Python ではじめる機械学習

4.2 ビニング、離散化、線形モデル、決定木

15T4057F　藤井　真

4.2ビニング、離散化、線形モデル、決定木

- **ビニング** (binning)とは

  bin（ビン）に -ing（すること）

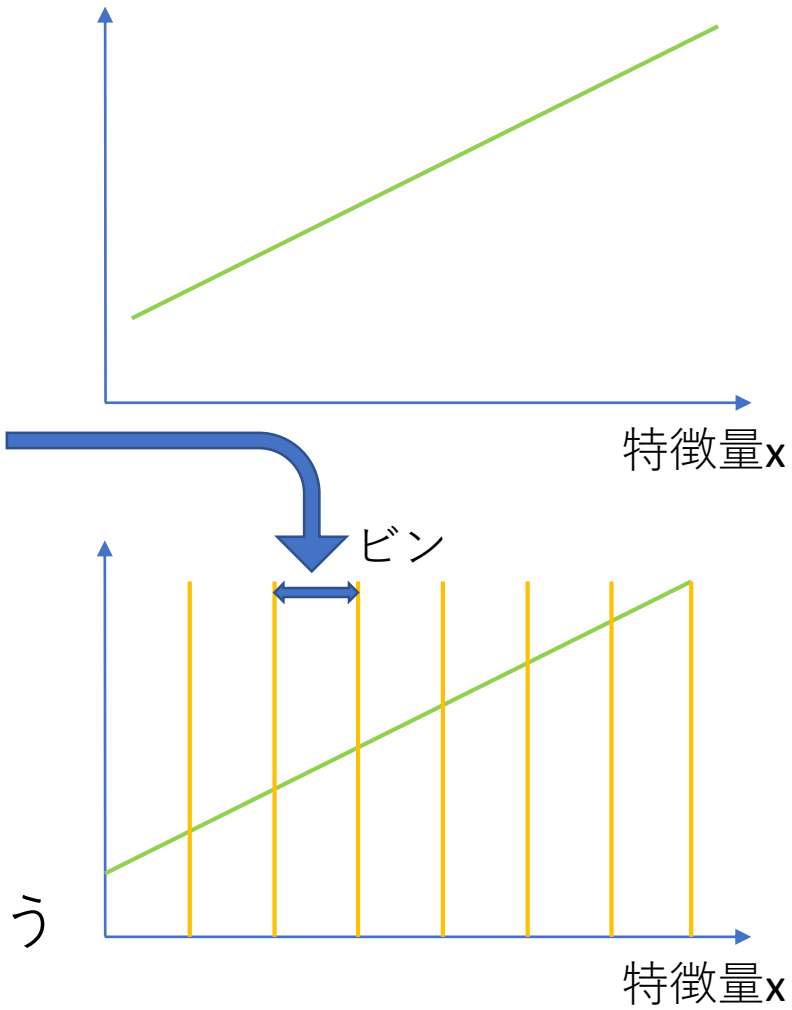
  - ビンとは

    ある1つの特徴量xを複数の特徴量に分割する際の単位

    つまりビニングとは

    ⇒ある1つの特徴量xを複数に分割すること

    離散化(discretization)とも言う



4.2ビニング、離散化、線形モデル、決定木

- 4章はデータ表現について
  ⇒最良のデータ表現
    ⇒利用する機械学習モデルによる

    実際に機械学習モデルによって最良のデータ表現が
    異なることを主なアルゴリズムで確認する。

  - 線形モデル
  - 決定木ベースのモデル

4.2ビニング、離散化、線形モデル、決定木

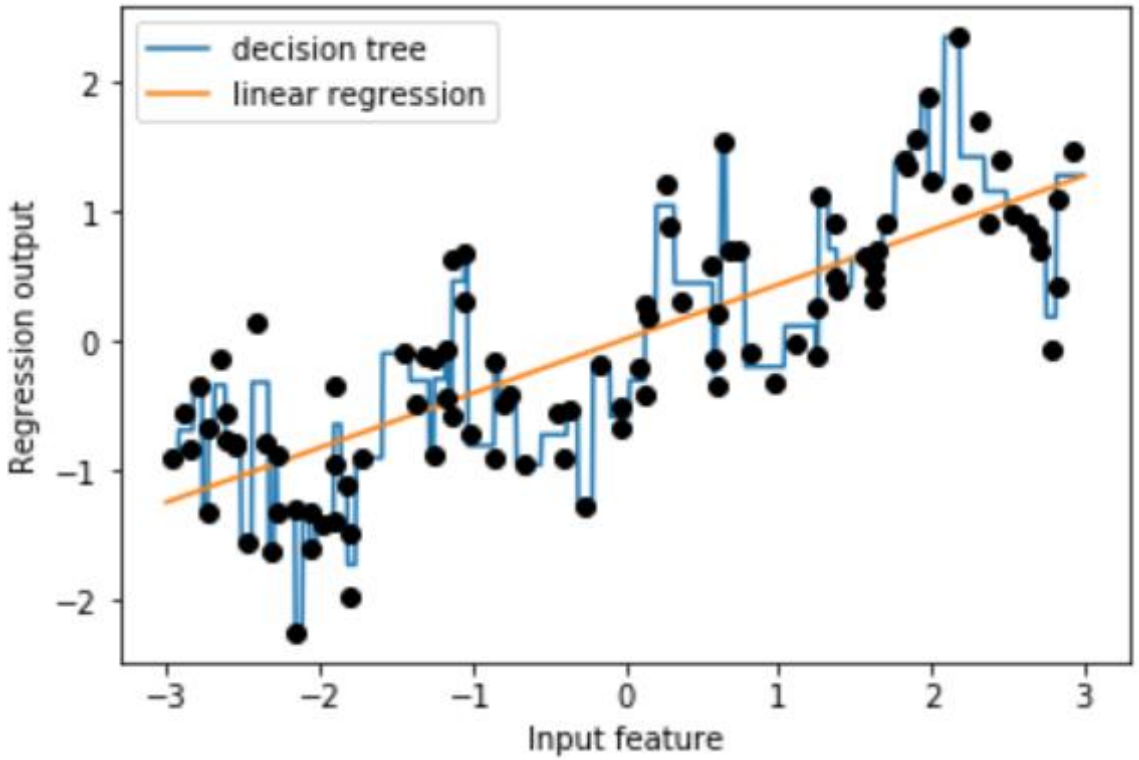
- データセット：waveデータセット（1次元＝特徴量１）

```
import mglearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

X, y = mglearn.datasets.make_wave(n_samples=100)
line = np.linspace(-3, 3, 1000, endpoint=False).reshape(-1, 1)
reg = DecisionTreeRegressor(min_samples_split=3).fit(X, y)
plt.plot(line, reg.predict(line), label="decision tree")

reg = LinearRegression().fit(X, y)
plt.plot(line, reg.predict(line), label="linear regression")

plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.legend(loc="best")

plt.show()
```

線形に等間隔の数列を生成する

4.2ビニング、離散化、線形モデル、決定木

- データセット：waveデータセット（1次元＝特徴量１）



線形モデル　：特徴量１なので直線（柔軟性なし
決定木ベース：複雑なモデルが構築される

4.2ビニング、離散化、線形モデル、決定木

- ビニングを用いる
  もとの特徴量の入力レンジ[-3 〜 +3]
  ビンの数　[10]
  ビンの境界[-3, -2.4, -1.8, -1.2, -0.6, 0. , 0.6, 1.2, 1.8, 2.4, 3.]

例えばwaveデータセットの
　　データポイント　-0.753　であれば　4のビン
　　データポイント　 2.704　であれば　10のビン
に入ることになる。

この離散値への置き換えを本書では
ワンホットエンコーディングで行っている。

4.2ビニング、離散化、線形モデル、決定木

- ビニングを用いる

```
bins = np.linspace(-3, 3, 11)
print("bins: {}".format(bins))
```

```
bins: [-3.  -2.4 -1.8 -1.2 -0.6  0.   0.6  1.2  1.8  2.4  3. ]
```

```
which_bin = np.digitize(X, bins=bins)
for i in range(5):
    print("{} => {}".format(X[i],which_bin[i]))
```

```
[-0.75275929] => [4]
[ 2.70428584] => [10]
[ 1.39196365] => [8]
[ 0.59195091] => [6]
[-2.06388816] => [2]
```

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False)
encoder.fit(which_bin)
X_binned = encoder.transform(which_bin)
print(X_binned[:5])
```

```
[[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

```
print("X_binned.shape: {}".format(X_binned.shape))
```

```
X_binned.shape: (100, 10)
```

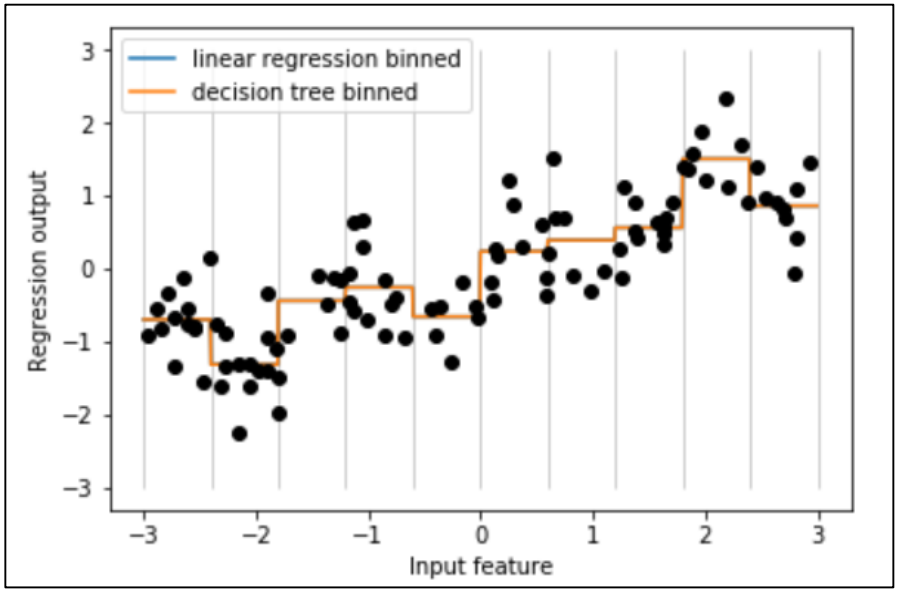
4.2ビニング、離散化、線形モデル、決定木

- ビニングを用いる

```
line_binned = encoder.transform(np.digitize(line, bins=bins))

reg = LinearRegression().fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), label='linear regression binned')

reg = DecisionTreeRegressor(min_samples_split=3).fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), label='decision tree binned')
plt.plot(X[:, 0], y, 'o', c='k')

plt.vlines(bins, -3, 3, linewidth=1, alpha=.2)
plt.legend(loc="best")
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.show()
```

4.2ビニング、離散化、線形モデル、決定木

- データセット：waveデータセット（1次元＝特徴量１）

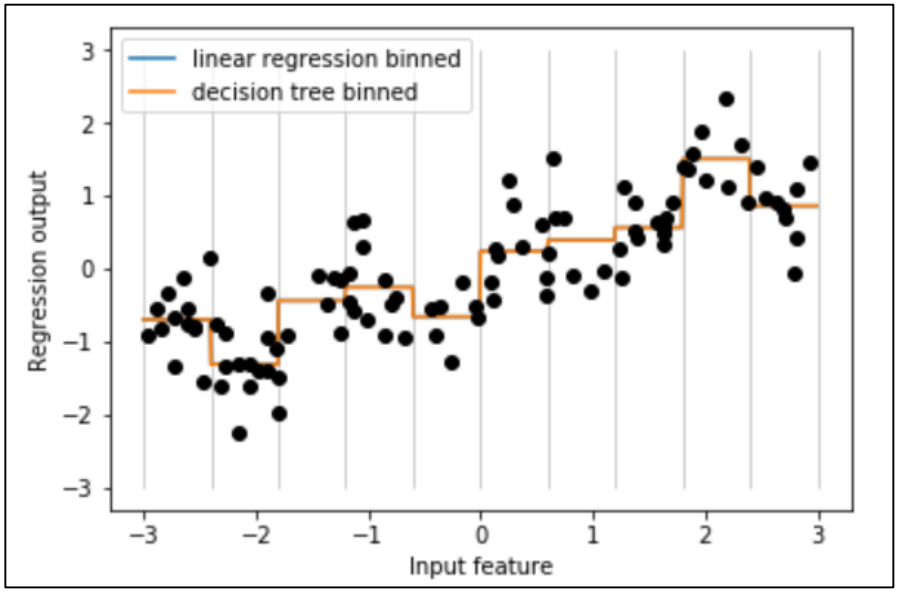


線形モデル　：元の直線よりは柔軟性がある
決定木ベース：モデルがかなり簡略化された

⇒今回の内容では、ビンごとの特徴量が一定になるので
どちらのモデルも同じ予測値を出し、重なってしまう。

4.2ビニング、離散化、線形モデル、決定木

- データセット：waveデータセット（1次元＝特徴量１）



まとめ

- ビニングは線形モデルについては有用。
- どうしても線形モデルを用いて柔軟性が欲しい場合に。
- 最良のデータ表現はモデルに依存する。