Classification for Prediction

module 11

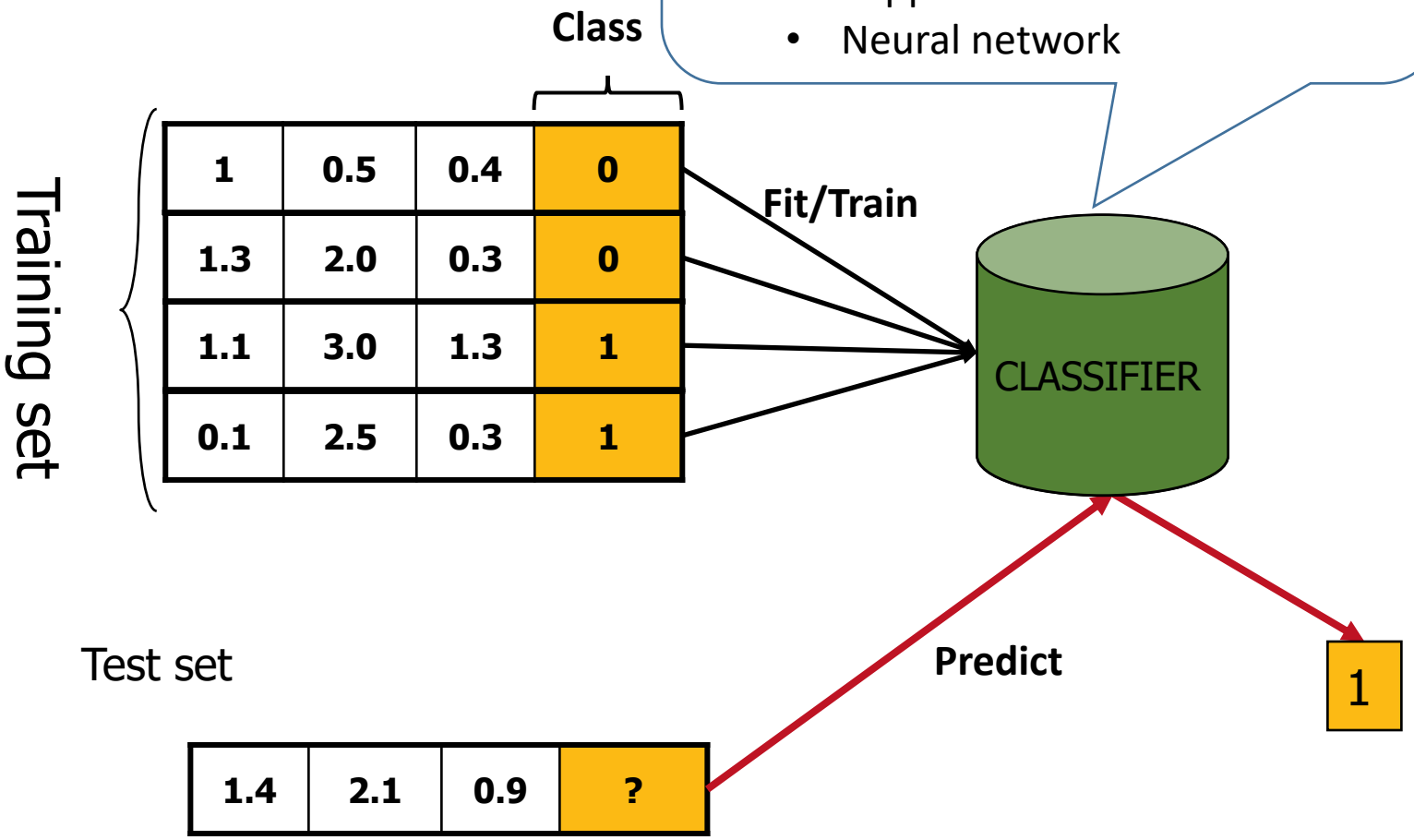
Project Presentations

- Final Project Group Presentation:
- Each group 22 +- 2 mins

- 6/08: Final Exam (Lucas 309, week 10 of class)

Classification for prediction

- Some classifiers are interpretable
  - Decision trees
  - Logistic regression
- Some aren't
  - Support vector machines
  - Neural network



Classification Jargon:

- **Class**: the target attribute that we want to predict
- **Training set**: the table (DataFrame) used to learn
- **Classifier**: the entity that learns the differences between classes
- **Fit/train**: the task of learning
- **Predict**: after training, the task of predicting the class of new objects

Today’s data set

- Affairs.csv:
  - One row = one person
  - Columns:

    - **age, children**: age and number of children
    - **religious**: the person's religiousness
    - **educ**: the person's education level
    - **occupation**: a code that identifies the person's occupation
    - **rate_marriage**: how the person rates his or her marriage,
    - **yrs_married**: length of the marriage, in years
    - **affairs**: time spent, in hours/week, in extra-marital affairs

  - Our goal is to classify cheaters

Data Preparation

Make binary attribute to indicate cheaters

```
df['affairsBin'] = (df.affairs > 0)*1.0
```

Make dummy attribute for occupations

```
df = pd.get_dummies(data= df, columns= ['occupation'])
```

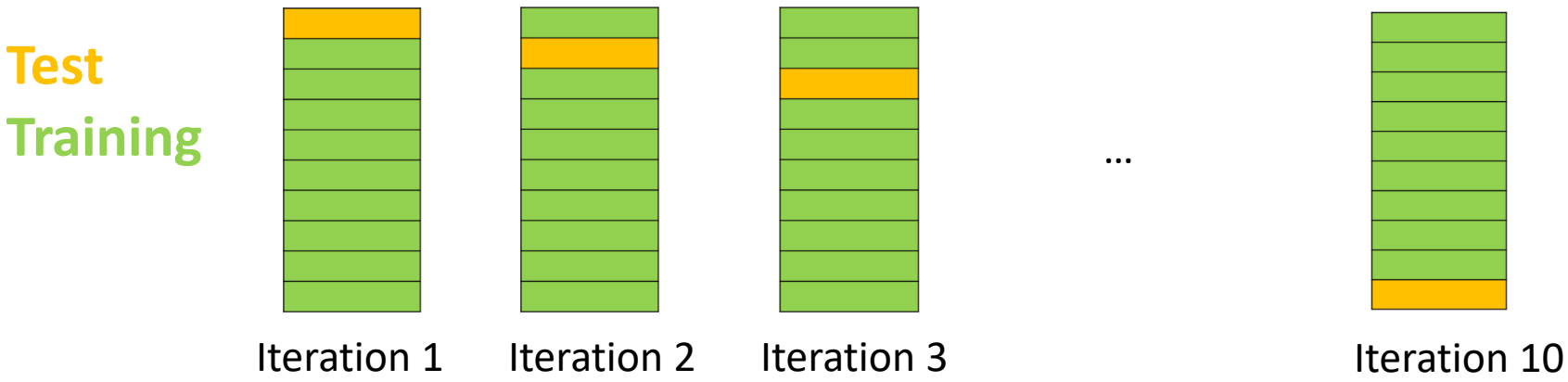
Make X and Y

```
X = df.drop(['affairs','affairsBin'],axis=1)
```

```
Y=df.affairsBin
```

Training and Testing

- Classifiers are trained on a **training set**
- Future predictive performance is <u>always</u> evaluated on a **<u>different</u> test set**
- We will see two methods:
  - Hold-out sample:
    1. split the data into two random partitions
    2. Train the model on one partition
    3. Predict the class of the other partition
  - Cross-validation:
    1. Partition the data set in k parts
    2. For k times, use the union of $k-1$ partition as training set and the partition left out as test set



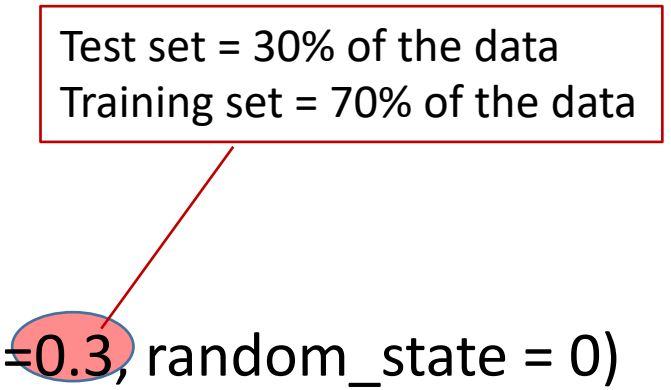
The prediction process

On a Hold-out sample

Step 1: split the data into training and test

Sklearn.model_selection.train_test_split

Splits the data into training set (X_train and Y_train) and test set (X_test and Y_test)




```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = 0)
```

Step 2: train the classifier on the training set

```
from sklearn.ensemble import RandomForestClassifier
cl = RandomForestClassifier(random_state = 0)
cl.fit(X_train,Y_train)
```

Fit on the training set

Here, we use a classification technique called *RandomForest*.
The process is the same for any classification technique

Step 3: predict on the test set

Method ***predict:***
Returns an array of binary predictions
(one for each element of the test set)

Method ***predict_proba:***
Returns a *n*-by-2 matrix of probabilities of belonging to each class.
(i,0) is the probability that element i belongs to class 0
(i,1) is the probability that element i belongs to class 1

```
y_pred = cl.predict(X_test)
y_proba = cl.predict_proba(X_test)
```

```
y_pred[:40]
```

```
array([ 1.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,
        0.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  1.])
```

```
y_proba[:40]
```

```
array([[ 0.3       ,  0.7       ],
       [ 0.48833333,  0.51166667],
       [ 1.        ,  0.        ],
       [ 0.83333333,  0.16666667],
       [ 0.25714286,  0.74285714],
       [ 0.5       ,  0.5       ],
       [ 0.775     ,  0.225     ],
       [ 0.775     ,  0.225     ],
```

Evaluate the classification performance

Input to Evaluate the performance

- Given:
  - y_test: the real class of each element of the test set
  - y_pred: the binary predictions for each element of the test set
  - y_proba: the class-membership probability of each element of the test set

- How to measure the predictive performance

CONFUSION MATRIX

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_pred,Y_test)
```

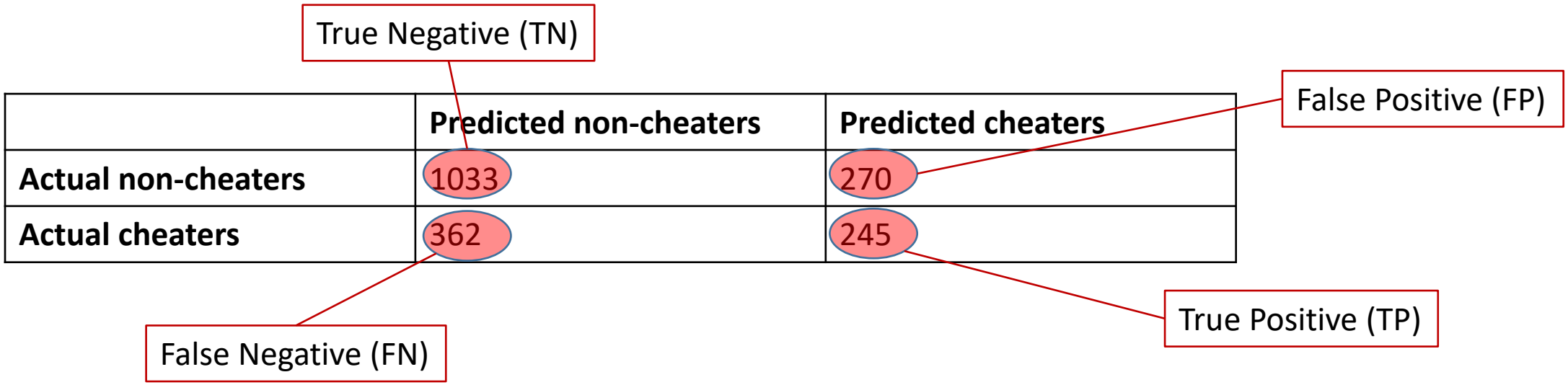
```
array([[1033,  362],
       [ 270,  245]])
```

- **Confusion Matrix**: a matrix of all outcomes on the test set:

	Predicted non-cheaters	**Predicted cheaters**
Actual non-cheaters	1033	270
Actual cheaters	362	245

POSITIVE AND NEGATIVE:
In most binary classification problems, we are interested in detecting one class, which is usually the minority class. That class is called POSITIVE; the other one is negative. In this example, “cheaters” is the POSITIVE class.

True/False Positive/Negative



	Predicted non-cheaters	**Predicted cheaters**
Actual non-cheaters	1033	270
Actual cheaters	362	245

Accuracy

$$\text{Accuracy} = \frac{TN+TP}{n}$$

ACCURACY IS A BAD METRIC FOR IMBALANCED DATA

True Negative (TN)

	Predicted non-cheaters	**Predicted cheaters**
Actual non-cheaters	1033	270
Actual cheaters	362	245

True Positive (TP)

In scikit-learn:

```
sklearn.metrics.accuracy_score(y_test,y_pred)
```

```
0.67015706806282727
```

Precision

Precision = $\frac{TP}{TP+FP}$. Out of the retrieved elements, how many are actually positive?

	Predicted non-cheaters	**Predicted cheaters**
Actual non-cheaters	1033	270
Actual cheaters	362	245

False Positive (FP)

True Positive (TP)

In scikit-learn:

```
sklearn.metrics.precision_score(y_test,y_pred)
```

```
0.50103519668737062
```

Recall

Recall = $\frac{TP}{TP+FN}$. Among the relevant elements, how many did I retrieve?

	Predicted non-cheaters	**Predicted cheaters**
Actual non-cheaters	1033	270
Actual cheaters	362	245

False Negative (FN)

True Positive (TP)

In scikit-learn:

```
sklearn.metrics.recall_score(y_test,y_pred)
```

```
0.38351822503961963
```

If time at the end

AUC score

AUC = Area Under the Curve (i.e., the Receiver Operating Characteristic Curve)

We won't see the details. It measures how good a classifier is to rank the elements from the most likely to the least likely to be positive.

- AUC = 0.50: random prediction
- AUC = 1.00: perfect prediction

This is a commonly used metric for imbalanced data.

In scikit-learn:

```
sklearn.metrics.roc_auc_score(y_test,y_proba[:,1])
```

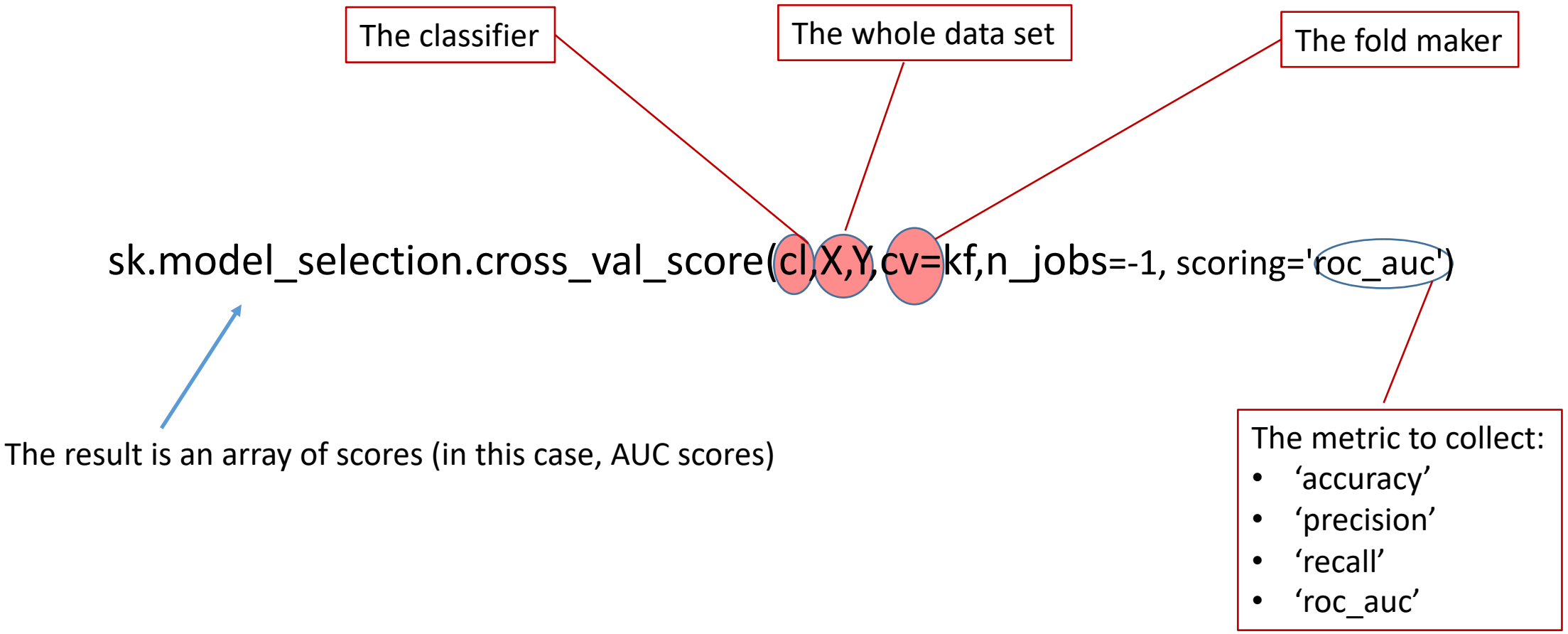
```
0.666869050082461
```

Cross-validation

Step 1: create the fold-maker

```
from sklearn.model_selection import KFold
nfolds = 10
kf = KFold(n_splits=nfolds,random_state=2,shuffle=True)
```

Step 2: Execute the cross-validation



Select the best classifier

Here are some classifiers

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

clfs = [DecisionTreeClassifier(), sk.ensemble.RandomForestClassifier(n_jobs=-1), sk.naive_bayes.GaussianNB(),
        sk.linear_model.LogisticRegression(n_jobs=-1),sk.tree.DecisionTreeClassifier(),sk.ensemble.AdaBoostClassifier(),
        QuadraticDiscriminantAnalysis(),MLPClassifier(),SVC()]
```

Find the best one

- For each classifier:
  - Run a cross validation and record the average AUC
  - Store the information on the classifier that obtains the largest AUC

```
maxAUC = -1
bestCL = ''
for cl in clfs:
    kf = KFold(n_splits=nfolds,random_state=2,shuffle=True)
    auc = sklearn.model_selection.cross_val_score(cl,X,y=Y,cv=kf,scoring='roc_auc').mean()
    if auc > maxAUC:
        bestCL = cl
        maxAUC = auc
print (str(bestCL) +': ' +str(maxAUC))
```

More Practice on Classification

Consider the data set cleaned_survey.csv

1. Exploratory: What is the difference between MSIS and MBA students?
2. Predictive: Can you accurately predict the program of a new unseen student whose program is unknown?