Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №4 з дисципліни
«Програмування інтелектуальних інформаційних систем»

„Тюнінг, ансамблювання та бустинг”

Виконав(ла) *ІП-11 Прищепа В.С.* __________
(шифр, прізвище, ім'я, по батькові)

Перевірив *Баришич Л. М.* __________
(прізвище, ім'я, по батькові)

Київ 2023

Завдання
1 Потюнити параметри за цим туторіалом. Порівняти з дефолтними. Зрозуміти роботу алгоритмічного підбору параметрів.

https://www.kaggle.com/code/shreayan98c/hyperparameter-tuning-tutorial

2 Зробити ансамблі і бустинг за цим туторіалом. Пояснити відмінності.

Порівняти з просто тюнингом. Пояснити коли і що використовувати.

https://www.kaggle.com/code/pavansanagapati/ensemble-learning-techniques-tutorial

Хід роботи:

1.Тюнінг

Код програми:

```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

# Importing and prepearing
data = pd.read_csv("data.csv")
data.drop(['Unnamed: 32', 'id'], axis=1, inplace=True)
print("Size of data:", data.shape)
# Spliting target variable and independent variables
X = data.drop(['diagnosis'], axis = 1)
y = data['diagnosis']
# Splitting the data into training set and testset
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state =
0)
print("Size of training set:", X_train.shape)
print("Size of test set:", X_test.shape)

from sklearn.tree import DecisionTreeClassifier
# Create a Decision tree classifier model
clf = DecisionTreeClassifier()
default_params = clf.get_params()
# Train the model using the training sets
clf.fit(X_train, y_train)
# Prediction on test set
y_pred = clf.predict(X_test)
```

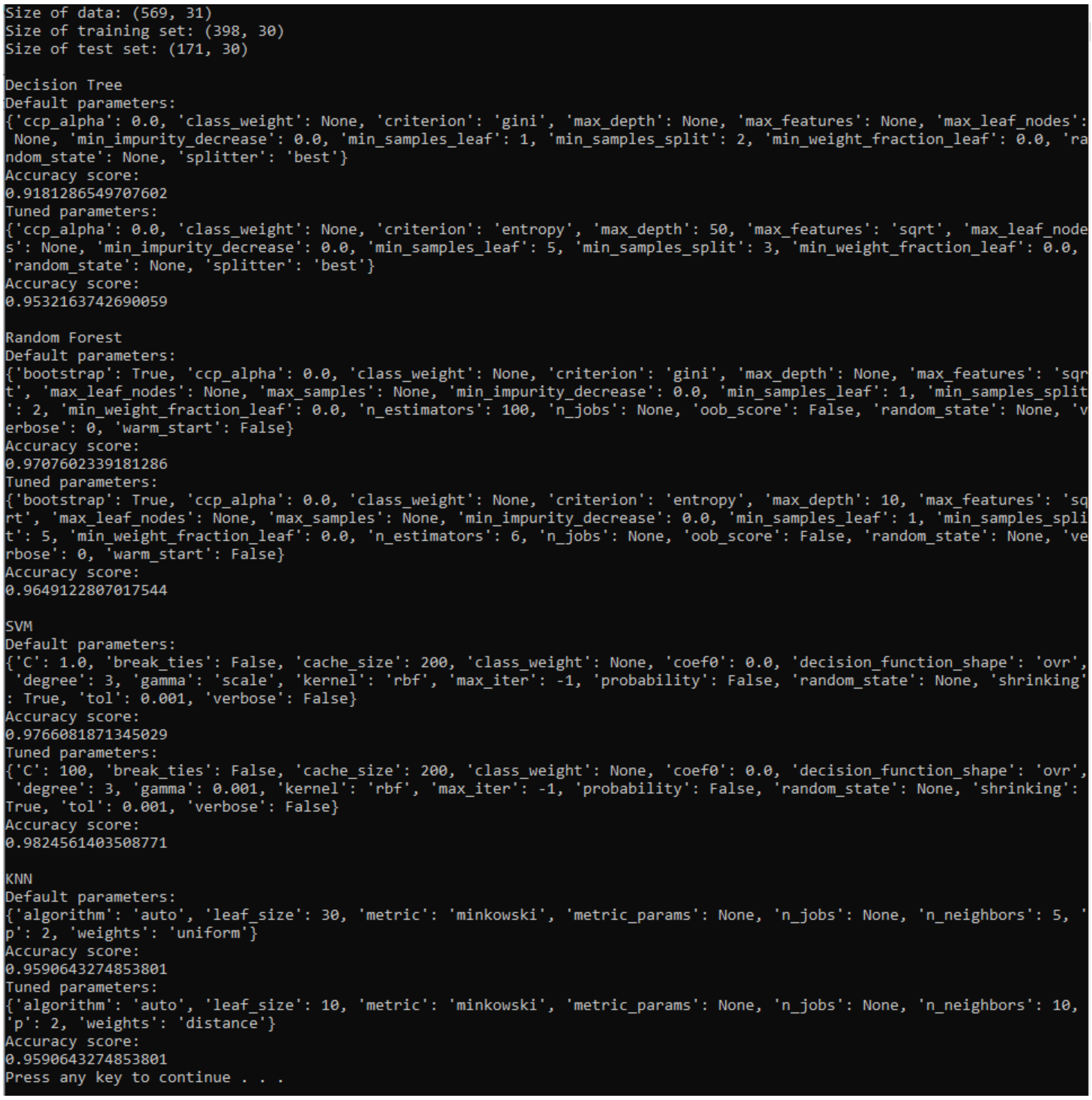
```
# Calculating the accuracy
acc_dt = metrics.accuracy_score(y_test, y_pred)
import warnings
# Turn off all warnings
warnings.filterwarnings("ignore")
# Create a Decision tree classifier model
clf = DecisionTreeClassifier()
# Hyperparameter Optimization
parameters = {'max_features': ['log2', 'sqrt','auto'],
'criterion': ['entropy', 'gini'],
'max_depth': [2, 3, 5, 10, 50],
'min_samples_split': [2, 3, 50, 100],
'min_samples_leaf': [1, 5, 8, 10]
}
# Run the grid search
grid_obj = GridSearchCV(clf, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_
tuned_params = clf.get_params()
# Train the model using the training sets
clf.fit(X_train, y_train)
# Prediction on test set
y_pred = clf.predict(X_test)
# Calculating the accuracy
acc_dt_tuned = metrics.accuracy_score(y_test, y_pred)
print('\nDecision Tree')
print('Default parameters:', default_params, 'Accuracy score:', acc_dt, sep='\n')
print('Tuned parameters:', tuned_params, 'Accuracy score:', acc_dt_tuned, sep='\n')

from sklearn.ensemble import RandomForestClassifier
# Create a Random Forest Classifier
rf = RandomForestClassifier()
default_params = rf.get_params()
# Train the model using the training sets
rf.fit(X_train,y_train)
# Prediction on test set
y_pred = rf.predict(X_test)
# Calculating the accuracy
```

```
acc_rf = metrics.accuracy_score(y_test, y_pred)
# Create a Random Forest Classifier
rf = RandomForestClassifier()
# Hyperparameter Optimization
parameters = {'n_estimators': [4, 6, 9, 10, 15],
'max_features': ['log2', 'sqrt','auto'],
'criterion': ['entropy', 'gini'],
'max_depth': [2, 3, 5, 10],
'min_samples_split': [2, 3, 5],
'min_samples_leaf': [1, 5, 8]
}
# Run the grid search
grid_obj = GridSearchCV(rf, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
# Set the rf to the best combination of parameters
rf = grid_obj.best_estimator_
tuned_params = rf.get_params()
# Train the model using the training sets
rf.fit(X_train,y_train)
# Prediction on test set
y_pred = rf.predict(X_test)
# Calculating the accuracy
acc_rf_tuned = metrics.accuracy_score(y_test, y_pred)
print('\nRandom Forest')
print('Default parameters:', default_params, 'Accuracy score:', acc_rf, sep='\n')
print('Tuned parameters:', tuned_params, 'Accuracy score:', acc_rf_tuned, sep='\n')

# Creating scaled set to be used in model to improve the results
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Import Library of Support Vector Machine model
from sklearn import svm
# Create a Support Vector Classifier
svc = svm.SVC()
default_params = svc.get_params()
# Train the model using the training sets
svc.fit(X_train,y_train)
```

```
# Prediction on test data
y_pred = svc.predict(X_test)
# Calculating the accuracy
acc_svm = metrics.accuracy_score(y_test, y_pred)
# Create a Support Vector Classifier
svc = svm.SVC()
# Hyperparameter Optimization
parameters = [
{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]
# Run the grid search
grid_obj = GridSearchCV(svc, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
# Set the svc to the best combination of parameters
svc = grid_obj.best_estimator_
tuned_params = svc.get_params()
# Train the model using the training sets
svc.fit(X_train,y_train)
# Prediction on test data
y_pred = svc.predict(X_test)
# Calculating the accuracy
acc_svm_tuned = metrics.accuracy_score(y_test, y_pred)
print('\nSVM')
print('Default parameters:', default_params, 'Accuracy score:', acc_svm, sep='\n')
print('Tuned parameters:', tuned_params, 'Accuracy score:', acc_svm_tuned, sep='\n')

# Import library of KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier
# Create a KNN Classifier
knn = KNeighborsClassifier()
default_params = knn.get_params()
# Train the model using the training sets
knn.fit(X_train,y_train)
# Prediction on test data
y_pred = knn.predict(X_test)
# Calculating the accuracy
acc_knn = metrics.accuracy_score(y_test, y_pred)
# Create a KNN Classifier
```

```
knn = KNeighborsClassifier()
# Hyperparameter Optimization
parameters = {'n_neighbors': [3, 4, 5, 10],
'weights': ['uniform', 'distance'],
'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
'leaf_size' : [10, 20, 30, 50]
}
# Run the grid search
grid_obj = GridSearchCV(knn, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
# Set the knn to the best combination of parameters
knn = grid_obj.best_estimator_
tuned_params = knn.get_params()
# Train the model using the training sets
knn.fit(X_train,y_train)
# Prediction on test data
y_pred = knn.predict(X_test)
# Calculating the accuracy
acc_knn_tuned = metrics.accuracy_score(y_test, y_pred)
print('\nKNN')
print('Default parameters:', default_params, 'Accuracy score:', acc_knn, sep='\n')
print('Tuned parameters:', tuned_params, 'Accuracy score:', acc_knn_tuned, sep='\n')
```

Результат виконання:



```
Size of data: (569, 31)
Size of training set: (398, 30)
Size of test set: (171, 30)

Decision Tree
Default parameters:
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes':
 None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'ra
ndom_state': None, 'splitter': 'best'}
Accuracy score:
0.9181286549707602
Tuned parameters:
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth': 50, 'max_features': 'sqrt', 'max_leaf_node
s': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 3, 'min_weight_fraction_leaf': 0.0,
'random_state': None, 'splitter': 'best'}
Accuracy score:
0.9532163742690059

Random Forest
Default parameters:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqr
t', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split
': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'v
erbose': 0, 'warm_start': False}
Accuracy score:
0.9707602339181286
Tuned parameters:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sq
rt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_spli
t': 5, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 6, 'n_jobs': None, 'oob_score': False, 'random_state': None, 've
rbose': 0, 'warm_start': False}
Accuracy score:
0.9649122807017544

SVM
Default parameters:
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr',
 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking'
: True, 'tol': 0.001, 'verbose': False}
Accuracy score:
0.9766081871345029
Tuned parameters:
{'C': 100, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr',
 'degree': 3, 'gamma': 0.001, 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking':
True, 'tol': 0.001, 'verbose': False}
Accuracy score:
0.9824561403508771

KNN
Default parameters:
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, '
p': 2, 'weights': 'uniform'}
Accuracy score:
0.9590643274853801
Tuned parameters:
{'algorithm': 'auto', 'leaf_size': 10, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 10,
'p': 2, 'weights': 'distance'}
Accuracy score:
0.9590643274853801
Press any key to continue . . .
```

Бачимо, що тюнінг покращив результати моделей SVM та Decision Tree, не змінив KNN та погіршив Random Forest. Хоча на цьому наборі результати без тюнингу і так задовільні, все одно тюнінг у більшості випадків видає кращий результат. Причиною погіршення Random Forest може бути те, що для тюнінгу було запропоновано не достатню кількість значень параметрів, з яких можна вибирати.

2.Ансамблі і бустинг

Код програми:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Importing the dataset
data = pd.read_csv("data.csv")
# Dropping the Unnamed: 32 and the id column since these do not provide any useful
information for our models.
data.drop(['Unnamed: 32', 'id'], axis=1, inplace=True)
# Spliting target variable and independent variables
X = data.drop(['diagnosis'], axis = 1)
y = data['diagnosis']
# Split on training and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.20,random_state =
42)
# Ensemble of Models
estimator = []
estimator.append(('LR', LogisticRegression(solver ='lbfgs',multi_class
='multinomial',max_iter = 5000)))
estimator.append(('SVC', SVC(gamma ='auto', probability = True)))
estimator.append(('DTC', DecisionTreeClassifier()))
print(estimator)

from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score
# Voting Classifier with hard voting
hard_voting = VotingClassifier(estimators = estimator, voting ='hard')
hard_voting.fit(X_train, y_train)
y_pred = hard_voting.predict(X_test)
# accuracy_score metric to predict Accuracy
score = accuracy_score(y_test, y_pred)
print("Hard Voting Score", score)

# Voting Classifier with soft voting
soft_voting = VotingClassifier(estimators = estimator, voting ='soft')
soft_voting.fit(X_train, y_train)
y_pred = soft_voting.predict(X_test)
# Using accuracy_score
score = accuracy_score(y_test, y_pred)
```

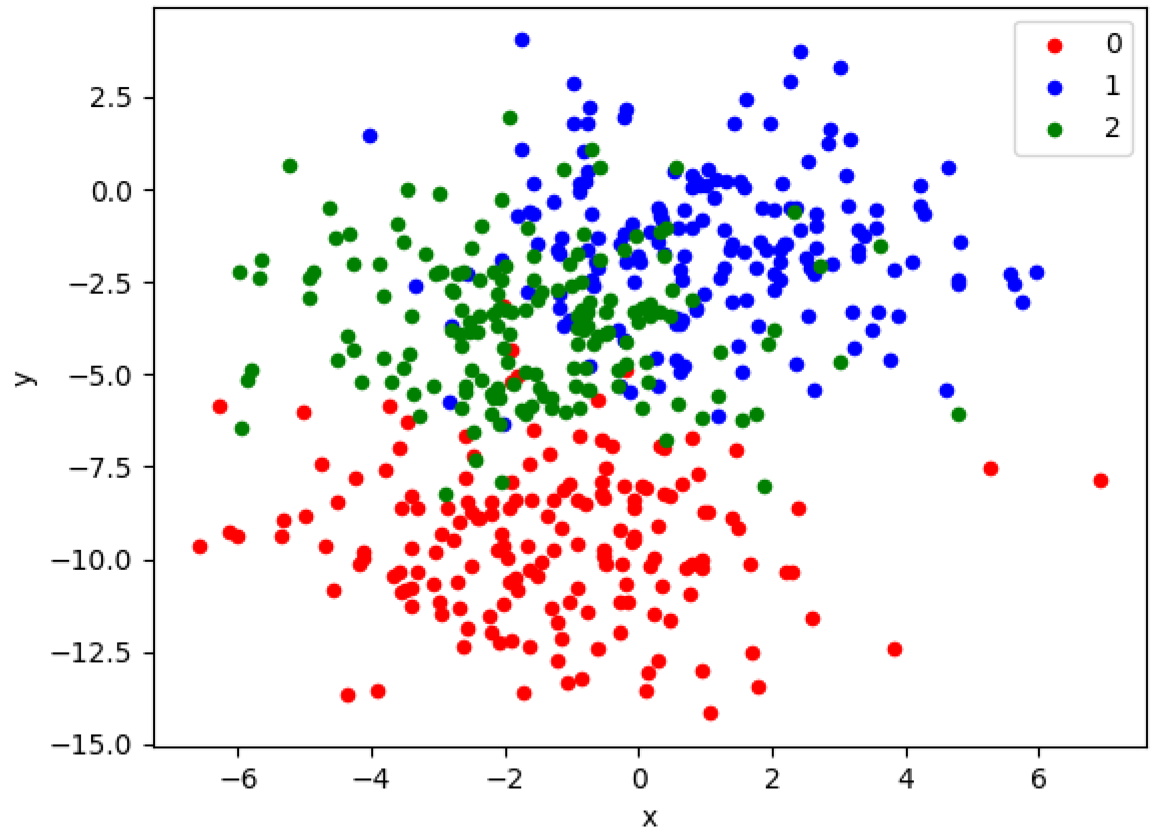
```
print("Soft Voting Score", score)

from sklearn.datasets import make_blobs
from matplotlib import pyplot
from pandas import DataFrame
# generate 2d classification dataset
X, y = make_blobs(n_samples=500, centers=3, n_features=2, cluster_std=2,
random_state=2)
# scatter plot, dots colored by class value
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue', 2:'green'}
fig, ax = pyplot.subplots()
grouped = df.groupby('label')
for key, group in grouped:
   group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
pyplot.show()

import numpy as np
# Importing the dataset
data = pd.read_csv("data.csv")
# Dropping the Unnamed: 32 and the id column since these do not provide any useful
information for our models.
data.drop(['Unnamed: 32', 'id'], axis=1, inplace=True)
# Spliting target variable and independent variables
X = data.drop(['diagnosis'], axis = 1)
y = data['diagnosis']
# Split on training and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.20,random_state =
42)
lr = LogisticRegression(solver ='lbfgs',multi_class ='multinomial',max_iter = 5000)
sv = SVC(gamma ='auto', probability = True)
dt = DecisionTreeClassifier()
# encoding
y_train = [0 if elem=='B' else 1 for elem in y_train]
lr.fit(X_train, y_train)
sv.fit(X_train, y_train)
dt.fit(X_train, y_train)
predicted = [lr.predict(X_test), sv.predict(X_test), dt.predict(X_test)]
best_weights = []
```

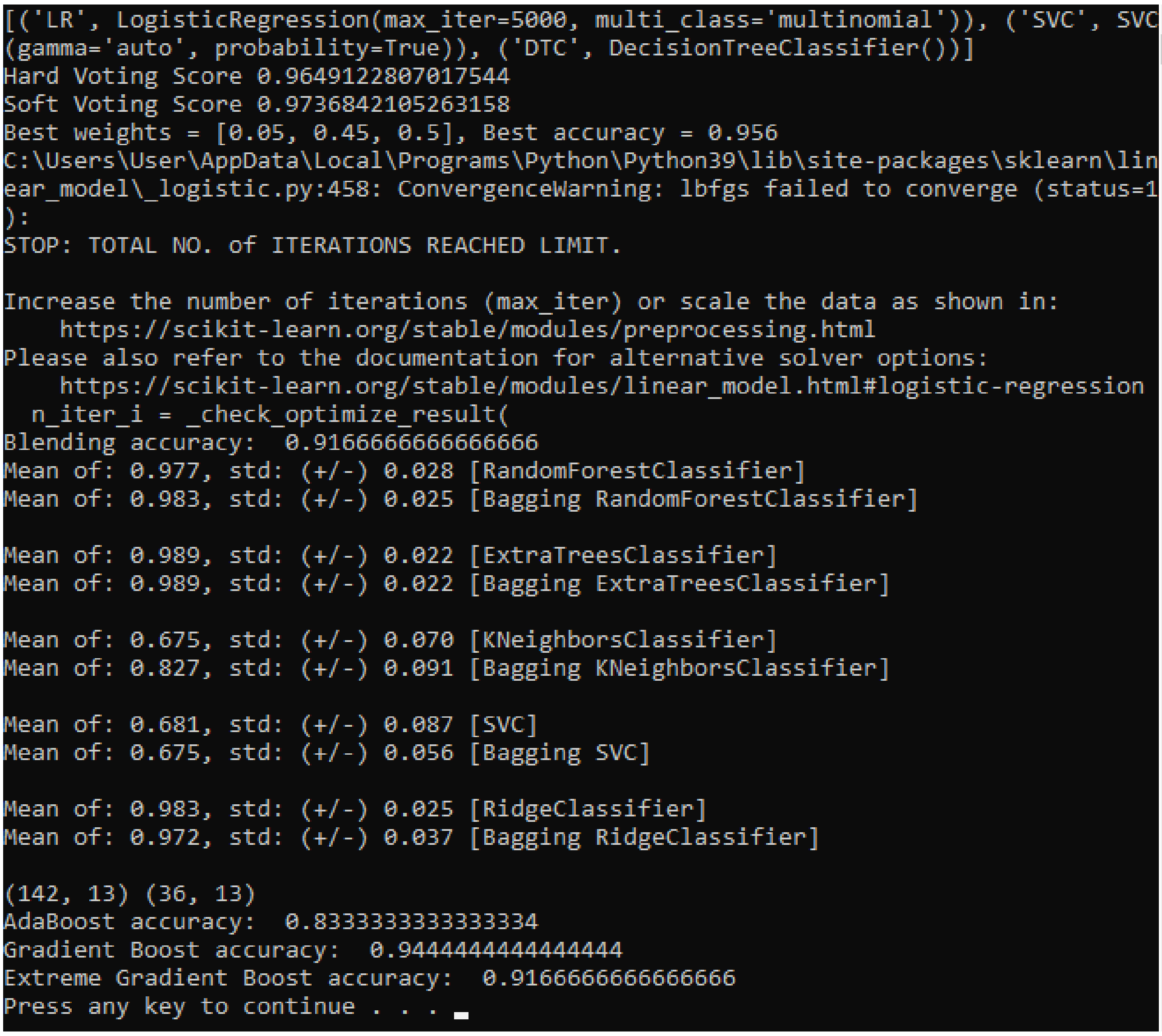
```
best_accuracy = 0
# Optimal wages
for w1 in np.arange(0.05, 0.95, 0.05):
    for w2 in np.arange(0.1, 0.95, 0.05):
        if w1+w2>0.95:
            continue
        else:
            w3 = 1-(w1+w2)
            y_pred = [predicted[0][i]*w1 + predicted[1][i]*w2 + predicted[2][i]*w3 for i
in range(len(y_test))]
            # decoding
            y_pred = ['B' if round(elem)==0 else 'M' for elem in y_pred]
            score = accuracy_score(y_test, y_pred)
            if score>best_accuracy:
                best_accuracy = score
                best_weights = [w1, w2, w3]
print(f'Best weights = {[round(x, 2) for x in best_weights]}, Best accuracy =
{round(best_accuracy, 3)}')

from sklearn.datasets import load_wine
from sklearn.neighbors import KNeighborsClassifier
# define dataset
X,y = load_wine().data,load_wine().target
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(
X_train, y_train, test_size=0.25, random_state=1)
x_val=pd.DataFrame(X_val)
x_test=pd.DataFrame(X_test)
model1 = DecisionTreeClassifier()
model1.fit(X_train, y_train)
val_pred1=model1.predict(X_val)
test_pred1=model1.predict(X_test)
val_pred1=pd.DataFrame(val_pred1)
test_pred1=pd.DataFrame(test_pred1)
model2 = KNeighborsClassifier()
model2.fit(X_train,y_train)
val_pred2=model2.predict(X_val)
test_pred2=model2.predict(X_test)
```

```
val_pred2=pd.DataFrame(val_pred2)
test_pred2=pd.DataFrame(test_pred2)
df_val=pd.concat([x_val, val_pred1,val_pred2],axis=1)
df_test=pd.concat([x_test, test_pred1,test_pred2],axis=1)
model = LogisticRegression()
model.fit(df_val,y_val)
print('Blending accuracy: ', model.score(df_test,y_test))

from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier,
BaggingClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import cross_val_score
# define dataset
X,y = load_wine().data,load_wine().target
# Create classifiers
rf = RandomForestClassifier()
et = ExtraTreesClassifier()
knn = KNeighborsClassifier()
svc = SVC()
rg = RidgeClassifier()
clf_array = [rf, et, knn, svc, rg]
for clf in clf_array:
   vanilla_scores = cross_val_score(clf, X, y, cv=10, n_jobs=-1)
   bagging_clf = BaggingClassifier(clf,max_samples=0.4, max_features=10,
random_state=42)
   bagging_scores = cross_val_score(bagging_clf, X, y, cv=10,n_jobs=-1)
   print ("Mean of: {1:.3f}, std: (+/-) {2:.3f}
[{0}]".format(clf.__class__.__name__,vanilla_scores.mean(), vanilla_scores.std()))
   print ("Mean of: {1:.3f}, std: (+/-) {2:.3f} [Bagging {0}]\
n".format(clf.__class__.__name__,bagging_scores.mean(), bagging_scores.std()))

from sklearn.datasets import load_wine
# define dataset
X,y = load_wine().data,load_wine().target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
print(X_train.shape, X_test.shape)

from sklearn.ensemble import AdaBoostClassifier
```

```
ada_boost = AdaBoostClassifier(random_state=1)
ada_boost.fit(X_train, y_train)
print('AdaBoost accuracy: ', ada_boost.score(X_test,y_test))

from sklearn.ensemble import GradientBoostingClassifier
grad_boost= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
grad_boost.fit(X_train, y_train)
print('Gradient Boost accuracy: ', grad_boost.score(X_test,y_test))

import xgboost as xgb
xgb_boost=xgb.XGBClassifier(random_state=1,learning_rate=0.01)
xgb_boost.fit(X_train, y_train)
print('Extreme Gradient Boost accuracy: ', xgb_boost.score(X_test,y_test))
```

Результат виконання:





```
[('LR', LogisticRegression(max_iter=5000, multi_class='multinomial')), ('SVC', SVC
(gamma='auto', probability=True)), ('DTC', DecisionTreeClassifier())]
Hard Voting Score 0.9649122807017544
Soft Voting Score 0.9736842105263158
Best weights = [0.05, 0.45, 0.5], Best accuracy = 0.956
C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\lin
ear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Blending accuracy:  0.9166666666666666
Mean of: 0.977, std: (+/-) 0.028 [RandomForestClassifier]
Mean of: 0.983, std: (+/-) 0.025 [Bagging RandomForestClassifier]

Mean of: 0.989, std: (+/-) 0.022 [ExtraTreesClassifier]
Mean of: 0.989, std: (+/-) 0.022 [Bagging ExtraTreesClassifier]

Mean of: 0.675, std: (+/-) 0.070 [KNeighborsClassifier]
Mean of: 0.827, std: (+/-) 0.091 [Bagging KNeighborsClassifier]

Mean of: 0.681, std: (+/-) 0.087 [SVC]
Mean of: 0.675, std: (+/-) 0.056 [Bagging SVC]

Mean of: 0.983, std: (+/-) 0.025 [RidgeClassifier]
Mean of: 0.972, std: (+/-) 0.037 [Bagging RidgeClassifier]

(142, 13) (36, 13)
AdaBoost accuracy:  0.8333333333333334
Gradient Boost accuracy:  0.9444444444444444
Extreme Gradient Boost accuracy:  0.9166666666666666
Press any key to continue . . .
```

Бачимо, що в середньому ансамблювання працює краще за бустинг, тобто поєднання моделей на цих даних спрацювало краще, ніж бустинг. Але, взагалом, загального сильного покращення результатів не спостерігається.

Висновок: Під час виконання лабораторної роботи було виявлено:
1) Зазвичай тюнінг покращує результати;
2) Ансамблювання в середньому краще за бустинг;
3) Ансамблювання та бустинг не набагато кращі за тюнінг.