I have used wine dataset from sklearn library which has a total of 178 data points with 13 features and the target has three different classes

Question 2:

(a) K-Nearest Neighbors (K-NN): find the optimal hyperparameters for the distance metric (i.e., test "Euclidean" and "Manhattan") and number of nearest neighbors (i.e., test at least 5 different values) when using k-Nearest Neighbors. Report the optimal hyperparameters found and how many hyperparameter combinations you tested in total. (Code and Write-up)

Ans. In order to find the best hyper parameter for KNN classifier I tried to train the model on a combination of different values of distance metric and number of nearest neighbors.

Type of Distance Metric Used: Manhattan (1) and Euclidean (2)
Number of nearest neighbors value: ranges from 3-15

In total there were 24 combinations of p and k that were tested. The best result was for p=1 and k=8 which gave a best score of 0.77 for KNN on stratifiedkfold validation.

b) Decision tree: find the optimal hyperparameters for the split criterion (i.e., test "gini" and "entropy") and tree depth (i.e., test at least 5 different values) when training a decision tree. Report the optimal hyperparameters found and how many hyperparameter combinations you tested in total. (Code and Write-up)

Ans. To get the best hyperparameter for decision tree classifier, I trained it on a combination of different criterion and max depth.

Criterion - Gini and Entropy
Max_Depth - ranges from 2-20

In total there were 36 combinations of criterion and depth that were tested. The best result was for criterion= gini and max_depth = 3 which gave a best score of 0.94 for decision tree classifier on stratifiedkfold validation.

c) Support Vector Machine (SVM): find the optimal hyperparameters for the polynomial degree, kernel bandwidth (i.e., gamma), and regularization parameter (i.e., C) when training a kernel SVM with a polynomial kernel. You must evaluate all possible combinations of at least 4 degree values (for the polynomial degree), at least 4 gamma values, and at least 4 C values. Report the optimal hyperparameters found and how many hyperparameter combinations you tested in total. (Code and Write-up)

Ans. To get the best hyperparameter for the SVM classifier, I trained it on a combination of different gamma , regularization C and polynomial degree values. Best hyperparameter are listed below:

Gamma - `[0.1,0.01,0.001,0.0001]`
Regularization - `[6,7,8,9,10]`
Polynomial Degree - `[2,3,4,5,6]`

In total there were 100 combinations of Gamma, Regularization and Polynomial Degree that were tested. The best result was for Gamma = 0.0001, Degree = 2 and C = 6 which gave a best score of 0.97 for SVM classifier on stratifiedkfold validation.

(b) Report the predictive performance on the test dataset for each of the four models from part (a) with respect to each of the following evaluation metrics: accuracy, precision, and recall. (Code and Write-up)

Ans. KNN Classifier

```
The accuracy score for KNN classifier is: 0.7777777777777778

The precision score for KNN classifier is:  0.7792397660818713

The recall score for KNN classifier is:  0.7777777777777778

The classification report is:

              precision    recall  f1-score   support

           0       0.90      0.95      0.92        19
           1       0.79      0.71      0.75        21
           2       0.60      0.64      0.62        14

    accuracy                           0.78        54
   macro avg       0.76      0.77      0.76        54
weighted avg       0.78      0.78      0.78        54
```

Decision Tree Classifier

```
The accuracy score for Decision Tree classifier is: 0.9444444444444444

The precision score for Decision Tree classifier is:  0.9513888888888888

The recall score for Decision Tree classifier is:  0.9444444444444444

The classification report is:

              precision    recall  f1-score   support

           0       1.00      0.89      0.94        19
           1       0.88      1.00      0.93        21
           2       1.00      0.93      0.96        14

    accuracy                           0.94        54
   macro avg       0.96      0.94      0.95        54
weighted avg       0.95      0.94      0.94        54
```

SVM Classifier

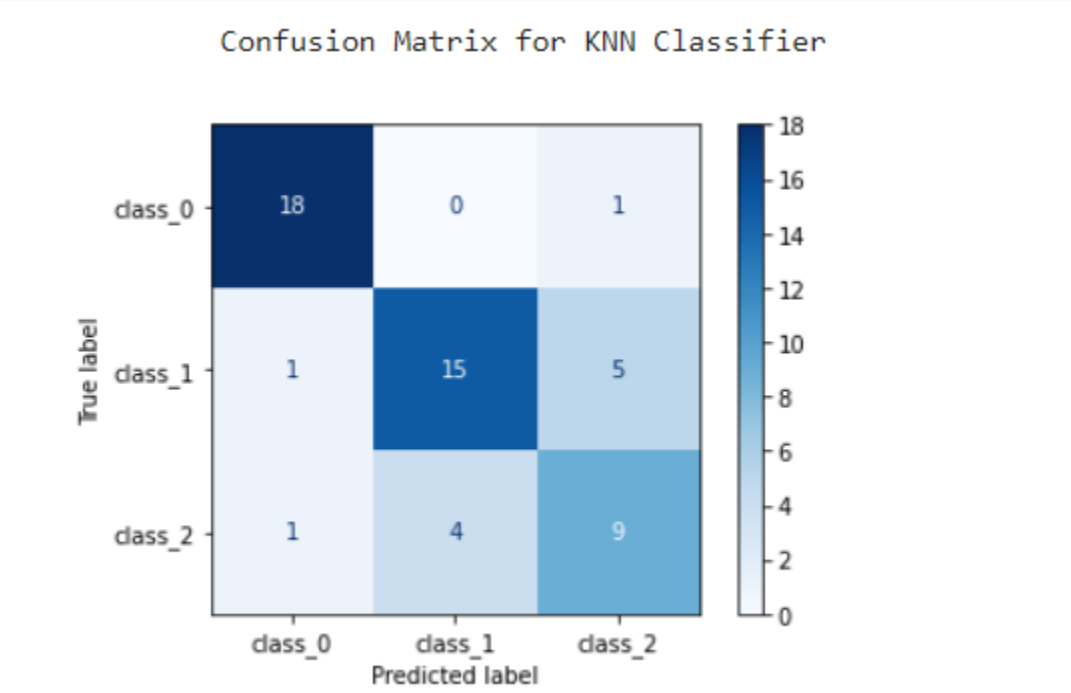
```
The accuracy score for SVM classifier is: 0.9814814814814815

The precision score for SVM classifier is:  0.9824074074074074

The recall score for SVM classifier is:  0.9814814814814815

The classification report is:

              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      0.95      0.98        21
           2       1.00      1.00      1.00        14

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

Naive Bayes Model

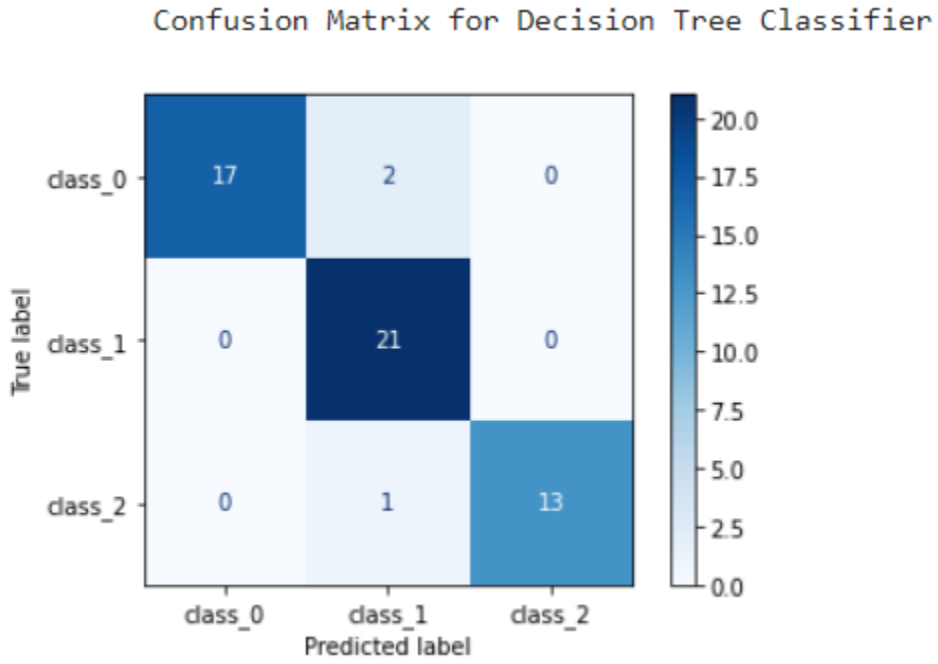
```
The accuracy score for Naive Bayes classifier is: 1.0

The precision score for Naive Bayes classifier is:  1.0

The recall score for Naive Bayes classifier is:  1.0

The classification report is:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54
```

(c) Visualize the predictive performance of each of the four models from part (a) by showing the resulting confusion matrix for each model. (Code and Write-up)

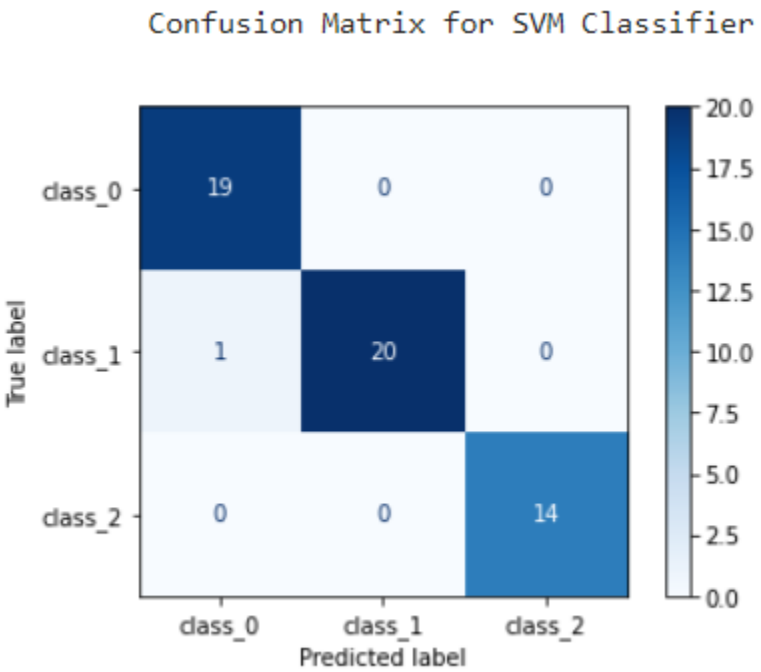
KNN Classifier



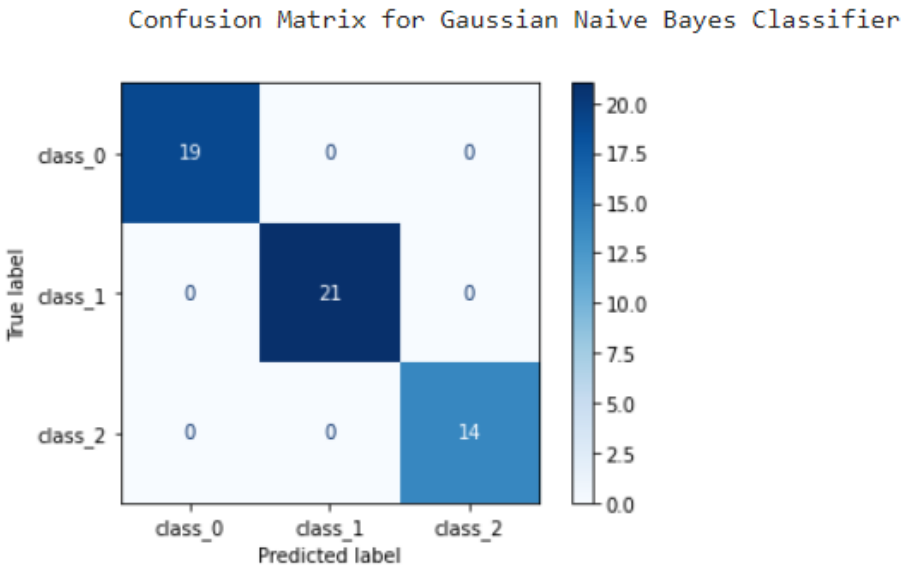
Decision Tree Classifier



SVM Classifier



Naive Bayes Model



(d) Write a discussion analyzing and comparing the performance of the four models. For example, which method(s) perform the best and why do you think so? Which method(s) perform the worst and why do you think so? What do the different performance metrics tell you about the results? Your discussion should consist of two to three paragraphs. (Write-up).

Ans. For the wine dataset, the SVM and Naive Bayes Classifier performed the best with test accuracy of 0.98 and 1.0 respectively. The decision tree classifier also performed comparatively better than KNN classifier with accuracy of 0.94 on the test set. One of the reasons that SVM works well on this dataset is because the data points are equally distributed in different classes. Since SVM classifiers work really well when classes in the data points are equally distributed it gives us better accuracy as compared to KNN Classifier and Decision Trees. Also, since we have used polynomial kernel based SVM it increases the chance that data will become linearly separable in this high-dimensional space.

The Naive Bayes Classifier actually works really well for multiclass prediction; it is working well for our dataset which has multiple classes. But since it assumes that all the predictors are independent of each other, it maye work better only when this assumption holds true, but in case the features are correlated then this model would not give us accurate results. The KNN Classifier performed the worst as compared to the other 3 models this may be because our predictors were not scaled before training the model. In case of KNN since the model uses euclidean or manhattan distance, it becomes necessary that features have the same scale, since absolute differences in features weigh the same.

The evaluation metric that is used for all the models are accuracy, precision and recall. Since this is a classification problem it is important to know the precision and recall alongwith the accuracy of the model in order to identify the percentage of misclassification. Precision tells us how accurate or precise our model is, it tells us how many data points are actually positive out of the total predicted positive. For our SVM classifier the weighted average of precision for all the three classes is 0.98, so 98% of the time model is able to predict how many positives are there out of the total positive predictions. Similarly for recall we actually calculate how many of the actual positives our model captures by labelling it as Positive. It is mostly used when the cost of False Negative is high. The confusion metric gives us the list of TP,TN, FP and FN for different classes and we can use these values to calculate the Precision, Recall and Accuracy.

Recall = TP/(TP+FN)
Precision = TP(TP+FP)
Accuracy = TP+TN / (TP +TN +FN +FP)

Q4. (b) Report in a table the mean precision, recall, and accuracy of each classifier you evaluated in the previous step for each dataset. (Write-up)

Model Name	**Precision**	**Recall**	**Accuracy**
Voting Classifier	**0.963**	**0.0.962**	**0.962**
Boosting Classifier	**0.931**	**0.925**	**0.925**
Bagging Classifier	**0.962**	**0.962**	**0.962**

(c) Write a discussion about the performance of the different ensemble methods. For example, what classification approaches did better/worse and why do you think so? How did ensemble methods compare to non-ensemble methods? What can you infer by observing the classification performance across the different datasets? Your discussion should consist of two to four paragraphs. (Write-up)

Ans. The performance of all the three ensemble methods : Voting Classifier, Bagging Classifier and Boosting are almost comparable for the given dataset. The ensemble methods performed better than the KNN Classifier and Decision Tree Classifiers. But the non ensemble methods like using a single SVM classifier and Naive Bayes gives us higher accuracy and precision values.

In the case of voting classifiers the model returns the most popular prediction from multiple prediction algorithms in our scenario we have used four models - KNN, decision trees, SVM

and Naive Bayes. This helps us to reduce the variance of predictions and give better model performance. In case of bagging we resample data to train algorithms on different random subsets, this would work better when we have a large dataset, in this case we did not have a large dataset so it may happen that our model is overfitting the data.In this case since the performance of ensembling methods is approximately similar or very close to single models it may be if we are ensembling two highly correlated models, the ensembled model will give almost the same result as your stand-alone models.

```
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

1. Construct Datasets for Training and Evaluation [4 points]

- (a) Load a real dataset of your choice that is designed for classification but was not used in class; e.g., from sklearn.datasets, Kaggle, or your own data. (Code)
- (b) Create a 70/30 train/test split of the dataset. (Code)

```
## a. Loading a real dataset
from sklearn.datasets import load_wine


wine_dataset = load_wine()


print("Keys of the Wine Dataset:\n", wine_dataset.keys(),'\n')
print("Target names: ", wine_dataset["target"][:10],'\n')
print("Feature names: ", wine_dataset["feature_names"],'\n')
print("Shape of data: ", wine_dataset['data'].shape,'\n')
print("Dataset description: \n", wine_dataset['DESCR'],'\n')
```

```
        Flavanoids:                   0.34  5.08    2.03  1.00
        Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
        Proanthocyanins:              0.41  3.58    1.59  0.57
        Colour Intensity:              1.3  13.0     5.1   2.3
        Hue:                          0.48  1.71    0.96  0.23
        OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
        Proline:                       278  1680     746   315
        ============================ ==== ===== ======= =====

        :Missing Attribute Values: None
        :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
        :Creator: R.A. Fisher
        :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
        :Date: July, 1988

    This is a copy of UCI ML Wine recognition datasets.
    https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

    The data is the results of a chemical analysis of wines grown in the same
    region in Italy by three different cultivators. There are thirteen different
    measurements taken for different constituents found in the three types of
    wine.

    Original Owners:

    Forina, M. et al, PARVUS -
```

```
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:


Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

```
# Spliting target variable and independent variables
X = wine_dataset["data"]
y = wine_dataset["target"]
```

```
print(X.shape)
print(y.shape)
```

```
(178, 13)
(178,)
```

```
y=y.reshape(-1,1)
y.shape
```

```
(178, 1)
```

```
# train is now 70% of the entire data set
Xb_train, Xb_test, yb_train, yb_test = train_test_split(X, y, test_size = 0.3, random_state=4
print("Number of samples in training set is: ",len(Xb_train))
print("Number of samples in remaining set is: ",len(Xb_test))
```

```
print( Number of samples in remaining set is:  ,len(Xb_test))
```

```
Number of samples in training set is:  124
Number of samples in remaining set is:  54
```

2. Optimize Hyperparameter(s) for Classification Models [32 points]:

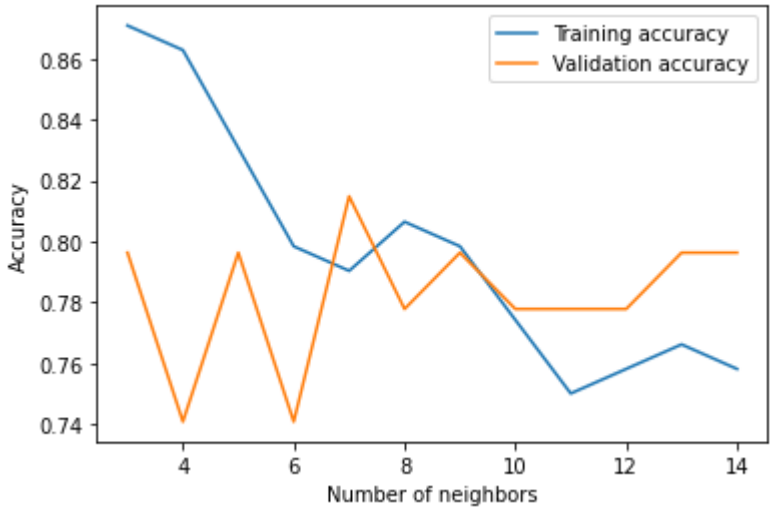
When evaluating each model in this problem, perform stratified 3-fold cross-validation on the training dataset and use the “accuracy” measure to assess performance.

(a) K-Nearest Neighbors (K-NN): find the optimal hyperparameters for the distance metric (i.e., test “Euclidean” and “Manhattan”) and number of nearest neighbors (i.e., test at least 5 different values) when using k-Nearest Neighbors. Report the optimal hyperparameters found and how many hyperparameter combinations you tested in total. (Code and Write-up)

```
# K-Nearest Neighbors: Test number of neighbors from 1 to 10
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold


Kfold = StratifiedKFold(n_splits=3,shuffle=True, random_state=0)

def get_optimal_k_value(p_value):
  training_accuracy = []
  test_accuracy = []
  best_score = 0
  k_values =  range(3,15)
  for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k, p=p_value)
    knn.fit(Xb_train, yb_train)

    # Test model on the training data
    cur_train_accuracy = knn.score(Xb_train, yb_train)
    training_accuracy.append(cur_train_accuracy)

    # Test model on the test data
    cur_test_accuracy = knn.score(Xb_test, yb_test)
    test_accuracy.append(cur_test_accuracy)

    fold_accuracies = cross_val_score(knn, Xb_train, yb_train,cv=Kfold)
    score = fold_accuracies.mean()

    if score > best_score:
      best_param = {'n_neighbors': k}
      best_score = score
```

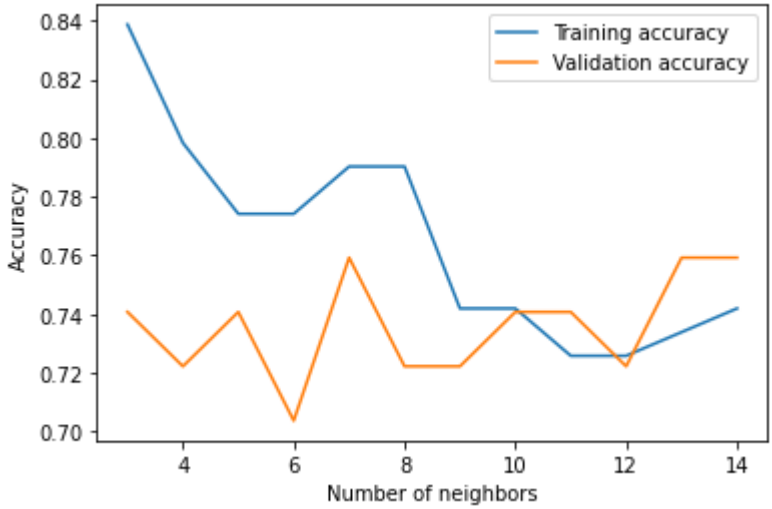
```
    plt.plot(k_values, training_accuracy, label="Training accuracy")
    plt.plot(k_values, test_accuracy, label="Validation accuracy")
    plt.xlabel("Number of neighbors")
    plt.ylabel("Accuracy")
    plt.legend()


    print("Best score on score-validation: {:0.2f}".format(best_score))
    print("Best parameters: {}".format(best_param))


get_optimal_k_value(1) # manhattan
```

```
Best score on score-validation: 0.77
Best parameters: {'n_neighbors': 8}
```




```
get_optimal_k_value(2) # euclidean
```

```
Best score on score-validation: 0.74
Best parameters: {'n_neighbors': 10}
```




```
#patrick:
# According to the question 2,
# you need to perform "stratified 3-fold cross-validation" on the training dataset
```

```
# you need to perform "stratified 3-fold cross validation" on the training dataset
# nd use the "accuracy" measure to assess performance.

# So, you will need to import StratifiedKFold like:
# from sklearn.model_selection import StratifiedKFold
# kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=0)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.h

# And in your code, you will need to set the parameter, "cv", equal to kfold in your "cross_v
# for example: fold_accuracies = cross_val_score(your_model, X_train, y_train, cv=kfold)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.ht

# By the way, there is another parameter you should aware of in "cross_val_score", which is '
# Please refer to https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-param
```

(b) Decision tree: find the optimal hyperparameters for the split criterion (i.e., test “gini” and “entropy”) and tree depth (i.e., test at least 5 different values) when training a decision tree. Report the optimal hyperparameters found and how many hyperparameter combinations you tested in total. (Code and Write-up)

```
# Train a decision tree model
from sklearn.tree import DecisionTreeClassifier


def get_decision_tree_hyperparamter(criterion):
  best_score = 0
  tree_depth_values = range(2,20)
  for d in tree_depth_values:
    decision_tree = DecisionTreeClassifier(criterion=criterion,max_depth=d,random_state=42)
    fold_accuracies = cross_val_score(decision_tree, Xb_train, yb_train, cv=Kfold)
    score = fold_accuracies.mean()
    if score > best_score:
      best_param = {'max_depth': d}
      best_score = score

  print("Best score on score-validation: {:0.2f}".format(best_score))
  print("Best parameters: {}".format(best_param))



get_decision_tree_hyperparamter('gini')
```

```
     Best score on score-validation: 0.94
     Best parameters: {'max_depth': 3}
```

```
get_decision_tree_hyperparamter('entropy')
```

```
     Best score on score-validation: 0.91
     Best parameters: {'max_depth': 4}
```

The decision tree performance was best when criterion is 'gini and max depth is 3

(c) Support Vector Machine (SVM): find the optimal hyperparameters for the poly□nomial degree, kernel bandwidth (i.e., gamma), and regularization parameter (i.e., C) when training a kernel SVM with a polynomial kernel. You must evaluate all possible combinations of at least 4 degree values (for the polynomial degree), at least 4 gamma values, and at least 4 C values. Report the optimal hyperparameters found and how many hyperparameter combinations you tested in total. (Code and Write-up)

```
# Tune hyperparameters for SVM using cross-validation
# One-vs-all/one-vs-rest classification

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC

svm_clf = SVC(kernel="poly", degree = 4, C = 7,gamma = 0.001)
ovr_clf = OneVsRestClassifier(svm_clf)
ovr_clf.fit(Xb_train, yb_train)
predicted_classes = ovr_clf.predict(Xb_test)
print(predicted_classes[:5])

ovr_clf_svm_accuracy = ovr_clf.score(Xb_test, yb_test)
print(ovr_clf_svm_accuracy)
```

```
[0 0 2 1 1]
0.9444444444444444
```

```
C = [6,7,8,9,10]
degree = [2,3,4,5,6]
gamma = [0.1,0.01,0.001,0.0001]

def get_best_params():
  best_score = 0
  for c in C:
    for d in degree:
      for g in gamma:
        svm_clf = SVC(kernel="poly", degree = d, C = c,gamma = g)
        ovr_clf = OneVsRestClassifier(svm_clf)
        fold_accuracies = cross_val_score(ovr_clf, Xb_train, yb_train, cv=Kfold)
        score = fold_accuracies.mean()
        if score > best_score:
          best_param = {'degree': d,'C':c,'gamma':g}
          best_score = score
  print("Best score on score-validation: {:0.2f}".format(best_score))
  print("Best parameters: {}".format(best_param))
```

```
get_best_params()
```

```
Best score on score-validation: 0.97
Best parameters: {'degree': 2, 'C': 6, 'gamma': 0.0001}
```

3. Comparative Analysis of Optimized Classification Models [32 points]: (Do not use GridSearchCV for this problem)

(a) Retrain each of the three models (i.e., Decision tree, K-NN, and SVM) on all the training data using the optimal hyperparameters found in part 2. Also train a Gaussian Naive Bayes model on all the training data. (Code)

Model no. 1 - KNN Classsifier

```
## KNN Classifier
## Hyperparameter for KNN k=8 and p=1 (manhattan)

knn = KNeighborsClassifier(n_neighbors=8, p=1)
knn.fit(Xb_train, yb_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=8, p=1,
                     weights='uniform')
```

Model No.2 Decision Tree Classifier

```
## Decsion Tree Classifier
## Best hyperparameter for Decision Tree Classifier is criterion is "gini" and max_depth = '3

tree_gini = DecisionTreeClassifier(criterion="gini", max_depth=3)
tree_gini.fit(Xb_train, yb_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Model No. 3 SVM Classifier

```
svm_clf = SVC(kernel="poly", degree = 2, C = 6,gamma = 0.001)
ovr_clf = OneVsRestClassifier(svm_clf)
ovr_clf.fit(Xb_train, yb_train)
```

```
OneVsRestClassifier(estimator=SVC(C=6, break_ties=False, cache_size=200,
                                  class_weight=None, coef0=0.0,
                                  decision_function_shape='ovr', degree=2,
                                  gamma=0.001, kernel='poly', max_iter=-1,
                                  probability=False, random_state=None,
                                  shrinking=True, tol=0.001, verbose=False),
                    n_jobs=None)
```

Model No. 4 Gaussian Naive Bayes Model

```
# Train and evaluate Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB

gaussian_model = GaussianNB()
gaussian_model.fit(Xb_train, yb_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

(b) Report the predictive performance on the test dataset for each of the four models from part (a) with respect to each of the following evaluation metrics: accuracy, precision, and recall. (Code and Write-up)

```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_
from sklearn import metrics
```

Model No. 1 KNN Classifier Evaluation

```
knn_predictions = knn.predict(Xb_test) ## Predictions for KNN

print('The accuracy score for KNN classifier is:',accuracy_score(yb_test,knn_predictions),'\r
print('The precision score for KNN classifier is: ',precision_score(yb_test,knn_predictions,a
print('The recall score for KNN classifier is: ', recall_score(yb_test,knn_predictions,averag

print('The classification report is: ')

print('\n',metrics.classification_report(yb_test, knn_predictions))
```

```
The accuracy score for KNN classifier is: 0.7777777777777778

The precision score for KNN classifier is:  0.7792397660818713

The recall score for KNN classifier is:  0.7777777777777778
```

```
The classification report is:

              precision    recall  f1-score   support

           0       0.90      0.95      0.92        19
           1       0.79      0.71      0.75        21
           2       0.60      0.64      0.62        14

    accuracy                           0.78        54
   macro avg       0.76      0.77      0.76        54
weighted avg       0.78      0.78      0.78        54
```

Model No. 2 Decision Tree Classifier Evaluation

```
tree_predictions = tree_gini.predict(Xb_test)

print('The accuracy score for Decision Tree classifier is:',accuracy_score(yb_test,tree_predi
print('The precision score for Decision Tree classifier is: ',precision_score(yb_test,tree_pr
print('The recall score for Decision Tree classifier is: ', recall_score(yb_test,tree_predict

print('The classification report is: ')

print('\n',metrics.classification_report(yb_test, tree_predictions))
```

```
The accuracy score for Decision Tree classifier is: 0.9444444444444444

The precision score for Decision Tree classifier is:  0.9513888888888888

The recall score for Decision Tree classifier is:  0.9444444444444444

The classification report is:

              precision    recall  f1-score   support

           0       1.00      0.89      0.94        19
           1       0.88      1.00      0.93        21
           2       1.00      0.93      0.96        14

    accuracy                           0.94        54
   macro avg       0.96      0.94      0.95        54
weighted avg       0.95      0.94      0.94        54
```

Model No. 3 SVM Classifier Evaluation

```
svm_predictions = ovr_clf.predict(Xb_test)

print('The accuracy score for SVM classifier is:',accuracy_score(yb_test,svm_predictions),'\r
print('The precision score for SVM classifier is: ',precision_score(yb_test,svm_predictions,a
print('The recall score for SVM classifier is: ', recall_score(yb_test,svm_predictions,averag
```

```
print('The classification report is: ')
print('\n',metrics.classification_report(yb_test, svm_predictions))
```

```
     The accuracy score for SVM classifier is: 0.9814814814814815

     The precision score for SVM classifier is:  0.9824074074074074

     The recall score for SVM classifier is:  0.9814814814814815

     The classification report is:

                   precision    recall  f1-score   support

               0       0.95      1.00      0.97        19
               1       1.00      0.95      0.98        21
               2       1.00      1.00      1.00        14

        accuracy                           0.98        54
       macro avg       0.98      0.98      0.98        54
    weighted avg       0.98      0.98      0.98        54
```

Model No. 4 Gaussian Naive Bayes Model

```
gnb_predictions = gaussian_model.predict(Xb_test)

print('The accuracy score for Naive Bayes classifier is:',accuracy_score(yb_test,gnb_predicti
print('The precision score for Naive Bayes classifier is: ',precision_score(yb_test,gnb_predi
print('The recall score for Naive Bayes classifier is: ', recall_score(yb_test,gnb_predictior

print('The classification report is: ')
print('\n',metrics.classification_report(yb_test, gnb_predictions))
```

```
     The accuracy score for Naive Bayes classifier is: 1.0

     The precision score for Naive Bayes classifier is:  1.0

     The recall score for Naive Bayes classifier is:  1.0

     The classification report is:

                   precision    recall  f1-score   support

               0       1.00      1.00      1.00        19
               1       1.00      1.00      1.00        21
               2       1.00      1.00      1.00        14

        accuracy                           1.00        54
       macro avg       1.00      1.00      1.00        54
    weighted avg       1.00      1.00      1.00        54
```

(c) Visualize the predictive performance of the each of the four models from part (a) by showing the resulting confusion matrix for each model. (Code and Write-up)

Model No. 1 KNN Classifier Evaluation

```
from sklearn.metrics import plot_confusion_matrix
```

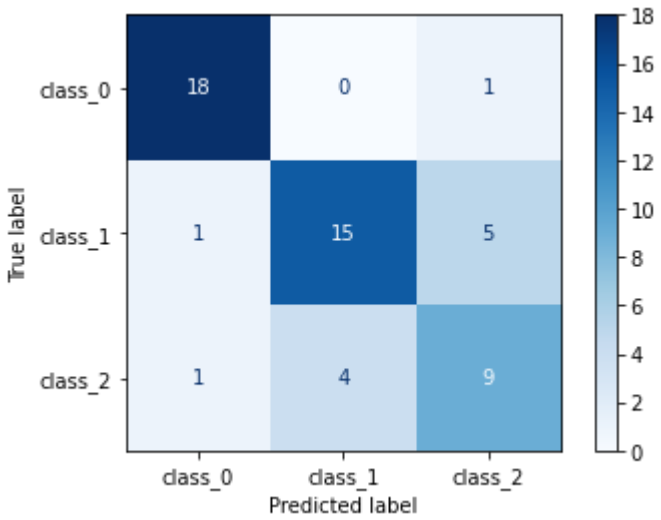
```
print(metrics.confusion_matrix(yb_test,knn_predictions))
```

```
[[18  0  1]
 [ 1 15  5]
 [ 1  4  9]]
```

```
class_names = wine_dataset.target_names
```

```
print("\t Confusion Matrix for KNN Classifier \n")
disp = plot_confusion_matrix(knn, Xb_test, yb_test,display_labels=class_names, cmap=plt.cm.Bl
plt.show()
```

```
        Confusion Matrix for KNN Classifier
```



Model No.2 Decision Tree Classifier

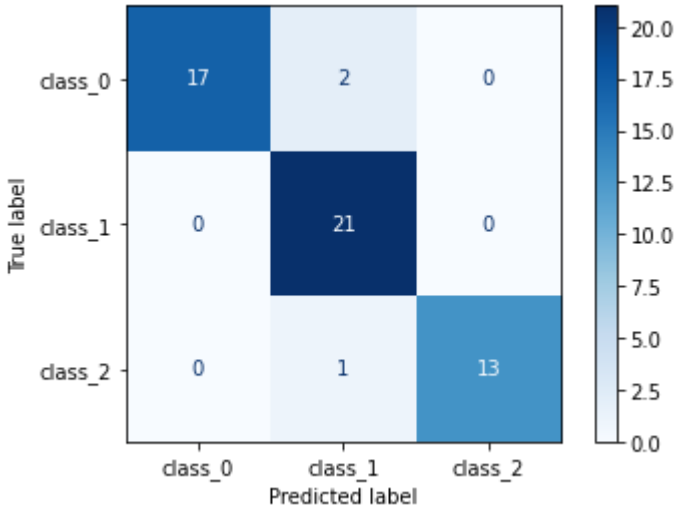
```
print(metrics.confusion_matrix(yb_test,tree_predictions))
```

```
[[17  2  0]
 [ 0 21  0]
 [ 0  1 13]]
```

```
print("\t Confusion Matrix for Decision Tree Classifier \n")
```

```
disp = plot_confusion_matrix(tree_gini, Xb_test, yb_test,display_labels=class_names, cmap=plt
plt.show()
```

```
Confusion Matrix for Decision Tree Classifier
```



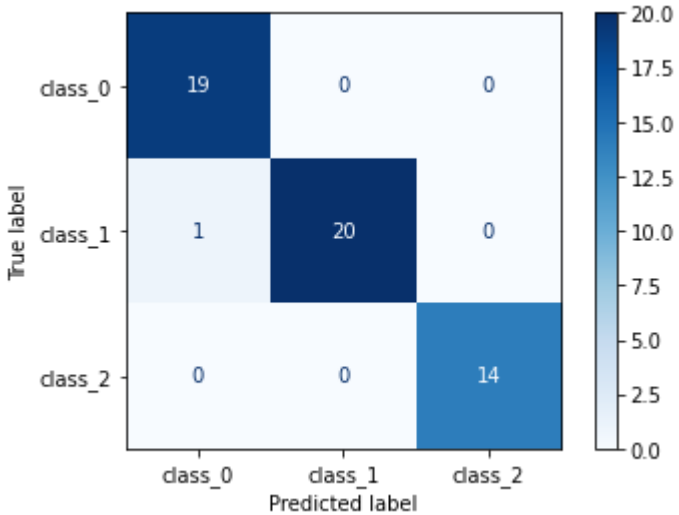
Model No. 3 SVM Classifier

```
print(metrics.confusion_matrix(yb_test,svm_predictions))
```

```
[[19  0  0]
 [ 1 20  0]
 [ 0  0 14]]
```

```
print("\t Confusion Matrix for SVM Classifier \n")
disp = plot_confusion_matrix(ovr_clf, Xb_test, yb_test,display_labels=class_names, cmap=plt.c
plt.show()
```

```
Confusion Matrix for SVM Classifier
```

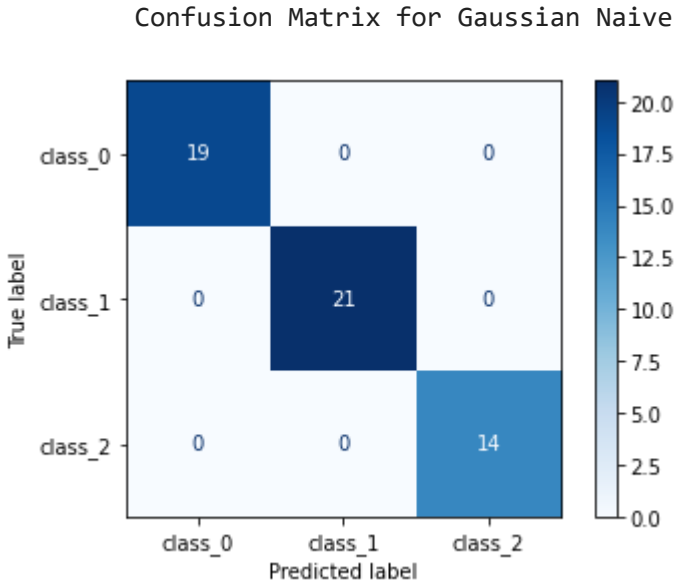


Model No. 4 Gaussian Naive Bayes Model

```
print(metrics.confusion_matrix(yb_test,gnb_predictions))
```

```
[[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
```

```
print("\t Confusion Matrix for Gaussian Naive Bayes Classifier \n")
disp = plot_confusion_matrix(gaussian_model, Xb_test, yb_test,display_labels=class_names, cma
plt.show()
```



4. Ensemble Learning [32 points]: Analyze the effects of using different types of ensembles for the classification task.

a) Evaluate each of the following classifiers using 3-fold cross validation(Code)

- Majority vote classifier that uses the four classifiers from part 3 of this assignment
- Bagging method
- Boosting method

```
# Ensemble learning

# 1. Majority vote ensemble learner

from sklearn.ensemble import VotingClassifier

voting_classifier = VotingClassifier(estimators=[('gnb', gaussian_model), ('dt', tree_gini),(
voting_classifier.fit(Xb_train, yb_train)
```

```
VotingClassifier(estimators=[('gnb',
                              GaussianNB(priors=None, var_smoothing=1e-09)),
                             ('dt',
                              DecisionTreeClassifier(ccp_alpha=0.0,
                                                     class_weight=None,
                                                     criterion='gini',
                                                     max_depth=3,
                                                     max_features=None,
                                                     max_leaf_nodes=None,
                                                     min_impurity_decrease=0.0,
                                                     min_impurity_split=None,
                                                     min_samples_leaf=1,
                                                     min_samples_split=2,
                                                     min_weight_fraction_leaf=0.0,
                                                     presort='deprecated',
                                                     ra...
                                                               decision_function_shape
                                                               degree=2,
                                                               gamma=0.001,
                                                               kernel='poly',
                                                               max_iter=-1,
                                                               probability=False,
                                                               random_state=None,
                                                               shrinking=True,
                                                               tol=0.001,
                                                               verbose=False),
                                                     n_jobs=None)),
                             ('knn',
                              KNeighborsClassifier(algorithm='auto',
                                                   leaf_size=30,
                                                   metric='minkowski',
                                                   metric_params=None,
                                                   n_jobs=None, n_neighbors=8,
                                                   p=1, weights='uniform'))],
                 flatten_transform=True, n_jobs=None, voting='hard',
                 weights=None)
```

```
fold_accuracies = cross_val_score(voting_classifier, Xb_train, yb_train,cv=Kfold)
voting_score = fold_accuracies.mean()
```

```
print(voting_score)
```

```
0.9355400696864112
```

Bagging Method

```
#2. Bagging ensemble
```

```
from sklearn.ensemble import BaggingClassifier
```

```
bagging = BaggingClassifier(ovr_clf)
```

```
bagging = BaggingClassifier(OVR_SVC)
bagging.fit(Xb_train, yb_train)
```

```
BaggingClassifier(base_estimator=OneVsRestClassifier(estimator=SVC(C=7,
                                                                   break_ties=False,
                                                                   cache_size=200,
                                                                   class_weight=None,
                                                                   coef0=0.0,
                                                                   decision_function_sh
                                                                   degree=4,
                                                                   gamma=0.001,
                                                                   kernel='poly',
                                                                   max_iter=-1,
                                                                   probability=False,
                                                                   random_state=None,
                                                                   shrinking=True,
                                                                   tol=0.001,
                                                                   verbose=False),
                                                     n_jobs=None),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=10, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

```
fold_accuracies = cross_val_score(bagging, Xb_train, yb_train,cv=Kfold)
bagging_score = fold_accuracies.mean()
```

```
print(bagging_score)
```

```
0.9353464963221061
```

Boosting Method

```
#3. Adaboost algorithm
from sklearn.ensemble import AdaBoostClassifier

adabooster = AdaBoostClassifier(n_estimators=50)
adabooster.fit(Xb_train, yb_train)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
```

```
fold_accuracies = cross_val_score(adabooster, Xb_train, yb_train,cv=Kfold)
boosting_score = fold_accuracies.mean()
```

```
print(boosting_score)
```

```
0.9111498257839722
```

b) Report in a table the mean precision, recall, and accuracy of each classifier you evaluated in the previous step for each dataset. (Write-up)

```
voting_predictions = voting_classifier.predict(Xb_test)


print('The accuracy score for voting classifier is:',accuracy_score(yb_test,voting_predictior
print('The precision score for voting classifier is: ',precision_score(yb_test,voting_predict
print('The recall score for voting classifier is: ', recall_score(yb_test,voting_predictions,

print('The classification report is: ')
print('\n',metrics.classification_report(yb_test, voting_predictions))
```

```
     The accuracy score for voting classifier is: 0.9629629629629629

     The precision score for voting classifier is:  0.9638888888888888

     The recall score for voting classifier is:  0.9629629629629629

     The classification report is:

                   precision    recall  f1-score   support

                0       0.95      1.00      0.97        19
                1       0.95      0.95      0.95        21
                2       1.00      0.93      0.96        14

         accuracy                           0.96        54
        macro avg       0.97      0.96      0.96        54
     weighted avg       0.96      0.96      0.96        54
```

```
bagging_predictions = bagging.predict(Xb_test)


print('The accuracy score for bagging classifier is:',accuracy_score(yb_test,bagging_predicti
print('The precision score for bagging classifier is: ',precision_score(yb_test,bagging_predi
print('The recall score for bagging classifier is: ', recall_score(yb_test,bagging_predictior

print('The classification report is: ')
print('\n',metrics.classification_report(yb_test, bagging_predictions))
```

```
     The accuracy score for bagging classifier is: 0.9629629629629629

     The precision score for bagging classifier is:  0.9629629629629629

     The recall score for bagging classifier is:  0.9629629629629629

     The classification report is:

                   precision    recall  f1-score   support
```

```
                0       0.95      0.95      0.95        19
                1       0.95      0.95      0.95        21
                2       1.00      1.00      1.00        14

         accuracy                           0.96        54
        macro avg       0.97      0.97      0.97        54
     weighted avg       0.96      0.96      0.96        54
```

```
boosting_predictions = adabooster.predict(Xb_test)

print('The accuracy score for boosting classifier is:',accuracy_score(yb_test,boosting_predic
print('The precision score for boosting classifier is: ',precision_score(yb_test,boosting_pre
print('The recall score for boosting classifier is: ', recall_score(yb_test,boosting_predicti

print('The classification report is: ')
print('\n',metrics.classification_report(yb_test, boosting_predictions))
```

```
     The accuracy score for boosting classifier is: 0.9259259259259259

     The precision score for boosting classifier is:  0.931682769726248

     The recall score for boosting classifier is:  0.9259259259259259

     The classification report is:

                   precision    recall  f1-score   support

                0       0.95      1.00      0.97        19
                1       0.87      0.95      0.91        21
                2       1.00      0.79      0.88        14

         accuracy                           0.93        54
        macro avg       0.94      0.91      0.92        54
     weighted avg       0.93      0.93      0.92        54
```

```
## End
```

✓ 0s completed at 2:08 PM