机器学习

实验三：支持向量机模型

报告提交时间：6.4（下下周三）前

提交方式：将压缩包发送至3120245650@bit.edu.cn

实验任务：

- 1. 使用SVC模型完成对手写数字的分类（load_digits)，并使用评测指标precision_score、recall_score、f1_score对分类结果评测
- 2. 使用SVR模型实现对加州房价的预测(fetch_california_housing)，并使用r2-score 对回归结果评测
- 3. 自定义核式回归，并对提供数据进行拟合

实验环境：

- Python 3.0以上版本，开发工具任选。
- 使用scikit-learn包中的机器学习模型
- 安装说明：https://scikit-learn.org/stable/install.html#installation-instructions

实验要求和评分标准

- 完成实验中的3个任务（前两个3分，后一个4分）

实验要求和评分标准

加分项：

- 使用GridSearchCV对实验任务一和二的模型调参，并将最佳参数和评分结果输出。（1分）
- 探究核函数对自定义核式回归的影响（1分）
- 多类分类问题：探索不同的策略如一对多（OvR）或一对一（OvO）策略，对比结果并分析。（1分）

作业提交内容：

- 所有代码文件
- 每个模型运行后的结果截图（保存到word中，word文档的命名规则为：学号-姓名.docx）

实验步骤

https://scikit-learn.org/

- 加载数据集
- 拆分数据集
- 构建模型
- 获取在训练集中的模型
- 在测试集上预测结果
- 模型评测

数据集

fetch_california_housing	Load the California housing dataset (regression).
fetch_covtype	Load the covertype dataset (classification).
fetch_file	Fetch a file from the web if not already present in the local folder.
fetch_kddcup99	Load the kddcup99 dataset (classification).
fetch_lfw_pairs	Load the Labeled Faces in the Wild (LFW) pairs dataset (classification).
fetch_lfw_people	Load the Labeled Faces in the Wild (LFW) people dataset (classification).
fetch_olivetti_faces	Load the Olivetti faces data-set from AT&T (classification).
fetch_openml	Fetch dataset from openml by name or dataset id.
fetch_rcv1	Load the RCV1 multilabel dataset (classification).
fetch_species_distributions	Loader for species distribution dataset from Phillips et.
get_data_home	Return the path of the scikit-learn data directory.
load_breast_cancer	Load and return the breast cancer wisconsin dataset (classification).
load_diabetes	Load and return the diabetes dataset (regression).
load_digits	Load and return the digits dataset (classification).

https://scikit-learn.org/stable/api/sklearn.datasets.html

拆分数据集

sklearn.model_selection.train_test_split

```
sklearn.model_selection.train_test_split(*arrays, **options)
```
[source]

```
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.2,random_state=10)
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split

SVM分类

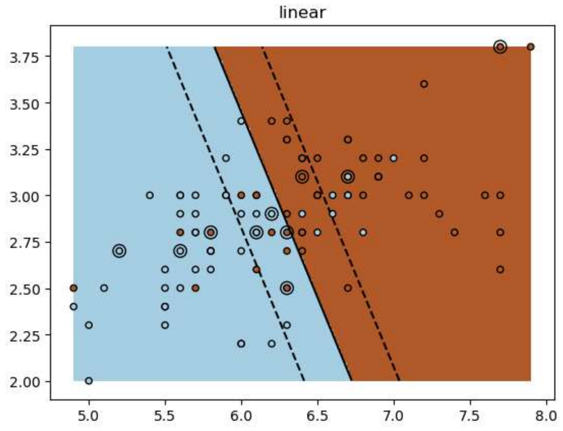
sklearn.svm.SVC

class sklearn.svm.SVC(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$

$$\zeta_i \geq 0, i = 1, ..., n$$

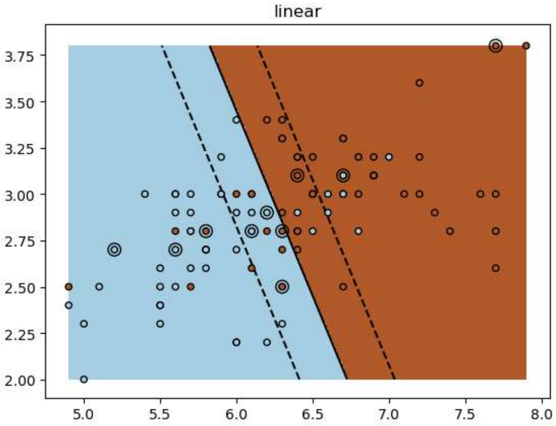


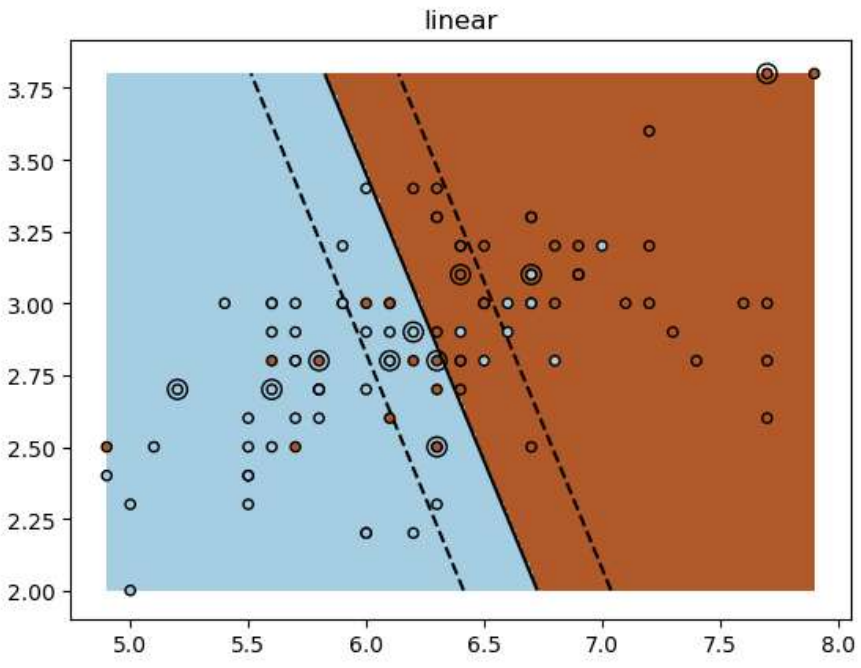
SVM分类

`sklearn.svm.SVC`

class `sklearn.svm.SVC`(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

- linear: $\langle x, x'\rangle$.
- polynomial: $(\gamma\langle x, x'\rangle + r)^d$, where d is specified by parameter `de`
- rbf: $\exp(-\gamma\|x - x'\|^2)$, where γ is specified by parameter `gamma`,
- sigmoid $\tanh(\gamma\langle x, x'\rangle + r)$, where r is specified by `coef0`.





support_ : *ndarray of shape (n_SV,)*

Indices of support vectors.

support_vectors_ : *ndarray of shape (n_SV, n_features)*

Support vectors.

n_support_ : *ndarray of shape (n_class,), dtype=int32*

Number of support vectors for each class.

分类问题中的评价指标

Binary classification

	Actual class (observation)	
Predicted class (expectation)	tp (true positive) Correct result	fp (false positive) Unexpected result
	fn (false negative) Missing result	tn (true negative) Correct absence of result

$$\text{precision} = \frac{tp}{tp + fp},$$

$$\text{recall} = \frac{tp}{tp + fn},$$

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}.$$

Multiclass and multilabel classification

- y the set of *predicted* $(sample, label)$ pairs
- $\hat{y}$ the set of *true* $(sample, label)$ pairs
- L the set of labels
- S the set of samples
- y_s the subset of y with sample s, i.e. $y_s := \{(s', l) \in y | s' = s\}$
- y_l the subset of y with label l
- similarly, $\hat{y}_s$ and $\hat{y}_l$ are subsets of $\hat{y}$
- $P(A, B) := \frac{|A \cap B|}{|A|}$ for some sets A and B
- $R(A, B) := \frac{|A \cap B|}{|B|}$ (Conventions vary on handling $B = \emptyset$; this implementation uses $R(A, B) := 0$, and similar for P.)
- $F_\beta(A, B) := \left(1 + \beta^2\right) \frac{P(A,B) \times R(A,B)}{\beta^2 P(A,B) + R(A,B)}$

`average`	**Precision**	**Recall**	**F_beta**
`"micro"`	$P(y, \hat{y})$	$R(y, \hat{y})$	$F_\beta(y, \hat{y})$
`"samples"`	$\frac{1}{\|S\|} \sum_{s \in S} P(y_s, \hat{y}_s)$	$\frac{1}{\|S\|} \sum_{s \in S} R(y_s, \hat{y}_s)$	$\frac{1}{\|S\|} \sum_{s \in S} F_\beta(y_s, \hat{y}_s)$
`"macro"`	$\frac{1}{\|L\|} \sum_{l \in L} P(y_l, \hat{y}_l)$	$\frac{1}{\|L\|} \sum_{l \in L} R(y_l, \hat{y}_l)$	$\frac{1}{\|L\|} \sum_{l \in L} F_\beta(y_l, \hat{y}_l)$
`"weighted"`	$\frac{1}{\sum_{l \in L} \|\hat{y}_l\|} \sum_{l \in L} \|\hat{y}_l\| P(y_l, \hat{y}_l)$	$\frac{1}{\sum_{l \in L} \|\hat{y}_l\|} \sum_{l \in L} \|\hat{y}_l\| R(y_l, \hat{y}_l)$	$\frac{1}{\sum_{l \in L} \|\hat{y}_l\|} \sum_{l \in L} \|\hat{y}_l\| F_\beta(y_l, \hat{y}_l)$
`None`	$\langle P(y_l, \hat{y}_l) \| l \in L \rangle$	$\langle R(y_l, \hat{y}_l) \| l \in L \rangle$	$\langle F_\beta(y_l, \hat{y}_l) \| l \in L \rangle$

Examples

```
>>> from sklearn.metrics import recall_score
>>> y_true = [0, 1, 2, 0, 1, 2]
>>> y_pred = [0, 2, 1, 0, 0, 1]
>>> recall_score(y_true, y_pred, average='macro')
0.33...
>>> recall_score(y_true, y_pred, average='micro')
0.33...
>>> recall_score(y_true, y_pred, average='weighted')
0.33...
```

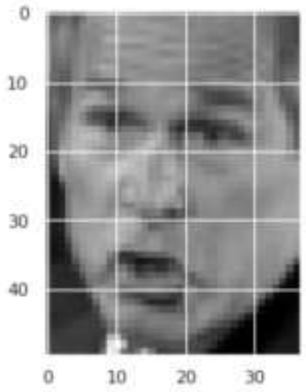
使用SVC实现人脸识别

使用SVC实现人脸识别

In [27]:
```python
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.decomposition import PCA
```

In [28]:
```python
# 1、加载数据集
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
```

In [29]:
```python
plt.imshow(lfw_people.images[6], cmap='gray')
plt.show()
```



In [30]:
```
n_samples, h ,w = lfw_people.images.shape
print(n_samples)
print(h)
print(w)
```

```
1288
50
37
```

In [31]:
```
lfw_people.data.shape
```

Out[31]:
```
(1288, 1850)
```

In [32]:
```
lfw_people.target
```

Out[32]:
```
array([5, 6, 3, ..., 5, 3, 5])
```

In [33]:
```
target_names = lfw_people.target_names
target_names
```

Out[33]:
```
array(['Ariel Sharon', 'Colin Powell', 'Donald Rumsfeld', 'George W Bush',
       'Gerhard Schroeder', 'Hugo Chavez', 'Tony Blair'], dtype='<U17')
```

In [34]:
```
n_classes = lfw_people.target_names.shape[0]
print(n_classes)
```

```
7
```

In [36]:
```python
# 2、拆分数据集
x_train, x_test, y_train, y_test = train_test_split(lfw_people.data, lfw_people.target, random_state=10)
```

In [37]:
```python
# 3、构建模型
model = SVC(kernel='rbf', class_weight='balanced')

# 4、训练集上训练模型
model.fit(x_train, y_train)
```

Out[37]:
```
SVC(C=1.0, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [38]:
```python
# 5、测试集上预测结果
predictions = model.predict(x_test)

# 6、模型评测
print(classification_report(y_test, predictions, target_names=lfw_people.target_names))
```

	precision	recall	f1-score	support
Ariel Sharon	0.00	0.00	0.00	23
Colin Powell	0.18	1.00	0.31	59
Donald Rumsfeld	0.00	0.00	0.00	28
George W Bush	0.00	0.00	0.00	138
Gerhard Schroeder	0.00	0.00	0.00	21
Hugo Chavez	0.00	0.00	0.00	14
Tony Blair	0.00	0.00	0.00	39
avg / total	0.03	0.18	0.06	322

```
In [39]: # 优化
         # PCA降维
         n_components = 100
         pca = PCA(n_components=n_components, whiten=True).fit(lfw_people.data)
         x_train_pca = pca.transform(x_train)
         x_test_pca = pca.transform(x_test)

In [40]: x_train_pca.shape

Out[40]: (966, 100)

In [41]: model = SVC(kernel='rbf', class_weight='balanced')
         model.fit(x_train_pca, y_train)

Out[41]: SVC(C=1.0, cache_size=200, class_weight='balanced', coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [42]: predictions = model.predict(x_test_pca)
         print(classification_report(y_test, predictions, target_names=lfw_people.target_names))

                   precision    recall  f1-score   support

     Ariel Sharon       1.00      0.57      0.72        23
     Colin Powell       0.69      0.95      0.80        59
  Donald Rumsfeld       0.88      0.79      0.83        28
    George W Bush       0.90      0.93      0.91       138
Gerhard Schroeder       0.72      0.86      0.78        21
      Hugo Chavez       1.00      0.79      0.88        14
       Tony Blair       1.00      0.62      0.76        39

      avg / total       0.87      0.84      0.84       322
```

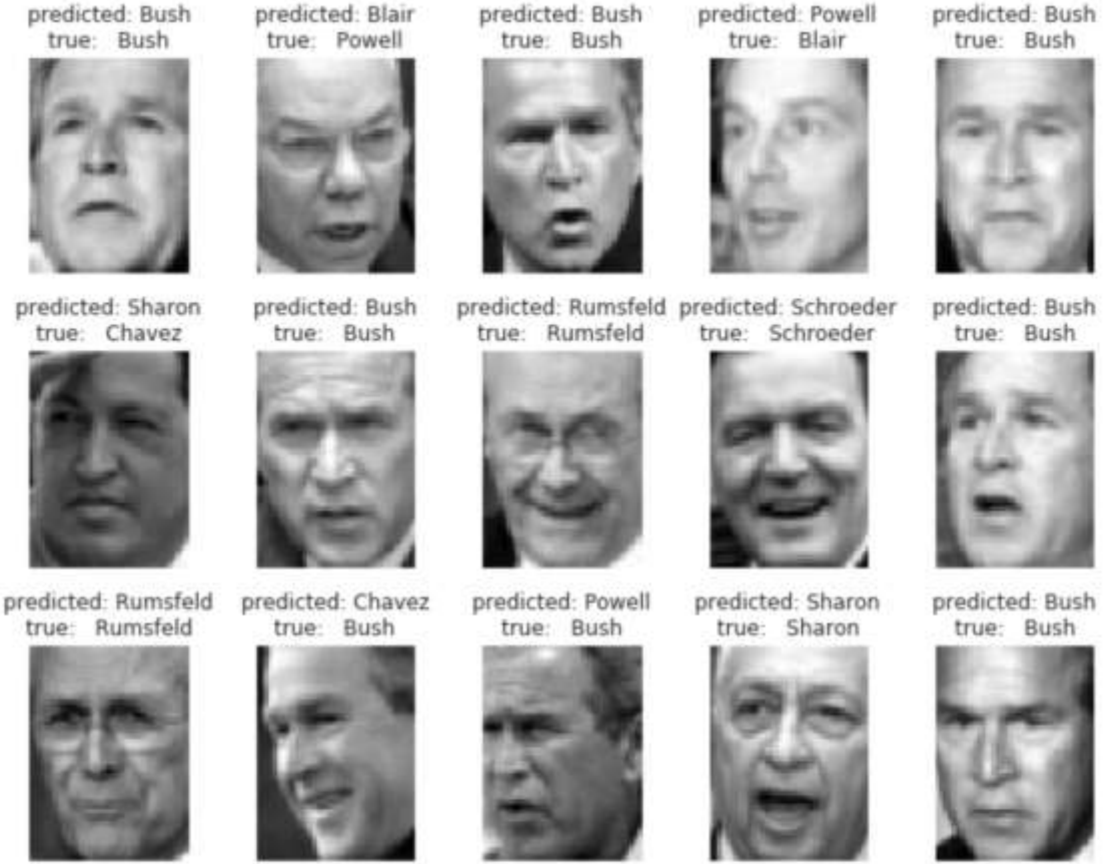
In [43]:
```python
# 调参优化
param_grid = {
    'C' : [0.1, 1, 5, 10, 100],
    'gamma' : [0.0005, 0.001, 0.005, 0.01],
}
model = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid )
model.fit(x_train_pca, y_train)
print(model.best_estimator_)
```

```
SVC(C=5, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.005, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [44]:
```python
predictions = model.predict(x_test_pca)
print(classification_report(y_test, predictions, target_names=lfw_people.target_names))
```

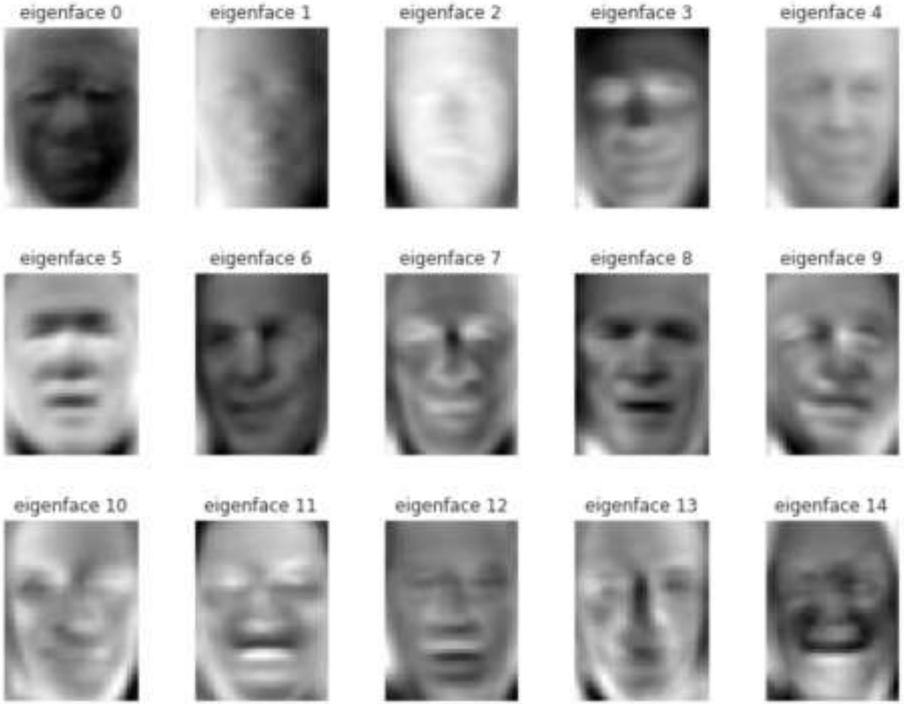
	precision	recall	f1-score	support
Ariel Sharon	0.88	0.61	0.72	23
Colin Powell	0.80	0.88	0.84	59
Donald Rumsfeld	0.76	0.79	0.77	28
George W Bush	0.86	0.93	0.89	138
Gerhard Schroeder	0.71	0.81	0.76	21
Hugo Chavez	0.85	0.79	0.81	14
Tony Blair	0.92	0.62	0.74	39
avg / total	0.84	0.83	0.83	322

```
In [48]: # 可视化
def plot_gallery(images, titles, h, w, n_row=3, n_col=5):
    plt.figure(figsize=(1.8*n_col, 2.4*n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row*n_col):
        plt.subplot(n_row, n_col, i+1)
        plt.imshow(images[i].reshape(h, w), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())

# 获取一张图片
def title(predictions, y_test, target_names, i):
    pred_name = target_names[predictions[i]].split(' ')[-1]
    true_name = target_names[y_test[i]].split(' ')[-1]
    return 'predicted: %s\ntrue:    %s' %(pred_name, true_name)

# 获取所有图片title
prediction_titles = [title(predictions, y_test, target_names, i) for i in range(len(predictions))]

# 画图
plot_gallery(x_test, prediction_titles, h, w)

plt.show()
```


predicted: Bush
true: Bush
predicted: Blair
true: Powell
predicted: Bush
true: Bush
predicted: Powell
true: Blair
predicted: Bush
true: Bush
predicted: Sharon
true: Chavez
predicted: Bush
true: Bush
predicted: Rumsfeld
true: Rumsfeld
predicted: Schroeder
true: Schroeder
predicted: Bush
true: Bush
predicted: Rumsfeld
true: Rumsfeld
predicted: Chavez
true: Bush
predicted: Powell
true: Bush
predicted: Sharon
true: Sharon
predicted: Bush
true: Bush

In [53]:

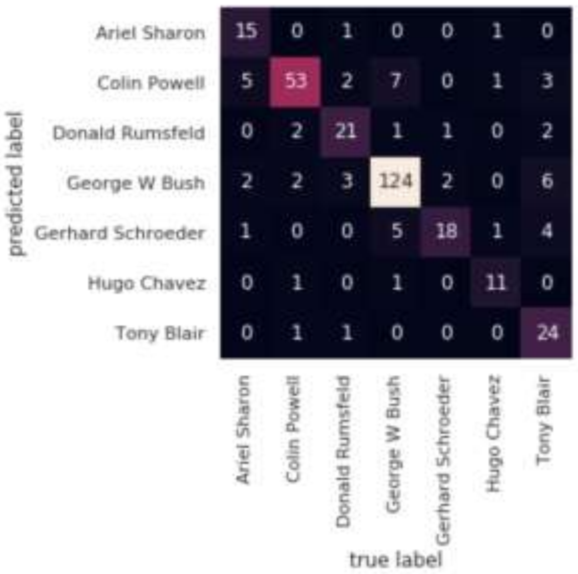
```
eigenfaces = pca.components_.reshape((n_components,h,w))
eigenface_titles = ['eigenface %d' % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces,eigenface_titles,h,w)

plt.show()
```



In [50]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns;sns.set()
mat = confusion_matrix(y_test, predictions)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=lfw_people.target_names,
            yticklabels=lfw_people.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[50]: Text(89.18,0.5,'predicted label')



自定义核式回归

- 常用核函数：

名称	表达式	参数
线性核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$	
多项式核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^\top \boldsymbol{x}_j)^d$	$d \geq 1$为多项式的次数
高斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\\|\boldsymbol{x}_i - \boldsymbol{x}_j\\|^2}{2\delta^2}\right)$	$\delta > 0$为高斯核的带宽(width)
拉普拉斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\\|\boldsymbol{x}_i - \boldsymbol{x}_j\\|}{\delta}\right)$	$\delta > 0$
Sigmoid核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta \boldsymbol{x}_i^\top \boldsymbol{x}_j + \theta)$	tanh为双曲正切函数, $\beta > 0$, $\theta < 0$

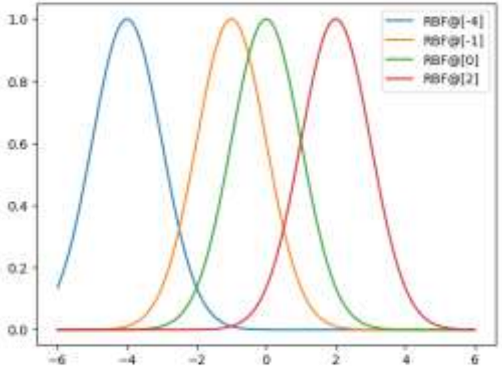
此外通过核函数的线性组合、内积运算也能得到新的核函数

特别的：若 k 为核函数，对于任意函数 $g(\boldsymbol{x})$, $\tilde{k}(\boldsymbol{x},\boldsymbol{z}) = g(\boldsymbol{x})k(\boldsymbol{x},\boldsymbol{z})g(\boldsymbol{z})$ 也是核函数.

```python
def RBF_kernel(X1,X2,sigma):
    """
    计算两组向量之间的 RBF 核
    参数:
        X1 - 一个 n1xd 矩阵，其中每行包含一个向量 x1_1,...,x1_n1
        X2 - 一个 n2xd 矩阵，其中每行包含一个向量 x2_1,...,x2_n2
        sigma - RBF/高斯核的带宽（即标准差）
    返回:
        大小为 n1xn2 的矩阵，位置 i,j 处的值为 exp(-||x1_i-x2_j||^2/(2 sigma^2))
    """
    return 0 #TODO
```

```
# RBF kernel
y = RBF_kernel(prototypes, xpts, 1)        "xpts": Unknown word.
for i in range(len(prototypes)):
    label = "RBF@"+str(prototypes[i,:])
    plt.plot(xpts, y[i,:], label=label)     "xpts": Unknown word.
plt.legend(loc = 'best')
plt.show()
```

✓ 0.0s



```python
class Kernel_Machine(object):
    def __init__(self, kernel, prototype_points, weights):
        """
        参数:
            kernel(X1, X2) - 一个函数，返回 X1 和 X2 的行之间的交叉核矩阵
            prototype_points - 一个 Rxd 矩阵，其中每行是 mu_1,...,mu_R
            weights - 一个长度为 R 的向量，其中的条目是 w_1,...,w_R
        """
        self.kernel = kernel
        self.prototype_points = prototype_points
        self.weights = weights

    def predict(self, X):
        """
        在由 X 的各行给定的点上评估核机器
        参数:
            X - 一个 nxd 矩阵，其中每行是输入 x_1,...,x_n
        返回:
            核机器在 X 中 n 个点上的评估值向量。具体来说，返回向量的第 j 个条目是
                Sum_{i=1}^R w_i k(x_j, mu_i)
        """
        return self.kernel(X, self.prototype_points) @ self.weights
```
✓ 0.0s

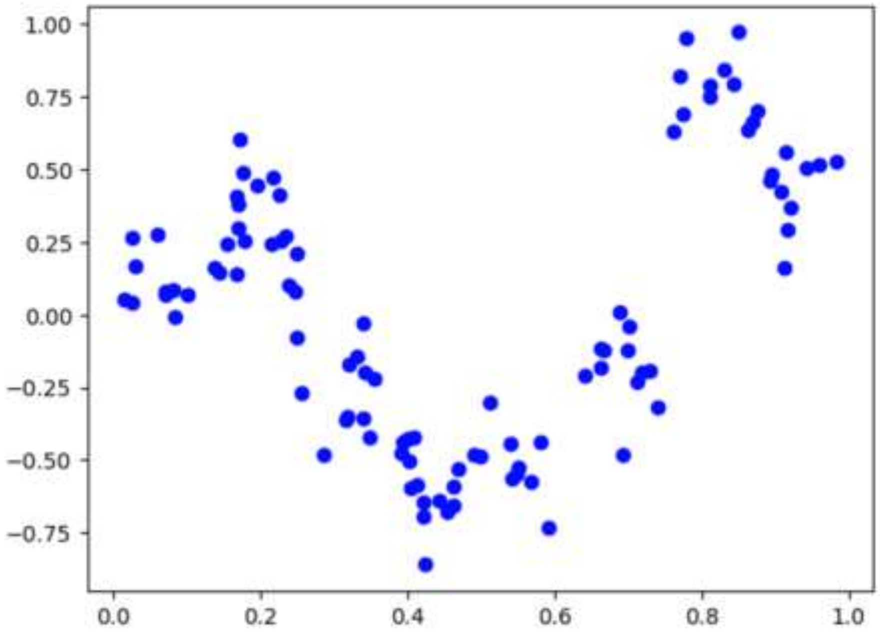
加载训练和测试数据；转换为列向量，以便能够更好地泛化到高维数据。

```
data_train,data_test = np.loadtxt("krr-train.txt"),np.loadtxt("krr-test.txt")
x_train, y_train = data_train[:,0].reshape(-1,1),data_train[:,1].reshape(-1,1)
x_test, y_test = data_test[:,0].reshape(-1,1),data_test[:,1].reshape(-1,1)
```

✓ 0.0s

```python
# plot training data
plt.plot(x_train, y_train, 'bo')
plt.show()
```

✓ 0.0s



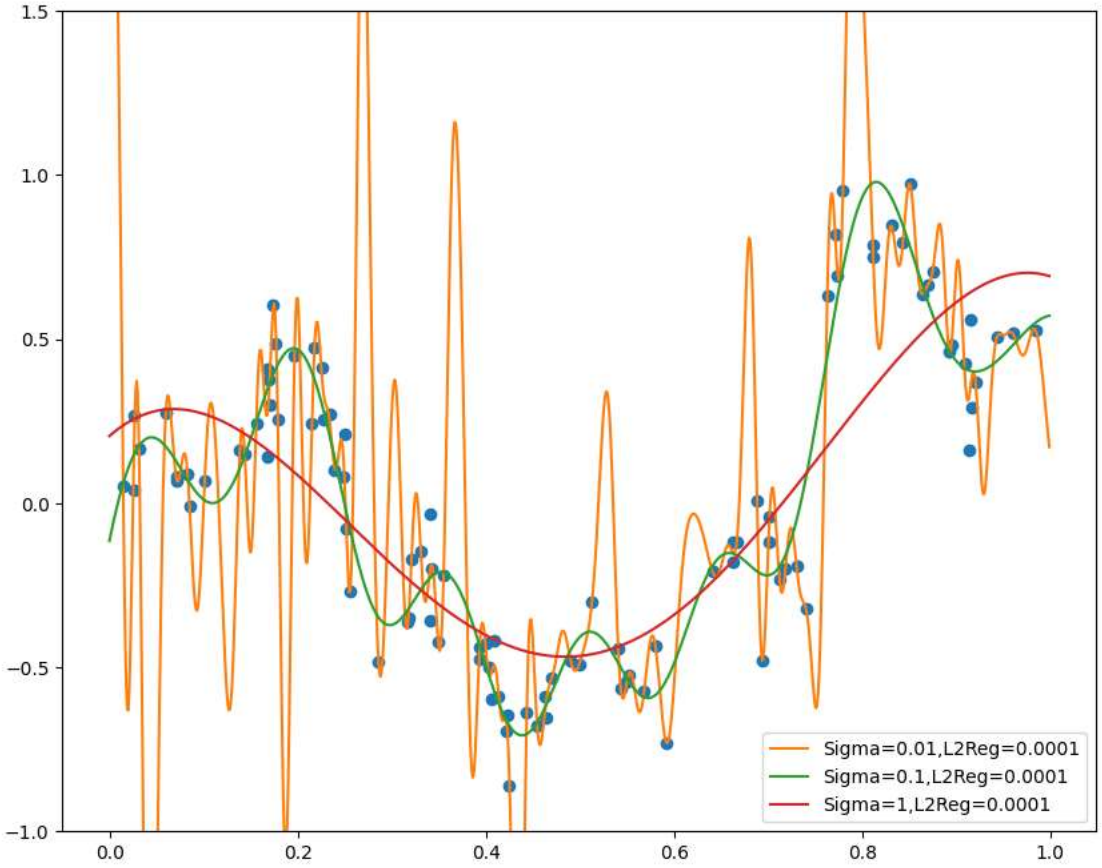
```python
def train_kernel_ridge_regression(X, y, kernel, l2reg):
    """
    训练一个核岭回归模型
    参数:
        X - 一个 nxd 矩阵，其中每行是一个训练样本 x_1,...,x_n
        y - 一个长度为 n 的向量，其中每个条目是对应样本的目标值 y_1,...,y_n
        kernel - 一个函数，返回 X 的行之间的核矩阵
        l2reg - L2 正则化参数（岭回归中的 λ)
    返回:
        一个训练好的 Kernel_Machine 对象
    提示:
        1.计算核矩阵
        2.计算alpha权重
        3.返回一个 Kernel_Machine 对象
    """
    #TODO
    return 0 #TODO
```

✓ 0.0s

$$\min_\alpha \|y - K\alpha\|^2 + \lambda\alpha^T K\alpha$$

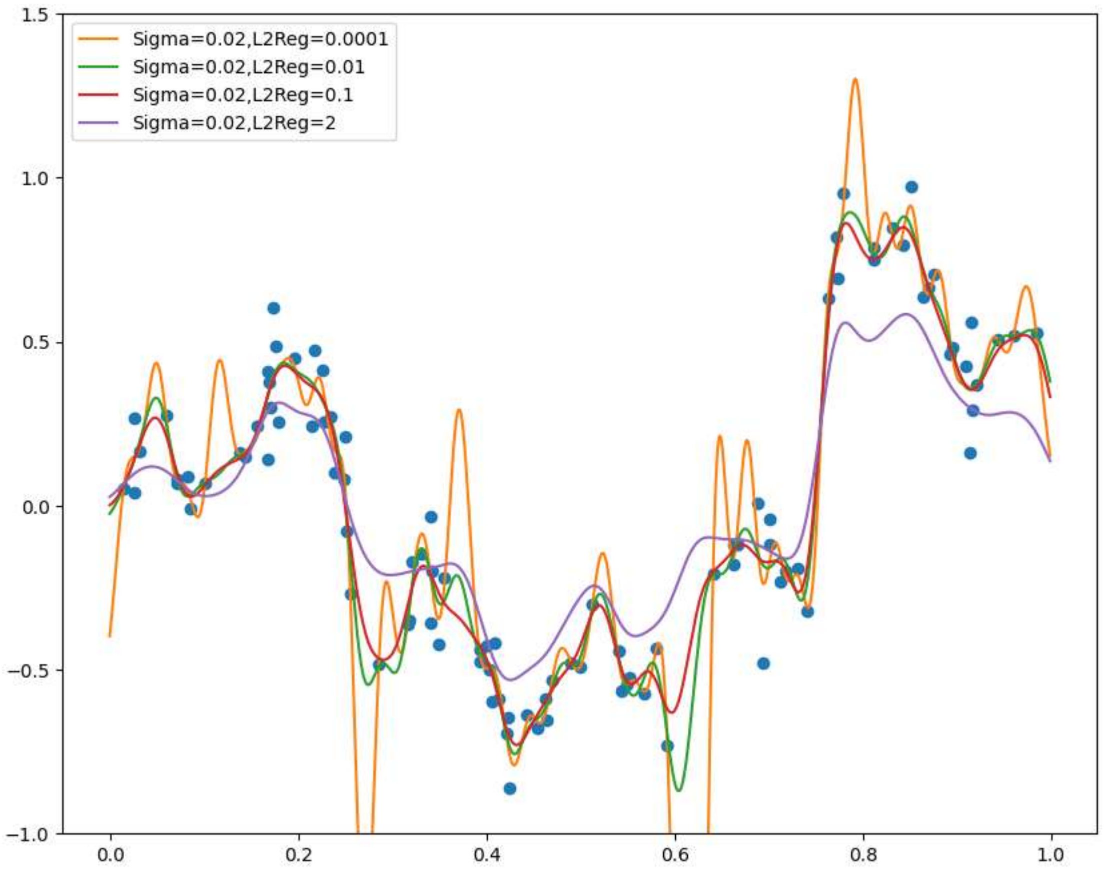
$$\alpha = (K + \lambda I)^{-1} y$$

```
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)    "xpts": Unknown word.
plt.figure(figsize=(10, 8))    "figsize": Unknown word.
plt.plot(x_train,y_train,'o')
l2reg = 0.0001
for sigma in [.01,.1,1]:
    k = functools.partial(RBF_kernel, sigma=sigma)
    f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
    label = "Sigma="+str(sigma)+",L2Reg="+str(l2reg)
    plt.plot(xpts, f.predict(xpts), label=label)    "xpts": Unknown word.
plt.legend(loc = 'best')
plt.ylim(-1,1.5)    "ylim": Unknown word.
plt.show()
```


1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
Sigma=0.01,L2Reg=0.0001
Sigma=0.1,L2Reg=0.0001
Sigma=1,L2Reg=0.0001

```python
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)    "xpts": Unknown word.
plt.figure(figsize=(10, 8))    "figsize": Unknown word.
plt.plot(x_train, y_train,'o')
sigma= .02
for l2reg in [.0001,.01,.1,2]:
    k = functools.partial(RBF_kernel, sigma=sigma)
    f = train_kernel_ridge_regression(x_train, y_train, k, l2reg=l2reg)
    label = "Sigma="+str(sigma)+",L2Reg="+str(l2reg)
    plt.plot(xpts, f.predict(xpts), label=label)    "xpts": Unknown word.
plt.legend(loc = 'best')
plt.ylim(-1,1.5)    "ylim": Unknown word.
plt.show()
```

✓ 0.2s


Sigma=0.02,L2Reg=0.0001
Sigma=0.02,L2Reg=0.01
Sigma=0.02,L2Reg=0.1
Sigma=0.02,L2Reg=2
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

加分项

```python
### Kernel function generators
def linear_kernel(X1, X2):
    """
    计算两组向量之间的线性核。
    参数:
        X1 - 一个 n1xd 矩阵, 其中每行包含一个向量 x1_1,...,x1_n1
        X2 - 一个 n2xd 矩阵, 其中每行包含一个向量 x2_1,...,x2_n2
    返回:
        大小为 n1xn2 的矩阵, 位置 i,j 处的值为 x1_i^T x2_j
    """
    return 0 #加分项TODO
```

```python
def polynomial_kernel(X1, X2, offset, degree):
    """
    计算两组向量之间的不齐次多项式核
    参数:
        X1 - 一个 n1xd 矩阵, 其中每行包含一个向量 x1_1,...,x1_n1
        X2 - 一个 n2xd 矩阵, 其中每行包含一个向量 x2_1,...,x2_n2
        offset, degree - 核的两个参数
    返回:
        大小为 n1xn2 的矩阵, 位置 i,j 处的值为 (offset + <x1_i,x2_j>)^degree
    """
    return 0 #加分项TODO
```

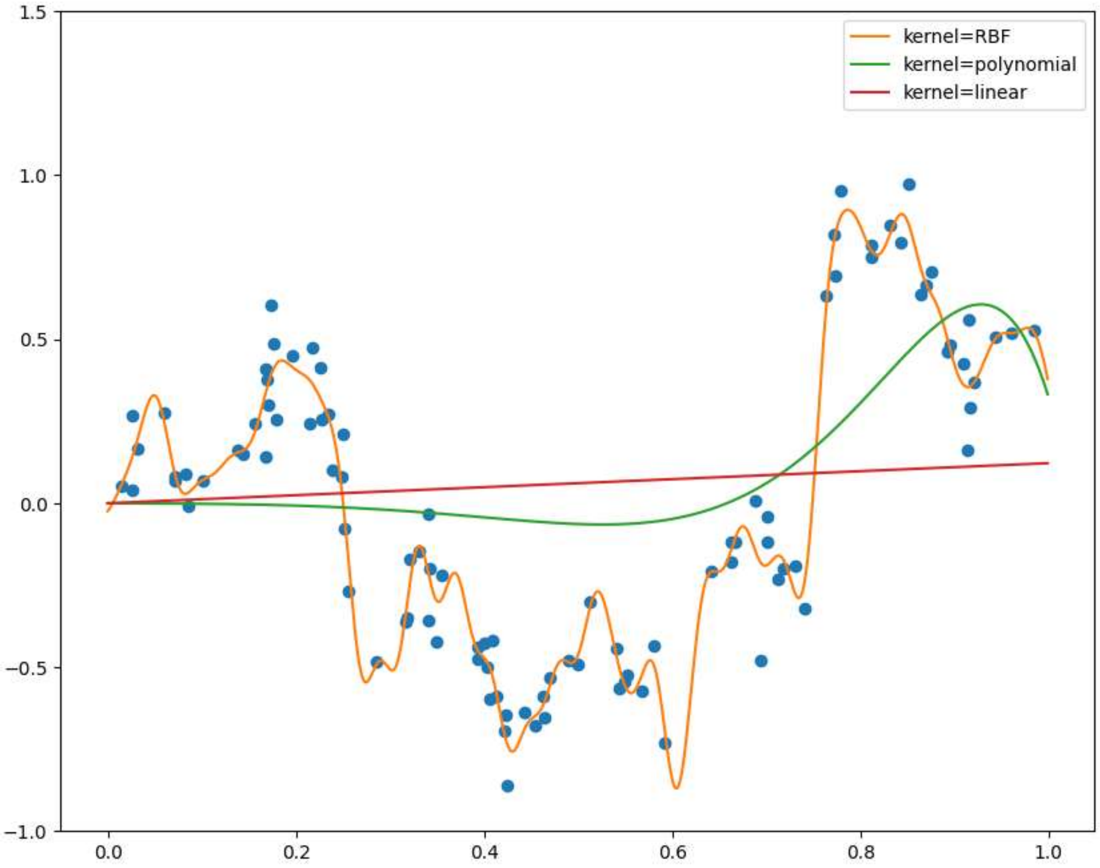
线性核 $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$

多项式核 $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^\top \boldsymbol{x}_j)^d$

加分项

```python
plot_step = .001
xpts = np.arange(0 , 1, plot_step).reshape(-1,1)    "xpts": Unknown word.
plt.figure(figsize=(10, 8))    "figsize": Unknown word.
plt.plot(x_train, y_train,'o')
sigma= .02
l2reg= .01
"""
提示: for函数分不同核函数对训练结果进行可视化查看效果,
"""
#for :
    #TODO
    #label = "kernel="+str(kernel_)
    #plt.plot(xpts, f.predict(xpts), label=label)    "xpts": Unknown word.
plt.legend(loc = 'best')
plt.ylim(-1,1.5)    "ylim": Unknown word.
plt.show()
```

✓ 0.2s


kernel=RBF
kernel=polynomial
kernel=linear
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0