Lab Assignment 2

1.(a)

```
from sklearn.datasets import load_wine
wine = load_wine()
x=wine.data
y=wine.target
```

(b)

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.25, random_state=2)
```

2.(a) decision tree

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train_scaled = ss.fit_transform(x_train)
x_test_scaled = ss.transform(x_test)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

best_score=0
for criterion in ['gini','entropy' ]:
    for max_depth in [2,3,4,5,6,7,8,9,10]:
        tree_giniIndex = DecisionTreeClassifier(criterion = criterion,
max_depth=max_depth)
        kfold = KFold(n_splits=10, shuffle = True, random_state=2)
        fold_accuracies = cross_val_score(tree_giniIndex, x_train_scaled,
y_train,cv=kfold)
        score = fold_accuracies.mean()
        if score > best_score:
            best_paramC = criterion
            best_paramD = max_depth
```

```
                best_score = score
print("Best score on cross-validation: {:2f}".format(best_score))
print("Best parametersC: {}".format(best_paramC))
print("Best parametersD: {}".format(best_paramD))
```

In [10]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

best_score=0
for criterion in ['gini','entropy' ]:
    for max_depth in [2,3,4,5,6,7,8,9,10]:
        tree_giniIndex = DecisionTreeClassifier(criterion = criterion, max_depth=max_depth)
        kfold = KFold(n_splits=10, shuffle = True, random_state=2)
        fold_accuracies = cross_val_score(tree_giniIndex, x_train_scaled, y_train,cv=kfold)
        score = fold_accuracies.mean()
        if score > best_score:
            best_paramC = criterion
            best_paramD = max_depth
            best_score = score

print("Best score on cross-validation: {:2f}".format(best_score))
print("Best parametersC: {}".format(best_paramC))
print("Best parametersD: {}".format(best_paramD))
```

```
Best score on cross-validation: 0.909890
Best parametersC: gini
Best parametersD: 3
```

Decision tree: optimal hyperparameters: criterion = ‘gini’, max_depth = 3;

The number of tested hyperparameter combinations: 18;

(b)KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

best_score=0
for curPvalue in [1,2]:
    neighbor_settings = range(1,11)
    for curKvalue in neighbor_settings:
        clf = KNeighborsClassifier(n_neighbors = curKvalue, p=curPvalue,
metric='minkowski')
        kfold = KFold(n_splits=10, shuffle = True, random_state=2)
        fold_accuracies = cross_val_score(clf, x_train_scaled, y_train,cv=kfold)
        score = fold_accuracies.mean()
```

```
            if score > best_score:
                best_paramP = curPvalue
                best_paramK = curKvalue
                best_score = score

print("Best score on cross-validation: {:2f}".format(best_score))
print("Best parametersP: {}".format(best_paramP))
print("Best parametersK: {}".format(best_paramK))
```

In [14]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

best_score=0
for curPvalue in [1,2]:
    neighbor_settings = range(1,11)
    for curKvalue in neighbor_settings:
        clf = KNeighborsClassifier(n_neighbors = curKvalue, p=curPvalue, metric='minkowski')
        kfold = KFold(n_splits=10, shuffle = True, random_state=2)
        fold_accuracies = cross_val_score(clf, x_train_scaled, y_train,cv=kfold)
        score = fold_accuracies.mean()
        if score > best_score:
            best_paramP = curPvalue
            best_paramK = curKvalue
            best_score = score

print("Best score on cross-validation: {:2f}".format(best_score))
print("Best parametersP: {}".format(best_paramP))
print("Best parametersK: {}".format(best_paramK))
```

```
Best score on cross-validation: 0.977473
Best parametersP: 1
Best parametersK: 1
```

KNN: optimal hyperparameters: p = 1 (Manhattan distance), K=1;

The number of tested hyperparameter combinations: 20;

(c)SVM

```
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

best_score=0
for degree in [2,3,4]:
    for curC in [1, 2,3,4,5,8,10]:
        for gamma in [1/13, 2/13, 3/13, 4/13]:
            svm = SVC(kernel = "poly", degree = degree, C=curC, gamma =
gamma)
```

```
                kfold = KFold(n_splits=10, shuffle = True, random_state=2)
                fold_accuracies = cross_val_score(svm, x_train_scaled,
y_train,cv=kfold)
                score = fold_accuracies.mean()
                if score > best_score:
                    best_paramD = degree
                    best_paramC = curC
                    best_paramG = gamma
                    best_score = score

print("Best score on cross-validation: {:2f}".format(best_score))
print("Best parametersD: {}".format(best_paramD))
print("Best parametersC: {}".format(best_paramC))
print("Best parametersG: {}".format(best_paramG))
```

In [16]:
```
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

best_score=0
for degree in [2,3,4]:
    for curC in [1, 2,3,4,5,8,10]:
        for gamma in [1/13, 2/13, 3/13, 4/13]:
            svm = SVC(kernel = "poly", degree = degree, C=curC, gamma = gamma)
            kfold = KFold(n_splits=10, shuffle = True, random_state=2)
            fold_accuracies = cross_val_score(svm, x_train_scaled, y_train,cv=kfold)
            score = fold_accuracies.mean()
            if score > best_score:
                best_paramD = degree
                best_paramC = curC
                best_paramG = gamma
                best_score = score

print("Best score on cross-validation: {:2f}".format(best_score))
print("Best parametersD: {}".format(best_paramD))
print("Best parametersC: {}".format(best_paramC))
print("Best parametersG: {}".format(best_paramG))
```

```
Best score on cross-validation: 0.947253
Best parametersD: 3
Best parametersC: 1
Best parametersG: 0.15384615384615385
```

SVM: optimal hyperparameters: degree = 3, C=1, gamma = 2/13;

The number of tested hyperparameter combinations: 84;

3.(a)

```
#decision tree
tree_giniIndex = DecisionTreeClassifier(criterion='gini', max_depth=3)
tree_giniIndex.fit(x_train_scaled, y_train)
```

test_score = tree_giniIndex.score(x_test_scaled, y_test)

print("Test set score: {:2f}".format(test_score))

In [12]:
```
tree_giniIndex = DecisionTreeClassifier(criterion='gini', max_depth=3)
tree_giniIndex.fit(x_train_scaled, y_train)
test_score = tree_giniIndex.score(x_test_scaled, y_test)
print("Test set score: {:2f}".format(test_score))
```

```
Test set score: 0.955556
```

#KNN

clf = KNeighborsClassifier(p=1, n_neighbors=1)

clf.fit(x_train_scaled, y_train)

test_score = clf.score(x_test_scaled, y_test)

print("Test set score: {:2f}".format(test_score))

In [18]:
```
clf = KNeighborsClassifier(p=1, n_neighbors=1)
clf.fit(x_train_scaled, y_train)
test_score = clf.score(x_test_scaled, y_test)
print("Test set score: {:2f}".format(test_score))
```

```
Test set score: 0.977778
```

#SVM

svm = SVC(degree= 3, C=1, gamma=2/13)

svm.fit(x_train_scaled, y_train)

test_score = svm.score(x_test_scaled, y_test)

print("Test set score: {:2f}".format(test_score))

In [19]:
```
svm = SVC(degree= 3, C=1, gamma=2/13)
svm.fit(x_train_scaled, y_train)
test_score = svm.score(x_test_scaled, y_test)
print("Test set score: {:2f}".format(test_score))
```

```
Test set score: 0.955556
```

Gaussian Naive Bayes

from sklearn.naive_bayes import GaussianNB

gaussian_model = GaussianNB()

gaussian_model.fit(x_train_scaled, y_train)

In [57]:
```
from sklearn.naive_bayes import GaussianNB

gaussian_model = GaussianNB()
gaussian_model.fit(x_train_scaled, y_train)
```

Out[57]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

(b)

```
# decision tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

y_predicted_tree = tree_giniIndex.predict(x_test_scaled)
print(accuracy_score(y_predicted_tree, y_test))
print(precision_score(y_predicted_tree, y_test, average='macro'))
print(recall_score(y_predicted_tree, y_test, average='macro'))
```

In [33]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

y_predicted_tree = tree_giniIndex.predict(x_test_scaled)
print(accuracy_score(y_predicted_tree, y_test))
print(precision_score(y_predicted_tree, y_test, average='macro'))
print(recall_score(y_predicted_tree, y_test, average='macro'))
```

```
0.9555555555555556
0.9649122807017544
0.9555555555555556
```

```
#KNN
y_predicted_knn = clf.predict(x_test_scaled)
print(accuracy_score(y_predicted_knn, y_test))
print(precision_score(y_predicted_knn, y_test, average='macro'))
print(recall_score(y_predicted_knn, y_test, average='macro'))
```

In [35]:
```
y_predicted_knn = clf.predict(x_test_scaled)
print(accuracy_score(y_predicted_knn, y_test))
print(precision_score(y_predicted_knn, y_test, average='macro'))
print(recall_score(y_predicted_knn, y_test, average='macro'))
```

```
0.9777777777777777
0.9743589743589745
0.9833333333333334
```

#SVM

y_predicted_svm = svm.predict(x_test_scaled)

print(accuracy_score(y_predicted_svm, y_test))

print(precision_score(y_predicted_svm, y_test, average='macro'))

print(recall_score(y_predicted_svm, y_test, average='macro'))

In [38]:
```
y_predicted_svm = svm.predict(x_test_scaled)
print(accuracy_score(y_predicted_svm, y_test))
print(precision_score(y_predicted_svm, y_test, average='macro'))
print(recall_score(y_predicted_svm, y_test, average='macro'))
```

```
0.9555555555555556
0.9568151147098515
0.9555555555555556
```

Gaussian Naive Bayes

y_predictedNB = gaussian_model.predict(x_test_scaled)

print(accuracy_score(y_predictedNB, y_test))

print(precision_score(y_predictedNB, y_test, average='macro'))

print(recall_score(y_predictedNB, y_test, average='macro'))

In [40]:
```
y_predictedNB = gaussian_model.predict(x_test_scaled)
print(accuracy_score(y_predictedNB, y_test))
print(precision_score(y_predictedNB, y_test, average='macro'))
print(recall_score(y_predictedNB, y_test, average='macro'))
```

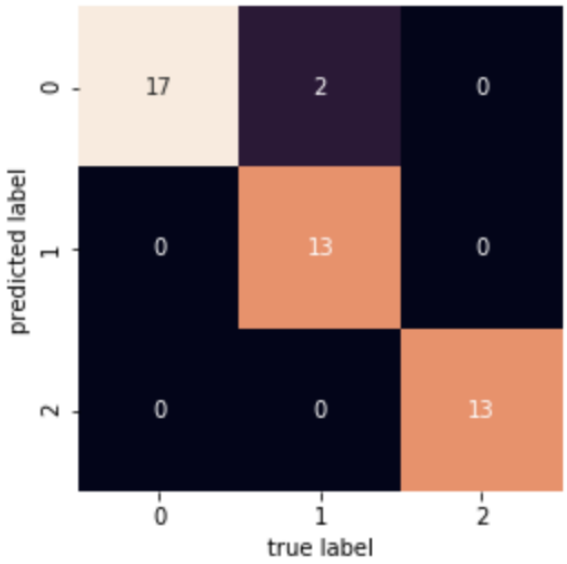
```
0.9777777777777777
0.9824561403508771
0.9761904761904763
```

(c)

#decision tree

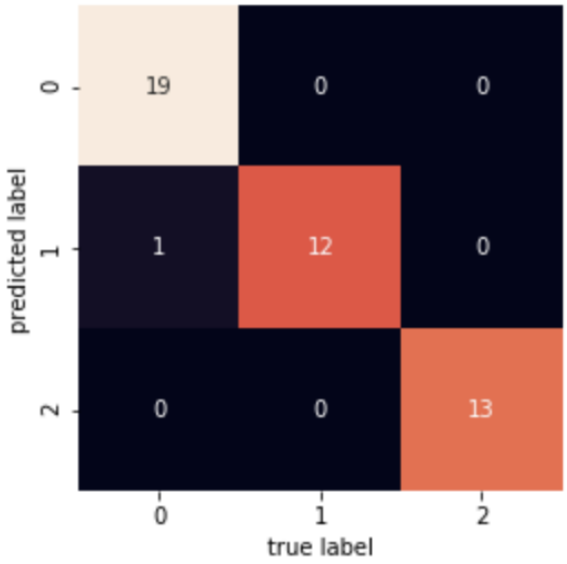
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

mat = confusion_matrix(y_predicted_tree, y_test)
sns.heatmap(mat.T, square=True, annot = True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[26]: Text(91.68, 0.5, 'predicted label')




```
#KNN
mat = confusion_matrix(y_predicted_knn, y_test)
sns.heatmap(mat.T, square=True, annot = True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

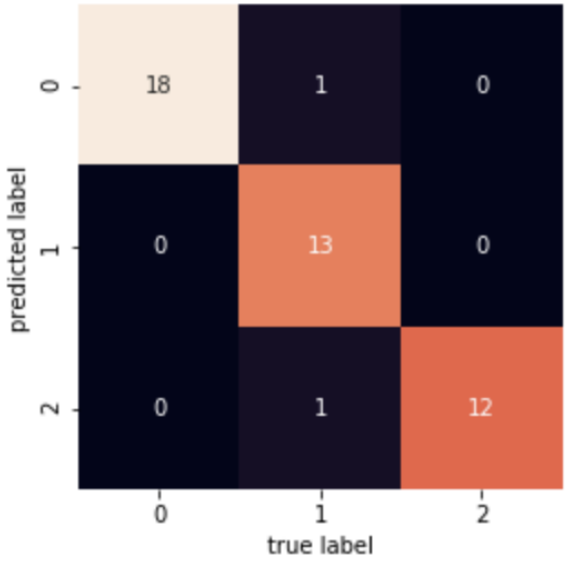
Out[24]: Text(91.68, 0.5, 'predicted label')



#SVM

```
mat = confusion_matrix(y_predicted_svm, y_test)
sns.heatmap(mat.T, square=True, annot = True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[28]: Text(91.68, 0.5, 'predicted label')

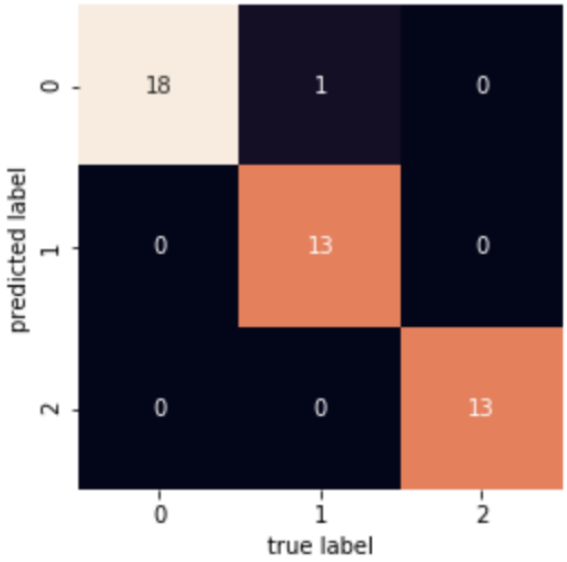


#Gaussian Naïve Bayes

```
mat = confusion_matrix(y_predictedNB, y_test)
sns.heatmap(mat.T, square=True, annot = True, fmt='d', cbar=False)
```

```
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[42]: Text(91.68, 0.5, 'predicted label')



(d)

	Test_set_score	accuracy	precision	recall
Decision tree	0.956	0.956	0.965	0.956
KNN	0.978	0.978	0.974	0.983
SVM	0.956	0.956	0.957	0.956
Gaussian Naïve Bayes	0.978	0.978	0.982	0.976

According to the table, we can find the KNN method performs the best and SVM performs the worst. I think there are two reasons. One reason is this dataset is not appropriate to the SVM method, but it is appropriate to the KNN method. The other reason is the number of tested hyperparameters is not enough and appropriate. Thus, the optimal hyperparameters are not found.

According to performance metrics, we can compute the precision, recall and accuracy. The value of precision can tell us what proportion of positive identifications are actually correct. The value of recall can tell us what proportion of actual positives are identified correctly. The value of accuracy can tell us what proportion of predictions are right. Thus, according to these values, we can compare and evaluate different models.