homework_2(1)

February 25, 2023

[ ]:
```python
#Manuel Aragon
#EUID:ma1162
```

[ ]:
```python

```

[ ]:
```python
#loading Training Data
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load the digits dataset
digits = datasets.load_digits()
X = digits.images.reshape((len(digits.images), -1))
y = digits.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[ ]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Train the SVM with default parameters on the training data
clf = SVC(kernel='rbf')
clf.fit(X_train, y_train)

# Evaluate the default accuracy
default_accuracy = clf.score(X_test, y_test)
print("Default accuracy:", default_accuracy)

# Define the range of values for hyperparameters gamma and C
gamma_range = [10**i for i in range(-5, 6)]
C_range = [10**i for i in range(-5, 6)]

# Create a grid of hyperparameter values to search
param_grid = {'gamma': gamma_range, 'C': C_range}

# Perform a grid search to find the best hyperparameters
```

```
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Report the best hyperparameters and accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
print("Best hyperparameters:", best_params)
print("Tuned Accuracy (Validation):", best_accuracy)

# Retrain the SVM with the best hyperparameters on the training data.
best_clf = SVC(**best_params, kernel='rbf')
best_clf.fit(X_train, y_train)

# Evaluate final accuracy
final_accuracy = best_clf.score(X_test, y_test)
print("Final Accuracy (Test):", final_accuracy)
```

```
Default accuracy: 0.9916666666666667
Best hyperparameters: {'C': 1, 'gamma': 0.001}
Tuned Accuracy (Validation): 0.9909528648857917
Final Accuracy (Test): 0.9916666666666667
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier


# Train the KNN with default parameters on the training data
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)

# Evaluate the default accuracy on the validation set
default_accuracy = clf.score(X_test, y_test)
print("Default accuracy:", default_accuracy)

# Define the range of values for hyperparameter k
k_range = [1, 3, 4, 7, 9]

# Create a grid of hyperparameter values to search
param_grid = {'n_neighbors': k_range}

# Perform a grid search to find the best hyperparameters
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Report the best hyperparameters and accuracy on the validation set
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

```python
print("Best hyperparameters:", best_params)
print("Accuracy (Validation):", best_accuracy)

# Retrain the KNN with the best hyperparameters on the combined training and validation data
best_clf = KNeighborsClassifier(**best_params)
best_clf.fit(X_train, y_train)

# Evaluate final accuracy
final_accuracy = best_clf.score(X_test, y_test)
print("Final Accuracy (Test):", final_accuracy)
```

```
Default accuracy: 0.975
Best hyperparameters: {'n_neighbors': 3}
Accuracy (Validation): 0.98677651955091
Final Accuracy (Test): 0.9833333333333333
```

[ ]:

```python
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


# Train the decision tree with default parameters on the training data
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Evaluate the default accuracy on the validation set
default_accuracy = clf.score(X_test, y_test)
print("Default accuracy:", default_accuracy)

# Define the range of values for hyperparameter min_samples_split
min_samples_split_range = [3, 5, 7, 9]

# Create a grid of hyperparameter values to search
param_grid = {'min_samples_split': min_samples_split_range}

# Perform a grid search to find the best hyperparameters
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Report the best hyperparameters and accuracy on the validation set
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
print("Best hyperparameters:", best_params)
print("Accuracy (Validation):", best_accuracy)

# Retrain the decision tree with the best hyperparameters on the combined training and validation data
```

```
best_clf = DecisionTreeClassifier(**best_params)
best_clf.fit(X_train, y_train)

# Evaluate final accuracy
final_accuracy = best_clf.score(X_test, y_test)
print("Final Accuracy (Test):", final_accuracy)
```

```
Default accuracy: 0.8444444444444444
Best hyperparameters: {'min_samples_split': 5}
Accuracy (Validation): 0.845501838946961
Final Accuracy (Test): 0.8555555555555555
```

[ ]:
```
from sklearn.linear_model import LogisticRegression



# Train the logistic regression with default parameters on the training data
clf = LogisticRegression(penalty='l1', solver='liblinear')  #using liblinear in␣
↪order to be able to use l1
clf.fit(X_train, y_train)

# Evaluate the default accuracy on the validation set
default_accuracy = clf.score(X_test, y_test)
print("Default accuracy:", default_accuracy)

# Define the range of values for hyperparameter C
C_range = [10**i for i in range(-5, 6)]

# Create a grid of hyperparameter values to search
param_grid = {'C': C_range}

# Perform a grid search to find the best hyperparameters
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Report the best hyperparameters and accuracy on the validation set
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
print("Best hyperparameters:", best_params)
print("Accuracy (Validation):", best_accuracy)

# Retrain the logistic regression with the best hyperparameters on the combined␣
↪training and validation data
best_clf = LogisticRegression(penalty='l1', solver='saga', **best_params)
best_clf.fit(X_train, y_train)

# Evaluate final accuracy
final_accuracy = best_clf.score(X_test, y_test)
```

```python
print("Final Accuracy (Test):", final_accuracy)
```

```
Default accuracy: 0.95
Best hyperparameters: {'C': 0.1}
Accuracy (Validation): 0.9679829655439413
Final Accuracy (Test): 0.9583333333333334
```

```
/home/manix/.local/lib/python3.10/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
```

Model	Default validation accuracy	Tuned validation accuracy	Selected hyperparamaters	Final test set accuracy
SVM	0.9916666666666667	0.9909528648857917	'C': 1, 'gamma': 0.001	0.9916666666666667
k-NN	0.975	0.98677651955091	n_neighbors': 3	0.9833333333333333
Decision Trees	0.8583333333333333	0.8427192218350754	min_samples_split': 5	0.8638888888888889
Logistic Regression	0.9444444444444444	0.9573647237516614	'C': 0.1	0.9583333333333334

Refrences: https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html