Python机器学习实践与Kaggle实战

Author: Miao Fan (范淼), Ph.D. candidate on Computer Science.

Affiliation: Tsinghua University / New York University

Bio: http://cslt.riit.tsinghua.edu.cn/mediawiki/images/b/bd/Miao_Fan_%282015_C.V.%29.pdf

Email: fanmiao.cslt.thu@gmail.com

Google Scholar: https://scholar.google.com/citations?user=aPlHReAAAAAJ&hl=en

Special Talk in NYU: http://cslt.riit.tsinghua.edu.cn/mediawiki/images/5/59/Special_talks_NYU--M.F.-.pdf

声明：

下面这些内容，都是学习《**Learning scikit-learn: Machine Learning in Python**》这本书的心得和一些拓展，写到哪算哪。**Scikit-learn**这个包我关注了**2**年，发展迅速；特别是它提供商业使用许可，比较有前景。

对于机器学习实践的"选手"，这是本入门的好书，国内目前没有中文译文版，我就先吃吃螃蟹。我个人认为，如果能够比较熟练掌握 **Scikit-learn**中的各种现有成熟模型的使用以及超参数优化（其实对超参数优化在工程中更加重要），那么**Kaggle**多数的竞赛大家基本可以进入**Top25%**。

这份长篇笔记中的代码链接目前都在本地，不就我会上传到**GITHUB**上。

平心而论，只有使用这些模型的经验丰富了，才能在实战中发挥作用，特别是对超参数和模型本身局限性的理解，恐怕不是书本所能教会的。

另外，更是有一些可以尝试的，并且曾经在**Kaggle**竞赛中多次获奖的模型包，比如 **Xgboost, gensim**等。**Tensorflow**究竟是否能够取得**Kaggle**竞赛的奖金，我还需要时间尝试。

同时，我个人近期也参与了《**Deep Learning**》这本优质新书多个章节贡献和校对，与三位作者平时的交流也深受启发。如果有兴趣的同学可以邮件本人，并一起参与中文笔记的撰文。

转载的朋友请注明来源，非常感谢。

平台选取：我个人推荐这个综合平台**Anaconda**进行练习**https://www.continuum.io/downloads**，同时新加入的其他包也可以在这个平台上拓展，几乎常用的操作系统都可以安装，一次性解决复杂的配置问题。

回国之后，对于我这个从来没摸过苹果电脑和系统的菜鸟，再添置一个**IMAC 27''**犒劳一下自己**:)** （题外话）。

因为后面的代码都是在**Ipython**环境下的，因此有一些地方没有**print**这个函数帮助输出，请读者留意。**In[*]/Out[*]**这种标记也是**Ipython**特有的。

这份笔记围绕**Python**下的机器学习实践一共探讨四个方面的内容：监督学习、无监督学习、特征和模型的选取 和 几个流行的强力模型包的使用。

我特别喜欢用几句话对某些东西做个总结，对于**Kaggle**的任务，我个人觉得大体需要这么几个固定的机器学习流程（不包括决定性的分析），如果按照这个流程，采用**scikit-learn & pandas**包的话，普遍都会进**Top25%**：

1) pandas 读 **csv**或者**tsv (Kaggle**上的数据基本都比较整洁**)**

2) 特征少的话，补全数据，**feature_extraction (DictVec, tfidfVec**等等，根据数据类型而异，文本，图像，音频，这些处理方式都不同**)**，**feature_selection, grid_searching the best hyperparameters(model_selection), ensemble learning** （或者综合好多学习器的结果），**predict** 或者 **proba_predict** （取决于任务的提交要求，是直接分类结果，还是分类概率，这个区别很大）。

3) 特征多的话，补全数据，**feature_extraction (DictVec, tfidfVec**等等，根据数据类型而异，文本，图像，音频，这些处理方式都不同**)**，数据降维度（**PCA**，**RBM**等等），**feature_selection (**如果降维度之后还有必要**), ensemble learning**

（或者综合好多学习器的结果），**predict** 或者 **proba_predict** （取决于任务的提交要求，是直接分类结果，还是分类概率，这个区别很大）。

1. 监督学习

~~~~~~.

~~~~~~.

1.1 线性分类器

使用**Scikit-learn**数据库中预装的牵牛花品种数据，进行线性分类实践。线性分类器中，**Logistic Regression**比较常用，适合做概率估计，即对分配给每个类别一个估计概率。这个在**Kaggle**竞赛里经常需要。

http://localhost:8888/notebooks/PythonNotebook/Scikit-learn/linear_classifier.ipynb

In [1]:

```
from sklearn.datasets import load_iris
from sklearn.cross_validation import train_test_split
from sklearn import preprocessing

# 读取数据
iris = load_iris()

# 选取特征与标签
X_iris, y_iris = iris.data, iris.target

# 选择前两列数据作为特征
X, y = X_iris[:, :2], y_iris

# 选取一部分，25%的训练数据作为测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state = 33)

# 对原特征数据进行标准化预处理，这个其实挺重要，但是经常被一些选手忽略
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import SGDClassifier

# 选择使用SGD分类器，适合大规模数据，随机梯度下降方法估计参数
clf = SGDClassifier()

clf.fit(X_train, y_train)

# 导入评价包
from sklearn import metrics

y_train_predict = clf.predict(X_train)

# 内测，使用训练样本进行准确性能评估
print metrics.accuracy_score(y_train, y_train_predict)

# 标准外测，使用测试样本进行准确性能评估
y_predict = clf.predict(X_test)
print metrics.accuracy_score(y_test, y_predict)
```

```
0.660714285714
0.684210526316
```

In [2]:

```
# 如果需要更加详细的性能报告，比如precision, recall, accuracy，可以使用如下的函数。
print metrics.classification_report(y_test, y_predict, target_names = iris.target_names)
```

```
             precision    recall  f1-score   support
```

```
     setosa       1.00      1.00      1.00         8
 versicolor       0.43      0.27      0.33        11
  virginica       0.65      0.79      0.71        19

avg / total       0.66      0.68      0.66        38
```

In [4]:

```
# 如果想详细探查SGDClassifier的分类性能，我们需要充分利用数据，因此需要把数据切分为N个部分，每个部分都用于测试一次模型性能。

from sklearn.cross_validation import cross_val_score, KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
# 这里使用Pipeline，便于精简模型搭建，一般而言，模型在fit之前，对数据需要feature_extraction, preprocessing, 等必要步骤。
# 这里我们使用默认的参数配置
clf = Pipeline([('scaler', StandardScaler()), ('sgd_classifier', SGDClassifier())])

# 5折交叉验证整个数据集合
cv = KFold(X.shape[0], 5, shuffle=True, random_state = 33)

scores = cross_val_score(clf, X, y, cv=cv)
print scores

# 计算一下模型综合性能，平均精度和标准差
print scores.mean(), scores.std()

from scipy.stats import sem
import numpy as np

# 这里使用的偏差计算函数略有不同，参考链接
http://www.graphpad.com/guides/prism/6/statistics/index.htm?stat_semandsdnotsame.htm

print np.mean(scores), sem(scores)
```

```
[ 0.56666667  0.73333333  0.83333333  0.76666667  0.8        ]
0.74 0.0928559218479
0.74 0.0464279609239
```

总结一下：线性分类器有几种， **Logistic_regression**在**scikit-learn**里也有实现。比起**SGD**这个分类器而言，前者使用更加精确，但是更加耗时的解析解。**SGD**分类器可以大体代表这些线性分类器的性能，但是由于是近似估计的参数，因此模型性能结果不是很稳定，需要通过调节超参数获得模型的性能上限。

~~~~~~.

1.2 SVM 分类器

这一部分，我们探究支持向量机，这是个强分类器，性能要比普通线性分类器强大一些，一般而言，基于的也是线性假设。但是由于可以引入一些核技巧(**kernel trick**)，可以将特征映射到更加高维度，甚至非线性的空间上，从而使数据空间变得更加可分。再加上**SVM**本身只是会选取少量的支持向量作为确定分类器超平面的证据，因此，即便数据变得高维度，非线性映射，也不会占用太多的内存空间，只是计算这些支持向量的**CPU**代价比较高。另外，这个分类器适合于直接做分类，不适合做分类概率的估计。

http://localhost:8888/notebooks/PythonNotebook/Scikit-learn/SVM_classifier.ipynb

这里我们使用 **AT&T 400**张人脸，这个经典数据集来介绍：

In [1]:

```
from sklearn.datasets import fetch_olivetti_faces

# 这部分数据没有直接存储在现有包中，都是通过这类函数在线下载
faces = fetch_olivetti_faces()
```

In [2]:

```
# 这里证明，数据是以Dict的形式存储的，与多数实验性数据的格式一致
faces.keys()
```

Out[2]:

```
['images', 'data', 'target', 'DESCR']
```

In [3]:
~~~~~~

```
# 使用shape属性检验数据规模
print faces.data.shape
print faces.target.shape
```

```
(400L, 4096L)
(400L,)
```

In [4]:

```
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC

# 同样是分割数据 25%用于测试
X_train, X_test, y_train, y_test = train_test_split(faces.data, faces.target, test_size=0.25, random_state=0)
```

In [5]:

```
from sklearn.cross_validation import cross_val_score, KFold
from scipy.stats import sem

# 构造一个便于交叉验证模型性能的函数（模块）
def evaluate_cross_validation(clf, X, y, K):
    # KFold 函数需要如下参数：数据量, 叉验次数, 是否洗牌
    cv = KFold(len(y), K, shuffle=True, random_state = 0)

    # 采用上述的分隔方式进行交叉验证，测试模型性能，对于分类问题，这些得分默认是accuracy，也可以修改为别的
    scores = cross_val_score(clf, X, y, cv=cv)
    print scores
    print 'Mean score: %.3f (+/-%.3f)' % (scores.mean(), sem(scores))

# 使用线性核的SVC （后面会说到不同的核，结果可能大不相同）
svc_linear = SVC(kernel='linear')
# 五折交叉验证 K = 5
evaluate_cross_validation(svc_linear, X_train, y_train, 5)
```

```
[ 0.93333333  0.86666667  0.91666667  0.93333333  0.91666667]
Mean score: 0.913 (+/-0.012)
```

~~~~~~.

1.3 朴素贝叶斯分类器（Naive Bayes)

这一部分我们探讨朴素贝叶斯分类器，大量实验证明，这个分类模型在对文本分类中能表现良好。究其原因，也许是对于邮件过滤这类任务，我们用于区分类别的文本特征彼此独立性较强，刚好模型的假设便是特征独立。

http://localhost:8888/notebooks/PythonNotebook/Scikit-learn/NB_classifier.ipynb

In [1]:

```
from sklearn.datasets import fetch_20newsgroups
```

In [2]:

```
# 与之前的人脸数据集一样，20类新闻数据同样需要临时下载函数的帮忙
news = fetch_20newsgroups(subset='all')
```

In [9]:

```
# 查验数据，依然采用dict格式，共有18846条样本
print len(news.data), len(news.target)
print news.target
```

```
18846 18846
[10  3 17 ...,  3  1  7]
```

In [4]:

```
# 查验一下新闻类别和种数
print news.target_names
print news.target_names.__len__()
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'mis
20
```

In [5]:

```
# 同样，我们选取25%的数据用来测试模型性能
from sklearn.cross_validation import train_test_split
```
~~~~~~

```
X_train, X_test, y_train, y_test = train_test_split(news.data, news.target, test_size=0.25)
```

In [6]:

```
print X_train.__len__()
print y_train.__len__()
print X_test.__len__()
```

```
14134
14134
4712
```

In [13]:

```
# 许多原始数据无法直接被分类器所使用，图像可以直接使用pixel信息，文本则需要进一步处理成数值化的信息
from sklearn.feature_extraction.text import CountVectorizer, HashingVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import *
from scipy.stats import sem
# 我们在NB_Classifier的基础上，对比几种特征抽取方法的性能。并且使用Pipline简化构建训练流程
clf_1 = Pipeline([('count_vec', CountVectorizer()), ('mnb', MultinomialNB())])
clf_2 = Pipeline([('hash_vec', HashingVectorizer(non_negative=True)), ('mnb', MultinomialNB())])
clf_3 = Pipeline([('tfidf_vec', TfidfVectorizer()), ('mnb', MultinomialNB())])

# 构造一个便于交叉验证模型性能的函数（模块）
def evaluate_cross_validation(clf, X, y, K):
    # KFold 函数需要如下参数，数据量，K,是否洗牌
    cv = KFold(len(y), K, shuffle=True, random_state = 0)
    # 采用上述的分隔方式进行交叉验证，测试模型性能，对于分类问题，这些得分默认是accuracy，也可以修改为别的
    scores = cross_val_score(clf, X, y, cv=cv)
    print scores
    print 'Mean score: %.3f (+/-%.3f)' % (scores.mean(), sem(scores))
```

In [14]:

```
clfs = [clf_1, clf_2, clf_3]
for clf in clfs:
    evaluate_cross_validation(clf, X_train, y_train, 5)
```

```
[ 0.83516095  0.83374602  0.84471171  0.83622214  0.83227176]
Mean score: 0.836 (+/-0.002)
[ 0.76052352  0.72727273  0.77538026  0.74778918  0.75194621]
Mean score: 0.753 (+/-0.008)
[ 0.84435798  0.83409975  0.85496993  0.84082066  0.83227176]
Mean score: 0.841 (+/-0.004)
```

In [15]:

```
# 从上述结果中，我们发现常用的两个特征提取方法得到的性能相当。 让我们选取其中之一，进一步靠特征的精细筛选提升性能。
clf_4 = Pipeline([('tfidf_vec_adv', TfidfVectorizer(stop_words='english')), ('mnb', MultinomialNB())])
evaluate_cross_validation(clf_4, X_train, y_train, 5)
```

```
[ 0.87053414  0.86664308  0.887867    0.87371772  0.86553432]
Mean score: 0.873 (+/-0.004)
```

In [16]:

```
# 如果再尝试修改贝叶斯分类器的平滑参数，也许性能会更上一层楼。
clf_5 = Pipeline([('tfidf_vec_adv', TfidfVectorizer(stop_words='english')), ('mnb', MultinomialNB(alpha=0.01))])
evaluate_cross_validation(clf_5, X_train, y_train, 5)
```

```
[ 0.90060134  0.89741776  0.91651928  0.90909091  0.90410474]
Mean score: 0.906 (+/-0.003)
```

~~~~~~.

1.4 决策树分类器（**Decision Tree**) **/** 集成分类器（**Ensemble Tree**）

之前的分类器大多有一下几点缺陷：

a)线性分类器对于特征与类别直接的关系是"线性假设"，如果遇到非线性的关系，就很难辨识，比如**Titanic**数据中，如果假设"年龄"与"生存"是正相关的，那么年龄越大，越有可能生存；但是事实证明，这个假设是错误的，不是正相关，而偏偏是老人与小孩更
~~~~~~

加容易获得生存的机会。这种情况，线性假设不完全成立，因此，需要非线性的分类器。
b)即便使用类似**SVM**的分类器，我们很难得到明确分类"依据"的说明，无法"解释"分类器是如何工作的，特别无法从人类逻辑的角度理解。高维度、不可解释性等，这些都是弊端。

决策树分类器解决了上述两点问题。我们使用**Titanic**（泰坦尼克号的救援记录）这个数据集来实践一个预测某乘客是否获救的分类器。

http://localhost:8888/notebooks/PythonNotebook/Scikit-learn/DecisionTree_classifier.ipynb

In [1]:

```
# 这里为了处理数据方便，我们引入一个新的工具包pandas

import pandas as pd
import numpy as np

titanic = pd.read_csv('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt')
```

In [2]:

```
#瞧瞧数据，什么数据特征的都有，有数值型的、类别型的，字符串，甚至还有缺失的数据等等。
titanic.head()
```

Out[2]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	boat	sex
0	1	1st	1	Allen, Miss Elisabeth Walton	29.0000	Southampton	St Louis, MO	B-5	24160 L221	2	female
1	2	1st	0	Allison, Miss Helen Loraine	2.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN	female
2	3	1st	0	Allison, Mr Hudson Joshua Creighton	30.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	(135)	male
3	4	1st	0	Allison, Mrs Hudson J.C. (Bessie Waldo Daniels)	25.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN	female
4	5	1st	1	Allison, Master Hudson Trevor	0.9167	Southampton	Montreal, PQ / Chesterville, ON	C22	NaN	11	male

In [3]:

```
# 使用pandas，数据都转入pandas独有的dataframe格式（二维数据表格），直接使用info()，查看数据的基本特征
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1313 entries, 0 to 1312
Data columns (total 11 columns):
row.names    1313 non-null int64
pclass       1313 non-null object
survived     1313 non-null int64
name         1313 non-null object
age          633 non-null float64
embarked     821 non-null object
home.dest    754 non-null object
room         77 non-null object
ticket       69 non-null object
boat         347 non-null object
sex          1313 non-null object
dtypes: float64(1), int64(2), object(8)
memory usage: 123.1+ KB
```

In [4]:

```
# 这份调查数据是真实的泰坦尼克号乘客个人和登船信息，有助于我们预测每位遇难乘客是否幸免。
# 一共1313条数据，有些特征是完整的（比如 pclass, survived, name），有些是有缺失的；有些是数值类型的信息（age: float64），有些则是字符串。
# 机器学习有一个不太被初学者重视，并且耗时，但是十分重要的一环，特征的选择，这个需要基于一些背景知识。根据我们对这场事故的了解，sex, age, pclass这些

# we keep pclass, age, sex.

X = titanic[['pclass', 'age', 'sex']]
y = titanic['survived']
```

In [5]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 1313 entries, 0 to 1312
Data columns (total 3 columns):
pclass    1313 non-null object
age       633 non-null float64
sex       1313 non-null object
dtypes: float64(1), object(2)
memory usage: 41.0+ KB
```

In [6]:

```
# 下面有几个对数据处理的任务
# 1) age这个数据列，只有633个
# 2) sex 与 pclass两个数据列的值都是类别型的，需要转化为数值特征，用0/1代替

# 首先我们补充age里的数据，使用平均数或者中位数都是对模型偏离造成最小影响的策略
X['age'].fillna(X['age'].mean(), inplace=True)
```

```
C:\Anaconda2\lib\site-packages\pandas\core\generic.py:2748: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
```

In [7]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1313 entries, 0 to 1312
Data columns (total 3 columns):
pclass    1313 non-null object
age       1313 non-null float64
sex       1313 non-null object
dtypes: float64(1), object(2)
memory usage: 41.0+ KB
```

In [8]:

```
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state = 33)

# 我们使用scikit-learn中的feature_extraction
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer(sparse=False)
X_train = vec.fit_transform(X_train.to_dict(orient='record'))
print vec.feature_names_
# 我们发现，凡是类别型的特征都单独剥离出来，独成一列特征，数值型的则保持不变
```

```
['age', 'pclass=1st', 'pclass=2nd', 'pclass=3rd', 'sex=female', 'sex=male']
```

In [9]:

```
X_test = vec.transform(X_test.to_dict(orient='record'))
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=5)
dtc.fit(X_train, y_train)
dtc.score(X_test, y_test)
```

Out[9]:

```
0.79331306990881456
```

In [10]:

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(max_depth=3, min_samples_leaf=5)
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)
```

Out[10]:

```
0.77203647416413379
```

In [11]:

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(max_depth=3, min_samples_leaf=5)

gbc.fit(X_train, y_train)
```

```
gbc.score(X_test, y_test)
```

Out[11]:

```
0.79027355623100304
```

In [13]:

```
from sklearn.metrics import classification_report

y_predict = gbc.predict(X_test)
print classification_report(y_predict, y_test)
# 这里的函数可以便于生成分类器性能报告（precision,recall)这些是在二分类背景下才有的指标。
```

```
             precision    recall  f1-score   support

          0       0.93      0.78      0.84       241
          1       0.57      0.83      0.68        88

avg / total       0.83      0.79      0.80       329
```

~~~~~~.

1.5 回归问题（Regressions）

回归问题和分类问题都同属于监督学习范畴，唯一不同的是，回归问题的预测目标是在无限的连续实数域，比如预测房价、股票价格等等；分类问题则是对有限范围的几个类别（离散数）进行预测。当然两者的界限不一定泾渭分明，也可以适度转化。比如，有一个经典的对红酒质量的预测，大体分为10等级，怎样看待这个预测目标，都是可以的。预测的结果，可以在（0-10]区间连续（回归问题），也可以只预测10个等级的某个值（分类问题）。

这里我们举一个预测美国波士顿地区房价的问题，这是个经典的回归问题，我们一步步采用各种用于回归问题的训练模型，一步步尝试提升模型的回归性能。

In [1]:

```
# 首先预读房价数据
from sklearn.datasets import load_boston

boston = load_boston()

# 查验数据规模
print boston.data.shape
```

```
(506L, 13L)
```

In [2]:

```
# 多多弄懂数据特征的含义也是一个好习惯
print boston.feature_names
print boston.DESCR
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
Boston House Prices dataset

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
```
~~~~~~

```
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Mach
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
```

In [3]:

```
# 这里多一个步骤，查验数据是否正规化，一般都是没有的
import numpy as np

print np.max(boston.target)
print np.min(boston.target)
print np.mean(boston.target)
```

```
50.0
5.0
22.5328063241
```

In [4]:

```
from sklearn.cross_validation import train_test_split
# 依然如故，我们对数据进行分割
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, test_size = 0.25, random_state=33)

from sklearn.preprocessing import StandardScaler

# 正规化的目的在于避免原始特征值差异过大，导致训练得到的参数权重不一
scalerX = StandardScaler().fit(X_train)
X_train = scalerX.transform(X_train)
X_test = scalerX.transform(X_test)

scalery = StandardScaler().fit(y_train)
y_train = scalery.transform(y_train)
y_test = scalery.transform(y_test)
```

In [5]:

```
# 先把评价模块写好，依然是默认5折交叉验证，只是这里的评价指标不再是精度，而是另一个函数R2，大体上，这个得分多少代表有多大百分比的回归结果可以被训练器覆
from sklearn.cross_validation import *

def train_and_evaluate(clf, X_train, y_train):
    cv = KFold(X_train.shape[0], 5, shuffle=True, random_state=33)
    scores = cross_val_score(clf, X_train, y_train, cv=cv)
    print 'Average coefficient of determination using 5-fold cross validation:', np.mean(scores)

#最后让我们看看有多少种回归模型可以被使用（其实有更多）。
# 比较有代表性的有3种
```

In [7]:

```
# 先用线性模型尝试， SGD_Regressor
from sklearn import linear_model
# 这里有一个正则化的选项penalty，目前14维特征也许不会有太大影响
```

```
clf_sgd = linear_model.SGDRegressor(loss='squared_loss', penalty=None, random_state=42)
train_and_evaluate(clf_sgd, X_train, y_train)
```

```
Average coefficient of determination using 5-fold cross validation: 0.710809853468
```

In [8]:

```
# 再换一个SGD_Regressor的penalty参数为l2,结果貌似影响不大，因为特征太少，正则化意义不大
clf_sgd_l2 = linear_model.SGDRegressor(loss='squared_loss', penalty='l2', random_state=42)
train_and_evaluate(clf_sgd_l2, X_train, y_train)
```

```
Average coefficient of determination using 5-fold cross validation: 0.71081206667
```

In [9]:

```
# 再看看SVM的regressor怎么样（都是默认参数），
from sklearn.svm import SVR
# 使用线性核没有啥子提升，但是因为特征少，所以可以考虑升高维度
clf_svr = SVR(kernel='linear')
train_and_evaluate(clf_svr, X_train, y_train)
```

```
Average coefficient of determination using 5-fold cross validation: 0.707838419194
```

In [11]:

```
clf_svr_poly = SVR(kernel='poly')
# 升高维度，效果明显，但是此招慎用@@，特征高的话，CPU还是受不了，内存倒是小事。其实到了现在，连我们自己都没办法直接解释这些特征的具体含义了。
train_and_evaluate(clf_svr_poly, X_train, y_train)
```

```
Average coefficient of determination using 5-fold cross validation: 0.779288545488
```

In [12]:

```
clf_svr_rbf = SVR(kernel='rbf')
# RBF (径向基核更是牛逼！)
train_and_evaluate(clf_svr_rbf, X_train, y_train)
```

```
Average coefficient of determination using 5-fold cross validation: 0.833662221567
```

In [14]:

```
# 再来个更猛的! 极限回归森林，放大招了！！！
from sklearn import ensemble
clf_et = ensemble.ExtraTreesRegressor()
train_and_evaluate(clf_et, X_train, y_train)
```

```
Average coefficient of determination using 5-fold cross validation: 0.853006383633
```

In [15]:

```
# 最后看看在测试集上的表现
clf_et.fit(X_train, y_train)
clf_et.score(X_test, y_test)
```

Out[15]:

```
0.83781467779895469
```

总结来看，我们可以通过这个例子得到机器学习不断进取的快感！一点点提高模型性能，并且，也能感觉到超参数的作用有时比更换模型的提升效果更好。而且这里也为后续的"特征选择"，"模型选择"等实用话题埋下了伏笔。

~~~~~~.

2. 无监督学习

~~~~~~.

~~~~~~.
~~~~~~

3. 特征、模型的选择（高级话题）

~~~~~~.

~~~~~~.

3.1 特征选择 (feature_selection)

这里我们继续沿用Titanic数据集，这次侧重于对模型的区分能力贡献最大的几个特征选取的问题。

不良的特征会对模型的精度"拖后腿"；冗余的特征虽然不会影响模型的精度，不过CPU计算做了无用功。

我个人理解，这种特征选择与PCA这类特征压缩选择主成分的略有区别：PCA重建之后的特征我们已经无法解释其意义了。

http://localhost:8888/notebooks/PythonNotebook/Scikit-learn/Feature_selection.ipynb

In [1]:

```
# 这部分代码和原著的第四章节有相同的效果，但是充分利用pandas会表达的更加简洁，因此我重新编写了更加清晰简洁的代码。
import pandas as pd
import numpy as np

titanic = pd.read_csv('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt')

print titanic.info()
# 还是这组数据
titanic.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1313 entries, 0 to 1312
Data columns (total 11 columns):
row.names    1313 non-null int64
pclass       1313 non-null object
survived     1313 non-null int64
name         1313 non-null object
age          633 non-null float64
embarked     821 non-null object
home.dest    754 non-null object
room         77 non-null object
ticket       69 non-null object
boat         347 non-null object
sex          1313 non-null object
dtypes: float64(1), int64(2), object(8)
memory usage: 123.1+ KB
None
```

Out[1]:

	row.names	pclass	survived	name	age	embarked	home.dest	room	ticket	boat	sex
0	1	1st	1	Allen, Miss Elisabeth Walton	29.0000	Southampton	St Louis, MO	B-5	24160 L221	2	female
1	2	1st	0	Allison, Miss Helen Loraine	2.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN	female
2	3	1st	0	Allison, Mr Hudson Joshua Creighton	30.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	(135)	male
3	4	1st	0	Allison, Mrs Hudson J.C. (Bessie Waldo Daniels)	25.0000	Southampton	Montreal, PQ / Chesterville, ON	C26	NaN	NaN	female
4	5	1st	1	Allison, Master Hudson Trevor	0.9167	Southampton	Montreal, PQ / Chesterville, ON	C22	NaN	11	male

In [2]:

```
# 我们丢掉一些过于特异的，不利于找到共同点的数据列， row.names, name, 同时分离出预测列。

y = titanic['survived']
X = titanic.drop(['row.names', 'name', 'survived'], axis = 1)
```

In [3]:

```
# 对于连续的数值特征，我们采用补完的方式
X['age'].fillna(X['age'].mean(), inplace=True)
```

```
X.fillna('UNKNOWN', inplace=True)
```

In [4]:

```
# 剩下的类别类型数据，我们直接向量化，这样的话，对于有空白特征的列，我们也单独视作一个特征

from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=33)

from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer()
X_train = vec.fit_transform(X_train.to_dict(orient='record'))
X_test = vec.transform(X_test.to_dict(orient='record'))
```

In [5]:

```
print len(vec.feature_names_)
```

```
474
```

In [6]:

```
X_train.toarray()
```

Out[6]:

```
array([[ 31.19418104,   0.        ,   0.        , ...,   0.        ,
          0.        ,   1.        ],
       [ 31.19418104,   0.        ,   0.        , ...,   0.        ,
          0.        ,   0.        ],
       [ 31.19418104,   0.        ,   0.        , ...,   0.        ,
          0.        ,   1.        ],
       ...,
       [ 12.        ,   0.        ,   0.        , ...,   0.        ,
          0.        ,   1.        ],
       [ 18.        ,   0.        ,   0.        , ...,   0.        ,
          0.        ,   1.        ],
       [ 31.19418104,   0.        ,   0.        , ...,   0.        ,
          0.        ,   1.        ]])
```

In [7]:

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train, y_train)
dt.score(X_test, y_test)
# 采用所有特征的测试精度
```

Out[7]:

```
0.81762917933130697
```

In [8]:

```
from sklearn import feature_selection
fs = feature_selection.SelectPercentile(feature_selection.chi2, percentile=20)

X_train_fs = fs.fit_transform(X_train, y_train)
dt.fit(X_train_fs, y_train)
X_test_fs = fs.transform(X_test)
dt.score(X_test_fs, y_test)
# 采用20%高预测性特征的测试精度
```

Out[8]:

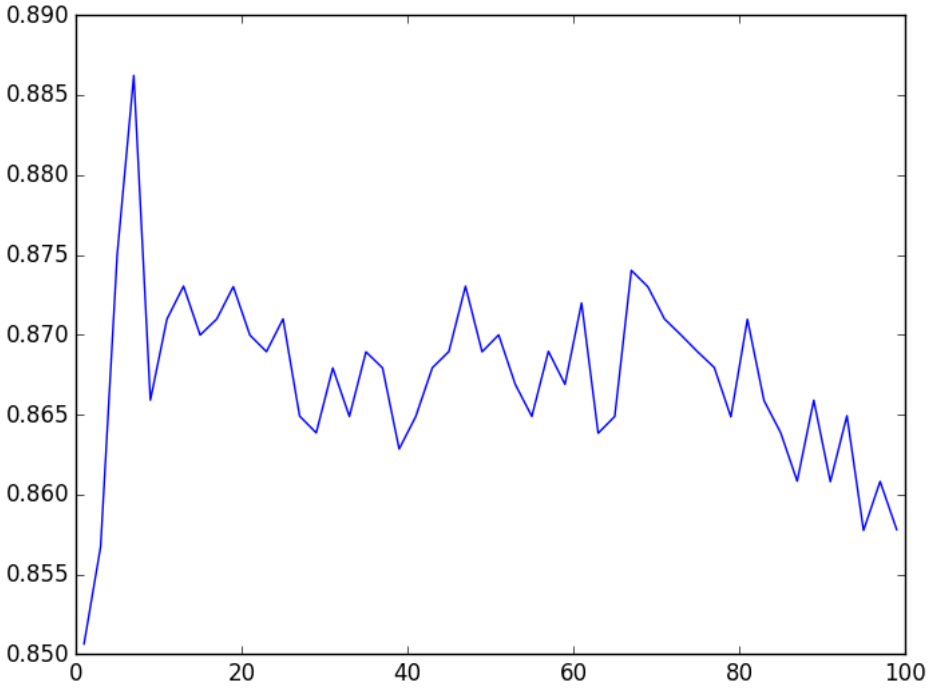
```
0.82370820668693012
```

In [9]:

```
from sklearn.cross_validation import cross_val_score
percentiles = range(1, 100, 2)

results = []

for i in percentiles:
    fs = feature_selection.SelectPercentile(feature_selection.chi2, percentile = i)
    X_train_fs = fs.fit_transform(X_train, y_train)
    scores = cross_val_score(dt, X_train_fs, y_train, cv=5)
    results = np.append(results, scores.mean())
print results

opt = np.where(results == results.max())[0]
print 'Optimal number of features %d' %percentiles[opt]
import pylab as pl
```

```
pl.plot(percentiles, results)
pl.show()
```

```
[ 0.85063904  0.85673057  0.87501546  0.88622964  0.86590394  0.87097506
  0.87303649  0.86997526  0.87097506  0.87300557  0.86997526  0.86893424
  0.87098536  0.86490414  0.86385281  0.86791383  0.86488353  0.86892393
  0.86791383  0.86284271  0.86487322  0.86792414  0.86894455  0.87303649
  0.86892393  0.86998557  0.86689342  0.86488353  0.86895485  0.86689342
  0.87198516  0.8638322   0.86488353  0.87402597  0.87299526  0.87098536
  0.86997526  0.86892393  0.86794475  0.86486291  0.87096475  0.86587302
  0.86387343  0.86083282  0.86589363  0.8608019   0.86492476  0.85774067
  0.8608122   0.85779221]
Optimal number of features 7
```



In [10]:

```
from sklearn import feature_selection
fs = feature_selection.SelectPercentile(feature_selection.chi2, percentile=7)

X_train_fs = fs.fit_transform(X_train, y_train)
dt.fit(X_train_fs, y_train)
X_test_fs = fs.transform(X_test)
dt.score(X_test_fs, y_test)
# 选取搜索到的最好特征比例的测试精度
```

Out[10]:

```
0.8571428571428571
```

In [ ]:

```
# 由此可见，这个技术对于工程上提升精度还是非常有帮助的。
```

3.2 模型（超参数）选择

由于超参数的空间是无尽的，因此超参数的组合配置只能是"更优"解，没有最优解。通常情况下，我们依靠"网格搜索"**(GridSearch)**对固定步长的超参数空间进行暴力搜索，对于每组超参数组合代入到学习函数中，视为新模型。为了比较新模型之间的性能，每个模型都会在相同的训练、开发数据集下进行评估，通常我们采用交叉验证。因此，这个过程非常耗时，但是一旦获取比较好的参数，则可以保持一段时间使用，也相对一劳永逸。好在，由于各个新模型的交叉验证之间是互相独立的，因此，可以充分利用多核甚至是分布式的计算资源来并行搜索（**Parallel Grid Search**）。

http://localhost:8888/notebooks/PythonNotebook/Scikit-learn/Model_selection.ipynb

4. 强力（流行）模型包的尝试（高级话题）

~~~~~~·

~~~~~~·

这个话题有几个独立的部分，对于**Xgboost**和**Tensorflow**的试验，需要**Linux**环境。待回国后用**IMAC**试试**:)**。

不过仍然有一份高级一点的**NLP**相关的内容可以探讨，其中就有**Kaggle**上面利用**Word2Vec**对情感分析任务助益的项目。我们这里先来分析一下。

https://www.kaggle.com/c/word2vec-nlp-tutorial

4.1. 词向量对NLP相关任务的助益