Overview

In [1]:

```
%matplotlib inline
```

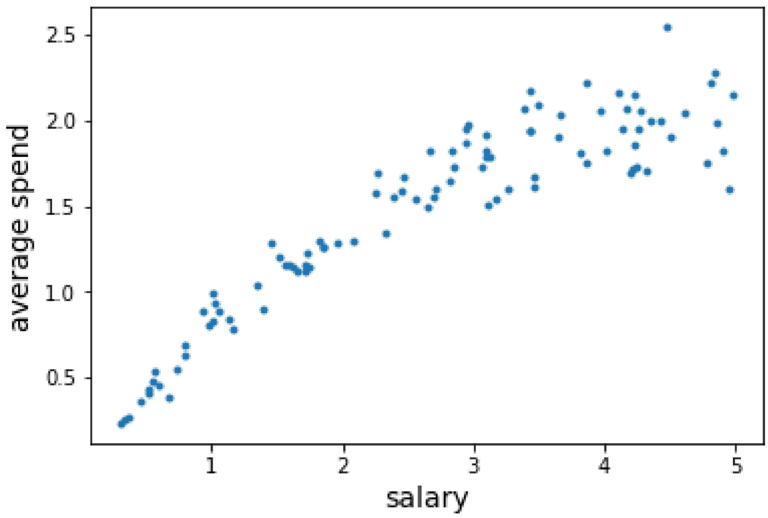
一个简单例子

经验数据 :100个人的工资水平和他平均每月的信用卡消费

In [2]:

```
import numpy as np
from matplotlib import pyplot as plt
from utils import load_salary

# x: salary, y: spending
x, y = load_salary()

print("x_min: %.2f x_max: %.2f" % (np.min(x), np.max(x)))
print("y_min: %.2f y_max: %.2f" % (np.min(y), np.max(y)))

plt.plot(x,y,'.')
plt.xlabel("salary", fontsize=14)
plt.ylabel("average spend", fontsize=14)

plt.show()
```

```
x_min: 0.31 x_max: 4.98
y_min: 0.23 y_max: 2.54
```

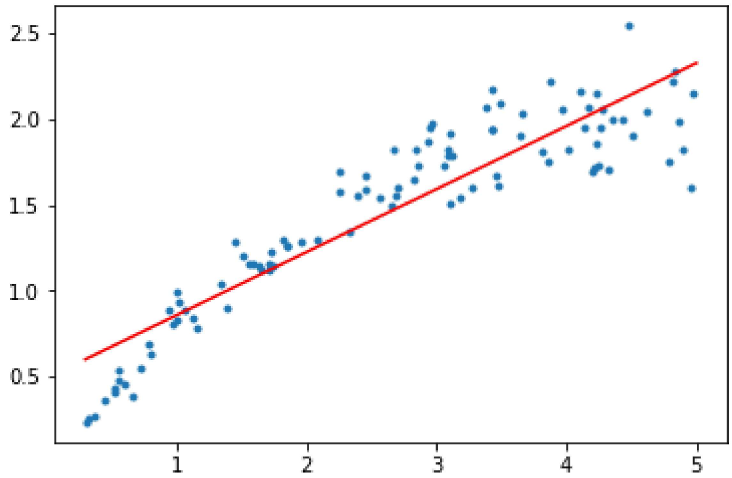


任务：已知新客户的工资水平是2.3万，预测该客户平均每月的信用卡消费多少

In [3]:

```
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
estimator = LinearRegression()

estimator.fit(x, y)
pred = estimator.predict([[2.3]])
print("predicted spending: %s" % pred[0] )

score = estimator.score(x,y)
print("training score: %s" % score)

x1=np.linspace(0.3,5,20).reshape((-1, 1))
y1=estimator.predict(x1)
plt.plot(x,y,'.')
plt.plot(x1, y1, 'r-')
plt.show()
```

```
predicted spending: 1.33427619762
training score: 0.836313262415
```



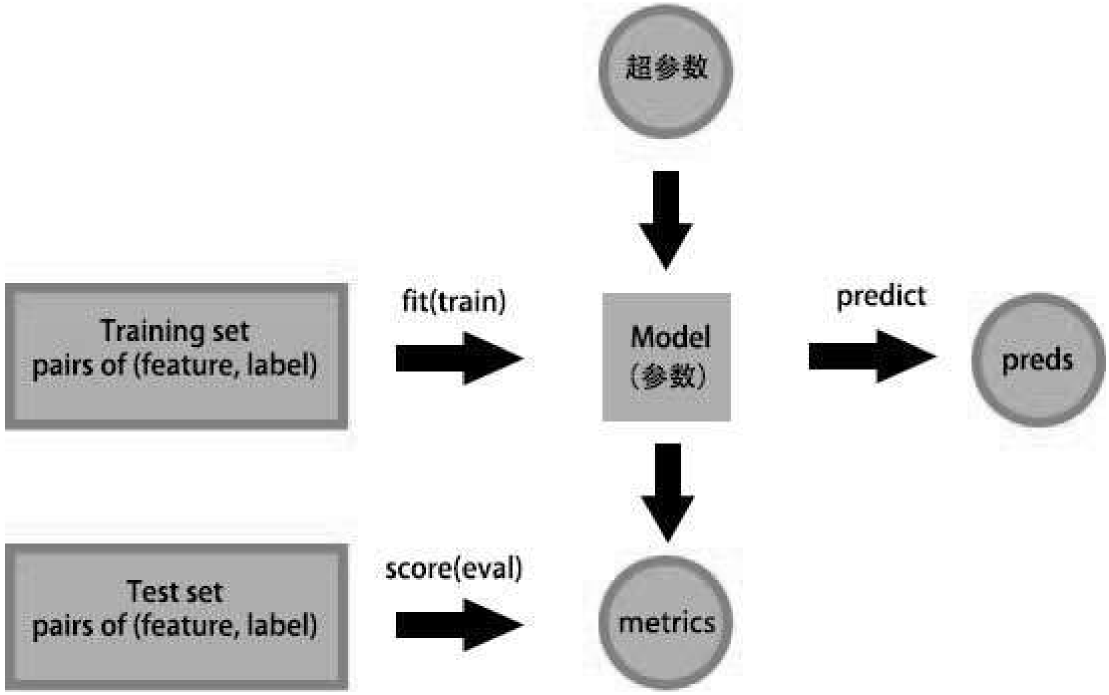
几种类型

根据“数据”和“任务”之间的相互关系，“机器学习”问题大致可分为如下三种类型：

- **supervised learning (监督式学习)**: labeled data (match the "tasks")
  - classification (分类)
    - binary classification
  - regression (回归)
  - *conditional generative models: such as translation, cGAN etc.
- **semi-supervised learning (半监督式学习)**: labeled data (but without enough information for the tasks)
  - reinforcement learning (增强学习)， delayed reward,
    - Q-learning: only know dead or live status, learn move strategy
  - transfer learning (迁移学习)
  - adversarial learning (对抗学习): such as GAN etc.
- **unsupervised learning (无监督式学习)**: unlabeled data
  - clustering (聚类)
  - transformation
    - 数据的 standardization 可视为一种 unsupervised learning

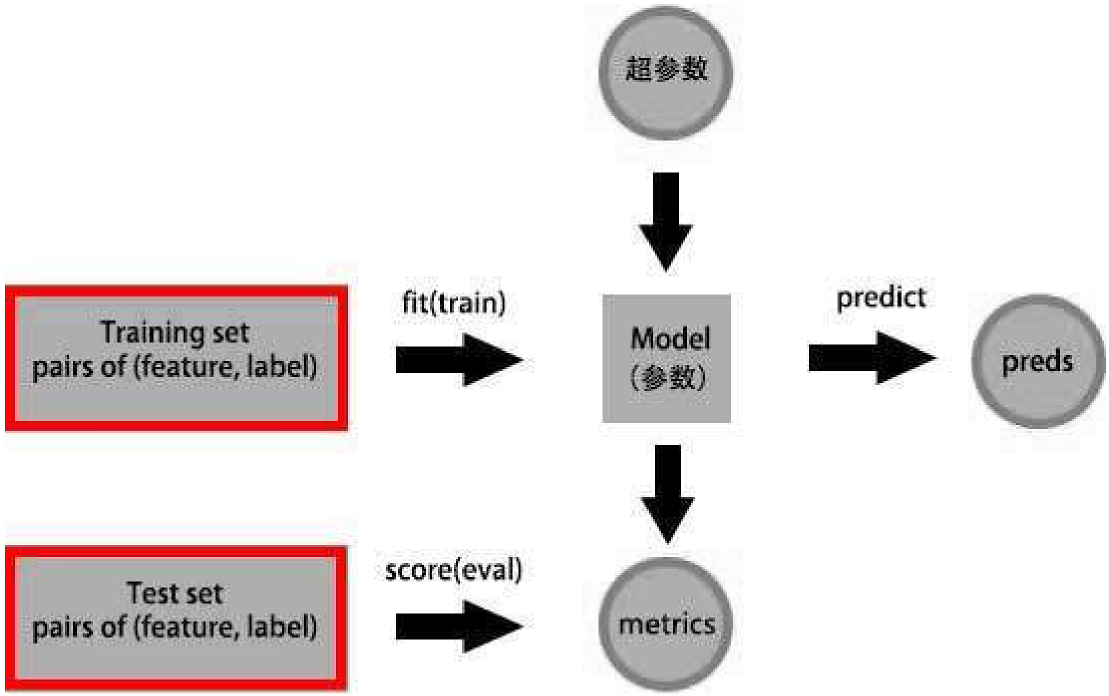
  - dimensionality reduction (降维) and data compression
    - PCA
    - autoencoder etc.
  - generative models, 如：GAN，

2. general procedures （操作步骤）



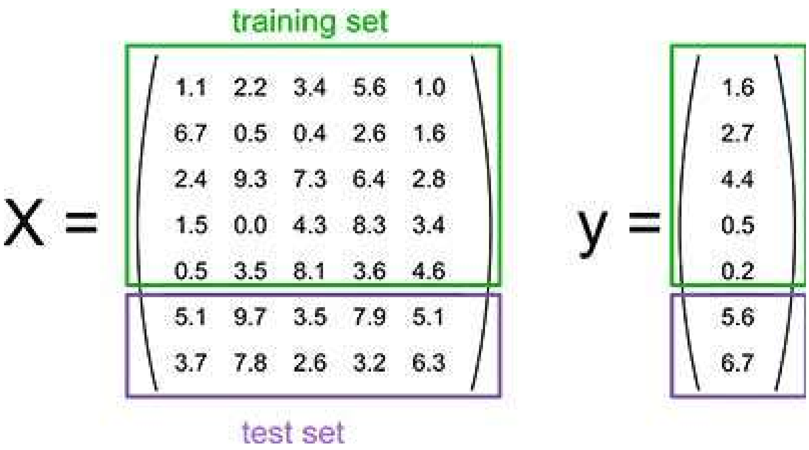
2.1 数据

训练集和测试集



- 训练集（training set）：用于调节“参数”
  - 验证集（validation set）：用于确保选取“超参数”的普适性

- 测试集（test set）： a simulation of "future data", 用于确保“泛化”



In [4]:

```
from utils import load_salary
from sklearn.model_selection import train_test_split

x, y = load_salary()

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8, random_state=1)

print(x_train.shape)
print(y_test.shape)
```

In [5]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from utils import load_blobs, plot_blobs

x, y = load_blobs()

print(np.bincount(y))

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8)
print(np.bincount(y_train))

# stratify 表示按哪个变量成比例划分
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8, stratify=y)
print(np.bincount(y_train))

plot_blobs(x, y)
```

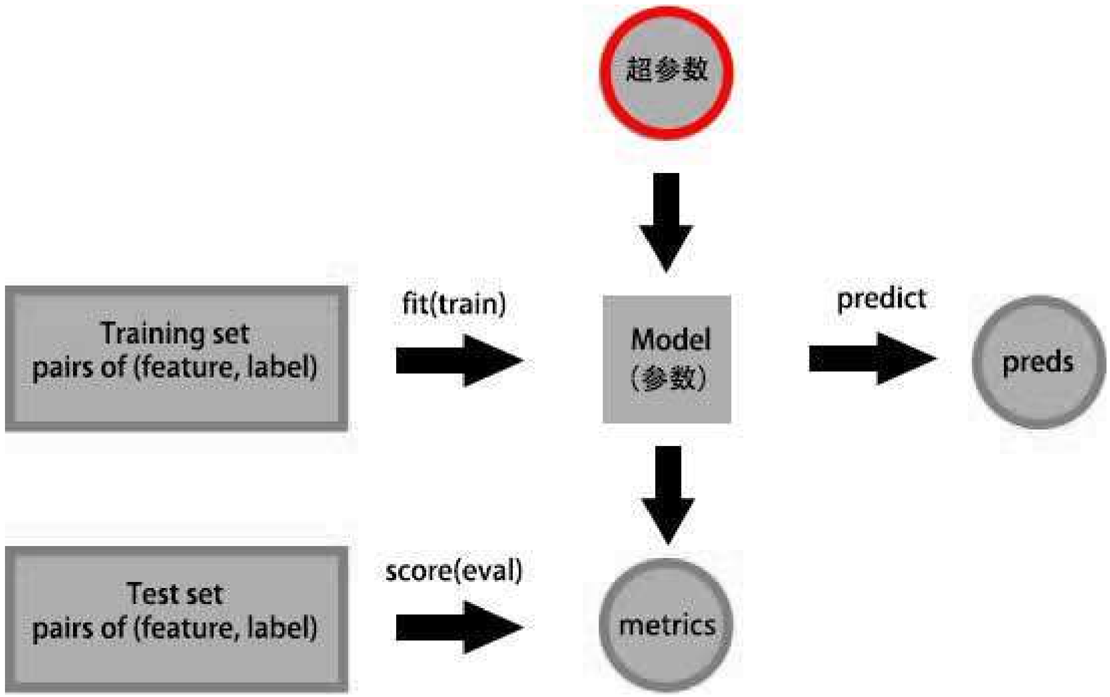
```
blob seed: 125
[50 50]
[37 43]
[40 40]
```

tips:

- training set 通常选取为总数据的80%左右
- 对于小样本数据，可以通过cross validation的方式减少validation导致的数据消耗
- 对于小样本数据，training set 内各类别成分的比例，最好同总数据的成分比例一致 （stratify split）

[特征工程 (./feature_engineering.ipynb)](./feature_engineering.ipynb)

2.2 超参数的选取

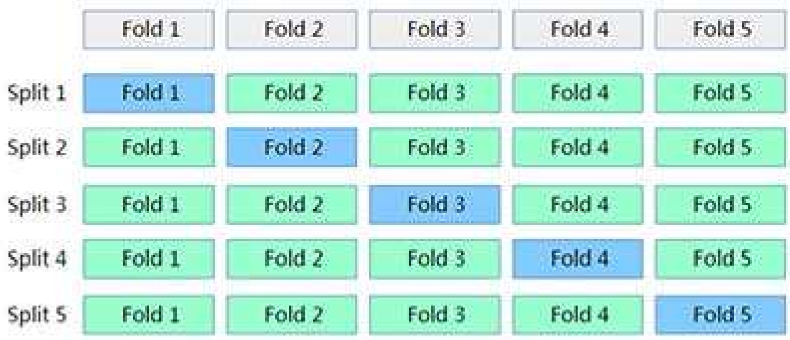


“参数”与“超参数”（hyperparameter）

“超参数”: 学习过程中选定的固定变量，fixed, 通常描述模型的某种特定“结构”约束

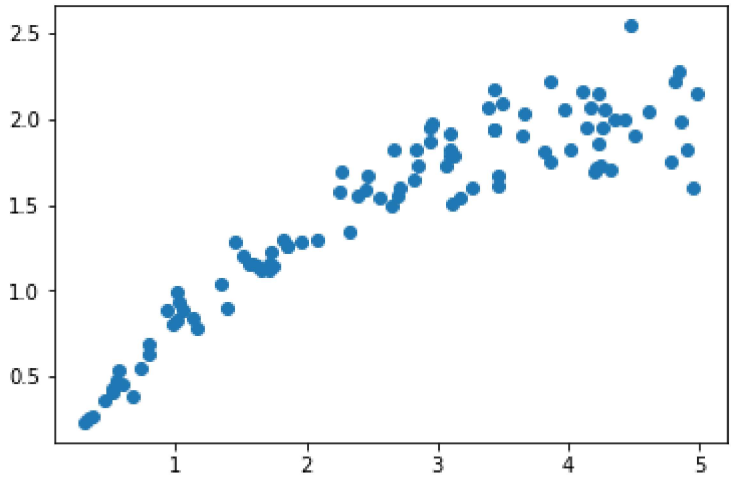
“参数”：学习过程中“自动”调节的模型变量, trainable, 通常描述某种特定“结构”的具体参数

cross validation



In [5]:

```
%matplotlib inline
from utils import load_salary
X, y = load_salary()
plt.plot(X, y, 'o')
plt.show()
```

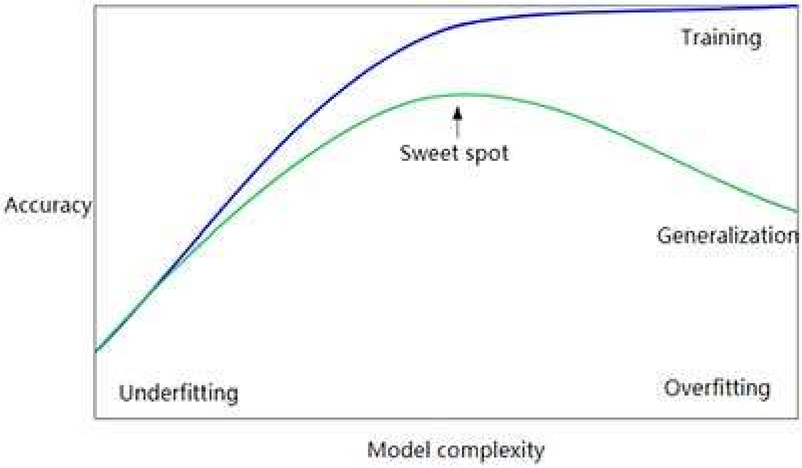


In [6]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score


def print_cv_score(degree=1):
    estimator = LinearRegression()
    X_extend = PolynomialFeatures(degree=degree).fit_transform(X)
    scores = cross_val_score(estimator, X_extend, y, cv=5)
    # print(scores)
    print("degree: %d, score: %.6f" % (degree, np.mean(scores)))

#degree=2 sweet spot
#degree=1 under-fitting
#degree>2 over-fitting
for d in [1,2,10,30]:
    print_cv_score(d) # underfitting
```

```
degree: 1, score: 0.801617
degree: 2, score: 0.903789
degree: 10, score: 0.879646
degree: 30, score: 0.671668
```

under-fitting vs over-fitting (bias-variance tradeoff)



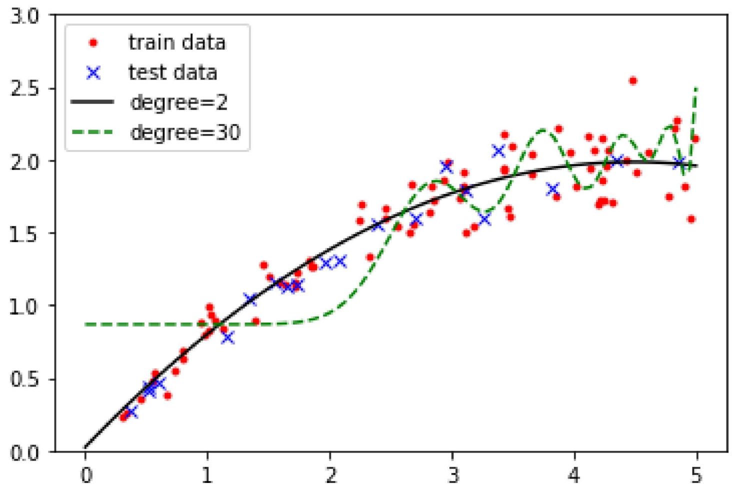
两个不同视角来理解underfitting 和 overfitting

- 从固定training dataset(因而model固定)，而选取不同test dataset 来看。underfitting 表示模型的推广性较好（test error和training error差别不大），但模型本身的拟合效果较差（即training error较大）；overfitting 表示模型本身的拟合能力强（training error较小），但generalization 较差（由于过分拟合training dataset的噪音，而导致 test error显著高于training error）

In [73]:

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=1)

ax = plt.gca()
ax.plot(X_train, y_train, 'r.', label="train data")
ax.plot(X_test, y_test, 'bx', label="test data")

for i, degree in enumerate([2, 30]):
    estimator = LinearRegression()
    transformer = PolynomialFeatures(degree=degree)

    X_train_extend = transformer.fit_transform(X_train)
    X_test_extend = transformer.fit_transform(X_test)

    estimator.fit(X_train_extend, y_train)

    scores = estimator.score(X_test_extend, y_test)
    print("degree: %d, score: %.6f" % (degree, np.mean(scores)))

    xx = np.linspace(0, 5, 100).reshape((-1, 1))
    xx_extend = transformer.fit_transform(xx)
    yy = estimator.predict(xx_extend)

    ax.plot(xx, yy, ['k-', 'g--'][i], label="degree=%d" % degree)
    ax.set_ylim([0,3])

    plt.legend()
plt.show()
```

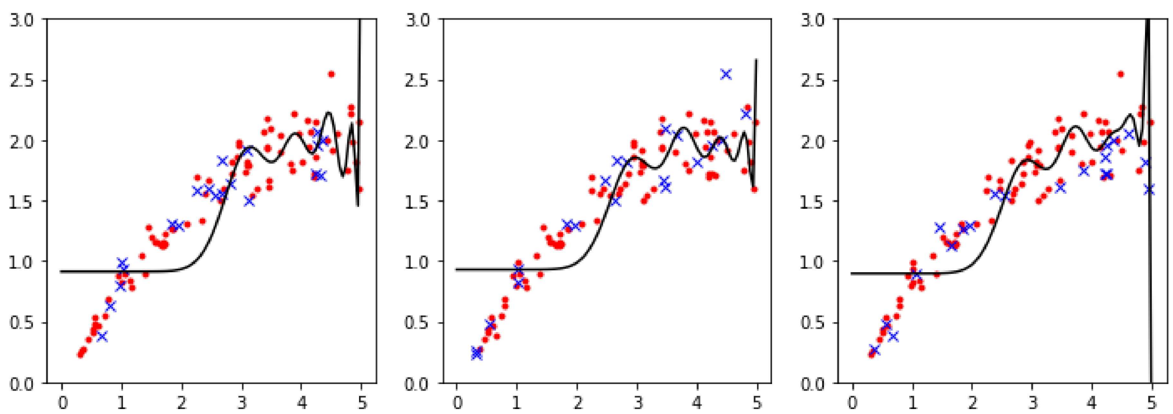
```
degree: 2, score: 0.965121
degree: 30, score: 0.718878
```



- 从不选取test dataset，但也不固定training dataset（因而model也不固定）的角度看。underfitting 表示model较稳定（不随training dataset的变化而显著变化），但模型本身的拟合能力不够用（training error较大）。overfitting 表示模型本身的拟合能力强（training error较小），但模型本身不稳定（随着training dataset的变化而显著变化)

In [84]:

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

degree = 30
seeds = [2, 10, 20]
estimator = LinearRegression()
transformer = PolynomialFeatures(degree=degree)

fig, axes =plt.subplots(1, 3, figsize=(12, 4))
for i, seed in enumerate(seeds):
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=seed)

    axes[i].plot(X_train, y_train, 'r.', label="train data")
    axes[i].plot(X_test, y_test, 'bx', label="test data")

    X_train_extend = transformer.fit_transform(X_train)
    X_test_extend = transformer.fit_transform(X_test)

    estimator.fit(X_train_extend, y_train)

    print(estimator.coef_)

    xx = np.linspace(0, 5, 100).reshape((-1, 1))
    xx_extend = transformer.fit_transform(xx)
    yy = estimator.predict(xx_extend)

    axes[i].plot(xx, yy, 'k-')
    axes[i].set_ylim([0,3])

plt.show()
```



减少 underfitting 可以通过选择更合理的feature、更高阶的模型等方式来解决。

抑制 overfitting 则涉及一些特别的技巧。

视角1对应的方法： 着眼于降低模型generalization error 的角度

- regularizaion（正则化）：penalize against complexity (Occam's razor 奥卡姆剃刀)

> Ridge Regularization (L2 norm):
>
> $$R = \lambda \sum |w_i|^2$$
>
> LASSO Regularization (L1 norm, Least Absolute Shrinkage and Selection Operator):
>
> $$R = \lambda \sum |w_i|$$
>
> The LASSO produces sparse parameters; most of the coefficients will become zero.

（正则化，可视为某种先验概率，详见下节）

视角2对应的方法： 着眼于增加模型的稳定性，或减少预测结果对模型细节的依赖性

- ensemble methods: random forest etc.
- dropout, batch normalization, data enhancement

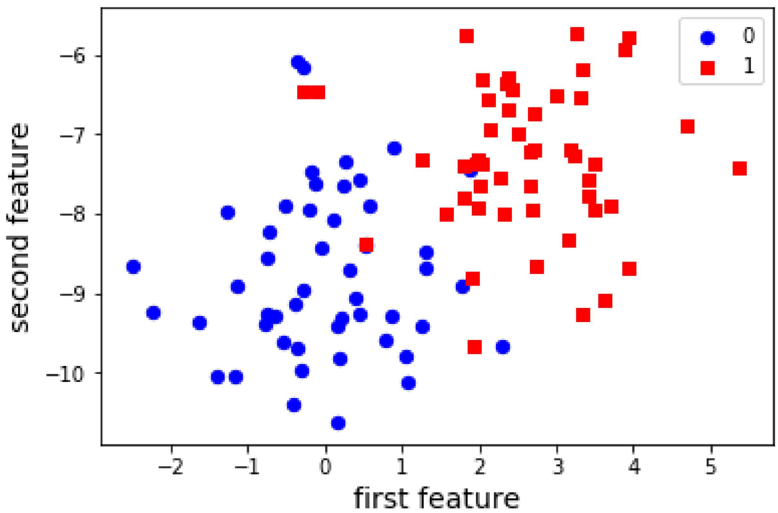
grid search

brute force search for hyperparameters （暴力搜索最佳的超参数组合）

In [7]:

```
from utils import load_blobs, plot_blobs
from matplotlib import pyplot as plt
x, y = load_blobs()
plot_blobs(x, y)
plt.show()
```

```
blob seed: 125
```



In [9]:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

clf = LogisticRegression() # logisitic classifier

for penalty in ["l2", "l1"]:
    for C in [0.02, 1, 50]:
        scores = cross_val_score(LogisticRegression(penalty=penalty, C=C), x, y, cv=5)
        print("penalty: %s, C: %f, average score: %f" % (penalty, C, np.mean(scores)))
```

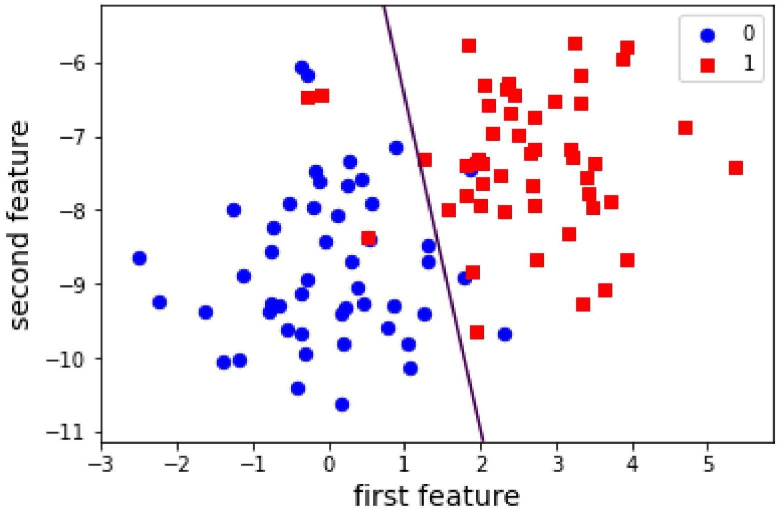
```
penalty: l2, C: 0.020000, average score: 0.930000
penalty: l2, C: 1.000000, average score: 0.940000
penalty: l2, C: 50.000000, average score: 0.920000
penalty: l1, C: 0.020000, average score: 0.730000
penalty: l1, C: 1.000000, average score: 0.930000
penalty: l1, C: 50.000000, average score: 0.920000
```

In [3]:

```
# make use of GridSearchCV
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.02, 1, 50], 'penalty': ["l2", "l1"]}

grid = GridSearchCV(LogisticRegression(), param_grid=param_grid, cv=5, verbose=3)

grid.fit(x, y) # an interface similar to estimator
print(grid.best_score_)
print(grid.best_params_)
```

In [11]:

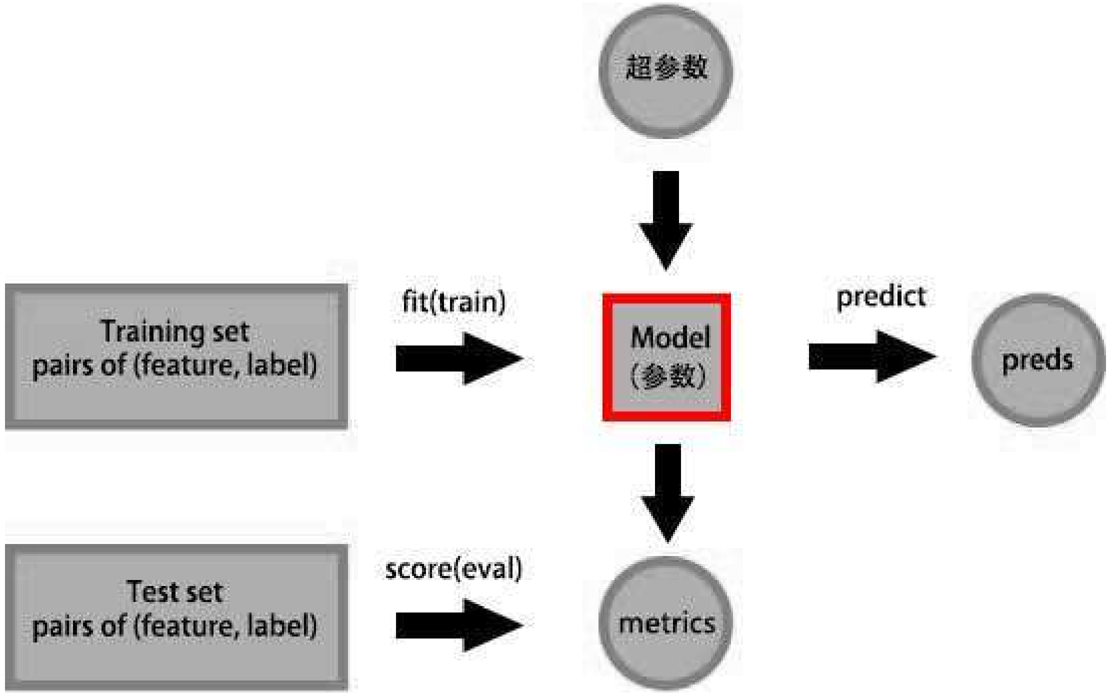
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from utils import plot_blobs, plot_2d_boundary

clf = LogisticRegression(penalty="l2", C=1) # logisitic classifier
clf.fit(x, y)

xmin, xmax = np.min(x[:, 0])-0.5, np.max(x[:, 0])+0.5
ymin, ymax = np.min(x[:, 1])-0.5, np.max(x[:, 1])+0.5
plot_blobs(x, y)
plot_2d_boundary(clf, xmin, xmax, ymin, ymax)
plt.show()

clf.score(x,y) #accuracy
# clf.predict(x) # classification result
# clf.predict_proba(x) # classification probs
# clf.decision_function(x) # signed distance
```



Out[11]:

0.93999999999999995

2.3 模型的建立



两种建模方式

“面向过程”式的模型： 如决策树， greedy search at local step，模型演化的每一步规则明确，而整体系统演化的target 不清晰

“面向对象”式的模型： 如svm，神经网络，optimize cost function globally, 模型演化的每一步可能存在随机性，而整体系统演化的 target 非常明确

[概率模型的数学基础 (Bayes.ipynb)](Bayes.ipynb)

几乎所有的深度学习模型，都是概率模型

model in scikit-learn: estimator interface

- estimator.fit, estimator.transform, estimator.predict, estimator.score
- 复合estimator： pipline(串联), GridSearchCV（并联）

In [17]:

```
# prepare data

import os

with open(os.path.join("../data", "SMSSpamCollection")) as f:
    lines = [line.strip().split("\t") for line in f.readlines()]

text = [p[1] for p in lines]
y = [p[0] == "ham" for p in lines]

from sklearn.model_selection import train_test_split
text_train, text_test, y_train, y_test = train_test_split(text, y)
```

In [22]:

```
# regression without pipline

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(text_train)

X_train = vectorizer.transform(text_train)
X_test = vectorizer.transform(text_test)

from sklearn.linear_model import LogisticRegression
regressor = LogisticRegression()
regressor.fit(X_train, y_train)
regressor.score(X_test, y_test)
```

Out[22]:

0.97130559540889527

In [23]:

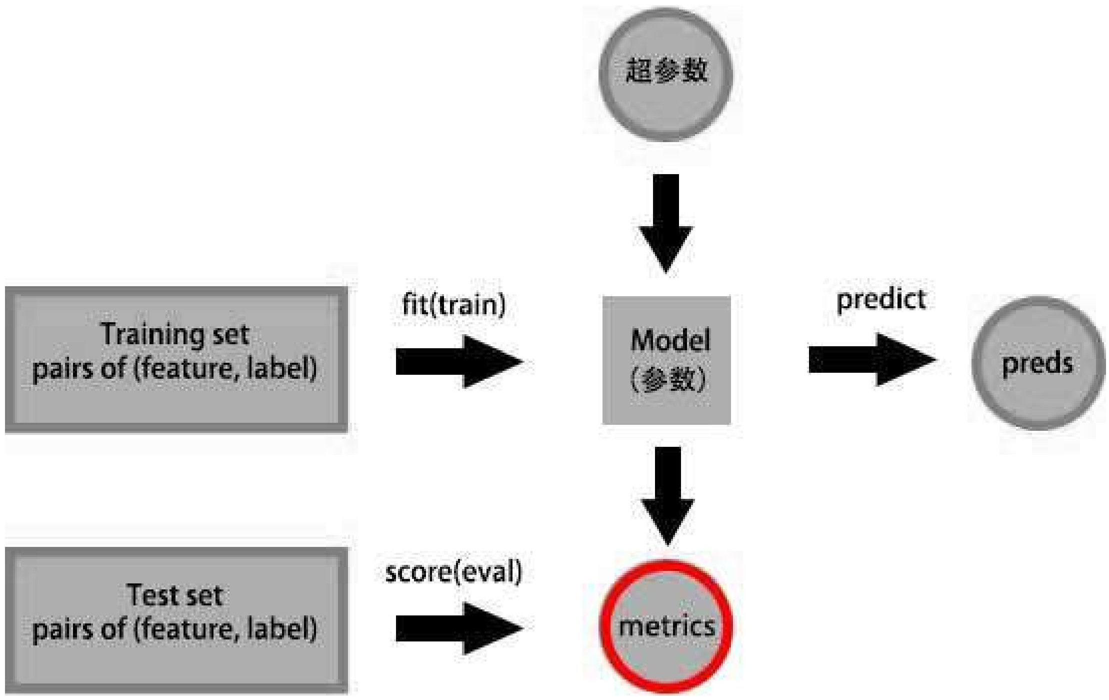
```
# regression with pipline

from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(TfidfVectorizer(), LogisticRegression())
pipeline.fit(text_train, y_train)
pipeline.score(text_test, y_test)
```

Out[23]:

0.97130559540889527

2.4 度量

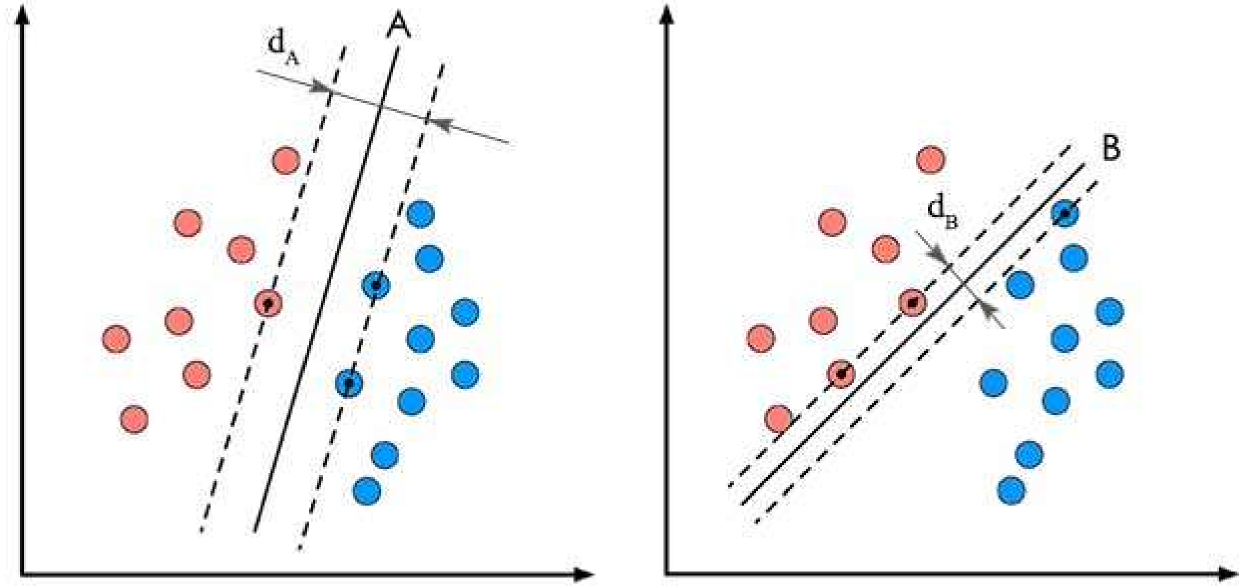
包括 **训练模型的度量** 和 **评测模型的度量**



训练模型的度量： 应用于 training 的SGD（stochastic Gradient Descent）

- 最大化 marginal distance

  代表模型 svm: $\max_{\mathbf{w},b} d_w \ \rightarrow \ \min_{\mathbf{w},b} \|\mathbf{w}\|^2$



- 最小化 MSE（mean square error)

  代表模型线性回归、逻辑回归: $\min_{\mathbf{w},b} \sum_i \|y_i - f(x_i)\|^2$

- Kullback–Leibler divergence 和 cross entroy

  神经网络等概率模型

$$H(P,Q) \equiv -\sum_i p_i \log q_i$$
$$D_{KL}(P\|Q) \equiv \sum_i p_i (\log p_i - \log q_i)$$

  $D_{KL}(P\|Q)$, is the amount of information lost when Q is used to approximate P.

  性质:

  1. 正定性: $D_{KL}(P\|Q) \geq 0$
  2. 可加性: 假设$p(x,y) \equiv p_1(x)p_2(y)$, $q(x,y) = q_1(x)q_2(y)$, 则 $D_{KL}(P\|Q) = D_{KL}(P_1\|Q_1) + D_{KL}(P_2\|Q_2)$

     $H(P,Q) = H(P) + D_{KL}(P\|Q)$，因此在一个最优化问题中，若所有参数集中在Q中，则minimize $D_{KL}(P\|Q)$ 等价于 minimize $H(P,Q)$

- other divergence, 如 Wasserstein divergence

评测模型的度量： 一般应用于test dataset

- R^2-score (regression)

$$r^2 \equiv 1 - \frac{\sum(y_i - f(x_i))^2}{\sum(y_i - \bar{y})^2}$$

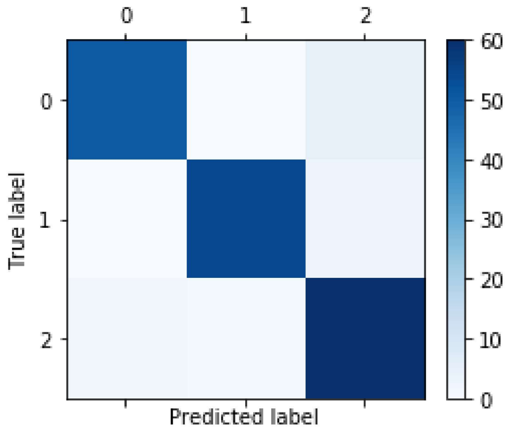
- Accuracy (classification)
- Confusion Matirx （classification)

$n = 175$	predicted label0	predicted label1	predicted label2

actual label0	50	0	5
actual label1	0	54	3
actual label2	2	1	60

In [85]:

```
y_test = [0]*55 + [1]*57 + [2]*63
y_test_pred = [0]*50 + [2]*5 + [1]*54 + [2]*3 + [0]*2 + [1]*1 + [2]*60
from sklearn.metrics import confusion_matrix

plt.matshow(confusion_matrix(y_test, y_test_pred), cmap="Blues")
plt.colorbar(shrink=0.8)
plt.xticks(range(3))
plt.yticks(range(3))
plt.xlabel("Predicted label")
plt.ylabel("True label");
```



- Precsion, Recall, F1-score (classification)

$$
\begin{aligned}
acc &= \frac{TP+TN}{all} \quad (\text{对角线上的样本数占样本总数的比例}) \\
pre &= \frac{TP}{TP+FP} \quad (\text{某列的对角线方块占该列样本数的比例}) \\
rec &= \frac{TP}{TP+FN} \quad (\text{某行的对角线方块占该行样本数的比例}) \\
\frac{2}{F} &= \frac{1}{pre} + \frac{1}{rec}
\end{aligned}
$$

In [34]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_pred))
```

```
             precision    recall  f1-score   support

          0       0.96      0.91      0.93        55
          1       0.98      0.95      0.96        57
          2       0.88      0.95      0.92        63

avg / total       0.94      0.94      0.94       175
```

- Fall-out，ROC_AUC score (binary classification)

应用于binary classification，如肿瘤positive/negative的诊断，或文章relevant/irrelevant的搜索

n=50	Predicted Positive	Predicted Negative
Actual Positive	TP=25	FN=2
Actual Negative	FP=1	TN=22

$$\text{fallOut} = \frac{FP}{FP + TN}$$

在文本搜索问题中，即

$$\text{fallOut} = \frac{\text{non-relevant documents} \cap \text{retrived documents}}{\text{non-relevant documents}}$$

ROC (Receiver operating characteristic) curve: a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. (Fallout vs Recall)

$$\text{ROC\_AUC score} = \text{area under ROC curve}$$

```
sklearn.metrics.roc_auc_score
```

其他常见度量

- confidence interval（置信区间）： An N% confidence interval for some parameter p is an interval that is expected with probability N% to contain p.
- winning rate 等