第 3 章

进　阶　篇

在第 2 章中，我们向读者介绍了大量经典的机器学习模型，并且使用 Python 编程语言分析这些模型在许多不同现实数据上的性能表现。然而，细心的读者在深入研究这些数据或者查阅 Scikit-learn 的文档之后就会发现：所有我们在第 2 章中使用过的数据几乎都经过了规范化处理，而且模型也大多只是采用了默认的初始化配置。换言之，尽管我们可以使用经过处理之后的数据，在默认配置下学习到一套用以拟合这些数据的参数，并且使用这些参数和默认配置取得一些看似良好的性能表现；但是我们仍然无法回答几个最为关键的问题：实际研究和工作中接触到的数据都是这样规整的吗？难道这些默认配置就是最佳的么？我们的模型性能是否还有提升的空间？本章“3.1 模型使用技巧”节将会帮助读者朋友解答上述疑问。阅读完这一节，相信各位读者朋友就会掌握如何通过抽取或者筛选数据特征、优化模型配置，进一步提升经典模型的性能表现。

然而，随着近些年机器学习研究与应用的快速发展，经典模型渐渐无法满足日益增长的数据量和复杂的数据分析需求。因此，越来越多更加高效而且强力的学习模型以及对应的程序库正逐渐被设计和编写，并慢慢被科研圈和工业界所广泛接受与采用。这些模型和程序库包括：用于自然语言处理的 NLTK 程序包；词向量技术 Word2Vec；能够提供强大预测能力的 XGBoost 模型，以及 Google 发布的用于深度学习的 Tensorflow 框架等等。更加令人振奋的是，上述这些最为流行的程序库和模型，不但提供了 Python 的编程接口 API，而且有些成为 Python 编程语言的工具包，更是方便了我们后续的学习和使用。因此，在“3.2 流行库/模型实践”节将会带领各位读者一同领略这些时下最为流行的程序库和新模型的奥妙。

3.1　模型实用技巧

这一节将向读者朋友传授一系列更加偏向于实战的模型使用技巧。相信各位读者在第 2 章中品味了多个经典的机器学习模型之后，就会发现：一旦我们确定使用某个模型，

本书所提供的程序库就可以帮助我们从标准的训练数据中，依靠默认的配置学习到模型所需要的参数(Parameters)；接下来，我们便可以利用这组得来的参数指导模型在测试数据集上进行预测，进而对模型的表现性能进行评价。

但是，这套方案并不能保证：(1) 所有用于训练的数据特征都是最好的；(2) 学习得到的参数一定是最优的；(3) 默认配置下的模型总是最佳的。也就是说，我们可以从多个角度对在前面所使用过的模型进行性能提升。本节将向大家介绍多种提升模型性能的方式，包括如何预处理数据、控制参数训练以及优化模型配置等方法。

3.1.1　特征提升

早期机器学习的研究与应用，受模型种类和运算能力的限制。因此，大部分研发人员把更多的精力放在对数据的预处理上。他们期望通过对数据特征的抽取或者筛选来达到提升模型性能的目的。所谓特征抽取，就是逐条将原始数据转化为特征向量的形式，这个过程同时涉及对数据特征的量化表示；而特征筛选则更进一步，在高维度、已量化的特征向量中选择对指定任务更有效的特征组合，进一步提升模型性能。

3.1.1.1　特征抽取

原始数据的种类有很多种，除了数字化的信号数据(声纹、图像)，还有大量符号化的文本。然而，我们无法直接将符号化的文字本身用于计算任务，而是需要通过某些处理手段，预先将文本量化为特征向量。

有些用符号表示的数据特征已经相对结构化，并且以字典这种数据结构进行存储。这时，我们使用 DictVectorizer 对特征进行抽取和向量化。比如下面的代码 55。

代码 55：DictVectorizer 对使用字典存储的数据进行特征抽取与向量化

```
>>>#定义一组字典列表，用来表示多个数据样本(每个字典代表一个数据样本)。
>>>measurements=[{'city': 'Dubai', 'temperature': 33.}, {'city': 'London', '
temperature': 12.}, {'city': 'San Fransisco', 'temperature': 18.}]
>>>#从 sklearn.feature_extraction 导入 DictVectorizer
>>>from sklearn.feature_extraction import DictVectorizer
>>>#初始化 DictVectorizer 特征抽取器
>>>vec=DictVectorizer()
>>>#输出转化之后的特征矩阵。
>>>print vec.fit_transform(measurements).toarray()
>>>#输出各个维度的特征含义。
>>>print vec.get_feature_names()
```

```
[[ 1.  0.  0.  33.]
 [ 0.  1.  0.  12.]
 [ 0.  0.  1.  18.]]
 ['city=Dubai', 'city=London', 'city=San Fransisco', 'temperature']
```

从代码55的输出可以看到：在特征向量化的过程中，DictVectorizer对于类别型(Categorical)与数值型(Numerical)特征的处理方式有很大差异。由于类别型特征无法直接数字化表示，因此需要借助原特征的名称，组合产生新的特征，并采用0/1二值方式进行量化；而数值型特征的转化则相对方便，一般情况下只需要维持原始特征值即可。

另外一些文本数据则表现得更为原始，几乎没有使用特殊的数据结构进行存储，只是一系列字符串。我们处理这些数据，比较常用的文本特征表示方法为词袋法(Bag of Words)：顾名思义，不考虑词语出现的顺序，只是将训练文本中的每个出现过的词汇单独视作一列特征。我们称这些不重复的词汇集合为词表(Vocabulary)，于是每条训练文本都可以在高维度的词表上映射出一个特征向量。而特征数值的常见计算方式有两种，分别是：CountVectorizer和TfidfVectorizer。对于每一条训练文本，CountVectorizer只考虑每种词汇(Term)在该条训练文本中出现的频率(Term Frequency)。而TfidfVectorizer除了考量某一词汇在当前文本中出现的频率(Term Frequency)之外，同时关注包含这个词汇的文本条数的倒数(Inverse Document Frequency)。相比之下，训练文本的条目越多，TfidfVectorizer这种特征量化方式就更有优势。因为我们计算词频(Term Frequency)的目的在于找出对所在文本的含义更有贡献的重要词汇。然而，如果一个词汇几乎在每篇文本中出现，说明这是一个常用词汇，反而不会帮助模型对文本的分类；在训练文本量较多的时候，利用TfidfVectorizer压制这些常用词汇的对分类决策的干扰，往往可以起到提升模型性能的作用。

我们通常称这些在每条文本中都出现的常用词汇为停用词(Stop Words)，如英文中的the、a等。这些停用词在文本特征抽取中经常以黑名单的方式过滤掉，并且用来提高模型的性能表现。下面的代码让我们重新对“20类新闻文本分类”问题进行分析处理，这一次的重点在于列举上述两种文本特征量化模型的使用方法，并比较他们的性能差异。

代码56：使用CountVectorizer并且不去掉停用词的条件下，对文本特征进行量化的朴素贝叶斯分类性能测试

```
>>>#从 sklearn.datasets 里导入 20 类新闻文本数据抓取器。
>>>from sklearn.datasets import fetch_20newsgroups
>>>#从互联网上即时下载新闻样本，subset='all'参数代表下载全部近 2 万条文本存储在变
```

```
量 news 中。
>>>news=fetch_20newsgroups(subset='all')

>>>#从 sklearn.cross_validation 导入 train_test_split 模块用于分割数据集。
>>>from sklearn.cross_validation import train_test_split
>>>#对 news 中的数据 data 进行分割,25%的文本用作测试集;75%作为训练集。
>>>X_train, X_test, y_train, y_test=train_test_split(news.data, news.target,
test_size=0.25, random_state=33)

>>>#从 sklearn.feature_extraction.text 里导入 CountVectorizer
>>>from sklearn.feature_extraction.text import CountVectorizer
>>>#采用默认的配置对 CountVectorizer 进行初始化(默认配置不去除英文停用词),并且赋
值给变量 count_vec。
>>>count_vec=CountVectorizer()

>>>#只使用词频统计的方式将原始训练和测试文本转化为特征向量。
>>>X_count_train=count_vec.fit_transform(X_train)
>>>X_count_test=count_vec.transform(X_test)

>>>#从 sklearn.naive_bayes 里导入朴素贝叶斯分类器。
>>>from sklearn.naive_bayes import MultinomialNB
>>>#使用默认的配置对分类器进行初始化。
>>>mnb_count=MultinomialNB()
>>>#使用朴素贝叶斯分类器,对 CountVectorizer(不去除停用词)后的训练样本进行参数
学习。
>>>mnb_count.fit(X_count_train, y_train)

>>>#输出模型准确性结果。
> > > print ' The accuracy of classifying 20newsgroups using Naive Bayes
(CountVectorizer without filtering stopwords):', mnb_count.score(X_count_
test, y_test)
>>>#将分类预测的结果存储在变量 y_count_predict 中。
>>>y_count_predict=mnb_count.predict(X_count_test)
>>>#从 sklearn.metrics 导入 classification_report。
>>>from sklearn.metrics import classification_report
>>>#输出更加详细的其他评价分类性能的指标。
>>>print classification_report(y_test, y_count_predict, target_names=news.
target_names)
```

```
The accuracy of classifying 20newsgroups using Naive Bayes (CountVectorizer
without filtering stopwords): 0.839770797963
                          precision    recall  f1-score   support

             alt.atheism       0.86      0.86      0.86       201
           comp.graphics       0.59      0.86      0.70       250
 comp.os.ms-windows.misc       0.89      0.10      0.17       248
comp.sys.ibm.pc.hardware       0.60      0.88      0.72       240
   comp.sys.mac.hardware       0.93      0.78      0.85       242
          comp.windows.x       0.82      0.84      0.83       263
            misc.forsale       0.91      0.70      0.79       257
               rec.autos       0.89      0.89      0.89       238
         rec.motorcycles       0.98      0.92      0.95       276
      rec.sport.baseball       0.98      0.91      0.95       251
        rec.sport.hockey       0.93      0.99      0.96       233
               sci.crypt       0.86      0.98      0.91       238
         sci.electronics       0.85      0.88      0.86       249
                 sci.med       0.92      0.94      0.93       245
               sci.space       0.89      0.96      0.92       221
  soc.religion.christian       0.78      0.96      0.86       232
      talk.politics.guns       0.88      0.96      0.92       251
   talk.politics.mideast       0.90      0.98      0.94       231
      talk.politics.misc       0.79      0.89      0.84       188
      talk.religion.misc       0.93      0.44      0.60       158

              avg / total       0.86      0.84      0.82      4712
```

从上面代码的输出，我们可以知道，使用 CountVectorizer 在不去掉停用词的条件下，对训练和测试文本进行特征量化，并利用默认配置的朴素贝叶斯分类器，在测试文本上可以得到 83.977%的预测准确性。而且，平均精度、召回率和 F1 指标，分别是 0.86、0.84 以及 0.82。

接下来，让我们使用与代码 56 相同的训练和测试数据，在不去掉停用词的条件下利用 TfidfVectorizer 进行特征量化，并且评估模型性能。

代码 57：使用 TfidfVectorizer 并且不去掉停用词的条件下，对文本特征进行量化的朴素贝叶斯分类性能测试

```
>>>#从 sklearn.feature_extraction.text 里分别导入 TfidfVectorizer。
```

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> #采用默认的配置对 TfidfVectorizer 进行初始化(默认配置不去除英文停用词),并且赋值给变量 tfidf_vec。
>>> tfidf_vec=TfidfVectorizer()

>>> #使用 tfidf 的方式,将原始训练和测试文本转化为特征向量。
>>> X_tfidf_train=tfidf_vec.fit_transform(X_train)
>>> X_tfidf_test=tfidf_vec.transform(X_test)

>>> #依然使用默认配置的朴素贝叶斯分类器,在相同的训练和测试数据上,对新的特征量化方式进行性能评估。
>>> mnb_tfidf=MultinomialNB()
>>> mnb_tfidf.fit(X_tfidf_train, y_train)
> > > print ' The accuracy of classifying 20newsgroups with Naive Bayes (TfidfVectorizer without filtering stopwords):', mnb_tfidf.score(X_tfidf_test, y_test)
>>> y_tfidf_predict=mnb_tfidf.predict(X_tfidf_test)
>>> print classification_report(y_test, y_tfidf_predict, target_names=news.target_names)
```

The accuracy of classifying 20newsgroups with Naive Bayes (TfidfVectorizer without filtering stopwords): 0.846349745331

	precision	recall	f1-score	support
alt.atheism	0.84	0.67	0.75	201
comp.graphics	0.85	0.74	0.79	250
comp.os.ms-windows.misc	0.82	0.85	0.83	248
comp.sys.ibm.pc.hardware	0.76	0.88	0.82	240
comp.sys.mac.hardware	0.94	0.84	0.89	242
comp.windows.x	0.96	0.84	0.89	263
misc.forsale	0.93	0.69	0.79	257
rec.autos	0.84	0.92	0.88	238
rec.motorcycles	0.98	0.92	0.95	276
rec.sport.baseball	0.96	0.91	0.94	251
rec.sport.hockey	0.88	0.99	0.93	233
sci.crypt	0.73	0.98	0.83	238
sci.electronics	0.91	0.83	0.87	249

sci.med	0.97	0.92	0.95	245
sci.space	0.89	0.96	0.93	221
soc.religion.christian	0.51	0.97	0.67	232
talk.politics.guns	0.83	0.96	0.89	251
talk.politics.mideast	0.92	0.97	0.95	231
talk.politics.misc	0.98	0.62	0.76	188
talk.religion.misc	0.93	0.16	0.28	158
avg / total	0.87	0.85	0.84	4712

由上述代码的输出结果，可得出结论：在使用 TfidfVectorizer 而不去掉停用词的条件下，对训练和测试文本进行特征量化，并利用默认配置的朴素贝叶斯分类器，在测试文本上可以得到比 CountVectorizer 更加高的预测准确性，即从 83.977%提升到 84.635%。而且，平均精度、召回率和 F1 指标都得到提升，分别是 0.87、0.85 以及 0.84。从而，证明了前面叙述的观点："在训练文本量较多的时候，利用 TfidfVectorizer 压制这些常用词汇的对分类决策的干扰，往往可以起到提升模型性能的作用"。

最后，让我们使用下面的代码继续验证另一个观点："这些停用词(Stop Words)在文本特征抽取中经常以黑名单的方式过滤掉，并且用来提高模型的性能表现"。

代码 58：分别使用 CountVectorizer 与 TfidfVectorizer，并且去掉停用词的条件下，对文本特征进行量化的朴素贝叶斯分类性能测试

```
>>>#继续沿用代码 56 与代码 57 中导入的工具包(在同一份源代码中或者不关闭解释器环境)，分别使用停用词过滤配置初始化 CountVectorizer 与 TfidfVectorizer。
>>>count_filter_vec, tfidf_filter_vec=CountVectorizer(analyzer='word', stop_words='english'), TfidfVectorizer(analyzer='word', stop_words='english')

>>>#使用带有停用词过滤的 CountVectorizer 对训练和测试文本分别进行量化处理。
>>>X_count_filter_train=count_filter_vec.fit_transform(X_train)
>>>X_count_filter_test=count_filter_vec.transform(X_test)

>>>#使用带有停用词过滤的 TfidfVectorizer 对训练和测试文本分别进行量化处理。
>>>X_tfidf_filter_train=tfidf_filter_vec.fit_transform(X_train)
>>>X_tfidf_filter_test=tfidf_filter_vec.transform(X_test)

>>>#初始化默认配置的朴素贝叶斯分类器，并对 CountVectorizer 后的数据进行预测与准确性评估。
```

```
>>>mnb_count_filter=MultinomialNB()
>>>mnb_count_filter.fit(X_count_filter_train, y_train)
>>>print ' The accuracy of classifying 20newsgroups using Naive Bayes
(CountVectorizer by filtering stopwords):', mnb_count_filter.score(X_count_
filter_test, y_test)
>>>y_count_filter_predict=mnb_count_filter.predict(X_count_filter_test)

>>>#初始化另一个默认配置的朴素贝叶斯分类器,并对 TfidfVectorizer 后的数据进行预测
与准确性评估。
>>>mnb_tfidf_filter=MultinomialNB()
>>>mnb_tfidf_filter.fit(X_tfidf_filter_train, y_train)
>>>print ' The accuracy of classifying 20newsgroups with Naive Bayes
(TfidfVectorizer by filtering stopwords):', mnb_tfidf_filter.score(X_tfidf_
filter_test, y_test)
>>>y_tfidf_filter_predict=mnb_tfidf_filter.predict(X_tfidf_filter_test)

>>>#对上述两个模型进行更加详细的性能评估。
>>>from sklearn.metrics import classification_report
>>>print classification_report(y_test, y_count_filter_predict, target_names=
news.target_names)
>>>print classification_report(y_test, y_tfidf_filter_predict, target_names=
news.target_names)
```

```
The accuracy of classifying 20newsgroups using Naive Bayes (CountVectorizer by
filtering stopwords): 0.863752122241
The accuracy of classifying 20newsgroups with Naive Bayes (TfidfVectorizer by
filtering stopwords): 0.882640067912
                          precision    recall  f1-score   support

             alt.atheism       0.85      0.89      0.87       201
           comp.graphics       0.62      0.88      0.73       250
 comp.os.ms-windows.misc       0.93      0.22      0.36       248
comp.sys.ibm.pc.hardware       0.62      0.88      0.73       240
   comp.sys.mac.hardware       0.93      0.85      0.89       242
          comp.windows.x       0.82      0.85      0.84       263
            misc.forsale       0.90      0.79      0.84       257
               rec.autos       0.91      0.91      0.91       238
```

```
                rec.motorcycles       0.98      0.94      0.96       276
             rec.sport.baseball       0.98      0.92      0.95       251
               rec.sport.hockey       0.92      0.99      0.95       233
                      sci.crypt       0.91      0.97      0.93       238
                sci.electronics       0.87      0.89      0.88       249
                        sci.med       0.94      0.95      0.95       245
                      sci.space       0.91      0.96      0.93       221
         soc.religion.christian       0.87      0.94      0.90       232
             talk.politics.guns       0.89      0.96      0.93       251
          talk.politics.mideast       0.95      0.98      0.97       231
             talk.politics.misc       0.84      0.90      0.87       188
             talk.religion.misc       0.91      0.53      0.67       158

                   avg / total       0.88      0.86      0.85      4712

                                precision    recall  f1-score   support

                    alt.atheism       0.86      0.81      0.83       201
                  comp.graphics       0.85      0.81      0.83       250
        comp.os.ms-windows.misc       0.84      0.87      0.86       248
       comp.sys.ibm.pc.hardware       0.78      0.88      0.83       240
          comp.sys.mac.hardware       0.92      0.90      0.91       242
                 comp.windows.x       0.95      0.88      0.91       263
                   misc.forsale       0.90      0.80      0.85       257
                      rec.autos       0.89      0.92      0.90       238
                rec.motorcycles       0.98      0.94      0.96       276
             rec.sport.baseball       0.97      0.93      0.95       251
               rec.sport.hockey       0.88      0.99      0.93       233
                      sci.crypt       0.85      0.98      0.91       238
                sci.electronics       0.93      0.86      0.89       249
                        sci.med       0.96      0.93      0.95       245
                      sci.space       0.90      0.97      0.93       221
         soc.religion.christian       0.70      0.96      0.81       232
             talk.politics.guns       0.84      0.98      0.90       251
          talk.politics.mideast       0.92      0.99      0.95       231
             talk.politics.misc       0.97      0.74      0.84       188
             talk.religion.misc       0.96      0.29      0.45       158

                   avg / total       0.89      0.88      0.88      4712
```

代码 58 的输出依旧证明 TfidfVectorizer 的特征抽取和量化方法更加具备优势；同时，通过与代码 56 和代码 57 的性能比较，我们发现：对停用词进行过滤的文本特征抽取方法，平均要比不过滤停用词的模型综合性能高出 3%～4%。

3.1.1.2　特征筛选

读者在实践了本书的一些数据样例之后，一定对如何有效地利用数据特征有自己的心得体会。总体来讲，良好的数据特征组合不需太多，便可以使得模型的性能表现突出。比如，我们在第 1 章的“良/恶性乳腺癌肿瘤预测”问题中，仅仅使用两个描述肿瘤形态的特征便可以取得很高的识别率。冗余的特征虽然不会影响到模型的性能，不过却使得 CPU 的计算做了无用功。比如，主成分分析主要用于去除多余的那些线性相关的特征组合，原因在于这些冗余的特征组合并不会对模型训练有更多贡献。而不良的特征自然会降低模型的精度。

特征筛选与 PCA 这类通过选择主成分对特征进行重建的方法略有区别：对于 PCA 而言，我们经常无法解释重建之后的特征；但是特征筛选不存在对特征值的修改，而更加侧重于寻找那些对模型的性能提升较大的少量特征。

这里我们在代码 59 中继续沿用 Titanic 数据集，这次试图通过特征筛选来寻找最佳的特征组合，并且达到提高预测准确性的目标。

代码 59：使用 Titanic 数据集，通过特征筛选的方法一步步提升决策树的预测性能

```
>>>#导入 pandas 并且更名为 pd。
>>>import pandas as pd
>>>#从互联网读取 titanic 数据。
>>> titanic = pd.read_csv ('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/
DataSets/titanic.txt')

>>>#分离数据特征与预测目标。
>>>y=titanic['survived']
>>>X=titanic.drop(['row.names', 'name', 'survived'], axis=1)

>>>#对对缺失数据进行填充。
>>>X['age'].fillna(X['age'].mean(), inplace=True)
>>>X.fillna('UNKNOWN', inplace=True)

>>>#分割数据，依然采样 25%用于测试。
>>>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)

>>>#类别型特征向量化。
>>>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'))

>>>#输出处理后特征向量的维度。
>>>print len(vec.feature_names_)
474

>>>#使用决策树模型依靠所有特征进行预测,并作性能评估。
>>>from sklearn.tree import DecisionTreeClassifier
>>>dt=DecisionTreeClassifier(criterion='entropy')
>>>dt.fit(X_train, y_train)
>>>dt.score(X_test, y_test)
0.81762917933130697

>>>#从 sklearn 导入特征筛选器。
>>>from sklearn import feature_selection
>>>#筛选前 20%的特征,使用相同配置的决策树模型进行预测,并且评估性能。
>>>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)
0.82370820668693012

>>>#通过交叉验证(下一节将详细介绍)的方法,按照固定间隔的百分比筛选特征,并作图展示
性能随特征筛选比例的变化。
>>>from sklearn.cross_validation import cross_val_score
>>>import numpy as np
```

```
>>>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

>>>#找到提现最佳性能的特征筛选的百分比。
>>>opt=np.where(results ==results.max())[0]
>>>print 'Optimal number of features %d' %percentiles[opt]
```

```
[ 0.85063904  0.85673057  0.87501546  0.88622964  0.86692435  0.86693465
  0.86690373  0.87100598  0.87097506  0.86996496  0.87200577  0.86995465
  0.86997526  0.86183261  0.86690373  0.858792    0.86386312  0.8648423
  0.86283241  0.86286333  0.86384251  0.86384251  0.86895485  0.86488353
  0.86386312  0.86895485  0.86995465  0.87199546  0.86489384  0.86892393
  0.87302618  0.86589363  0.87504638  0.86791383  0.86993403  0.86589363
  0.86590394  0.87404659  0.86487322  0.86895485  0.87301587  0.86285302
  0.8608122   0.86286333  0.86590394  0.86589363  0.86287363  0.8597918
  0.8608122   0.86284271]
Optimal number of features 7
```

```
>>>import pylab as pl
>>>pl.plot(percentiles, results)
>>>pl.xlabel('percentiles of features')
>>>pl.ylabel('accuracy')
>>>pl.show()

>>>#使用最佳筛选后的特征,利用相同配置的模型在测试集上进行性能评估。
>>>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)
0.8571428571428571
```

通过代码59中的几个关键输出，我们可以总结如下：

（1）经过初步的特征处理后，最终的训练与测试数据均有474个维度的特征；

（2）如果直接使用全部474个维度的特征用于训练决策树模型进行分类预测，那么模型在测试集上的准确性约为81.76%；

（3）如果筛选前20%维度的特征，在相同的模型配置下进行预测，那么在测试集上表现的准确性约为82.37%；

（4）如果我们按照固定的间隔采用不同百分比的特征进行训练与测试，那么如图3-1所示，通过3.1.3.2交叉验证得出的准确性有着很大的波动，并且最好的模型性能表现在选取前7%维度的特征的时候；

（5）如果使用前7%维度的特征，那么最终决策树模型可以在该分类预测任务的测试集上表现出85.71%的准确性，比起最初使用全部特征的模型性能高出接近4个百分点。

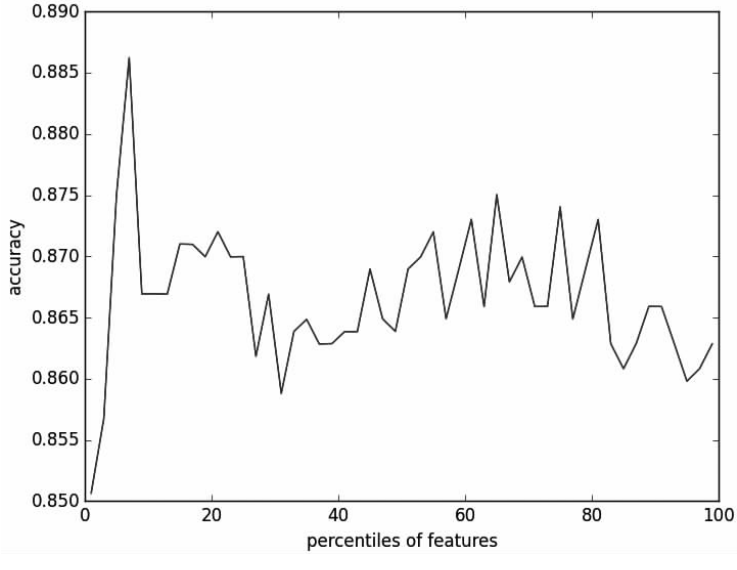

图3-1　代码59中，模型交叉验证的准确性随特征筛选百分比的变化曲线

3.1.2　模型正则化

本书一直在向读者们申明一个重要观点：任何机器学习模型在训练集上的性能表现，都不能作为其对未知测试数据预测能力的评估。并且，我们从最开始 1.1 机器学习综述中就向大家强调过要重视模型的泛化力(Generalization)，只是没有过多地展开讨论。这一节，我们将详细解释什么是模型的泛化力，以及如何保证模型的泛化力。3.1.2.1 欠拟合与过拟合将首先阐述模型复杂度与泛化力的关系；紧接着，3.1.2.2 L_1 范数正则化与 3.1.2.3 L_2 范数正则化将分别介绍如何使用这两种正则化(Regularization)的方式来加强模型的泛化力，避免模型参数过拟合(Overfitting)。

3.1.2.1　欠拟合与过拟合

所谓拟合，是指机器学习模型在训练的过程中，通过更新参数，使得模型不断契合可观测数据(训练集)的过程。本节，我们将使用一个“比萨饼价格预测”的例子来说明。如表 3-1 所示，美国一家比萨饼店出售不同尺寸的比萨，其中每种直径(Diameter)都对应一个报价。我们所要做的，就是设计一个学习模型，可以有效地根据表 3-2 中比萨的直径特征来预测售价。

表 3-1　美国某比萨饼店已知训练数据

Training Instance	*Diameter (in inches)*	*Price(in U. S. dollars)*
1	6	7
2	8	9
3	10	13
4	14	17.5
5	18	18

表 3-2　美国某比萨饼店未知测试数据

Testing Instance	*Diameter (in inches)*	*Price(in U. S. dollars)*
1	6	?
2	8	?
3	11	?
4	16	?

目前我们所知，共有 5 组训练数据、4 组测试数据，并且其中测试数据的比萨报价未知。根据我们的经验，如果只考虑比萨的尺寸与售价的关系，那么使用线性回归模型比较直观，如代码 60 所示。

代码 60：使用线性回归模型在比萨训练样本上进行拟合

```
>>>#输入训练样本的特征以及目标值，分别存储在变量 X_train 与 y_train 之中。
>>>X_train=[[6], [8], [10], [14], [18]]
>>>y_train=[[7], [9], [13], [17.5], [18]]

>>>#从 sklearn.linear_model 中导入 LinearRegression。
>>>from sklearn.linear_model import LinearRegression
>>>#使用默认配置初始化线性回归模型。
>>>regressor=LinearRegression()
>>>#直接以比萨的直径作为特征训练模型。
>>>regressor.fit(X_train, y_train)

>>>#导入 numpy 并且重命名为 np。
>>>import numpy as np
>>>#在 x 轴上从 0 至 25 均匀采样 100 个数据点。
>>>xx=np.linspace(0, 26, 100)
>>>xx=xx.reshape(xx.shape[0], 1)
>>>#以上述 100 个数据点作为基准，预测回归直线。
>>>yy=regressor.predict(xx)

>>>#对回归预测到的直线进行作图。
>>>import matplotlib.pyplot as plt
>>>plt.scatter(X_train, y_train)

>>>plt1,=plt.plot(xx, yy, label="Degree=1")

>>>plt.axis([0, 25, 0, 25])
>>>plt.xlabel('Diameter of Pizza')
>>>plt.ylabel('Price of Pizza')
>>>plt.legend(handles=[plt1])
>>>plt.show()

>>>#输出线性回归模型在训练样本上的 R-squared 值。
```

```
>>> print 'The R-squared value of Linear Regressor performing on the training data is', regressor.score(X_train, y_train)
The R-squared value of Linear Regressor performing on the training data is 0.910001596424
```

根据代码 60 所输出的图 3-2，以及当前模型在训练集上的表现（R-squared 值为 0.9100），我们进一步猜测，也许比萨饼的面积与售价的线性关系[①]更加明显。因此，我们试图将原特征升高一个维度，使用（2 次）多项式回归（Polynominal Regression）对训练样本进行拟合，继续如代码 61 所示。

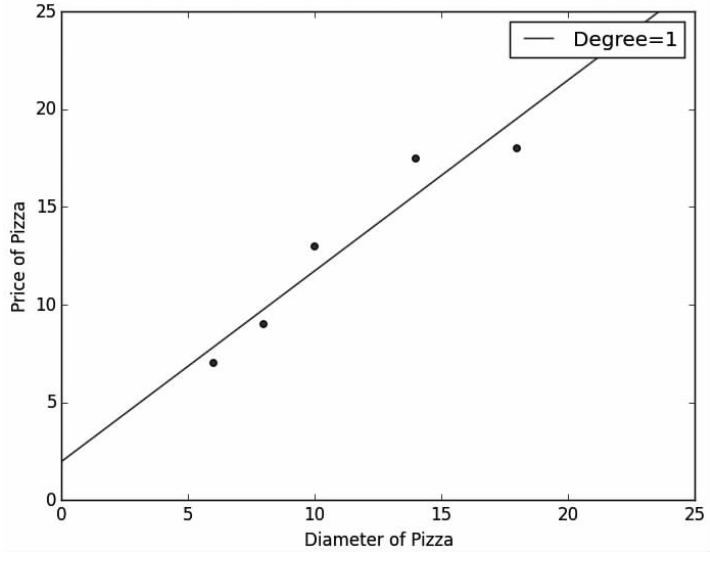

图 3-2　线性回归模型在比萨训练样本上的拟合情况（见彩图）

代码 61：使用 2 次多项式回归模型在比萨训练样本上进行拟合

```
>>> #从 sklearn.preproessing 中导入多项式特征产生器
>>> from sklearn.preprocessing import PolynomialFeatures
>>> #使用 PolynominalFeatures(degree=2)映射出 2 次多项式特征，存储在变量 X_train_poly2 中。
```

① 拓展小贴士 23：尽管我们在代码 61 中依然使用线性回归器作为模型基础，但是由于我们将特征上升到多项式层面，因此通常我们称这类模型为多项式回归（Polynomial Regression）。

```
>>>poly2=PolynomialFeatures(degree=2)
>>>X_train_poly2=poly2.fit_transform(X_train)

>>>#以线性回归器为基础,初始化回归模型。尽管特征的维度有提升,但是模型基础仍然是线
性模型。
>>>regressor_poly2=LinearRegression()

>>>#对 2 次多项式回归模型进行训练。
>>>regressor_poly2.fit(X_train_poly2, y_train)

>>>#从新映射绘图用 x 轴采样数据。
>>>xx_poly2=poly2.transform(xx)

>>>#使用 2 次多项式回归模型对应 x 轴采样数据进行回归预测。
>>>yy_poly2=regressor_poly2.predict(xx_poly2)

>>>#分别对训练数据点、线性回归直线、2 次多项式回归曲线进行作图。
>>>plt.scatter(X_train, y_train)

>>>plt1,=plt.plot(xx, yy, label='Degree=1')
>>>plt2,=plt.plot(xx, yy_poly2, label='Degree=2')

>>>plt.axis([0, 25, 0, 25])
>>>plt.xlabel('Diameter of Pizza')
>>>plt.ylabel('Price of Pizza')
>>>plt.legend(handles=[plt1, plt2])
>>>plt.show()

>>>#输出 2 次多项式回归模型在训练样本上的 R-squared 值。
>>>print 'The R-squared value of Polynominal Regressor (Degree=2) performing on
the training data is', regressor_poly2.score(X_train_poly2, y_train)
```

```
The R-squared value of Polynominal Regressor (Degree=2) performing on the
training data is 0.98164216396
```

果然,在升高了特征维度之后,2 次多项式回归模型在训练样本上的性能表现更加突出,R-squared 值从 0.910 上升到 0.982。并且根据代码 61 所输出的图 3-3 所示,2 次多项式回归曲线(绿色)比起线性回归直线(蓝色),对训练数据的拟合程度也增加了许多。

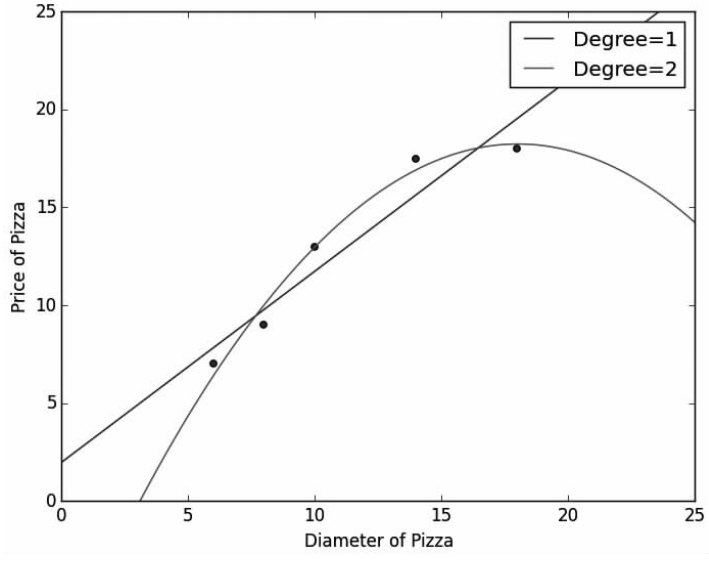

图 3-3　2 次多项式回归与线性回归模型在比萨训练样本上的拟合情况比较(见彩图)

由此,我们更加大胆地进一步升高特征维度,如代码 62 所示,增加到 4 次多项式。

代码 62:使用 4 次多项式回归模型在比萨训练样本上进行拟合

```
>>>#从 sklearn.preprocessing 导入多项式特征生成器。
>>>from sklearn.preprocessing import PolynomialFeatures
>>>#初始化 4 次多项式特征生成器。
>>>poly4=PolynomialFeatures(degree=4)
>>>X_train_poly4=poly4.fit_transform(X_train)

>>>#使用默认配置初始化 4 次多项式回归器。
>>>regressor_poly4=LinearRegression()
>>>#对 4 次多项式回归模型进行训练。
>>>regressor_poly4.fit(X_train_poly4, y_train)

>>>#从新映射绘图用 x 轴采样数据。
>>>xx_poly4=poly4.transform(xx)
>>>#使用 4 次多项式回归模型对应 x 轴采样数据进行回归预测。
>>>yy_poly4=regressor_poly4.predict(xx_poly4)
```

```
>>>#分别对训练数据点、线性回归直线、2次多项式以及4次多项式回归曲线进行作图。
>>>plt.scatter(X_train, y_train)
>>>plt1,=plt.plot(xx, yy, label='Degree=1')
>>>plt2,=plt.plot(xx, yy_poly2, label='Degree=2')

>>>plt4,=plt.plot(xx, yy_poly4, label='Degree=4')
>>>plt.axis([0, 25, 0, 25])
>>>plt.xlabel('Diameter of Pizza')
>>>plt.ylabel('Price of Pizza')
>>>plt.legend(handles=[plt1, plt2, plt4])
>>>plt.show()

>>>print 'The R-squared value of Polynominal Regressor (Degree=4) performing on
the training data is',regressor_poly4.score(X_train_poly4, y_train)
```

```
The R-squared value of Polynominal Regressor (Degree=4) performing on the
training data is 1.0
```

如图 3-4 所示，4 次多项式曲线几乎完全拟合了所有的训练数据点，对应的 R-squared 值也为 1.0。但是，如果这时觉得已经找到了完美的模型，那么显然是高兴过早了。

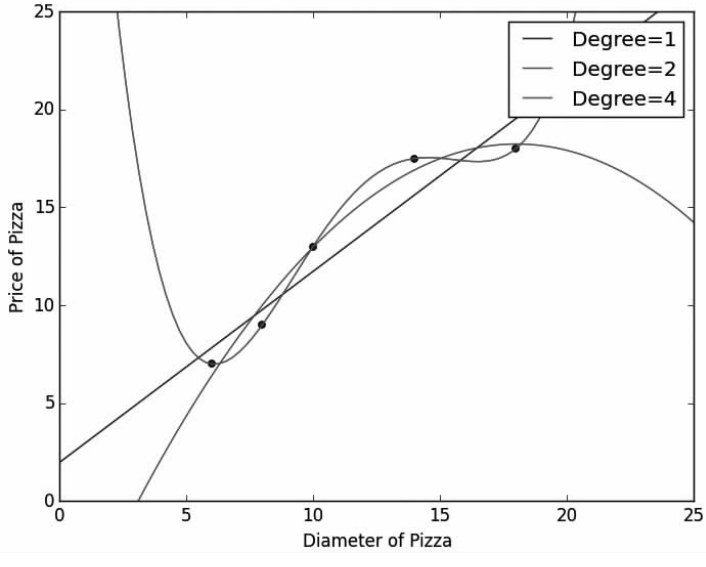

图 3-4　4 次多项式回归与其他模型在比萨训练样本上的拟合情况比较

表 3-3 揭示了测试比萨的真实价格。

表 3-3　美国某比萨饼店真实测试数据

Testing Instance	*Diameter (in inches)*	*Price(in U. S. dollars)*
1	6	8
2	8	12
3	11	15
4	16	18

代码 63：评估 3 种回归模型在测试数据集上的性能表现

```
>>>#准备测试数据。
>>>X_test=[[6], [8], [11], [16]]
>>>y_test=[[8], [12], [15], [18]]

>>>#使用测试数据对线性回归模型的性能进行评估。
>>>regressor.score(X_test, y_test)
0.80972683246686095

>>>#使用测试数据对 2 次多项式回归模型的性能进行评估。
>>>X_test_poly2=poly2.transform(X_test)
>>>regressor_poly2.score(X_test_poly2, y_test)
0.86754436563450543

>>>#使用测试数据对 4 次多项式回归模型的性能进行评估。
>>>X_test_poly4=poly4.transform(X_test)
>>>regressor_poly4.score(X_test_poly4, y_test)
0.8095880795781909
```

如果我们使用代码 63 评估上述 3 种模型在测试集上的表现，并将输出对比之前在训练数据上的拟合情况，制成表 3-4；最终的结果却令人咋舌：当模型复杂度很低(Degree＝1)时，模型不仅没有对训练集上的数据有良好的拟合状态，而且在测试集上也表现平平，这种情况叫做欠拟合(Underfitting)；但是，当我们一味追求很高的模型复杂度(Degree＝4)，尽管模型几乎完全拟合了所有的训练数据，但如图 3-4 所示，模型也变得非常波动，几乎丧失了对未知数据的预测能力，这种情况叫做过拟合(Overfitting)。这两种情况都是缺乏模型泛化力的表现。