

# 目で見る過学習と正則化

---

内山 雄司 (@y\_\_uti)

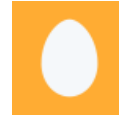
2019-03-11 社内勉強会

# 自己紹介

---

内山 雄司 (@y\_uti)

- <http://y-uti.hatenablog.jp/>



(phpusers-ja)

## 仕事

- 受託開発の会社 (株式会社ピコラボ) でプログラミングをしています

## 興味

- プログラミング言語処理系
- 機械学習

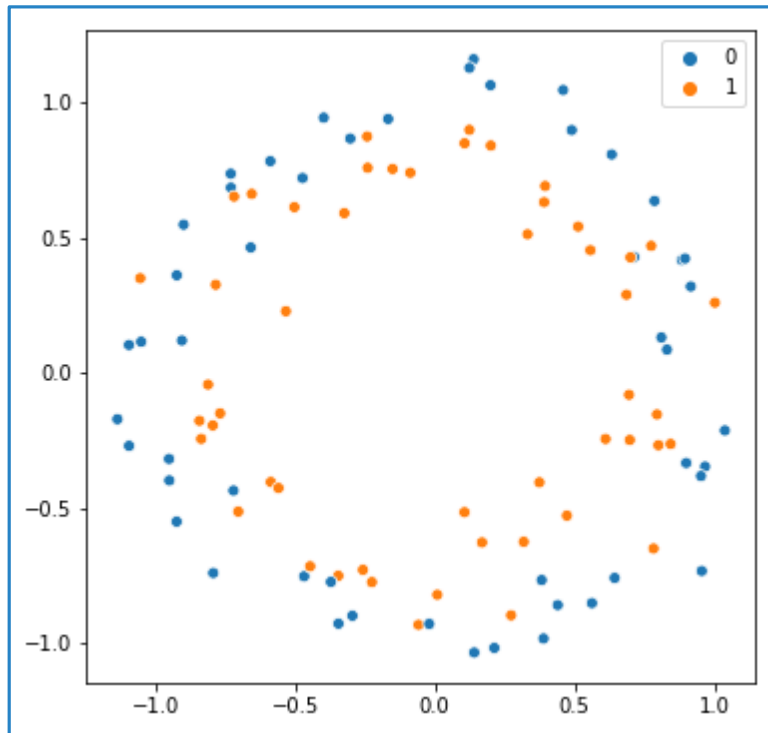
# 目標 (説明しようとしていること)

---

1. 特徴量を増やせば複雑なデータも分類できる
2. 増やし過ぎると過学習が発生する
3. 正則化という手法で過学習を抑えこむ

# 本日の食材

`sklearn.datasets.make_circles`



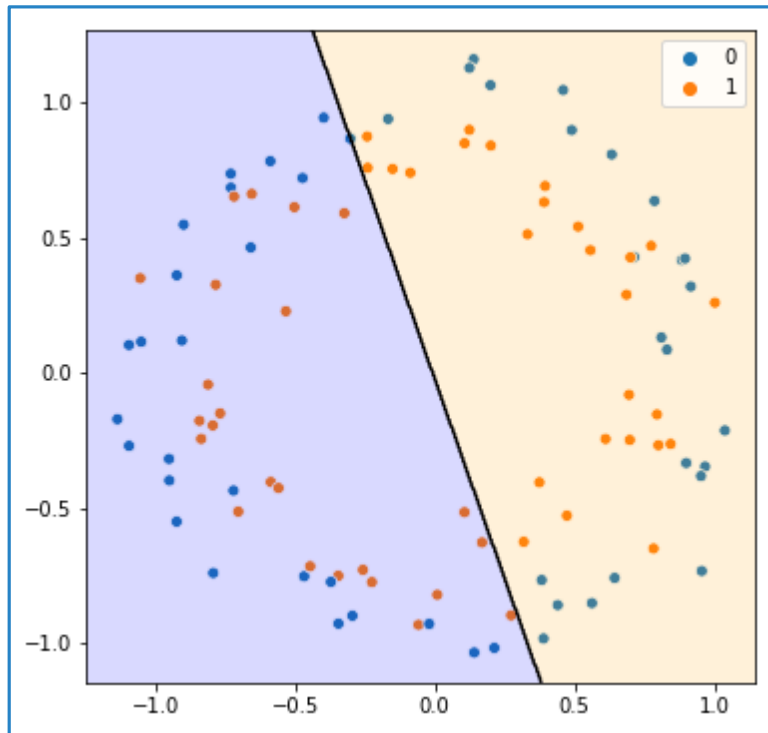
## データの生成方法

1. 半径 0.8 の円周上に **1** を配置
2. 半径 1.0 の円周上に **0** を配置
3.  $\sigma = 0.1$  のガウス雑音を加える

☞ 半径 0.9 の円 が正しい決定境界

# 本日の食材

`sklearn.datasets.make_circles`



## データの生成方法

1. 半径 0.8 の円周上に **1** を配置
2. 半径 1.0 の円周上に **0** を配置
3.  $\sigma = 0.1$  のガウス雑音を加える

☞ 半径 0.9 の円 が正しい決定境界

☞ 直線では上手く分類できない

# 線形基底関数モデル

---

# アプローチ

---

今回のデータは「直線では分類できない」

もう少し正確に言うと：

特徴量  $x, y$  に対して  $ax + by + c$  の正負では分類できない

そこで  $x, y$  にさまざまな関数を適用して特徴量を増やす

$$w_1 f_1(x, y) + w_2 f_2(x, y) + w_3 f_3(x, y) + \dots + c$$

うまく  $f_i$  を見つければ分類できるかもしれない

ポイント： $f_i$  で変換した後はただのロジスティック回帰

# 多項式特徴量

---

sklearn.preprocessing.PolynomialFeatures

```
from sklearn.preprocessing import PolynomialFeatures

polynomial_features = PolynomialFeatures(
    degree=3,
    interaction_only=False,
    include_bias=True).fit_transform([[2, 3]])

print(polynomial_features)
```

結果 : [[ 1. 2. 3. 4. 6. 9. 8. 12. 18. 27.] ]



# 多項式特徴量

sklearn.preprocessing.PolynomialFeatures

```
from sklearn.preprocessing import PolynomialFeatures

polynomial_features = PolynomialFeatures(
    degree=3,
    interaction_only=False,
    include_bias=True).fit_transform([[2, 3]])

print(polynomial_features)
```

結果 : [[ 1. 2. 3. 4. 6. 9. 8. 12. 18. 27.]]

組合せ :

次数 : 0次      1次      2次      3次

# 多項式特徴量を利用する

sklearn.pipeline.Pipeline

1. 元の特徴量から多項式特徴量を作成する
2. ロジスティック回帰を適用する

```
pipeline = Pipeline(steps=[
    ('poly', PolynomialFeatures(degree=3,
                                interaction_only=False, include_bias=False)),
    ('clf', LogisticRegression(
        C=1e100, solver='lbfgs', max_iter=10000)))

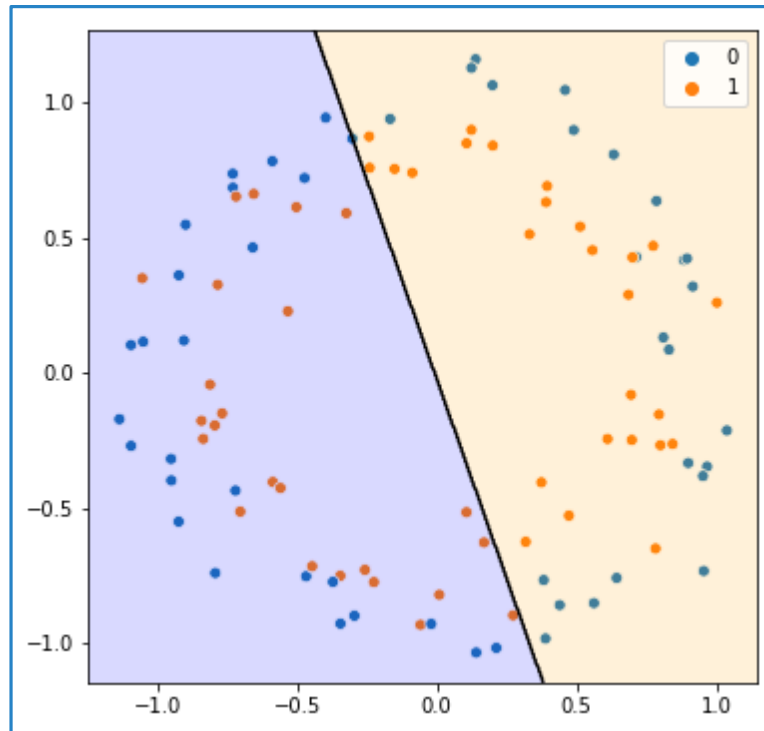
pipeline.fit(features, labels)
```

パラメータのうち特に注意が必要なものは以下の2つ

- `include_bias = False`      定数項 (0 次の特徴量) を除外する
- `C=1e100`      正則化なしで試すため大きな数を指定

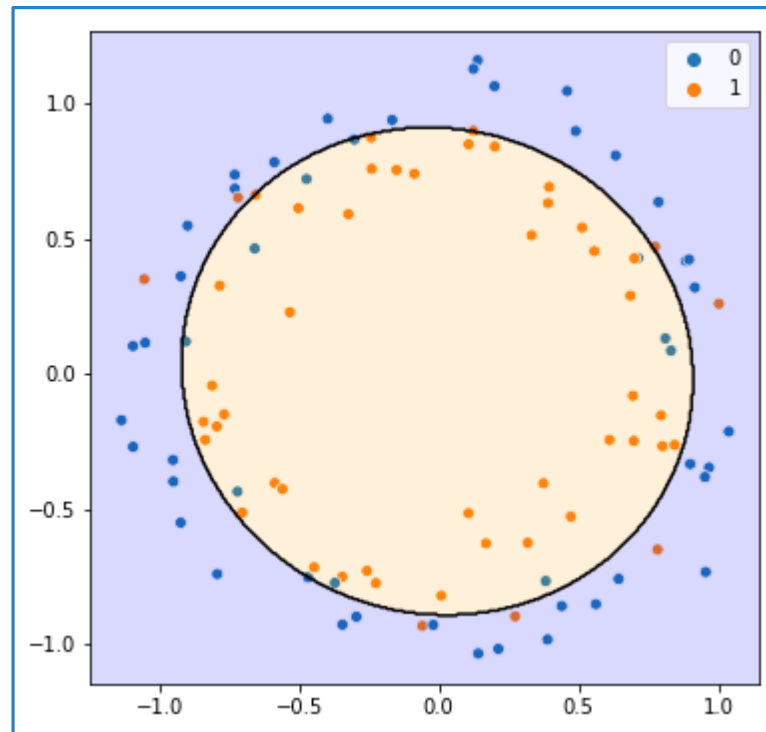
# 多項式特徴量の威力

degree = 1 (正解率 52%)



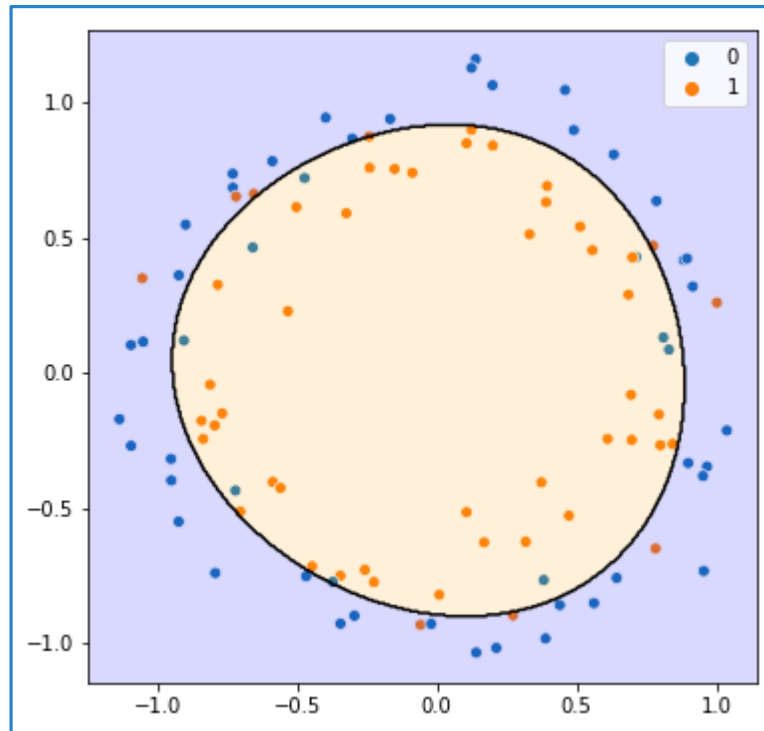
# 多項式特徴量の威力

degree = 2 (正解率 82%)



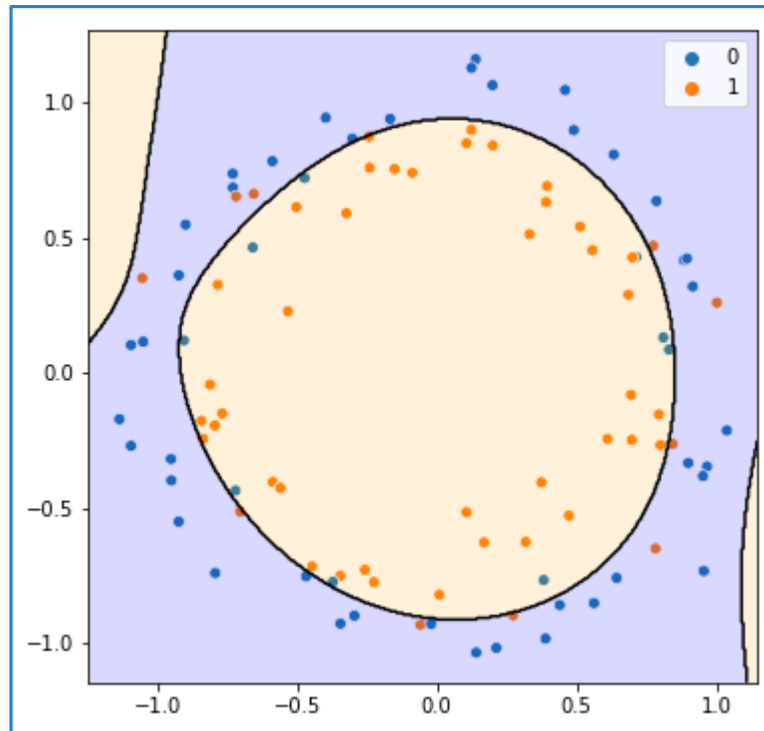
# 多項式特徴量の威力

degree = 3 (正解率 82%)



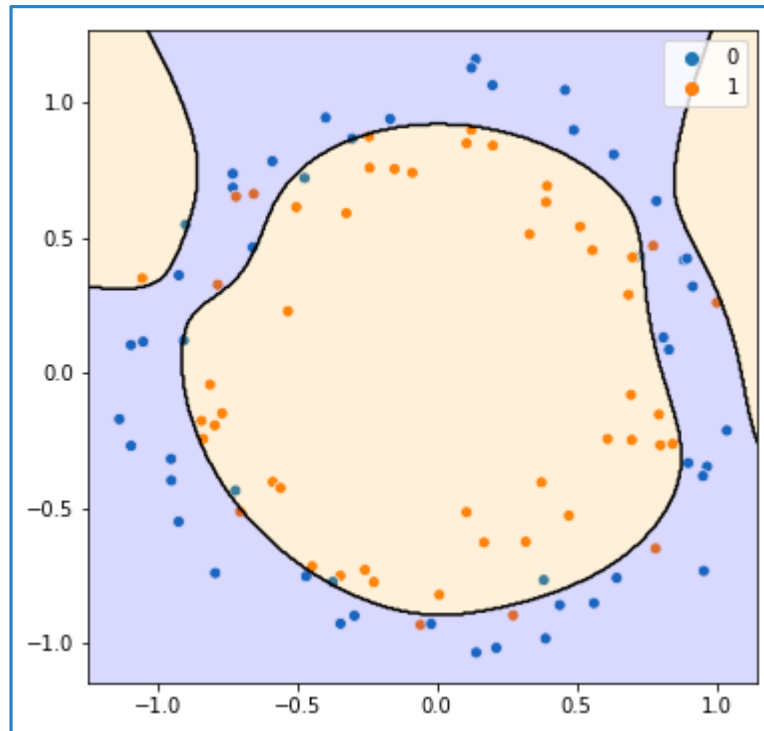
# 多項式特徴量の威力

degree = 4 (正解率 81%)



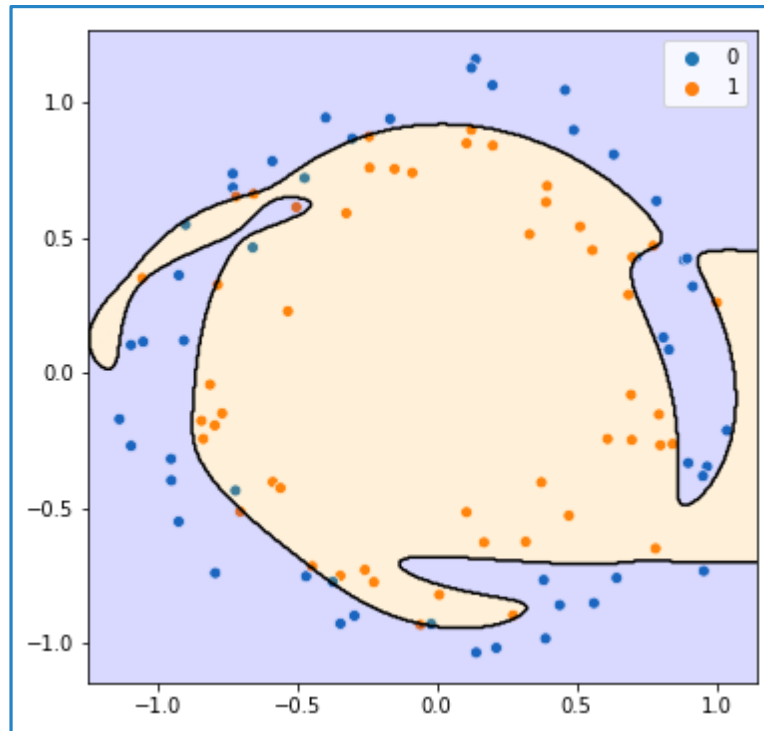
# 多項式特徴量の威力

degree = 5 (正解率 84%)



# 多項式特徴量の威力

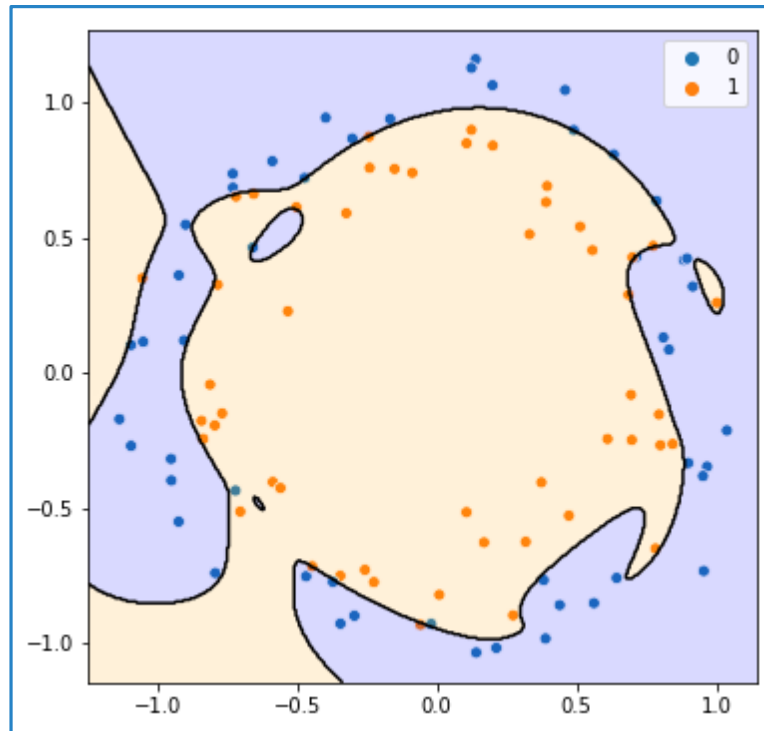
degree = 6 (正解率 89%)





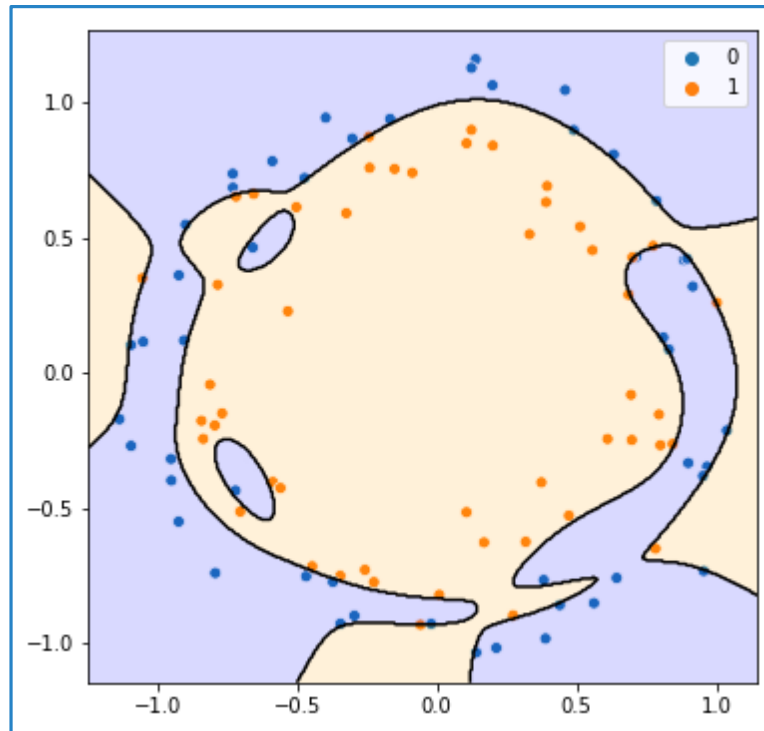
# 多項式特徴量の威力

degree = 7 (正解率 96%)



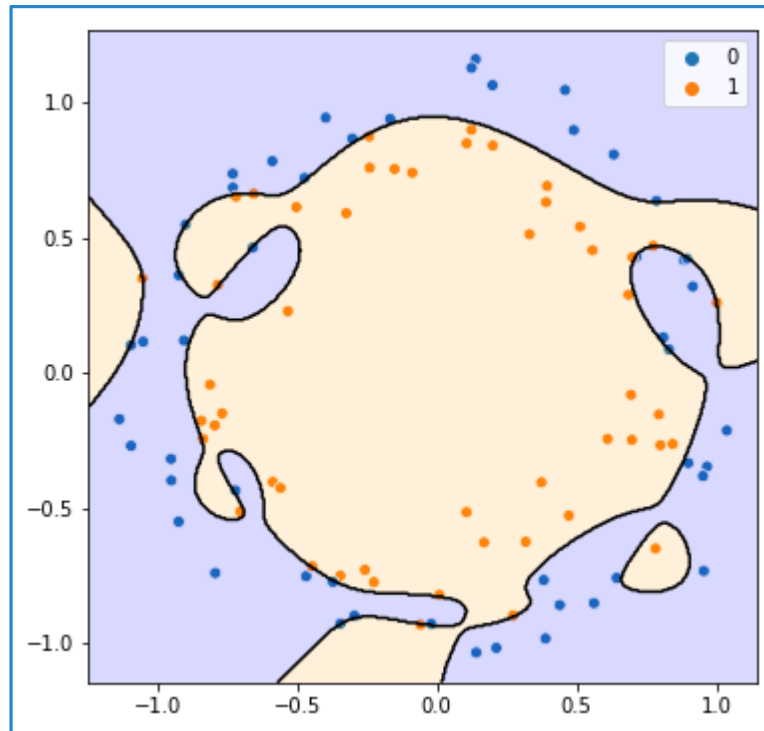
# 多項式特徴量の威力

degree = 8 (正解率 100%)



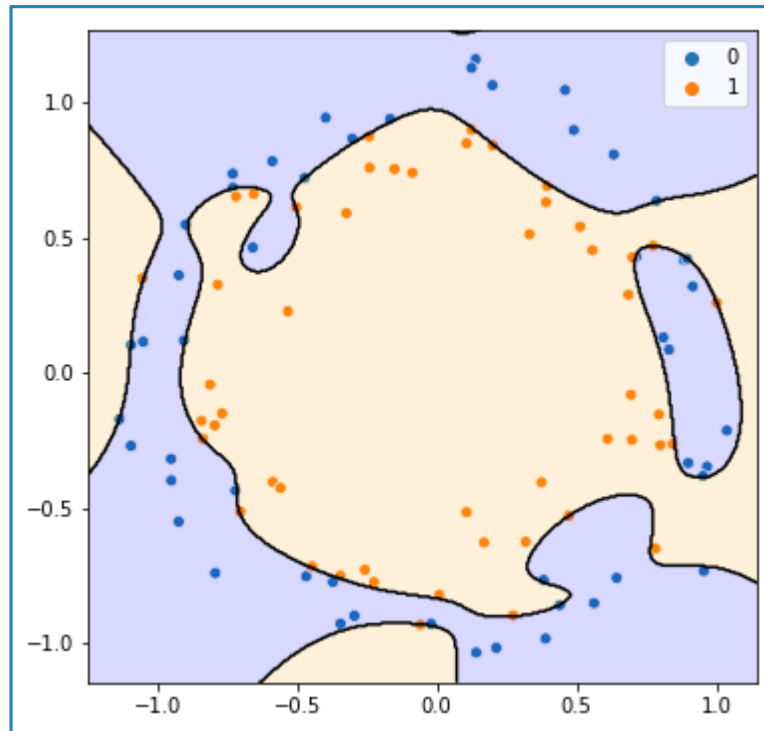
# 多項式特徴量の威力

degree = 9 (正解率 100%)



# 多項式特徴量の威力

degree = 10 (正解率 100%)



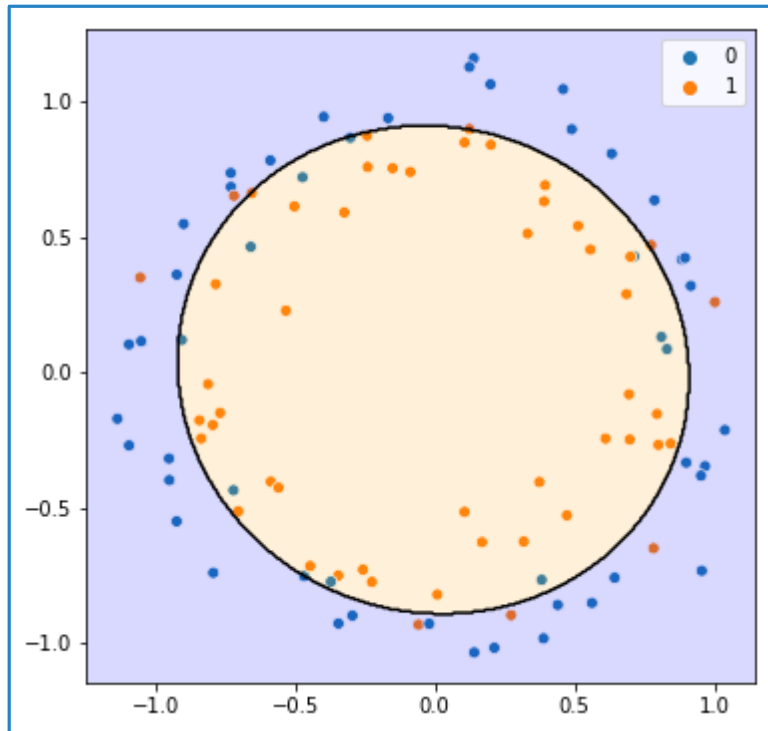
# 過學習

---

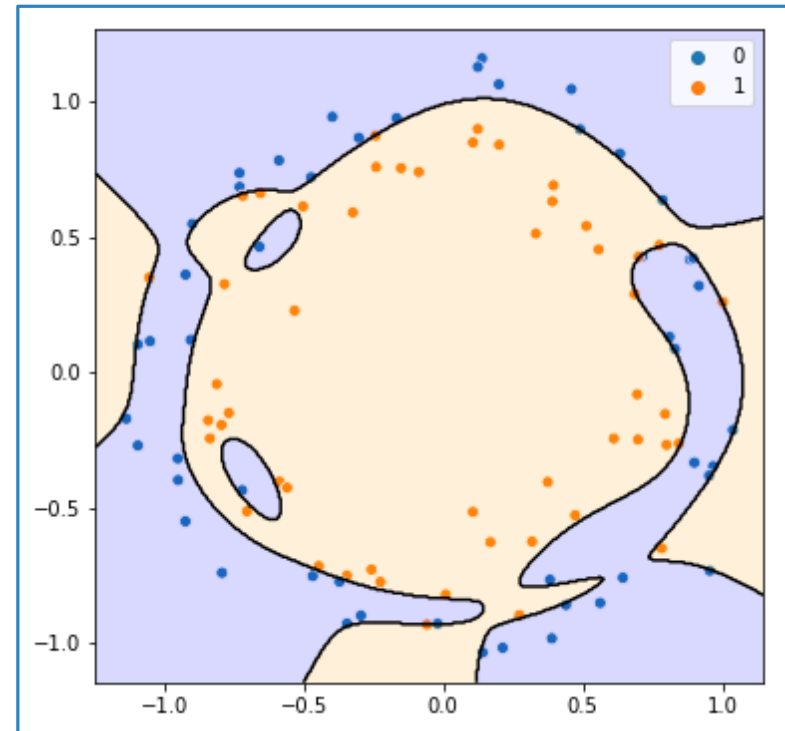
# 学習データに対する正解率

8 次の特徴量まで利用すると正解率 100% になったが . . .

degree = 2 (正解率 82%)



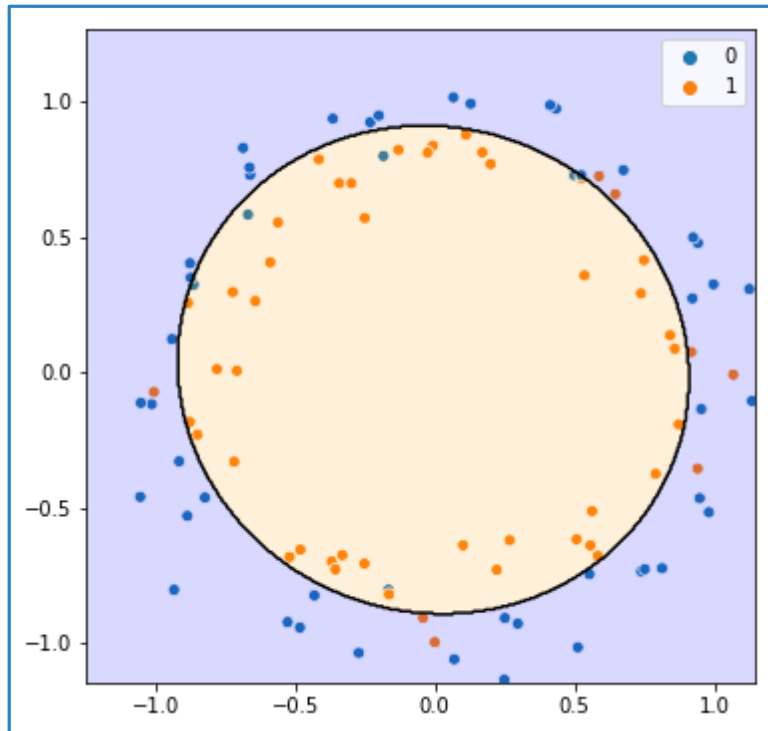
degree = 8 (正解率 **100%**)



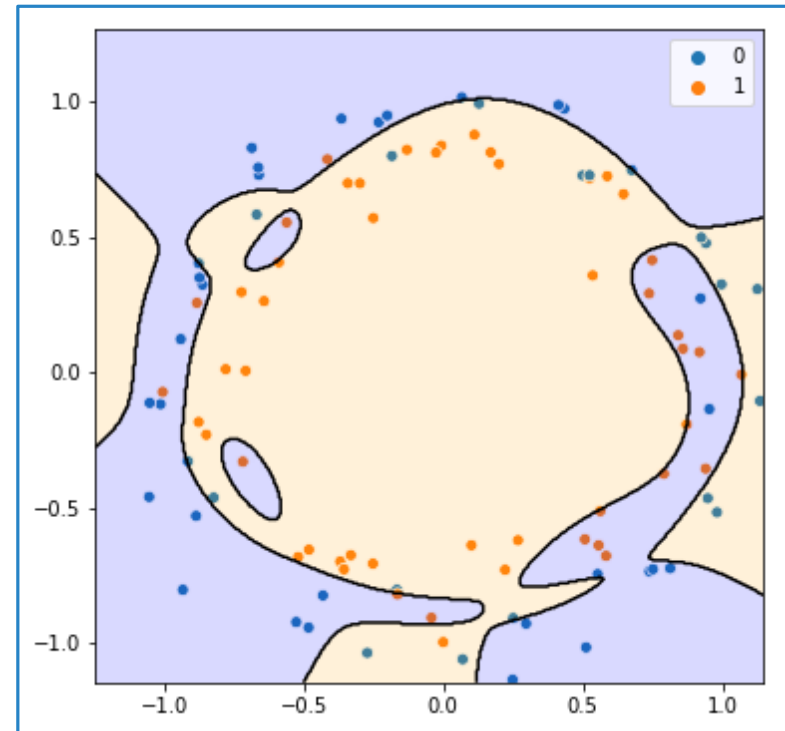
# ≠ 予測性能

同じ分布から生成された別のサンプルデータに適用すると・・・

degree = 2 (正解率 87%)



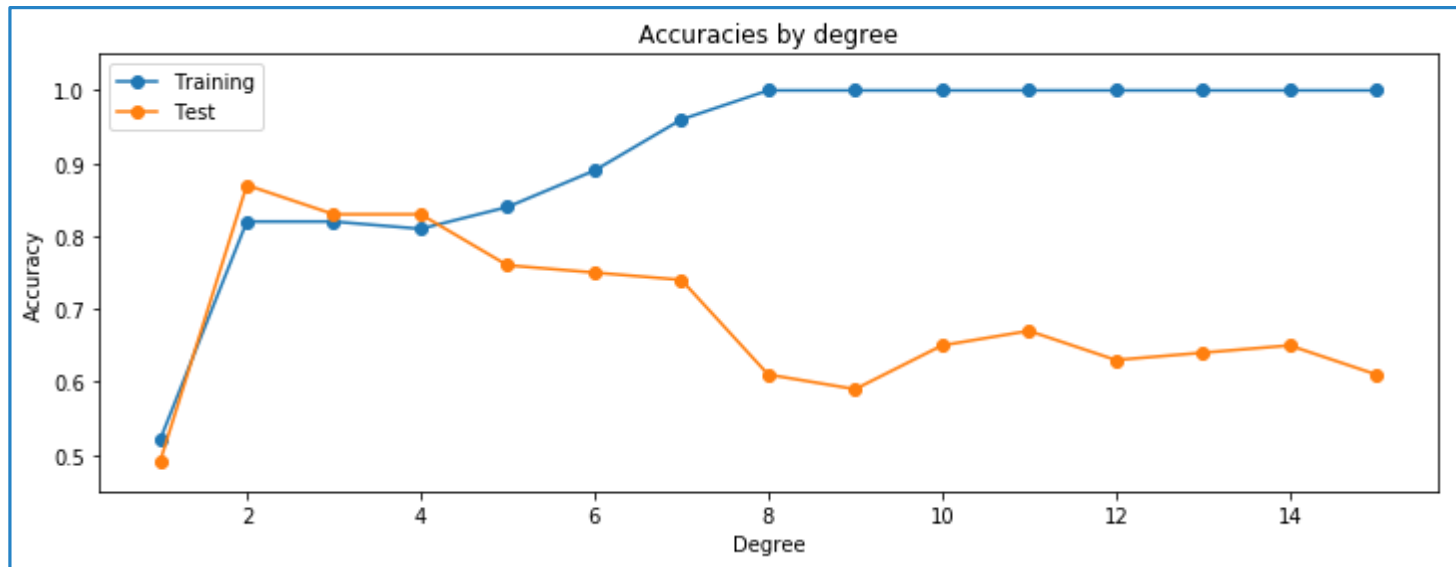
degree = 8 (正解率 **61%**)



# 過学習とは

学習データに含まれるノイズを過剰に拾ってしまう

- 結果として未知のデータに対する予測性能は下がってしまう
- 特徴量が高次元になるほど過学習が発生しやすくなる





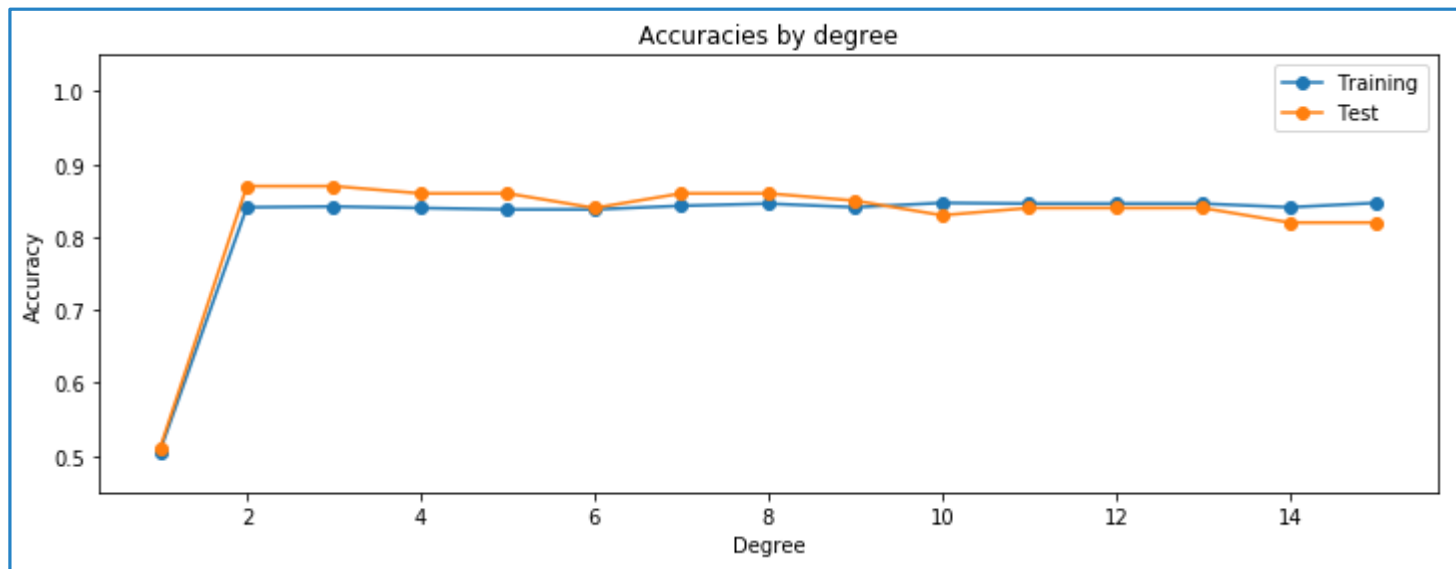
# 過学習を避ける方法

---

1. 学習データを増やす
2. 良い特徴量を設計する (良いモデルを選択する)
3. 正則化

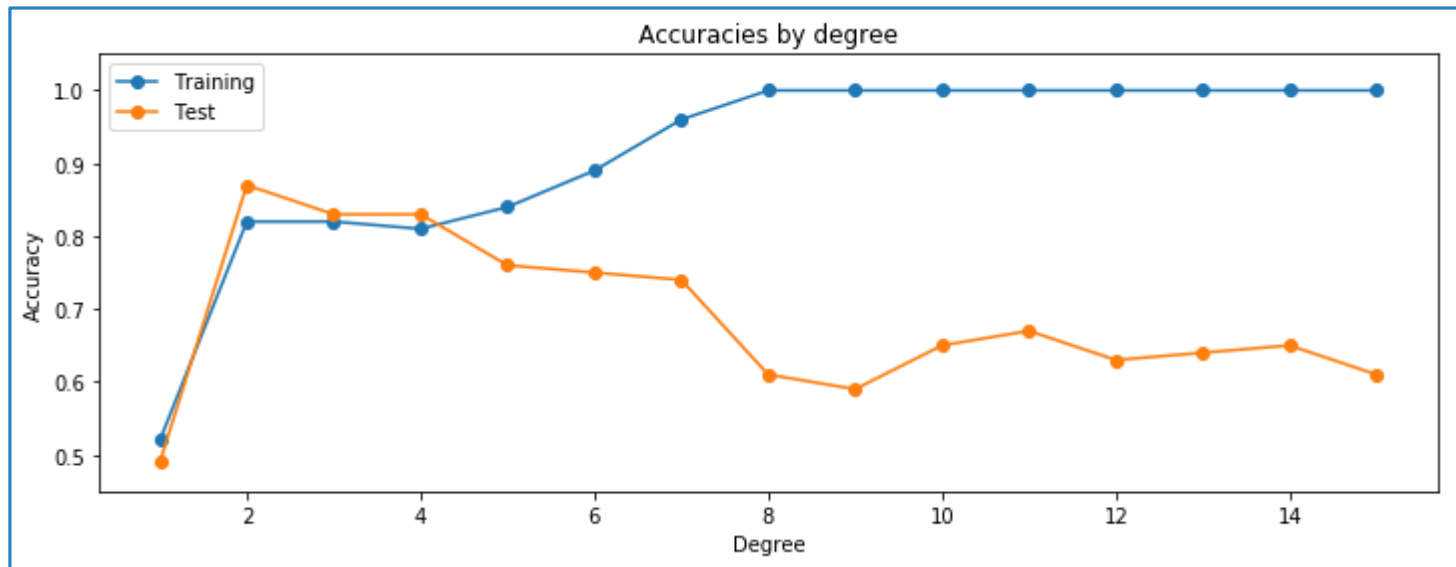
# 学習データを増やす

学習データを 100 個から 1,000 個に増やした場合



# 良い特徴量を設計する

下図のような結果から適切なモデルを選択する



データをよく見て適切な特徴量を見つける

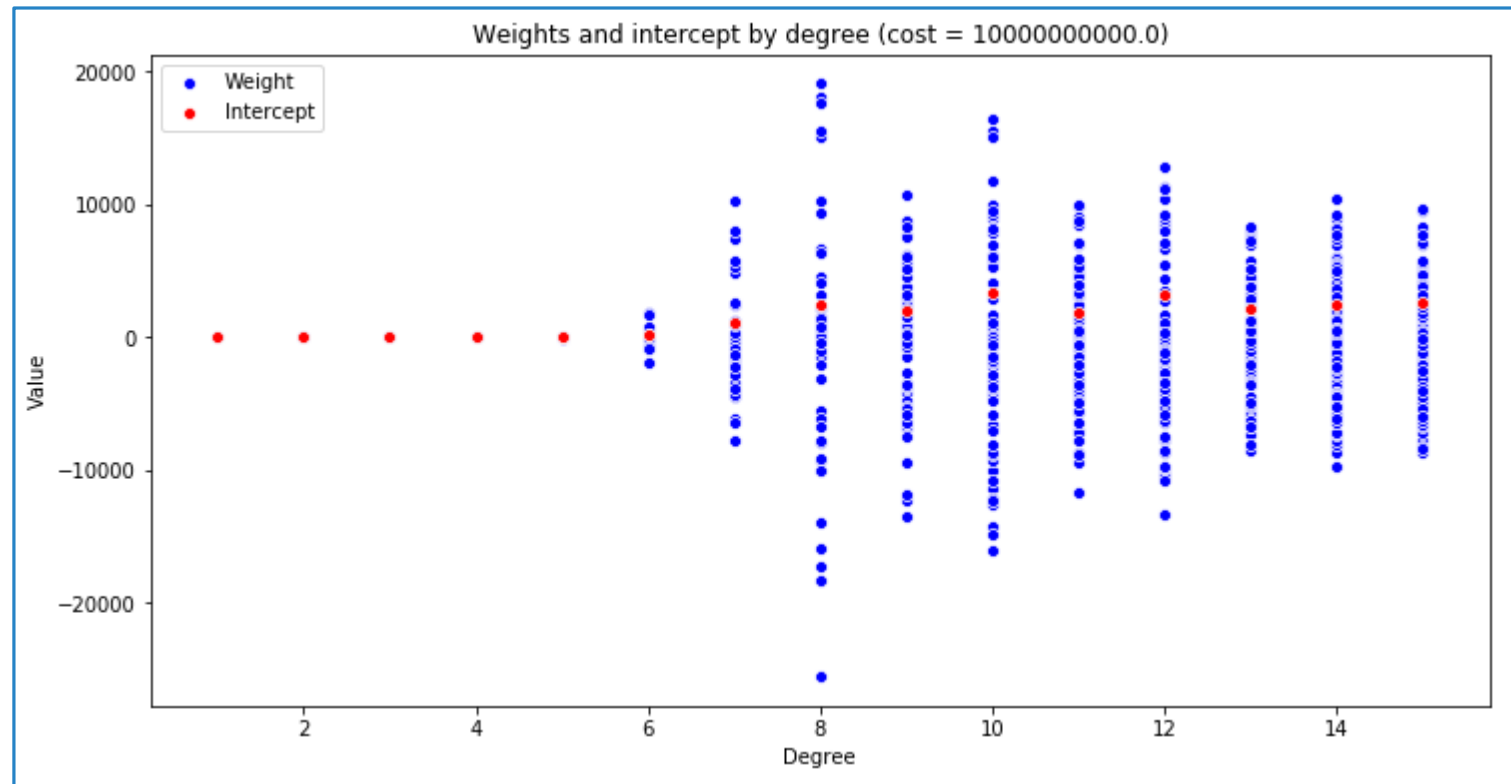
- 今回の問題なら「原点からの距離」を特徴量とすればよい

# 正則化

---

# 特徴量の重みの分布

過学習の状況では特徴量の重みが爆発



# 正則化とは

---

特徴量の重みに応じた値を損失関数に上乗せする

L2 正則化 :

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^n \log(\exp(-y_i (X_i^T w + c)) + 1).$$

([https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) より引用)

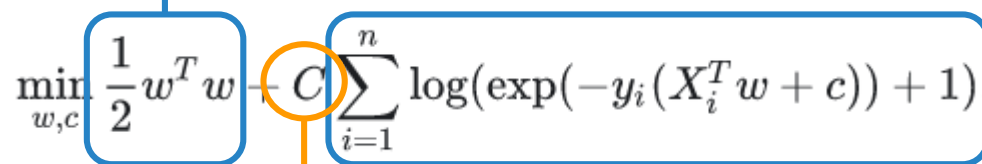
# 正則化とは

特徴量の重みに応じた値を損失関数に上乗せする

L2 正則化 :

正則化項 : 特徴量の重みを 0 にしようとする力

負の対数尤度 : データを正しく分類しようとする力

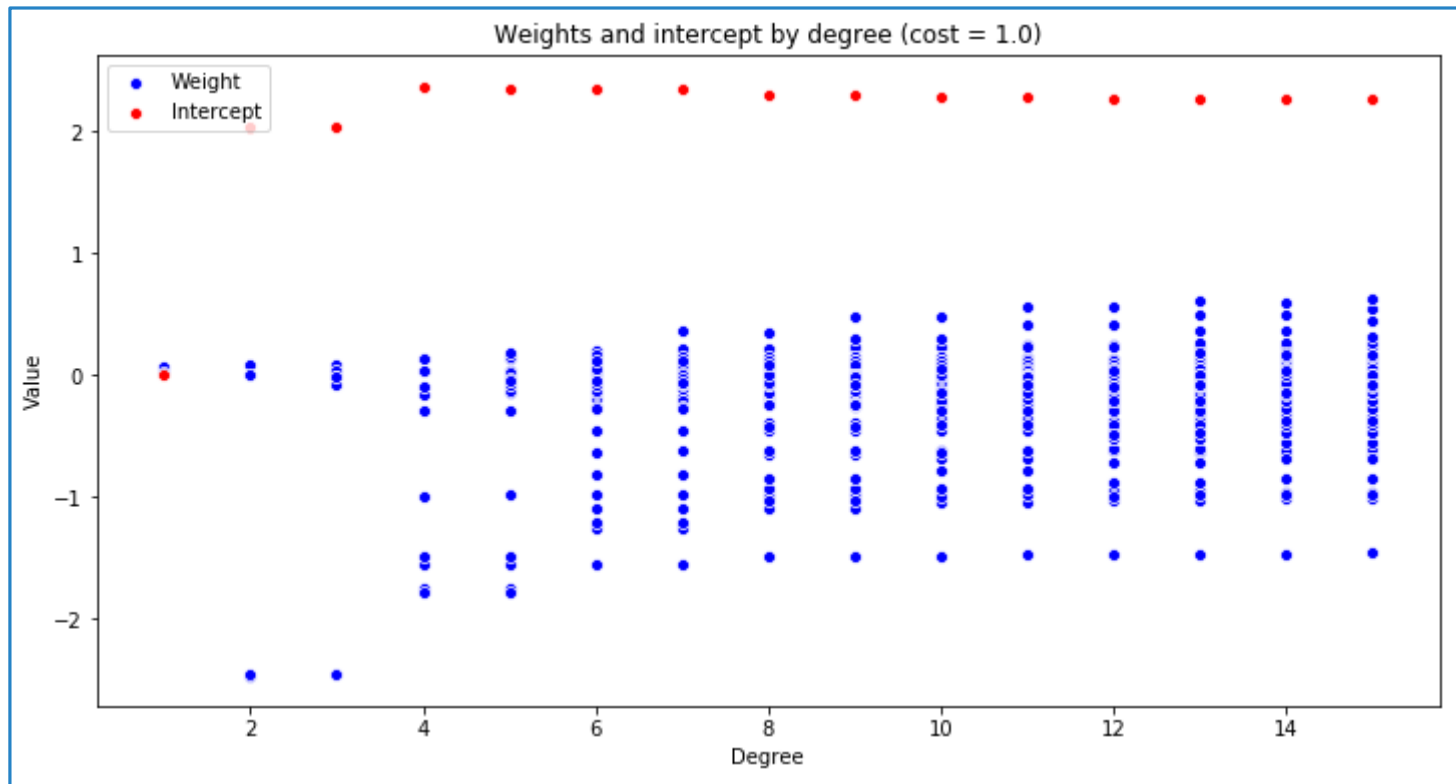
$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^n \log(\exp(-y_i (X_i^T w + c)) + 1)$$


([https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) より引用)

コスト : 両者のバランスを決めるパラメータ

# 特徴量の重みの分布 (C=1)

コストを小さく設定すると特徴量の重みが小さく抑えられる

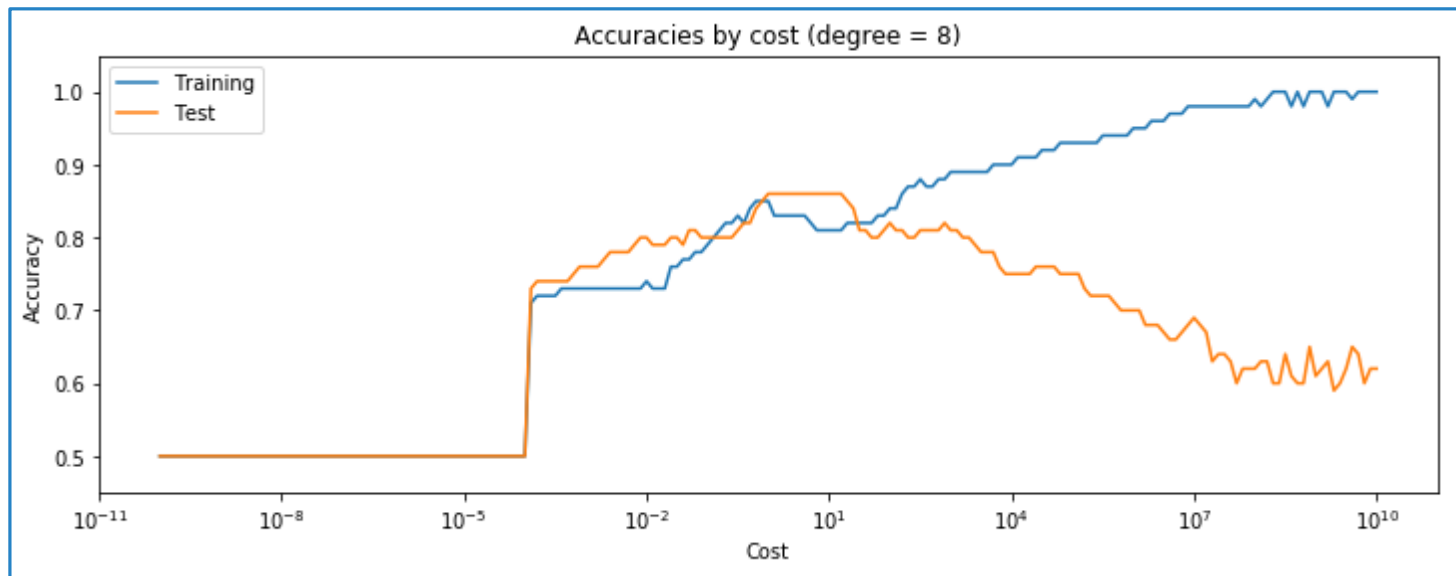




# コストによる正解率の変化

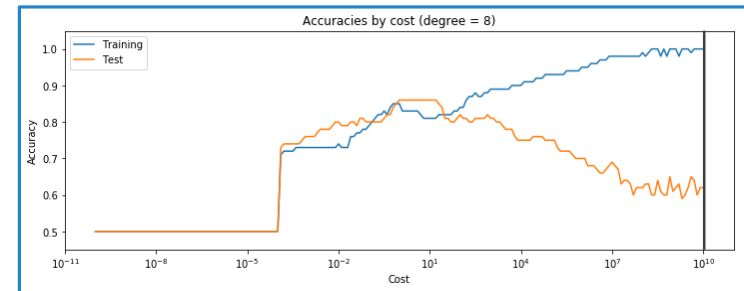
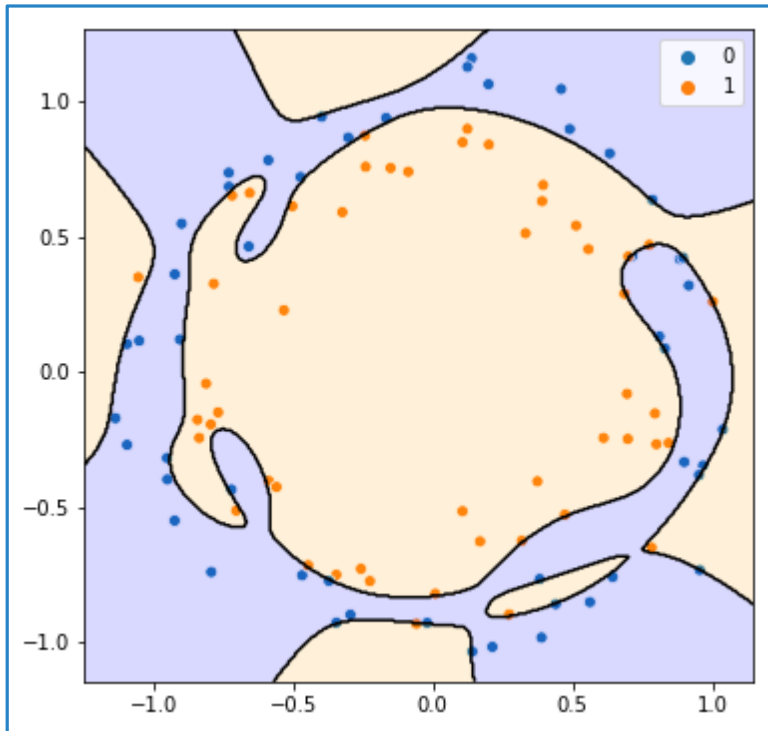
## 実験内容

- 8 次の特徴量まで利用
- コストパラメータの設定値を変えながら正解率を描画



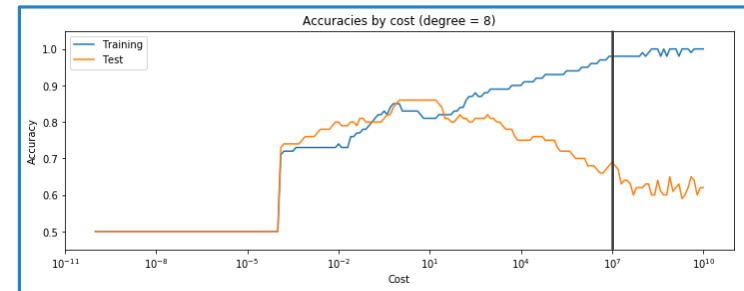
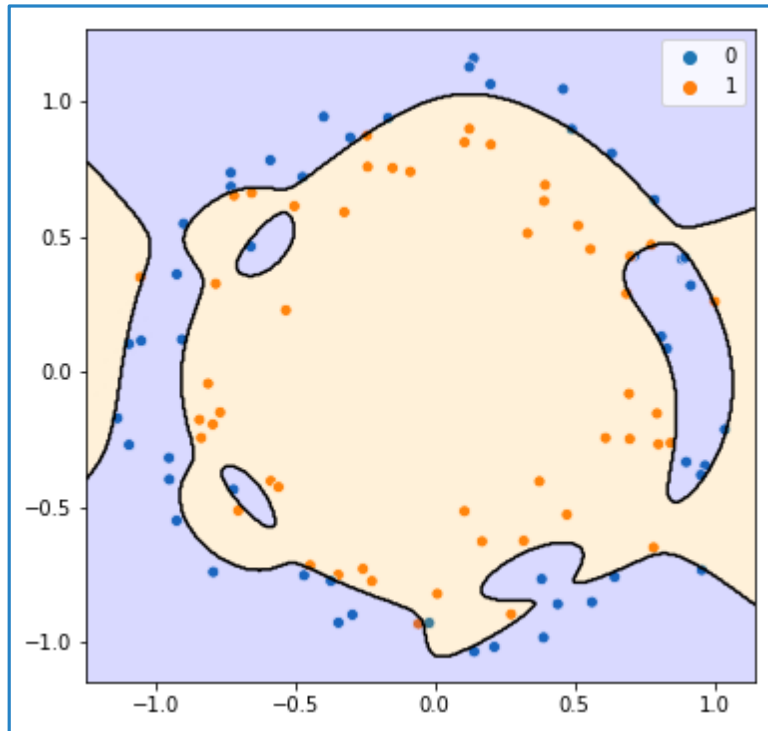
# 決定境界が変化する様子

degree=8, C=1e10



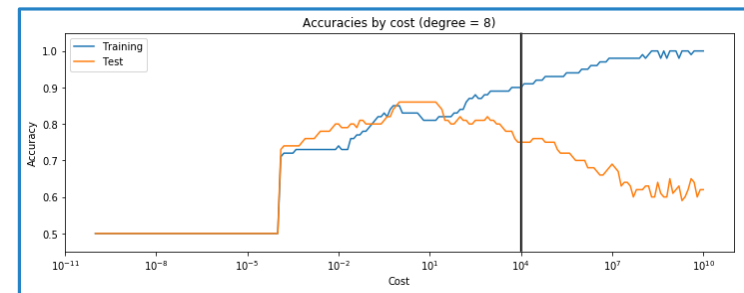
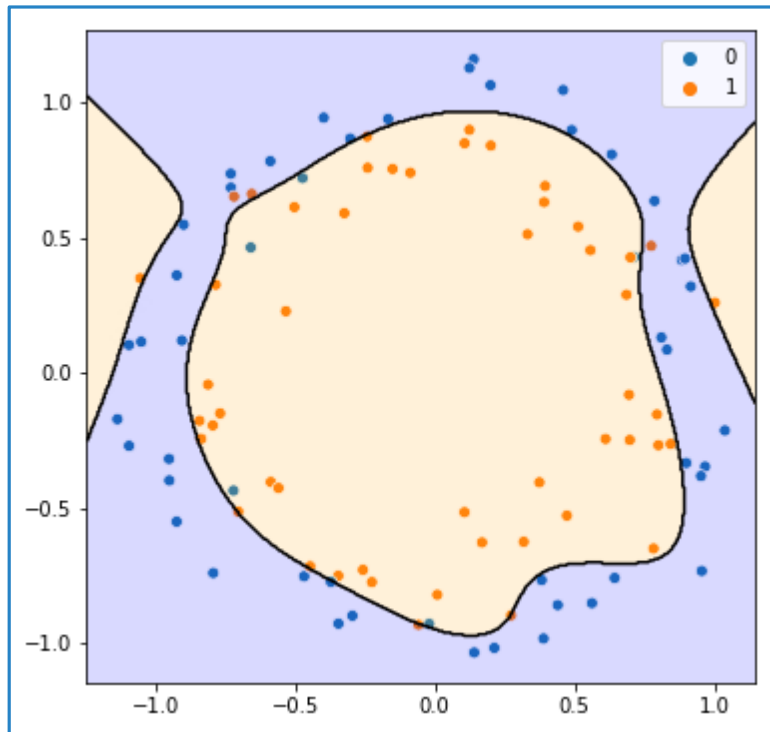
# 決定境界が変化する様子

degree=8,  $C=1e7$



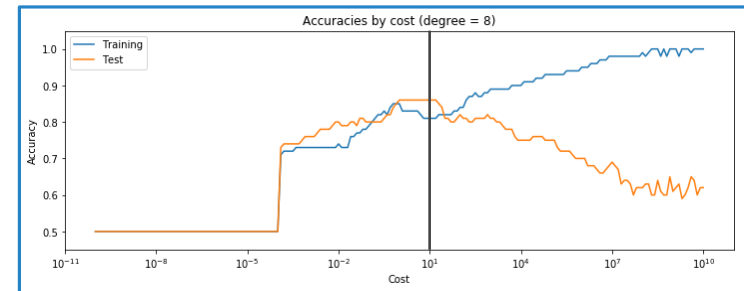
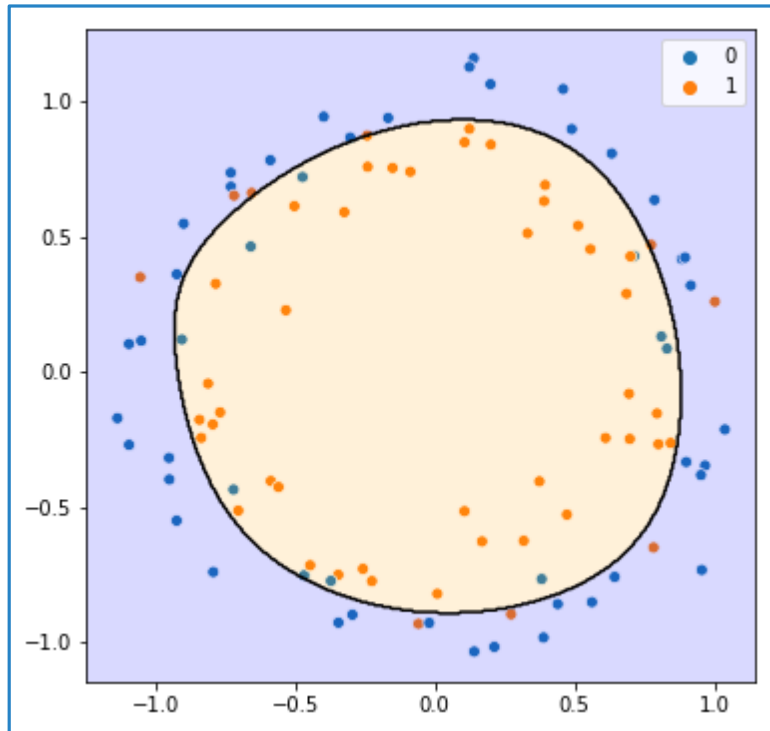
# 決定境界が変化する様子

degree=8, C=1e4



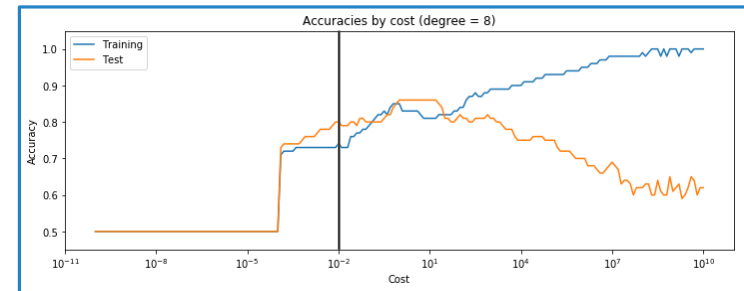
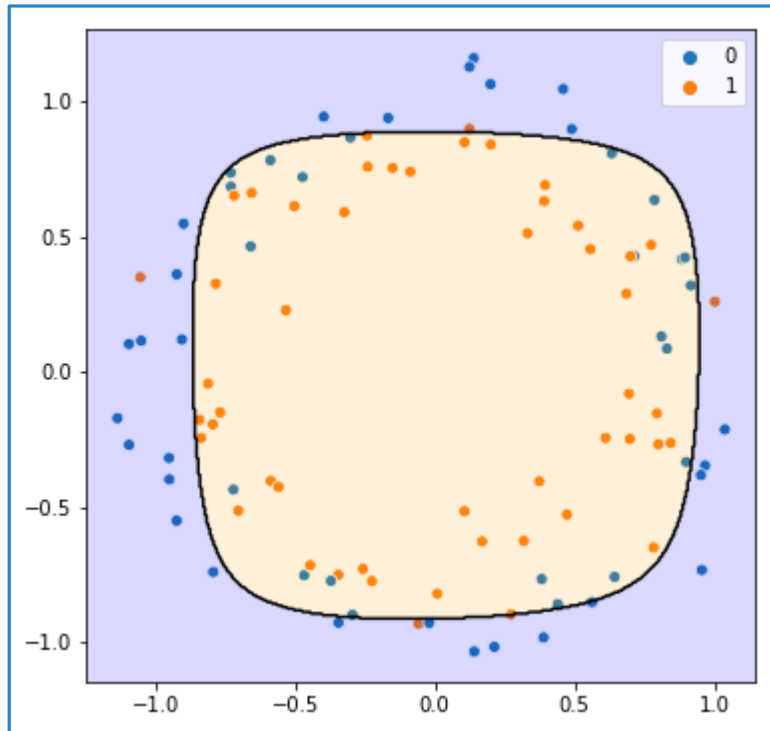
# 決定境界が変化する様子

degree=8, C=1e1



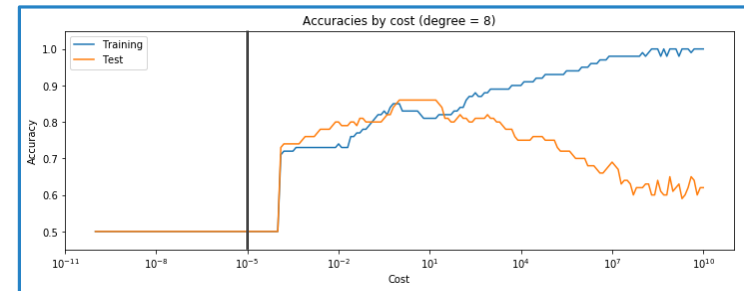
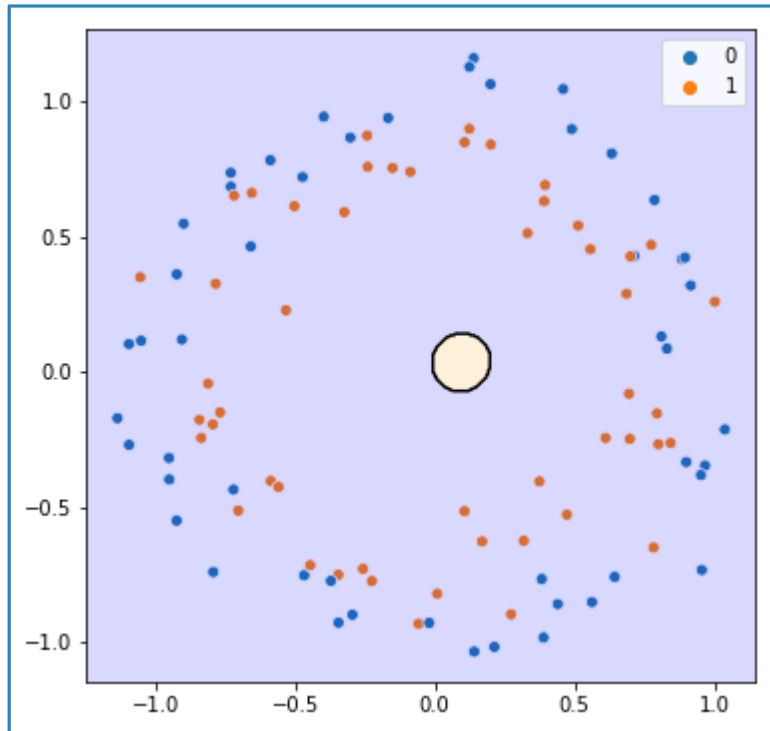
# 決定境界が変化する様子

degree=8, C=1e-2



# 決定境界が変化する様子

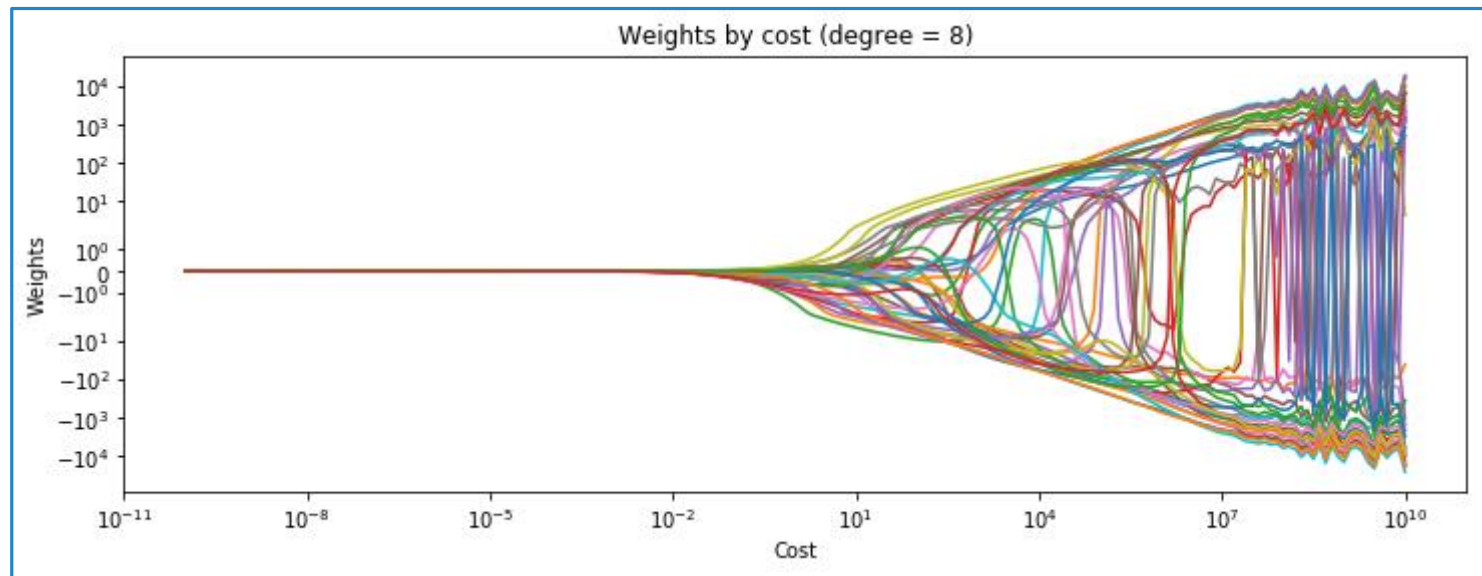
degree=8,  $C=1e-5$



# 各特徴量の重み

## 実験内容

- 8 次の特徴量まで利用
- コストパラメータの設定値を変えながら各特徴量の重みを描画

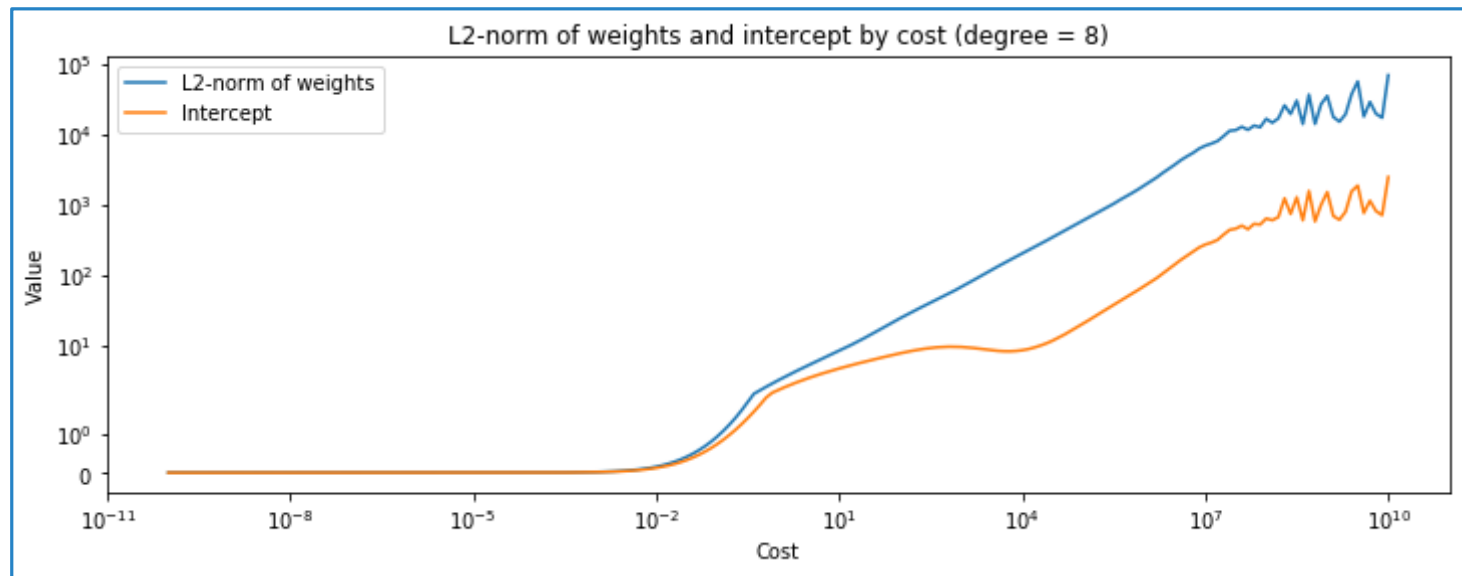




# 重みの L2-norm と切片

## 実験内容

- 8 次の特徴量まで利用
- コストパラメータの設定値を変えながら重みの L2-norm と切片を描画



# L1 正則化

---

# L1 正則化

---

疎な解を求めたいときに利用する

L2 正則化 (特徴量の二乗和) :

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^n \log(\exp(-y_i (X_i^T w + c)) + 1).$$

L1 正則化 (特徴量の絶対値の和) :

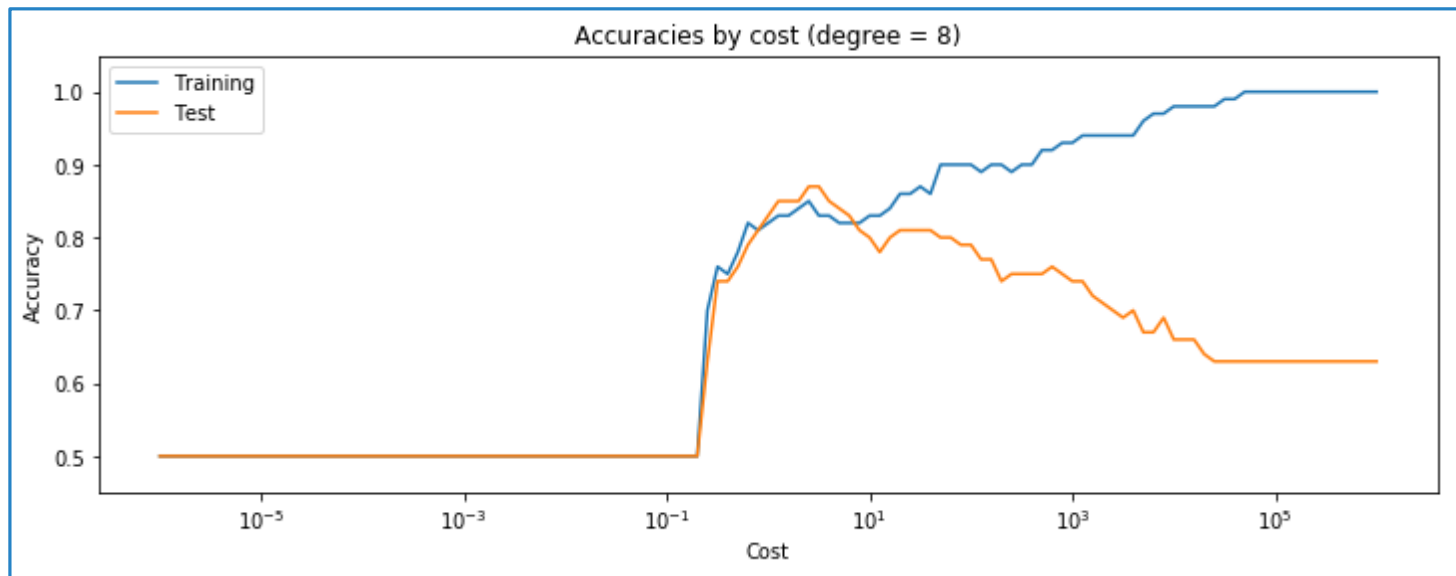
$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^n \log(\exp(-y_i (X_i^T w + c)) + 1).$$

(数式はいずれも [https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) より引用)

# コストによる正解率の変化

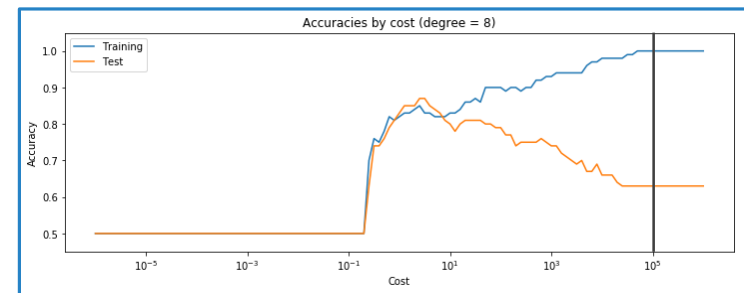
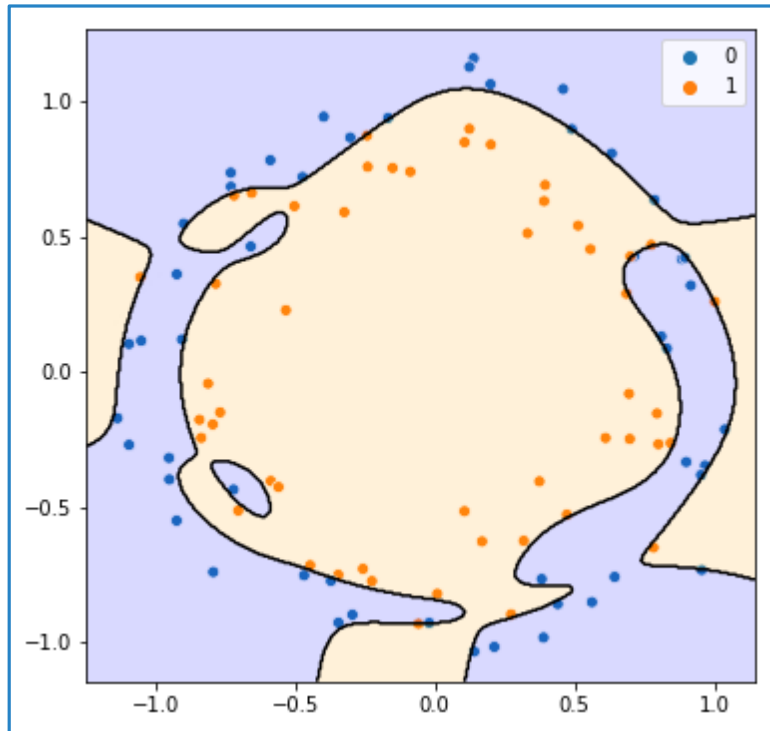
## 実験内容

- 8 次の特徴量まで利用。solver='liblinear', penalty='l1' を指定
- コストパラメータの設定値を変えながら正解率を描画



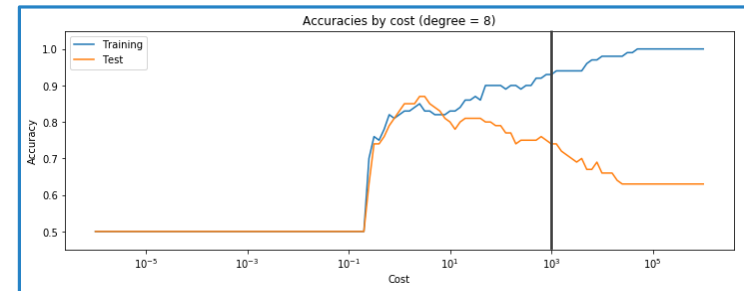
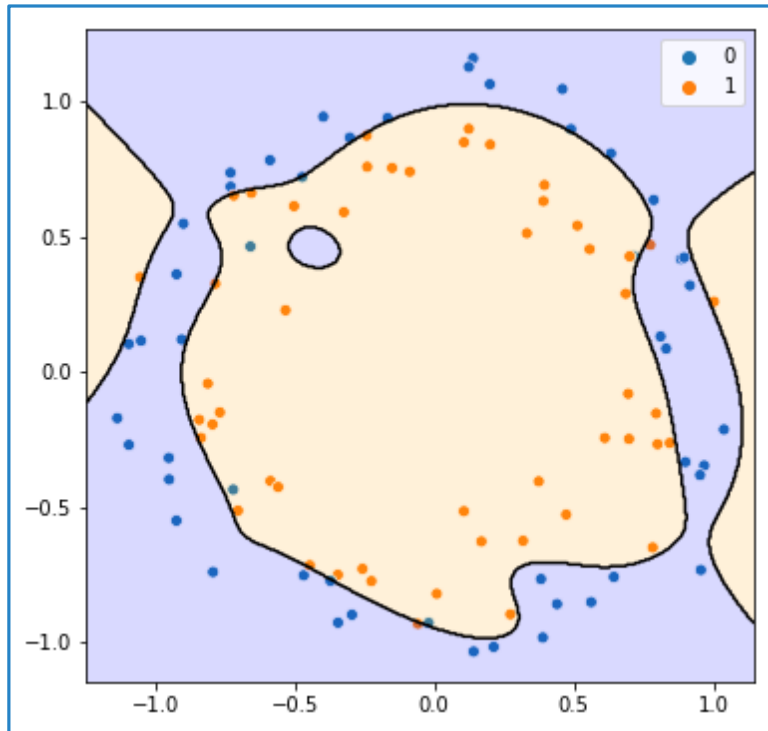
# 決定境界が変化する様子

degree=8, C=1e5



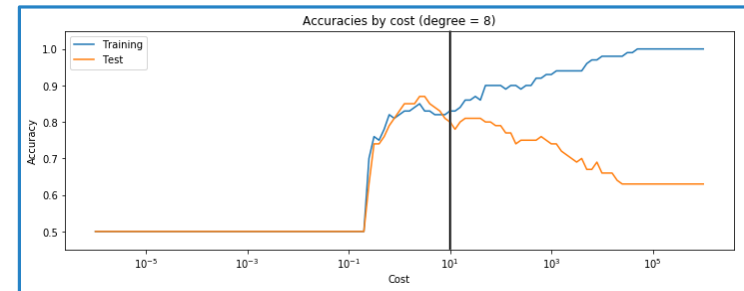
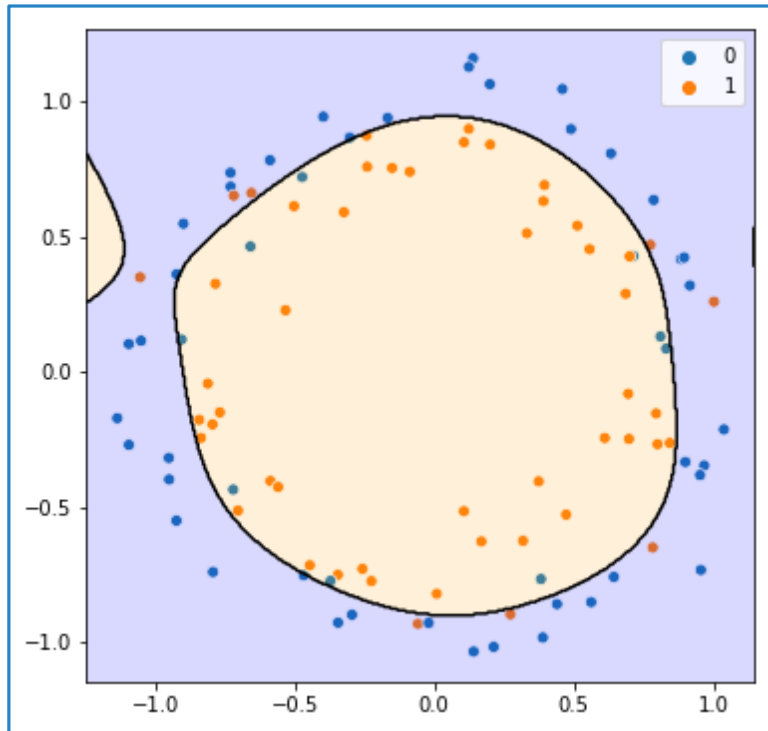
# 決定境界が変化する様子

degree=8,  $C=1e3$



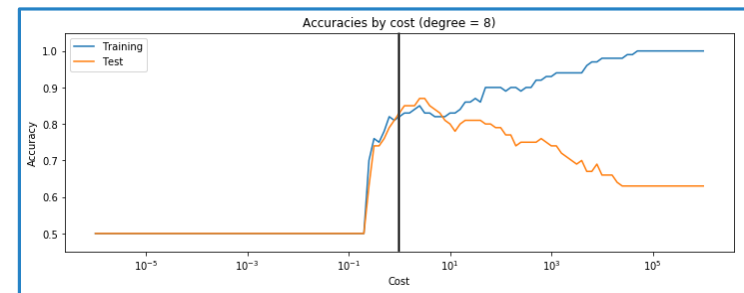
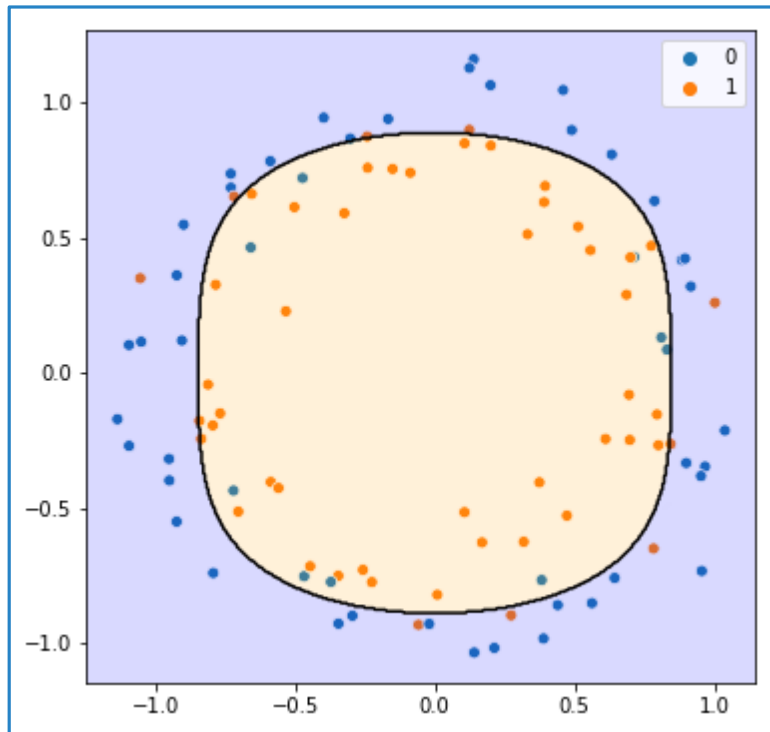
# 決定境界が変化する様子

degree=8, C=1e1



# 決定境界が変化する様子

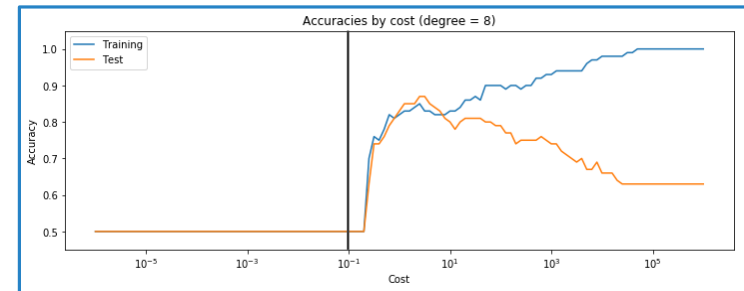
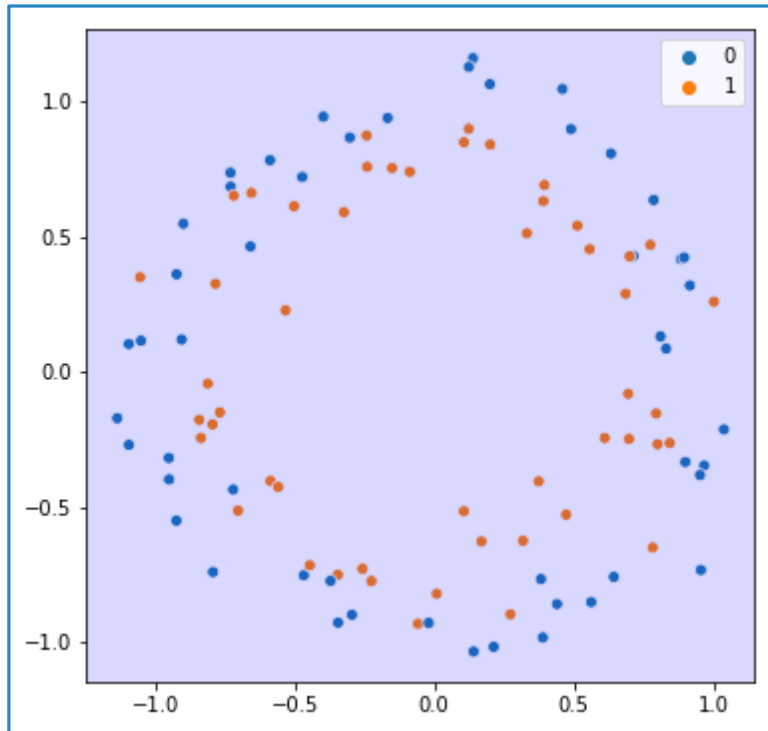
degree=8, C=1e0





# 決定境界が変化する様子

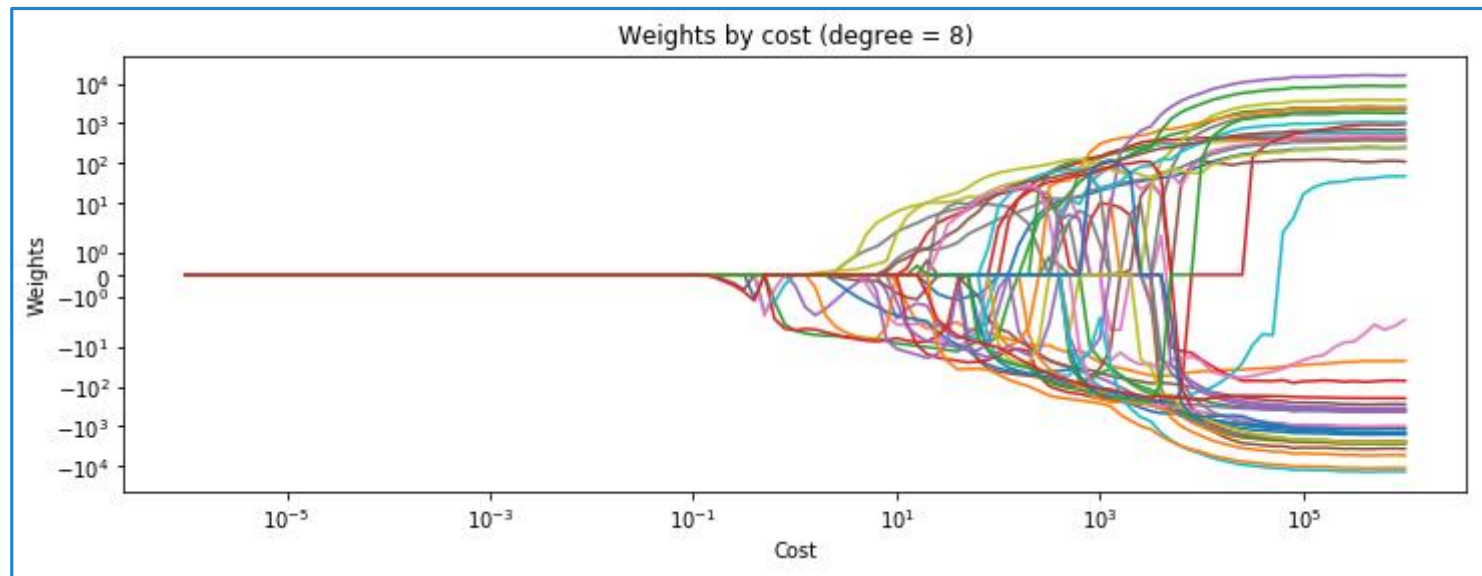
degree=8,  $C=1e-1$



# 各特徴量の重み

## 実験内容

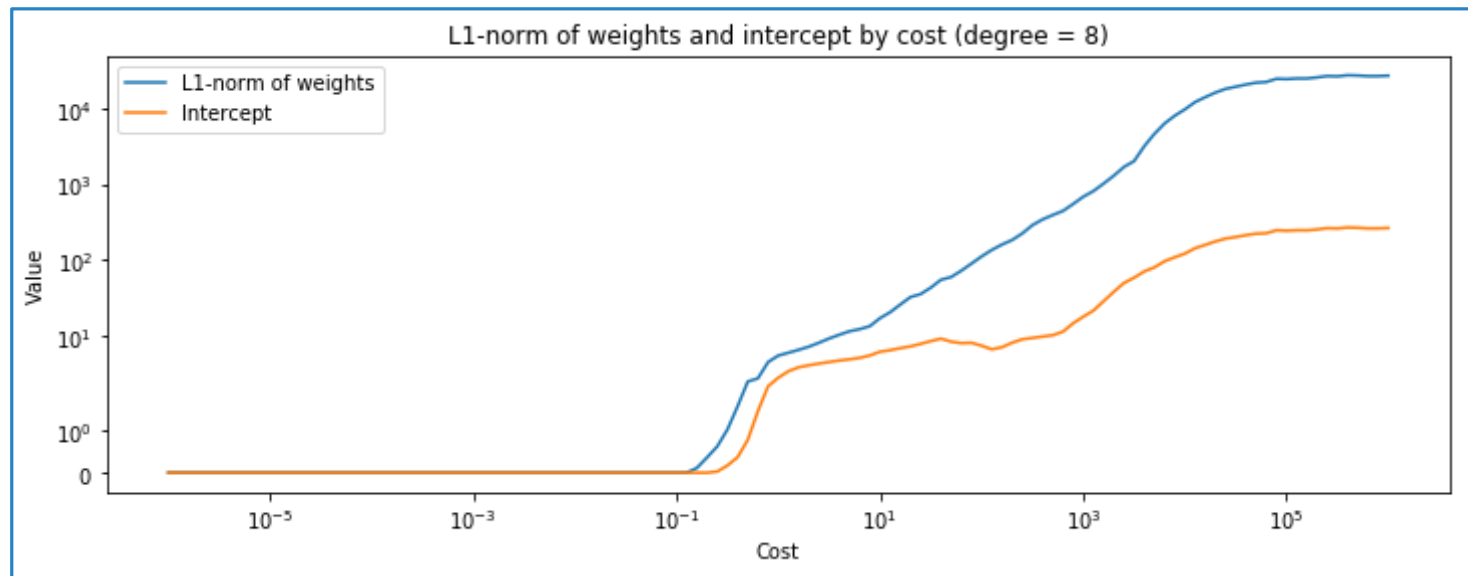
- 8 次の特徴量まで利用。solver='liblinear', penalty='l1' を指定
- コストパラメータの設定値を変えながら各特徴量の重みを描画



# 重みの L1-norm と切片

## 実験内容

- 8 次の特徴量まで利用。solver='liblinear', penalty='l1' を指定
- コストパラメータの設定値を変えながら重みの L2-norm と切片を描画



# おわり

---

& 質疑応答