線形回帰モデル

線形回帰モデルとは説明変数の線形変換により目的変数を回帰する手法である。

教師有学習の1つであり、目的変数と予測値との誤差が最も小さくなるように、最小二乗法により重みwを求める。

■回帰

説明変数に演算を施すことにより目的変数を導くこと。この時、目的変数を導く関数fを回帰モデルと呼ぶ。

■線形変換

ベクトルx=(x1, x2, …, xn)の線形変換とは、xの各成分の一次結合により新たな変数を得ることである。すなわち

$y = w1x1 + w2x2 + , \cdots, +wnxn + b$

により、xベクトルの線形変換yを得ることができる。

上記関数をグラフ化すると、超平面（※xが単成分の場合は直線）で表すことができる。

線形変換は、重みベクトルwとxとの内積と一致する。

■実装演習（ボストン住宅データセットを元に、部屋数と犯罪率から住宅価格を線形回帰）

予測例）犯罪率0.3, 部屋数4の住宅価格は？

```
In [21]: # 説明変数
         data2 = df.loc[:, ['CRIM', 'RM']].values
         # 目的変数
         target2 = df.loc[:, 'PRICE'].values

In [22]: # オブジェクト生成
         model2 = LinearRegression()

In [23]: # fit関数でパラメータ推定
         model2.fit(data2, target2)

Out[23]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [24]: model2.predict([[0.3, 4]])

Out[24]: array([4.24007956])
```

約4,200ドルと予測された。

次に、このモデルの予測精度を下記で算出

```
# 平均二乗誤差を評価するためのメソッドを呼び出し
from sklearn.metrics import mean_squared_error
# 学習用、検証用データに関して平均二乗誤差を出力
print('MSE Train : %.3f, Test : %.3f' % (mean_squared_error(y_train, y_train_pred), mean_squared_error(y_test, y_test_pred)))
# 学習用、検証用データに関してR^2を出力
print('R^2 Train : %.3f, Test : %.3f' % (model.score(X_train, y_train), model.score(X_test, y_test)))
```

```
MSE Train : 44.983, Test : 40.412
R^2 Train : 0.500, Test : 0.434
```

Testデータに対するR2が0.43、すなわちTestデータの住宅価格の43%しか上記モデルでは説明できてないと言える。これは住宅価格を部屋数と犯罪率のみから線形回帰することに無理があるからと考える。

非線形回帰モデル

非線形回帰モデルとは説明変数の非線形変換により目的変数を回帰する手法である。

非線形変換は、説明変数 xj を種々の関数で変換したϕj(xj)の線形結合を施すことで行われる（基底展開法）。

$y = w1\phi1(x1) + w2\phi2(x2) + \cdots + wn\phi n(xn) + b$

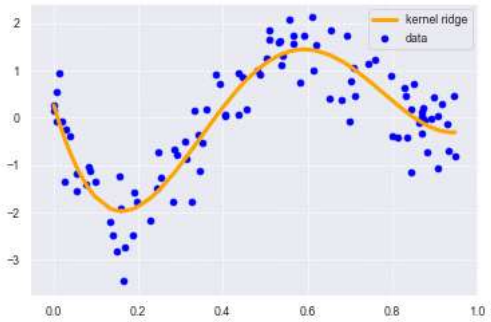
教師有学習の１つであり、基底展開法の場合、最尤法により重み w を求めることができる。

■実装演習（4次関数に従うデータの予測）

rbf[基底放射関数]による基底展開で非線形回帰

In [12]:
```
from sklearn.kernel_ridge import KernelRidge

clf = KernelRidge(alpha=0.0002, kernel='rbf')
clf.fit(data, target)

p_kridge = clf.predict(data)

plt.scatter(data, target, color='blue', label='data')

plt.plot(data, p_kridge, color='orange', linestyle='-', linewidth=3, markersize=6, label='kernel ridge')
plt.legend()
#plt.plot(data, p, color='orange', marker='o', linestyle='-', linewidth=1, markersize=6)
```

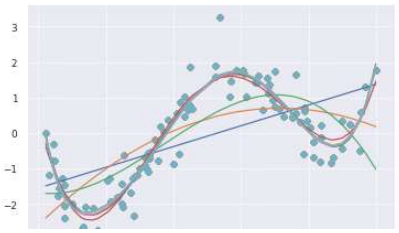
Out[12]: <matplotlib.legend.Legend at 0x19f09a1b108>



元は4次関数に基づくデータであるが、KernelRidgeによりガウスカーネルを線形結合することで、データにかなりフィットした予測モデルを得ることができた。一般に、無限個のガウスカーネルの線形結合で任意の非線形関数を任意の精度で近似できることが知られている。

多項式関数による基底展開で非線形回帰

```
#PolynomialFeatures(degree=1)

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(data, target)
    # make predictions
    p_poly = regr.predict(data)
    # plot regression result
    plt.scatter(data, target, label='data')
    plt.plot(data, p_poly, label='polynomial of degree %d' % (d))
```



4次関数に基づくデータのため、4次以上の多項式でデータに十分にフィットしたモデルが得られる。

ロジスティック回帰モデル

ロジスティック回帰モデルとは説明変数に対し、線形変換→シグモイド関数という 2 段階の変換をかけることで(0, 1)の範囲で目的変数を回帰する手法である。

シグモイド関数 $\sigma(x) = \{1 + \exp(-ax)\}^{-1}$

ロジスティック回帰モデルでは確率値を予測できるよう、シグモイド関数は全実数範囲に対して(0 ,1)の範囲の出力を行う関数となっている。

■実装演習（タイタニック予測：30歳男性）

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean')
x_tr = x.copy()
x_tr.loc[:, ['Age']] = imp.fit_transform(x_tr.loc[:, ['Age']])
x_tr.head(2)
```

Out[45]:

	Sex	Age
0	1	22.0
1	0	38.0

```
In [40]: from sklearn.linear_model import LogisticRegression
         clf = LogisticRegression()
         clf.fit(x_tr, y)
```

```
C:¥Users¥owner¥Anaconda3¥lib¥site-packages¥sklearn¥linear_model¥logistic.py:432: Future
0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
Out[40]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='warn', n_jobs=None, penalty='l2',
                            random_state=None, solver='warn', tol=0.0001, verbose=0,
                            warm_start=False)
```

```
In [43]: clf.predict([[1, 30]])
```

```
Out[43]: array([0], dtype=int64)
```

予測値は0(死亡)が返った。

なおタイタニックデータのように、実際のデータには欠損値が含まれることが多く、今回の演習では、年齢の欠損値を平均値補完することでモデルを構築した。また、分類問題の評価指標としては正解率ではなく、ROCやPRCの下側面積（AUC）で評価する点も重要である。

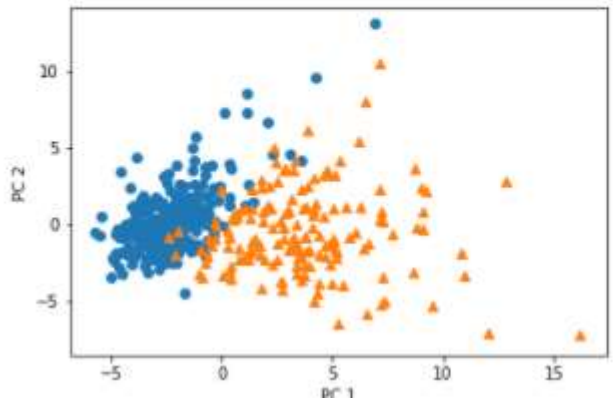
主成分分析

主成分分析（PCA）は複数変数のもつ情報をなるべく損なうことなく、別のより少数の変数に変換する教師なし学習手法である。機械学習においては説明変数が多すぎると計算量が増えるだけでなく、Variance増加による汎化性能（予測精度）の低下が問題となる。そのため、主成分分析等の方法により説明変数の数を減らすことが、汎化性能向上にとってしばしば有効である。

■実装演習（2値分類問題におけるPCAの活用）

30個の説明変数をPCAで2変数に圧縮しても2値分類が上手くできている例

```
n [12]: # PCA
        # 次元数2まで圧縮
        pca = PCA(n_components=2)
        X_train_pca = pca.fit_transform(X_train_scaled)
        print('X_train_pca shape: {}'.format(X_train_pca.shape))
        # X_train_pca shape: (426, 2)

        # 寄与率
        print('explained variance ratio: {}'.format(pca.explained_variance_ratio_))
        # explained variance ratio: [ 0.43315126  0.19586506]

        # 散布図にプロット
        temp = pd.DataFrame(X_train_pca)
        temp['Outcome'] = y_train.values
        b = temp[temp['Outcome'] == 0]
        m = temp[temp['Outcome'] == 1]
        plt.scatter(x=b[0], y=b[1], marker='o') # 良性は〇でマーク
        plt.scatter(x=m[0], y=m[1], marker='^') # 悪性は△でマーク
        plt.xlabel('PC 1') # 第1主成分をx軸
        plt.ylabel('PC 2') # 第2主成分をy軸

        X_train_pca shape: (426, 2)
        explained variance ratio: [0.43315126 0.19586506]

Out[12]: Text(0, 0.5, 'PC 2')
```



PCAの結果、新しい2成分の情報量はもともとの説明変数の30個の説明変数の約63%(寄与率の和)である。それでもグラフ中の「〇、△」で示される通り、2クラスがこの2成分で上手く分類できることがわかる。つまりこの例ではPCAにより計算量を削減できる上、次元削減効果により、過学習を防ぐことが期待できる。

アルゴリズム

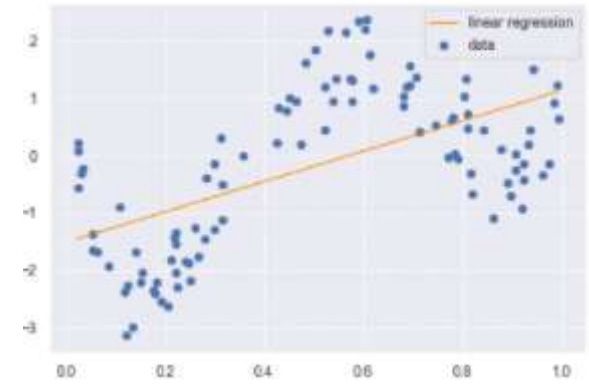
アルゴリズムとは、説明変数から目的変数を求めるための演算手法である。説明変数に種々のアルゴリズムを施すことで、目的変数を得ることができる。アルゴリズム自体は、線形変換、規定展開法、ロジスティック回帰などSVMなど様々あるが、どのアルゴリズムで目的変数を予測しようとするかは各エンジニアが任意に定める。機械学習ではそれぞれのアルゴリズムにおいて、予測精度が最も高くなるようなパラメータをモデルが学習する。

一般的には、1つの課題に対して種々のアルゴリズムで予測をし、最も精度がよいものを採択する、または各予測の平均値や最頻値を採択する（アンサンブル学習）ことが多い。

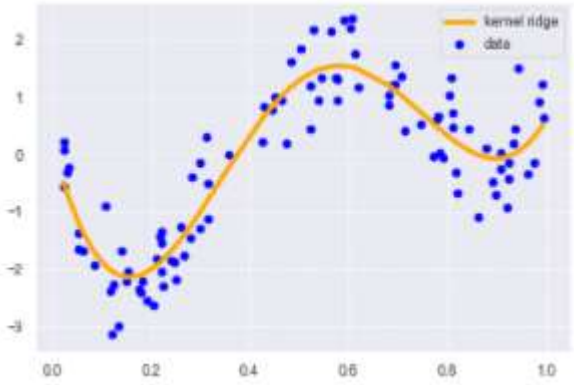
■実装演習（4次関数に従うデータの予測）

下記の通り、同じデータを予測するにあたり、用いるモデル（アルゴリズム）の違いにより結果は大きく異なり、予測の良し悪しに直結する。

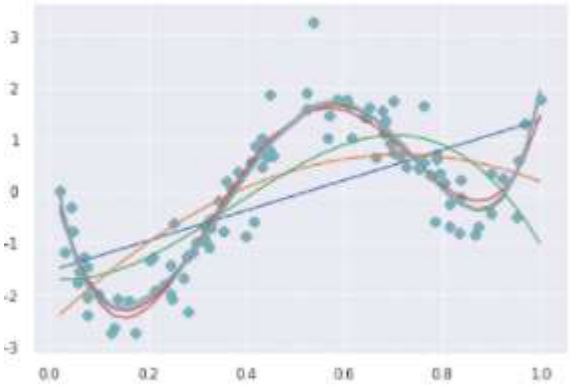
1）線形回帰



2）RBFによる非線形回帰



3）多項式関数による非線形回帰



サポートベクターマシン

サポートベクターマシン（SVM）とは、教師有2値分類モデルの1つであり、教師データを分割する超平面を想定するモデルである。数学的には、各クラスの最近傍のデータとの距離（マージン）を最大化するような超平面を求めることで学習を行う。全ての教師データに対して誤分類を一切許さないハードマージン、誤分類を許容するソフトマージンがある。また、線形分離では解けない問題に対

しては、変数を非線形変換することで、変換後の変数で線形分類（変換前の元の変数で非線形分類）が可能となることがある。この時に使われる計算上の手法がカーネルトリックである。

■実装演習（線形分離可能な2クラスデータを線形SVMによって分離）

学習フェーズでは、サポートベクトルとの距離（マージン）が最大化するような決定境界および判別関数を、勾配降下法により目的関数を最大化することで求める。その後、予測フェーズでは得られた判別関数をデータに当てはめることで、予測値（どちらのクラスか）が出力される

```
Out[12]: <matplotlib.quiver.Quiver at 0x7f49d3114e10>
```