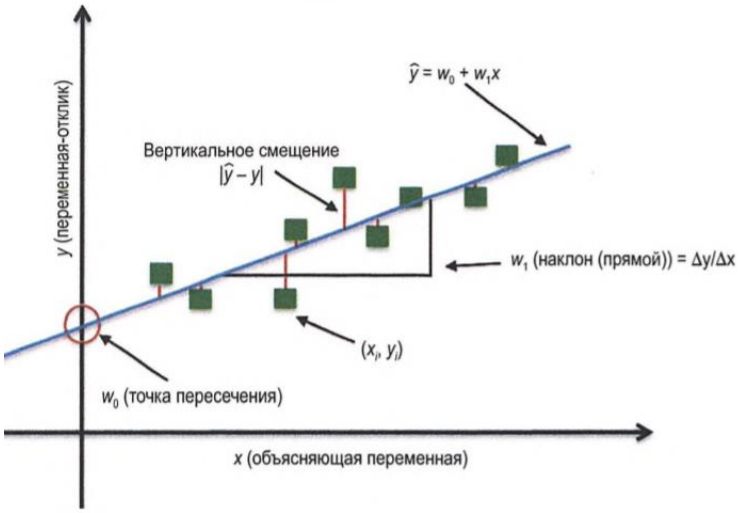
Простая регрессионная модель

Задача простой (одномерной) линейной регрессии состоит в том, чтобы смоделировать связь между единственным признаком (объясняющей переменной x) и откликом (целевой переменной y). Уравнение линейной модели с одной объясняющей переменной определяется следующим образом:

$$y = \omega_0 + \omega_1 x,$$

здесь вес ω_0 – это точка пересечения оси Y и ω_1 – коэффициент объясняющей переменной. Наша задача – извлечь веса линейного уравнения, чтобы описать связь между объясняющей и целевой переменными, которую затем можно использовать для предсказания откликов новых объясняющих переменных, не бывших частью обучающего набора данных.

Основываясь на линейном уравнении, определенном нами выше, линейная регрессия может пониматься как нахождение оптимально подогнанной прямой линии, проходящей через точки образцов данных.



Оптимально подогнанная прямая линия также называется *линией регрессии*, а вертикальные прямые от линии регрессии до точек данных – это так называемые *смещения* – ошибки нашего предсказания. Частный случай, состоящий из одной объясняющей переменной, *называется простой линейной регрессией*, но, разумеется, мы также можем обобщить линейную регрессионную модель на две и более объясняющих переменных. Отсюда этот процесс называется множественной линейной регрессией:

$$y = \omega_0 x_0 + \omega_1 x_1 + \cdots + \omega_m x_m.$$

Подгонка линейной модели методом RANSAC

Выбросы могут оказывать сильное воздействие на линейные регрессионные модели. В определенных ситуациях незначительное подмножество наших данных может иметь большой эффект на оцениваемые модельные коэффициенты. Для обнаружения выбросов используются

разнообразные статистические тесты, рассмотрение которых выходит за рамки данной книги. Однако во время удаления выбросов всегда требуется наше собственное суждение как аналитика данных, а также наше знание предметной области. В качестве альтернативы исключению выбросов рассмотрим устойчивый метод регрессии с использованием алгоритма **RANSAC**, который выполняет подгонку регрессионной модели на подмножестве данных, так называемых *не-выбросах* (inliers), т.е. хороших точках данных.

Итеративный алгоритм RANSAC можно резюмировать следующим образом:

1. Выбрать случайное число образцов в качестве *не-выбросов* и выполнить подгонку модели.
2. Проверить все остальные точки данных на подогнанной модели и добавить те точки, которые попадают в пределы заданного пользователями допуска для не-выбросов.
3. Выполнить повторную подгонку модели с использованием всех не-выбросов.
4. Оценить ошибку подогнанной модели относительно *не-выбросов*.
5. Завершить алгоритм, в случае если качество соответствует определенному заданному пользователем порогу либо если было достигнуто фиксированное число итераций; в противном случае вернуться к шагу 1.

Пример:

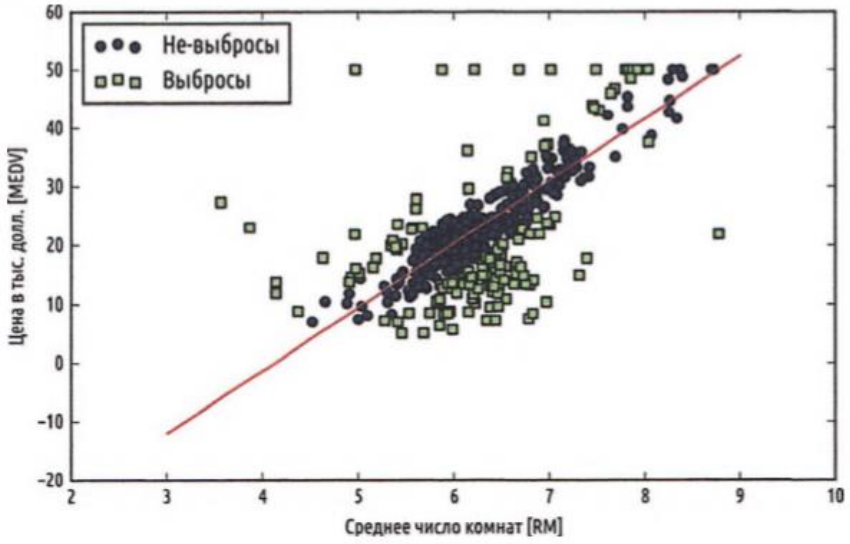
```
X = df[['RM']].values
y = df['MEDV'].values

from sklearn.linear_model import RANSACRegressor
ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100,
                         min_samples=50,
                         residual_metric=lambda x: np.sum(np.abs(x), axis=1),
                         residual_threshold=5.0,
                         random_state=0)
ransac.fit(X, y)
```

По умолчанию в библиотеке scikit-learn для отбора порога не-выбросов используется оценка на основе медианного абсолютного отклонения (оценка MAD) целевых значений *y*, где MAD – это аббревиатура для Median Absolute Deviation. Однако выбор надлежащего значения для порога не-выбросов зависит от конкретной задачи, в чем кроется один из недостатков алгоритма RANSAC.

После подгонки модели RANSAC получим не-выбросы и выбросы из подогнанной линейной регрессионной модели RANSAC и построим совместный график с линейной подгонкой:

```
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
line_X = np.arange(3, 10, 1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])
plt.scatter(X[inlier_mask], y[inlier_mask],
            c='blue', marker='o', label='Не-выбросы')
plt.scatter(X[outlier_mask], y[outlier_mask],
            c='lightgreen', marker='s', label='Выбросы')
plt.plot(line_X, line_y_ransac, color='red')
plt.xlabel('Среднее число комнат [RM]')
plt.ylabel('Цена в тыс. долл. [MEDV]')
plt.legend(loc='upper left')
plt.show()
```

Как видно на следующем ниже точечном графике, линейная регрессионная модель была подогнана на обнаруженном подмножестве не-выбросов, показанных как круги:



Используя модель RANSAC, мы уменьшили в этом наборе данных потенциальное влияние выбросов, но мы не знаем, имеет ли этот подход положительное влияние на предсказательную способность на ранее не встречавшихся данных.

Применение регуляризованных методов для регрессии

Самыми популярными подходами к регуляризованной линейной регрессии являются так называемый метод *гребневой регрессии* (ridge regression), метод *lasso* (оператор наименьшего абсолютного стягивания и отбора, least absolute shrinkage and selection operator, lasso) и метод *эластичной сети* (elastic net).

Гребневая регрессия (ridge) – это модель с $L2$-штрафом, где к нашей функции стоимости на основе МНК мы просто добавляем квадратичную сумму весов:

$$J(w)_{\text{Гребень}} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \lambda\|w\|_2^2.$$

$$L2: \quad \lambda\|w\|_2^2 = \lambda\sum_{j=1}^{m} w_j^2.$$

Увеличивая значение гиперпараметра λ, мы увеличиваем силу регуляризации и стягиваем веса нашей модели. Отметим, что мы не регуляризуем член уравнения для точки пересечения ω_0. Альтернативный подход, который может привести к разреженным моделям, представлен методом lasso. В зависимости от силы регуляризации определенные веса могут стать нулевыми, что делает метод lasso пригодным для применения *в качестве метода отбора признаков с учителем*:

$$J(w)_{\text{Лассо}} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \lambda w_1.$$

Здесь

$$L1: \quad \lambda\|w\|_1 = \lambda\sum_{j=1}^{m}|w_j|.$$

Однако ограничение метода lasso состоит в том, что он отбирает не более n переменных, если $m>n$. Компромиссом между гребневой регрессией и методом lasso является эластичная сеть, при которой имеются $L1$-штраф для генерирования разреженности и $L2$-штраф для преодоления некоторых ограничений метода lasso, таких как число отобранных переменных.

$$J(w)_{\text{ЭластичнаяСеть}} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \lambda_1\sum_{j=1}^{m} w_j^2 + \lambda_2\sum_{j=1}^{m}|w_j|.$$

Все эти модели регуляризованной регрессии имеются в библиотеке scikit-learn, и их использование подобно обычной регрессионной модели, за исключением того, что нам нужно указать силу регуляризации в параметре λ, например оптимизированным методом k-блочной перекрестной проверки.

Отметим, что сила регуляризации регулируется параметром alpha, который аналогичен параметру λ.

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)

from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1.0)

from sklearn.linear_model import ElasticNet
lasso = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

Например, если установить ll_ratio равным 1.0, то регрессор эластичной сети ElasticNet будет равен lasso. По поводу более подробной информации о разных реализациях линейной регрессии, пожалуйста, обратитесь к документации на http://scikit-learn.org/stable/modules/linear_model.html.

Полиномиальная регрессия

В предыдущих разделах мы допустили наличие между объясняющей переменной и переменной отклика линейной связи. Один из способов объяснить нарушение допущения о линейности связи состоит в том, чтобы использовать модель полиномиальной регрессии, добавив в уравнение полиномиальные члены:

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d.$$

Здесь d обозначает степень полинома. Несмотря на то, что мы можем использовать полиномиальную регрессию для моделирования нелинейных связей, она по- прежнему рассматривается как модель множественной линейной регрессии, ввиду линейных коэффициентов регрессии ω.

Теперь обсудим вопрос использования класса-преобразователя для полиномиальных признаков PolynomialFeatures библиотеки scikit-learn с целью добавления в задачу простой регрессии с одной объясняющей переменной квадратичного члена ($d = 2$) и сравнения полинома с линейной подгонкой. Соответствующие шаги показаны ниже.

1. Добавить член с полиномом второй степени:

```
from sklearn.preprocessing import PolynomialFeatures
X = np.array([258.0, 270.0, 294.0,
              320.0, 342.0, 368.0,
              396.0, 446.0, 480.0,
              586.0])[:, np.newaxis]
y = np.array([236.4, 234.4, 252.8,
              298.6, 314.2, 342.2,
              360.8, 368.0, 391.2,
              390.8])
lr = LinearRegression()
pr = LinearRegression()
quadratic = PolynomialFeatures(degree=2)
X_quad = quadratic.fit_transform(X)
```

2. Для сравнения выполнить подгонку простой линейной регрессионной модели:

```
lr.fit(X, y)
X_fit = np.arange(250,600,10)[:, np.newaxis]
y_lin_fit = lr.predict(X_fit)
```

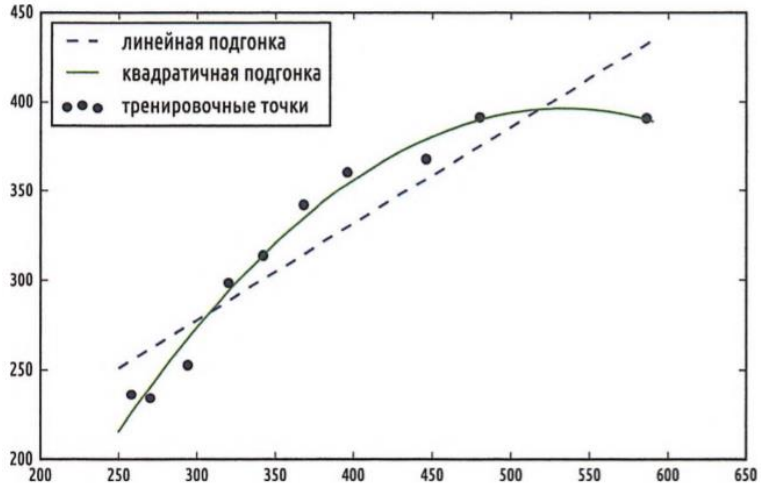
3. Выполнить подгонку множественной регрессионной модели на преобразованных признаках для полиномиальной регрессии:

```
pr.fit(X_quad, y)
y_quad_fit = pr.predict(quadratic.fit_transform(X_fit))
```

4. Построить график результатов:

```
plt.scatter(X, y, label='тренировочные точки')
plt.plot(X_fit, y_lin_fit,
         label='линейная подгонка', linestyle='--')
plt.plot(X_fit, y_quad_fit,
         label='квадратичная подгонка')
plt.legend(loc='upper left')
plt.show()
```

На итоговом графике видно, что полиномиальная подгонка захватывает связь между переменной отклика и объясняющей переменной намного лучше, чем линейная подгонка:



Регрессия на основе дерева решений

Преимущество алгоритма дерева решений состоит в том, что он не требует преобразования признаков, в случае если мы имеем дело с нелинейными данными. Из ранее изученного, помним, что мы строим дерево решений путем итеративного расщепления его узлов, пока листы не станут однородными, либо не будет удовлетворен критерий остановки. Когда мы применяли деревья решений для классификации данных, мы определили *энтропию* как меру неоднородности для установления того, какое расщепление признака максимизирует прирост информации.

Для использования дерева решений для регрессии мы заменим энтропию как меру неоднородности узла t на MSE:

$$I(t) = \mathrm{MSE}(t) = \frac{1}{N_t} \sum_{i \in D_t}^{n} (y^{(i)} - \hat{y}_t)^2.$$

Здесь N_t – число тренировочных образцов в узле t, D_t – тренировочное подмножество в узле i, $y^{(i)}$– истинное целевое значение и y_t – предсказанное целевое значение (эмпирическое среднее):

$$\hat{y}_t = \frac{1}{N} \sum_{i \in D_t} y^{(i)}.$$

В контексте регрессии на основе дерева решений показатель MSE часто также упоминается как внутриузловая дисперсия, и по этой причине критерий расщепления также более известен как *сокращение дисперсии*.