Xin Yi -asg1

February 2, 2020

1 Question 1

Load the Iris dataset and plot the pairs plot of the data, which includes the scatter plots of every dimension versus another dimension.

[1]:
```
# import the libraries
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import svm
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
# plot the pairs plot of Iris dataset
iris = sns.load_dataset("iris")
sns.set_style('whitegrid')
sns.pairplot(iris, hue='species')
plt.show()
```

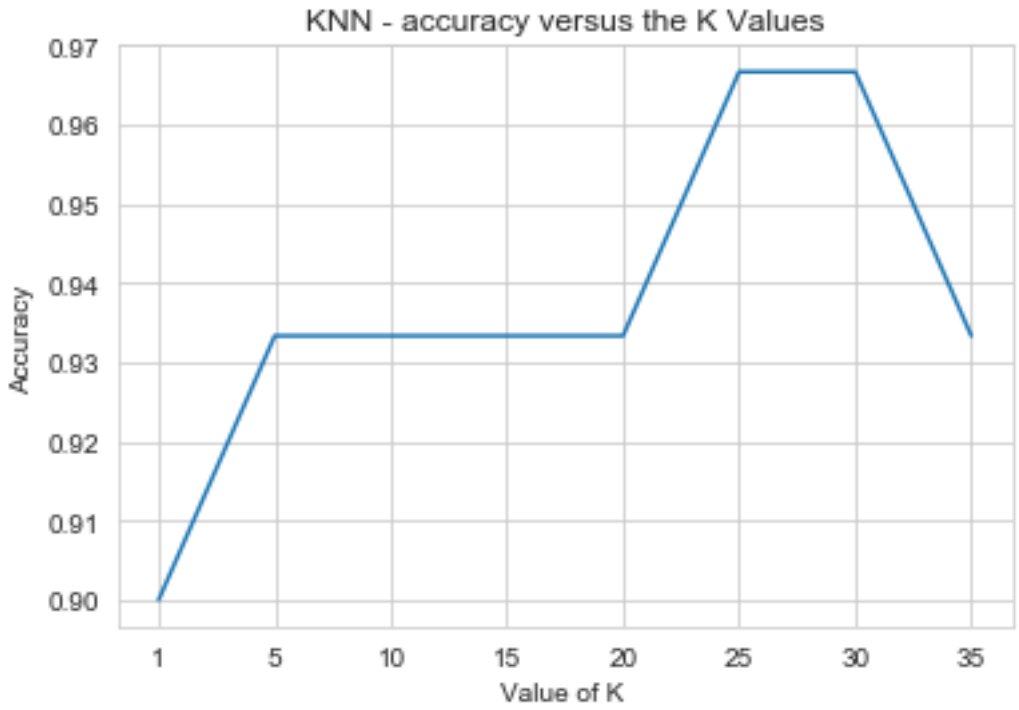
Pairplot gives a feature pairwise comparison graph, plus, the diagonal line shows the Kernel Density Estimates since x-axis and y-axis are the same feature. From the figure it can be seen that the three classes of dataset have certain separability in terms of feature pairs. Especially the setosa, which turns to be clustered far from the other two classes and is linear separable from them. While the versicolor and virginica seems not have a clear boundary and some of them are difficult to separate. For Kernel Density Estimates, it is also obvious that the setosa distributed differently from the other two classes in petal_length and petal_width especially(most setosa has smaller petal_length and peta_width). Such difference of density also shows in sepal_length and sepal_width, not so obvious but still can be observed, however, in terms of sepal_width, the three of them have more similar density distribution, especially versicolor and virginica.

2 Question 2: KNN

Classify the data using a KNN classifier. Tune the hyperparameters of the classifier using sklearn functions. Plot the different validation accuracies against the values of the parameter and select the best hyperparameter to train the model. Report the resulting accuracy.

[2]:
```
# load the dataset and divide the data into train, validation, and test sets␣
 ↪(60%, 20%, 20%)
iris_dataset= load_iris()
X_train_val,X_test,y_train_val,y_test =␣
 ↪train_test_split(iris_dataset['data'],iris_dataset['target'],test_size=0.
 ↪2,random_state=42)
X_train,X_val,y_train,y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25,random_state=42)

# train the with each classifier's default parameters using train set
knn_default = KNeighborsClassifier()
knn_default.fit(X_train,y_train)

# test the model on the test set and store the accuracy of the model as␣
 ↪"accuracy1"
y_pred1 = knn_default.predict(X_test)
accuracy1 = accuracy_score(y_test, y_pred1)
print("Accuracy (default k):",accuracy1)

# knn with different k values
k_values =[1, 5, 10, 15, 20, 25, 30, 35]
test_accuracy = np.empty(len(k_values))
for i,k in enumerate (k_values):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_val)
    test_accuracy[i] = accuracy_score(y_val,y_pred)

# plot the figure
plt.title('KNN - accuracy versus the K Values')
plt.xticks(np.arange(len(k_values)),k_values)
plt.plot(test_accuracy)
plt.xlabel('Value of K ')
plt.ylabel('Accuracy')
plt.show()

# find the best k from and train the model
best_k = 25
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train,y_train)

# test the model on the test set and find the test accuracy as "accuracy2"
y_pred2= best_knn.predict(X_test)
accuracy2 = accuracy_score(y_test, y_pred2)
print("Accuracy (best k):",accuracy2)
```

```
Accuracy (default k): 0.9666666666666667
```




```
Accuracy (best k): 1.0
```

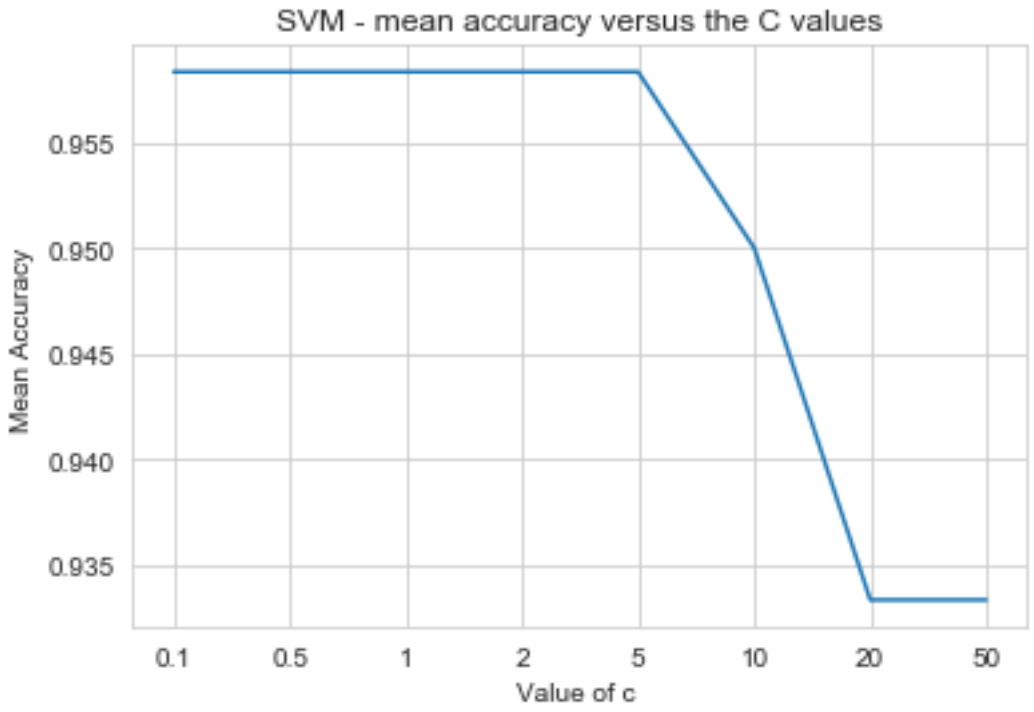
The figure show that when k=25 or k=30, the accuracy reaches its highest value. From these two options, here I choose K=25 as the best parmeter. Because an odd number seems more reasonable and avoid two classes labels achieving the same score. Using the K=25, train the model with training set and test the model on the test, the accuracy got turns to be 1.0.

3 Question 3: SVM

Classify data using a linear SVM classifier. Evaluate the best value for the term C, use 10-fold cross validation.For every C value, a mean accuracy of the folds is found. The best mean accuracy determines the best value for C. Plot the mean accuracy versus the C values. Finally, report the test accuracy.

[3]:
```
# linear SVM with different c values
c_values = [0.1, 0.5, 1, 2, 5, 10, 20, 50]
test_accuracy_c = np.empty(len(c_values))
for i,c in enumerate (c_values):
    svm_model = svm.SVC(kernel='linear', C=c,random_state = 42)
    # apply 10-fold cross validation on train-validation set and find a mean␣
 ↪accuracy for each c
    scores = cross_val_score(svm_model, X_train_val, y_train_val, cv=10)
    test_accuracy_c[i] = scores.mean()
```

```
# plot the figure
plt.title('SVM - mean accuracy versus the C values')
plt.xticks(np.arange(len(c_values)),c_values)
plt.plot(test_accuracy_c)
plt.xlabel('Value of c ')
plt.ylabel('Mean Accuracy')
plt.show()

best_c = 0.1
best_svm = svm.SVC(kernel='linear', C=best_c,random_state = 42)
best_svm.fit(X_train,y_train)

# test the model on the test set and find the test accuracy as "accuracy2"
y_predc= best_svm.predict(X_test)
accuracyc = accuracy_score(y_test, y_predc)
print("Accuracy (best c):", accuracyc)
```



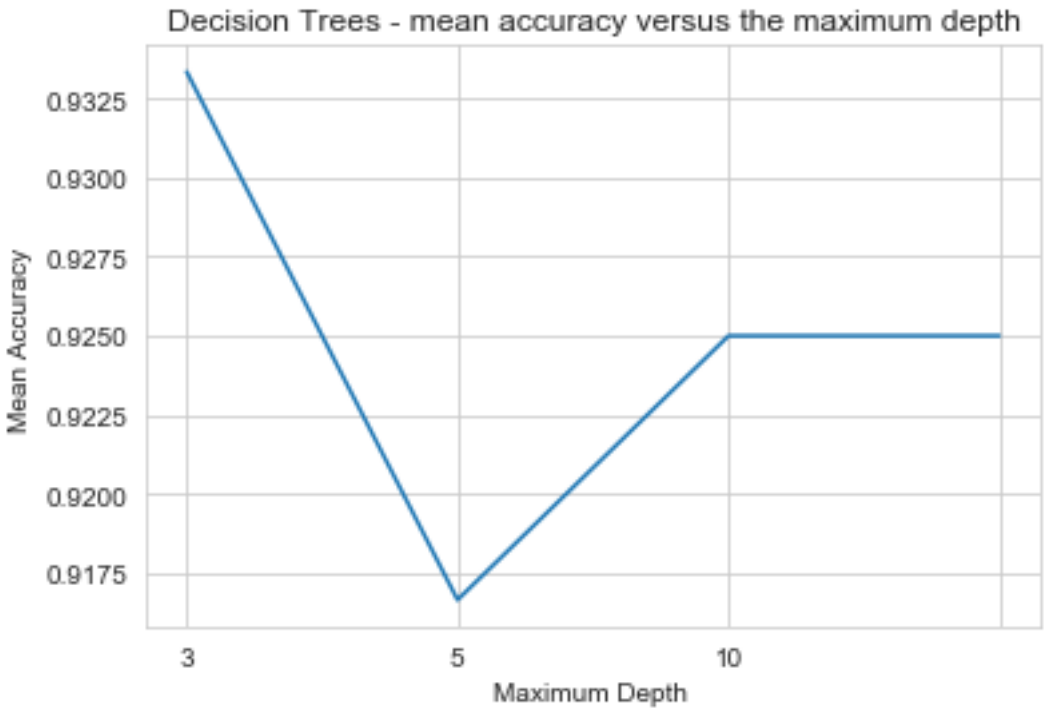

```
Accuracy (best c): 1.0
```

The figure shows the mean accuacy reaches the highest when c=0.1,0.5,1,2,5, and then decreases when value of c continues to increase. Here I choose the best value for C is C=0.1 because with the increase of C, there is no obvious improvement of the model. I trained the SVM model using train set with C=1 and got the test accuracy = 1.0.

4 Question 4: Tree-based Classifiers

Classify the data using three tree-based classifiers: Decision Trees, Random Forests and Gradient Tree Boosting. Tune the hyper-parameters of the classifier using 10-fold cross validation and sklearn functions (as in Question 3). Evaluate the best value for the number of trees and maximum depth of trees.

[4]:
```
# set the number of trees and max depths for tree-based classifiers
n_trees= [5, 10, 50, 150, 200]
m_depth = [3, 5, 10, None]

# decision trees
test_accuracy_d = np.empty(len(m_depth))
for i,m in enumerate (m_depth):
    dtc_model = tree.DecisionTreeClassifier(max_depth=m,random_state = 42)
    scores = cross_val_score(dtc_model, X_train_val, y_train_val,cv=10)
    test_accuracy_d[i] = scores.mean()

plt.title('Decision Trees - mean accuracy versus the maximum depth')
plt.xticks(np.arange(len(m_depth)),m_depth)
plt.xlabel('Maximum Depth ')
plt.ylabel('Mean Accuracy')
plt.plot(test_accuracy_d)
plt.show()

best_m = 3
best_dtc = tree.DecisionTreeClassifier(max_depth=best_m,random_state = 42)
best_dtc.fit(X_train,y_train)

# test the model on the test set and find the test accuracy as "accuracy2"
y_predd= best_dtc.predict(X_test)
accuracyd = accuracy_score(y_test, y_predd)
print("Accuracy (best max depth):", accuracyd)
```



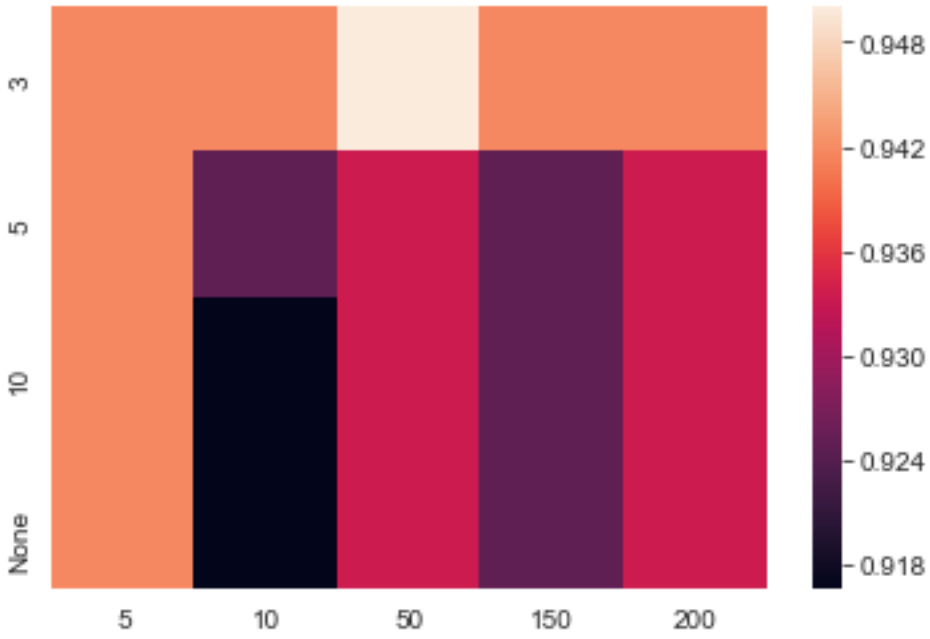

```
Accuracy (best max depth): 0.9666666666666667
```

For decision tree, when Maximum Depth is 3, the mean accuracy is the highest. So I choose the best parameter as max_depth=3, the test accuracy is 0.97.

[5]:
```
# random forest
test_accuracy_r= np.empty([len(m_depth),len(n_trees)])
for i,m in enumerate (m_depth):
    for j, n in enumerate (n_trees):
        rfc_model = RandomForestClassifier(n_estimators =␣
 ↪n,max_depth=m,random_state = 42)
        scores = cross_val_score(rfc_model, X_train_val, y_train_val,cv=10)
        test_accuracy_r[i][j]=scores.mean()

# plot the 5*4 heatmap

ax = sns.heatmap(test_accuracy_r,xticklabels = ['5', '10', '50', '150',␣
 ↪'200'],yticklabels = ['3', '5', '10', 'None'])

# fit with the best parameter and test the model on the test set and find the␣
 ↪test accuracy as "accuracyr"
best_rfc= RandomForestClassifier(n_estimators = 50,max_depth=3,random_state =␣
 ↪42)
best_rfc.fit(X_train,y_train)
```

```
y_predr= best_rfc.predict(X_test)
accuracyr = accuracy_score(y_test, y_predr)
print("Accuracy (best max depth and number of trees):", accuracyr)
```

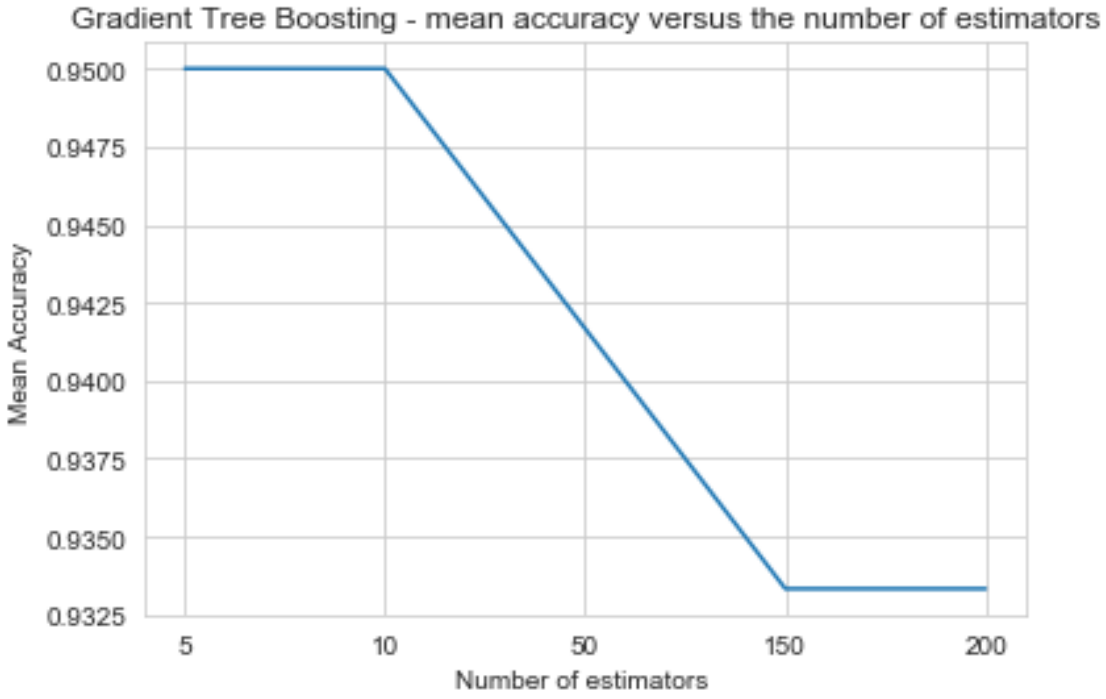
```
Accuracy (best max depth and number of trees): 1.0
```



For random forest, from the heatmap it can be seen when the parameters are: number of trees is 50, and the max depth is 3, the highest accuracy is reached.I trained the model with n_estimator=50, max_depth=3, test the model and get the accuracy=1.0.

[6]:
```
# gradient tree boosting
test_accuracy_g = np.empty([len(n_trees)])
for i,n in enumerate (n_trees):
    gbc_model = GradientBoostingClassifier(n_estimators = n,random_state = 42)
    scores = cross_val_score(gbc_model, X_train_val, y_train_val)
    test_accuracy_g[i]=scores.mean()

plt.title('Gradient Tree Boosting - mean accuracy versus the number of␣
 ↪estimators')
plt.xticks(np.arange(len(n_trees)),n_trees)
plt.xlabel('Number of estimators ')
plt.ylabel('Mean Accuracy')
plt.plot(test_accuracy_g)
plt.show()

best_gbc=  GradientBoostingClassifier(n_estimators = 10,random_state = 42)
```

```
best_gbc.fit(X_train,y_train)

# test the model on the test set and find the test accuracy as "accuracy2"
y_predg= best_gbc.predict(X_test)
accuracyg = accuracy_score(y_test, y_predg)
print("Accuracy (best number of trees):", accuracyg)
```




```
Accuracy (best number of trees): 0.9666666666666667
```

For gradient tree boosting, the mean accuracy reaches its best when number of trees is 5 or 10. I choose n_estimator=10 as the best parameter, since in Gradient boosting, n_estimator is the number of boosting stages to perform, a larger number usually results in better performance. The test accuracy is 0.97.

5 Question 5: Analysis

1.Explain why you had to split the dataset into train and test sets?

Training set is a set of samples to fit the model, which already has acutal value that the model should have predicted,the model learns from this data. After training, we need to know whether the model we trained is good or not, so we need test dataset which is never used in the model before, ohterwise it may cause that the model only perform well on predicting the value for the training set instead of make a general prediction.

2.Explain why when finding the best parameters for KNN you didn't evaluate directly on the test set and had to use a validation test.

The test set is used once the model is completedly trained with the train set and the validation set. It is also to improve generalization, when we are trying to evaluate a model, what we care about is how it performances on a new dataset, not the dataset that the model already knows. Since in the process of finding the best parameter, we need to use the performance of the model on the validation data as feedback. therefore, when we adjust the parameter based on the performance of the model on the validation set, it may also cause the model overfitting on the validation set, even if we did not directly train the model on the validation set, so we still need a test set which has nothing to do with the model to evaluate the performance.

3.What was the effect of changing k for KNN. Was the accuracy always affected the same way with an increase of k? Why do you think this happened?

Selecting an appropiate K is the key to do the classification correctly. The accuracy is not like always increase/decrease with the change of k, in this case, the accuracy rises along with the K increases to a certain value, and then decline. This is because when K is too small, the prediction results will be very sensitive to the neighbors, it is more likely to overfit, with the increase of K, the variance tends to be lower but if K is too large, when selecting nearest neighbors, the data which is not similar will also be included, which results in underfitting in prediction.

4.What was the relative effect of changing the max depths for decision tree and random forests? Explain the reason for this.

The max_depth parameter specifies the maximum depth of each tree for decision tree and random forests,the deeper the tree is, the more splits it has and it captures more information about the data. From the oberservation, here the accuracy decrease with the increasing of the max depth. The reason may be it caused overfitting, since the model becomes more complicated with the increase of the max depth. Although with the increase of max depth, the performance will become better on training set, it may not perform as well on testing set. Since the dataset is rather small, there may be no need to set the max depth as a very large value.

5.Comment on the effect of the number of estimators for Gradient Tree Boosting and what was the relative effect performance of gradient boosting compared with random forest. Explain the reason for this.

It seems that it is better to use fewer trees with Gradient Tree Boosting than with Random Forests,For Gradient Tree Boosting, when the number of estimators increases, the accuracy turns to be lower. While for Random Forest, the accuracy is not always decrease with the increase of number of estimators, the highest accuracy happens when the number reaches a certain value, with fluctuation. It may be because random forests are more not likely to overfit than gradient boosting trees. The gradient boosting tree is more sensitive to the noise, the boosting algorithm reduces error mainly by reducing bias, while random forest reduces error by reducing the variance, and when the variance is high, we think the model is more likey to be overfitting. Plus, in this question, we use max_depth in random forest, which can also avoid overfitting.

6.What does the parameter C define in the SVM classifier? What effect did you observe and why do you think this happened?

The C parameter ,as a regularization parameter in SVM, trades off correct classification of training examples against maximization of the decision function's margin.With larger C, the classifier focus on making fewer mistakes on the training data, which will cause overfitting. With a very small C, the classifier focus on a larger-margin separating hyperplane, while not care about misclassification too much, so the classification performance will be poor.From the obeservation, there are multiple

C values (from 0.1 to 5) turns to make the same accuracy on the train-validation set, I think one of the reason is when C=0.1, the SVM is already able to correctly make the classification, so when C increases, there is no obvious improvement on the dataset, therefore there is no need to make C larger.