МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа по курсу «Методы, средства и технологии мультимедиа»

Студент: А.С. Федоров

Группа: М8О-407Б-19

Преподаватель: Б.В. Вишняков

Дата:

Оценка:

Подпись:

Москва, 2022

Курсовая работа

Задача: Выбрать задачу (классификация или регрессия), датасет и метрику качества. Выбранные данные необходимо визуализировать и проанализировать. После этого выполнить препроцессинг. Затем реализовать алгоритм (случайный лес) для задачи (регрессии), проверить качество обучения, сравнить с моделью из sklearn.

Описание данных

Ссылка на данные: https://www.kaggle.com/datasets/mohannapd/mobile-price-prediction

Датасет содержит информацию о клиентах автосалонов США. Таблица имеет 500 строк и 9 колонок признаков. Целевым признаком является сумма, которую клиент собирается потратить на новый автомобиль. Признаками, по которым требуется определить целевой, являются:

- Имя клиента (номинальный)
- Email (номинальный)
- Страна (номинальный)
- Пол (бинарный)
- Возраст (числовой)
- Годовой оклад (числовой)
- Задолженность по кредитной карте (числовой)
- Чистые активы (числовой)

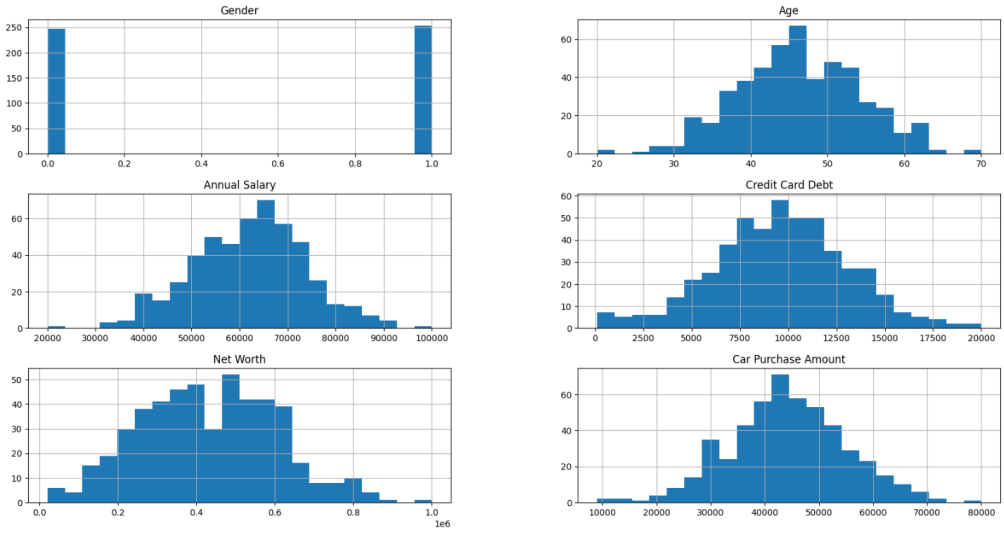
Данные не содержат пропусков.

Предобработка и анализ данных

Для обучения не буду использовать информацию об имени и почте из предположения, что имя и почта никак не влияет на сумму, которую клиент собирается потратить. Также, так как все клиенты из США, не буду использовать колонку со страной.

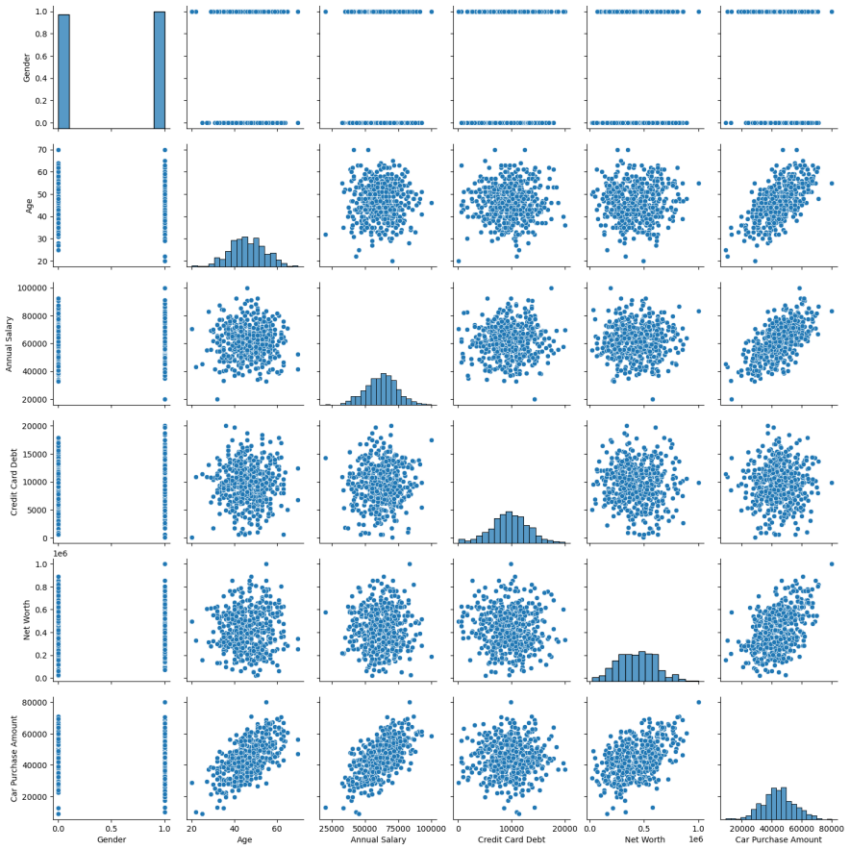
Проанализирую данные, взглянув на гистограммы признаков, проекции объектов на плоскости всех возможных пар признаков и матрицу корреляции.

Гистограммы



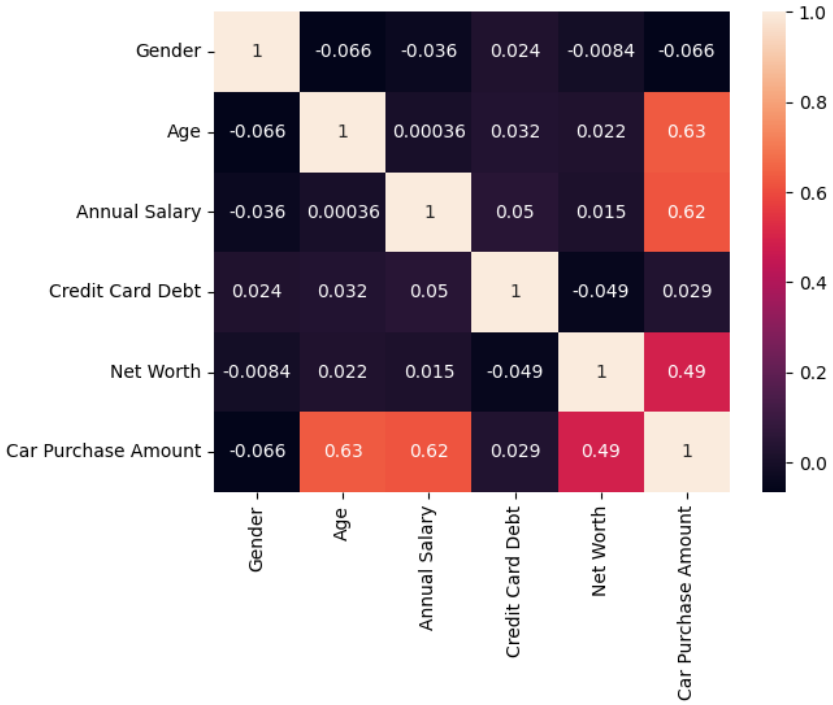
Ярко выраженных всплесков на гистограммах нет.

Проекции на плоскости всех возможных пар признаков



Похоже, что признаки независимы между собой. Из визуального анализа можно предположить, что есть зависимость целевого признака от возраста и годового дохода клиента.

Матрица корреляции



Предположение, сделанное ранее, подтвердилось. Признаки мало коррелируют между собой, что хорошо, так как нет мультиколлинеарности. Также выявлена зависимость суммы, которую хочет потратить клиент на новый автомобиль от его возраста, годового дохода и стоимости активов. Это хорошо, так как повышает вероятность того, что модели будет достаточно данной совокупности признаков, чтобы уловить закономерность.

Так как в качестве алгоритма для предсказаний был выбран случайный лес, нормировать значения не требуется.

Разобью данные для обучения и валидации. 80% и 20% соответственно.

Описание алгоритма случайный лес

Основная идея заключается в использовании ансамбля решающих деревьев. Все деревья ансамбля обучены на случайных подвыборках из изначальной, что делает их предсказания немного отличающимися. Предсказание всей модели агрегируется из предсказаний каждого отдельно взятого дерева. В случае задачи регрессии берется среднее (или медианное) от предсказаний всех деревьев. Хоть каждое дерево в отдельности дает невысокое качество предсказаний, их больше количество и разнообразие способно сильно улучшить результат. Аналогией из реального мира может послужить совокупность судей, каждый из которых руководствуется своим, немного отличающемся от остальных,

опытом. Если отдельно взятый судья предсказывает в целом хорошо, но недостаточно, то разнообразный набор таких судей, в совокупности, может предсказывать гораздо качественнее.

Решающее дерево

Решающее дерево – это бинарное дерево, узлами которого являются решающие правила. Решающее правило состоит из признака, по которому происходит деление и значения. Все объекты, признак которых оказался меньше барьера отправляются к левым детям, а остальные к правым. Если узел не имеет детей, то вместо того, чтобы бить выборку надвое, он предсказывает значение целевого признака. В задачи регрессии это либо среднее, либо медианное значение целевого признака от пришедшей сверху выборки.

При обучении дерева выбор барьерного значения в узле производится исходя из информативности узла до разбиения и после. Прирост информативности можно вычислить по формуле:

$$H(X_m) - \frac{|X_l|}{|X_m|} H(X_l) - \frac{|X_r|}{|X_m|} H(X_r),$$

где H – функция информативности множества, X_m, X_l, X_r – выборка до разбиения, левая и правая часть после разбиения соответственно.

Для задачи регрессии информативность можно считать по разным критериям. В своей реализации буду использовать MSE и MAE, они вычисляются по таким формулам.

MSE:

$$H(X_m) = \sum_{y_i \in Y} \frac{(y_i - \bar{y})^2}{|X_m|},$$

где y_i – значение целевого признака, $\bar{y}$ – среднее значение целевых признаков выборки.

где у с палочкой среднее значение признака по выборке. В своей реализации попробую использовать не только среднее, но и медианное.

MAE:

$$H(X_m) = \sum_{y_i \in Y} \frac{|y_i - \bar{y}|}{|X_m|},$$

где $\bar{y}$ – медианное значение целевых признаков.

Аналогично MSE попробую использовать не только медианное значение, но и среднее.

Значение признака, для которого прирост информативности оказался максимальным запоминается узлом, выборка делится согласно этому значению и ее половины передаются дочерним узлам для выполнения той же процедуры рекурсивно. Условием остановки рекурсии может быть ограничение глубины дерева, ограничение минимального числа объектов в листе или ограничение максимального числа признаков, по которым произошло разбиение и так далее. В своей реализации буду использовать только первые два ограничения.

Реализация алгоритма

Реализую алгоритм решающего дерева, согласно описанию выше:

```
def mse(y, estimation):
    return (1/len(y))*sum((y-estimation)**2)

def mae(y, estimation):
    return (1/len(y))*sum(y-estimation)

class Decision_node:
        def __init__(self):
            self.prediction = None
            self.Value = None
            self.Left = None
            self.Right = None

class Decision_tree(BaseEstimator, ClassifierMixin):
    def fit_node(self, node, X, Y, features, depth, min_leaf_size, criterion,
estimation_type = 'mean'):
        x = X
        y = Y
        n = len(x)

        if estimation_type == 'mean':
            node.prediction = np.mean(y)
        if estimation_type == 'median':
            node.prediction = np.median(y)

        if depth == 1 or len(x) == 1 or len(x) == 0:
            return

        h = criterion(y, node.prediction)

        max_gain = -1
        left_ids = []
        right_ids = []
```

```
        for feature in features:
            sorted_Ids = x[:, feature].argsort()

            left_n = 0
            for i in range(n-1):
                left_n += 1

                left_y = y[sorted_Ids[:(i+1)]]
                right_y = y[sorted_Ids[(i+1):]]

                left_prediction = np.mean(left_y)
                right_prediction = np.mean(right_y)

                left_h = criterion(left_y, left_prediction)
                right_h = criterion(right_y, right_prediction)

                gain = h-(left_n*left_h+(n-left_n)*right_h)/n

                if gain > max_gain:
                    max_gain = gain
                    node.feature = feature
                    node.value = (x[sorted_Ids[i], feature]+x[sorted_Ids[i+1],
feature])/2
                    left_ids = sorted_Ids[:i+1]
                    right_ids = sorted_Ids[i+1:]

        if len(left_ids) < min_leaf_size or len(right_ids) < min_leaf_size:
            return

        node.Left = Decision_node()
        node.Right = Decision_node()


        self.prediction = None
        self.fit_node(node.Left, x[left_ids], y[left_ids], features, depth-1,
min_leaf_size, criterion, estimation_type)
        self.fit_node(node.Right, x[right_ids], y[right_ids], features, depth-1,
min_leaf_size, criterion, estimation_type)

    def __init_(self):
        pass

    def fit(self, X, Y, features, depth, min_leaf_size, criterion, estimation_type):
        classes = len(np.unique(Y))
        self.root = Decision_node()
```

```
        self.fit_node(self.root, X, Y, features, depth, min_leaf_size, criterion,
estimation_type)

    def predict(self, X):
        prediction = np.empty(X.shape[0])
        for i, object in enumerate(X):
            current_node = self.root
            while current_node.Left != None and current_node.Right != None:
                if object[current_node.feature] < current_node.value:
                    current_node = current_node.Left
                else:
                    current_node = current_node.Right
            prediction[i] = current_node.prediction
        return prediction
```

Каждый Decision_node имеет информацию о левом, правом ребенке, признаке, барьерном значении признака и предсказании. Построение происходит рекурсивно, начиная с корневого узла. Предсказание также происходит рекурсивно, со спуском до какого-либо листа дерева.

На основе реализации решающего дерева реализую случайный лес:

```
class Random_forest(BaseEstimator, ClassifierMixin):
    def __init__(self, n_estimators = 10, criterion = mse, max_depth = 20,
min_samples_leaf = 2, estimation_type = 'mean', max_features = 'all',
data_portion = 1.0, answer_assumption = 'median'):
        self.n_estimators = n_estimators
        self.criterion = criterion
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.estimation_type = estimation_type
        self.max_features = max_features
        self.data_portion = data_portion
        self.answer_assumption = answer_assumption

    def fit(self, X, Y):
        features = np.arange(X.shape[1])

        self.forest = []
        for _ in range(self.n_estimators):
            np.random.shuffle(features)
            tree = Decision_tree()
            slice = int(self.data_portion*X.shape[0])-1
            portion_ids = np.random.permutation(X.shape[0])[:slice]

            if self.max_features == 'sqrt':
```

```
                tree.fit(X[portion_ids], Y[portion_ids],
features[:int(np.floor(np.sqrt(len(features))))], self.max_depth,
self.min_samples_leaf, self.criterion, self.estimation_type)
            if self.max_features == 'all':
                tree.fit(X[portion_ids], Y[portion_ids], features,
self.max_depth, self.min_samples_leaf, self.criterion, self.estimation_type)
            self.forest.append(tree)

    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.forest])
        if self.answer_assumption == 'median':
            forest_predictions = np.median(predictions, axis=0)
        if self.answer_assumption == 'mean':
            forest_predictions = np.mean(predictions, axis=0)
        return forest_predictions
```

В своей сути, случайный лес реализован списком решающих деревьев, обученных на случайных подвыборках объектов из обучающей выборки. При вызове метода для предсказания, объект опрашивает все деревья из списка и возвращает либо медианное, либо среднее значение от всех предсказаний.

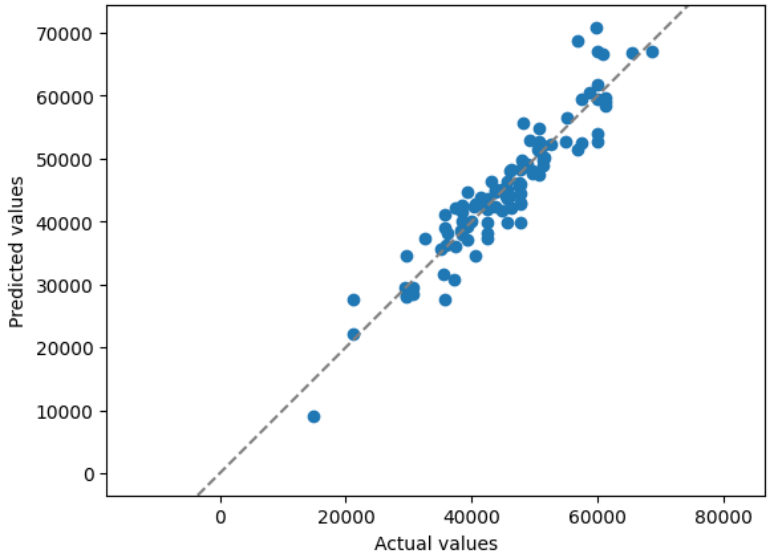
Метрика качества

Целевой метрикой выберу MSE, так как она подчеркивает большие ошибки над малыми. Ее буду использовать как целевую и при подборе оптимальных гиперпараметров на кросс-валидации. Помимо MSE буду смотреть на MAE, максимальное отклонение и коэффициент детерминации.

Демонстрация полученных результатов

Обучу случайный лес с параметрами по умолчанию и взгляну на метрики и построю график предсказаний относительно истинных значений.

- Max error: 11809.728575000001
- MAE: 2818.5079360435384
- MSE: 13638113.728929207
- R^2: 0.8601830414156021



Попытаюсь улучшить результат с помощью кросс-валидации.

Список перебираемых параметров и их значений:

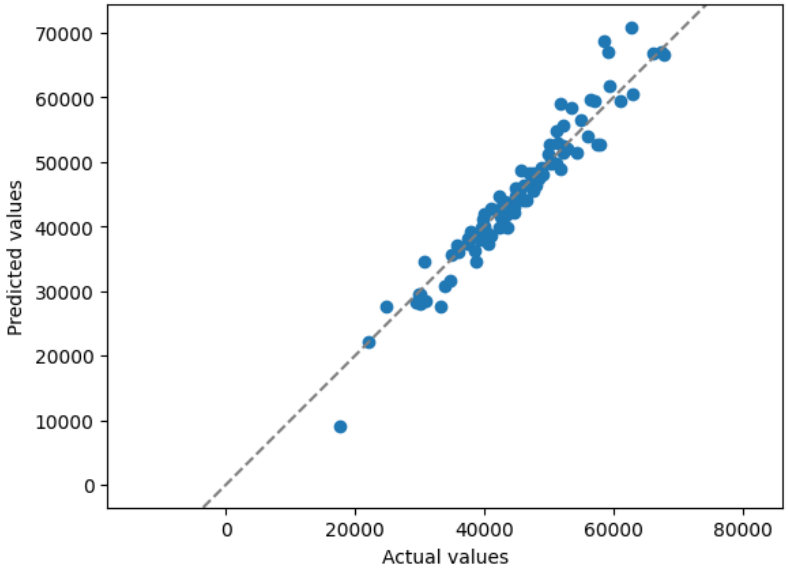
- n_estimators: [10, 25, 50];
- criterion: [mse, mae];
- max_depth: [5, 10, 15, 20];
- min_samples_leaf: [1, 2, 3];
- estimation_type: [mean, median];
- max_features: [sqrt, all];
- data_portion: [0.25, 0.50, 0.75, 1.0];
- answer_assumption: [mean, median]

Список лучших значений гиперпараметров:

- answer_assumption: mean;
- criterion: mse;
- data_portion: 0.75;
- estimation_type: median;
- max_depth: 20;
    - max_features: all;
    - min_samples_leaf: 1;
    - n_estimators: 50

Результаты предсказаний новой модели:

- Max error: 10012.816998400012
- MAE: 1911.6664294499972
- MSE: 7472130.436181851
- R^2: 0.9160774191891411



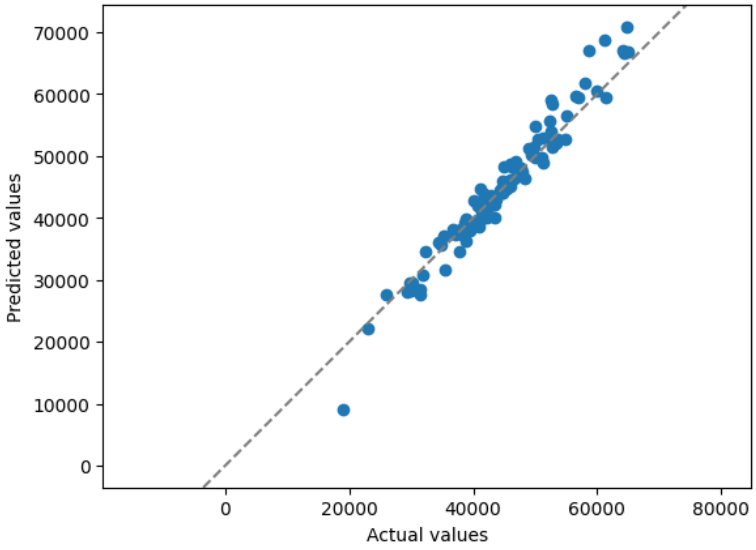
Все метрики стали лучше. Визуально видно, что точки стали более собранными вдоль прямой.

Сравнение с sklearn

Сравню реализацию такого же алгоритма в sklearn. Гиперпараметры переберу аналогично своей реализации с такими же списками возможных значений.

Результат предсказаний sklearn:

- Max error: 9987.988681199997
- MAE: 1838.4489051439994
- MSE: 6480470.792447598
- R^2: 0.9209183090528335



Результаты модели из sklearn оказались лучше, но не слишком сильно. Картина, в целом, схожая. Некоторые метрики отличаются совсем немного.

Вывод

В ходе выполнения курсовой работы был взят алгоритм случайного леса из лабораторной работы, улучшен добавлением возможности обучать деревья на случайных выборках из обучающих данных, а также адаптирован для задач регрессии. Результаты предсказаний оказались чуть хуже, чем у реализации того же самого алгоритма в sklearn. Возможно, причиной может быть ситуация, когда некоторые объекты их обучающих данных ввиду случая не попали ни в одну выборку, на которой обучалось какое-либо дерево, что означает, что модель не учла полный объем информации для обучения. Также стоит отметить, что реализация sklearn прошла процесс кросс-валидации гораздо быстрее, предположительно не только потому, что у модели перебиралось меньшее количество всех возможных комбинаций гиперпараметров, но и благодаря более оптимизированной реализации.