НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №4 з дисципліни

«Програмування інтелектуальних інформаційних систем»

Прийняв

аспірант кафедри ІПІ

Баришич Л.М.

Виконав

студент групи ІП-12

Казаков В.О.

“10” листопада 2023 р.

Київ – 2023

Імпорт необхідних модулів:

import warnings

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import VotingClassifier

from sklearn.datasets import load\_wine

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.linear\_model import RidgeClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBClassifier

from sklearn.linear\_model import Ridge, Lasso, LogisticRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.base import BaseEstimator, RegressorMixin, clone

Підготовка даних:

warnings.filterwarnings(action='ignore')

# Data preparation

df = pd.read\_csv("data.csv")

df.drop(['Unnamed: 32', 'id'], axis=1, inplace=True)

df['diagnosis']=df['diagnosis'].astype('category').cat.codes

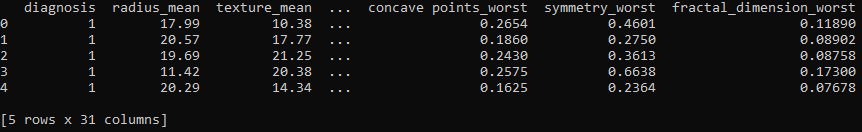
X = df.drop(['diagnosis'], axis = 1)

y = df['diagnosis']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size =

0.3, random\_state = 42)

print(df.head())



# Decision tree

base\_dt = DecisionTreeClassifier()

base\_dt.fit(X\_train, y\_train)

base\_dt\_y\_pred = base\_dt.predict(X\_test)

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]

}

grid\_obj = GridSearchCV(base\_dt, parameters)

grid\_obj = grid\_obj.fit(X\_train, y\_train)

tuned\_dt = grid\_obj.best\_estimator\_

tuned\_dt.fit(X\_train, y\_train)

tuned\_dt\_y\_pred = tuned\_dt.predict(X\_test)

acc\_base\_dt = round(metrics.accuracy\_score(y\_test, base\_dt\_y\_pred) \*

100, 2)

acc\_tuned\_dt = round(metrics.accuracy\_score(y\_test, tuned\_dt\_y\_pred) \*

100, 2)

print('Accuracy of base Decision Tree model: ', acc\_base\_dt)

print('Accuracy of tuned Decision Tree model: ', acc\_tuned\_dt)



# Random forest

base\_rf = RandomForestClassifier()

base\_rf.fit(X\_train, y\_train)

base\_rf\_y\_pred = base\_rf.predict(X\_test)

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]

}

grid\_obj = GridSearchCV(base\_rf, parameters)

grid\_obj = grid\_obj.fit(X\_train, y\_train)

tuned\_rf = grid\_obj.best\_estimator\_

tuned\_rf.fit(X\_train, y\_train)

tuned\_rf\_y\_pred = tuned\_rf.predict(X\_test)

acc\_base\_rf = round(metrics.accuracy\_score(y\_test, base\_rf\_y\_pred) \*

100, 2)

acc\_tuned\_rf = round(metrics.accuracy\_score(y\_test, tuned\_rf\_y\_pred) \*

100, 2)

print('Accuracy of base Random Forest model: ', acc\_base\_rf)

print('Accuracy of tuned Random Forest model: ', acc\_tuned\_rf)



# Support Vector Machine

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

base\_svc = SVC()

base\_svc.fit(X\_train, y\_train)

base\_svc\_y\_pred = base\_svc.predict(X\_test)

parameters = [

{'C': [1, 10, 100, 1000], 'kernel': ['linear']},

{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel':

['rbf']},

]

grid\_obj = GridSearchCV(base\_svc, parameters)

grid\_obj = grid\_obj.fit(X\_train, y\_train)

tuned\_svc = grid\_obj.best\_estimator\_

tuned\_svc.fit(X\_train, y\_train)

tuned\_svc\_y\_pred = tuned\_svc.predict(X\_test)

acc\_base\_svc = round(metrics.accuracy\_score(y\_test, base\_svc\_y\_pred) \*

100, 2)

acc\_tuned\_svc = round(metrics.accuracy\_score(y\_test, tuned\_svc\_y\_pred)

\* 100, 2)

print('Accuracy of base SVC model: ', acc\_base\_svc)

print('Accuracy of tuned SVC model: ', acc\_tuned\_svc)



# K-Nearest Neighbors

base\_knn = KNeighborsClassifier()

base\_knn.fit(X\_train, y\_train)

base\_knn\_y\_pred = base\_knn.predict(X\_test)

parameters = {'n\_neighbors': [3, 4, 5, 10],

'weights': ['uniform', 'distance'],

'algorithm' : ['auto', 'ball\_tree', 'kd\_tree', 'brute'],

'leaf\_size' : [10, 20, 30, 50]

}

grid\_obj = GridSearchCV(base\_knn, parameters)

grid\_obj = grid\_obj.fit(X\_train, y\_train)

tuned\_knn = grid\_obj.best\_estimator\_

tuned\_knn.fit(X\_train, y\_train)

tuned\_knn\_y\_pred = tuned\_knn.predict(X\_test)

acc\_base\_knn = round(metrics.accuracy\_score(y\_test, base\_knn\_y\_pred) \*

100, 2)

acc\_tuned\_knn = round(metrics.accuracy\_score(y\_test, tuned\_knn\_y\_pred)

\* 100, 2)

print('Accuracy of base KNN model: ', acc\_base\_knn)

print('Accuracy of tuned KNN model: ', acc\_tuned\_knn)



# Max Voting

estimators = []

estimators.append(('LR', LogisticRegression(solver='lbfgs',

multi\_class='multinomial', max\_iter=200)))

estimators.append(('SVC', SVC(gamma='auto', probability=True)))

estimators.append(('DTC', DecisionTreeClassifier()))

hard\_voting = VotingClassifier(estimators=estimators, voting='hard')

hard\_voting.fit(X\_train, y\_train)

y\_pred = hard\_voting.predict(X\_test)

score = metrics.accuracy\_score(y\_test, y\_pred)

print("Accuracy of Hard Voting model: %f" % score)

soft\_voting = VotingClassifier(estimators=estimators, voting='soft')

soft\_voting.fit(X\_train, y\_train)

y\_pred = soft\_voting.predict(X\_test)

score = metrics.accuracy\_score(y\_test, y\_pred)

print("Accuracy of Soft Voting model: %f" % score)



# Weighted Averaging

class AverageWeight(BaseEstimator, RegressorMixin):

def \_\_init\_\_(self,model,weight):

self.model = model

self.weight = weight

def fit(self,X,y):

self.models\_ = [clone(x) for x in self.model]

for model in self.models\_:

model.fit(X,y)

return self

def predict(self,X):

w = list()

pred = np.array([model.predict(X) for model in self.models\_])

# for every data point, single model prediction times weight, then add them together

for data in range(pred.shape[1]):

single = [pred[model,data]\*weight for model,weight in zip(range(pred.shape[0]), self.weight)]

w.append(np.sum(single))

return w

def rmse\_cv(model,X,y):

rmse = np.sqrt(-cross\_val\_score(model,X,y,scoring="neg\_mean\_squared\_error",cv=5))

return rmse

estimators = []

estimators.append(LogisticRegression())

estimators.append(DecisionTreeRegressor())

estimators.append(Lasso())

estimators.append(Ridge())

w1 = 0.2

w2 = 0.3

w3 = 0.4

w4 = 0.1

weight\_avg = AverageWeight(model=estimators, weight=[w1, w2, w3, w4])

score = rmse\_cv(weight\_avg, X, y)

print("Accuracy of Weighted Averaging model: %f" % score.mean())



# Blending

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train,

test\_size=0.25, random\_state=42)

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("Accuracy of Blending model: ", model.score(df\_test, y\_test))



# Bagging

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()))

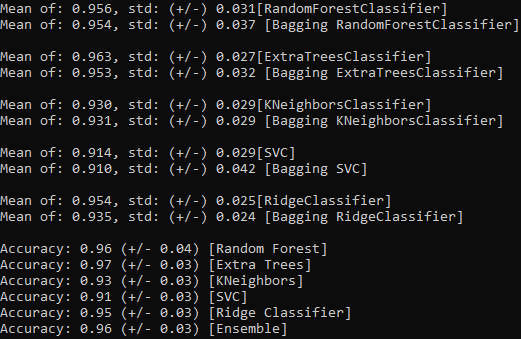
clf = [rf, et, knn, svc, rg]

eclf = VotingClassifier(estimators=[('Random Forests', rf), ('Extra Trees', et), ('KNeighbors', knn), ('SVC', svc), ('Ridge Classifier', rg)], voting='hard')

for clf, label in zip([rf, et, knn, svc, rg, eclf], ['Random Forest', 'Extra Trees', 'KNeighbors', 'SVC', 'Ridge Classifier', 'Ensemble']):

scores = cross\_val\_score(clf, X, y, cv=10, scoring='accuracy')

print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))



# Boosting

ada\_boost = AdaBoostClassifier(random\_state=42)

ada\_boost.fit(X\_train, y\_train)

ada\_boost.score(X\_test,y\_test)

grad\_boost=GradientBoostingClassifier(learning\_rate=0.01,random\_state=42)

grad\_boost.fit(X\_train, y\_train)

grad\_boost.score(X\_test,y\_test)

xgb\_boost=XGBClassifier(random\_state=1,learning\_rate=0.01)

xgb\_boost.fit(X\_train, y\_train)

xgb\_boost.score(X\_test,y\_test)

eclf = VotingClassifier(estimators=[('Ada Boost', ada\_boost), ('Grad Boost', grad\_boost), ('XG Boost', xgb\_boost)], voting='hard')

clf = [rf, et, knn, svc, rg]

for clf, label in zip([ada\_boost, grad\_boost, xgb\_boost,eclf], ['Ada Boost','Grad Boost','XG Boost','Ensemble']):

scores = cross\_val\_score(clf, X, y, cv=10, scoring='accuracy')

print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

