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 problem of identifying students who might need early intervention is a classification problem.

The reason is that our aim is to predict the need (or absence of the need) of early intervention, which is a discrete target. Problems with discrete targets are classification problems, whereas ones with continuous targets are regression problems.

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")
print("Student data read successfully!")
# Note: The last column 'passed' is the target/label, all other are feature co
```

```
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]:

```
# DONE / TODO: Compute desired values - replace each '?' with an appropriate e
n_students = student_data.shape[0]
n_features = student_data.shape[1] - 1 # subtract one for target
n_passed = np.ma.count(student_data[student_data.passed == 'yes'].passed)
n_failed = np.ma.count(student_data[student_data.passed == 'no'].passed)
grad_rate = float(n_passed) / n_students * 100
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%
```

The dataset contains 30 features for 395 students. The graduation rate is 67.09%. All looks good so far with regard to the learning task, since we have a couple of features that may be relevant for forecasting the graduating rate and we observe a sizeable set of students that both pass and fail.

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 last are fea
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 students
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', 'stud
ytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities',
'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetim
e', '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  s
ervices
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, initially empty

    # Check each column
    for col, col_data in X.iteritems():
        # If data type is non-numeric, try to replace all yes/no values with 1
        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 variables
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix=col)  # e.g. 'school' =

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

    return outX

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

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R',
'address_U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatu
s_T', 'Medu', 'Fedu', 'Mjob_at_home', 'Mjob_health', 'Mjob_othe
r', 'Mjob_services', 'Mjob_teacher', 'Fjob_at_home', 'Fjob_healt
h', 'Fjob_other', 'Fjob_services', 'Fjob_teacher', 'reason_cours
e', 'reason_home', 'reason_other', 'reason_reputation', 'guardia
n_father', 'guardian_mother', 'guardian_other', 'traveltime', 'st
udytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activitie
s', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freet
ime', 'goout', 'Dalc', '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

# DONE / TODO: Then, select features (X) and corresponding labels (y) for the
# Note: Shuffle the data or randomly select samples to avoid any bias due to c
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=nu
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 training 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 is the theoretical O(n) time & space complexity in terms of input size?
- What are the general applications of this model? What are its strengths and weaknesses?
- Given what you know about the data so far, why did you choose this model to apply?
- 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.

In [7]:

```
# Train a model
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))

# DONE / TODO: Choose a model, import it and instantiate an object
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()

# Fit model to training data
train_classifier(clf, X_train, y_train)  # note: using entire training set her
#print clf  # you can inspect the learned model by printing it
```

```
Training LogisticRegression...
Done!
Training time (secs): 0.004
```

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__.__name__))
    start = time.time()
    y_pred = clf.predict(features)
    end = time.time()
    print("Done!\nPrediction time (secs): {:.3f}".format(end - start))
    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: {}".format(train_f1_score))
```

```
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.8520971302428257
```

In [9]:

```
# Predict on test data
print("F1 score for test set: {}".format(predict_labels(clf, X_test, y_test)))
```

```
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7737226277372262
```

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: {}".format(predict_labels(clf, X_train,
    print("F1 score for test set: {}".format(predict_labels(clf, X_test, y_tes

# DONE / TODO: Run the helper function above for desired subsets of training d
# Note: Keep the test set constant
train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training LogisticRegression...
Done!
Training time (secs): 0.002
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.8714285714285714
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7559055118110235
------------------------------------------
Training set size: 200
Training LogisticRegression...
Done!
Training time (secs): 0.003
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for training set: 0.8484848484848486
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.000
F1 score for test set: 0.7794117647058824
------------------------------------------
Training set size: 300
Training LogisticRegression...
Done!
Training time (secs): 0.006
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.8520971302428257
Predicting labels using LogisticRegression...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7737226277372262
```

In [11]:

```
# DONE / TODO: Train and predict using two other models
# Predict using SVC
from sklearn.svm import SVC
clf = SVC()

train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
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: 0.840764331210191
Predicting labels using SVC...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.7466666666666666
------------------------------------------
Training set size: 200
Training SVC...
Done!
Training time (secs): 0.009
Predicting labels using SVC...
Done!
Prediction time (secs): 0.007
F1 score for training set: 0.8633540372670807
Predicting labels using SVC...
Done!
Prediction time (secs): 0.011
F1 score for test set: 0.761904761904762
------------------------------------------
Training set size: 300
Training SVC...
Done!
Training time (secs): 0.020
Predicting labels using SVC...
Done!
Prediction time (secs): 0.011
F1 score for training set: 0.869022869022869
Predicting labels using SVC...
Done!
Prediction time (secs): 0.009
F1 score for test set: 0.7534246575342465
```

In [12]:

```
# DONE / TODO: Train and predict using two other models
# Predict using RandomForest Classifier
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=10)

train_predict(clf, X_train[:100], y_train[:100], X_test, y_test)
train_predict(clf, X_train[:200], y_train[:200], X_test, y_test)
train_predict(clf, X_train, y_train, X_test, y_test)
```

```
------------------------------------------
Training set size: 100
Training RandomForestClassifier...
Done!
Training time (secs): 0.010
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.9777777777777777
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for test set: 0.6716417910447761
------------------------------------------
Training set size: 200
Training RandomForestClassifier...
Done!
Training time (secs): 0.034
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.011
F1 score for training set: 0.9855072463768116
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.016
F1 score for test set: 0.7313432835820897
------------------------------------------
Training set size: 300
Training RandomForestClassifier...
Done!
Training time (secs): 0.020
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.001
F1 score for training set: 0.960919540229885
Predicting labels using RandomForestClassifier...
Done!
Prediction time (secs): 0.002
F1 score for test set: 0.7007299270072992
```

Model 1: Logistic regression classifier

[COMPLEXITY] For the logistic regression, both the space and prediction time complexity are constant [O(1)] in the size of the test data and linear [O(d)] in the number of features. The time complexity is constant in the size of the test data, because we just need to plug in the feature vector for prediction and multiply it with the fitted model parameters for each prediction/test (complexity is linear in these parameters/features, but constant in test set size). The space complexity is constant in test size and linear in parameters/features, since we only need to store the model parameters once for all predictions.

[APPLICATIONS/STRENGTH/WEAKNESSES] Logistic regression is a linear (in parameters) model for classification which is widely applicable. It is relatively easy to implement and can serve as a first simple (linear) model for fitting the data. In some cases, a weakness is that the model assumes the log-odds ratio to be linear in the independent variables (which is an issue only if there is a reason to believe this assumption is violated). Moreover, the model assumes independence of the observations, which - depending mostly on sampling - may be an issue.

[WHY CHOOSEN] Given we face a classification task, I wanted to first apply a very simple classification model. This is also motivated by the fact that computing resources are an issue for the client.

Table 1: Logistic regression classifier

Training Size	**100**	**200**	**300**
Training time	0.002	0.002	0.004
Prediction time	0.000	0.000	0.000
F1 score (training)	0.879	0.863	0.837
F1 score (test)	0.758	0.777	0.788

Model 2: Support vector classifier

[COMPLEXITY] The prediction complexity for nonlinear SVC's using an rbf kernel (the default in SKLEARN and used here) is O(m*d), where m is the number of support vectors and d is the feature dimension (see "Fast Prediction with SVM Models Containing RBF Kernels", Marc Claesen, Frank De Smet, Johan A.K. Suykens, Bart De Moor (2014)). For this case, we have around 200 support vectors for both classes and 48 features. With regard to the time complexity in the size of the test set, we again have constant complexity. The space complexity is again linear in the number of support vectors (same as prediction complexity). The reason is that all support vectors need to be stored.

[APPLICATIONS/STRENGTH/WEAKNESSES] The support vector classifier can be applied to many different classification problems and can be tuned to specific problems via parameters such as the kernel, the kernel coefficient (gamma) and the penalty parameter (C). Advantages of this classifier are that it is effective even in high-dimensional feature spaces and quite versatile due to the flexibility in the choice of a kernel function. Although SVC's are versatile due to the kernel choice, it is a priori unclear which kernel is a good choice for a given problem. This can be viewed as a weakness of SVCs. Also, the training time complexity for nonlinear SVC's is generally quadratic in the number of training samples (depending on solver an kernel), which may be a problem in some applications.

[WHY CHOOSEN] After the use of a linear classifier (logistic regression), my goal was to try out a nonlinear classifier and hence I decided to go with a SVC with an rbf-kernel. This classifier has a much larger function space to search over and I was wondering whether more flexibility results in better predictive performance. The results, however, are mixed. The SVC does only marginally better than the logistic regression.

Table 2: Support vector classifier

Training Size	100	200	300
Training time	0.002	0.003	0.008
Prediction time	0.001	0.002	0.005
F1 score (training)	0.868	0.860	0.855
F1 score (test)	0.767	0.779	0.789

Model 3: Random forest classifier

[COMPLEXITY] The prediction complexity for random forrest is linear in the number of trees, features and the depth of the tree [O(n_tree, n_features, depth)]. It is linear in the trees, because we need to average over all trees and hence need to compute each tree at first. It is linear in the features, because we need to compare the features (or a subset of them) at each node of the tree and it is linear in the depth of the tree, as we need to run an operation at each level of depth of the tree. Note that sometimes, we may not need to go down the full depth of the three. A similar arguments holds for the space complexity, since we need to store each tree, where the tree has a given depth and the feature values at each node need to be stored.

[APPLICATIONS/STRENGTH/WEAKNESSES] A random forrest classifier is a meta estimator that fits a number of decision trees and aggregates them. Decision trees are very flexible, nonparametric classifiers and as such widely applicable. A disadvantage of decision trees is that they tend to overfit the data, even though random forrests mitigates this problem by averaging over a set of trees. Moreover, decision trees can be quite sensitive to small variations in the data. On the other hand, decision trees are very flexible tools that require little assumptions. Moreover, there is little need to do data preparation for decision trees.

[WHY CHOOSEN] Having explored two parametric models, I chose random forrests in order to experiment with a very flexible nonparametric model. The goal was to see if a very flexible model generates a good F1 score in the test set. Even though random forrests did very well on the training set (as expected), the performance on the test set was much less convincing.

Table 3: Random forest classifier

Training Size	100	200	300
Training time	0.014	0.012	0.016
Prediction time	0.001	0.002	0.002
F1 score (training)	0.992	0.979	0.979
F1 score (test)	0.778	0.748	0.786

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?

Given the tradeoff between resources and predictive performance, I consider logistic regression to be the preferable model for the task at hand.

The reason is that the predictive performance of logistic regression as measured by the F1-score increases with more training data and is only marginally inferior to the best performing model in terms of predictive accuracy (SVC) for all training set sizes. For example, for a training size of 100 (200/300), logistic regression achieves an F1-score of 0.758 (0.777/0.788) and the SVC scores 0.767 (0.779/0.789). That is, the SVC performs only marginally better than the logistic regression. However, the training time of logistic regression is very short and the prediction time is negligible. In general, logistic regression is never slower and many times faster than the other algorithms, for both training and prediction. For example, with a training size of N=300, logistic regression needs only 0.004 seconds to train, whereas SVM requires 0.008 and random forrest 0.016 seconds. Because of the training complexities of these algorithms, these time gaps will only widen when the size of the training sample is increased further. Moreover, logistic regression is space efficient, since we only need to store the coefficients and can actually discard the data. Finally, the random forrest classifier only performs reasonable with a training size of 100 (measured by F1-score) and does show only slightly better performance (if at all) with a larger training size. Nevertheless predictive performance is always inferior to SVC and logistic regression for training sizes starting from N=200 (including). Also, training time is much worse.

Another point worth mentioning (besides model/algorithm performance) is that the coefficients of a logistic regression are readily interpretable. This can be very helpful to implement policy measures aiming at better student performance, since logistic regression directly reveals factors relevant to performance.

- 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).

The task for making a prediction using logistic regression can be split into two steps. In the first step, the factors important for a student's success or failure are weighted and a simple score is produced. That is, the characteristics of a student such as age, internet and absences are condensed into a single number using the parameters of the model. In the second step, this number is standardized between zero and one and can be viewed as the probability of successful graduation for a particular student. If desired, this standardized number can further be simplyfied by comparing it to a threshold to produce flags for students that need intervention.

- 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?

In [13]:

```
# DONE / TODO: Fine-tune your model and report the best F1 score
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
scorer = make_scorer(f1_score, pos_label='yes', average='binary')

parameters = {'C':[1, 1.2, 1.4, 1.6, 1.8, 2.0]}
logreg = LogisticRegression()
clf = GridSearchCV(logreg, parameters, scoring=scorer)
clf.fit(X_train, y_train)
print("Best estimator: ", clf.best_estimator_)
print("F1 score for test set (tuned model): {}".format(predict_labels(clf, X_t
```

```
Best estimator:  LogisticRegression(C=1.6, class_weight=None, dua
l=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', to
l=0.0001,
          verbose=0)
Predicting labels using GridSearchCV...
Done!
Prediction time (secs): 0.000
F1 score for test set (tuned model): 0.7647058823529412
```

The logistic regression with C=1.4 performs best. This model achieves an F1-score of 0.79411 on the test set, which is higher than the score of the default logistic regression model.