NCTU Pattern Recognition, Homework 4

Deadline: May 25, 23:59

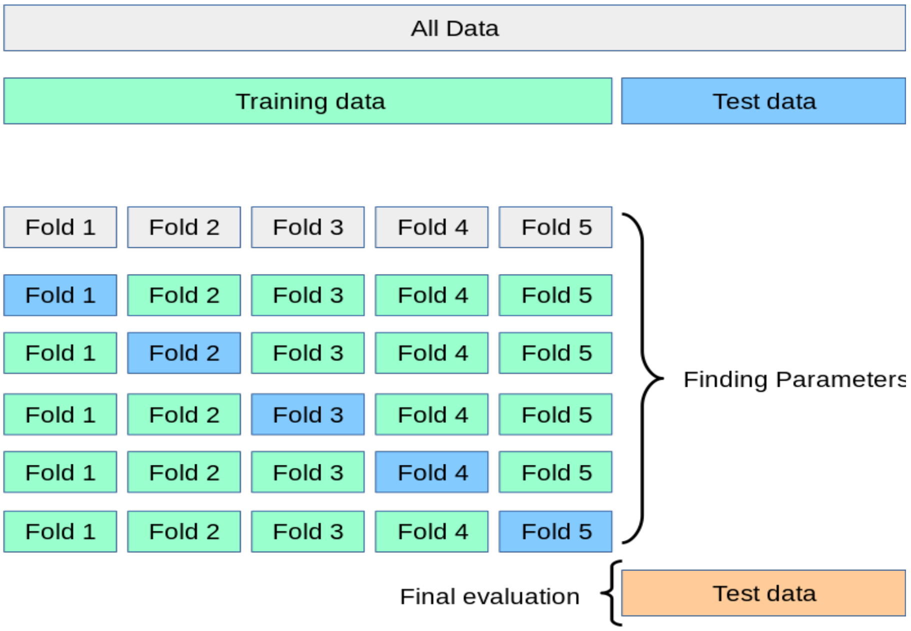
Part. 1, Coding (50%):

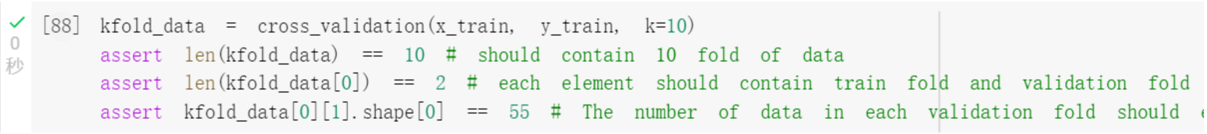
In this coding assignment, you need to implement the cross-validation and grid search using only NumPy, then train the SVM model from scikit-learn on the provided dataset and test the performance with testing data. Find the sample code and data on the GitHub page https://github.com/NCTU-VRDL/CS_AT0828/tree/main/HW4

Please note that only NumPy can be used to implement cross-validation and grid search. You will get no points by simply calling sklearn.model_selection.GridSearchCV.

1. (10%) K-fold data partition: Implement the K-fold cross-validation function. Your function should take K as an argument and return a list of lists (*len(list) should equal to K*), which contains K elements. Each element is a list containing two parts, the first part contains the index of all training folds (index_x_train, index_y_train), e.g., Fold 2 to Fold 5 in split 1. The second part contains the index of the validation fold, e.g., Fold 1 in split 1 (index_x_val, index_y_val)

   Note: You need to handle if the sample size is not divisible by K. Using the strategy from sklearn. The first n_samples % n_splits folds have size n_samples // n_splits + 1, other folds have size n_samples // n_splits, where n_samples is the number of samples, n_splits is K, % stands for modulus, // stands for integer division. See this post for more details
   Note: Each of the samples should be used **exactly once** as the validation data
   Note: Please **shuffle** your data before partition

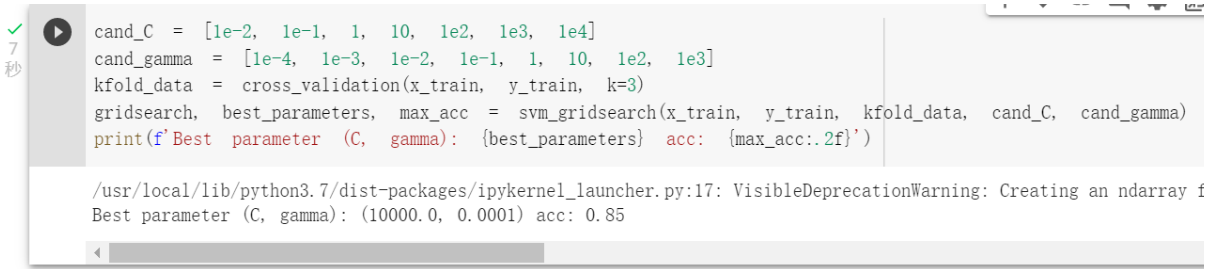




```
[88] kfold_data = cross_validation(x_train, y_train, k=10)
     assert len(kfold_data) == 10 # should contain 10 fold of data
     assert len(kfold_data[0]) == 2 # each element should contain train fold and validation fold
     assert kfold_data[0][1].shape[0] == 55 # The number of data in each validation fold should 
```

2. (20%) Grid Search & Cross-validation: using sklearn.svm.SVC to train a classifier on the provided train set and conduct the grid search of “C” and “gamma,” “kernel’ =’ rbf’ to find the best hyperparameters by cross-validation. Print the best hyperparameters you found.

Note: I use k=3, since k=5 or k=10 have lower performance on testing data.

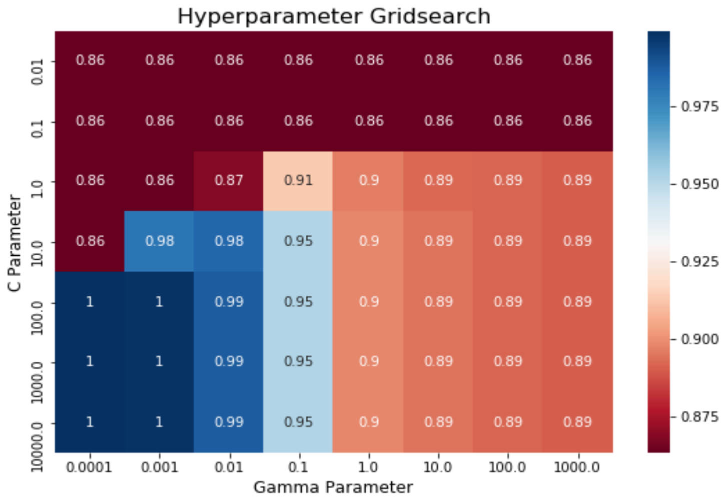


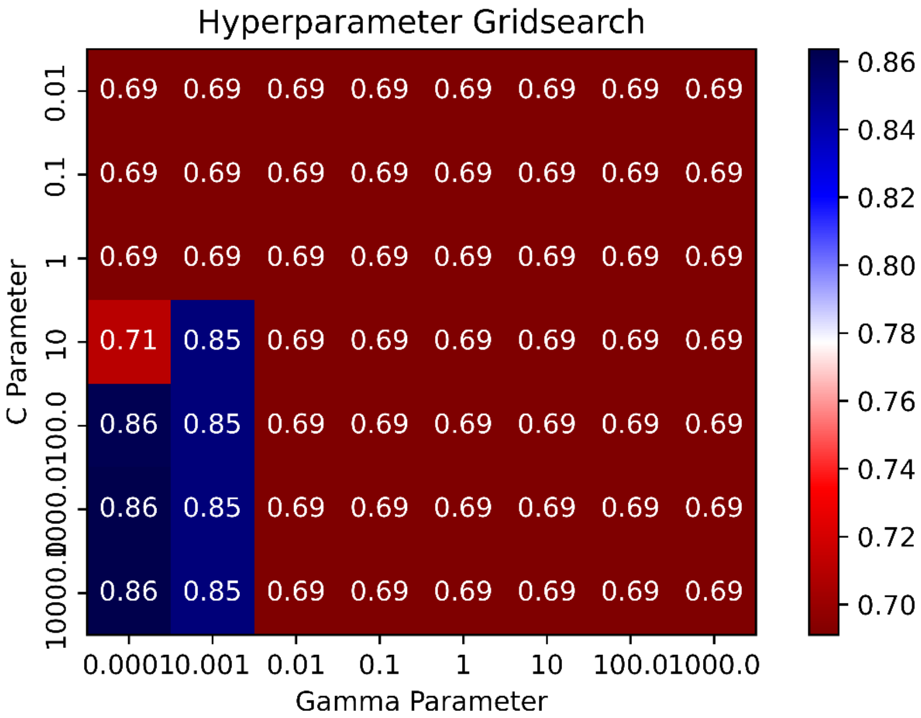
```
cand_C = [1e-2, 1e-1, 1, 10, 1e2, 1e3, 1e4]
cand_gamma = [1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 1e2, 1e3]
kfold_data = cross_validation(x_train, y_train, k=3)
gridsearch, best_parameters, max_acc = svm_gridsearch(x_train, y_train, kfold_data, cand_C, cand_gamma)
print(f'Best parameter (C, gamma): {best_parameters} acc: {max_acc:.2f}')

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: VisibleDeprecationWarning: Creating an ndarray f
Best parameter (C, gamma): (10000.0, 0.0001) acc: 0.85
```

3. (10%) Plot the grid search results of your SVM. The x and y represent “gamma” and “C” hyperparameters, respectively. And the color represents the average score of validation folds.

Note: This image is for reference, not the answer

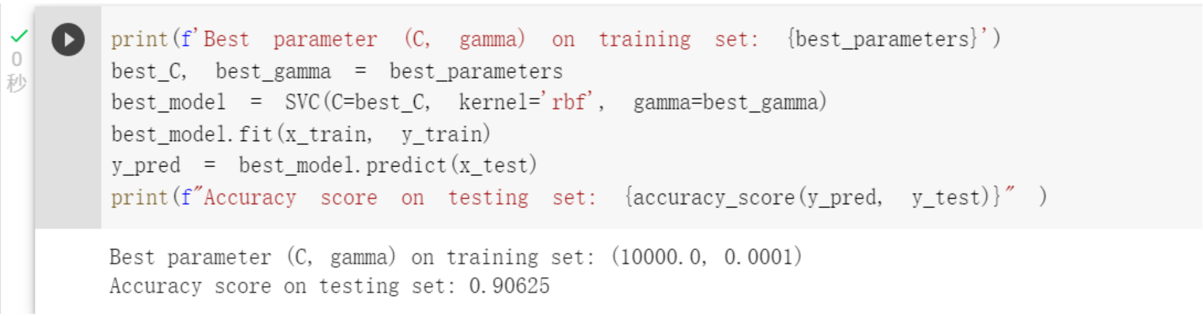
Note: matplotlib is allowed to use





4. (10%) Train your SVM model by the best hyperparameters you found from question 2 on the whole training data and evaluate the performance on the test set.

Accuracy	Your scores
acc > 0.9	10points
0.85 <= acc <= 0.9	5 points
acc < 0.85	0 points



```
print(f'Best parameter (C, gamma) on training set: {best_parameters}')
best_C, best_gamma = best_parameters
best_model = SVC(C=best_C, kernel='rbf', gamma=best_gamma)
best_model.fit(x_train, y_train)
y_pred = best_model.predict(x_test)
print(f"Accuracy score on testing set: {accuracy_score(y_pred, y_test)}" )
```

```
Best parameter (C, gamma) on training set: (10000.0, 0.0001)
Accuracy score on testing set: 0.90625
```