In [1]:
```python
# Construct datasets for training and evaluation
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

wine = load_wine()
X = wine.data
y = wine.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
In [2]: print(wine.DESCR)
```

```
Wine Data Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the clas
s
    :Attribute Information:
                - 1) Alcohol
                - 2) Malic acid
                - 3) Ash
                - 4) Alcalinity of ash
                - 5) Magnesium
                - 6) Total phenols
                - 7) Flavanoids
                - 8) Nonflavanoid phenols
                - 9) Proanthocyanins
                - 10)Color intensity
                - 11)Hue
                - 12)OD280/OD315 of diluted wines
                - 13)Proline
                - class:
                - class_0
                - class_1
                - class_2

    :Summary Statistics:

    ============================= ==== ===== ======= =====
                                   Min   Max   Mean     SD
    ============================= ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    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
 (https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.dat
a)

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 differ
ent
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 Correlatio
n.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

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

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).
```

In [3]:
```
# Decision tree with gini split criterion
# Tune hyperparameter using cross-validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

best_score = 0
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
for maxDepth in [1, 2, 3, 4, 5, 6]:
    tree_gini = DecisionTreeClassifier(criterion='gini', random_state=42, ma
    fold_accuracies = cross_val_score(tree_gini, X_train, y_train, cv=kfold,
    score = fold_accuracies.mean()
    print("Max depth:",maxDepth, "Score:", score)
    if score > best_score:
        best_param = {'max_depth': maxDepth}
        best_score = score
tree_gini = DecisionTreeClassifier(criterion='gini', random_state=42, **best
tree_gini.fit(X_train, y_train)
test_score = tree_gini.score(X_test, y_test)
print("Best score on cross-validation: {:0.2f}".format(best_score))
print("Best parameters: {}".format(best_param))
print("Test set score: {:.2f}".format(test_score))
```

```
Max depth: 1 Score: 0.5895604395604395
Max depth: 2 Score: 0.9021978021978022
Max depth: 3 Score: 0.9324175824175824
Max depth: 4 Score: 0.9401098901098901
Max depth: 5 Score: 0.9247252747252748
Max depth: 6 Score: 0.9247252747252748
Best score on cross-validation: 0.94
Best parameters: {'max_depth': 4}
Test set score: 0.96
```

In [4]:
```python
# Decision tree with entropy split criterion
# Tune hyperparameter using cross-validation
best_score = 0
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
for maxDepth in [1, 2, 3, 4, 5, 6]:
    tree_entropy = DecisionTreeClassifier(criterion='entropy', random_state=
    fold_accuracies = cross_val_score(tree_entropy, X_train, y_train, cv=kfo
    score = fold_accuracies.mean()
    print("Max depth:",maxDepth, "Score:", score)
    if score > best_score:
        best_param = {'max_depth': maxDepth}
        best_score = score
tree_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42,
tree_entropy.fit(X_train, y_train)
test_score = tree_entropy.score(X_test, y_test)
print("Best score on cross-validation: {:0.2f}".format(best_score))
print("Best parameters: {}".format(best_param))
print("Test set score: {:.2f}".format(test_score))
```

```
Max depth:  1 Score:  0.5368131868131868
Max depth:  2 Score:  0.873076923076923
Max depth:  3 Score:  0.9247252747252748
Max depth:  4 Score:  0.9175824175824177
Max depth:  5 Score:  0.9175824175824177
Max depth:  6 Score:  0.9175824175824177
Best score on cross-validation: 0.92
Best parameters: {'max_depth': 3}
Test set score: 0.89
```

2. (a)

The optimal hyperparameters are gini split criterion and 4 tree depth. I have tested gini and entropy split criterions, each with 6 different depth ranging from 1 to 6.

In [5]:
```python
# Normalize data
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

```
In [6]: # K-Nearest Neighbors with Euclidean distance metric
        # Tune hyperparameter using cross-validation
        from sklearn.neighbors import KNeighborsClassifier

        best_score = 0
        kfold = KFold(n_splits=10, shuffle=True, random_state=42)
        for curKvalue in [1, 2, 3, 4, 5, 6, 7, 8]:
            knn = KNeighborsClassifier(p=2, metric='euclidean', n_neighbors=curKvalu
            fold_accuracies = cross_val_score(knn, X_train_norm, y_train, cv=kfold,
            score = fold_accuracies.mean()
            print("K_numbers:",curKvalue, "Score:", score)
            if score > best_score:
                best_param = {'n_neighbors': curKvalue}
                best_score = score

        knn = KNeighborsClassifier(p=2, metric='euclidean', **best_param)
        knn.fit(X_train_norm, y_train)
        test_score = knn.score(X_test_norm, y_test)
        print("Best score on cross-validation: {:0.2f}".format(best_score))
        print("Best parameters: {}".format(best_param))
        print("Test set score: {:.2f}".format(test_score))
```

```
K_numbers: 1 Score: 0.9483516483516483
K_numbers: 2 Score: 0.9478021978021978
K_numbers: 3 Score: 0.9703296703296704
K_numbers: 4 Score: 0.9483516483516483
K_numbers: 5 Score: 0.9626373626373625
K_numbers: 6 Score: 0.9626373626373625
K_numbers: 7 Score: 0.9697802197802197
K_numbers: 8 Score: 0.9697802197802197
Best score on cross-validation: 0.97
Best parameters: {'n_neighbors': 3}
Test set score: 0.96
```

```
In [7]: # K-Nearest Neighbors with Manhattan distance metric
        # Tune hyperparameter using cross-validation

        best_score = 0
        kfold = KFold(n_splits=10, shuffle=True, random_state=42)
        for curKvalue in [1, 2, 3, 4, 5, 6, 7, 8]:
            knn = KNeighborsClassifier(p=1, metric='manhattan', n_neighbors=curKvalu
            fold_accuracies = cross_val_score(knn, X_train_norm, y_train, cv=kfold,
            score = fold_accuracies.mean()
            print("K_numbers:",curKvalue, "Score:", score)
            if score > best_score:
                best_param = {'n_neighbors': curKvalue}
                best_score = score

        knn = KNeighborsClassifier(p=1, metric='manhattan', **best_param)
        knn.fit(X_train_norm, y_train)
        test_score = knn.score(X_test_norm, y_test)
        print("Best score on cross-validation: {:0.2f}".format(best_score))
        print("Best parameters: {}".format(best_param))
        print("Test set score: {:.2f}".format(test_score))
```

```
K_numbers: 1 Score: 0.9780219780219781
K_numbers: 2 Score: 0.9478021978021978
K_numbers: 3 Score: 0.9703296703296704
K_numbers: 4 Score: 0.9708791208791208
K_numbers: 5 Score: 0.9703296703296704
K_numbers: 6 Score: 0.9626373626373625
K_numbers: 7 Score: 0.9626373626373625
K_numbers: 8 Score: 0.9549450549450549
Best score on cross-validation: 0.98
Best parameters: {'n_neighbors': 1}
Test set score: 0.93
```

2. (b)

The optimal hyperparameters are Manhattan distance metric and 1 nearest neighbor. I have tested Euclidean and Manhattan distance metric, each with different nearest neighbors ranging from 1 to 8.

In [8]:
```python
# Support Vector Machine
# Tune hyperparameter using cross-validation
from sklearn.svm import SVC

best_score = 0
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
for curC in [0.001, 0.01, 0.1, 1, 10, 100]:
    for curDegree in [1, 2, 3, 4, 5]:
        for curGamma in [0.2, 0.4, 0.6, 0.8]:
            svm = SVC(kernel='poly', C=curC, degree=curDegree, gamma=curGamm
            fold_accuracies = cross_val_score(svm, X_train_norm, y_train, cv
            score = fold_accuracies.mean()
            if score > best_score:
                best_C = {'C': curC}
                best_degree = {'degree': curDegree}
                best_gamma = {'gamma': curGamma}
                best_score = score

svm = SVC(kernel='poly', **best_C, **best_degree, **best_gamma)
svm.fit(X_train_norm, y_train)
test_score = svm.score(X_test_norm, y_test)
print("Best score on cross-validation: {:0.2f}".format(best_score))
print("Best parameters: {}".format(best_C))
print("Best parameters: {}".format(best_degree))
print("Best parameters: {}".format(best_gamma))
print("Test set score: {:.2f}".format(test_score))
```

```
Best score on cross-validation: 0.99
Best parameters: {'C': 1}
Best parameters: {'degree': 2}
Best parameters: {'gamma': 0.8}
Test set score: 1.00
```

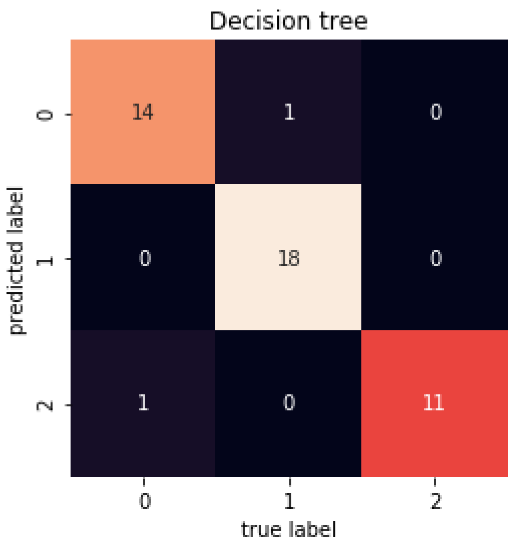
2. (c)

The optimal hyperparameters are C=1, polynomial degree=2, and gamma=0.8. I have tested all possible combinations of 5 degree values, 6 C values, and 4 gamma values. Degree values are 1, 2, 3, 4, 5. C values are 0.001, 0.01, 0.1, 1, 10, 100. Gamma values are 0.2, 0.4, 0.6, 0.8.

```
In [9]: # Comparative analysis of optimized classification model
        # Train and evaluate decision tree
        from sklearn import metrics
        from sklearn.metrics import accuracy_score

        decision_tree = DecisionTreeClassifier(criterion='gini', max_depth=4, random
        decision_tree.fit(X_train, y_train)

        y_predicted = decision_tree.predict(X_test)
        print(metrics.classification_report(y_predicted, y_test))
        print("Accuracy:", accuracy_score(y_predicted, y_test))
```

```
             precision    recall  f1-score   support

          0       0.93      0.93      0.93        15
          1       1.00      0.95      0.97        19
          2       0.92      1.00      0.96        11

avg / total       0.96      0.96      0.96        45

Accuracy: 0.9555555555555556
```

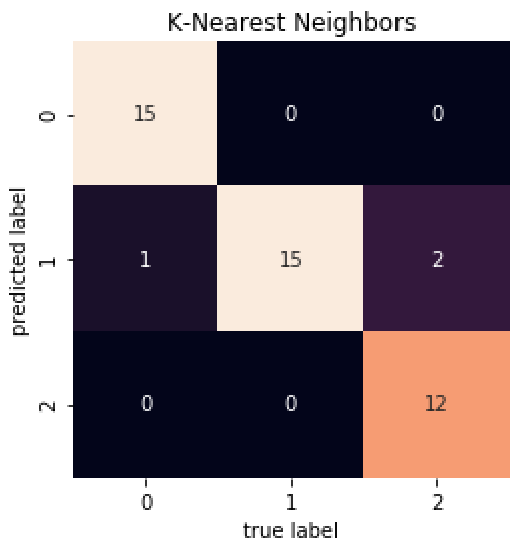
```
In [10]: # Visualize confusion matrix
         from sklearn.metrics import confusion_matrix
         import seaborn as sns
         import matplotlib.pyplot as plt
         %matplotlib inline

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



In [11]:
```
# Train and evaluate KNN

knn = KNeighborsClassifier(p=1, metric='manhattan')
knn.fit(X_train_norm, y_train)

y_predicted = knn.predict(X_test_norm)
print(metrics.classification_report(y_predicted, y_test))
print("Accuracy:", accuracy_score(y_predicted, y_test))
```

```
             precision    recall  f1-score   support

          0       1.00      0.94      0.97        16
          1       0.83      1.00      0.91        15
          2       1.00      0.86      0.92        14

avg / total       0.94      0.93      0.93        45

Accuracy: 0.9333333333333333
```

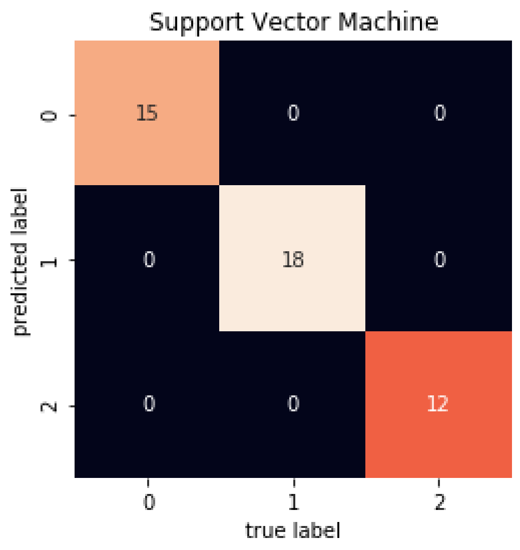
In [12]:
```
# Visualize confusion matrix

mat = confusion_matrix(y_predicted, y_test)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title('K-Nearest Neighbors')
plt.show()
```



In [13]:
```python
# Train and evaluate SVM

svm = SVC(kernel='poly', C=1, degree=2, gamma=0.8)
svm.fit(X_train_norm, y_train)

y_predicted = svm.predict(X_test_norm)
print(metrics.classification_report(y_predicted, y_test))
print("Accuracy:", accuracy_score(y_predicted, y_test))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        15
          1       1.00      1.00      1.00        18
          2       1.00      1.00      1.00        12

avg / total       1.00      1.00      1.00        45

Accuracy: 1.0
```

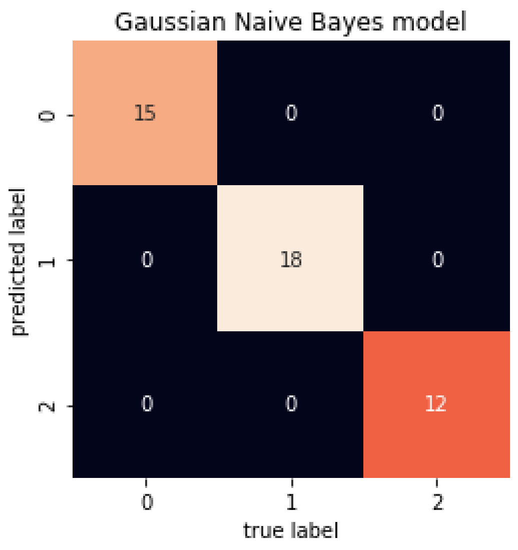
In [14]:
```python
# Visualize confusion matrix

mat = confusion_matrix(y_predicted, y_test)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title('Support Vector Machine')
plt.show()
```



```
In [15]: # Train and evaluate Gaussian Naive Bayes model
         from sklearn.naive_bayes import GaussianNB

         gaussian_model = GaussianNB()
         gaussian_model.fit(X_train, y_train)
         y_predictedNB = gaussian_model.predict(X_test)
         print(metrics.classification_report(y_predictedNB, y_test))
         print("Accuracy:", accuracy_score(y_predicted, y_test))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        15
          1       1.00      1.00      1.00        18
          2       1.00      1.00      1.00        12

avg / total       1.00      1.00      1.00        45

Accuracy: 1.0
```

```
In [16]: # Visualize confusion matrix

         mat = confusion_matrix(y_predicted, y_test)
         sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
         plt.xlabel('true label')
         plt.ylabel('predicted label')
         plt.title('Gaussian Naive Bayes model')
         plt.show()
```



3. (b)

The predictive performance on the test dataset for each of the four models are as follows:

Decision tree: precision: 0.96; recall: 0.96; accuracy: 0.96.

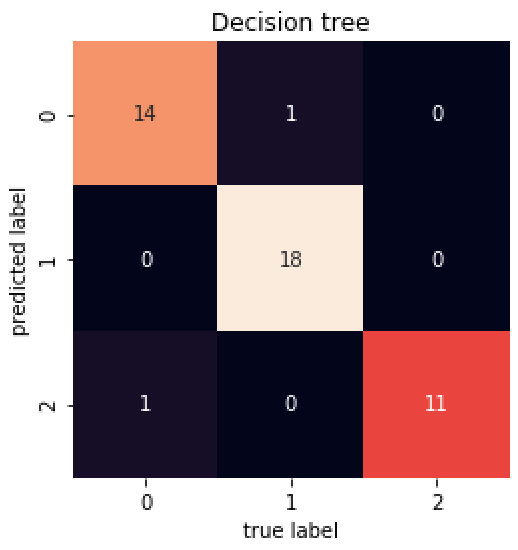
K-Nearest Neighbors: precision: 0.94; recall: 0.93; accuracy: 0.93.

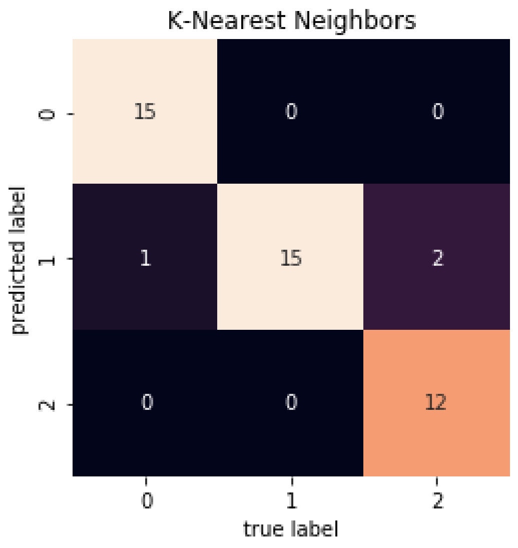
Support Vector Machine: precision: 1.00; recall: 1.00; accuracy: 1.00.

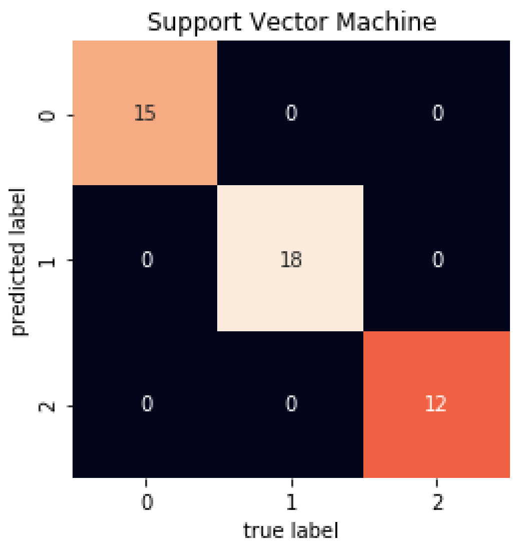
Gaussian Naive Bayes: precision: 1.00; recall: 1.00; accuracy: 1.00.

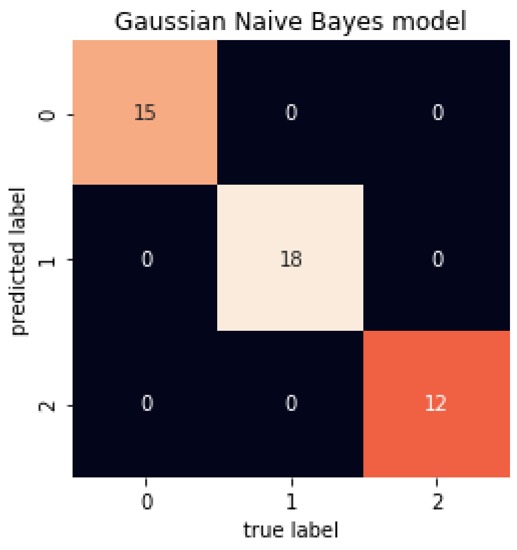
3. (c)

The resulting confusion matrix for each model are as follows:









3. (d)

SVM and Gaussian Naive Bayes model perform the best because they have the highest precision, recall and accuracy.

K-Nearest Neighbors performs the worst because it has the lowest precision, recall and accuracy.

Precision measures the correctness of predicted positive labels. Recall measures how many positive labels are successfully predicted amongst all positive labels. Accuracy measures the number of correctly classified examples.

```
In [ ]:
```