非線形回帰モデル (sklearn)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn import model_selection, preprocessing, linear_model, svm
from sklearn.model_selection import train_test_split

from tensorflow.keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
```

In [2]:

```
# seabornの設定
sns.set()
# 背景変更
sns.set_style("darkgrid", {'grid.linestyle': '--'})
# 大きさ(スケール変更)
sns.set_context("paper")
```

true_func関数：

$z = 1 - 48x + 218x^2 - 315x^3 + 145x^4$

In [3]:

```
def true_func(x):
    """ z = 1 - 48x + 218x^2 - 315x^3 + 145x^4 """
    z = 1 - 48 * x + 218 * x**2 - 315 * x**3 + 145 * x**4
    return z
```

真の関数からノイズを伴うデータを生成

In [4]:

```
data_num = 100  # データ数
```

In [5]:

```
# 真のデータ
true_data = np.linspace(0, 1, data_num)
true_target = true_func(true_data)
```

In [6]:

```
# 真の関数からデータ生成
data = np.random.rand(data_num).astype(np.float32)  # 真のデータに乱数を付与
data = np.sort(data)
target = true_func(data)
```

In [7]:

```
# 説明変数と目的変数のサイズを確認
print(data.shape)
print(target.shape)
```

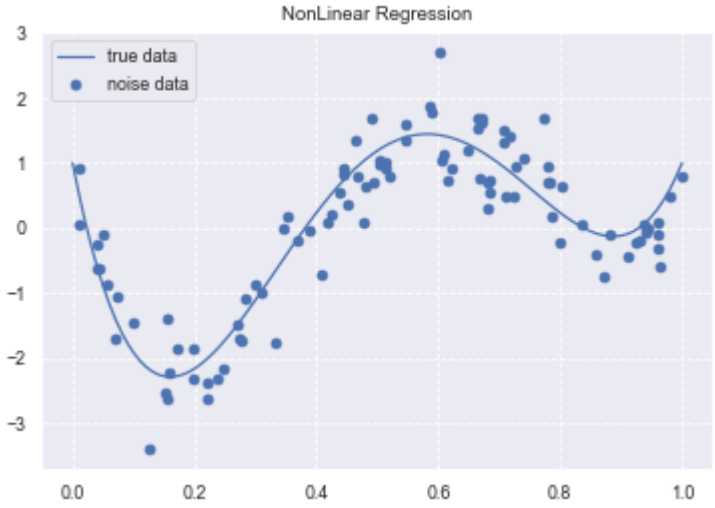
```
(100,)
(100,)
```

In [8]:

```
# ノイズを加える
noise = 0.5 * np.random.randn(data_num)
target = target  + noise
```

In [9]:

```
# ノイズ付きデータを描画
plt.plot(true_data, true_target, label='true data')
plt.scatter(data, target, label='noise data')
plt.title('NonLinear Regression')
plt.legend(loc=2)
plt.show()
```



線形回帰による予測を行う

In [10]:

```
clf = LinearRegression()  # 線形回帰モデルのオブジェクト生成
```

In [11]:

```
# clfに入力するため、二次元データに変換する
reshaped_data = data.reshape(-1,1)
reshaped_target = target.reshape(-1,1)
```

In [12]:

```
print(reshaped_data.shape)
print(reshaped_target.shape)
```

```
(100, 1)
(100, 1)
```

In [13]:

```
clf.fit(reshaped_data, reshaped_target)  # モデルの学習を行う
```

Out[13]:

```
LinearRegression()
```

In [14]:

```
pred_target = clf.predict(reshaped_data)  # 学習したモデルから予測を行う
```

In [15]:

```
print(pred_target.shape)
```

```
(100, 1)
```

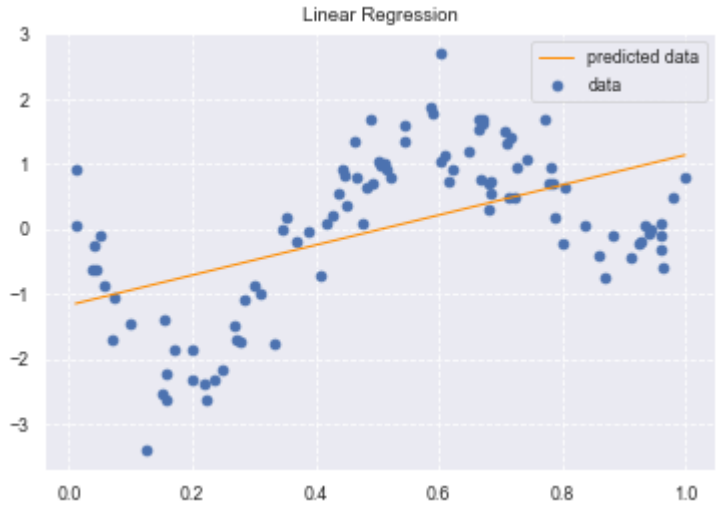
In [16]:

```
score = clf.score(reshaped_data, reshaped_target)
print(f'決定係数: {score}')
```

```
決定係数: 0.27080805038113176
```

In [17]:

```
# 元のデータは散布図で描画する
plt.scatter(data, target, label='data')
# 線形回帰モデルによる予測値は直線で描画する
plt.plot(data, pred_target, color='darkorange',
         marker='', linestyle='-', linewidth=1, markersize=6, label='predicted data')
plt.title('Linear Regression')
plt.legend()
plt.show()
```



考察：
非線形のデータセットに対しては、線形回帰ではうまく予測ができていない。
決定係数の低さからもうかがえる。

カーネルリッジ回帰

In [18]:

```
kr_model = KernelRidge(alpha=0.0002, kernel='rbf')  # カーネルリッジモデルのオブジェクト作成
```

In [19]:

```
kr_model.fit(reshaped_data, reshaped_target)  # モデルを学習
```

Out[19]:

```
KernelRidge(alpha=0.0002, kernel='rbf')
```

In [20]:

```
pred_target = kr_model.predict(reshaped_data)  # 学習したモデルから予測
```

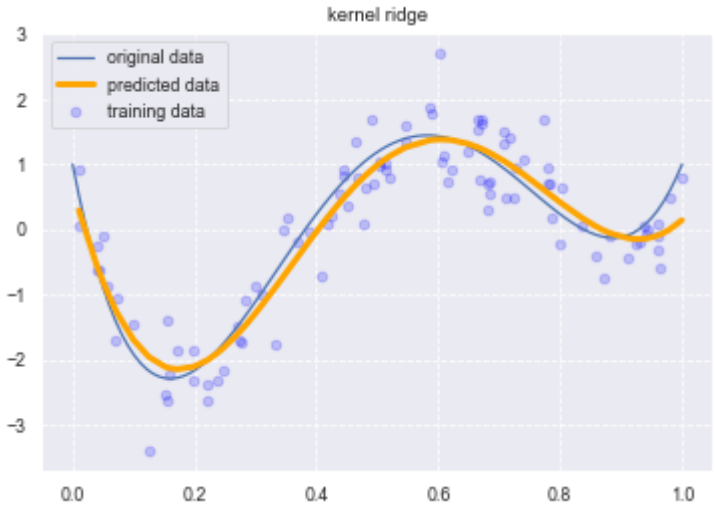
In [21]:

```
score = kr_model.score(reshaped_data, reshaped_target)
print(f'決定係数: {score}')
```

決定係数: 0.8632312123447169

In [44]:

```
plt.plot(true_data, true_target, label='original data')
# 学習用データのプロットは散布図
plt.scatter(data, target, label='training data', color='blue', alpha=0.2)
# 予測データのプロットは曲線
plt.plot(data, pred_target, color='orange', linestyle='-', linewidth=3, markersize=6, label='predicted data')
plt.title('kernel ridge')
plt.legend()
plt.show()
```



考察：
予測値は、ノイズつきの学習用データよりも、元データに対してうまくフィットしていることがわかる。
学習用データではx=0.12辺りやx=0.6辺りに元データと大きく離れたデータ点があるが、予測値はそういった値に引っ張られていない。

Ridge回帰

In [23]:

```
kx = rbf_kernel(X=reshaped_data, Y=reshaped_data, gamma=50)  # rbfカーネル
clf = Ridge(alpha=30)
```

In [24]:

```
clf.fit(kx, target)  # モデル学習
```

Out[24]:

Ridge(alpha=30)

In [25]:

```
pred_ridge = clf.predict(kx)  # 予測
```

In [26]:

```
print(clf.score(kx, target))  # 決定係数を表示
```

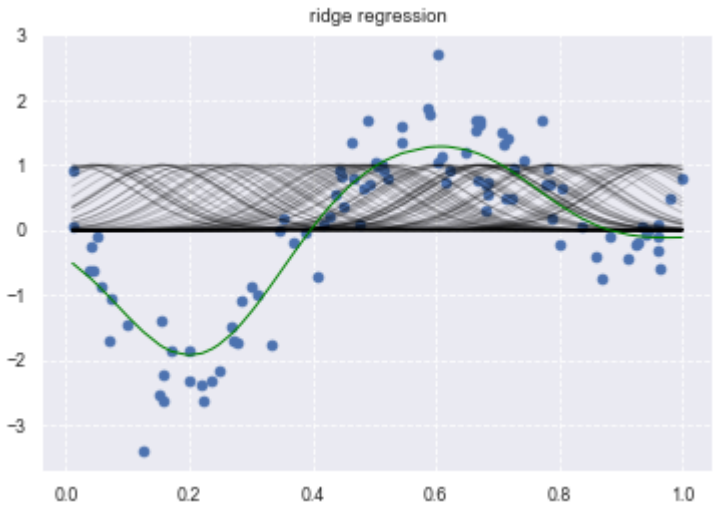
0.829307885082686

In [27]:

```
plt.scatter(reshaped_data, reshaped_target, label='data')

for i in range(len(kx)):
    plt.plot(reshaped_data, kx[i], color='black', linestyle='-', linewidth=1, markersize=3, label=
'rbf', alpha=0.2)

plt.plot(reshaped_data, pred_ridge, color='green', linestyle='-', linewidth=1, markersize=3, la
bel='ridge regression')
plt.title('ridge regression')
plt.show()
```



Ridge回帰も、カーネルリッジ回帰に近い結果となった。
ただし、決定係数はカーネルリッジ回帰よりも多少悪い。

多項式回帰

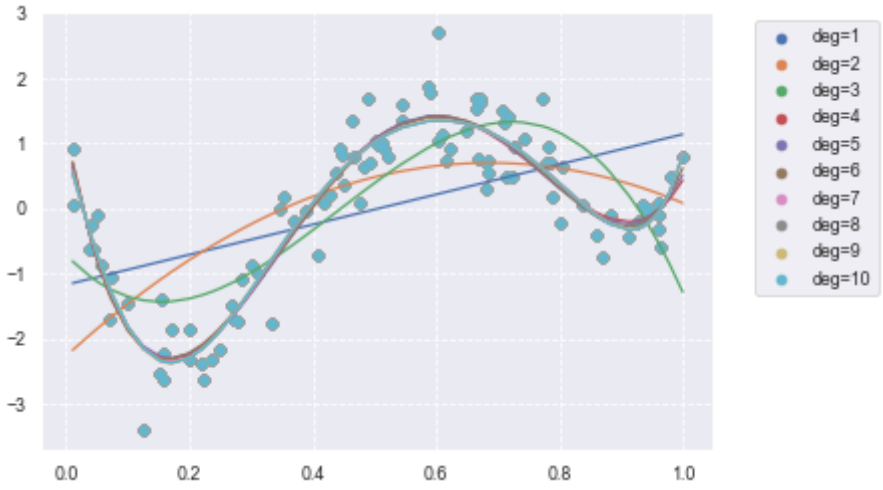
In [28]:

```
deg = [1,2,3,4,5,6,7,8,9,10]

for d in deg:
    regr = Pipeline([
        ('poly', PolynomialFeatures(degree=d)),
        ('linear', LinearRegression())
    ])
    regr.fit(reshaped_data, reshaped_target)  # モデルの学習
    pred_poly = regr.predict(reshaped_data)  # 予測

    # 結果の描画
    plt.scatter(reshaped_data, reshaped_target, label=f'deg={d}')
    plt.plot(reshaped_data, pred_poly)

plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



考察：上記グラフから、以下のことがわかる。

- deg=1 では直線になっており当てはまりが悪い。
- deg=2,3,4 になるにつれて当てはまりが良くなってくる。
- deg>=5 だとほぼ同じようなグラフになる。

Lasso回帰

In [29]:

```
kx = rbf_kernel(X=reshaped_data, Y=reshaped_data, gamma=5)  # RBFカーネル
```

In [113]:

```
lasso_clf = Lasso(alpha=0.001, max_iter=10000)  # Lassoオブジェクト作成
lasso_clf.fit(kx, reshaped_target)  # モデルの学習
```

```
e:\venv\datascience\lib\site-packages\sklearn\linear_model\_coordinate_descen
t.py:531: ConvergenceWarning: Objective did not converge. You might want to i
ncrease the number of iterations. Duality gap: 0.07034873962402344, toleranc
e: 0.015521559864282608
  positive)
```

Out[113]:

```
Lasso(alpha=0.001, max_iter=10000)
```

In [114]:

```
pred_lasso = lasso_clf.predict(kx)  # 予測
```

In [115]:

```
print(lasso_clf.coef_)
```

```
[-0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -5.1600657e+00 -6.8605366e+00
 -8.3150828e-01 -1.9046776e-01 -2.7367422e-01 -2.0178556e+00
 -1.9545608e+00 -6.6051534e-03 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
  0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
  0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
  0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
  0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00 -0.0000000e+00
 -0.0000000e+00 -2.3157418e+00 -9.9326622e-01 -1.3880043e+00
 -1.2993487e+00 -6.4537477e-01 -2.5427449e+00 -1.0036070e-01
 -2.0385225e-01 -5.2772439e-01 -2.4007287e+00 -2.8278468e+00]
```

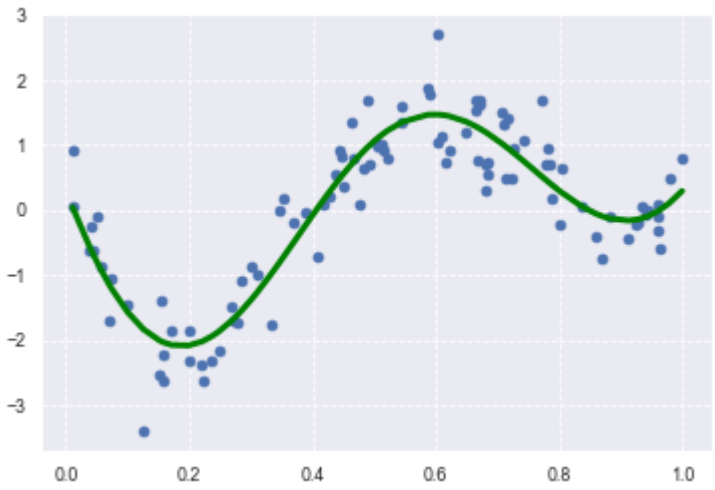
In [116]:

```
print(lasso_clf.score(kx, reshaped_target))
```

```
0.8574366938491821
```

In [117]:

```
# 元の学習用データ
plt.scatter(reshaped_data, reshaped_target)
# 予測したデータ
plt.plot(reshaped_data, pred_lasso, color='green', linestyle='-', linewidth=3, markersize=3)
plt.show()
```



考察： alphaの値を変化させることで、スコアが変化する。

- alpha=1.0 のとき、score=0.0
- alpha=0.1 のとき、score=0.5048101864787449
- alpha=0.01 のとき、score=0.7292533548899374
- alpha=0.001 のとき、score=0.8574366938491821
- alpha=0.0001 のとき、score=0.860314306317248

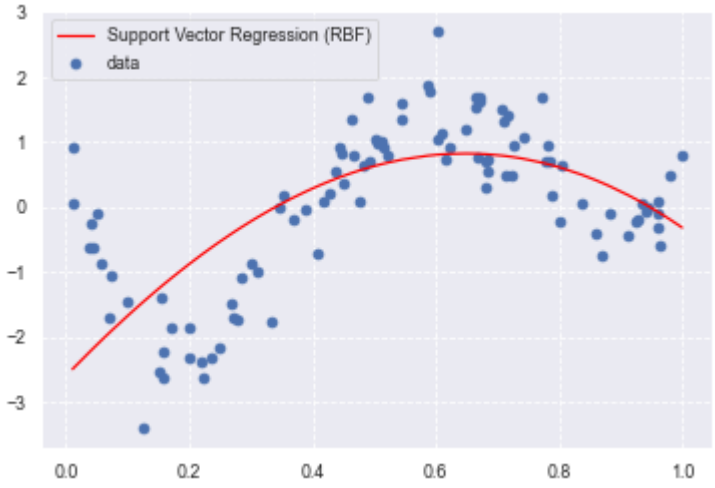
SVR-rbf

In [33]:

```
clf_svr = svm.SVR(kernel='rbf', C=1e3, gamma=0.1, epsilon=0.1)
clf_svr.fit(reshaped_data, reshaped_target)
pred_rbf = clf_svr.predict(reshaped_data)
```

In [34]:

```
plt.scatter(reshaped_data, reshaped_target, label='data')
plt.plot(reshaped_data, pred_rbf, color='red', label='Support Vector Regression (RBF)')
plt.legend()
plt.show()
```



In [35]:

```
X_train, X_test, y_train, y_test = train_test_split(
    reshaped_data, reshaped_target, test_size=0.1, random_state=0)
```

In [36]:

```
cb_cp = ModelCheckpoint(
    r'/out/weights.{epoch:02d}-{val_loss:.2f}.hdf5',
    verbose=1,
    save_weights_only=True)
```

In [37]:

```
cb_tf  = TensorBoard(
    log_dir='/out/tensorBoard',
    histogram_freq=0)
```

In [38]:

```
def relu_reg_model():
    """モデルの構築"""
    model = Sequential()
    model.add(Dense(10, input_dim=1, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='linear'))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

In [39]:

```
estimator = KerasRegressor(build_fn=relu_reg_model, epochs=100, batch_size=5, verbose=1
)
```

In [41]:

```
y_pred = estimator.predict(X_train)
```

18/18 [==============================] - 0s 4ms/step

In [45]:

```
plt.plot(reshaped_data, true_func(reshaped_data), label='original data')
plt.plot(reshaped_data, reshaped_target, '.', label='training data')
plt.plot(X_train, y_pred, "o", label='predicted keras')
plt.title('NonLiner Regressions via DL by ReLU')
plt.legend(loc='lower right')
plt.show()
```

考察：

kerasによるディープラーニング予測結果は、元データにうまくフィットしている所もあれば、$0.1 \leq x \leq 0.25$ 辺りのように元データからずれている部分もある。