



## 今回の実施内容

線形回帰をスクラッチで作成することが目的である。そのため、テストデータ分割はsklearnのtrain\_test\_splitを利用するデータはhousePriceのデータを利用する

### 平均二乗誤差の計算式

yハット： $\hat{y}$ とは、予測値を表します。予測値と正解との差分を2乗して

$$L(\theta) = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

```
In [1]: # 平均二乗誤差の計算
def MSE(y_pred, y):
    # 行数をカウントする
    num = len(y_pred)
    # 平均二乗誤差を取得する
    # y_pred : 予測値
    # y : 正解ラベル
    mse = np.sum((y_pred - y)**2) / 2 * num
    return mse
```

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [3]: train = pd.read_csv("./Users/tsuneo/kaggle/houseprice/train.csv")
test = pd.read_csv("./Users/tsuneo/kaggle/houseprice/test.csv")
```

```
In [4]: # 訓練データを2変数だけに固定
X = train[["GrLivArea", "YearBuilt"]].values
X.shape
```

```
Out[4]: (1460, 2)
```

```
In [5]: # バイアス対応のため、値が1の列を、最後の列に追加
X = np.hstack((X, np.ones((X.shape[0], 1))))
X.shape
```

```
Out[5]: (1460, 3)
```

```
In [6]: # 正解ラベル(教師データ)を作成する
y = train[["SalePrice"]].values
```

### データセットの確認

次の組み合わせの散布図を描く。

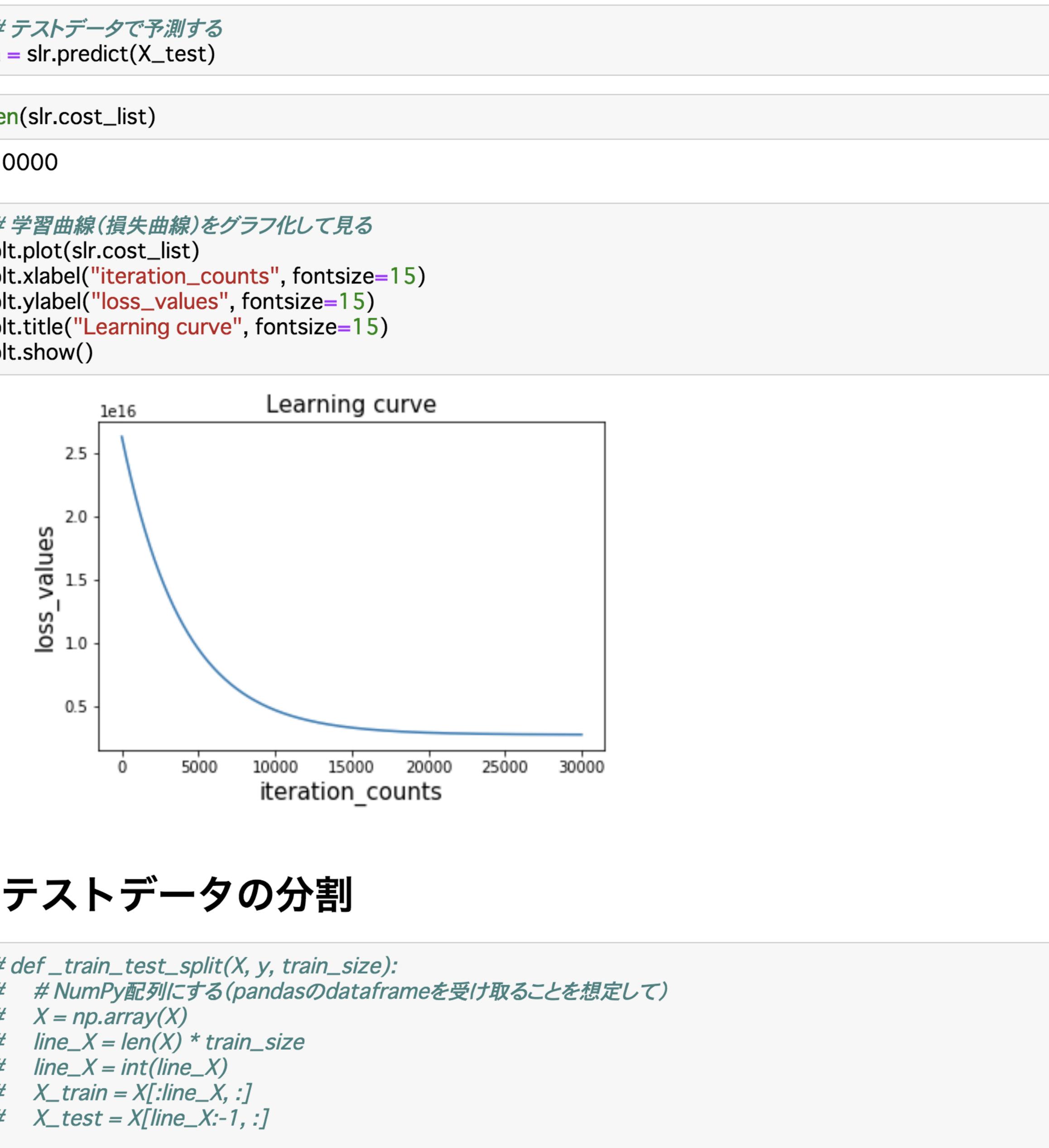
GrLivAreaとSalePrice

GrLivAreaとYearBuilt

```
In [7]: # 描画する
plt.figure(figsize=(10,5))
plt.subplot(1,2)
plt.xlabel("GrLivArea", fontsize=15)
plt.ylabel("SalePrice", fontsize=15)
plt.title("Relationship between GrLivArea and SalePrice", fontsize=12)
plt.scatter(X[:, 0], y)

plt.subplot(1,2)
plt.xlabel("GrLivArea", fontsize=15)
plt.ylabel("YearBuilt", fontsize=15)
plt.title("Relationship between GrLivArea and YearBuilt", fontsize=12)
plt.scatter(X[:, 0], X[:, 1])

plt.tight_layout()
plt.show()
```



```
In [8]: # データ分割のためのモジュールをインポートする
from sklearn.model_selection import train_test_split
```

```
In [9]: # 本当は自分で作成すべきだが、今回はsklearnを利用する
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
In [10]: print("X_trainとX_testのshape", X_train.shape, X_test.shape)
print("y_trainとy_testのshape", y_train.shape, y_test.shape)
```

X\_trainとX\_testのshape (1168, 3) (292, 3)

y\_trainとy\_testのshape (1168, 1) (292, 1)

【モード】最急降下法により学習させる。j番目のパラメータの更新式は以下のようになる。def gradientdescentで使用している。

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m [(h_\theta(x^i) - y^i)x_j^i]$$

```
In [11]: class ScratchLinearRegression():
    def __init__(self, X, y, train):
        # 重みの初期化はコンストラクタの中では行わない。これでは、任意の特微量に対応できなければ
        # self.theta = np.random.rand(1,2).T # 重みの初期値
        self.alpha = 0.0000000001 # 学習率
        self.iteration = iteration # イテレーション回数
        self.cost_list = [] # 損失値リスト化していく
        self.theta_list = [] # 重みパラメータをリスト化しておく
        self.bias = bias # バイアス項を入れない場合はTrue

    def __init__(self, X, train, y_train):
        # 重みを算出する
        self.fit(X, train, y_train)

    def fit(self, X, train, y_train):
        # コンストラクタに記載していた重みの初期化をfitの中で実装した
        # また、バイアス項を入れないロジックを追加
        if self.bias:
            self.theta = np.random.rand(X.shape[1]-1, 1)
            X_train = np.delete(X_train, obj=1, axis=1)
        else:
            self.theta = np.random.rand(X.shape[1], 1)

        self._gradient_descent(X_train, y_train)

        # 平均二乗誤差を計算する。MSEは共通の関数を作つておき呼び出す
        def _compute_cost(self, X_train, y_train):
            # 予測値を算出する
            y_pred = np.dot(X_train, self.theta)
            loss = MSE(y_pred, y_train)
            return loss

        # 勾配下降法で重みを更新する
        # メンバ変数を参照したかったら、第一引数にselfをつけること
        def _gradient_descent(self, X_train, y_train):
            for i in range(self.iteration):
                # 仮定関数(予測値)と実際データとの差分を取得し、転置する
                A = np.dot(X_train, self.theta) - y_train).T

                # 重みを更新する
                self.theta = self.theta - (self.alpha/len(X_train)) * np.sum(np.dot(A, X_train))

                # 損失値を取得する
                cost = self._compute_cost(X_train, y_train)

                # 格納していく
                self.theta_list.append(self.theta)
                self.cost_list.append(cost)

        def predict(self, X):
            # バイアス項を入れないロジックを追加
            if self.bias:
                X_test = np.delete(X, obj=1, axis=1)
                final_answer = np.dot(X_test, self.theta)
            else:
                final_answer = np.dot(X, self.theta)

            return final_answer
```

```
In [12]: # インスタンス化する
slr = ScratchLinearRegression(iteration=30000, bias=False)
```

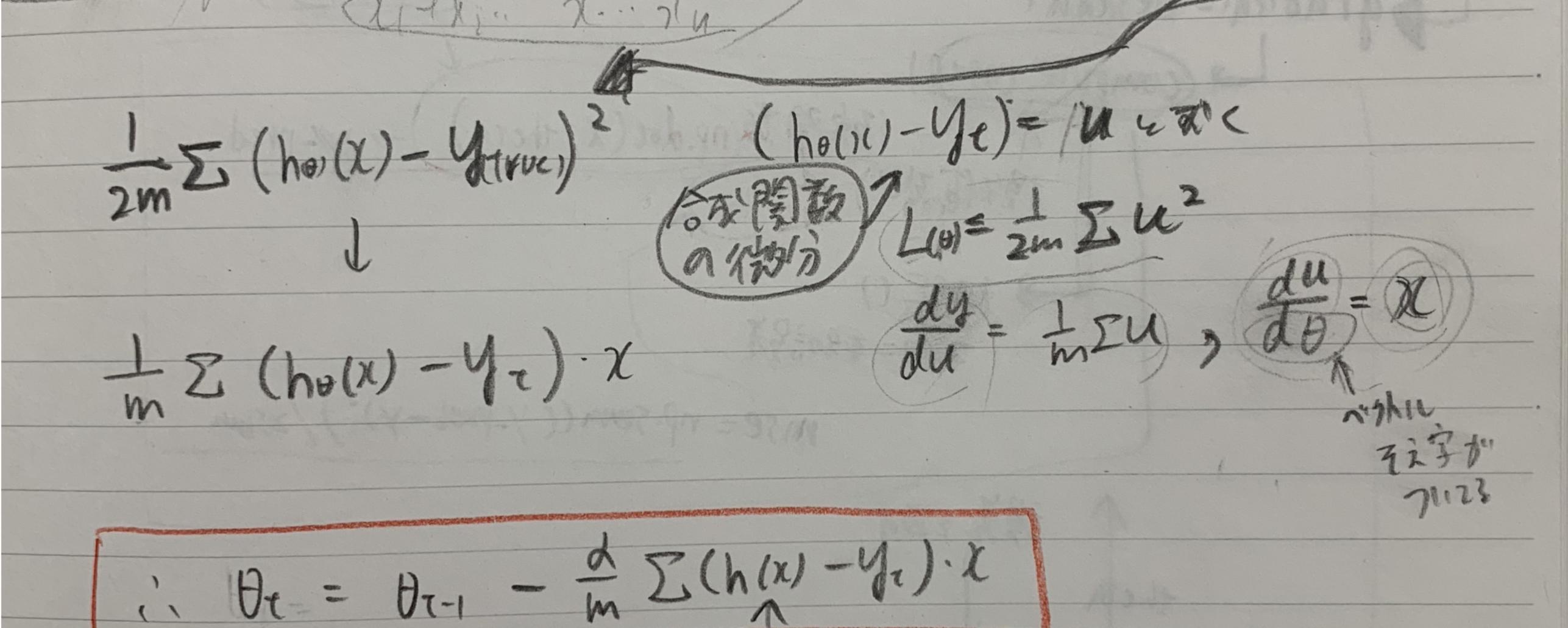
```
In [13]: # 重みパラメータを求める(学習する)
slr.fit(X_train, y_train)
```

```
In [14]: # テストデータで予測する
a = slr.predict(X_test)
```

```
In [15]: len(slr.cost_list)
```

```
Out[15]: 30000
```

```
In [16]: # 学習曲線(損失曲線)をグラフ化して見る
plt.plot(slr.cost_list)
plt.xlabel("iteration_counts", fontsize=15)
plt.ylabel("loss_values", fontsize=15)
plt.title("Learning curve", fontsize=15)
plt.show()
```



## テストデータの分割

```
In [17]: # def train_test_split(X, y, train_size):
#     # NumPy配列にする(pandasのdataframeを受け取ることを想定して)
#     X = np.array(X)
#     # line_X = len(X) * train_size
#     # line_X = int(line_X)
#     X_train = X[:line_X, :]
#     X_test = X[line_X-1, :]

#     # y = np.array(y)
#     # line_y = len(y) * (1 - train_size)
#     # line_y = int(line_y)
#     # y_train = y[:line_y, :]
#     # y_test = y[line_y-1, :]

#     # return X_train, X_test, y_train, y_test
```

### 説明

#### 前提知識

##### 機械学習全般

・教師あり学習とは、正解データと訓練データの組み合わせで学習すること

・分類問題：犬、猫など場合分けしたい場合（クラスラベルを予測する）

・回帰問題：身長・体重のデータセットで学習して、その中に含まれていない身長から体重を算出したい場合（目的変数が連続値となる場合に適用する）や、企業の今後数ヶ月の売上を予測したい場合の問題

・教師なし学習とは、学習時に答えとなるデータがない学習。クラスタリングを行う学習のこと。データ間の離れたパターンや構造を見つける。

##### フィーチャースケーリング

・正規化 特徴量の範囲を一定の範囲におさめる変換になります。主に[0,1]か、[-1,1]の範囲内におさめることができます。

・標準化 特徴量の平均を0、分散を1にする変換になります。

・正規化と標準化の使い分け 基本は、標準化を用います。正規化だと、外れ値が強く影響してしまうからです。標準化であれ、変換後も外れ値は外れ値として扱われます。

正規化は、画像処理におけるRGBの強さ[0,255] sigmoid, tanhなどの活性化関数を用いるNNのいくつかのモデル[0,1], [-1,1]の場合

外れ値がある場合などにも有効で、多く使われています。

標準化は、ロジスティック回帰、SVM、NNなど勾配法を用いたモデル kNN, k-meansなどの距離を用いるモデル PCA, LDA, kernel PCAなどのfeature extractionの手法の場合

正規化とは違い、minとmaxが固定されてしまいません。ただ、平均と分散を計算すると、0と1になります。

・フィーチャースケーリングのメリット 単位系や次元が違うもの同士でも比較ができるようになります。

### 学習率

学習率について、どのような値から選択するべきか、それはなぜかを述べよ。以下のキーワードを用いること。

#### 【回答】

学習率は人間が決めるハイパーパラメータの一種です。

##### Overshooting (過学習)

訓練データに対して学習されているが、未知データ（テストデータ）に対しては適合できていない、汎化できていない状態を指します。例えば、過去の試験内容にばかり注力してしまった、少しだけ誤差が取れなくなる

・Underfitting (未学習) 過学習の逆で、学習が進んでいないことです。訓練誤差が十分に減少していないければUnderfittingとなります。これを解決するためには、ハイパーパラメータを調整しながら、とにかく訓練誤差が下がるまで実験を繰り返す必要があります。

##### Underfitting (未学習)

過学習の逆で、学習が進んでいないことです。訓練誤差が十分に減少していないければUnderfittingとなります。これを解決するためには、ハイパーパラメータを調整しながら、とにかく訓練誤差が下がるまで実験を繰り返す必要があります。

### 学習曲線

以下の内容を含め学習曲線について述べよ。

・どのように時に使うのか

・学習曲線からどのようにが分かるか

学習曲線は学習回数とコストのプロットした結果になります。これでその学習率で収束したかどうか、収束速度がひと目で分かるため学習率のチューニングに使用されます。

### 正規方程式

係数θを求める方法として、最急降下法（Gradient Descent）を使用するのではなく、正規方程式（Normal Equation）を使用する方法がある。最急降下法と正規方程式のメリットとデメリットを計算量と計算速度の観点から述べよ。

最急降下法（gradient descent）

学習率の設定が必要

計算を繰り返す（ループ処理をする）必要がある

学習率を設定する必要なし

ループ処理の必要なし

学習率のチューニングが必要 逆行列の計算があるため特微量、データ数が増えると遅くなる

参考サイト

<https://omedstu.jimdo.com/2018/04/21/%E6%AD%A3%E8%A6%8F%E6%96%B9%E7%A8%BB%E5%BC%8F/>

### 更新式の導出

最急降下法の更新式が導出される過程を説明せよ。

```
In [17]: # def train_test_split(X, y, train_size):
#     # NumPy配列にする(pandasのdataframeを受け取ることを想定して)
#     X = np.array(X)
#     # line_X = len(X) * train_size
#     # line_X = int(line_X)
#     X_train = X[:line_X, :]
#     X_test = X[line_X-1, :]

#     # y = np.array(y)
#     # line_y = len(y) * (1 - train_size)
#     # line_y = int(line_y)
#     # y_train = y[:line_y, :]
#     # y_test = y[line_y-1, :]

#     # return X_train, X_test, y_train, y_test
```

### 説明

#### 前提知識

##### 機械学習全般

・教師あり学習とは、正解データと訓練データの組み合わせで学習すること

・分類問題：犬、猫など場合分けしたい場合（クラスラベルを予測する）

・回帰問題：身長・体重のデータセットで学習して、その中に含まれていない身長から体重を算出したい場合（目的変数が連続値となる場合に適用する）や、企業の今後数ヶ月の売上を予測したい場合の問題

・教師なし学習とは、学習時に答えとなるデータがない学習。クラスタリングを行う学習のこと。データ間の離れたパターンや構造を見つける。

##### フィーチャースケーリング

単位系や次元が違うもの同士でも比較ができるようになります。

・正規化 特徴量の範囲を一定の範囲におさめる変換になります。主に[0,1]か、[-1,1]の範囲内におさめることができます。

・標準化 特徴量の平均を0、分散を1にする変換になります。

・正規化と標準化の使い分け 基本は、標準化を用います。正規化だと、外れ値が強く影響してしまうからです。標準化であれ、変換後も外れ値は外れ値として扱われます。

正規化は、画像処理におけるRGBの強さ[0,255] sigmoid, tanhなどの活性化関数を用いるNNのいくつかのモデル[0,1], [-1,1]の場合

外れ値がある場合などにも有効で、多く使われています。

標準化は、ロジスティック回帰、SVM、NNなど勾配法を用いたモデル kNN, k-meansなどの距離を用いるモデル PCA, LDA, kernel PCAなどのfeature extractionの手法の場合

正規化とは違い、minとmaxが固定されてしまいません。ただ、平均と分散を計算すると、0と1になります。

・フィーチャースケーリングのメリット 単位系や次元が違うもの同士でも比較ができるようになります。

まずはなぜかを説明せよ。

### 学習率

学習率について、どのような値から選択するべきか、それはなぜかを述べよ。以下のキーワードを用いること。

#### 【回答】

学習率は人間が決めるハイパーパラメータの一種です。

##### Overshooting (過学習)

訓練データに対して学習されているが、未知データ（テストデータ）に対しては適合できていない、汎化できていない状態を指します。例えば、過去の試験内容にばかり注力してしまった、少しだけ誤差が取れなくなる

##### Underfitting (未学習)

過学習の逆で、学習が進んでいないことです。訓練誤差が十分に減少していないければUnderfittingとなります。これを解決するためには、ハイパーパラメータを調整しながら、とにかく訓練誤差が下がるまで実験を繰り返す必要があります。

### 学習曲線

以下の内容を含め学習曲線について述べよ。