PROJECT SPECIFICATION ¶

Machine Learning Capstone Project

This capstone project involves machine learning modeling and analysis of clinical, demographic, and brain related derived anatomic measures from human MRI (magnetic resonance imaging) tests (http://www.oasis-brains.org/ (http://www.oasis-brains.org/)). The objectives of these measurements are to diagnose the level of Dementia in the individuals and the probability that these individuals may have Alzheimer's Disease (AD).

In published studies, Machine Learning has been applied to Alzheimer's/Dementia identification from MRI scans and related data in the academic papers/theses in References 10 and 11 listed in the References Section below. Recently, a close relative of mine had to undergo a sequence of MRI tests for cognition difficulties.The motivation for choosing this topic for the Capstone project arose from the desire to understand and analyze potential for Dementia and AD from MRI related data. Cognitive testing, clinical assessments and demographic data related to these MRI tests are used in this project. This Capstone project does not use the MRI "imaging" data and does not focus on AD, focusses only on Dementia.

Problem Statement

[The problem which needs to be solved is clearly defined. A strategy for solving the problem, including discussion of the expected solution, has been made.]

• Cross-Sectional and longitudinal OASIS MRI structural and demographic data (clinical, demographic, and brain related derived anatomic measures) from human MRI (magnetic resonance imaging) tests (http://www.oasis-brains.org/ (http://www.oasis-brains.org/)) will be used to train a set of linear and non-linear machine learning classification models.

• **Clinical Dementia Rating (CDR) values provided in the data set will be used as "labels" for training the classification models. [Clinical Dementia Ratings (CDR values: 0=nondemented; 0.5 = very mild dementia; 1 = mild dementia; 2 = moderate dementia)].**

• Pandas will be used for data loading and Python scikit-learn library for modeling.

• The goal is to train machine learning models to predict whether the individuals in the cross-validation set (test set) have dementia (CDR>0), and if they do, the severity level of dementia (CDR values of 0.5, 1, and 2). The problem will be formulated both as a binary classification problem (CDR=0, and CDR>0), and multi-label classification problem (CDR values in the dataset: 0, 0.5, 1, and 2). In the binary classification formulation, the CDR>0 the values in the sliced dataset will be relabeled as CDR=1.

• Classification Accuracy will be used as the primary metric. Additionally, for binary classification AUC/ROC values will be reported. For multi-label classification (multiple CDR labels) F-1 score will be reported. The results from the best model will be reported along with those from the other models.

• About 80% of the data in the dataset will be used for training the models. About 20% of data will used prediction of the CDR label for the k-fold cross-validation with k=10. Sensitivity studies with proportion other than 80:20, e.g. 70:30, will be considered to test sensitivity of this split on the accuracy.

• The base case uses a dataset that combines the cross-sectional and the longitudinal MRI datasets. This has the benefit of having a larger dataset.

• Data cleaning (e.g. removal of NaN values), data exploration, data preparation, data visualization, and data preprocessing used will be described and their impact on appropriate prediction metrics will be discussed.

References

1. The Open Access Series of Imaging Studies (OASIS), http://www.oasis-brains.org/app/template/Index.vm;jsessionid=6926BBF18A3D5CD974E750FAC8ED01CE (http://www.oasis-brains.org/app/template/Index.vm;jsessionid=6926BBF18A3D5CD974E750FAC8ED01CE)
2. OASIS Fact Sheet (rev. 2007-8-20) Cross-Sectional Data Across the Adult Lifespan, Marcus et al., 2007, http://www.oasis-brains.org/pdf/oasis_cross-sectional_facts.pdf (http://www.oasis-brains.org/pdf/oasis_cross-sectional_facts.pdf)
3. MRI Reliability data across the adult lifespan, http://www.oasis- (http://www.oasis-) brains.org/app/action/BundleAction/bundle/OAS1_RELIABILITY
4. Buckner, RL, Head, D, Parker, J, Fotenos, AF, Marcus, D, Morris, JC, Snyder, AZ, 2004, "A unified approach for morphometric and functional data analysis in young, old, and demented adults using automated atlas-based head size normalization: reliability and validation against manual measurement of total intracranial volume", Neuroimage 23, 724-38.
5. Fotenos, AF, Snyder, AZ, Girton, LE, Morris, JC, and Buckner, RL, 2005, "Normative estimates of crosssectional and longitudinal brain volume decline in aging and AD", Neurology, 64: 1032-1039.
6. Marcus, DS, Wang, TH, Parker, J, M, Csernansky, JG, Morris, JC, Buckner, RL, 2007. "Open Access Series of Imaging Studies (OASIS): Cross-Sectional MRI Data in Young, Middle Aged, Nondemented and Demented Older Adults", Journal of Cognitive Neuroscience, 19, 1498-1507.
7. Morris, JC, 1993. "The Clinical Dementia Rating (CDR): current version and scoring rules", Neurology 43, 2412b-2414b.
8. Rubin, EH, Storandt, M, Miller, JP, Kinscherf, DA, Grant, EA, Morris, JC, Berg, L, "A prospective study of cognitive function and onset of dementia in cognitively healthy elders". Arch Neurol. 55, 395- 401.
9. Zhang, Y, Brady, M, Smith, S, "Segmentation of brain MR images through a hidden Markov random field model and the expectation maximization algorithm". IEEE Trans. on Medical Imaging, 20(1):45-57.
10. "Diagnosis of Alzheimer's Disease Based on Structural MRI Images Using a Regularized Extreme Learning Machine and PCA Feature", https://www.hindawi.com/journals/jhe/2017/5485080/ (https://www.hindawi.com/journals/jhe/2017/5485080/)
11. "Use of Machine Learning Technology in the Diagnosis of Alzheimer's Disease", http://doras.dcu.ie/21356/1/Noel_s_Master_s_thesis__Copy_(1).pdf (http://doras.dcu.ie/21356/1/Noel_s_Master_s_thesis__Copy_(1).pdf)
12. Scikit-learn, http://scikit-learn.org/stable/index.html (http://scikit-learn.org/stable/index.html)
13. Model evaluation: quantifying the quality of predictions, http://scikit-learn.org/stable/modules/model_evaluation.html (http://scikit-learn.org/stable/modules/model_evaluation.html)).
14. Precision and Recall, http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py (http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py)
15. Confusion Matrix, http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix (http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix)
16. Support, http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html)

17. F1-score, http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)
18. "Conditional data slicing in a Pandas dataframe", https://stackoverflow.com/questions/17071871/select-rows-from-a-dataframe-based-on-values-in-a-column-in-pandas (https://stackoverflow.com/questions/17071871/select-rows-from-a-dataframe-based-on-values-in-a-column-in-pandas)
19. "Matplotlib colormap examples and color schemes for using in heatmap", http://pyhogs.github.io/colormap-examples.html (http://pyhogs.github.io/colormap-examples.html); https://matplotlib.org/examples/color/colormaps_reference.html (https://matplotlib.org/examples/color/colormaps_reference.html)
20. "Usefulness of data from magnetic resonance imaging to improve prediction of dementia: population based cohort study" http://www.bmj.com/content/350/bmj.h2863 (http://www.bmj.com/content/350/bmj.h2863)
21. C - Statistics: http://www.statisticshowto.com/c-statistic/ (http://www.statisticshowto.com/c-statistic/)
22. The Use of MRI and PET for Clinical Diagnosis of Dementia and Investigation of Cognitive Impairment: A Consensus Report https://www.alz.org/national/documents/imaging_consensus_report.pdf (https://www.alz.org/national/documents/imaging_consensus_report.pdf)
23. Knopman DS, DeKosky ST, Cummings JL, Chui H, Corey-Bloom J, Relkin N, et al. Practice parameter: Diagnosis of dementia (an evidence-based review). Report of the Quality Standards Subcommittee of the American Academy of Neurology. Neurology 2001;56(9):1143–1153.
24. Machine Learning Mastery, https://machinelearningmastery.com/ (https://machinelearningmastery.com/)
25. "The Role of Balanced Training and Testing Data Sets for Binary Classifiers in Bioinformatics", https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3706434/ (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3706434/)
26. "8 Proven Ways for improving the “Accuracy” of a Machine Learning Model", https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/ (https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/)
27. “A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning”, https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/ (https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/)
28. "Gradient Boosting Tree vs. Random Forest", https://stats.stackexchange.com/questions/173390/gradient-boosting-tree-vs-random-forest (https://stats.stackexchange.com/questions/173390/gradient-boosting-tree-vs-random-forest)
29. "Does the dataset size influence a machine learning algorithm?", https://stackoverflow.com/questions/25665017/does-the-dataset-size-influence-a-machine-learning-algorithm?rq=1 (https://stackoverflow.com/questions/25665017/does-the-dataset-size-influence-a-machine-learning-algorithm?rq=1)
30. http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.Sel (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.Sel
31. "Plotting Learning Curves", http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py (http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py)

Metrics

*[Metrics used to measure performance of a model or result are clearly defined. Metrics are justified based on the characteristics of the problem.]

Classification Accuracy is used as the primary metric. This metric is applicable for binary classification where the number of records for the two labels are balanced. As shown later in this notebook, the ratio of the number of processed records with CDR=0 and CDR=1 is 60% to 40%, and is considered balanced. Classification accuracy is defined as the number of records correctly classified divided by the total number of records classified. Additionally, for binary classification here, AUC/ROC values are reported. Accuracy results are also be reported in sklearn Confusion Matrix format to evaluate classifier output quality, and in Classification Report format (provides, precision, recall, and f1-score) which are quite appropriate for the dataset used to train models for CDR classification. References 13 through 17 have details and discussion of these sklearn metrics.

Benchmark

[Student clearly defines a benchmark result or threshold for comparing performances of solutions obtained.\

My primary benchmark is a neural network model (Appendix 1 in the Main Notebook) based on the same data discussed above and as used in this problem: Keras is used as the frontend with tensorflow backend. My secondary benchmark will be results of the study in the two papers below:

Paper title: Usefulness of data from magnetic resonance imaging to improve prediction of dementia: population based cohort study, Reference 20

"Results During 10 years of follow-up, there were 119 confirmed cases of dementia, 84 of which were Alzheimer's disease. The conventional risk model incorporated age, sex, education, cognition, physical function, lifestyle (smoking, alcohol use), health (cardiovascular disease, diabetes, systolic blood pressure), and the apolipoprotein genotype (C statistic for discrimination performance was 0.77, 95% confidence interval 0.71 to 0.82). No significant differences were observed in the discrimination performance of the conventional risk model compared with models incorporating data from MRI including white matter lesion volume (C statistic 0.77, 95% confidence interval 0.72 to 0.82; P=0.48 for difference of C statistics, Reference 21), brain volume (0.77, 0.72 to 0.82; P=0.60), hippocampal volume (0.79, 0.74 to 0.84; P=0.07), or all three variables combined (0.79, 0.75 to 0.84; P=0.05). Inclusion of hippocampal volume or all three MRI variables combined in the conventional model did, however, lead to significant improvement in reclassification measured by using the integrated discrimination improvement index (P=0.03 and P=0.04) and showed increased net benefit in decision curve analysis. Similar results were observed when the outcome was restricted to Alzheimer's disease."

**Paper Title: The Use of MRI and PET for Clinical Diagnosis of Dementia and Investigation of Cognitive Impairment: A Consensus Report, Reference 22.

"Once the presence of dementia has been established, the role of imaging in the diagnosis of dementia subtypes is very much a function of the clinical diagnosis. The accuracy of the clinical diagnosis of Alzheimer's disease (AD) is quite good. Pathological AD has a prevalence of about 70% (range 50% to above 80% depending upon whether the AD occurs in isolation or with other entities) among all dementias (see evidence Table 1 in Reference 23); thus, even clinicians with limited neurological expertise should have a diagnostic accuracy, for AD at least, at about that level. A review of 13 published studies gave average values for sensitivity and specificity of the clinical diagnosis of AD of 81% and 70%, respectively(Reference 23). The overall accuracy of the clinical diagnosis of AD versus not-AD compared with the neuropathological standard based on those values for prevalence, sensitivity, and specificity, is 78%. "

Datasets and Inputs

Reference 1 provides the downloadable MRI related data in csv format. Reference 2 provides metadata and additional facts about the cross-sectional MRI.

OASIS Cross-sectional MRI Data in Young, Middle Aged, Non-demented and Demented Older Adults

• This dataset consists of a cross-sectional collection for 416 persons aged 18 to 96

• For each person, 3 to 4 T1-weighted MRI scans that were obtained in single scan sessions are included.

• The persons include both men and women, and are all right-handed.

• In this dataset, one hundred persons over the age of 60 have been clinically diagnosed with very mild to moderate Alzheimer's disease (AD).

• Also, a reliability data set , Reference 3, is included which contains 20 non-demented subjects imaged on a subsequent visit within 90 days of their initial session.

• Dementia related Additional Data below for the cross-sectional MRI cases used this project. Features based on these Additional Data will be used to train classification models to predict the labels for the outcome (CDR).

OASIS: Longitudinal MRI Data in Non-demented and Demented Older Adults

This set consists of a longitudinal collection of 150 subjects aged 60 to 96. Each subject was scanned on two or more visits, separated by at least one year for a total of 373 imaging sessions. For each subject, 3 or 4 individual T1-weighted MRI scans obtained in single scan sessions are included.

Note: MRI image pixel data are NOT used in this problem, only related features prefixed with @ sign (below) will be used.

• The subjects are all right-handed and include both men and women.

• 72 of the subjects were characterized as non-demented throughout the study.

• 64 of the included subjects were characterized as demented at the time of their initial visits and remained so for subsequent scans, including 51 individuals with mild to moderate Alzheimer's disease.

• Another 14 subjects were characterized as non-demented at the time of their initial visit and were subsequently characterized as demented at a later visit.

• Dementia related **Additional Data** below for the longitudinal MRI cases are used this project.

Features based on the **Additional Data** are relevant to finding machine learning solutions to the problem defined above, and will be used to train classification models to predict the labels for the outcome (Critical Dementia Rating, CDR).

Additional Data: Specific References in parentheses below covering features are from Reference 2. These features include Demographic, clinical, and derived anatomic measures related to brain that are located in the file oasis_crosssectional.csv. Features prefixed with @ will be used for the problem.

Demographic data:

```
- @Gender (M/F), categorical data
- Handedness (Right or Left Handed), categorical data, all of which are right hande
d in the dataset.
- @Age (numeric),
- @Education (Educ, categorical). Education codes correspond to the following level
s of education:
    1=Less than high school graduate.
    2=High school graduate.
    3=Some college education
    4=College graduate.
    5=Beyond college.
```

Clinical data:

```
- @Mini-Mental State Examination (MMSE, Reference 8),

- @Clinical Dementia Rating (CDR, Reference 7)
     0 = non-demented (341 data points)
     0.5 = very mild dementia (193 data points)
     1 = mild dementia (69 data points)
     2 = moderate dementia (5 data points)
```

There are some records with NaN values in one or more fields; these records will be removed from datasets prior to analysis. All participants with dementia (CDR >0) were diagnosed with probable Alzheimer's Disease.

Derived anatomic volumes data:

```
- @Estimated total intracranial volume (eTIV, mm3), Reference 4
- @Atlas scaling factor (ASF), Reference 4
- @Normalized whole brain volume (nWBV, mm3), Reference 5
```

Load Libraries

In [1]:
```
# Load libraries
import numpy as np
from pandas import read_csv
from pandas.tools.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
```

Load CSV data to Pandas Dataframes

In [2]:
```
# Read a local csvfile in Pandas for the OASIS cross-sectional MRI dataset dow
nloaded from OASIS web site (Reference 1)
import pandas as pd
dfoasx=pd.read_csv('oasis_cross-sectional.csv')
dfoasx.shape
```

Out[2]: (436, 12)

436 rows and 12 columns of data

In [3]:
```
# Read a csvfile in Pandas for the OASIS longitudinal MRI dataset downloaded f
rom the OASIS web site (Reference 1)
import pandas as pd
dfoasl=pd.read_csv('oasis_longitudinal.csv')
dfoasl.shape
```

Out[3]: (373, 15)

373 rows and 15 columns of data.

Analysis

Data Exploration

[If a dataset is present, features and calculated statistics relevant to the problem have been reported and discussed, along with a sampling of the data. In lieu of a dataset, a thorough description of the input space or input data has been made. Abnormalities or characteristics about the data or input that need to be addressed have been identified.]

Following type of information are obtained below from data exploration. Later, graphical exploration of the datasets via plots provide more insights into data distribution, correlation among features (columns), and outliers (box plots).

- Dataset shape (number of rows and columns)
- Column headings
- Data Summary and statistics
- Data Types
- Count of data by classifier label(CDR) values

OASIS Cross-sectional Data Exploration

In [4]:
```
# Column names in the dataset, eleven columns, including the index column (ID)
list(dfoasx.columns.values)
```

Out[4]:
```
['ID',
 'M/F',
 'Hand',
 'Age',
 'Educ',
 'SES',
 'MMSE',
 'CDR',
 'eTIV',
 'nWBV',
 'ASF',
 'Delay']
```

In [5]:
```
# Summary statistics for cross-sectional MRI related dataset.
dfoasx.describe()
```

Out[5]:

	Age	Educ	SES	MMSE	CDR	eTIV	nWBV
count	436.000000	235.000000	216.000000	235.00000	235.000000	436.000000	436.000
mean	51.357798	3.178723	2.490741	27.06383	0.285106	1481.919725	0.79167
std	25.269862	1.311510	1.120593	3.69687	0.383405	158.740866	0.05993
min	18.000000	1.000000	1.000000	14.00000	0.000000	1123.000000	0.64400
25%	23.000000	2.000000	2.000000	26.00000	0.000000	1367.750000	0.74275
50%	54.000000	3.000000	2.000000	29.00000	0.000000	1475.500000	0.80900
75%	74.000000	4.000000	3.000000	30.00000	0.500000	1579.250000	0.84200
max	96.000000	5.000000	5.000000	30.00000	2.000000	1992.000000	0.89300

We note the following from the demographic and clinical features and cognitive test data for the cross-sectional MRI data, the following:

Mean Age for cross-sectional MRI data is about 51 years, mean for Education is 3.2 years, SES about 2.5, MMSE 27, eTIV 1481, nWBV 0.79, ASF 1.2.

Also note the missing (NaN) data from the count values. Those columns that have count of 436 have missing (NaN) data. The Delay column has most missing data followed by Educ, SES, MMSE, and the label (CDR) which have mising data in many rows. An option is to remove rows with missing column values, and that option will be chosen as seen later in this notebook. Another option, not used here, is to replace the NaN values with mean values listed for colmuns below. That latter choice would be appropriate if the column values have a normal or an uniform distribution. Also note that the values in the different columns are of different orders of magnitude, Some on the order of unity (SES, CDR, nWBV, and ASF), and other column values of higher of magnitude (Age and eTIV). Later in the notebook, rescaling or normalizing the columns(features) will be chosen as sensitivity studies for impact on accuracy and results,

In [6]:
```
# Note that the column data types are object, float64, and int64 data types.
dfoasx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 436 entries, 0 to 435
Data columns (total 12 columns):
ID       436 non-null object
M/F      436 non-null object
Hand     436 non-null object
Age      436 non-null int64
Educ     235 non-null float64
SES      216 non-null float64
MMSE     235 non-null float64
CDR      235 non-null float64
eTIV     436 non-null int64
nWBV     436 non-null float64
ASF      436 non-null float64
Delay    20 non-null float64
dtypes: float64(7), int64(2), object(3)
memory usage: 41.0+ KB
```

In [7]:
```
dfoasx.head(5)
```

Out[7]:

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	OAS1_0001_MR1	F	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	NaN
1	OAS1_0002_MR1	F	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	NaN
2	OAS1_0003_MR1	F	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207	NaN
3	OAS1_0004_MR1	M	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105	NaN
4	OAS1_0005_MR1	M	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010	NaN

Note categorical data for gender M/F, and label (CDR) values>=0.0, and NaN. Cross-sectional MRI dataset has many rows with NaN values in multiple columns. These NaN values will be removed after merging the dataset with the Longitudinal MRI dataset.

In [8]:
```
# Determine the count of various values for the label CDR
print (dfoasx.CDR[(dfoasx.CDR == 0.0)].count(),",", dfoasx.CDR[(dfoasx.CDR ==
0.5)].count())
print (dfoasx.CDR[(dfoasx.CDR == 1.0)].count(),",", dfoasx.CDR[(dfoasx.CDR ==
2)].count())
# Calculate number of records with missing label(CDR) values.
sum(pd.isnull(dfoasx['CDR']))
```

```
135 , 70
28 , 2
```

Out[8]: 201

- Data frequency for CDR label: ` There are 135, 70, 28, 2, and 201 rows with CDR label values 0, 0.5, 1.0, and 2, and NaN.
- We see only 2 data rows for CDR= 2.

OASIS Longitudinal Data Exploration

In [9]:
```
list(dfoasl.columns.values)
```

Out[9]:
```
['Subject ID',
 'MRI ID',
 'Group',
 'Visit',
 'MR Delay',
 'M/F',
 'Hand',
 'Age',
 'EDUC',
 'SES',
 'MMSE',
 'CDR',
 'eTIV',
 'nWBV',
 'ASF']
```

The three additional columns (Subject ID, Group, and Visit) in dfoasl dataset are not in the dfoasx dataset. These three columns are not meaningful for the CDR classification, and will be dropped before merging the dloasx and dfoasl datasets. The MR Delay and Delay columns in the two datasets have same meaning and the MR Delay column will be repositioned to be in the same column order as the Delay column.

In [10]:
```
dfoasl.describe()
```

Out[10]:

	Visit	MR Delay	Age	EDUC	SES	MMSE	CDR
count	373.000000	373.000000	373.000000	373.000000	354.000000	371.000000	373.000
mean	1.882038	595.104558	77.013405	14.597855	2.460452	27.342318	0.29088
std	0.922843	635.485118	7.640957	2.876339	1.134005	3.683244	0.37455
min	1.000000	0.000000	60.000000	6.000000	1.000000	4.000000	0.00000
25%	1.000000	0.000000	71.000000	12.000000	2.000000	27.000000	0.00000
50%	2.000000	552.000000	77.000000	15.000000	2.000000	29.000000	0.00000
75%	2.000000	873.000000	82.000000	16.000000	3.000000	30.000000	0.50000
max	5.000000	2639.000000	98.000000	23.000000	5.000000	30.000000	2.00000

Mean Age for longitudinal MRI data is about 77 years, mean for Education is 14.6 years, SES about 2.5, MMSE 27, eTIV 1488, nWBV 0.73, ASF 1.2. Also note the missing (NaN) data from the count values. Those columns that have count of 436 have missing (NaN) data. The Delay column has most missing data followed by Educ, SES, MMSE, and the label (CDR) which have mising data in many rows. Another option, not used here, is to replace the NaN values with mean values listed for colmuns below. That latter choice would be appropriate if the column values have a normal or an uniform distribution. Also note that the values in the different columns are of different orders of magnitude, some on the order of unity (SES, CDR, nWBV, and ASF), and other column values of higher of magnitude (Age and eTIV).

In [11]:
```
dfoasl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 15 columns):
Subject ID    373 non-null object
MRI ID        373 non-null object
Group         373 non-null object
Visit         373 non-null int64
MR Delay      373 non-null int64
M/F           373 non-null object
Hand          373 non-null object
Age           373 non-null int64
EDUC          373 non-null int64
SES           354 non-null float64
MMSE          371 non-null float64
CDR           373 non-null float64
eTIV          373 non-null int64
nWBV          373 non-null float64
ASF           373 non-null float64
dtypes: float64(5), int64(5), object(5)
memory usage: 43.8+ KB
```

In [12]:
```
dfoasl.head(5)
```

Out[12]:

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	NaN
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	NaN
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	NaN

Longitudinal MRI dataset has many rows with NaN values in multiple columns. These NaN values will be removed after merging the dataset with the Cross-sectional MRI dataset.

In [13]:
```
typesl=dfoasx.dtypes
print(typesl)
```

```
ID        object
M/F       object
Hand      object
Age        int64
Educ     float64
SES      float64
MMSE     float64
CDR      float64
eTIV       int64
nWBV     float64
ASF      float64
Delay    float64
dtype: object
```

In [14]:
```
# Determine the count of various values for the label CDR
print (dfoasl.CDR[(dfoasl.CDR == 0.0)].count(),",", dfoasl.CDR[(dfoasl.CDR ==
0.5)].count())
print (dfoasl.CDR[(dfoasl.CDR == 1.0)].count(),",", dfoasl.CDR[(dfoasl.CDR ==
2)].count())
```

```
206 , 123
41 , 3
```

In [15]:
```
import pandas as pd
## Calculate number of records with missing label(CDR) values.
sum(pd.isnull(dfoasl['CDR']))
```

Out[15]: 0

For the longitidinal MRI dataset, there are no records with missing values (NaN) in the CDR column. Earlier we found 201 rows with missing values in the cross-sectional dataset.

Data frequency for CDR label:

- In the longitudinal MRI dataset, there are 206, 123, 41, and 3 rows with CDR label values 0, 0.5, 1.0, and 2, respectively.
- There are very few (3) data points with CDR=2. This along woth no CDR=2 data in the cross-sectional dataset, is a potential limitation in modeling the classification problem as multi-label (not binary) classification since the some cross validation sets may have very few or even no data points. Metrics for multi-label clasification may not be meaningful for CDR=2 data. One alternative is to not include the few CDR=2 data points in the multi-label classification case.Another alternative is to duplicate the existing CDR=2 data rows a few times and appenf to the combined dataset.

We find 201 missing CDR values in the cross-sectional dataset (dfoasx), and no missing values in the longitudinal MRI dataset) dfoasl).

The count of various CDR labels for the two datasets and the total values are shown in the Table below. Again we see that the total number of records for the combined dataset would be 5, a small number of datarecords to calculate meaningful metrics.

- CDR = 0, 0.5, 1.0, 2, NaN
- dfoasx =135, 70, 28, 2, 201
- dfoasl= 206, 123, 41, 3, 0
- Total = 341, 193, 69, 5, 201

Methodology

Data Preprocessing

- Replace gender categorical data (M, F) with numerical values (0, 1).
- Select longitudinal dataset (dfoasl) columns that have data similar to the dataset for cross-sectional dataset (dfoasx). This will facilitate combining the two datasets to create a merged dataset (dfoas_merge).
- Explore the merged dataset
- Remove NaN rows; drop rows with NaN values (missing data) in at least one column. This will make classification metrics meaningful.

In [16]:
```
# Cross-sectional  MRI data preparation
# Replace gender data "M" and "F" with numerical inputs 0, and 1
dfoasx['M/F'] = dfoasx['M/F'].replace('F', 1)
dfoasx['M/F'] = dfoasx['M/F'].replace('M', 0)
```

In [17]:
```
# Longitudinal  MRI data preparation
# Replace gender data "M" and "F" with numerical inputs 0, and 1
# Convert CDR>0 values to 1 to make this a binary classification problem (CDR
 values: 0 or 1)
dfoasl['M/F'] = dfoasl['M/F'].replace('F', 1)
dfoasl['M/F'] = dfoasl['M/F'].replace('M', 0)
```

In [18]:
```
# Select longitudinal dataset columns that are similar to the dataset for cros
s-sectional dataset
dfoasl2= dfoasl[['MRI ID','M/F','Hand','Age',
 'EDUC','SES','MMSE','CDR','eTIV','nWBV','ASF', 'MR Delay']]

dfoasl2.head()
```

Out[18]:

	MRI ID	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF	MR Dela
0	OAS2_0001_MR1	0	R	87	14	2.0	27.0	0.0	1987	0.696	0.883	0
1	OAS2_0001_MR2	0	R	88	14	2.0	30.0	0.0	2004	0.681	0.876	457
2	OAS2_0002_MR1	0	R	75	12	NaN	23.0	0.5	1678	0.736	1.046	0
3	OAS2_0002_MR2	0	R	76	12	NaN	28.0	0.5	1738	0.713	1.010	560
4	OAS2_0002_MR3	0	R	80	12	NaN	22.0	0.5	1698	0.701	1.034	1895

In [19]:
```
# Rename columns:MRI ID to ID, EDUC to Educ, MR Delay to Delay similar to cros
s-sectional MRI dataset.
dfoasl2=dfoasl2.rename(columns={'EDUC':'Educ', 'MRI ID':'ID', 'MR Delay':'Dela
y'})
```

Merge the cross-sectional and the Longitudinal MRI datasets

In [20]:
```
# Merge the cross-sectional and the longitudinal MRI datasets
dfoas_merge = dfoasx.append(dfoasl2, ignore_index=True)
dfoas_merge.shape
```

Out[20]: (809, 12)

In [21]:
```
dfoas_merge.head(10)
```

Out[21]:

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	OAS1_0001_MR1	1	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	NaN
1	OAS1_0002_MR1	1	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	NaN
2	OAS1_0003_MR1	1	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207	NaN
3	OAS1_0004_MR1	0	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105	NaN
4	OAS1_0005_MR1	0	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010	NaN
5	OAS1_0006_MR1	1	R	24	NaN	NaN	NaN	NaN	1131	0.862	1.551	NaN
6	OAS1_0007_MR1	0	R	21	NaN	NaN	NaN	NaN	1516	0.830	1.157	NaN
7	OAS1_0009_MR1	1	R	20	NaN	NaN	NaN	NaN	1505	0.843	1.166	NaN
8	OAS1_0010_MR1	0	R	74	5.0	2.0	30.0	0.0	1636	0.689	1.073	NaN
9	OAS1_0011_MR1	1	R	52	3.0	2.0	30.0	0.0	1321	0.827	1.329	NaN

The merged dataset dfoas_merge has 809 rows and 12 columns

Explore the merged dataset

In [22]:
```
dfoas_merge.describe()
```

Out[22]:

	M/F	Age	Educ	SES	MMSE	CDR	eTIV
count	809.000000	809.000000	608.000000	570.00000	606.000000	608.000000	809.0000
mean	0.594561	63.186650	10.184211	2.47193	27.234323	0.288651	1484.782
std	0.491280	23.117511	6.058388	1.12805	3.687980	0.377697	166.9116
min	0.000000	18.000000	1.000000	1.00000	4.000000	0.000000	1106.000
25%	0.000000	49.000000	4.000000	2.00000	26.000000	0.000000	1361.000
50%	1.000000	72.000000	12.000000	2.00000	29.000000	0.000000	1475.000
75%	1.000000	80.000000	16.000000	3.00000	30.000000	0.500000	1583.000
max	1.000000	98.000000	23.000000	5.00000	30.000000	2.000000	2004.000

Mean values for the merged dataset are: Age(63.2), Educ(10.2), SES(2.5). MMSE(27.2), eTIV(1485), nWBV(0.76), ASF (1.20)

In [23]:
```
# Replace NaN values for the Delay column to dataset mean of 566 to facilita
te removal of rows with any NaN later.
#This is because in the longitudinal MRI dataset most rows have Delay value
 of NaN.
# So those rows will also be removed unless we replace the Delay values with
 a value other than NaN.
# We will replace NaN with 565, the mean value of the Delay column.

dfoas_merge['Delay'] = dfoas_merge['Delay'].replace(np.NaN, 566)
```

In [24]:
```
dfoas_merge.head(10)
```

Out[24]:

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	OAS1_0001_MR1	1	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	566.0
1	OAS1_0002_MR1	1	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	566.0
2	OAS1_0003_MR1	1	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207	566.0
3	OAS1_0004_MR1	0	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105	566.0
4	OAS1_0005_MR1	0	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010	566.0
5	OAS1_0006_MR1	1	R	24	NaN	NaN	NaN	NaN	1131	0.862	1.551	566.0
6	OAS1_0007_MR1	0	R	21	NaN	NaN	NaN	NaN	1516	0.830	1.157	566.0
7	OAS1_0009_MR1	1	R	20	NaN	NaN	NaN	NaN	1505	0.843	1.166	566.0
8	OAS1_0010_MR1	0	R	74	5.0	2.0	30.0	0.0	1636	0.689	1.073	566.0
9	OAS1_0011_MR1	1	R	52	3.0	2.0	30.0	0.0	1321	0.827	1.329	566.0

Drop rows with NaN

Now drop rows in the merged dataset which have NaN in any column. We see that there are 809-570=239 rows with NaN values (missing data) in at least one column.

In [25]:
```
# Drop rows with NaN
dfoas_merge=dfoas_merge.dropna(how='any')
```

In [26]:
```
dfoas_merge.shape
```

Out[26]: (570, 12)

In [27]:
```
dfoas_merge.dtypes
```

Out[27]:
```
ID        object
M/F        int64
Hand      object
Age        int64
Educ     float64
SES      float64
MMSE     float64
CDR      float64
eTIV       int64
nWBV     float64
ASF      float64
Delay    float64
dtype: object
```

In [28]:
```
dfoas_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 570 entries, 0 to 808
Data columns (total 12 columns):
ID       570 non-null object
M/F      570 non-null int64
Hand     570 non-null object
Age      570 non-null int64
Educ     570 non-null float64
SES      570 non-null float64
MMSE     570 non-null float64
CDR      570 non-null float64
eTIV     570 non-null int64
nWBV     570 non-null float64
ASF      570 non-null float64
Delay    570 non-null float64
dtypes: float64(7), int64(3), object(2)
memory usage: 57.9+ KB
```

In [29]:
```
# Convert CDR>0 values to 1 to make this a binary classification problem (CDR
 values: 0 or 1)
# Leave CDR values of 0 as is
# Convert CDR values >0 to 1

dfoas_merge['CDR'] = dfoas_merge['CDR'].replace(0.5, 1)
dfoas_merge['CDR'] = dfoas_merge['CDR'].replace(2, 1)
dfoas_merge.head(10)
```

Out[29]:

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Dela
0	OAS1_0001_MR1	1	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	566.
1	OAS1_0002_MR1	1	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	566.
2	OAS1_0003_MR1	1	R	73	4.0	3.0	27.0	1.0	1454	0.708	1.207	566.
8	OAS1_0010_MR1	0	R	74	5.0	2.0	30.0	0.0	1636	0.689	1.073	566.
9	OAS1_0011_MR1	1	R	52	3.0	2.0	30.0	0.0	1321	0.827	1.329	566.
11	OAS1_0013_MR1	1	R	81	5.0	2.0	30.0	0.0	1664	0.679	1.055	566.
14	OAS1_0016_MR1	0	R	82	2.0	4.0	27.0	1.0	1477	0.739	1.188	566.
16	OAS1_0018_MR1	0	R	39	3.0	4.0	28.0	0.0	1636	0.813	1.073	566.
17	OAS1_0019_MR1	1	R	89	5.0	1.0	30.0	0.0	1536	0.715	1.142	566.
18	OAS1_0020_MR1	1	R	48	5.0	2.0	29.0	0.0	1326	0.785	1.323	566.

In [30]:
```
print (dfoas_merge.CDR[(dfoas_merge.CDR == 0.0)].count(),",", dfoas_merge.CDR
[(dfoas_merge.CDR == 0.5)].count(), ",",
       dfoas_merge.CDR[(dfoas_merge.CDR == 1.0)].count(), ",", dfoas_merge.CDR
[(dfoas_merge.CDR == 2)].count())
```

```
339 , 0 , 231 , 0
```

Binary Labels(CDR values: 0,1): We see from above exploratory results that the combined cross-sectional and longitudinal datasets have 339 records for CDR=0.0, and 231 records for CDR=1 (recall CDR values of 0.5 and 2.0 have been replaced by 1.0 to make it a binary classification problem. Thus the labels are "balanced", the number of records with CDR=0.0 and CDR=1.0 are 59% and 41% respectively. This justifies the use of Classification accuracy as a metric in addition to ROC/AUC, Confusion Matrix (sensitivity, specificity, recall, and support metrics) from sklearn.

Exploratory Visualization

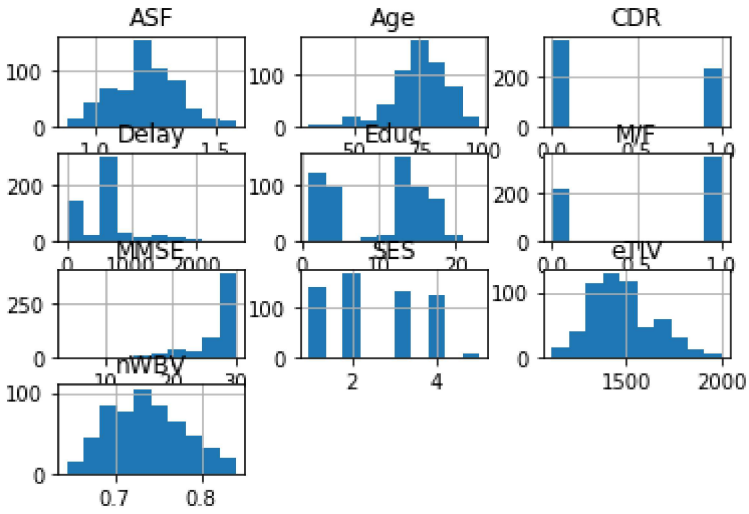
[A visualization has been provided that summarizes or extracts a relevant characteristic or feature about the dataset or input data with thorough discussion. Visual cues are clearly defined.]

The following visualizations of the merged dataset (dfoas_merge dataframe) have been provided below.

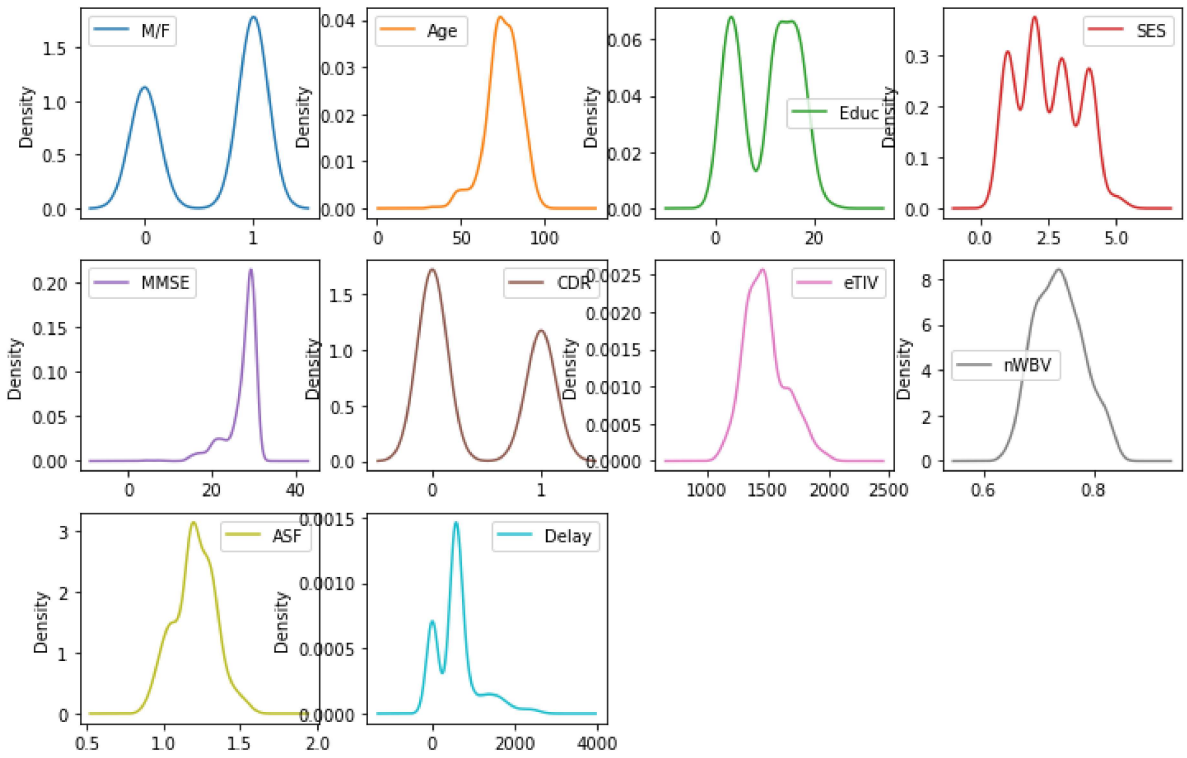
- Histogram
- Density Plots
- Box Plots
- Scatter Plots
- Correlation matrix

From the histogram. density plots, and box plots for the feature variables, we note the following: Age, eTIV, nWBV, and ASF have approximately normal distribution. The feature variables, Educ, and SES have bi-modal or multimodal distribution. As expected CDR after transformation of CDR>0 values to 1, indicates binary distribution (0, and 1). The gender variable M/F indicates transformed numerical values of 0 and 1 as expected.

In [31]:
```
from matplotlib import pyplot
dfoas_merge.hist()
fig_size = pyplot.rcParams["figure.figsize"]
fig_size[0] = 8
fig_size[1] = 10
pyplot.rcParams["figure.figsize"] = fig_size
pyplot.show();
```



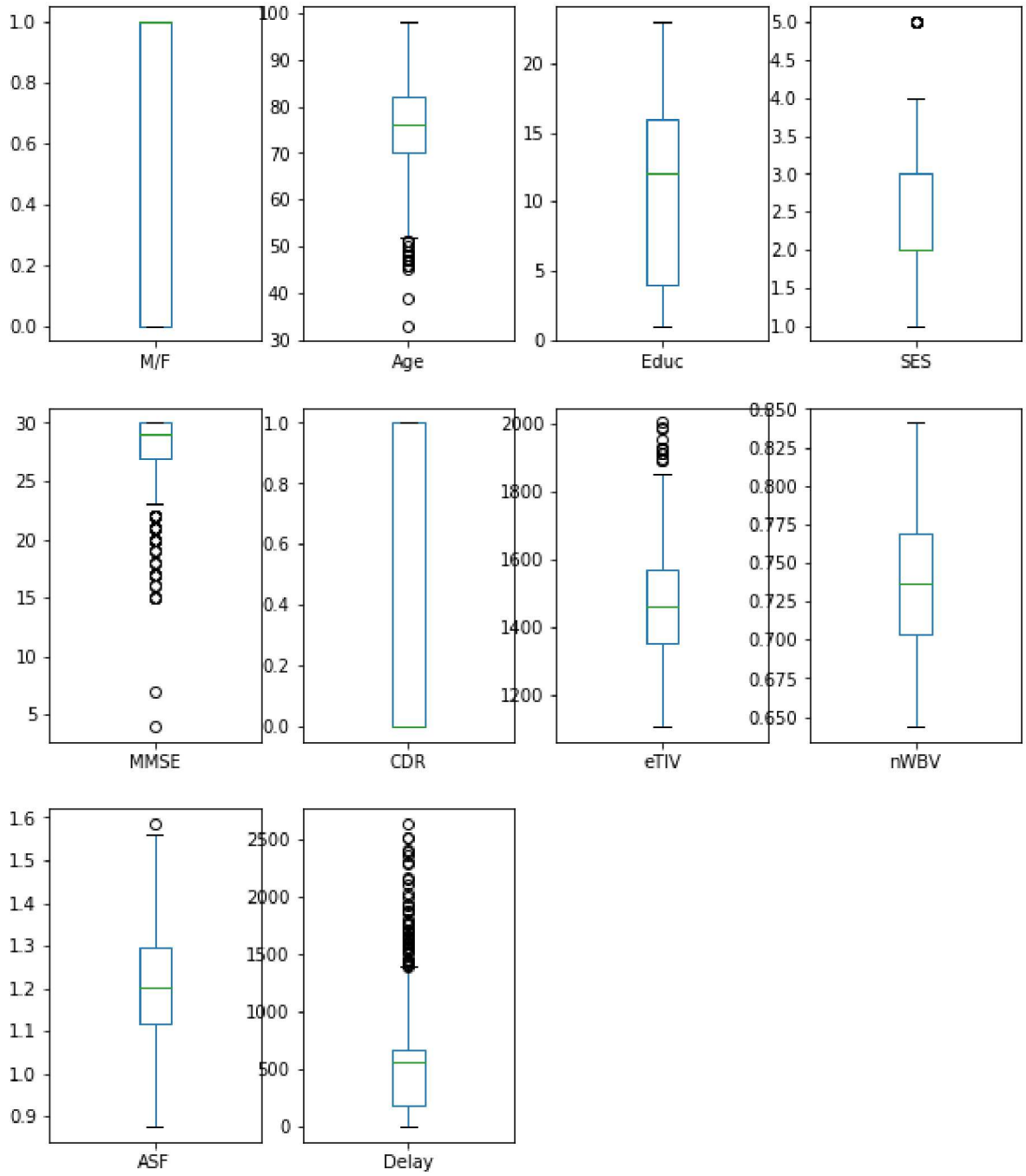
In [32]:
```
# Univariate Density Plots
from matplotlib import pyplot
dfoas_merge.plot(kind='density', subplots=True, layout=(3, 4), figsize=(12, 8
), sharex=False)
pyplot.show();
```



Effect of removing Outliers from the datasets

Outliers are column values located, outside the bands defined by 1.5 times greater than the size of spread of the middle 50% of the data. We see from the box and whiskers plots that Age, SES, eTIV, MMSE, and Delay columns have some outliers. The base classification case will be run without removing the outliers. A sensitivity study was performed separately to train the model by removing the outliers. That case is not shown here. The results/metrics were not noticeably different from the base case with outliers.

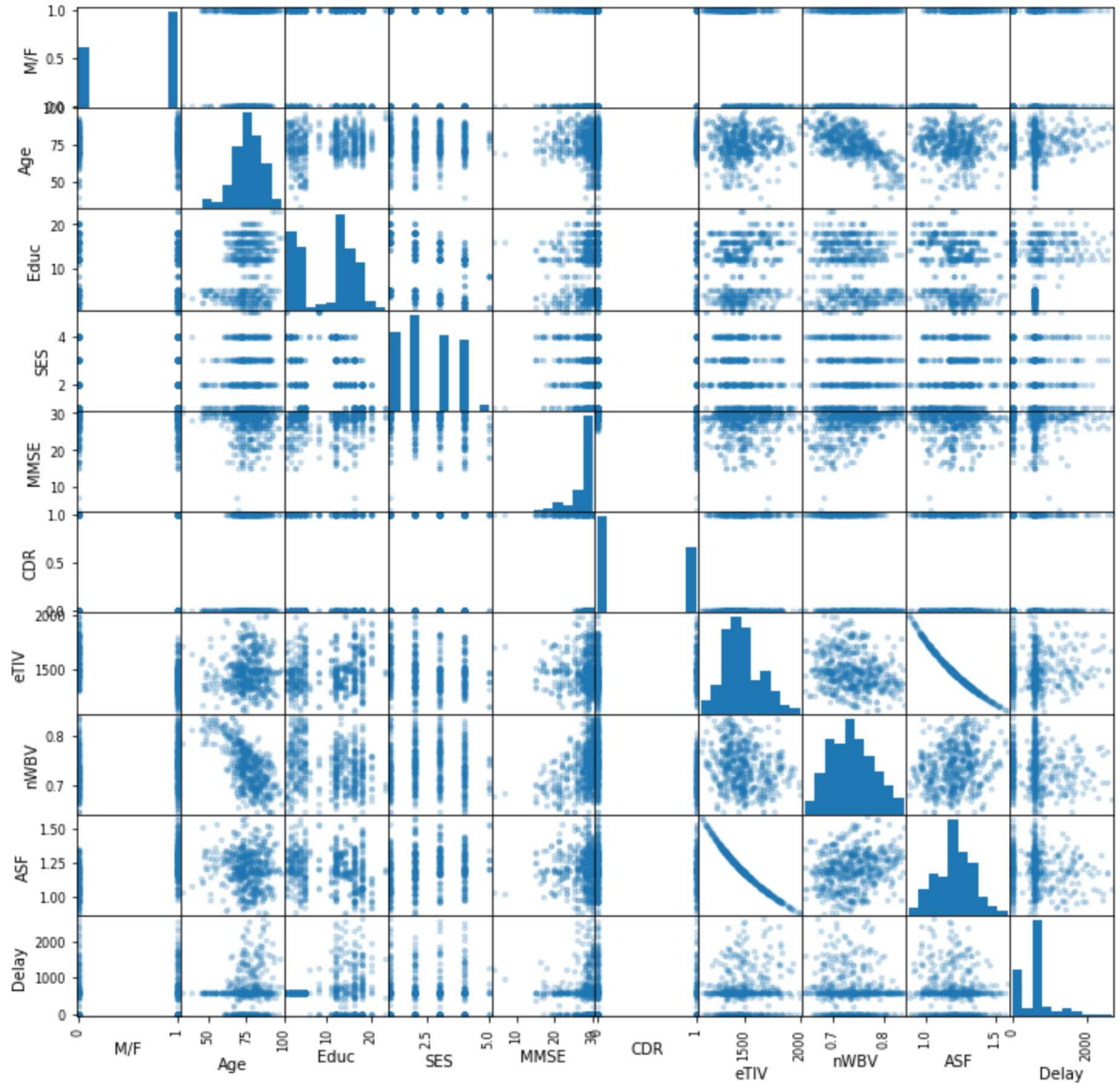
In [33]:
```
dfoas_merge.plot(kind='box', subplots=True, layout=(3, 4), figsize=(10,12), sharex=False, sharey=False)
pyplot.show();
```



Multivariate Plots

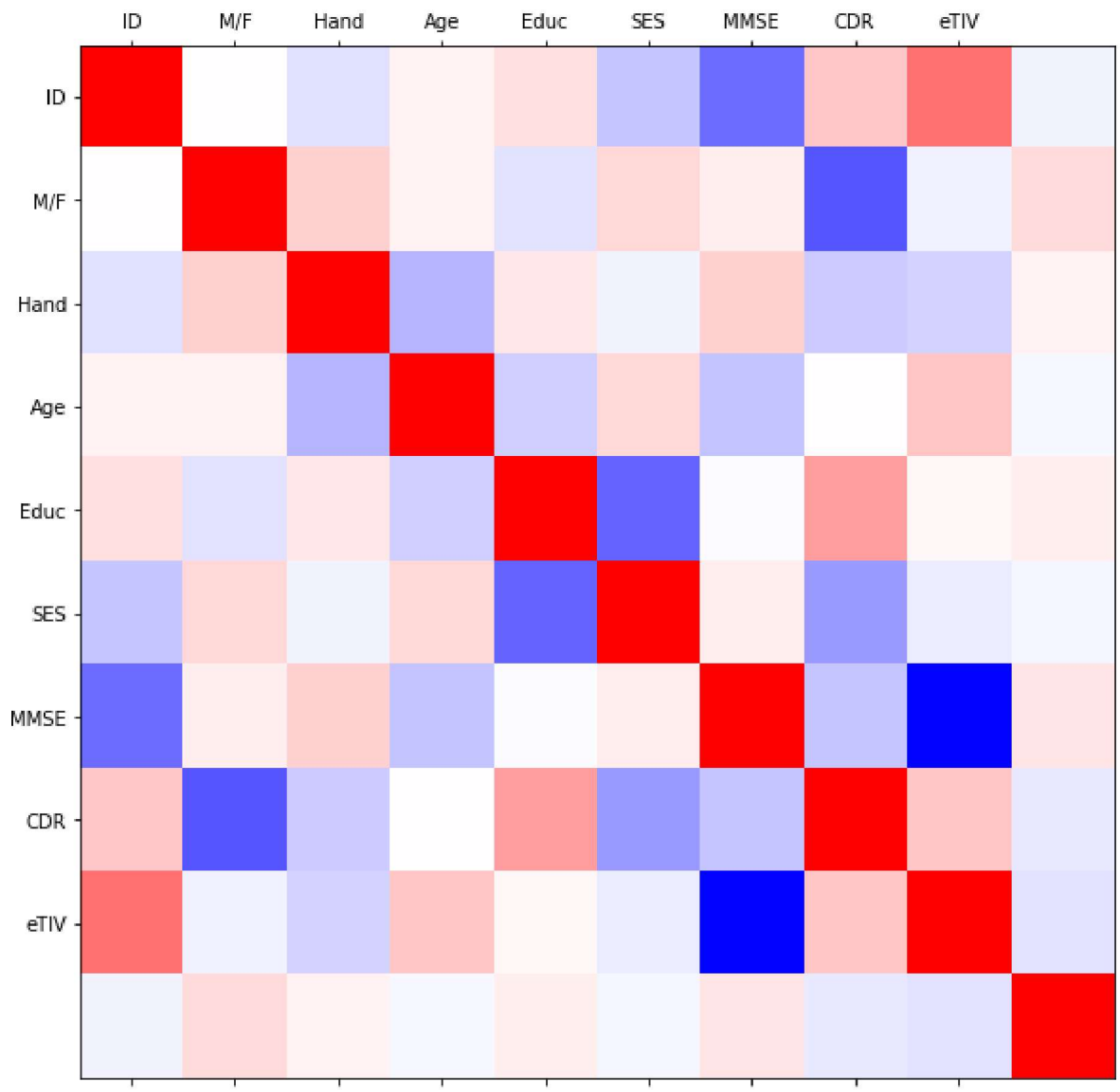
Two multivariate plots are displayed below. The Scatter plot and the Correlation plot. These plots indicate whether there are any dependencies/interactions and correlation between the features in the dataset. For example, the scatter plot shows that eTIV and ASF have interactions/negative depenence, whereas the features Age has negative correlation with nWBV. Usually only one of the pair of the variables that are correlated should be included and the other removed in order to improve classification accuracy. We will use well known feature elimination algorithms such as Recursive Feature Elimination (RFE) and view its sensitivity on the classification results/metrics.

In [34]:
```
# Scatterplot Matrix
from matplotlib import pyplot
from pandas.tools.plotting import scatter_matrix
scatter_matrix(dfoas_merge, alpha=0.2, figsize=(12, 12))
pyplot.show();
```



In [35]:
```
# Correlation Matrix Plot
from matplotlib import pyplot
from matplotlib import cm
import numpy as np
correlations = dfoas_merge.corr()
# plot correlation matrix
figoasx = pyplot.figure()
cm = pyplot.cm.bwr
fig = pyplot.figure()
ax = fig.add_subplot(111);
cax = ax.matshow(correlations, cmap=cm, vmin=-1, vmax=1);
ticks = np.arange(0,9,1);
names= list(dfoas_merge.columns.values)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
fig.set_figheight(10)
fig.set_figwidth(10)
pyplot.show();
```

```
<matplotlib.figure.Figure at 0x1d3ee584ac8>
```



We see that SES and Educ are correlated and MMSE and eTIV are correlated.

Methodology- Solution

Solution Statement

• Train a supervised machine learning classification model to properly classify the OASIS data according to clinical dementia ratings(CDR values).

• Train a number of candidate models from the scikit-learn library (Reference 12) such as Logistic Regression, Linear Discriminant Analysis, KNN, Naive Bayes, CART, and SVM, Random Forest Classifier (RFC) and Gradient Boosting Machine (GBM).

• Select the best model based on Classification "Accuracy", as metric. Additionally, for this binary classification AUC/ROC values will be reported. For multi- class classification (multiple CDR labels) F-1 score will be reported. The results from the best model will be provided along with those from the other models.

• Combine the cross-sectional and longitudinal MRI related demographic and clinical data into a single dataset. This increases number of data points for training the classifier.

• Split the resulting dataset into training dataset (80%) and the remaining data(20%) for typical ten-fold cross validation.

• Report the prediction accuracy of the models and identify the model that yields the higest classification accuracy. Report accuracy results in sklearn Confusion Matrix format (to evaluate classifier output quality) and, Classification Report format (provides, precision, recall, f1-score). See References 13 through 17.

Free-Form Visualization (Learning Curves)

Learning curves are provided in **Appendix 2** later in this notebook. These curves provide visual demonstration of the dependence of the training and validation accuracy on the size of the dataset (number of records or samples) for the merged OASIS dataset for a number of classifiers such as Random Forest, Gradient Boosting. This visualization clarifies the the number of samples required for a particular prediction accuracy threshold, and qualitatively/visually demonstrates the level of overfitting in the training dataset.

Methodology - Select Training and Test/Validation Sets

In [36]:
```
# Split-out validation dataset
array = dfoas_merge.values
#X = array[:,[1, 3, 4, 5, 6, 8, 9, 10]] Backup/extra
X = array[:,[1, 3, 4, 5, 6, 8, 9, 10]]
#X = array[:,[3, 6, 9, 10]] #Feature Extraction with RFE
Y = array[:,7] # all rows and CDR column used for AUC ROC calculations later
Y_float64 = array[:,7] # all rows and CDR column used for AUC ROC calculations
 later
Y=Y.astype(np.str)
validation_size = 0.20
seed = 12345
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,
test_size=validation_size, random_state=seed)
print (X)
print (Y)
```

```
[[1 74 2.0 ..., 1344 0.743 1.306]
 [1 55 4.0 ..., 1147 0.81 1.531]
 [1 73 4.0 ..., 1454 0.708 1.207]
 ...,
 [1 61 13.0 ..., 1319 0.8009999999999999 1.331]
 [1 63 13.0 ..., 1327 0.7959999999999999 1.3230000000000002]
 [1 65 13.0 ..., 1333 0.8009999999999999 1.317]]
['0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '0.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0' '0.0' '0.0'
 '1.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '0.0'
 '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0'
 '0.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '1.0'
 '1.0' '1.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0'
 '1.0' '0.0' '1.0' '1.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0'
 '0.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0'
 '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '1.0' '1.0' '1.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0'
 '0.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '1.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '0.0']
```

Algorithms and Techniques

[Algorithms and techniques used in the project are thoroughly discussed and properly justified based on the characteristics of the problem.]

(Per recommendations from Reference 24)

Spot Check Algorithms

Let's evaluate eight different algorithms:

- Logistic Regression (LR).
- Linear Discriminant Analysis (LDA).
- k-Nearest Neighbors (KNN).
- Classification and Regression Trees (CART).
- Gaussian Naive Bayes (NB).
- Support Vector Machines (SVM).
- Gradient Boosting Machine (GBM).
- Random Forest Classifier (RFC).

This list is a good mixture of simple linear (LR, LDA), and nonlinear (KNN, CART, NB and SVM) algorithms. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable. Let's build and evaluate the eight models:

In [37]:
```
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RFC', RandomForestClassifier()))
# evaluate each model in turn
results = []
names = []
print('(', 'Model, ', 'Cross-Validation Accuracy: Mean, Stdev',')')
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='a
ccuracy')
    results.append(cv_results)
    names.append(name)
    msg = (name, format(cv_results.mean(), '.2f'), format(cv_results.std(), '.
2f'))
    print(msg)
```

```
( Model,  Cross-Validation Accuracy: Mean, Stdev )
('LR', '0.80', '0.05')
('LDA', '0.80', '0.04')
('KNN', '0.64', '0.04')
('CART', '0.81', '0.07')
('NB', '0.81', '0.06')
('SVM', '0.62', '0.07')
('GBM', '0.87', '0.04')
('RFC', '0.87', '0.05')
```

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(max_depth=4)
classifier.fit(X_train, Y_train)
X_test = X_validation
Y_test = Y_validation
prediction = classifier.predict(X_test)
print (classifier.score(X_train, Y_train))
print (classifier.score(X_test, Y_test))
```

```
0.855263157895
0.842105263158
```

In [39]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(max_depth=4)
classifier.fit(X_train, Y_train)
X_test = X_validation
Y_test = Y_validation
prediction = classifier.predict(X_test)
print (classifier.score(X_train, Y_train))
print (classifier.score(X_test, Y_test))
```

```
0.879385964912
0.850877192982
```

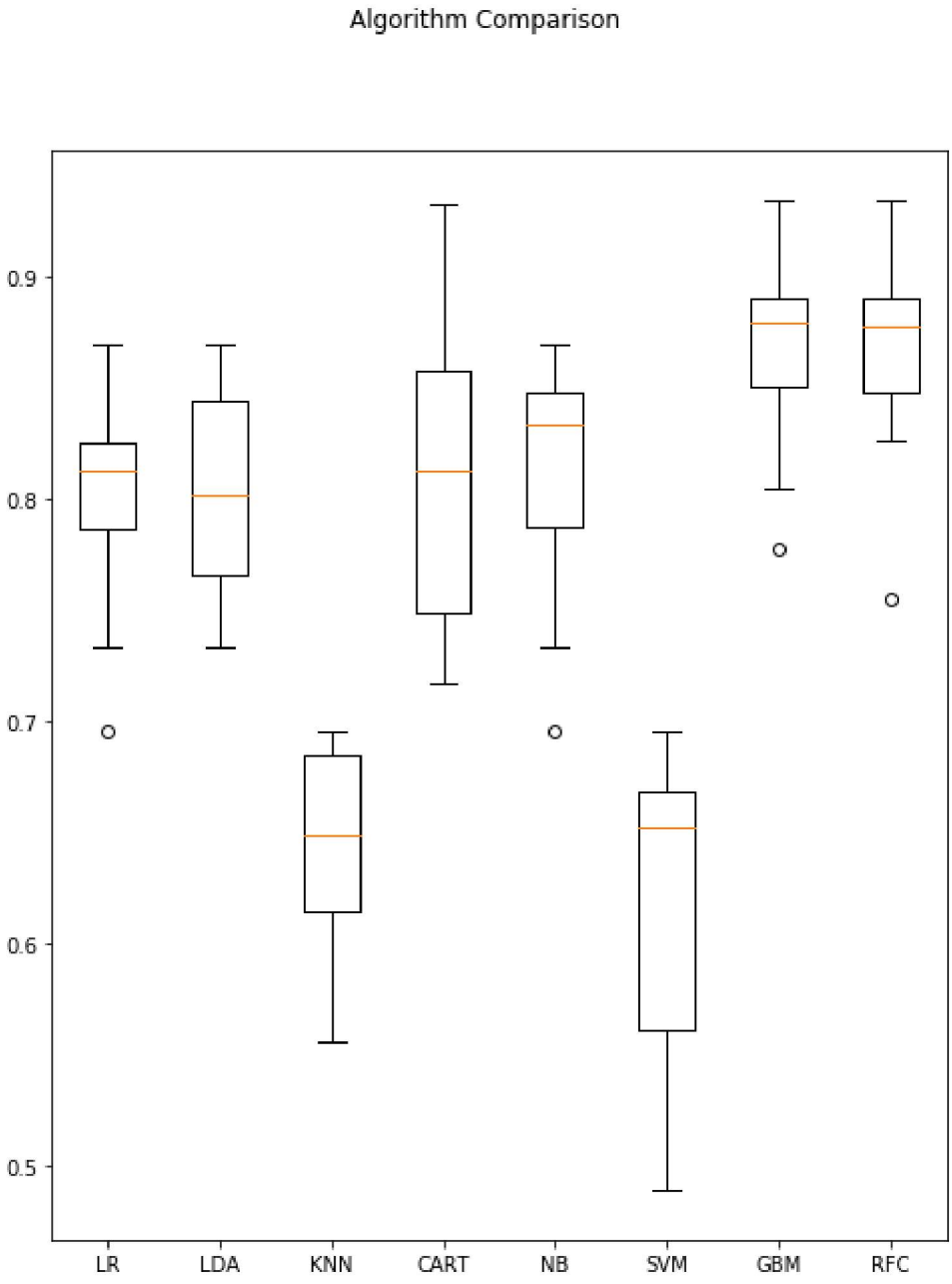
In [40]:
```
from sklearn.ensemble import GradientBoostingClassifier
classifier = GradientBoostingClassifier(max_depth=4)
classifier.fit(X_train, Y_train)
X_test = X_validation
Y_test = Y_validation
prediction = classifier.predict(X_test)
print (classifier.score(X_train, Y_train))
print (classifier.score(X_test, Y_test))
```

```
0.997807017544
0.894736842105
```

Using a 10-fold cross-validation to estimate accuracy. This splits the dataset into 10 parts. Train on 9 parts and test on 1 part. Repeat for all combinations of train-test splits. Using the metric of accuracy to evaluate models which is a ratio of the number of correctly predicted instances divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate).

Compare Accuracy of Algorithms

In [41]:
```
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show();
```



We see that GBM and RFC algorithms have the high accuracy scores, followed LR, CART, LDA and NB that have mid range estimated accuracy scores, with SVM and KNN having much lower scores. Recall these are mean accuracies because each algorithm was evaluated 10 times (10 fold cross-validation).

Algorithm: Bagged Decision Trees for Classification

In [42]:
```
# Bagged Decision Trees for Classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
seed = 122620171
kfold = KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 10
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_
state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.826315789474
```

* The Decision Tree algorithm was quite accurate model that we tested with some possible overfitting during training. Now we want to get an idea of the accuracy of the Decision Tree model on our validation dataset. This will give us an independent check on the accuracy of the best model. We can run the Decision Tree model directly on the validation set and summarize the results as a final accuracy score, a confusion matrix and a classification report.

Random Forest classifier

Random Forest (Reference 24) is one of the most popular and most powerful machine learning algorithms. It is a type of ensemble machine learning algorithm called Bootstrap Aggregation or bagging.

Random Forests (see simpler visual explanations here: https://www.youtube.com/watch?v=D_2LkhMJcfY (https://www.youtube.com/watch?v=D_2LkhMJcfY)) are an improvement over bagged decision trees. Decision Trees are an important type of algorithm for predictive modeling machine learning. Decision trees are high variance algorithms. Bootstrap Aggregation or Bagging is a general procedure that can be used to reduce the variance for high variance algorithms. Modern variations of decision trees like random forest are among the most powerful techniques available.

A problem with decision trees is that they are greedy. They choose which variable to split on using a greedy algorithm that minimizes error. Random forest changes the algorithm for the way that the sub-trees are learned so that the resulting predictions from all of the subtrees have less correlation.

The number of features that can be searched at each split point (m) must be specified as a parameter to the algorithm. We can try different values and tune it using cross-validation. For classification a good default is: m = sqrt(p), where m is the number of randomly selected features that can be searched at a split pointand p is the number of input variables. For example, if a dataset had 25 input variables for a classification problem, then: m = sqrt(25) = 5.

For the results below, the following definitions apply to the Random Forest and the Gradient Boosting Classifiers:

Accuracy score = (number of true positives + number of true negatives)/total number of data points in the dataset.

Confusion Matrix C is defined such that element C_{i, j} is equal to the number of observations known to be in group i but predicted to be in group j.

Thus in binary classification, from the confusion matrix, the count of true negatives is C*{0,0}, false negatives is C*{1,0}, true positives is C*{1,1} and false positives is C*{0,1}.

Precision and Recall: Precision is [True positives/(True positives + False positives)]. Recall is [True positives/(True positives + False Negatives)]. Precision is a measure of exactness or quality, recall is a measure of completeness or quantity. High precision means that an algorithm returned substantially more relevant results than irrelevant ones, while high recall means that an algorithm returned most of the relevant results.

f1-score is the harmonic mean of precision and recall. The scores corresponding to every class tells us the accuracy of the classifier in classifying the data points in that particular class compared to all other classes.

Support is the number of samples of the true response that lie in that class. The total support should equal the number of datapoints in the test set.

In [43]:
```
# Make predictions on validation dataset using the Random Forest classifier
# X_train, Y_train, X_validation, and Y_validation are as defined earlier in t
his notebook.
from sklearn.ensemble import RandomForestClassifier
from time import time
tstart=time()
RFC=RandomForestClassifier()
RFC.fit(X_train, Y_train)
predictions = RFC.predict(X_validation)
predictions_RFC=predictions
print("*Accuracy score: ", accuracy_score(Y_validation, predictions), "\n")
print("*Confusion Matrix: ")
print(confusion_matrix(Y_validation, predictions))
print("*Classification Report: ")
print(classification_report(Y_validation, predictions))
tend=time()
print ('Time required (in milliseconds)', (tend-tstart))
```

```
*Accuracy score:  0.868421052632

*Confusion Matrix:
[[71  2]
 [13 28]]
*Classification Report:
             precision    recall  f1-score   support

        0.0       0.85      0.97      0.90        73
        1.0       0.93      0.68      0.79        41

avg / total       0.88      0.87      0.86       114

Time required (in milliseconds) 0.13800621032714844
```

Gradient Boosting Classifier (Reference 27)

https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/ (https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/)

Boosting is a general ensemble method that creates a strong classifier from a number of weak classifers; it is a general technique that keeps adding weak learners to correct classification errors. A model is built from the training data, then a second model is created that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added. AdaBoost was the first really successful boosting algorithm developed for binary classification. Modern boosting methods build on AdaBoost, most notably stochastic gradient boosting machines.

Gradient boosting involves three elements:

a) A loss function to be optimized. b) A weak learner to make predictions. c) An additive model to add weak learners to minimize the loss function.

The loss function chosen depends on the problem being solved. For example, regression may use a squared error and classification may use logarithmic loss.

Decision trees are used as the weak learner in gradient boosting. Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss. The number of decision trees used generally has 4-to-8 levels. We used max depth = 4 for this classifier. Weak learners are constrained in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes to ensure that the learners remain weak, but can still be constructed in a greedy manner.

Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees. Gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error. Instead of parameters, we have decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). This is done by parameterizing the tree, then modifying the parameters of the tree and movin in the right direction by (reducing the residual loss). This approach is known as functional gradient descent or gradient descent with functions.

Make predictions on validation dataset using the Gradient Boosting classifier: Accuracy, Confusion Matrix, and Classification Report

In [44]:
```
# Make predictions on validation dataset using Gradient Boosting Classifier
# X_train, Y_train, X_validation, and Y_validation are as defined earlier in t
his notebook.
from sklearn.ensemble import GradientBoostingClassifier
GBM = GradientBoostingClassifier()
GBM.fit(X_train, Y_train)
predictions = GBM.predict(X_validation)
print("*Accuracy score for GBM: ", accuracy_score(Y_validation, predictions),
"\n")
print("*Confusion Matrix for GBM: ")
print(confusion_matrix(Y_validation, predictions))
print("*Classification Report for GBM: ")
print(classification_report(Y_validation, predictions))
```

```
*Accuracy score for GBM:  0.885964912281

*Confusion Matrix for GBM:
[[67  6]
 [ 7 34]]
*Classification Report for GBM:
             precision    recall  f1-score   support

        0.0       0.91      0.92      0.91        73
        1.0       0.85      0.83      0.84        41

avg / total       0.89      0.89      0.89       114
```

Cross Validation Classification ROC AUC

In [45]:
```
# Cross Validation Classification ROC AUC
# Uses the RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import label_binarize
# Input array X and Label array Y are as defined previously
array = dfoas_merge.values
# For AUC, Y needs to be int64 data type. If float64 data type originally, con
vert to int64 data type.
# Also, If Y is str or object data type, it does not convert to int64
Y_int64=Y_float64.astype(np.int64)
kfold = KFold(n_splits=10, random_state=6779)
model = RandomForestClassifier()
scoring = 'roc_auc'
results = cross_val_score(model, X, Y_int64, cv=kfold, scoring=scoring)
print("AUC: ", results.mean(), "AUC_stdev: ", results.std())
```

```
AUC:  0.907921046692 AUC_stdev:  0.0711557493647
```

In [46]:
```
# Cross Validation Classification ROC AUC
# Uses the GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import label_binarize
# Input array X and Label array Y are as defined previously
array = dfoas_merge.values
# For AUC, Y needs to be int64 data type. If float64 data type originally, con
vert to int64 data type.
# Also, If Y is str or object data type, it does not convert to int64
Y_int64=Y_float64.astype(np.int64)
kfold = KFold(n_splits=10, random_state=6888)
model = GradientBoostingClassifier()
scoring = 'roc_auc'
results = cross_val_score(model, X, Y_int64, cv=kfold, scoring=scoring)
print("AUC: ", results.mean(), "AUC_stdev: ", results.std())
```

```
AUC:  0.913924841025 AUC_stdev:  0.0546778600743
```

Create a PIPELINE that extracts features from the data then creates a model (from Reference 24)

In [47]:
```
# Create a PIPELINE that extracts features from the data then creates a model
 (from Reference 24)
# Uses DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
# X, Y as defined/calculated previously
# create feature union
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=4)))
feature_union = FeatureUnion(features)
# create pipeline
estimators = []
estimators.append(('feature_union', feature_union))
estimators.append(('CART', DecisionTreeClassifier()))
model = Pipeline(estimators)
# evaluate PIPELINE
kfold = KFold(n_splits=10, random_state=877)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.79298245614
```

In [48]:
```
# Create a PIPELINE that extracts features from the data then creates a model
 (from Reference 24, 26)
# Uses GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
# X, Y as defined/calculated previously
# create feature union
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=4)))
feature_union = FeatureUnion(features)
# create pipeline
estimators = []
estimators.append(('feature_union', feature_union))
estimators.append(('GBC', GradientBoostingClassifier()))
model = Pipeline(estimators)
# evaluate pipeline
kfold = KFold(n_splits=10, random_state=899)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.840350877193
```

Refinement

We will use feature selection as refinement. The number of columns will be reduced using recursive feature selection methodology, and the training/validation process will be repeated. The resulting metrics will be reported. **Appendix 1** contains methodology and results of this refinement.

Feature Selection Methodology

Feature Extraction with Univariate Statistical Tests

In [49]:
```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classi
fication)
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

```
[  1.101e+01   1.729e+01   3.470e+00   6.500e+00   9.865e+01   5.495e+01
   2.233e-01   4.834e-02]
[[  1.000e+00   7.400e+01   2.900e+01   1.344e+03]
 [  1.000e+00   5.500e+01   2.900e+01   1.147e+03]
 [  1.000e+00   7.300e+01   2.700e+01   1.454e+03]
 [  0.000e+00   7.400e+01   3.000e+01   1.636e+03]
 [  1.000e+00   5.200e+01   3.000e+01   1.321e+03]]
```

We see the scores for each attribute and the 4 attributes chosen (those with the highest scores): Age, MMSE, and ASF. We can identify the names for the chosen attributes by matching/mapping the index of the 4 highest scores with the index of the attribute names.

Recursive Feature Elimination (RFE)

The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute. The example below uses RFE with the logistic regression algorithm to select the top 3 features. The choice of algorithm does not matter too much as long as it is skillful and consistent. We see columns Age, SES, MMSE are the top three features

In [50]:
```
# Feature Extraction with RFE
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# feature extraction
model = DecisionTreeClassifier()
rfe = RFE(model, 4)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

```
Num Features: 4
Selected Features: [False  True False False  True False  True  True]
Feature Ranking: [3 1 5 4 1 2 1 1]
```

RFE results above rank highly, the features Age, MMSE, NWBV, and ASF with column index values 3, 6, 9, and 10, respectively.

Appendix 1 in this notebook, includes rerun and results of the algorithms used earlier with the reduced feature set (Age, MMSE, NWBV, and ASF)

In [51]:
```
# Cross Validation Classification ROC AUC with Reduced Features (4 columns)
# Uses the RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import label_binarize
# Input array X and Label array Y are as defined previously
array = dfoas_merge.values
# For AUC, Y needs to be int64 data type. If float64 data type originally, con
vert to int64 data type.
# Also, If Y is str or object data type, it does not convert to int64
Y_int64=Y_float64.astype(np.int64)
kfold = KFold(n_splits=10, random_state=6779)
model = RandomForestClassifier()
scoring = 'roc_auc'
results = cross_val_score(model, X, Y_int64, cv=kfold, scoring=scoring)
print("AUC: ", results.mean(), "AUC_stdev: ", results.std())
```

```
AUC:  0.91658102819 AUC_stdev:  0.0354851357646
```

We see that the Area under the curve (AUC) value is quite high for application of the Random Forest classifier to the merged dataset with reduced features (4 columns)

Principal Component Analysis

Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form. Generally this is called a data reduction technique. A property of PCA is that we can choose the number of dimensions or principal components in the transformed result. In the example below, we use PCA and select 3 principal components.

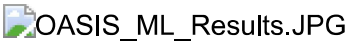
In [52]:
```
# Feature Extraction with PCA
from sklearn.decomposition import PCA
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

```
Explained Variance: [ 0.995  0.003  0.001]
[[ -1.635e-03   3.337e-03   6.285e-03  -1.549e-03  -3.211e-04   1.000e+00
   -5.605e-05  -7.778e-04]
 [ -1.529e-03  -9.876e-01  -1.516e-01  -4.476e-03   3.937e-02   4.252e-03
    2.799e-03  -9.396e-05]
 [  1.511e-03   1.454e-01  -9.804e-01   5.541e-02  -1.204e-01   5.726e-03
   -1.704e-04  -2.076e-04]]
```

We can see that the transformed dataset (3 principal components) bear little resemblance to the source data.

Appendix 1 has description and code for a Pipeline that includes selection of reduced features using PCA, trains a Gradient Boosting Classifier, and reports Accuracy of prediction.

Results



Conclusions, Justification, and Reflections

[Student adequately summarizes the end-to-end problem solution and discusses one or two particular aspects of the project they found interesting or difficult.]

- The formulation of OASIS data (Ref 1 and 2) in terms of a dementia classification problem based on demographic and clinical data only (and without directly using the MRI image data), is a simplification that has major advantages and appeal. This means the trained model can classify whether an individual has dementia or not with about 87% accuracy, without having to wait for radiological interpretation of MRI scans. This can provide an early alert for intervention and initiation of treatment for those with onset of dementia.
- The assumption that the combined cross-sectional and longitudinal datasets would lead to dementia label classification of acceptable accuracy came out to be true. The method required careful data cleaning and data preparation work, converting it to a binary classification problem, as outlined in this notebook.
- At the outset it was not clear which algorithm(s) would be more appropriate for the binary and multi-label classification problem. The approach of spot checking the algorithms early for accuracy led to the determination of a smaller set of algorithms with higher accuracy (e.g. Gadient Boosting and Random Forest) for a deeper dive examination, e.g. use of a k-fold cross-validation approach in classifying the CDR label.
- The neural network benchmark model accuracy of 78% for binary classification was exceeded by the classification accuracy of the main output of this study, the trained Gradient Boosting and Random Forest classification models. This builds confidence in the latter model for further training with new data and further classification use for new patients.

Free-Form Visualization (Learning Curves)

Learning curves are provided later in this notebook. These curves provide visual demonstration of the dependence of the training and validation accuracy on the size of the dataset (number of records or samples) for the merged OASIS dataset for a number of classifiers such as Random Forest, Gradient Boosting. This visualization clarifies the the number of samples required for a particular prediction accuracy threshold, and qualitatively/visually demonstrates the level of overfitting in the training dataset.

Appendix 1- Rerun the Binary Classification cases with Reduced Features.

Rerun the machine learning algorithm classification cases with reduced features (4 columns, Age, MMSE, NWBV, and ASH), and compare the metrics.

In [53]:
```
# Split-out validation dataset
array = dfoas_merge.values
X = array[:,[3, 6, 9, 10]] #Feature Extraction with RFE
Y = array[:,7] # all rows and CDR column used for AUC ROC calculations later
Y_float64 = array[:,7] # all rows and CDR column used for AUC ROC calculatio
ns later
Y=Y.astype(np.str)
validation_size = 0.20
seed = 12345
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,
test_size=validation_size, random_state=seed)
#print (X)
#print (Y)
```

In [54]:
```
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RFC', RandomForestClassifier()))
# evaluate each model in turn
results = []
names = []
print('(', 'Model, ', 'Cross-Validation Accuracy: Mean, Stdev',')')
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='a
ccuracy')
    results.append(cv_results)
    names.append(name)
    msg = (name, format(cv_results.mean(), '.2f'), format(cv_results.std(), '.
2f'))
    print(msg)
```

```
( Model,  Cross-Validation Accuracy: Mean, Stdev )
('LR', '0.78', '0.06')
('LDA', '0.81', '0.04')
('KNN', '0.80', '0.05')
('CART', '0.82', '0.05')
('NB', '0.84', '0.06')
('SVM', '0.82', '0.06')
('GBM', '0.85', '0.04')
('RFC', '0.88', '0.04')
```

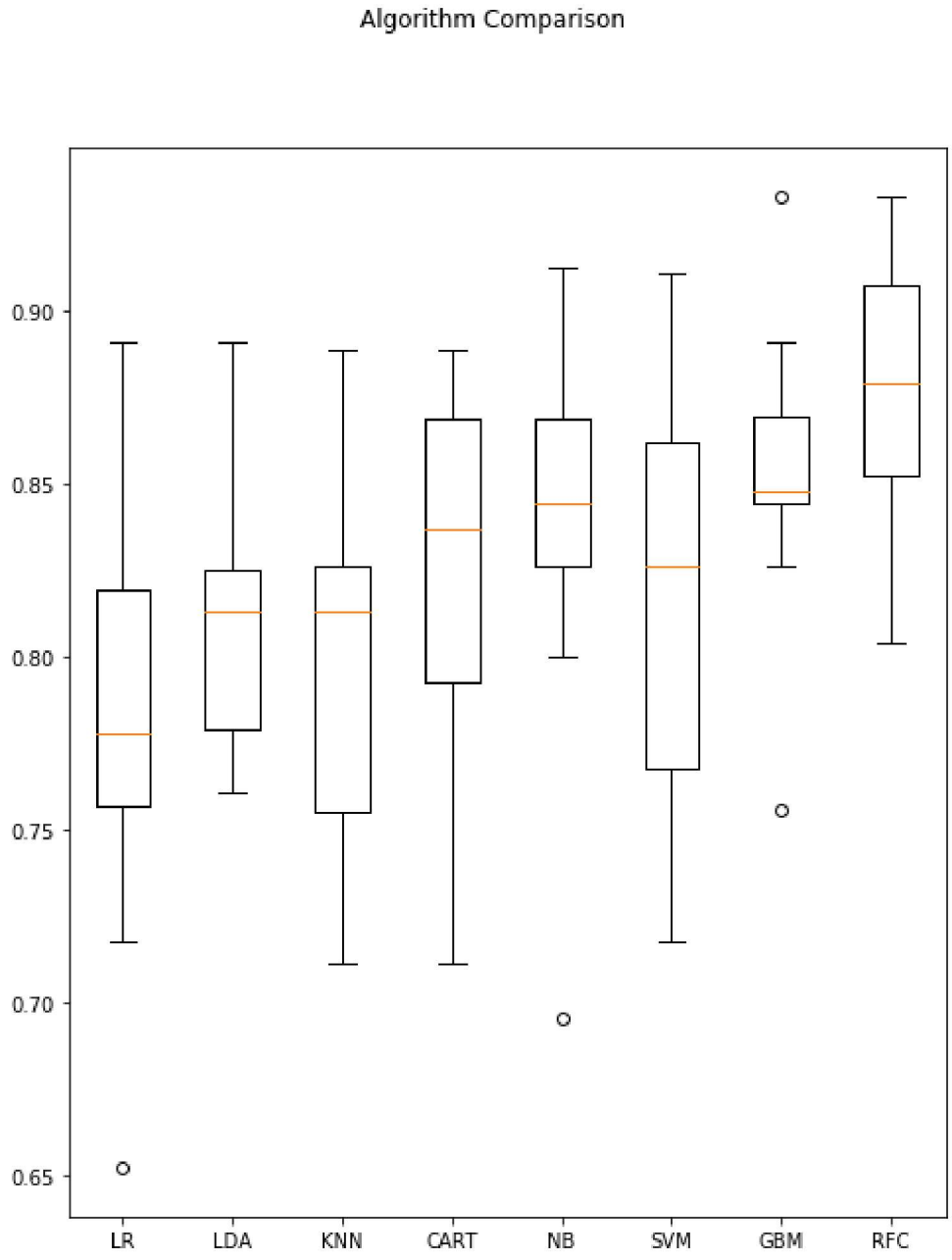
Results comparison: For the reduced feature set, accuracy increased from low sixty percent to about eighty percent for KNN and SVM classifiers. Accuracy remained about the same for other classifiers spot checked. Gradient Boosting Machine and Random Forest classifier remained the accuracy leaders.

In [55]:
```
from sklearn.ensemble import GradientBoostingClassifier
classifier = GradientBoostingClassifier(max_depth=4)
classifier.fit(X_train, Y_train)
X_test = X_validation
Y_test = Y_validation
prediction = classifier.predict(X_test)
print (classifier.score(X_train, Y_train))
print (classifier.score(X_test, Y_test))
```

```
0.997807017544
0.842105263158
```

We see that cross-validation accuracy with reduced features determined using RFE, (4 columns vs. 8) has a small increase (0.89 vs. 0.88)

In [56]:
```
# Compare Algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show();
```



The box plots show that the accuracy of all the classifiers (LR through RFC) increased and the high accuracy classifiers GBM and RFC retained their accuracy.Also for the Random Forest classifier, the cross-validated prediction accuracy remained nearly same as that for the full feature set.

In [57]:
```
# Make predictions on validation dataset using the RandomForestclassifier (Reduced feature,4 column dataset)

from sklearn.ensemble import RandomForestClassifier
from time import time
tstart = time()
RFC=RandomForestClassifier()
RFC.fit(X_train, Y_train)
predictions = RFC.predict(X_validation)
predictions_RFC=predictions
print("*Accuracy score: ", accuracy_score(Y_validation, predictions), "\n")
print("*Confusion Matrix: ")
print(confusion_matrix(Y_validation, predictions))
print("*Classification Report: ")
print(classification_report(Y_validation, predictions))
tend = time()
print ('Time required (in milliseconds)', (tend-tstart))
```

```
*Accuracy score:  0.824561403509

*Confusion Matrix:
[[67  6]
 [14 27]]
*Classification Report:
             precision    recall  f1-score   support

        0.0       0.83      0.92      0.87        73
        1.0       0.82      0.66      0.73        41

avg / total       0.82      0.82      0.82       114

Time required (in milliseconds) 0.1040034294128418
```

Create a PIPELINE that extracts features from the data then creates a model (from Reference 24)

In [58]:
```
# Create a PIPELINE that extracts features from the data using PCA, then creat
es a model (from Reference 24, 26)
# Uses GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
# X, Y as defined/calculated previously
# create feature union
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=4)))
feature_union = FeatureUnion(features)
# create pipeline
estimators = []
estimators.append(('feature_union', feature_union))
estimators.append(('GBC', GradientBoostingClassifier()))
model = Pipeline(estimators)
# evaluate pipeline
kfold = KFold(n_splits=10, random_state=899)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.863157894737
```

Appendix 2 - Plots of Learning Curves for Various Classifiers

In [59]:
```
# Binary Classification (CDR values 0 and 1)
# Split-out validation dataset
array = dfoas_merge.values
#dfoas_merge_doubled = dfoas_merge.append(dfoas_merge, ignore_index=True)
#array = dfoas_merge_doubled.values
Xtp = array[:,[1, 3, 4, 5, 6, 8, 9, 10]]
Ytp = array[:,7] # all rows and CDR column used for AUC ROC calculations later
Ytp_float64 = array[:,7] # all rows and CDR column used for AUC ROC calculatio
ns later
Ytp=Ytp.astype(np.str)
validation_size = 0.20
seed = 1234567
X_train, X_validation, Y_train, Y_validation = train_test_split(Xtp, Ytp,
test_size=validation_size, random_state=seed)
print (Xtp)
print (Ytp)
```

```
[[1 74 2.0 ..., 1344 0.743 1.306]
 [1 55 4.0 ..., 1147 0.81 1.531]
 [1 73 4.0 ..., 1454 0.708 1.207]
 ...,
 [1 61 13.0 ..., 1319 0.8009999999999999 1.331]
 [1 63 13.0 ..., 1327 0.7959999999999999 1.3230000000000002]
 [1 65 13.0 ..., 1333 0.8009999999999999 1.317]]
['0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '0.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0' '0.0' '0.0'
 '1.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '0.0'
 '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0'
 '0.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '1.0'
 '1.0' '1.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0'
 '1.0' '0.0' '1.0' '1.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0'
 '0.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0'
 '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '0.0' '1.0' '1.0' '1.0' '0.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '1.0' '1.0' '1.0'
 '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '1.0'
 '1.0' '0.0' '0.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0'
 '0.0' '1.0' '1.0' '1.0' '1.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '0.0'
 '1.0' '1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0'
 '0.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '0.0' '0.0' '1.0' '1.0'
 '1.0' '1.0' '1.0' '0.0' '0.0' '0.0']
```

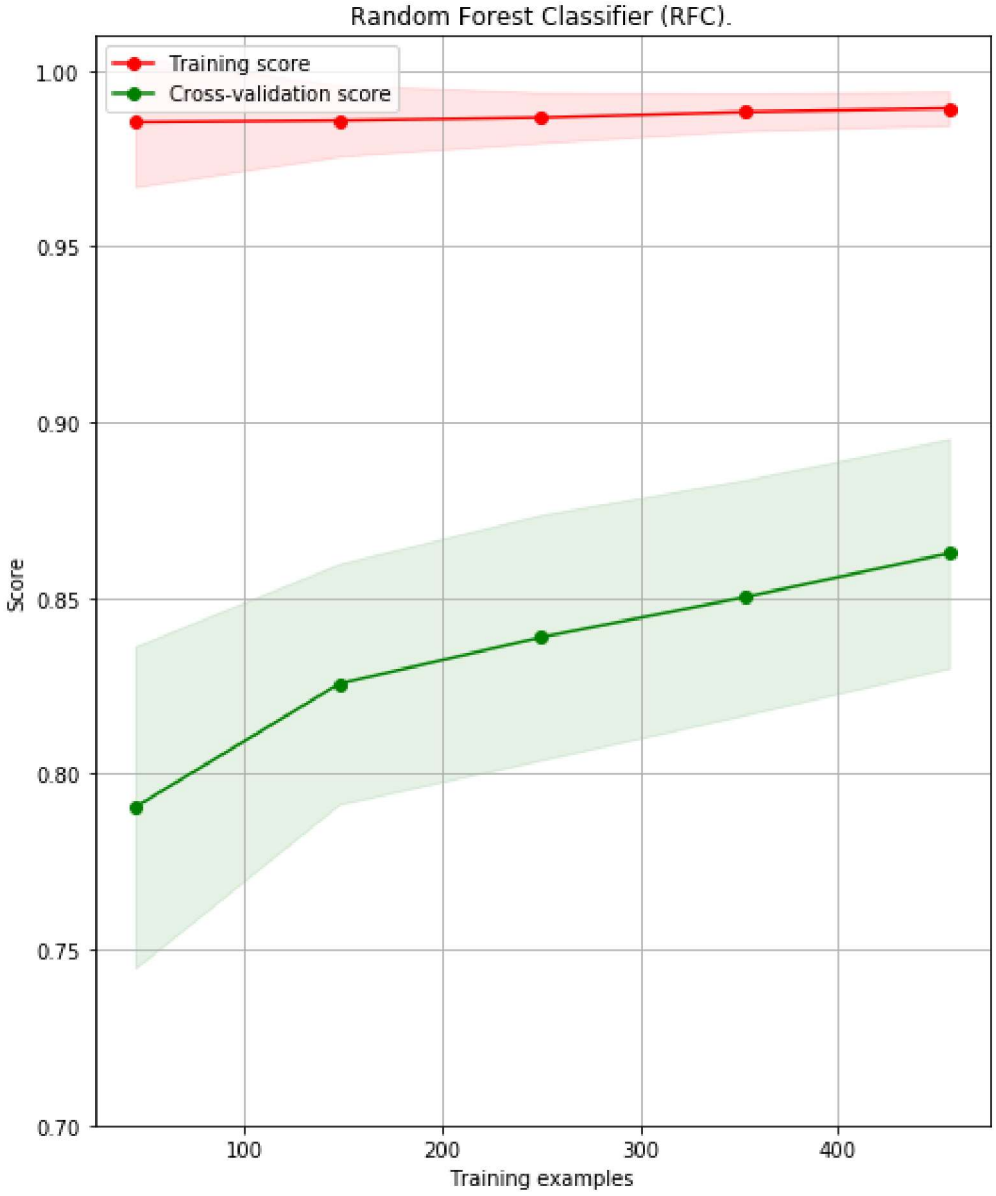
Learning Curves for the merged dataset for Random Forest, Gradient Boosting, Naive Bayes, and the Decision Tree classifiers.

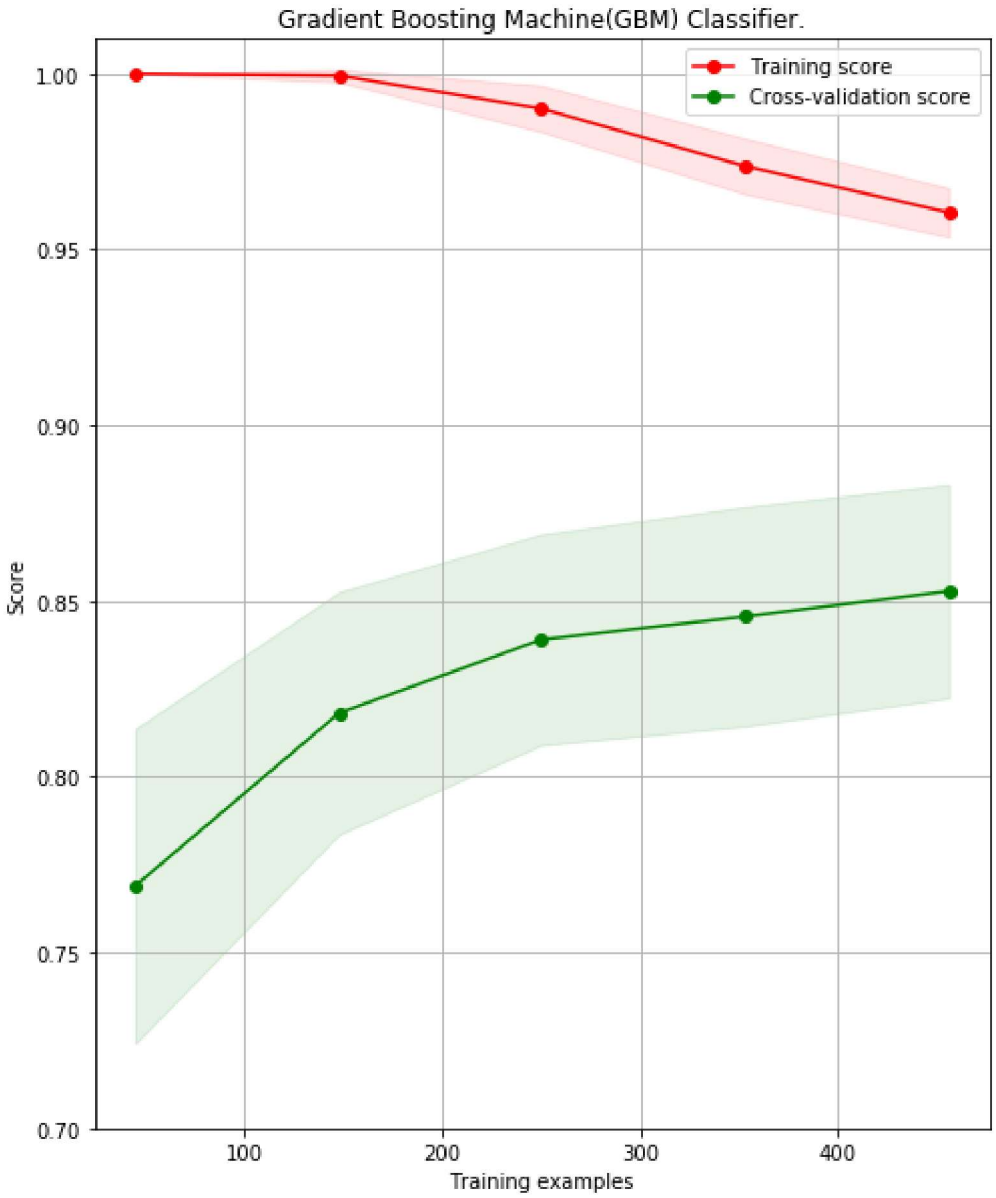
In [60]:
```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
#from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit


def plot_learning_curve(estimator, title, Xtp, Ytp, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable, optional
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:
          - None, to use the default 3-fold cross-validation,
          - integer, to specify the number of folds.
          - An object to be used as a cross-validation generator.
          - An iterable yielding train/test splits.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : integer, optional
        Number of jobs to run in parallel (default 1).
    """
```

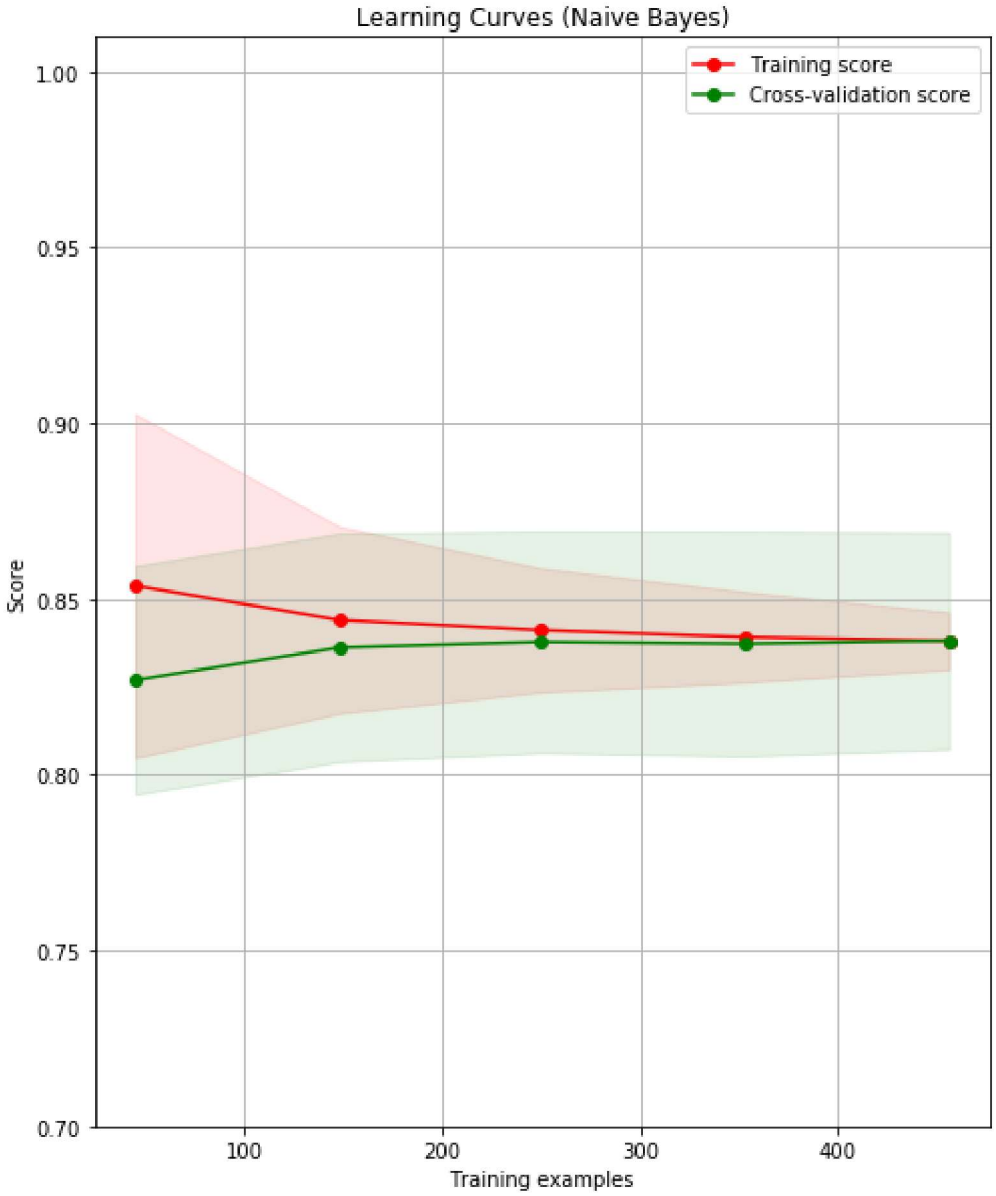
```
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, Y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

title = "Random Forest Classifier (RFC)."
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = RandomForestClassifier()
plot_learning_curve(estimator, title, X, Y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)


title = "Gradient Boosting Machine(GBM) Classifier."
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = GradientBoostingClassifier()
plot_learning_curve(estimator, title, X, Y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)


title = "Learning Curves (Naive Bayes)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = GaussianNB()
plot_learning_curve(estimator, title, X, Y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)

title = "Decision Tree Classifier (DTC)."
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)
```

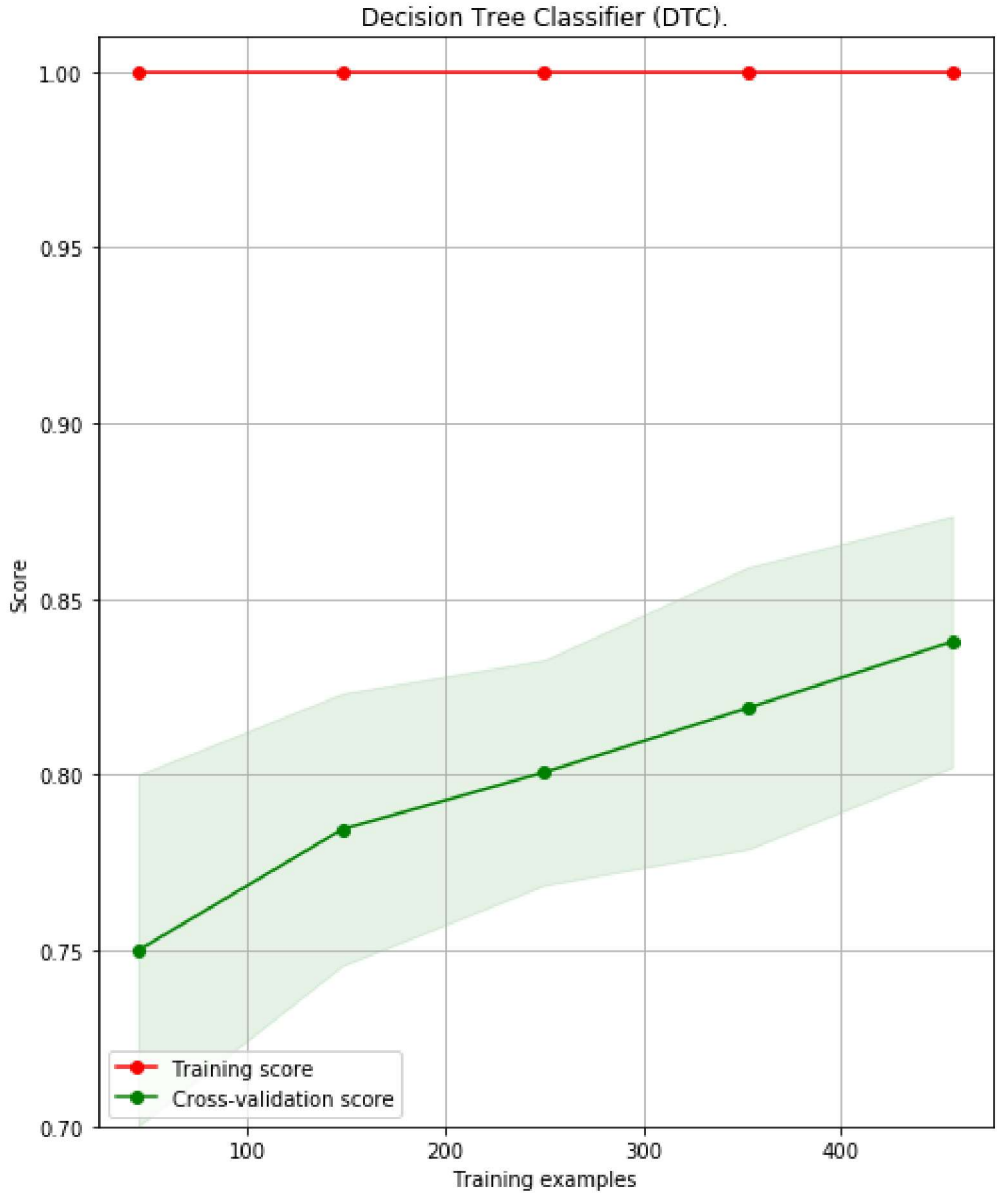
```
estimator = DecisionTreeClassifier()
plot_learning_curve(estimator, title, X, Y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)


plt.show();
```

```
Automatically created module for IPython interactive environment
```




Gradient Boosting Machine(GBM) Classifier.
Training score
Cross-validation score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Score
100
200
300
400
Training examples


Learning Curves (Naive Bayes)
Training score
Cross-validation score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Score
100
200
300
400
Training examples


Decision Tree Classifier (DTC).
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Score
100
200
300
400
Training examples
Training score
Cross-validation score

In [ ]: