教師あり学習（回帰）の応用

2.1 モデルの汎化

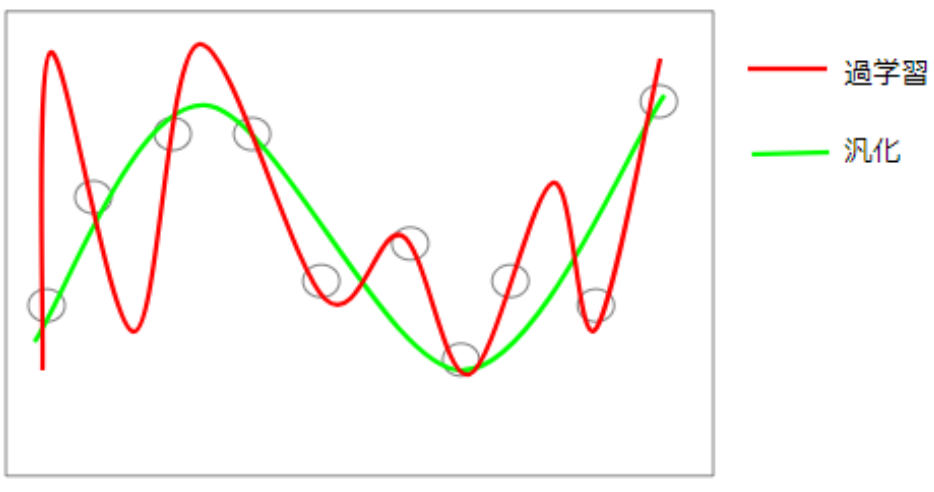
2.1.1 汎化とは

回帰分析の目的は、過去のデータからモデルを学習し、未知のデータを予測することです。 Chapter1では回帰分析を過去のデータを用いて値を予測するようにモデルを設定しました。

しかし、過去のデータは株価変動や売り上げ変動などの事象を完全に説明しているわけではありません。
データの予測には幅が存在し、また入力するデータが同じでも実際の結果が変わってしまうということもあり得ます。

過去のデータを信頼しすぎることによってデータの予測に破綻 が生じる場合があります。
これを **過学習** と呼び、予測精度が下がってしまう原因となります。

過学習を防ぐために取られるアプローチが **汎化** です。汎化を意識したモデルを作ることで、学習に使ったデータに適合しすぎず、一般的なケースに対応できるようになります。具体的な汎化手法は今後のセッションを見てみましょう。



問題

- 以下の文章のうち汎化について正しいものを選んでください。

- モデルの予測精度を下げるために汎化が行われる。
- 過去のデータに特化した予測が行えるようにすること。
- モデルによるデータの推定を一般化すること。
- モデルは複雑な方が関係性を説明できて良い。

ヒント

- 汎化によってデータの関係性はよりシンプルになる傾向があります。

解答

モデルによるデータの推定を一般化すること。

2.1.2 正則化

線形回帰では、 **汎化手法として正則化** が用いられます。 正則化とは、回帰分析を行うモデルに対し、モデルが推定したデータ同士の関係性の複雑さに対してペナルティを加えることによってモデルが推定するデータ同士の関係性を一般化しようとするアプローチです。

正則化には二種類存在し一つはL1正則化、もう一つはL2正則化です。

L1正則化 は予測に用いられるデータのうち、「予測したいデータに対する寄与が薄いデータ」や「他の予測に用いられるデータとの関係性が強いデータ」の正則化に向いており、これらを回帰分析の際に結果に対する寄与が小さくなるように係数を小さくする方法です。 **データとして余分な情報がたくさん存在するようなデータ** の回帰分析を行う際に重宝します。

L2正則化 は予測に用いるデータの範囲を算出し、データの範囲を揃えるようにデータに対する係数を小さくすることによって回帰分析のモデルの一般化を図ろうとする方法です。 データの範囲とは、データが取りうる数値の範囲のことで、揃える場合は大抵は0から1の範囲になるように調整されます。データの範囲を揃えることによって同じ尺度で全てのデータの予測に対する寄与が比較可能になり、 **滑らかなモデルを得やすい（汎化しやすい）** という特徴があります。

問題

- 正則化について述べたもののうち正しいものを選んでください。

- 予測精度を上げるためにデータに対して行う処理のこと。
- 予測に用いるデータの比重を下げること。

- L1正則化は予測するデータとの関係性が高いものを削ってモデルの説明力を高めたものです。
- L2正則化はデータの値の範囲を揃えるように係数を操作することで説明力を高めようとするアプローチです。

ヒント

- L1正則化とL2正則化の違いも理解しましょう。

解答

L2正則化はデータの値の範囲を揃えるように係数を操作することで説明力を高めようとするアプローチです。

2.1.3 ラッソ回帰

ラッソ回帰 とは **L1正則化を行いながら線形回帰の適切なパラメータを設定する回帰モデル** です。

機械学習では予測に用いるデータ同士の関連性を人間が認識しにくい場合があります。L1正則化では、 **データとして余分な情報がたくさん存在するようなデータの回帰分析を行う際** に重宝すると確認しました。そのため、データセットの数（行数）に比べて、パラメータの数（列数）が多いなどといった場合には、ラッソ回帰を利用するのが良いでしょう。

scikit-learnのlinear_modelモジュール内にあるLasso()というモデルがラッソ回帰のモデルにあたります。

```
from sklearn.linear_model import Lasso
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

X, y = make_regression(n_samples=100, n_features=100, n_informative=60, n_targets=1, ra
ndom_state=42)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

model = Lasso()
model.fit(train_X, train_y)
print(model.score(test_X, test_y))
```

線形回帰の model =LinearRegression() を model = Lasso() に変更するだけで、ラッソ回帰で分析ができます。

問題

- 正則化を行わない線形回帰とラッソ回帰のモデルの説明力の違いの比較を行います。
- 余計な情報が多分に含まれたデータを渡しますので線形回帰とラッソ回帰を行い、test_X, test_yに対する決定係数を出力してください。

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データを生成
X, y = make_regression(n_samples=100, n_features=100, n_informative=60, n_targets=1, randor
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください
# 線形回帰


# test_X, test_yに対する決定係数を出力してください

# ラッソ回帰


# test_X, test_yに対する決定係数を出力してください

```

ヒント

- 線形回帰とラッソ回帰の学習の流れは同じです。
- 今回のデータは予測に用いるデータが100のうち余分なデータが40あるデータです。

解答例

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データを生成
X, y = make_regression(n_samples=100, n_features=100, n_informative=60, n_targets=1, randor
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください
# 線形回帰
model = LinearRegression()
model.fit(train_X, train_y)
# test_X, test_yに対する決定係数を出力してください
print("線形回帰:{}".format(model.score(test_X, test_y)))
# ラッソ回帰
model = Lasso()
model.fit(train_X, train_y)
# test_X, test_yに対する決定係数を出力してください
print("ラッソ回帰:{}".format(model.score(test_X, test_y)))
```

2.1.4 リッジ回帰

リッジ回帰 とは **L2正則化を行いながら線形回帰の適切なパラメータを設定する回帰モデル** です。

リッジ回帰には、シンプルな線形回帰モデルよりも **滑らかなモデルを得やすい（汎化しやすい）** という特徴がありましたね。 scikit-learnのlinear_modelモジュール内にある Ridge() というモデルがリッジ回帰のモデルにあたります。

実装方法は、シンプルな線形回帰モデル、ラッソ回帰と全く同じでモデル名を差し替えるだけでOKです。

```
from sklearn.linear_model import Ridge
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

X, y = make_regression(n_samples=100, n_features=50, n_informative=50, n_targets=1, noi
se=100.0, random_state=42)
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

model = Ridge()
model.fit(train_X, train_y)
print(model.score(test_X, test_y))
```

問題

- 線形回帰とリッジ回帰によるモデルの説明力の違いを確認したいと思います。
- データが渡されるのでtest_X, test_yに対する決定係数を出力してください。

In [ ]:

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データを生成
X, y = make_regression(n_samples=100, n_features=50, n_informative=50, n_targets=1, noise=1
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください
# 線形回帰


# test_X, test_yに対する決定係数を出力してください

# リッジ回帰


# test_X, test_yに対する決定係数を出力してください

```

ヒント

- リッジ回帰もモデルの学習の流れは線形回帰と同じです。
- ラッソ回帰のときほど大きな違いは出ません。

解答例

In [ ]:

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# データを生成
X, y = make_regression(n_samples=100, n_features=50, n_informative=50, n_targets=1, noise=1
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42)

# 以下にコードを記述してください
# 線形回帰
model = LinearRegression()
model.fit(train_X, train_y)
# test_X, test_yに対する決定係数を出力してください
print("線形回帰:{}".format(model.score(test_X, test_y)))
# リッジ回帰
model = Ridge()
model.fit(train_X, train_y)
# test_X, test_yに対する決定係数を出力してください
print("リッジ回帰:{}".format(model.score(test_X, test_y)))
```

2.1.5 ElasticNet回帰

さて、最後にElasticNet回帰を紹介します。 **ElasticNet回帰とは、ラッソ回帰とリッジ回帰を組み合わせて正則化項を作るモデル** となります。

メリットとしては、ラッソ回帰で取り扱った **余分な情報がたくさん存在するようなデータに対して情報を取捨選択してくれる点** と、リッジ回帰で取り扱った **滑らかなモデルを得やすい（汎化しやすい）点** の組み合わせとなるので、両方のメリットをバランスよく用いてモデルを作りたい時にはベストな手法となります。

ElasticNet回帰を使ってモデルを構築する場合、いままでと同じように以下のようにモデルを呼び出せばOKです。

```
model = ElasticNet()
```

なお、scikit-learnのElasticNet()には l1_ratio という引数を指定できます。

```
model = ElasticNet(l1_ratio=0.3)
```

以上のように設定すると、L1正則化とL2正則化の割合を指定することができます。以上の場合、L1正則化が30%、L2正則化が70%効いていることを示しています。（指定しないと、丁度半々のElasticNet回帰モデルで指定されます。）

問題

- ElasticNet回帰に関して説明している以下の文章のうち **間違っているもの** を選んでください。

- Lasso回帰とRidge回帰を組み合わせて正則化項を作るモデル。
- Lasso回帰の「余分な情報がたくさん存在するようなデータに対して情報を取捨選択してくれる」メリットがある。
- Ridge回帰の「滑らかなモデルを得やすい（汎化しやすい）」メリットがある。
- L1正則化とL2正則化の割合は常に半々である。

ヒント

- l1_ratio という引数の使い方を確認しましょう。

解答

L1正則化とL2正則化の割合は常に半々である。

2.2 添削問題

これまでの課題ではかなり回帰の決定係数が高くなるように生成されたデータを用いて学習を行いました。

実際のデータは線形では決定できないくらい複雑なモデルになると思います。
整形されたデータではありますが生のデータに近いデータセットを用いた学習を行ってみましょう。

問題

- ボストンの家屋に関するデータセットが渡されます。
- ラッソ回帰やリッジ回帰を用いて決定係数を出力してください。

In [ ]:

```
# 必要なモジュールのインポート
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
# 以下に必要なモジュールを追記してください


# データの取得
boston_data = load_boston()
train_X, test_X, train_y, test_y = train_test_split(boston_data.data, boston_data.target, random_

# 以下にコードを記述してください。

```

ヒント

- 今回はモジュールのインポートを行っていません。必要なモジュールをインポートしてください。
- データセットに関する詳しい情報は print(boston_data.DESCR) を実行してみてください。

解答例

In [ ]:

```
# 必要なモジュールのインポート
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
# 以下に必要なモジュールを追記してください
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

# データの取得
boston_data = load_boston()
train_X, test_X, train_y, test_y = train_test_split(boston_data.data, boston_data.target, random_

# 以下にコードを記述してください。
# 線形回帰
model = LinearRegression()
model.fit(train_X, train_y)
print("線形回帰:{}".format(model.score(test_X, test_y)))

# ラッソ回帰
model = Lasso()
model.fit(train_X, train_y)
print("ラッソ回帰:{}".format(model.score(test_X, test_y)))

# リッジ回帰
model = Ridge()
model.fit(train_X, train_y)
print("リッジ回帰:{}".format(model.score(test_X, test_y)))
```

解答例の三種類では線形回帰が一番モデルを説明できているという結果になりました。
整形がすでに行われているためさらに正則化を行う必要がなかったと思われます。