MRI and Alzheimer

- Predict Dementia Using Longitudinal MRI data in Nondemented and Demented Older Adults

Huiyu (Shirley) Sun

May 16, 2019

Background

- What is Dementia?
    - Dementia is a general term for a decline in mental ability severe enough to interfere with daily life (e.g. Memory loss). Dementia is not a specific disease. It’s an overall term that describe a group of symptoms associated with a decline in memory or other thinking skills severe enough to reduce a person’s ability to perform everyday activities.
    - Alzheimer’s disease accounts for 60% - 80% of cases.
- Diagnose Dementia
    - There is no one test to determine if someone has dementia.
    - Careful medical history, physical examination, laboratory tests, and characteristics changing thinking, day-to-tay function and behavior associated with each type.
    - It’s hard to determine the exact type of dementia because the symptoms and brain changes of different dementias can overlap.

Data

Data Features (150 subjects)

- Subject ID
- MRI ID
- Visit
- MR Delay

Demographic

- M/F
- Hand - Right hand
- Age - (60 - 98)
- EDUC (years)
- SES - Socioeconomic status 1: less than high school grad., 2: high school grad., 3: some college, 4: college grad., 5: beyond college.

Clinical

- MMSE - Mini-Mental State Examination (commonly used set of questions for screening cognitive function; 0-10 = Severe, 10-20 = Moderate; 20-25 = Mild; 25-30 = Questionably Significant)
- CDR - Clinical Dementia Rating (CDR; 0 = nondemented; 0.5 = very mild dementia; 1 = mild dementia; 2 = moderate dementia) (Morris, 1993). All participants with dementia (CDR >0) were diagnosed with probable AD.

Derived anatomic volumes

- eTIV - Estimated total intracranial volume
- nWBV - Normalized whole brain volume
- ASF - Atlas scaling factor

Response to be predicted

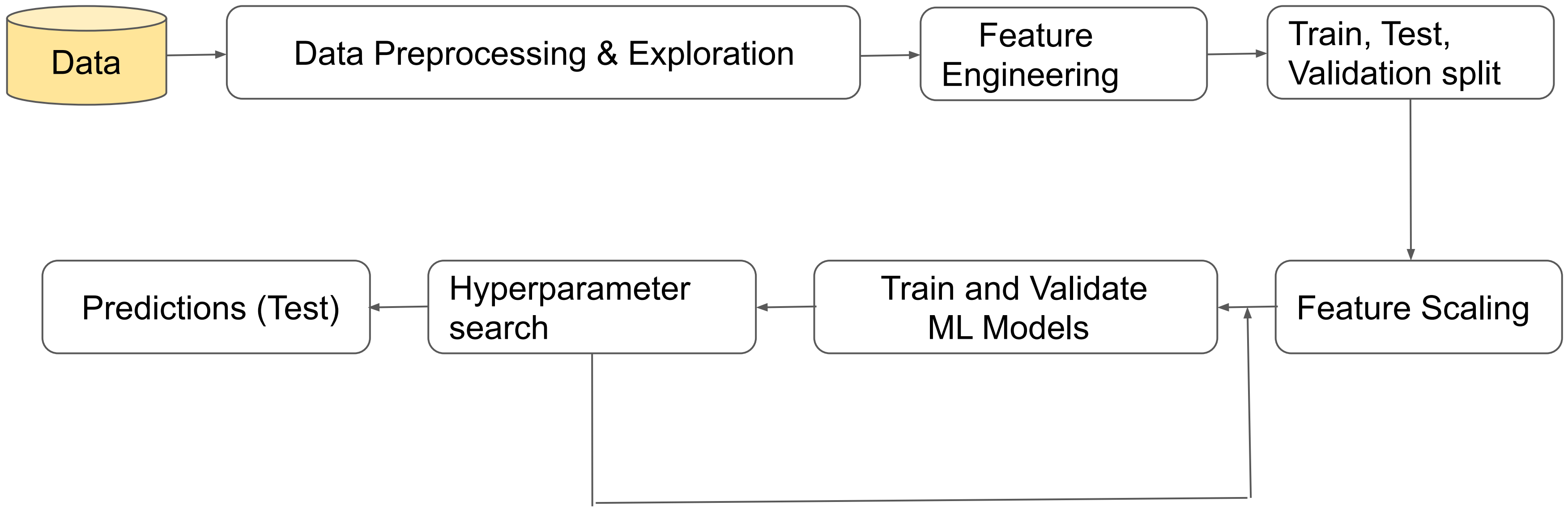
- Group - Demented, Nondemented, Converted

Data (cont)

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

mri_long = pd.read_csv('oasis_longitudinal.csv', sep = ',')
mri_long.head()
```

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0	27.0	0.0	1987	0.696	0.883
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0	30.0	0.0	2004	0.681	0.876
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	NaN	23.0	0.5	1678	0.736	1.046
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	NaN	28.0	0.5	1738	0.713	1.010
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	NaN	22.0	0.5	1698	0.701	1.034

Workflow


Data
Data Preprocessing & Exploration
Feature Engineering
Train, Test, Validation split
Feature Scaling
Train and Validate ML Models
Hyperparameter search
Predictions (Test)

Methods

Classification

- Logistic Regression
- Decision Tree Classifier
- Random Forest
- Gradient Boosting
- Naive Bayes
- SVM
- KNN

Data Preprocessing

```
# Drop the trivial/unrelated predictors
df = mri_long
df = df.drop(['Subject ID', 'MRI ID', 'Hand'], axis = 1)
```

```
df['Group'] = df['Group'].replace(['Converted'],['Demented'])
df['Group_code'] = LabelEncoder().fit_transform(df['Group'])
df['CDR_code']= LabelEncoder().fit_transform(df['CDR'])
```

1	Nondemented	2	457	M	88	14	2.0	30.0	0.0	2004	0.681	0.876
2	Demented	1	0	M	75	12	NaN	23.0	0.5	1678	0.736	1.046
3	Demented	2	560	M	76	12	NaN	28.0	0.5	1738	0.713	1.010
4	Demented	3	1895	M	80	12	NaN	22.0	0.5	1698	0.701	1.034

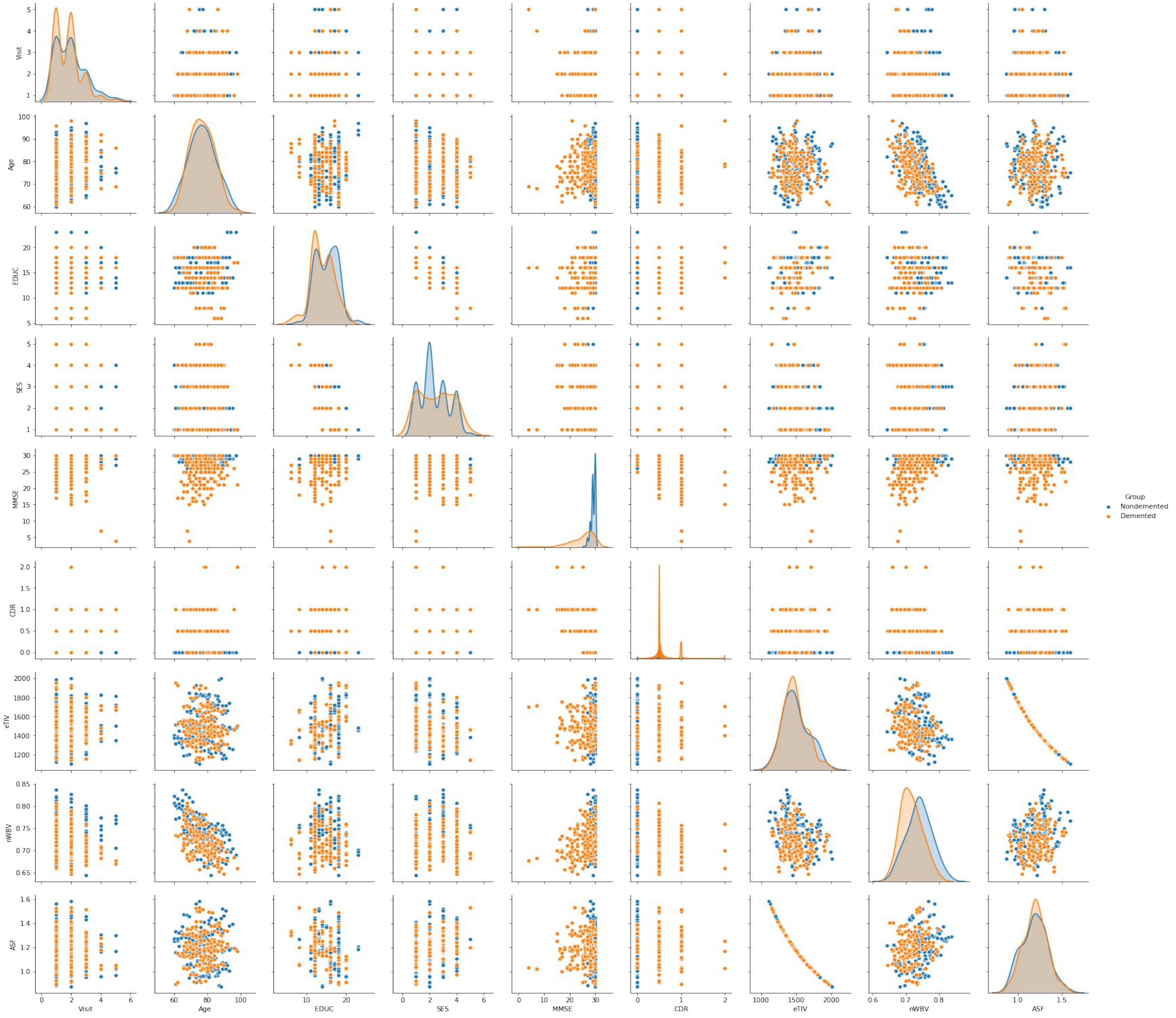
```
(373, 12)
```

Data Exploration

Age ~ -nWBV

eTIV ~ -nWBV

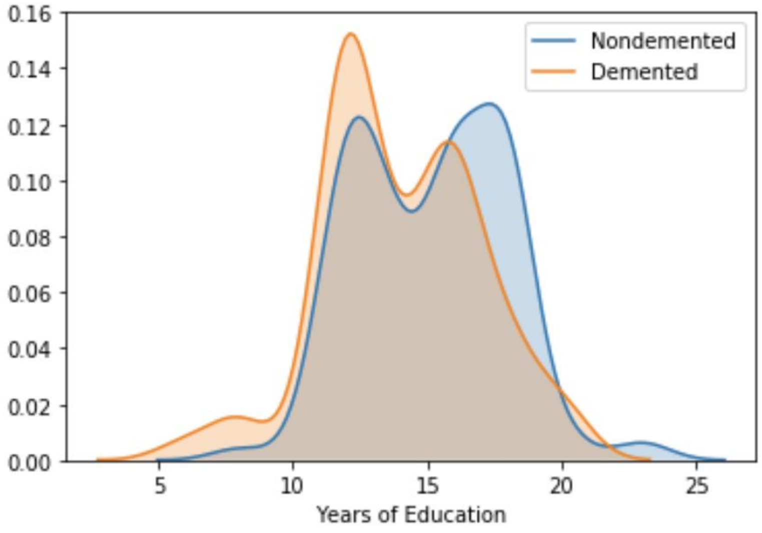
eTIV ~ -ASF (linear)



Data Exploration (cont)

```
# Years of education
# More Demented people with less years of education
sns.kdeplot(df.EDUC[df.Group=='Nondemented'], label='Nondemented', shade=True)
sns.kdeplot(df.EDUC[df.Group=='Demented'], label='Demented', shade=True)
plt.xlabel('Years of Education')
```

```
Text(0.5,0,'Years of Education')
```



Data Spliting and Scaling

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# X, y
x_cols = ['Gender_code', 'Visit', 'Age', 'EDUC', 'SES', 'MMSE', 'CDR_code', 'eTIV', 'nWBV', 'MR Delay']
X = df_rmna[x_cols]
y = df_rmna['Group_code'].astype('category')

# Training & Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)

num_cols = ['Visit', 'Age', 'EDUC', 'SES', 'MMSE', 'eTIV', 'nWBV', 'MR Delay']

# normalization
# training set
X_train_norm = StandardScaler().fit_transform(X_train[num_cols])
training_norm_col = pd.DataFrame(X_train_norm, index = X_train[num_cols].index, columns = X_train[num_cols].columns)
X_train.update(training_norm_col)

# test set
X_test_norm = StandardScaler().fit_transform(X_test[num_cols])
test_norm_col = pd.DataFrame(X_test_norm, index = X_test[num_cols].index, columns = X_test[num_cols].columns)
X_test.update(test_norm_col)
```

Modeling (cont)

```
# Random Forest (bagging)
from sklearn.ensemble import RandomForestClassifier

rf= RandomForestClassifier(random_state = 123)
rf.fit(X_train, y_train)
y_predict = rf.predict(X_test)
mean_squared_error(y_test, y_predict)

se_rf = np.round(mean_squared_error(y_test, y_predict),2)
acc_rf = np.round(accuracy_score(y_test, y_predict),2)
print('MSE:', se_rf)
print('Accuracy:', acc_rf)

fea_ip = rf.feature_importances_
sns.barplot(x = fea_ip, y = X.columns)
```

```
MSE: 0.24
Accuracy: 0.76
```

```
# KNN
from sklearn.neighbors import KNeighborsClassifier

y_predict = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train).predict(X_test)

se_knn = np.round(mean_squared_error(y_test, y_predict),2)
acc_knn = np.round(accuracy_score(y_test, y_predict),2)
print('MSE:', se_knn)
print('Accuracy:', acc_knn)
```

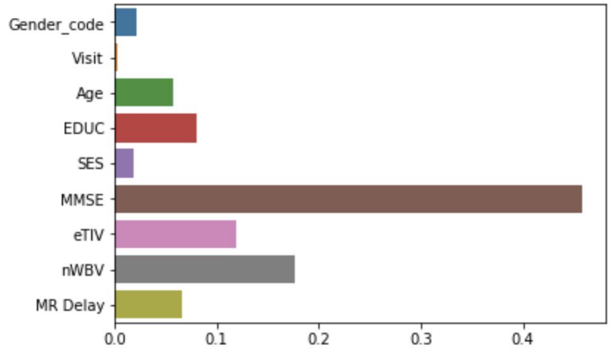
```
MSE: 0.48
Accuracy: 0.52
```

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB

y_predict = GaussianNB().fit(X_train, y_train).predict(X_test)

se_nb = np.round(mean_squared_error(y_test, y_predict),2)
acc_nb = np.round(accuracy_score(y_test, y_predict),2)
print('MSE:', se_nb)
print('Accuracy:', acc_nb)
```

```
MSE: 0.2
Accuracy: 0.8
```

```
# Gradient Boosting (boosting)
# sequential improvement of models by training on their errors
# improves errors, one tree each step
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(random_state = 0)
gb.fit(X_train, y_train)
y_predict = gb.predict(X_test)
mean_squared_error(y_test, y_predict)

se_gb = np.round(mean_squared_error(y_test, y_predict),2)
acc_gb = np.round(accuracy_score(y_test, y_predict),2)
print('MSE:', se_gb)
print('Accuracy:', acc_gb)

fea_ip = gb.feature_importances_
sns.barplot(x = fea_ip, y = X.columns)
```

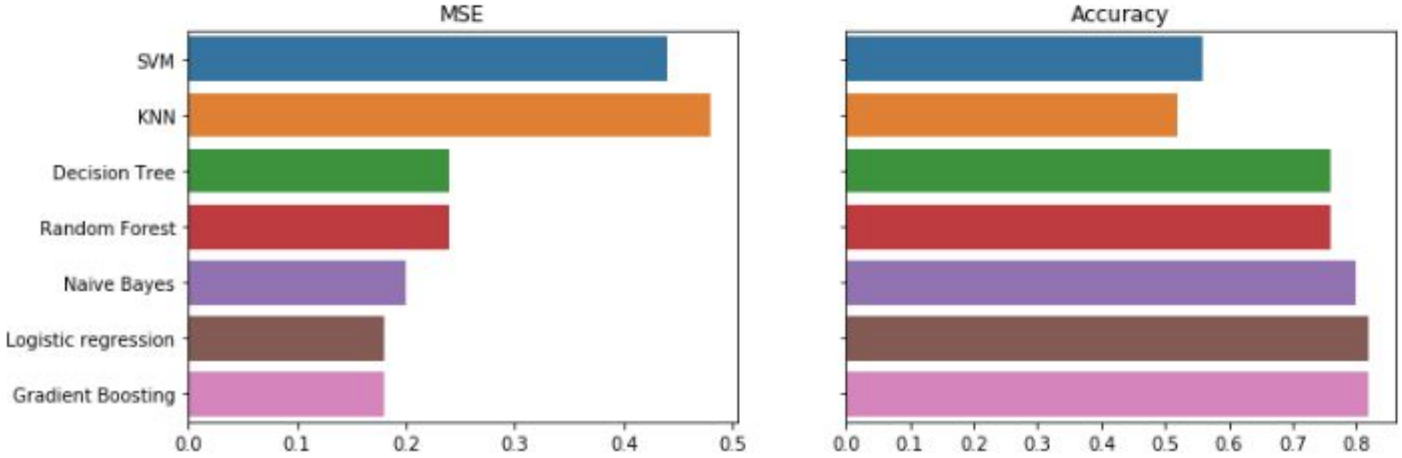
```
MSE: 0.18
Accuracy: 0.82

<matplotlib.axes._subplots.AxesSubplot at 0x1a2f3e64a8>
```



Model comparison (MSE, Test Accuracy)

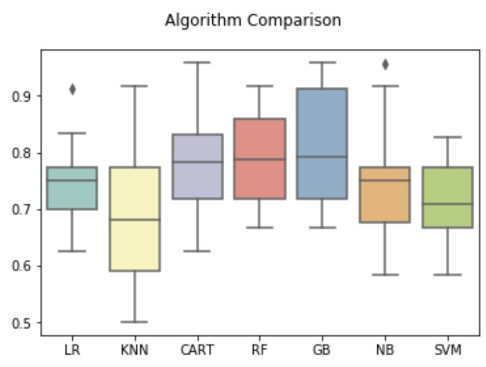
```
# visualize all methods
methods = ['SVM', 'KNN','Decision Tree', 'Random Forest', 'Naive Bayes', 'Logistic regression', 'Gradient Boosting']

se = [se_clf, se_knn, se_dtr, se_rf, se_nb, se_lr, se_gb]
acc = [acc_clf, acc_knn, acc_dtr, acc_rf, acc_nb, acc_lr, acc_gb]

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 4), sharey=True)
sns.barplot(x = se, y = methods, ax = ax1).set_title('MSE')
sns.barplot(x = acc, y = methods, ax = ax2).set_title('Accuracy')
```

```
Text(0.5,1,'Accuracy')
```



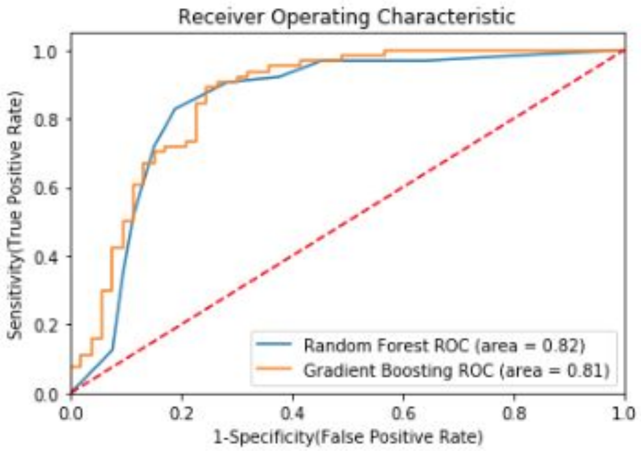
Model comparison (CV scores)

```python
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = KFold(n_splits=10, random_state=123)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
sns.boxplot(names, results, palette="Set3")
ax.set_xticklabels(names)
```



Performance (ROC curves)

```
# ROC curve RandomForest and GradientBoosting

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score

plt.figure()

# Add the models to the list that you want to view on the ROC plot
models = [
{
    'label': 'Random Forest',
    'model': RandomForestClassifier(),
},
{
    'label': 'Gradient Boosting',
    'model': GradientBoostingClassifier(),
}
]

# Below for loop iterates through your models list
for m in models:
    model = m['model'] # select the model
    model.fit(X_train, y_train) # train the model
    y_pred=model.predict(X_test) # predict the test data
# Compute False postive rate, and True positive rate
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
# Calculate Area under the curve to display on the plot
    auc = roc_auc_score(y_test, model.predict(X_test))
# Now, plot the computed values
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))

# Custom settings for the plot
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()   # Display
```



Hyperparameter Optimization

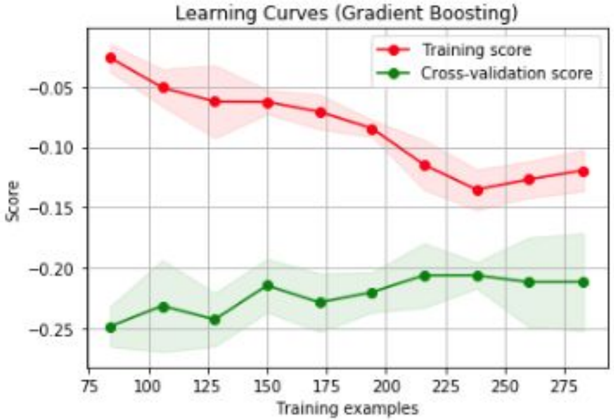
```
from sklearn.model_selection import GridSearchCV
```

```
gb = GradientBoostingClassifier(criterion='friedman_mse', init=None,
                learning_rate=0.05, loss='deviance', max_depth=3,
                max_features='sqrt', max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=20,
                min_weight_fraction_leaf=0.0, n_estimators=60,
                presort='auto', random_state=10, subsample=0.8, verbose=0,
                warm_start=False)
gb.fit(X_train, y_train)
y_predict = gb.predict(X_test)
mean_squared_error(y_test, y_predict)

se_gb = np.round(mean_squared_error(y_test, y_predict),2)
acc_gb = np.round(accuracy_score(y_test, y_predict),2)
print('MSE:', se_gb)
print('Accuracy:', acc_gb)
```

```
MSE: 0.16
Accuracy: 0.84
```

Not too bad

```
# test overfit
title = "Learning Curves (Gradient Boosting)"

# Create the CV iterator
cv_iterator = KFold(n_splits=5, shuffle=True, random_state=10)
model = GradientBoostingClassifier(criterion='friedman_mse', init=None,
               learning_rate=0.05, loss='deviance', max_depth=3,
               max_features='sqrt', max_leaf_nodes=None,
               min_impurity_decrease=0.0, min_impurity_split=None,
               min_samples_leaf=1, min_samples_split=20,
               min_weight_fraction_leaf=0.0, n_estimators=60,
               presort='auto', random_state=10, subsample=0.8, verbose=0,
               warm_start=False)
plot_learning_curve(model, title, X_norm, y, cv=cv_iterator, n_jobs=4)
plt.show()
```



Need more data...

THANK YOU!