Instituto de Biomecánica Clínica

Pre-Fall – Sistema inteligente para la prevención y predicción de caídas

E3.2 – Modelos iniciales de aprendizaje automático

Proyecto	Pre-Fall – Sistema inteligente para la prevención y predicción de caídas
Entregable	E3.2 – Modelos iniciales de aprendizaje automático

Contenido

1 Introducción

Este entregable está enmarcado en la tarea "T3.2: Modelos iniciales de aprendizaje automático", perteneciente al paquete de trabajo "PT3 – Sistema experto de prevención de caídas" dentro del proyecto PRE-FALL. En este documento se presentarán las secciones más relevantes del software entregado.

2 Descripción del código

Para llevar a cabo el proceso de entrenamiento de los modelos de clasificación de machine learning se ha utilizado un script el lenguaje Python en el que se realiza un proceso de *Grid Search* para encontrar el modelo y configuración óptimos para el problema actual.

```
# Obtener los argumentos del script
argumentos = sys.argv[1:]

# Número requerido de argumentos
num_arg_requeridos = 2


# Verificar que se han pasado el número correcto de argumentos
if len(argumentos) != num_arg_requeridos:
    print(f"Error: se requieren {num_arg_requeridos} argumentos, pero se han recibido {len(argumentos)}.")
    sys.exit()

elif argumentos[0]!='0' and argumentos[0]!='1':
    print(f"Error: El primer argumento debe tener valor 0 o 1.\n"+
    "El valor 0 indica que no existen datos de entrenamiento por lo que se crearán datos artificiales\n/"+
    "El valor 1 indica que existen datos de entrenamiento así que se tomarán de la carpeta train")
    sys.exit()

 #Definimos una variable apra el repositorio de datos
home='./Datos_generados/'
home_train=home+'/train/'
home_test=home+'/test/'
```

En primer lugar, se comprueba que los argumentos pasados al script son correctos y posteriormente el valor de los mismos, ya que estos determinan el modo de ejecución. En este caso se han diseñado dos modos:

- Modo artificial: es el que se emplea en esta configuración y consiste en tomar datos reales contenidos en la carpeta Datos_reales, crear datos artificiales a partir de estos tal y como se explica en el entregable E3.1 y finalmente realizar el proceso de *Grid Search* con la información artificial resultante.
- Modo normal: este modo toma datos de entrenamiento reales que se encuentren en la carpeta train y realizará el proceso de *Grid Search* con ellos. Esta opción se utilizará más adelante cuando se cuente con más ficheros de mediciones.

Para realizar el análisis de rendimiento de los diferentes modelos primero han de importarse las librerías pertinentes y preparar los datos.

```
# Grid search cross validation
from numpy import mean
from numpy import std
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn import metrics

print("\nGRID SEARCH\n")

features=X_train
labels=y_train

#Definimos una semilla
np.random.seed(1234)
```

A continuación, se presentan capturas de las secciones de entrenamiento con *Grid Search* para los diferentes tipos de modelos e hiperparámetros.

2.1 Regresión logística

```
#Regresión Logística
grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
logreg=LogisticRegression(random_state=0)
logreg_cv=GridSearchCV(logreg,grid,cv=5)
logreg_cv.fit(features,labels)

print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
#tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2'}
#accuracy : 0.7189542483660131

#Regresión Logística
logreg = LogisticRegression(C=logreg_cv.best_params_["C"],penalty=logreg_cv.best_params_["penalty"])
cv = KFold(n_splits=5, random_state=1, shuffle=True)
scores_RL = cross_val_score(logreg, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores_RL), std(scores_RL)))
```

2.2 Naive-Bayes

```
#Naive-Bayes
grid={'var_smoothing': np.logspace(0,-9, num=100)}# l1 lasso l2 ridge
gnb=GaussianNB()
gnb_cv=GridSearchCV(gnb,grid,scoring='accuracy',cv=5)
gnb_cv.fit(features,labels)

print("tuned hpyerparameters :(best parameters) ",gnb_cv.best_params_)
print("accuracy :",gnb_cv.best_score_)



gnb = GaussianNB(var_smoothing= gnb_cv.best_params_["var_smoothing"])
cv = KFold(n_splits=5, random_state=1, shuffle=True)
scores_GNB = cross_val_score(gnb, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores_GNB), std(scores_GNB)))
```

2.3 Random Forest

```
#Random Forest

param_grid = [
{'n_estimators': list(range(1,7)), 'max_features': list(range(1,15)),
 'max_depth': [10, 20,30,40,50, None], 'bootstrap': [True, False]}
]
forest=RandomForestClassifier(random_state=0)
grid_search_forest = GridSearchCV(forest, param_grid, cv=5)
grid_search_forest.fit(features, labels)
#find the best model of grid search
grid_search_forest.best_estimator_
#bootstrap=False, max_depth=20, max_features=13,
                    #n_estimators=6, random_state=0
RF = RandomForestClassifier(bootstrap = grid_search_forest.best_params_["bootstrap"], max_depth=grid_search_forest.best_params_["max_depth"],
max_features=grid_search_forest.best_params_["max_features"],
                        n_estimators=grid_search_forest.best_params_["n_estimators"], random_state=0)
cv = KFold(n_splits=5, random_state=1, shuffle=True)
scores_RF = cross_val_score(RF, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores_RF), std(scores_RF)))
```

2.4 Neural Network

```
#Neural Network
param_grid = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
NN = MLPClassifier()
grid_search_nn = GridSearchCV(NN, param_grid, cv=5,refit=True,verbose=2)
grid_search_nn.fit(features, labels)
#find the best model of grid search
grid_search_nn.best_estimator_
NN = MLPClassifier(activation=grid_search_nn.best_params_["activation"],
                   solver=grid_search_nn.best_params_["solver"],
                   hidden_layer_sizes=grid_search_nn.best_params_["hidden_layer_sizes"],
                   alpha=grid_search_nn.best_params_["alpha"],
                   learning_rate=grid_search_nn.best_params_["learning_rate"],
                   random_state=1)
cv = KFold(n_splits=5, random_state=1, shuffle=True)
scores_NN = cross_val_score(NN, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores_NN), std(scores_NN)))
```

2.5 KNN

```
#K-NN

param_grid = [
{'n_neighbors': list(range(1,100))}
]
knn = KNeighborsClassifier()
grid_search_knn = GridSearchCV(knn, param_grid, cv=5)
grid_search_knn.fit(features, labels)
#find the best model of grid search
grid_search_knn.best_estimator_

knn = KNeighborsClassifier(n_neighbors=grid_search_knn.best_params_["n_neighbors"])
cv = KFold(n_splits=5, random_state=1, shuffle=True)
scores_KNN = cross_val_score(knn, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores_KNN), std(scores_KNN)))
```

2.6 Árbol de decisión

```
#Árbol de decisión

param_grid = {"max_depth": range(1, 15),
              "min_samples_leaf": range(10, 50, 5),
              "min_samples_split": range(10, 50, 5),
              "criterion": ['gini', 'entropy']}

n_folds = 5

# Instantiate the grid search model
dtree = DecisionTreeClassifier(random_state=0)
grid_search_tree = GridSearchCV(estimator = dtree,
                          param_grid = param_grid,scoring='accuracy',
                          cv = n_folds)

# Fit the grid Search to the data
grid_search_tree.fit(features,labels)
print("best accuracy: ", grid_search_tree.best_score_)
print(grid_search_tree.best_estimator_)

tree = DecisionTreeClassifier(max_depth=grid_search_tree.best_params_["max_depth"],
                              min_samples_leaf=grid_search_tree.best_params_["min_samples_leaf"],
                              min_samples_split=grid_search_tree.best_params_["min_samples_split"],
                              criterion=grid_search_tree.best_params_["criterion"],
                         random_state=0)
scores_DT = cross_val_score(tree, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores_DT), std(scores_DT)))
```

2.7 XGBoost

```
#XGBoost
estimator = XGBClassifier(
    objective= 'binary:logistic',
    nthread=4,
    seed=42
)
parameters = {"booster": ['gbtree', 'gblinear','dart'],
              "eta":[0.3,0.6,1],
              "gamma":[25,50],
    'max_depth': [1,5,10],
    'n_estimators': [50,100]
}

grid_search_xgb = GridSearchCV(
    estimator=estimator,
    param_grid=parameters,scoring='accuracy',
    verbose=True
)
grid_search_xgb.fit(features,labels)
print("best accuracy: ", grid_search_xgb.best_score_)
print(grid_search_xgb.best_estimator_)

xgb = XGBClassifier(base_score=0.5, booster=grid_search_xgb.best_params_["booster"], colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None,
              enable_categorical=False, eta=grid_search_xgb.best_params_["eta"], gamma=grid_search_xgb.best_params_["gamma"], gpu_id=-1,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.5, max_delta_step=None, max_depth=grid_search_xgb.best_params_["max_depth"],
              min_child_weight=None,  monotone_constraints=None,
              n_estimators=grid_search_xgb.best_params_["n_estimators"], n_jobs=4, nthread=4, num_parallel_tree=None,
              predictor=None, random_state=42, reg_alpha=0, reg_lambda=0,
              scale_pos_weight=1, seed=42, subsample=None, tree_method=None,
              validate_parameters=1)

scores_XGB = cross_val_score(xgb, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.3f (%.3f)' % (mean(scores_XGB), std(scores_XGB)))
```

2.8 SVM

```
#SVM
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(features,labels)
print(grid.best_estimator_)


svc = SVC(C=grid.best_params_["C"], gamma=grid.best_params_["gamma"])
scores_SVM = cross_val_score(svc, features, labels, scoring='accuracy', cv=cv, n_jobs=-1)
print('Accuracy: %.3f (%.3f)' % (mean(scores_SVM), std(scores_SVM)))
```

2.9 Resultados

Finalmente, si se desea se pueden imprimir por pantalla los resultados del proceso de *Grid Search*.

```
#accuracy
print("RL tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print('Accuracy RL: %.3f (%.3f)' % (mean(scores_RL), std(scores_RL)))
print()
print("NB tuned hpyerparameters :(best parameters) ",gnb_cv.best_params_)
print('Accuracy NB: %.3f (%.3f)' % (mean(scores_GNB), std(scores_GNB)))
print()
print("RF tuned hpyerparameters :(best parameters) ",grid_search_forest.best_params_)
print('Accuracy RF: %.3f (%.3f)' % (mean(scores_RF), std(scores_RF)))
print()
print("NN tuned hpyerparameters :(best parameters) ",grid_search_nn.best_params_)
print('Accuracy NN: %.3f (%.3f)' % (mean(scores_NN), std(scores_NN)))
print()
print("KNN tuned hpyerparameters :(best parameters) ",grid_search_knn.best_params_)
print('Accuracy KNN: %.3f (%.3f)' % (mean(scores_KNN), std(scores_KNN)))
print()
print("Decision Tree tuned hpyerparameters :(best parameters) ",grid_search_tree.best_params_)
print('Accuracy Decision Tree: %.3f (%.3f)' % (mean(scores_DT), std(scores_DT)))
print()
print("Bagging tuned hpyerparameters :(best parameters) ",grid_search_bag.best_params_)
print('Accuracy Bagging: %.3f (%.3f)' % (mean(scores_BAG), std(scores_BAG)))
print()
print("XGB tuned hpyerparameters :(best parameters) ",grid_search_xgb.best_params_)
print('Accuracy XGB: %.3f (%.3f)' % (mean(scores_XGB), std(scores_XGB)))
print()
print("SVM tuned hpyerparameters :(best parameters) ",grid.best_params_)
print('Accuracy SVM: %.3f (%.3f)' % (mean(scores_SVM), std(scores_SVM)))
```