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»

Виконав студент ІП-12 Басараб Олег Андрійович

Перевірив Баришич Лука Маріянович

Київ 2023

Lab-4

```
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,
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

warnings.filterwarnings(action='ignore')
seed = 42
```

Import data and data preprocessing

```
df = pd.read_csv("resources/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 = seed)

df.head()
```

```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
```

```
3           1        11.42          20.38             77.58      386.1
4           1        20.29          14.34            135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33           184.60
1         0.1812  ...         24.99          23.41           158.80
2         0.2069  ...         23.57          25.53           152.50
3         0.2597  ...         14.91          26.50            98.87
4         0.1809  ...         22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 31 columns]
```

Hyperparameter tuning

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)
```

```
Accuracy of base Decision Tree model:  94.15
Accuracy of tuned Decision Tree model:  97.08
```

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)

Accuracy of base Random Forest model:  96.49
Accuracy of tuned Random Forest model:  97.08
```

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)

Accuracy of base SVC model:  97.66
Accuracy of tuned SVC model:  97.66
```

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)
```

```
Accuracy of base KNN model:  95.91
Accuracy of tuned KNN model:  95.91
```

Tuning results

```
models = pd.DataFrame({
    'Model': ['Base Decision Tree', 'Tuned Decision Tree', 'Base
Random Forest', 'Tuned Random Forest',
              'Base Support Vector Machines', 'Tuned Support Vector
Machines', 'Base K-Nearest Neighbors', 'Tuned K-Nearest Neighbors'],
    'Accuracy': [acc_base_dt, acc_tuned_dt, acc_base_rf, acc_tuned_rf,

              acc_base_svc, acc_tuned_svc, acc_base_knn,
acc_tuned_knn]})

models.sort_values(by='Accuracy', ascending=False)
```

	Model	Accuracy
4	Base Support Vector Machines	97.66
5	Tuned Support Vector Machines	97.66
1	Tuned Decision Tree	97.08
3	Tuned Random Forest	97.08
2	Base Random Forest	96.49
6	Base K-Nearest Neighbors	95.91
7	Tuned K-Nearest Neighbors	95.91
0	Base Decision Tree	94.15

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)
```

```
Accuracy of Hard Voting model: 0.988304
Accuracy of Soft Voting model: 0.988304
```

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())
```

```
Accuracy of Weighted Averaging model: 0.234249
```

Blending

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=seed)

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))
```

```
Accuracy of Blending model:  0.9941520467836257
```

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=seed)
    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()))

Mean of: 0.961, std: (+/-) 0.030 [RandomForestClassifier]
Mean of: 0.954, std: (+/-) 0.037 [Bagging RandomForestClassifier]

Mean of: 0.967, std: (+/-) 0.028 [ExtraTreesClassifier]
Mean of: 0.953, std: (+/-) 0.032 [Bagging ExtraTreesClassifier]

Mean of: 0.930, std: (+/-) 0.029 [KNeighborsClassifier]
Mean of: 0.931, std: (+/-) 0.029 [Bagging KNeighborsClassifier]

Mean of: 0.914, std: (+/-) 0.029 [SVC]
Mean of: 0.910, std: (+/-) 0.042 [Bagging SVC]

Mean of: 0.954, std: (+/-) 0.025 [RidgeClassifier]
Mean of: 0.935, std: (+/-) 0.024 [Bagging RidgeClassifier]


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))

Accuracy: 0.96 (+/- 0.03) [Random Forest]
Accuracy: 0.96 (+/- 0.02) [Extra Trees]
```

```
Accuracy: 0.93 (+/- 0.03) [KNeighbors]
Accuracy: 0.91 (+/- 0.03) [SVC]
Accuracy: 0.95 (+/- 0.03) [Ridge Classifier]
Accuracy: 0.96 (+/- 0.02) [Ensemble]
```

Boosting

```
ada_boost = AdaBoostClassifier(random_state=seed)
ada_boost.fit(X_train, y_train)
ada_boost.score(X_test,y_test)

grad_boost=
GradientBoostingClassifier(learning_rate=0.01,random_state=seed)
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))
```

```
Accuracy: 0.96 (+/- 0.03) [Ada Boost]
Accuracy: 0.95 (+/- 0.02) [Grad Boost]
Accuracy: 0.95 (+/- 0.02) [XG Boost]
Accuracy: 0.96 (+/- 0.02) [Ensemble]
```