Using k-fold cross-validation to evaluate a decision tree model

We will use scikit-learn library and its built-in IRIS dataset for the demonstration of k-fold cross-validation.We will demonstrate how we can implement the k-fold cross-validation using **the built-in cross_val_score method**.

The IRIS dataset – Sample dataset

The IRIS dataset comes bundled with the Scikit-learn library. It has 150 observations that consist of 50 samples of each of three species of Iris flower. These species of IRIS flowers are “setosa“, “versicolor” and “virginica“. This dataset is a standard, cleansed, and preprocessed multivariate dataset. Each IRIS dataset sample has four input features that are:

Sepal length (cm) Sepal width (cm) Petal length (cm), and Petal width (cm)

k-fold cross-validation using custom for loop

It is essential to use cross-validation to avoid overfitting the models. To use cross-validation, we can use the below sample code. Note that, in the below code, we are using the IRIS dataset.

In [24]:
```
#import datasets from sklearn library
from sklearn import datasets
data = datasets.load_iris()

#Import decision tree classification model and cross validation
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score, KFold, c
from sklearn.metrics import accuracy_score
```

In [36]:
```
df_observe = pd.DataFrame(data=data.data, columns=data.feature_names)
df_observe
```

Out[36]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [25]:
```
#Extract a holdout set at the very begining
X_train_set, X_holdout, y_train_set, y_holdout = train_test_split(data.data, da
                                  stratify = data.target, random_state = 42, test
```

N.B. Sometimes referred to as “testing” data, a holdout subset provides a final estimate of the machine learning model's performance after it has been trained and validated. Holdout sets should never be used to make decisions about which algorithms to use or for improving or tuning algorithms

In [26]:
```
#Get input and output datasets values in X and Y variables
X = X_train_set
y = y_train_set
```

In [23]:
```
# Decision Tree

dt = DecisionTreeClassifier(criterion='gini', max_depth = 2, \
                        min_samples_leaf = 0.10, random_state = 42)

scores = cross_val_score(dt, X, y, cv = 5)
print("\n" + ("*" * 100))
print("The cross-validation scores using cross_val_score method are \n{0}".form
print("*" * 100)

import numpy as np
print("\n" + ("*" * 100))
print("Mean of k-fold scores using cross_val_score method is {0}".format(np.mea
print("*" * 100)
print("\n")
```

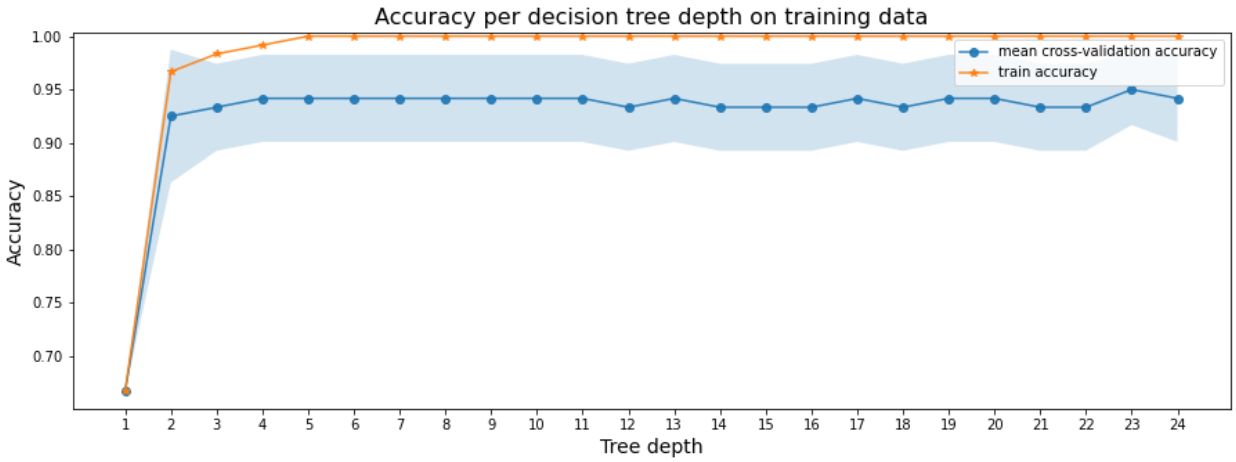
```
**************************************************************************
**********
The cross-validation scores using cross_val_score method are
[0.875      0.95833333 0.95833333 0.91666667 0.91666667]
**************************************************************************
**********

**************************************************************************
**********
Mean of k-fold scores using cross_val_score method is 0.925
**************************************************************************
**********
```

Finding Optimal Depth via K-fold Cross-Validation

The trick is to choose a range of tree depths to evaluate and to plot the estimated performance +/- 2 standard deviations for each depth using K-fold cross validation. We provide a Python code that can be used in any situation, where you want to tune your decision tree given a predictor tensor X and labels Y. The code includes the training set performance in the plot, while scaling the y-axis to focus on the cross-validation performance.

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt

# function for fitting trees of various depths on the training data using cross
def run_cross_validation_on_trees(X, y, tree_depths, cv=5, scoring='accuracy'):
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    accuracy_scores = []
    for depth in tree_depths:
        tree_model = DecisionTreeClassifier(max_depth=depth)
        cv_scores = cross_val_score(tree_model, X, y, cv=cv, scoring=scoring)
        cv_scores_list.append(cv_scores)
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())
        accuracy_scores.append(tree_model.fit(X, y).score(X, y))
    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    accuracy_scores = np.array(accuracy_scores)
    return cv_scores_mean, cv_scores_std, accuracy_scores

# function for plotting cross-validation results
def plot_cross_validation_on_trees(depths, cv_scores_mean, cv_scores_std, accur
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accuracy
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std, cv_scores_mean+2*cv
    ylim = plt.ylim()
    ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Tree depth', fontsize=14)
```

```
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_ylim(ylim)
    ax.set_xticks(depths)
    ax.legend()

# fitting trees of depth 1 to 24
sm_tree_depths = range(1,25)
sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores = run_cross_validation_

# plotting accuracy
plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean, sm_cv_scores_
                               'Accuracy per decision tree depth on training da
```



In [29]:
```
idx_max = sm_cv_scores_mean.argmax()
sm_best_tree_depth = sm_tree_depths[idx_max]
sm_best_tree_cv_score = sm_cv_scores_mean[idx_max]
sm_best_tree_cv_score_std = sm_cv_scores_std[idx_max]
print('The depth-{} tree achieves the best mean cross-validation accuracy {} +/
      sm_best_tree_depth, round(sm_best_tree_cv_score*100,5), round(sm_best_tre
```

```
The depth-23 tree achieves the best mean cross-validation accuracy 95.0 +/- 1.
66667% on training dataset
```

The method selects tree depth 23 because it achieves the best average accuracy on training data using cross-validation folds with size 23. The lower bound of the confidence interval of the accuracy is high enough to make this value significant. When more nodes are added to the tree, it is clear that the cross-validation accuracy changes towards zero.

The tree of depth 6 achieves perfect accuracy (100%) on the training set, this means that each leaf of the tree contains exactly one sample and the class of that sample will be the prediction. Depth-6 tree is overfitting to the training set.

The tree depth 5 we chose via cross-validation helps us avoiding overfitting and gives a better chance to reproduce the accuracy and generalize the model on test data as presented below.

In [37]:
```
# function for training and evaluating a tree
def run_single_tree(X, y, X_holdout, y_holdout, depth):
    model = DecisionTreeClassifier(max_depth=depth).fit(X, y)
    accuracy_train = model.score(X, y)
    accuracy_test = model.score(X_holdout, y_holdout)
```

```
    print('Single tree depth: ', depth)
    print('Accuracy, Training Set: ', round(accuracy_train*100,5), '%')
    print('Accuracy, Test Set: ', round(accuracy_test*100,5), '%')
    return accuracy_train, accuracy_test


# train and evaluate a 5-depth tree
sm_best_tree_accuracy_train, sm_best_tree_accuracy_test = run_single_tree(X, y,
                                                                           X_hol
                                                                           sm_be
```

```
Single tree depth:  23
Accuracy, Training Set:  100.0 %
Accuracy, Test Set:  93.33333 %
```

In [ ]: