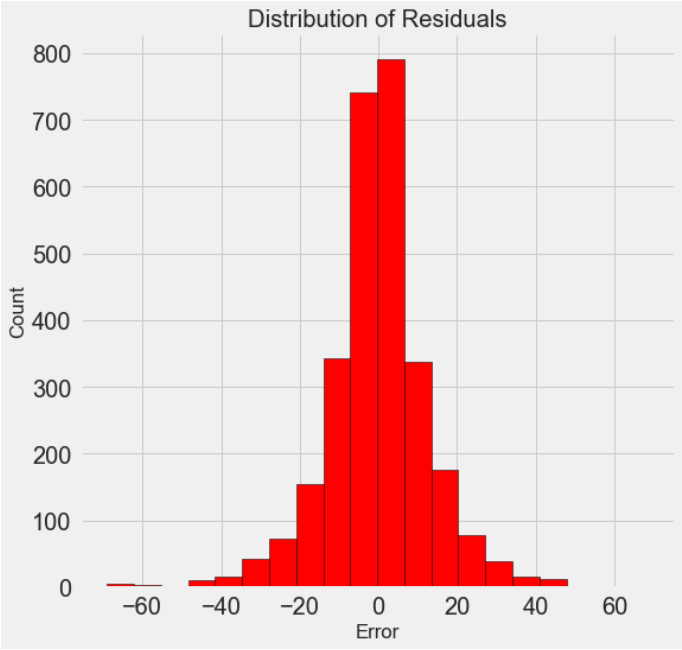
А вот настроенная модель уже выполняется сильно медленнее:

```
%%timeit -n 1 -r 5
final_model.fit(X, y)
12.1 s ± 1.33 s per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

Эта ситуация иллюстрирует фундаментальный аспект машинного обучения: всё дело в компромиссах. Постоянно приходится выбирать баланс между точностью и интерпретируемостью, между смещением и дисперсией, между точностью и временем работы, и так далее. Правильное сочетание полностью определяется конкретной задачей. В моем случае 12-кратное увеличение длительности работы в относительном выражении велико, но в абсолютном — незначительно.

Были получены финальные результаты прогнозирования, Слева показан график плотности прогнозных и реальных значений, справа — гистограмма погрешности:





Прогноз модели неплохо повторяет распределение реальных значений, при этом на обучающих данных пик плотности расположен ближе к медианному значению (66), чем к реальному пику плотности (около 100). Погрешности имеют почти нормальное распределение, хотя есть несколько больших отрицательных значений, когда прогноз модели сильно отличается от реальных данных.

5. Список литературы

1. К. Элбон. Машинное обучение с использованием Python. Сборник рецептов— Санкт-Петербург, Вильямс, 2019 – 200 с.

2. Ю.Е. Гапанюк. Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных. — [Электронный ресурс] — Режим доступа. — URL: https://nbviewer.org/github/ugapanyuk/ml_course_2020/blob/master/common/notebooks/missing/handling_missing_norm.ipynb (Дата обращения: 15.12.2021).

3. А. Мюллер, С. Гвидо. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными. — Санкт-Петербург, Вильямс, 2020 – 480 с.