Project 2: Supervised Learning

Building a Student Intervention System

1. Classification vs Regression

Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why?

> The provided data set and question lead to a classification problem in supervised machine learning. The data is categorical (discreet) for the most part and labels are provided. Continues data would work better with a regression, but this is not provided in this case. And labeled data makes clustering unnecessary.

2. Exploring the Data

Let's go ahead and read in the student dataset first.

*To execute a code cell, click inside it and press **Shift+Enter**.*

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
```

In [2]:
```
# Read student data
student_data = pd.read_csv("student-data.csv")
#Read CSV (comma-separated) file into DataFrame
#http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_
csv.html
print "Student data read successfully!"
# Note: The last column 'passed' is the target/label, all other are
feature columns
```

```
Student data read successfully!
```

Now, can you find out the following facts about the dataset?

- Total number of students
- Number of students who passed
- Number of students who failed
- Graduation rate of the class (%)
- Number of features

*Use the code block below to compute these values. Instructions/steps are marked using **TODO**s.*

In [3]:
```
# TODO: Compute desired values - replace each '?' with an appropria
te expression/function call

n_students = student_data.shape[0]
#http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataF
rame.shape.html#pandas.DataFrame.shape

n_features = student_data.shape[1]-1#lable column
#http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataF
rame.shape.html#pandas.DataFrame.shape

n_passed = (student_data[student_data.passed == 'yes']).shape[0]
#http://www.datacarpentry.org/python-ecology/05-loops-and-functions

n_failed = (student_data[student_data.passed == 'no']).shape[0]
#http://www.datacarpentry.org/python-ecology/05-loops-and-functions

grad_rate = n_passed *100.00 / n_students

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

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

3. Preparing the Data

In this section, we will prepare the data for modeling, training and testing.

Identify feature and target columns

It is often the case that the data you obtain contains non-numeric features. This can be a problem, as most machine learning algorithms expect numeric data to perform computations with.

Let's first separate our data into feature and target columns, and see if any features are non-numeric.
Note: For this dataset, the last column (`'passed'`) is the target or label we are trying to predict.

In [4]:
```
# Extract feature (X) and target (y) columns
feature_cols = list(student_data.columns[:-1])  # all columns but l
ast are features
target_col = student_data.columns[-1]  # last column is the target/
label
print "Feature column(s):-\n{}".format(feature_cols)
print "Target column: {}".format(target_col)

X_all = student_data[feature_cols]  # feature values for all studen
ts
y_all = student_data[target_col]  # corresponding targets/labels
print "\nFeature values:-"
print X_all.head()  # print the first 5 rows
```

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu',
'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'study
time', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'n
ursery', '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   t
eacher
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  se
rvices
4     GP   F   16       U     GT3       T     3     3    other
other

    ...    higher internet  romantic  famrel  freetime goout Dalc
Walc health  \
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]
```

Preprocess feature columns

As you can see, there are several non-numeric columns that need to be converted! Many of them are simply `yes`/`no`, e.g. `internet`. These can be reasonably converted into `1`/`0` (binary) values.

Other columns, like `Mjob` and `Fjob`, have more than two values, and are known as *categorical variables*. The recommended way to handle such a column is to create as many columns as possible values (e.g. `Fjob_teacher`, `Fjob_other`, `Fjob_services`, etc.), and assign a `1` to one of them and `0` to all others.

These generated columns are sometimes called *dummy variables*, and we will use the `pandas.get_dummies()` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies) function to perform this transformation.

In [5]:
```
# Preprocess feature columns
def preprocess_features(X):
    outX = pd.DataFrame(index=X.index)  # output dataframe, initial
ly empty

    # Check each column
    for col, col_data in X.iteritems():
        # If data type is non-numeric, try to replace all yes/no va
lues with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])
        # Note: This should change the data type for yes/no columns
to int

        # If still non-numeric, convert to one or more dummy variab
les
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix=col)  # e.g.
'school' => 'school_GP', 'school_MS'

        outX = outX.join(col_data)  # collect column(s) in output d
ataframe

    return outX

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

```
Processed feature columns (48):-
['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', 'Mj
ob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_health', 'Fjob
_other', 'Fjob_services', 'Fjob_teacher', 'reason_course', 'reason
_home', 'reason_other', 'reason_reputation', 'guardian_father', 'g
uardian_mother', 'guardian_other', 'traveltime', 'studytime', 'fai
lures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'h
igher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Da
lc', 'Walc', 'health', 'absences']
```

Split data into training and test sets

So far, we have converted all *categorical* features into numeric values. In this next step, we split the data (both features and corresponding labels) into training and test sets.

In [6]:
```
# First, decide how many training vs test samples you want
num_all = student_data.shape[0]  # same as len(student_data)
num_train = 300  # about 75% of the data
num_test = num_all - num_train

# TODO: Then, select features (X) and corresponding labels (y) for
the training and test sets
# Note: Shuffle the data or randomly select samples to avoid any bi
as due to ordering in the dataset

from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, t
est_size=(float(num_test)/num_all), random_state=1)

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
e=(num_test/num_all))
#source: http://scikit-learn.org/stable/modules/generated/sklearn.c
ross_validation.train_test_split.html


X_train = X_train
X_test = X_test [:95]
y_train = y_train
y_test = y_test [:95]

print "Training set: {} samples".format(X_train.shape[0])
print "Test set: {} samples".format(X_test.shape[0])
# Note: If you need a validation set, extract it from within traini
ng data
```

```
Training set: 300 samples
Test set: 95 samples
```

4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

- What are the general applications of this model? What are its strengths and weaknesses?

knn

The general application of nearest neighbors is instance-based learning i.e. non-generalizing learning.

pros:

- As a non-generalizing classifier knn can be successful even when the decision boundary is very blurred
- Nearest neighbors classifiers are lazy learners. They are fast at leaning time by prolonging the computational expensive tasks to prediction time
- The classifier is easy to use, because it works without normalizing the data and it can handle missing data by default

cons:

- You need domain knowledge to pic a number of neighbors that works for the specific problem
- Depending on the choice for k, the classifier can be susceptible to noise. At low k values it is sensitive to outliers and irrelevant attributes (overfitting)
- Due to not creating a general internal model the classifier is computational expensive at prediction time, because it has to work over the whole training set. There is no "inexpensive" parametric model to use

sources :http://scikit-learn.org/stable/modules/neighbors.html#classification (http://scikit-learn.org/stable/modules/neighbors.html#classification), http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeigh (http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeigh https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5396240646/m-672718835 (https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5396240646/m-672718835)

decision tree

The general application of decision trees is non-parametric supervised learning.

pros

- Decision trees are easily understood by humans, compared with other classifiers
- The calssifier is modest concerning data preparation. No normalization is needed
- Overfitting can be handled by pruning
- Decision trees are able to process categorical data

cons

- Without pruning decision trees tend to be over-complex, which leads to overfitting.
- Optimal decision trees are computational expensive to calculate, therefore locally optimal algorithms are used to create nods
- XOR is hard to express in a decision tree
- Decision trees do not work well with skewed data sets

sources: http://scikit-learn.org/stable/modules/tree.html (http://scikit-learn.org/stable/modules/tree.html), https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5414400946/m-313175595 (https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5414400946/m-313175595)

SVM

The general application of SVMs is parametric supervised learning.

pros

- SVMs handle high dimensional spaces very well.
- They are effective even when the number of dimensions exceeds the number of samples
- They perform well with data, where there is a clear separation
- The classifier can be tuned by different kernel functions to perform well on specific problems

cons

- On large data sets a long training time is required
- They do not work well with very noisy data
- Probability estimates cannot be derived directly from a SVM

sources: [http://scikit-learn.org/stable/modules/svm.html#classification (http://scikit-learn.org/stable/modules/svm.html#classification)](http://scikit-learn.org/stable/modules/svm.html#classification), [https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5447009165/m-2384188710 (https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5447009165/m-2384188710)](https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5447009165/m-2384188710)

- Given what you know about the data so far, why did you choose this model to apply?

> Knn was chosen to investigate, if an instance based learning algorithm might perform better on the data set, than a generalizing algorithm. Decision tree was chosen, because it delivers the best humanly understandable results. SVM/SVC was chosen because it is the most versatile algorithm for classification problems.

- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F_1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

Produce a table showing training time, prediction time, F_1 score on training set and F_1 score on test set, for each training set size.

Note: You need to produce 3 such tables - one for each model.

KNN \ traning set n=	**300**	**200**	**100**
training time (sec.)	0.002	0.001	0.001
prediction time (sec.)	0.012	0.007	0.005
F1 training set	0.856	0.834	0.788
F1 test set	0.768	0.797	0.773

DT \ traning set n=	**300**	**200**	**100**
training time (sec.)	0.004	0.002	0.001
prediction time (sec.)	0.000	0.000	0.000
F1 training set	1.000	1.000	1.000
F1 test set	0.677	0.777	0.700

SVC \ traning set n=	**300**	**200**	**100**
training time (sec.)	0.011	0.006	0.002
prediction time (sec.)	0.009	0.005	0.002
F1 training set	0.858	0.858	0.859
F1 test set	0.846	0.841	0.833

In [7]:
```
# Train a model
# KNN
import time

def train_classifier(clf, X_train, y_train):
    print "Training {}...".format(clf.__class__.__name__)
    start = time.time()
    clf.fit(X_train, y_train)
    end = time.time()
    print "Done!\nTraining time (secs): {:.3f}".format(end - start)

# TODO: Choose a model, import it and instantiate an object
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=5)
#http://scikit-learn.org/stable/modules/generated/sklearn.neighbors
.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

# Fit model to training data
print "Training set size: {}".format(len(X_train))
train_classifier(clf, X_train, y_train)  # note: using entire train
ing set here
#print clf  # you can inspect the learned model by printing it
```

```
Training set size: 300
Training KNeighborsClassifier...
Done!
Training time (secs): 0.006
```

In [8]:
```
# Predict on training set and compute F1 score
from sklearn.metrics import f1_score

def predict_labels(clf, features, target):
    print "Predicting labels using {}...".format(clf.__class__.__na
me__)
    start = time.time()
    y_pred = clf.predict(features)
    end = time.time()
    print "Done!\nPrediction time (secs): {:.3f}".format(end - star
t)
    return f1_score(target.values, y_pred, pos_label='yes')

train_f1_score = predict_labels(clf, X_train, y_train)
print "F1 score for training set: {:.3f}".format(train_f1_score)
```

```
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.016
F1 score for training set: 0.856
```

In [9]:
```
# Predict on test data
print "F1 score for test set: {:.3f}".format(predict_labels(clf, X_
test, y_test))
```

```
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.007
F1 score for test set: 0.768
```

In [10]:
```
# Train and predict using different training set sizes
def train_predict(clf, X_train, y_train, X_test, y_test):
    print "------------------------------------------"
    print "Training set size: {}".format(len(X_train))
    train_classifier(clf, X_train, y_train)
    print "F1 score for training set: {:.3f}".format(predict_labels
(clf, X_train, y_train))
    print "F1 score for test set: {:.3f}".format(predict_labels(clf
, X_test, y_test))

# TODO: Run the helper function above for desired subsets of traini
ng data

# Predict for treining set n=200
X_train_200 = X_train[:200]
y_train_200 = y_train[:200]

train_predict(clf, X_train_200, y_train_200, X_test, y_test)

# Predict for treining set n=100
X_train_100 = X_train[:100]
y_train_100 = y_train[:100]

train_predict(clf, X_train_100, y_train_100, X_test, y_test)

# Note: Keep the test set constant
```

```
------------------------------------------
Training set size: 200
Training KNeighborsClassifier...
Done!
Training time (secs): 0.002
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.834
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.004
F1 score for test set: 0.797
------------------------------------------
Training set size: 100
Training KNeighborsClassifier...
Done!
Training time (secs): 0.002
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.004
F1 score for training set: 0.788
Predicting labels using KNeighborsClassifier...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.773
```

In [11]:

```
# TODO: Train and predict using two other models
```

In [12]:

```
#DECISION TREE
#http://scikit-learn.org/stable/modules/generated/sklearn.tree.Deci
sionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

# TODO: Choose a model, import it and instantiate an object
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Train and predict using different training set sizes
train_predict(clf, X_train, y_train, X_test, y_test)
train_predict(clf, X_train_200, y_train_200, X_test, y_test)
train_predict(clf, X_train_100, y_train_100, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.005
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 1.000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.721
------------------------------------------
Training set size: 200
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 1.000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.765
------------------------------------------
Training set size: 100
Training DecisionTreeClassifier...
Done!
Training time (secs): 0.001
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for training set: 1.000
Predicting labels using DecisionTreeClassifier...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.650
```

In [13]:
```
#SVM/SVC
#http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.h
tml#sklearn.svm.SVC

# TODO: Choose a model, import it and instantiate an object
from sklearn.svm import SVC
clf = SVC()

# Train and predict using different training set sizes
train_predict(clf, X_train, y_train, X_test, y_test)
train_predict(clf, X_train_200, y_train_200, X_test, y_test)
train_predict(clf, X_train_100, y_train_100, X_test, y_test)
```

```
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.011
Predicting labels using SVC...
Done!
Prediction time (secs): 0.006
F1 score for training set: 0.858
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.846
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.004
Predicting labels using SVC...
Done!
Prediction time (secs): 0.003
F1 score for training set: 0.858
Predicting labels using SVC...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.841
------------------------------------------
Training set size: 100
Training SVC...
Done!
Training time (secs): 0.002
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.859
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.833
```

5. Choosing the Best Model

- Based on the experiments you performed earlier, in 1-2 paragraphs explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?

> SVC is the most appropriate model because it performed best on the test set with as little as 100 samples. Prediction time is comparable with knn. Training time is the longest of all used algorithms, but not dramatically long. As expected from a generalizing algorithm the training time is longer than the prediction time. The data must not be kept, which lowers memory cost compared to an instance based algorithm like knn.

- In 1-2 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a Decision Tree or Support Vector Machine, how does it make a prediction).

A support vector machine tries to separate the data by a line (or hyperplane) into a group of successful students and a group of unsuccessful students. The classifier does this in a way that maximizes the distance from the line to the nearest points of each group, as shown in the diagram.

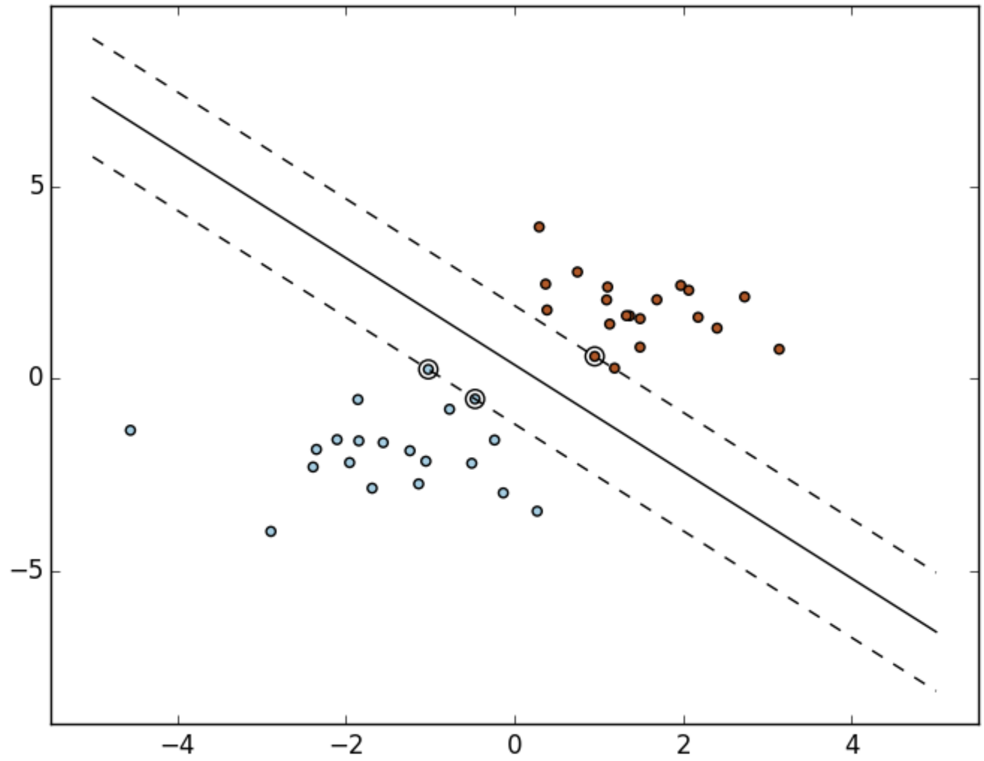

Image of separating hyperplane

Because there often are no clear decision boundaries, SVM use kernel functions. These functions create new features by recombining existing ones. Technically the separation problem is lifted into a higher dimensional space. This "trick" makes a successful separation more likely. More so by the right kernel function very complex decision boundaries can be realized on a data set. When you provide the data for a new student, the SVM can predict the group the student most likely belongs to. The SVM uses the decision boundary found with the additional features provided by the kernel function for this prediction.

sources: [http://scikit-learn.org/stable/modules/svm.html#classification (http://scikit-learn.org/stable/modules/svm.html#classification)](http://scikit-learn.org/stable/modules/svm.html#classification), [http://scikit-learn.org/stable/_images/plot_separating_hyperplane_001.png (http://scikit-learn.org/stable/_images/plot_separating_hyperplane_001.png)](http://scikit-learn.org/stable/_images/plot_separating_hyperplane_001.png), [https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5447009165/m-2384188710 (https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5447009165/m-2384188710)](https://www.udacity.com/course/viewer#!/c-ud726-nd/l-5447009165/m-2384188710)

- Fine-tune the model. Use Gridsearch with at least one important parameter tuned and with at least 3 settings. Use the entire training set for this.

- What is the model's final F_1 score?

> F1: 0.848

In [14]:
```
# TODO: Fine-tune your model and report the best F1 score
```

In [15]:
```
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import f1_score, make_scorer

# Parameters
# the range of C and gamma are adjusted in a way that the best para
meter result is not a corner case
parameters =  [{'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.0
1, 0.001], 'kernel': ['rbf']},]

# Classifier
clf = GridSearchCV(SVC (C=1), param_grid = parameters, scoring = 'f
1') #SUB2: added the F1 scoring here

# Preparing lables for F1 scorer
y_train_1_0 = y_train
if y_train_1_0.dtype == object:
    y_train_1_0 = y_train_1_0.replace(['yes', 'no'], [1, 0])

y_test_1_0 = y_test
if y_test_1_0.dtype == object:
    y_test_1_0 = y_test_1_0.replace(['yes', 'no'], [1, 0])

# Fitting
clf.fit(X_train, y_train_1_0)

# Best Hyper-Parameters
print '\n' "Best parameter from grid search: " + str(clf.best_param
s_) +'\n'

# Predict and calculate F1 on TRAINING set
print "F1 score for training set: {:.3f}".format(f1_score(y_train_1
_0, clf.best_estimator_.predict(X_train) ))

# Predict and calculate F1 on TEST set
print "F1 score for test set: {:.3f}".format(f1_score(y_test_1_0, c
lf.best_estimator_.predict(X_test)))
```

```
Best parameter from grid search: {'kernel': 'rbf', 'C': 1, 'gamma'
: 0.1}

F1 score for training set: 0.975
F1 score for test set: 0.848
```

In [16]:
```
#Scikit-learn ref:

#@article{scikit-learn,
# title={Scikit-learn: Machine Learning in {P}ython},
# author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Mich
el, V.
#         and Thirion, B. and Grisel, O. and Blondel, M. and Prette
nhofer, P.
#         and Weiss, R. and Dubourg, V. and Vanderplas, J. and Pass
os, A. and
#         Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesn
ay, E.},
# journal={Journal of Machine Learning Research},
# volume={12},
# pages={2825--2830},
# year={2011}
#}
```