

机器学习工程师纳米学位

监督学习

项目 2: 搭建一个学生干预系统

欢迎来到机器学习工程师纳米学位的第二个项目！在此文件中，有些示例代码已经提供给你，但你还需要实现更多的功能让项目成功运行。除非有明确要求，你无须修改任何已给出的代码。以'练习'开始的标题表示接下来的代码部分中有你必须要实现的功能。每一部分都会有详细的指导，需要实现的部分也会在注释中以'**TODO**'标出。请仔细阅读所有的提示！

除了实现代码外，你还必须回答一些与项目和你的实现有关的问题。每一个需要你回答的问题都会以'**问题 X**'为标题。请仔细阅读每个问题，并且在问题后的'回答'文字框中写出完整的答案。我们将根据你对问题的回答和撰写代码所实现的功能来对你提交的项目进行评分。

提示：Code 和 Markdown 区域可通过 **Shift + Enter** 快捷键运行。此外，Markdown可以通过双击进入编辑模式。

问题 1 - 分类 vs. 回归

在这个项目中你的任务是找出那些如果不给予帮助，最重可能无法毕业的学生。你觉得这个问题是哪种类型的监督学习问题，是分类问题还是回归问题？为什么？

答案:

是分类问题

原因是分类问题还是回归问题主要判断依据是输出值是否是连续的，也就是说判断是定性问题还是定量问题

如果是连续输出就是回归问题，离散输出就是分类问题

根据上述，判断学士是否不能毕业是离散输出，是一个定性问题，也就是说是分类问题。

分析数据

运行下面区域的代码以载入学生数据集，以及一些此项目所需的Python库。注意数据集的最后一列 '**passed**' 是我们的预测的目标（表示学生是毕业了还是没有毕业），其他的列是每个学生的属性。

In [13]:

```
# 载入所需要的库
import numpy as np
import pandas as pd
from time import time
from sklearn.metrics import f1_score

# 载入学生数据集
student_data = pd.read_csv("student-data.csv")
# print student_data.shape
# print student_data.head()
# print student_data.columns
print student_data[student_data.passed == "yes"].shape[0]
print student_data[student_data.passed == "no"].shape[0]

print student_data[student_data.passed == "yes"].shape[0] /
student_data.shape[0]
print "Student data read successfully!"
```

265

130

0

Student data read successfully!

练习: 分析数据

我们首先通过调查数据，以确定有多少学生的信息，并了解这些学生的毕业率。在下面的代码单元中，你需要完成如下的运算：

- 学生的总数， `n_students`。
- 每个学生的特征总数， `n_features`。
- 毕业的学生的数量， `n_passed`。
- 未毕业的学生的数量， `n_failed`。
- 班级的毕业率， `grad_rate`， 用百分数表示(%)。

In [14]:

```
# TODO: 计算学生的数量
n_students = student_data.shape[0]

# TODO: 计算特征数量
n_features = student_data.shape[1] - 1

# TODO: 计算通过的学生数
n_passed = student_data[student_data.passed == "yes"].shape[0]

# TODO: 计算未通过的学生数
n_failed = student_data[student_data.passed == "no"].shape[0]

# TODO: 计算通过率
grad_rate = n_passed * 100.0 / n_students

# 输出结果
print "Total number of students: {}".format(n_students)
print "Number of features: {}".format(n_features)
print "Number of students who passed: {}".format(n_passed)
print "Number of students who failed: {}".format(n_failed)
print "Graduation rate of the class: {:.2f}%".format(grad_rate)
```

```
Total number of students: 395
Number of features: 30
Number of students who passed: 265
Number of students who failed: 130
Graduation rate of the class: 67.09%
```

数据准备

在这个部分中，我们将要为建模、训练和测试准备数据

识别特征和目标列

你获取的数据中通常都会包含一些非数字的特征，这会导致一些问题，因为大多数的机器学习算法都会期望输入数字特征进行计算。

运行下面的代码单元将学生数据分成特征和目标列看一看他们中是否有非数字特征。

In [15]:

```
# 提取特征列
feature_cols = list(student_data.columns[:-1])

# 提取目标列 'passed'
target_col = student_data.columns[-1]

# 显示列的列表
print "Feature columns:\n{}".format(feature_cols)
print "\nTarget column: {}".format(target_col)

# 将数据分割成特征数据和目标数据 (即x_all 和 y_all)
X_all = student_data[feature_cols]
y_all = student_data[target_col]

# 通过打印前5行显示特征信息
print "\nFeature values:"
print X_all.head()
```

Feature columns:

```
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
```

Target column: passed

Feature values:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob
0	GP	F	18	U	GT3	A	4	4	at_home	teacher
1	GP	F	17	U	GT3	T	1	1	at_home	other
2	GP	F	15	U	LE3	T	1	1	at_home	other
3	GP	F	15	U	GT3	T	4	2	health	services
4	GP	F	16	U	GT3	T	3	3	other	other

	...	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc
0	...	yes	no	no	4	3	4	1	
1	3								
1	...	yes	yes	no	5	3	3	1	
1	3								
2	...	yes	yes	no	4	3	2	2	
3	3								
3	...	yes	yes	yes	3	2	2	1	
1	5								
4	...	yes	no	no	4	3	2	1	
2	5								

	absences
0	6
1	4
2	10
3	2
4	4

[5 rows x 30 columns]

预处理特征列

正如你所见，我们这里有几个非数值的列需要做一定的转换！它们中很多是简单的yes/no，比如internet。这些可以合理地转化为1/0（二元值，binary）值。

其他的列，如Mjob和Fjob，有两个以上的值，被称为分类变量（*categorical variables*）。处理这样的列的推荐方法是创建和可能值一样多的列（如：Fjob_teacher, Fjob_other, Fjob_services等），然后将其中一个的值设为1另外的设为0。

这些创建的列有时候叫做 虚拟变量（*dummy variables*），我们将用

`pandas.get_dummies()`

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies

函数来完成这个转换。运行下面代码单元的代码来完成这里讨论的预处理步骤。

In [16]:

```
def preprocess_features(X):
    ''' 预处理学生数据，将非数字的二元特征转化成二元值（0或1），将分类的变量转换成虚拟变量
    '''

    # 初始化一个用于输出的DataFrame
    output = pd.DataFrame(index = X.index)

    # 查看数据的每一个特征列
    for col, col_data in X.iteritems():

        # 如果数据是非数字类型，将所有的yes/no替换成1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])

        # 如果数据类型是类别的 (categorical)，将它转换成虚拟变量
        if col_data.dtype == object:
            # 例子: 'school' => 'school_GP' and 'school_MS'
            col_data = pd.get_dummies(col_data, prefix = col)

        # 收集转换后的列
        output = output.join(col_data)

    return output

X_all = preprocess_features(X_all)
print "Processed feature columns ({} total features):\n{}".format(len(X_all.columns), list(X_all.columns))
```

Processed feature columns (48 total features):

```
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason_home', 'reason_other', 'reason_reputation', 'guardian_father', 'guardian_mother', 'guardian_other', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
```

实现: 将数据分成训练集和测试集

现在我们已经将所有的 分类的 (*categorical*) 特征转换成数值了。下一步我们将把数据 (包括特征和对应的标签数据) 分割成训练集和测试集。在下面的代码单元中, 你需要完成下列功能:

- 随机混洗 (shuffle) 切分数据(`x_all, y_all`) 为训练子集和测试子集。
 - 使用300个数据点作为训练集 (约75%), 使用95个数据点作为测试集 (约25%)。
 - 如果可能的话, 为你使用的函数设置一个`random_state`。
 - 将结果存储在`x_train, x_test, y_train`和`y_test`中。

In [17]:

```
# TODO: 在这里导入你可能需要使用的另外的功能
from sklearn.cross_validation import train_test_split

# TODO: 设置训练集的数量
num_train = 300

# TODO: 设置测试集的数量
num_test = X_all.shape[0] - num_train

# TODO: 把数据集混洗和分割成上面定义的训练集和测试集
X_train = None
X_test = None
y_train = None
y_test = None

X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=num_test, random_state=42)

# 显示分割的结果
print "Training set has {} samples.".format(X_train.shape[0])
print "Testing set has {} samples.".format(X_test.shape[0])
```

Training set has 300 samples.

Testing set has 95 samples.

训练和评价模型

在这个部分, 你将选择3个适合这个问题并且在`scikit-learn`中已有的监督学习的模型。首先你需要说明你选择这三个模型的原因, 包括这些数据集有哪些特点, 每个模型的优点和缺点各是什么。然后, 你需要将这些模型用不同大小的训练集 (100个数据点, 200个数据点, 300个数据点) 进行训练, 并用 F_1 的值来衡量。你需要制作三个表, 每个表要显示训练集大小, 训练时间, 预测时间, 训练集上的 F_1 值和测试集上的 F_1 值 (每个模型一个表)。

这是目前 `scikit-learn` (http://scikit-learn.org/stable/supervised_learning.html) 里有的监督学习模型, 你可以从中选择:

- Gaussian Naive Bayes (GaussianNB) 朴素贝叶斯
- Decision Trees 决策树
- Ensemble Methods (Bagging, AdaBoost, Random Forest, Gradient Boosting)
- K-Nearest Neighbors (KNeighbors)
- Stochastic Gradient Descent (SGDC)
- Support Vector Machines (SVM) 向量模型机
- Logistic Regression 逻辑回归

问题 2 - 应用模型

列出三个适合这个问题的监督学习算法模型。每一个你选择的模型：

- 描述一个该模型在真实世界的一个应用场景。（你需要为此做点研究，并给出你的引用出处）
- 这个模型的优势是什么？他什么情况下表现最好？
- 这个模型的缺点是什么？什么条件下它表现很差？
- 根据我们当前数据集的特点，为什么这个模型适合这个问题。

回答:

我选择Decision Trees/AdaBoost/Random Forest

Decision Trees 决策树

- 场景：应用场景很多，例如之前判断房价，再比如很多股票选股票选择市盈率高，高薪企业、近五年主营业务保持50%高速增长来判断股票是否可以长期持有还是不能长期持有
- 优势：
 - 计算简单，易于理解，可解释性强
 - 比较适合处理有缺失属性的样本
 - 能够处理不相关的特征
 - 在相对短的时间内能够对大型数据源做出可行且效果良好的结果
- 劣势
 - 容易发生过拟合
 - 忽略了数据之间的相关性
 - 对于那些各类别样本数量不一致的数据，在决策树当中,信息增益的结果偏向于那些具有更多数值的特征
 - 模型的特征易于理解，可解释性强，可以较好地根据特征进行分类，决策树比较符合这些特点

AdaBoost

- 场景：应用于根据球员数据和球队相关数据判断比赛胜负，
- 优势
 - 速度快
 - 精度比较高
 - 可以使用各种子方法构建子分类器，整体构造简单
 - 不用做特征筛选
- 劣势
 - 对孤立点敏感
 - 对均匀噪音效果很差
 - 过于复杂或者过于弱的弱分类器容易发生过拟合
- 本模型有30个特征，不同特征间有一定的联系，adaboost不用做特征筛选，不需要了解哪些特征会比较重要的问题，学生数据是实际采集的，特异点较少，所以可以选择adaboost

Random Forest（随机森林）

- 场景：应用于全基因组数据表达和选择上用的比较多
- 优势
 - 不用做特征筛选，能够处理高维度的数据
 - 训练速度快，可以并行处理
 - 对于不平衡数据集，可以平衡误差
 - 训练完成后可以给出哪些特征比较重要
- 劣势
 - 某些噪音较大的分类或回归问题上会过拟合
 - 取值划分较多的属性会对随机森林产生更大的影响
- 本模型和Adaboost类似，有30个特征，不同特征间有一定的联系，Random Forest不用做特征筛选，不需要了解哪些特征会比较重要的问题，学生数据是实际采集的，特异点较少，所以可以选择Random Forest进行尝试

准备

运行下面的代码单元以初始化三个帮助函数，这三个函数将能够帮你训练和测试你上面所选择的三个监督学习算法。这些函数是：

- `train_classifier` - 输入一个分类器和训练集，用数据来训练这个分类器。
- `predict_labels` - 输入一个训练好的分类器、特征以及一个目标标签，这个函数将帮你做预测并给出 F_1 的值。
- `train_predict` - 输入一个分类器以及训练集和测试集，它可以运行`train_classifier`和`predict_labels`.
 - 这个函数将分别输出训练集的 F_1 值和测试集的 F_1 值

In [18]:

```
def train_classifier(clf, X_train, y_train):
    ''' 用训练集训练分类器 '''

    # 开始计时，训练分类器，然后停止计时
    start = time()
    clf.fit(X_train, y_train)
    end = time()

    # Print the results
    print "Trained model in {:.4f} seconds".format(end - start)

def predict_labels(clf, features, target):
    ''' 用训练好的分类器做预测并输出F1值 '''

    # 开始计时，作出预测，然后停止计时
    start = time()
    y_pred = clf.predict(features)
    end = time()

    # 输出并返回结果
    print "Made predictions in {:.4f} seconds.".format(end - start)
    return f1_score(target.values, y_pred, pos_label='yes')

def train_predict(clf, X_train, y_train, X_test, y_test):
    ''' 用一个分类器训练和预测，并输出F1值 '''

    # 输出分类器名称和训练集大小
    print "Training a {} using a training set size of {}".format(clf.__class__.__name__, len(X_train))

    # 训练一个分类器
    train_classifier(clf, X_train, y_train)

    # 输出训练和测试的预测结果
    print "F1 score for training set: {}".format(predict_labels(clf, X_train, y_train))
    print "F1 score for test set: {}".format(predict_labels(clf, X_test, y_test))
```

练习: 模型评价指标

借助于上面定义的函数，你现在需要导入三个你选择的监督学习模型，然后为每一个模型运行`train_predict`函数。请记住，对于每一个模型你需要在不同大小的训练集（100，200和300）上进行训练和测试。所以，你在下面应该会有9个不同的输出（每个模型都有训练集大小不同的三个输出）。在接下来的代码单元中，你将需要实现以下功能：

- 引入三个你在上面讨论过的监督式学习算法模型。
- 初始化三个模型并将它们存储在`clf_A`，`clf_B`和`clf_C`中。
 - 如果可能对每一个模型都设置一个`random_state`。
 - **注意:** 这里先使用每一个模型的默认参数，在接下来的部分中你将需要对某一个模型的参数进行调整。
- 创建不同大小的训练集用来训练每一个模型。
 - *不要再混洗和再分割数据！新的训练集要取自`x_train`和`y_train`。*
- 对于每一个模型要用不同大小的训练集来训练它，然后在测试集上做测试（总共需要9次训练测试）
注意: 在下面的代码单元后面我们提供了三个表用来存储你的结果。

In [46]:

```
# TODO: 从sklearn中引入三个监督学习模型
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier

# TODO: 初始化三个模型
clf_A = tree.DecisionTreeClassifier()
clf_B = AdaBoostClassifier()
clf_C = RandomForestClassifier()

# TODO: 设置训练集大小
X_train_100 = X_train.head(100)
y_train_100 = y_train.head(100)

X_train_200 = X_train.head(200)
y_train_200 = y_train.head(200)

X_train_300 = X_train
y_train_300 = y_train

# TODO: 对每一个分类器和每一个训练集大小运行'train_predict'
train_predict(clf_A, X_train_100, y_train_100, X_test, y_test)
train_predict(clf_A, X_train_200, y_train_200, X_test, y_test)
train_predict(clf_A, X_train_300, y_train_300, X_test, y_test)

train_predict(clf_B, X_train_100, y_train_100, X_test, y_test)
train_predict(clf_B, X_train_200, y_train_200, X_test, y_test)
train_predict(clf_B, X_train_300, y_train_300, X_test, y_test)

train_predict(clf_C, X_train_100, y_train_100, X_test, y_test)
train_predict(clf_C, X_train_200, y_train_200, X_test, y_test)
train_predict(clf_C, X_train_300, y_train_300, X_test, y_test)
```

Training a DecisionTreeClassifier using a training set size of 100.

. .

Trained model in 0.0011 seconds

Made predictions in 0.0003 seconds.

F1 score for training set: 1.0000.

Made predictions in 0.0004 seconds.

F1 score for test set: 0.6435.

Training a DecisionTreeClassifier using a training set size of 200.

. .

Trained model in 0.0013 seconds

Made predictions in 0.0003 seconds.

F1 score for training set: 1.0000.

Made predictions in 0.0003 seconds.

F1 score for test set: 0.7481.

Training a DecisionTreeClassifier using a training set size of 300.

. .

Trained model in 0.0024 seconds

Made predictions in 0.0003 seconds.

F1 score for training set: 1.0000.

Made predictions in 0.0004 seconds.

F1 score for test set: 0.6504.

Training a AdaBoostClassifier using a training set size of 100. . .

Trained model in 0.1336 seconds

Made predictions in 0.0054 seconds.

F1 score for training set: 0.9481.

Made predictions in 0.0055 seconds.

F1 score for test set: 0.7669.

Training a AdaBoostClassifier using a training set size of 200. . .

Trained model in 0.1524 seconds

Made predictions in 0.0062 seconds.

F1 score for training set: 0.8927.

Made predictions in 0.0054 seconds.

F1 score for test set: 0.8281.

Training a AdaBoostClassifier using a training set size of 300. . .

Trained model in 0.1291 seconds

Made predictions in 0.0089 seconds.

F1 score for training set: 0.8637.

Made predictions in 0.0055 seconds.

F1 score for test set: 0.7820.

Training a RandomForestClassifier using a training set size of 100.

. .

Trained model in 0.0346 seconds

Made predictions in 0.0012 seconds.

F1 score for training set: 0.9844.

Made predictions in 0.0010 seconds.

F1 score for test set: 0.7287.

Training a RandomForestClassifier using a training set size of 200.

. .

Trained model in 0.0377 seconds

Made predictions in 0.0015 seconds.

F1 score for training set: 0.9928.

Made predictions in 0.0011 seconds.

F1 score for test set: 0.7727.

Training a RandomForestClassifier using a training set size of 300.

. .

Trained model in 0.0330 seconds

Made predictions in 0.0015 seconds.

F1 score for training set: 0.9927.

Made predictions in 0.0011 seconds.

F1 score for test set: 0.7407.

训练集大小	训练时间	预测时间 (测试)	F1值 (训练)	F1值 (测试)
-------	------	-----------	----------	----------

结果表格

编辑下面的表格看看在[Markdown \(https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables\)](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables)中如何设计一个表格。你需要把上面的结果记录在表格中。

分类器 1 - Decision Trees

训练集大小	训练时间	预测时间 (测试)	F1值 (训练)	F1值 (测试)
100	0.0011	0.0004	1.0000	0.6435
200	0.0013	0.0003	1.0000	0.7481
300	0.0024	0.0004	1.0000	0.6504

分类器 2 - AdaBoost

训练集大小	训练时间	预测时间 (测试)	F1值 (训练)	F1值 (测试)
100	0.1336	0.0055	0.9481	0.7669
200	0.1524	0.0054	0.8927	0.8281
300	0.1291	0.0055	0.8637	0.7820

分类器 3 - Random Forest

训练集大小	训练时间	预测时间 (测试)	F1值 (训练)	F1值 (测试)
100	0.0346	0.0010	0.9844	0.7287
200	0.0377	0.0011	0.9928	0.7727
300	0.0330	0.0011	0.9927	0.7407

选择最佳模型

在最后这一部分中，你将从三个监督学习模型中选择一个用在学生数据上的最佳模型。然后你将在最佳模型上用全部的训练集（`x_train`和`y_train`）运行一个网格搜索算法，在这个过程中，你要至少调整一个参数以提高模型的 F_1 值（相比于没有调参的模型的分值有所提高）。

问题 3 - 选择最佳模型

给予你上面做的实验，用一到两段话，向（学校）监事会解释你将选择哪个模型作为最佳的模型。哪个模型在现有的数据，有限的资源、开支和模型表现综合来看是最好的选择？

回答:

从训练和测试的F1值对上述三种模型进行比较，随着训练数据的增加，AdaBoost测试效果在逐步提升，且所有测试数据的F1是最高的，虽然训练F1值的数据没有Decision Trees高，主要是由于Decision Trees全拟合的原因

从训练时间和测试数据的预测时间来看，耗费时间是三种模型中表现最差的，最耗费资源的，随着训练数据的增加，训练时间和测试时间呈现小幅增高。

综上，在有限资源开支、模型表现综合来看，如果对F1值权重较高的情况来看，Adaboost是较好的选择，如果也考虑资源、开支、耗费时间，Random Forest也不失为一个选择。综合来看，个人偏向于Random Forest分类器。

问题 4 - 用通俗的语言解释模型

用一到两段话，向（学校）监事会用外行也听得懂的话来解释最终模型是如何工作的。你需要解释所选模型的主要特点。例如，这个模型是怎样被训练的，它又是如何做出预测的。避免使用高级的数学或技术术语，不要使用公式或特定的算法名词。

回答:

我们选择的最终模型是采用如下的方式工作的:

首先我们有395个学生的资料，我们将学生资料分成两组，一组300个，另外一组95个。

通过如下方式训练，部分具体细节被简化： 步骤1：从300个学生资料中，我们每次任意的拿出一个学生资料，保存下来然后再把资料放回去，然后再次任意的拿出一个学生资料，保存复制下来，有可能跟上一次拿出的学生资料重复，依次反复重复300次，这样就得到了一个有300个可能重复的学生资料集 步骤2：每个学生都有学生的学校、性别、年龄、每周学习时间...等等相关学生数据资料，记录下来的学生属性一共有30个，我们随机的选取几个属性，然后对学生根据选取的属性进行分类，比如根据性别将学生分成男、女两类，看看这两类别中，无法毕业的学生比例，如果男生和女生中都有无法毕业的学生，则我们再通过其他属性进行分类，知道我们认为已经分类清晰，每一个分类都是无法毕业的学生或者是可以毕业的学生，记录你的分类步骤过程。这样一个新的学生可以同此分类步骤来判断是否可以毕业，或者不能毕业 步骤3：重复步骤1和步骤2，比如10次，就得到了10组分类步骤过程，这个一个模型就建立好了

预测的时候，我们把要预测的学生通过上述11组分类步骤进行11次投票预测为无法毕业的或者是可以顺利毕业的，这样找出投票最多的那次预测作为我们最终的预测结果。这样就完成一次预测。

练习: 模型调参

细调选择的模型的参数。使用网格搜索 (GridSearchCV) 来至少调整模型的重要参数 (至少调整一个), 这个参数至少需给出并尝试3个不同的值。你要使用整个训练集来完成这个过程。在接下来的代码单元中, 你需要实现以下功能:

- 导入 `sklearn.grid_search.GridSearchCV` (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html) 和 `sklearn.metrics.make_scorer` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
- 创建一个对于这个模型你希望调整参数的字典。
 - 例如: `parameters = {'parameter' : [list of values]}`。
- 初始化你选择的分类器, 并将其存储在`clf`中。
- 使用`make_scorer` 创建 F_1 评分函数并将其存储在`f1_scorer`中。
 - 需正确设定参数`pos_label`的值!
- 在分类器`clf`上用`f1_scorer` 作为评价函数运行网格搜索, 并将结果存储在`grid_obj`中。
- 用训练集(`x_train, y_train`)训练grid search object, 并将结果存储在`grid_obj`中。

In [57]:

```
# TODO: 导入 'GridSearchCV' 和 'make_scorer'
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score

# TODO: 创建你希望调整的参数列表
parameters = {'n_estimators': [3, 7, 10, 15],
              'max_depth': [None, 1, 3, 5],
              'random_state': [None, 42, 50]}

# TODO: 初始化分类器
clf = RandomForestClassifier()

# TODO: 用 'make_scorer' 创建一个 f1 评分函数
f1_scorer = make_scorer(f1_score, pos_label='yes')

# TODO: 在分类器上使用 f1_scorer 作为评分函数运行网格搜索
grid_obj = GridSearchCV(clf, param_grid=parameters, scoring=f1_scorer)

# TODO: Fit the grid search object to the training data and find the optimal parameters
# TODO: 用训练集训练 grid search object 来寻找最佳参数
grid_obj = grid_obj.fit(X_train, y_train)

# Get the estimator
# 得到预测的结果
print grid_obj.best_params_
clf = grid_obj.best_estimator_

# Report the final F1 score for training and testing after parameter tuning
# 输出经过调参之后的训练集和测试集的 F1 值
print "Tuned model has a training F1 score of {:.4f}.".format(predict_labels(clf, X_train, y_train))
print "Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf, X_test, y_test))

{'n_estimators': 15, 'random_state': None, 'max_depth': 3}
Made predictions in 0.0029 seconds.
Tuned model has a training F1 score of 0.8249.
Made predictions in 0.0018 seconds.
Tuned model has a testing F1 score of 0.7742.
```

问题 5 - 最终的 F_1 值

最终模型的训练和测试的 F_1 值是多少？这个值相比于没有调整过参数的模型怎么样？

回答:

最终模型的训练和测试的 F_1 的值分别是 0.8529 和 0.8056。

这个值相对于没有调整过参数的模型训练和测试趋于一致。测试 F_1 有所提高，训练 F_1 有所降低

注意: 当你写完了所有的代码，并且回答了所有的问题。你就可以把你的 iPython Notebook 导出成 HTML 文件。你可以在菜单栏，这样导出 **File -> Download as -> HTML (.html)** 把这个 HTML 和这个 iPython notebook 一起做为你的作业提交。