Model comparison

This Notebook's key objective is to compare different models applied on datasets produced with different data imputation strategies.

In [5]:

```
#importing libraries
%matplotlib inline
import os
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from itertools import product
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.preprocessing import scale
from sklearn.model_selection import learning_curve
from glob import glob
from sklearn import preprocessing
from matplotlib_venn import venn3

import statsmodels.api as sm
from statsmodels.regression.linear_model import OLS

# import custom dependencies
import ADNI_utilities as utils
```

Reading and preparing the data

We have 5 datasets that we will use for modeling. Three of them use model-based imputation with different thresholds (100%, 50%, and 30%) for missing values. Two use mean/mode based imputation at 30% and 50% thresholds.

In [2]:

```
#Initializing the data strucuture that will hold the test scores from different models
classifiers = {}
classifiers['LogisticRegression'] = 'Logistic'
classifiers['KNeighborsClassifier'] = 'kNN'
classifiers['DecisionTreeClassifier'] = 'Decision Tree'
classifiers['BaggingClassifier'] = 'Bagging'
classifiers['AdaBoostClassifier'] = 'Boosting'
classifiers['RandomForestClassifier'] = 'Random Forest'

filenames = {}
filenames['mean - 30pct'] = 'data_mean_upto_30pct_missing.csv'
filenames['mean - 50pct'] = 'data_mean_upto_50pct_missing.csv'
filenames['model - 30pct'] = 'data_modeled_upto_30pct_missing.csv'
filenames['model - 50pct'] = 'data_modeled_upto_50pct_missing.csv'
filenames['model - 100pct'] = 'data_modeled_upto_100pct_missing.csv'

result_container_multi = pd.DataFrame(index=list(filenames.keys()), columns=list(classifier
result_container_binary = pd.DataFrame(index=list(filenames.keys()), columns=list(classifie
estimators = {f_name:[] for f_name in list(filenames.values())}
```

In [3]:

```
#Setting up common parameters/config
import warnings
warnings.filterwarnings('ignore')

data_path = '../data/Imputed/'
resp_variable = 'DX_FINAL'
resp_vars = ['DXCOMB', 'DX_CHANGE', 'DX_FINAL', 'DX_BASE', 'DX_bl']
testsize = 0.2
rs = 42 # set random state so results are repeatable
run_binary = 1 #Set this variable to 1 if models with binary response variable are to be ru
```

We will try a variety of models on the imputed data. We've decided to use KNN, DecisionTree, LogisticRegression, AdaBoost, Bagging, and Random Forest. The function below will perform a Grid CV search over all imputed datasets and accumulate the results of the best models.

In [4]:

```
def train_classifier_cv(estimator, param_grid, cv=5):
    """Trains the given estimator on each design matrix using grid search and
    cross validation. The best estimator and score are saved.

    # Arguments
        estimator: The estimator/classifier to train/score
        param_grid: the parameters to be used in the grid search
        cv: number of folds to be used for cross validation
    """
    for file_key in list(filenames.keys()):
        #Loading data
        file_nm = filenames[file_key]
        file_w_path = data_path + file_nm
        est_name = estimator.__repr__().split('(')[0]

        df = pd.read_csv(file_w_path, index_col='RID')
        df = df.drop(columns=['CDRSB','mPACCtrailsB', 'mPACCdigit'])

        if 'modeled' in file_nm:
            df = utils.reverse_one_hot(resp_vars, df)

        df_train, df_test = train_test_split(df, test_size=testsize, shuffle=True, random_s

        y_train_multi = df_train[resp_variable]
        if (run_binary == 1 and not 'modeled' in file_nm):
            y_train_bin = df_train[resp_variable].apply(lambda x: 1 if x == 3 else 0)

        X_train = df_train.drop(resp_vars, axis=1).select_dtypes(['number'])

        y_test_multi = df_test[resp_variable]
        if (run_binary == 1 and not 'modeled' in file_nm):
            y_test_bin = df_test[resp_variable].apply(lambda x: 1 if x == 3 else 0)

        X_test = df_test.drop(resp_vars, axis=1).select_dtypes(['number'])

        #Running the model and storing results
        gs = GridSearchCV(estimator, param_grid=param_grid, cv=cv, n_jobs=-1,
                          return_train_score=True, iid=False)
        gs.fit(X_train, y_train_multi)
        score = gs.score(X_test, y_test_multi)
        result_container_multi.loc[file_key, classifiers[est_name]] = score
        estimators[file_nm].append((score, gs.best_estimator_))

        if (run_binary == 1 and not 'modeled' in file_nm):
            gs = GridSearchCV(estimator, param_grid=param_grid, cv=cv, n_jobs=-1,
                              return_train_score=True, iid=False)
            gs.fit(X_train, y_train_bin)
            result_container_binary.loc[file_key, classifiers[est_name]] = gs.score(X_test,
```

In [5]:

```
# kNN
knn = KNeighborsClassifier(n_jobs=-1)
param_grid = {'n_neighbors': [2, 5, 10, 20, 50, 75, 100]}

train_classifier_cv(knn, param_grid)
```

In [6]:

```
# LogisticRegression
logr = LogisticRegression(multi_class="ovr", penalty='l2', max_iter=1000)
logr_params = {'C':10.0 ** np.arange(-4,4)}

train_classifier_cv(logr, logr_params)
```

In [7]:

```
# Decision Tree
dt_clf = DecisionTreeClassifier()
dt_clf_params = {'max_depth':[2, 3, 5, 10, 20],
              'min_samples_leaf': [1, 2, 4, 6, 20]}

train_classifier_cv(dt_clf, dt_clf_params)
```

In [8]:

```
# Bagging
dt_clf = DecisionTreeClassifier()
bag_clf = BaggingClassifier(dt_clf, n_jobs=-1, n_estimators=100)
bag_clf_params = {'base_estimator__max_depth':[2, 3, 5, 10, 20],
          'base_estimator__min_samples_leaf': [1, 2, 4, 6, 20]}

train_classifier_cv(bag_clf, bag_clf_params)
```

In [9]:

```
# Boosting
dt_clf = DecisionTreeClassifier()
ada_clf = AdaBoostClassifier(dt_clf, n_estimators=100, learning_rate=0.75)
ada_clf_params = {'base_estimator__max_depth':[2, 3, 5, 10, 20],
          'base_estimator__min_samples_leaf': [1, 2, 4, 6, 20]}

train_classifier_cv(ada_clf, ada_clf_params)
```

In [10]:

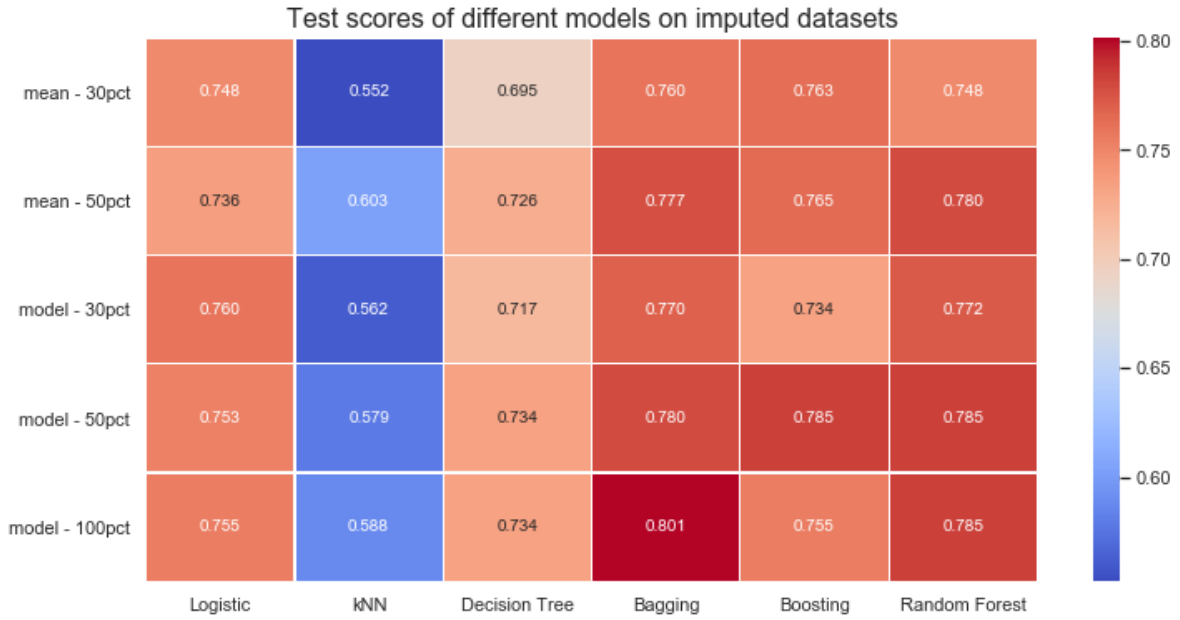
```
# Random Forest
rf_clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
rf_clf_params = {'max_depth':[2, 3, 5, 10, 20],
      'min_samples_leaf': [1, 2, 4, 6, 20]}

train_classifier_cv(rf_clf, rf_clf_params)
```

Performance comparison

This section will present the performance differences across the models tested by our group.

In [11]:

```
#Plotting the results as a heat map
fig, ax = plt.subplots(1,1, figsize=(12,6))

ax = sns.heatmap(result_container_multi.astype('float64'), annot=True, linewidths=0.1, fmt=
ax.set_title('Test scores of different models on imputed datasets', fontsize=16);
plt.show()
```



The heatmap above represents a brief summary of our findings. It shows test scores for the tested models on datasets built with different imputation strategies. The first finding is that kNN and Logistic Classification are consistently outperformed by any strategy using Decision Trees - both the simple trees and the ensemble methods using trees.

Among the methods using decision trees, Bagging shows the greatest performance - achieving more than 80% accuracy regardless of the imputation strategy. Especially for Bagging, modeling seems like a better imputation strategy than using the mean of the initial values found on the data. Although this conclusion is not true with all models, it seems generally that on our problem, model-based imputation slightly improves model performance vs using the mean/mode imputation.

To further evaluate the our design matrices, we will also train the top two estimators on the baseline ADNI Merge dataset.

In [45]:

```
# Try Bagging on the baseline set
estimators = utils.get_ADNI_baseline(estimators)
```

In [46]:

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
models = estimators['baseline_mean_upto_50pct_missing.csv']
models = [model[1] for model in models]
for ax, model in zip(axes, models):
    X, y = utils.get_ADNI_baseline_data('../data/Imputed/baseline_mean_upto_50pct_missing.c
    train_sizes, train_scores, test_scores = learning_curve(
    model, X, y, cv=5, n_jobs=-1)
    train_scores_mn = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mn = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)


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


    ax.set_title(model.__repr__().split('(')[0], size=15)
    ax.set_xlabel("Training samples", size=14)
    ax.set_ylabel("Scores", size=14)
    ax.legend(fontsize=14)

fig.suptitle('Train/Test scores on ANDI Merged baseline data', size=18, y=1.02)
plt.show()
```

We will now save the model results to disk for further analysis.

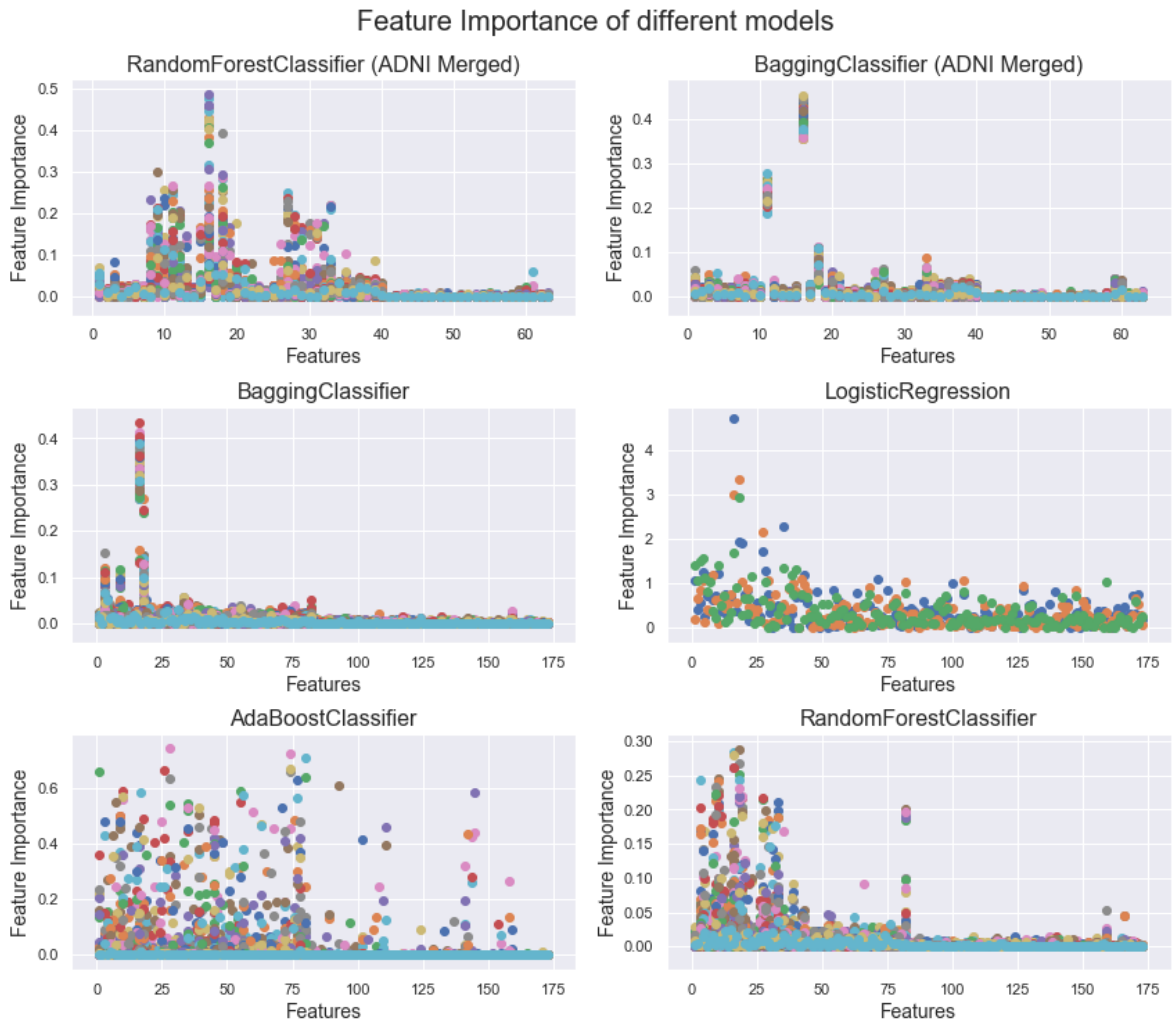
In [47]:

```
from joblib import dump, load

# Save the feature importance information from the ensemble classifiers
model_path = '../data/Models/Feature_Importance/'
for key in list(estimators.keys()):
    features = utils.get_feature_names(data_path, key, resp_vars)
    key_sub = key[key.index('_')+1:].split('pct')[0]
    bl = ''
    if 'baseline' in key:
        bl = 'baseline_'

    for score, model in estimators[key]:
        model_nm = model.__repr__().split('(')[0]
        dump(model, f'../data/Models/{model_nm}_{bl}{key_sub}.joblib')

        # Check to see if estimators_ attribute exists (ensemble classifiers)
        if getattr(model, 'estimators_', None) is not None:
            df = pd.DataFrame(columns=features)
            for estimator in model.estimators_:
                df = df.append(pd.Series(estimator.feature_importances_, index=df.columns),
                               ignore_index=True)
            df.to_csv(model_path + f'{model_nm}_{bl}{key_sub}.csv')

        # Check to see if coef_ attribute exists
        elif getattr(model, 'coef_', None) is not None:
            df = pd.DataFrame(data=model.coef_, columns=features, index=model.classes_)
            df.to_csv(model_path + f'{model_nm}_{bl}{key_sub}.csv')
```

Next we will explore the distribution of feature importance across the different models.

In [48]:

```
fig, axes = plt.subplots(3, 2, figsize=(12,10))

feature_import = glob('../data/Models/Feature_Importance/*baseline_mean_upto_50.csv') + glo
    '../data/Models/Feature_Importance/*_modeled_upto_50.csv')

for fi, ax in zip(feature_import, axes.ravel()):
    df = pd.read_csv(fi, index_col=0)
    x = np.arange(1, df.shape[1]+1)

    bl = ''
    if 'baseline' in fi:
        bl = ' (ADNI Merged)'
    title = fi.split('/')[-1].split('_')[0] + bl
    ax.set_xlabel('Features', size=14)
    ax.set_ylabel('Feature Importance', size=14)
    ax.set_title(title, size=16)

    for i in df.index:
        ax.scatter(x, np.abs(df.loc[i]))

fig.tight_layout()
fig.suptitle("Feature Importance of different models", size=20, y=1.03)
plt.show()
```

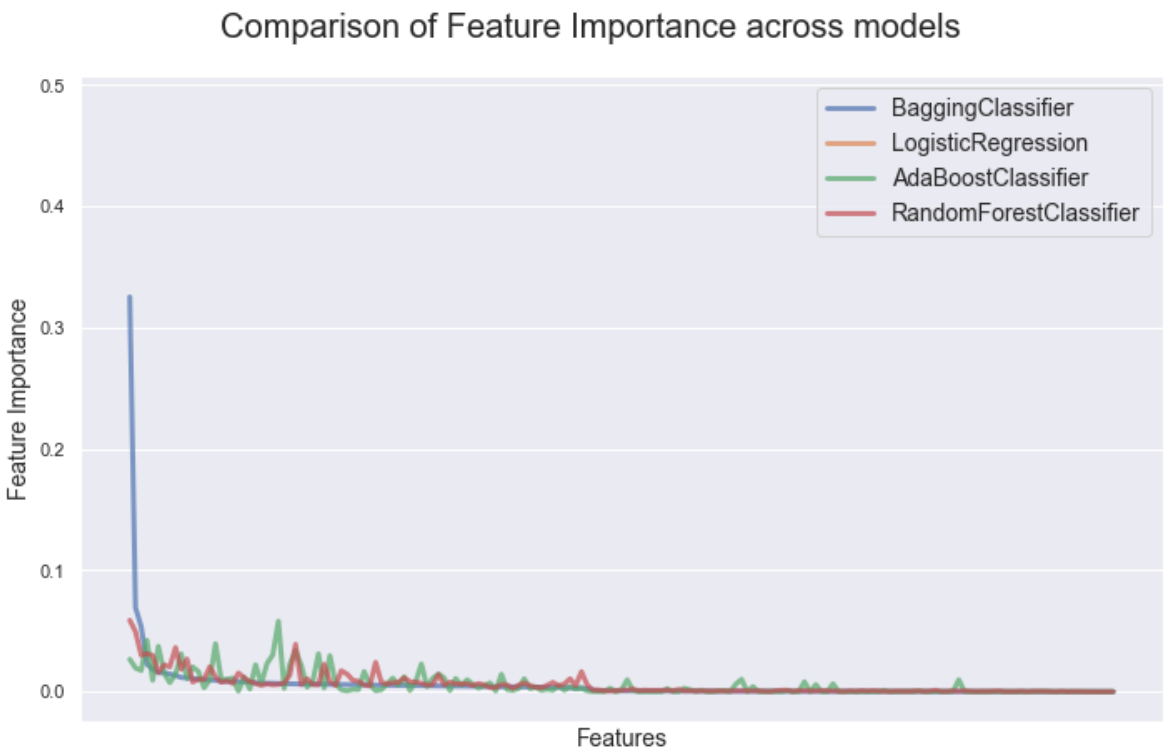


From the charts above, we see that Bagging ensemble methods tend to smooth out and only keep the most significant predictors. Boosting tends to be sensitive to the most predictors, however its score doesn't quite

match Bagging or RandomForest. RandomForest is in between in terms of feature importance, but the score is similar to Bagging. LogisticRegression gives `num categories *` feature-importance scores for each feature, which makes it a little harder to interpret. We didn't include KNN because the performance was so poor, and there's no easy way to get those metrics for KNN.

In [37]:

```
# Re-index based on feature importance Hi-Lo from Bagging results
bag_df = pd.read_csv(
    '../data/Models/Feature_Importance/BaggingClassifier_modeled_upto_50.csv',
    index_col=0)

idx = bag_df.mean().sort_values(ascending=False).index
scaler = MinMaxScaler()

# Read in feature importance matrices that were previously saved
feature_import = glob('../data/Models/Feature_Importance/*_modeled_upto_50.csv')

fig, ax = plt.subplots(1, 1, figsize=(10,6))
legend = []

for fi in feature_import:
    df = pd.read_csv(fi, index_col=0)

    estimator = fi.split('/')[-1].split('_')[0]
    # Re-index based on feature importance
    df = df.reindex_axis(idx, axis=1)

    if estimator == 'LogisticRegression':
        df = scaler.fit_transform(df.abs())
    else:
        df = pd.DataFrame(df.abs())

    df = df.mean()
    legend.append(estimator)

    ax.plot(df, lw=3, alpha=.7)

ax.set_xlabel('Features', size=14)
ax.set_ylabel('Feature Importance', size=14)
ax.legend(legend, fontsize=14)
ax.set_xticklabels('')
ax.set_xticks([])
fig.tight_layout()
fig.suptitle("Comparison of Feature Importance across models", size=20, y=1.05)
plt.show()
```



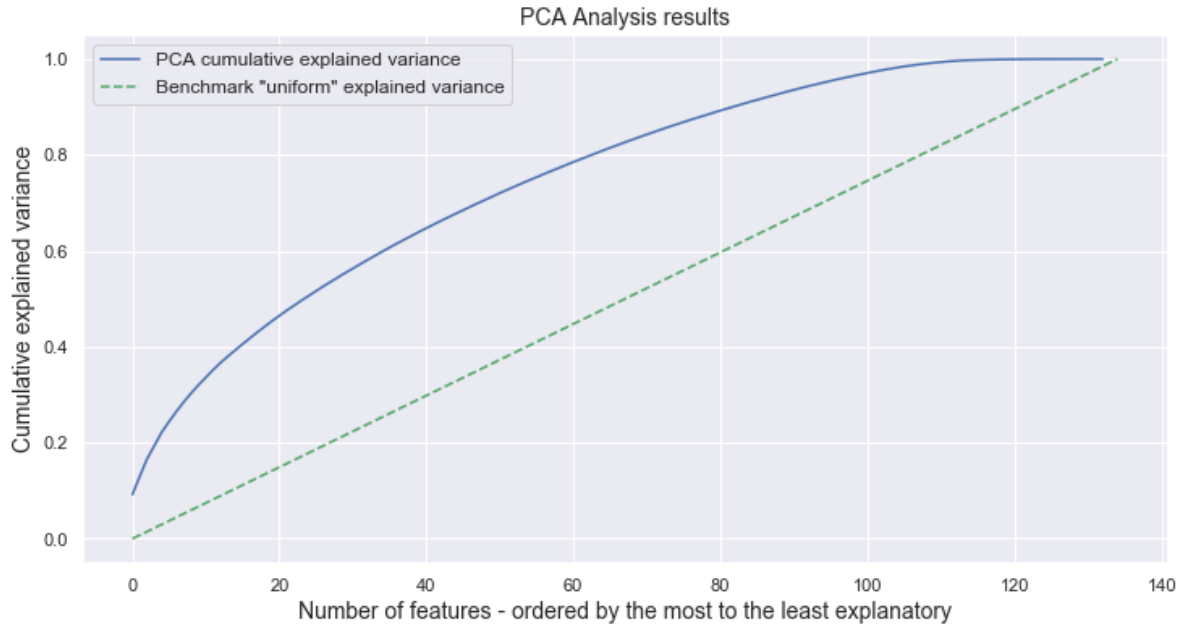
The plot above shows averaged feature importance scores of each classifier together. For easier comparison, the features have been ordered by importance (descending) based on the results from the BaggingClassifier. You can see that the BaggingClassifier puts higher importance on fewer features relative to the other classifiers.

Dimensionality reduction

We have also explored the possibility of using PCA to reduce the dimensionality of our dataset and reach additional conclusions regarding our dataset and regarding AD diagnosis prediction.

In [36]:

```
#PCA

#Reading the "strictest" dataset to apply PCA techniques
df = pd.read_csv(data_path + 'data_mean_upto_30pct_missing.csv')

df_train, df_test = train_test_split(df, test_size=testsize, shuffle=True, random_state=rs)
y_train_multi = df_train[resp_variable]
X_train = df_train.drop(resp_vars, axis=1).select_dtypes(['number'])
X_train = scale(X_train)

#Running the PCA routine
pca = PCA()
pca.fit(X_train)

#Plotting the results
fig, ax = plt.subplots(1,1, figsize=(12,6))

plt.plot(np.cumsum(pca.explained_variance_ratio_), label='PCA cumulative explained variance
plt.plot(np.linspace(0,1,135), 'g--', label='Benchmark "uniform" explained variance')
plt.legend(fontsize=12)
ax.set_xlabel('Number of features - ordered by the most to the least explanatory', fontsize
ax.set_ylabel('Cumulative explained variance', fontsize=14)
ax.set_title('PCA Analysis results', fontsize=14);
```



As one can see in the plot above, there is not a strong explanatory power concentration on any principal component. Out of 133 components on the select dataset, more than 80 are needed to achieve 90% variance explanation. The top 20 components explain less than 50% of the variance. It is interesting to notice that, after the 20th and before the 100th component, the blue line is almost "parallel" to the green benchmark line - showing that there is very limited explanatory power concentration.

For this reason, added to the fact that similar results hold for all imputation strategies, we have decided not to use PCA as a tool to further improve our models.

Comparison of the top features from each model

What is the consensus between what the different models call the most important features of the dataset?

To answer this question, we will look at the top 100 important features of every model and plot overlaps between these important features.

In [6]:

```
rf_50pc_modeled = pd.read_csv('../data/Models/Feature_Importance/RandomForestClassifier_mod
ab_50pc_modeled = pd.read_csv('../data/Models/Feature_Importance/AdaBoostClassifier_modeled
bg_50pc_modeled = pd.read_csv('../data/Models/Feature_Importance/BaggingClassifier_modeled_
lr_50pc_modeled = pd.read_csv('../data/Models/Feature_Importance/LogisticRegression_modeled
```

In [7]:

```
#lets mean across the estimators for each model and put values in dataframe
rf = rf_50pc_modeled.mean()
ab = ab_50pc_modeled.mean()
lr = lr_50pc_modeled.mean()
bg = bg_50pc_modeled.mean()
features_50pc = pd.concat([rf.rename('RandomForest'), ab.rename('Adaboost'),  lr.rename('Lo
features = features_50pc.index
features_50pc.head()
```

Out[7]:

	RandomForest	Adaboost	LogReg	Bagging
AGE	0.005879	0.022760	-2.790952	0.002792
PTEDUCAT	0.003593	0.001057	0.145347	0.002356
FDG	0.022923	0.022680	-0.027745	0.029754
AV45	0.020883	0.014378	-0.347403	0.017745
ABETA	0.020470	0.054085	-0.514659	0.011245

In [8]:

```
x = features_50pc.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
features_50pc_scaled = min_max_scaler.fit_transform(x)
features_50pc_scaled = pd.DataFrame(features_50pc_scaled)
features_50pc_scaled.columns = ['RandomForest', 'Adaboost', 'Logistic Regression', 'Bagging
```

In [9]:

```
features_50pc_scaled.describe()
```

Out[9]:

	RandomForest	Adaboost	Logistic Regression	Bagging
count	176.000000	176.000000	176.000000	176.000000
mean	0.075221	0.076072	0.597343	0.014743
std	0.136933	0.163522	0.121599	0.082846
min	0.000000	0.000000	0.000000	0.000000
25%	0.006512	0.000006	0.579304	0.000294
50%	0.014270	0.003665	0.606138	0.001842
75%	0.082136	0.078188	0.630904	0.010384
max	1.000000	1.000000	1.000000	1.000000

In [10]:

```
features_50pc_scaled['feature'] = features_50pc.index
```

In [11]:

```
features_50pc_scaled.head()
```

Out[11]:

	RandomForest	Adaboost	Logistic Regression	Bagging	feature
0	0.077825	0.304721	0.291966	0.007244	AGE
1	0.047569	0.014145	0.625673	0.006113	PTEDUCAT
2	0.303470	0.303656	0.606002	0.077202	FDG
3	0.276460	0.192507	0.569673	0.046042	AV45
4	0.271000	0.724132	0.550664	0.029178	ABETA

In [12]:

```
#lets order the features and select the top 100 features of each of the model.

features_50pc_ordered_rf= features_50pc_scaled.sort_values(by = 'RandomForest', axis=0, asc
features_50pc_ordered_rf = features_50pc_ordered_rf.head(100)

features_50pc_ordered_bg = features_50pc_scaled.sort_values(by = 'Bagging', axis=0, ascendi
features_50pc_ordered_bg = features_50pc_ordered_bg.head(100)

features_50pc_ordered_ab = features_50pc_scaled.sort_values(by = 'Adaboost', axis=0, ascenc
features_50pc_ordered_ab = features_50pc_ordered_ab.head(100)

features_50pc_ordered_lr = features_50pc_scaled.sort_values(by = 'Logistic Regression', axi
features_50pc_ordered_lr = features_50pc_ordered_lr.head(100)
```

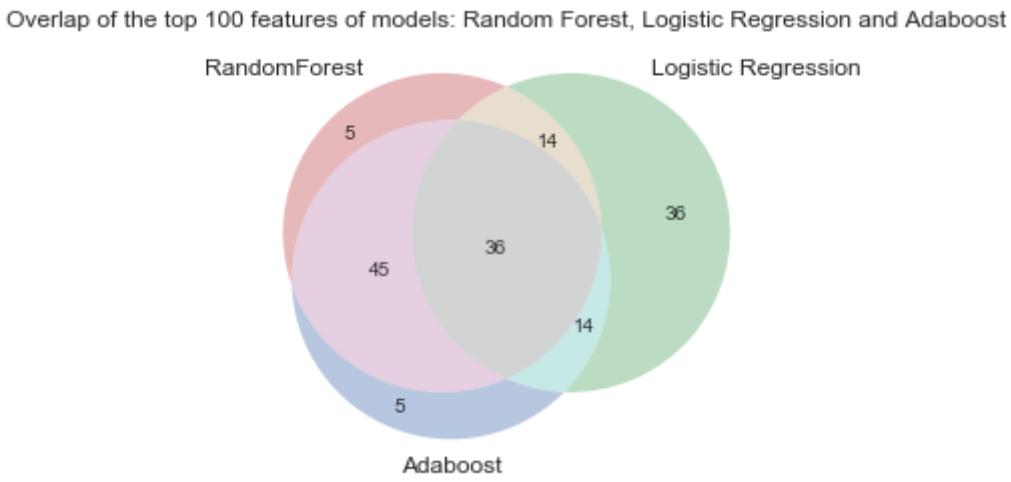
We have ordered all the feature importances for each model from the most to least important. Next, we will look into what is the overlap between these top features by way of a Venn diagram.

In [13]:

```
setLabels2 = ['RandomForest','Logistic Regression', 'Adaboost']
plt.figure()
ax2 = plt.gca()
v2 = venn3([set(features_50pc_ordered_rf.feature), set(features_50pc_ordered_lr.feature),
       set(features_50pc_ordered_ab.feature)], set_labels = setLabels2, ax = ax2)
ax2.set_title('Overlap of the top 100 features of models: Random Forest, Logistic Regressio
```

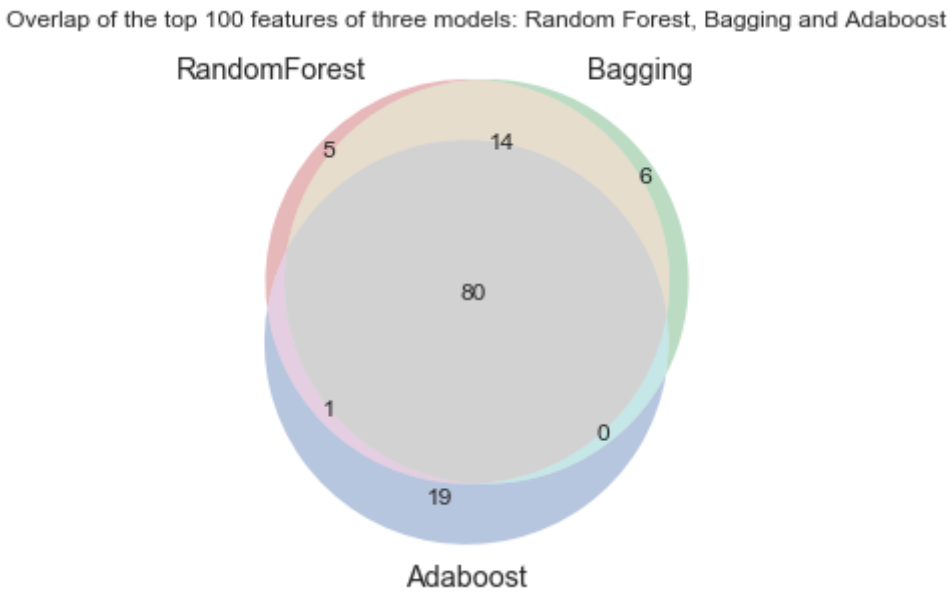
Out[13]:

```
Text(0.5,1,'Overlap of the top 100 features of models: Random Forest, Logist
ic Regression and Adaboost')
```



In [14]:

```
sns.set()
setLabels1 = ['RandomForest','Bagging', 'Adaboost']
plt.figure(figsize=(5, 5))
ax1 = plt.gca()
v1 = venn3([set(features_50pc_ordered_rf.feature), set(features_50pc_ordered_bg.feature),
       set(features_50pc_ordered_ab.feature)], set_labels = setLabels1, ax = ax1)

ax1.set_title('Overlap of the top 100 features of three models: Random Forest, Bagging and

set1 = set(features_50pc_ordered_ab.feature)
set2 = set(features_50pc_ordered_rf.feature)
set3 = set(features_50pc_ordered_bg.feature)

set4 = set1.intersection(set2)
set5 = set4.intersection(set3)
common = list(set5)
```



Of the four models we are testing, we find a large degree of congruence between the top features identified by Random Forest, AdaBoost and Bagging. However, logistic regression shows the least overlap in what it calls the most important features in the dataset. This, together with the interpretability of the features led us to converge on the the three ensemble methods as the most reliable models.

In [16]:

```
selected_common_features = features_50pc_ordered_rf.loc[features_50pc_ordered_rf['feature']
```

In [17]:

```
selected_common_features = selected_common_features.drop(['Logistic Regression'], axis =1)
selected_common_features.head()
selected_common_features.to_csv('Selected_common_features.csv')
```

In [18]:

```
features_50pc_ordered_rf_melt = pd.melt(features_50pc_ordered_rf, id_vars="feature", var_na
features_50pc_ordered_lr_melt = pd.melt(features_50pc_ordered_lr, id_vars="feature", var_na
features_50pc_ordered_ab_melt = pd.melt(features_50pc_ordered_ab, id_vars="feature", var_na
features_50pc_ordered_bg_melt = pd.melt(features_50pc_ordered_bg, id_vars="feature", var_na
```

Is the relative importance assigned by each model to the top features similar?

Bagging and random forest assign similar relative importance to the features. Adaboost despite having an overlap in top features with the other models assigns a different relative importance to individual features.

In [20]:

```
sns.set()
plt.figure(figsize=(11,8))
ax = sns.scatterplot(x = 'feature', y = 'importance', hue = 'model', data = features_50pc_c
#sns.barplot(x = 'feature', y = 'ab', data = features_30pc_ordered_melt.head(40))
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
ax.set_title('Feature Importance 30pc ordered by the importance computed by Random Forest')
```

Out[20]:

```
Text(0.5,1,'Feature Importance 30pc ordered by the importance computed by Ra
ndom Forest')
```