分析部分代码—— 包含超参数调节，与决策树和 Sklearn_AdaBoost 在真实数据上进行比较，并对分析结果可视化。

```python
#!/usr/bin/env python
# coding: utf-8

# In[151]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.ensemble

from AdaBoost import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RepeatedKFold
from sklearn.datasets.samples_generator import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from scipy.stats import mode
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_score, confusion_matrix


# # 1. Grid Search and Tuning Hyperparameters on Synthetic Dataset

# In[152]:


# Load the synthetic_data.csv we have created in the 'Creating synthetic data.pyid'
data = pd.read_csv('C:/Users/nasie/OneDrive/Desktop/QBUS6850/GA/synthetic_data.csv',index_col=0)
data.head()


# In[153]:


y = data['y'].values
X = data[data.columns.difference(['y'])].values


# In[154]:


# Randomly split the dataset to training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)


# ## 1.1. Analyse the impact of the number of base estimators
```

In[76]:

```
np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
n_estimators = np.arange(1, 20, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for e in n_estimators:
        cla = AdaBoostClassifier(n_estimators=e)
        cla.fit(X_train, y_train)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(n_estimators, mean_accuracy_train)
plt.plot(n_estimators, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('n_estimators')
plt.show()
```

1.2. Analyse the impact of number of sample trees

In[77]:

```
np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
n_trees = np.arange(1, 20, 1)
```

```
for i in range(repeats):
  accs_trains = []
  accs_validations = []
  for t in n_trees:
    cla = AdaBoostClassifier()
    cla.fit(X_train, y_train, n_trees=t)

    acc_train = cla.score(X_train,y_train)
    acc_validation = cla.score(X_test,y_test)
    accs_trains.append(acc_train)
    accs_validations.append(acc_validation)
  all_results_train.append(accs_trains)
  all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(n_trees, mean_accuracy_train)
plt.plot(n_trees, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('n_trees')
plt.show()


# ## 1.3. Analyse the impact of number of boostrapped sample size

# In[78]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []

# Proportion of dataset size
ratios = np.arange(0.1, 1.1, 0.1)

for i in range(repeats):
  accs_trains = []
  accs_validations = []
  for r in ratios:
    # Convert proportion to sample size
    s = int(r * X_train.shape[0])
    cla = AdaBoostClassifier()
    cla.fit(X_train, y_train, sample_size=s)
```

```
        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(ratios, mean_accuracy_train)
plt.plot(ratios, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('ratios')
plt.show()


# ## 1.4. Analyse the impact of depth of trees

# In[156]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
max_depth = np.arange(1, 20, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for d in max_depth:
        cla = AdaBoostClassifier()
        cla.fit(X_train, y_train, max_depth=d)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
```

```
plt.plot(max_depth, mean_accuracy_train)
plt.plot(max_depth, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('max_depth')
plt.show()


# ## 1.5. Analyse the impact of number of features

# In[71]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
max_features = np.arange(1, X_train.shape[1]+1, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for f in max_features:
        cla = AdaBoostClassifier()
        cla.fit(X_train, y_train, max_features=f)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(max_features, mean_accuracy_train)
plt.plot(max_features, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('max_features')
plt.show()


# ## 1.6. Analyse the impact of changing min_samples_split

# In[10]:
```

```
np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations

min_samples_split = np.arange(2, 20, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for p in min_samples_split:
        cla = AdaBoostClassifier()
        cla.fit(X_train, y_train, min_samples_split=p)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(min_samples_split, mean_accuracy_train)
plt.plot(min_samples_split, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('min_samples_split')
plt.show()


# ## 1.7. Analyse the impact of changing min_samples_leaf

# In[11]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
min_samples_leaf = np.arange(1, 20, 1)

for i in range(repeats):
```

```
    accs_trains = []
    accs_validations = []
    for l in min_samples_leaf:
        cla = AdaBoostClassifier()
        cla.fit(X_train, y_train, min_samples_leaf=l)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(min_samples_leaf, mean_accuracy_train)
plt.plot(min_samples_leaf, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('min_samples_leaf')
plt.show()


# ## 1.8. Analyse the impact of changing min_weight_fraction_leaf

# In[12]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
min_weight_fraction_leaf = np.arange(0, 0.51, 0.01)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for f in min_weight_fraction_leaf:
        cla = AdaBoostClassifier()
        cla.fit(X_train, y_train, min_weight_fraction_leaf=f)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
```

```
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(min_weight_fraction_leaf, mean_accuracy_train)
plt.plot(min_weight_fraction_leaf, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('min_weight_fraction_leaf')
plt.show()


# ## 1.9. Analyse the impact of changing max_leaf_nodes

# In[13]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
max_leaf_nodes = np.arange(2, 20, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for ln in max_leaf_nodes:
        cla = AdaBoostClassifier()
        cla.fit(X_train, y_train)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(max_leaf_nodes, mean_accuracy_train)
plt.plot(max_leaf_nodes, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('max_leaf_nodes')
plt.show()
```

2. Comparison to Sklearn AdaBoost on Real Dataset and Decision Tree Classifier

In[106]:

```
# Load the synthetic_data.csv we have created in the 'Creating synthetic data.pyid'
data =
pd.read_csv('C:/Users/nasie/OneDrive/Desktop/QBUS6850/GA/Submission/datasets_376751_731448_london_me
rged.csv')
data.head()
```

In[107]:

```
data.info()
```

In[108]:

```
y = data['is_weekend'].values
X_raw = data[data.columns.difference(['is_weekend'])]
X_not_dum = X_raw[['cnt', 't1', 't2', 'hum', 'wind_speed']]
X_dum = X_raw[['am', 'day', 'is_holiday', 'month', 'season', 'time', 'weather_code', 'year']]
X_dum = pd.get_dummies(X_dum.astype(str), drop_first=True)
X = pd.concat([X_not_dum, X_dum], axis=1).values
```

In[109]:

```
index = (y==0)
y[index] = -1
```

In[110]:

```
# Randomly split the dataset to training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

2.1. Tuning the parameters on real dataset

2.1.1. Tuning the hyper-parameters for AdaBoost

```
# #### (1). Tuning n_estimators

# In[30]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
n_estimators = np.arange(1, 30, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for e in n_estimators:
        cla = AdaBoostClassifier(n_estimators=e)
        cla.fit(X_train, y_train, max_depth=30)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(n_estimators, mean_accuracy_train)
plt.plot(n_estimators, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('n_estimators')
plt.show()


# #### (2). Tuning max_depth

# In[33]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
```

```
max_depth = np.arange(1, 30, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for d in max_depth:
        cla = AdaBoostClassifier(n_estimators=6)
        cla.fit(X_train, y_train, max_depth=d)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(max_depth, mean_accuracy_train)
plt.plot(max_depth, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('max_depth')
plt.show()


# ### 2.1.2. Tuning the hyper-parameters for sklearn AdaBoost

# #### (1). Tuning n_estimators

# In[37]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
n_estimators = np.arange(1, 30, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for e in n_estimators:
        cla = sklearn.ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=30), n_estimators=e,)
        cla.fit(X_train, y_train)
```

```
        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(n_estimators, mean_accuracy_train)
plt.plot(n_estimators, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('n_estimators')
plt.show()


# #### (2). Tuning max_depth

# In[39]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
max_depth = np.arange(1, 30, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for d in max_depth:
        cla = sklearn.ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=d))
        cla.fit(X_train, y_train)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
```

```
plt.plot(max_depth, mean_accuracy_train)
plt.plot(max_depth, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('max_depth')
plt.show()


# ### 2.1.3 Tuning the hyper-parameters for Decision Tree

# #### (1). Tuning max_depth

# In[40]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
max_depth = np.arange(1, 30, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for d in max_depth:
        cla = DecisionTreeClassifier(max_depth=d)
        cla.fit(X_train, y_train)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(max_depth, mean_accuracy_train)
plt.plot(max_depth, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('max_depth')
plt.show()


# #### (2). Tuning min_samples_leaf
```

```
# In[41]:


np.random.seed(0)
# Number of times to repeat the experiment
repeats = 10
all_results_train = []
all_results_validation = []
# The range of iterations
min_samples_leaf = np.arange(1, 30, 1)

for i in range(repeats):
    accs_trains = []
    accs_validations = []
    for l in min_samples_leaf:
        cla= DecisionTreeClassifier(min_samples_leaf = l)
        cla.fit(X_train, y_train)

        acc_train = cla.score(X_train,y_train)
        acc_validation = cla.score(X_test,y_test)
        accs_trains.append(acc_train)
        accs_validations.append(acc_validation)
    all_results_train.append(accs_trains)
    all_results_validation.append(accs_validations)
mean_accuracy_train = np.mean(all_results_train, axis=0)
mean_accuracy_validation = np.mean(all_results_validation, axis=0)

#Plot
plt.figure()
plt.plot(min_samples_leaf, mean_accuracy_train)
plt.plot(min_samples_leaf, mean_accuracy_validation)
plt.legend(['train accuracy','validation accuracy'])
plt.xlabel('min_samples_leaf')
plt.show()


# ## 2.2. Make comparison to sklearn AdaBoost and Decison Tree

# ## 2.2.1. AdaBoostClassifier()

# In[138]:


import datetime
start = time.time()

# AdaBoost
cla_ada = AdaBoostClassifier(n_estimators=6)
cla_ada.fit(X_train, y_train, max_depth=23)
```

```
y_pred = cla_ada.predict(X_test)

y_test = y_test.reshape(-1, 1)

# We use confusion matrix, confusion, error_rate, sensitivity, specificity, precision to validate our models
confusion  = confusion_matrix(y_test, y_pred)
error_rate = (1 - cla_ada.score(X_test, y_pred)).round(3)
sensitivity = (confusion[1,1]/np.sum(confusion[1,:])).round(3)
specificity = (confusion[0,0]/np.sum(confusion[0,:])).round(3)
precision = precision_score(y_test, y_pred).round(3)

end = time.time()

print('AdaBoost')
print('')
print({'confusion': confusion})
print({'validation_error': error_rate})
print({'sensitivity': sensitivity})
print({'specificity': specificity})
print({'precision': precision})
Ada_time = end- start


# ## 2.2.2. sklearn.AdaBoostClassifier()

# In[139]:


start = time.time()
# Sklearn.AdaBoost
cla_skl_ada = sklearn.ensemble.AdaBoostClassifier( DecisionTreeClassifier(max_depth=30), n_estimators=30)
cla_skl_ada.fit(X_train, y_train)

y_pred = cla_skl_ada.predict(X_test)
y_test = y_test.reshape(-1, 1)
# We use confusion matrix, confusion, error_rate, sensitivity, specificity, precision to validate our models
confusion  = confusion_matrix(y_test, y_pred)
error_rate = (1 - accuracy_score(y_test, y_pred)).round(3)
sensitivity = (confusion[1,1]/np.sum(confusion[1,:])).round(3)
specificity = (confusion[0,0]/np.sum(confusion[0,:])).round(3)
precision = precision_score(y_test, y_pred).round(3)

end = time.time()

print('sklearn AdaBoost')
print('')
print({'confusion': confusion})
print({'validation_error': error_rate})
print({'sensitivity': sensitivity})
```

```
print({'specificity': specificity})
print({'precision': precision})

skl_ada_time = end-start


# ## 2.2.3. Decision Tree()

# In[146]:


start = time.time()
# Decision Tree

dt = DecisionTreeClassifier(max_depth = 15, min_samples_leaf = 10)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)
y_test = y_test.reshape(-1, 1)

# We use confusion matrix, auc, confusion, error_rate, sensitivity, specificity, precision to validate our models
confusion  = confusion_matrix(y_test, y_pred)
error_rate = (1 - accuracy_score(y_test, y_pred)).round(3)
sensitivity = (confusion[1,1]/np.sum(confusion[1,:])).round(3)
specificity = (confusion[0,0]/np.sum(confusion[0,:])).round(3)
precision = precision_score(y_test, y_pred).round(3)

end = time.time()

print('sklearn sklearn decision tree')
print('')
print({'confusion': confusion})
print({'validation_error': error_rate})
print({'sensitivity': sensitivity})
print({'specificity': specificity})
print({'precision': precision})

dt_time = end - start


# In[147]:


print({'AdaBoost: ': Ada_time})
print({'sklearn AdaBoost: ': skl_ada_time})
print({'sklern DecisionTree: ': dt_time})


# In[148]:
```

```
print({'AdaBoost: ': Ada_time})
print({'sklern DecisionTree: ': dt_time})


# In[ ]:
```