Exercise 8.1

The Python code for this exercise can be found in listing 1.

(a) It this case, we should make sure that the training set and test set both contain a distribution of classes similar to the original data set. The optional parameter `stratify` to `train_test_split` can be used to retain the same proportion of classes in the different sets. When we check this, we find that the original data set contains 63% benign tumors and 37% malignant tumors, and that this proportion is retained in the test and training sets as well.

(b) The different metrics are calculated as follows:

(i) Accuracy: The ratio of correct predictions to the number of total predictions.

(ii) Precision: The ratio of true positives to the sum of true positives and false positives.

(iii) Recall: The ratio of true positives to the sum of true positives and false negatives.

(iv) F1: This is the harmonic mean of the precision and recall.

(v) The confusion matrix: This is a table that summarizes the predictions. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.

One example of a case where false positives should be avoided is a classification where youtube videos are classified as safe for kids or not. Here, a false positive would mean that we think a video is safe, but it is in reality not.

One example of a case where false negatives should be avoided is the case we are considering in this exercise: A malignant tumor classified as a benign one could mean that we miss a potentially life-threatening condition for a patient.

In this exercise, we will use the recall. The reason for this is that a high recall is achieved when we have a large number of true positives and a small number of false negatives.

(c) The results for the k-nearest neighbor classifier with 3 neighbors are given in table 1. The confusion matrix is shown in Fig. 1.

(d) The average recall obtained by the `GridSearchCV` as a function of the number of nearest neighbors is shown in Fig. 2. In this case, the largest recall is obtained for 5 neighbors. However, we note that the standard deviations are rather large here, so it is difficult to pick the optimal number of neighbors. We see that using 3 neighbors is approximately as good as using 5 neighbors. We will in the following use 5 neighbors as the optimal number as this is the value found by `GridSearchCV` and we can then compare it with the classifier using 3 neighbors.

Variable	Unit
Accuracy	0.98
Precision	0.97
Recall	0.97
F1	0.97

Table 1: *Results for the* 3*-nearest neighbor classifier.*

The results for the k-nearest neighbor classifier with 5 neighbors are given in table 2. The confusion matrix is shown in Fig. 3.

Variable	Unit
Accuracy	0.97
Precision	0.99
Recall	0.94
F1	0.96

Table 2: *Results for the* 5*-nearest neighbor classifier.*

(e) The results for the decision tree classifier are given in table 3. The confusion matrix is

Variable	Unit
Accuracy	0.90
Precision	0.83
Recall	0.93
F1	0.88

Table 3: *Results for the decision tree classifier.*

shown in Fig. 4.

(f) The average recall obtained by the `GridSearchCV` as a function of the depth is shown in Fig. 5. In this case, the largest recall is obtained for a depth of 8. Again, we see that there are relatively large standard deviations for the recall values in Fig. 5. Decision trees are easily overfitted and we would expect that a tree with a smaller depth is less prone to overfitting. Based on this, we could here choose a decision tree with a lower depth, say 5 and try this one for the test set. We will in the following trust `GridSearchCV` and use the optimized classifier found from this method.

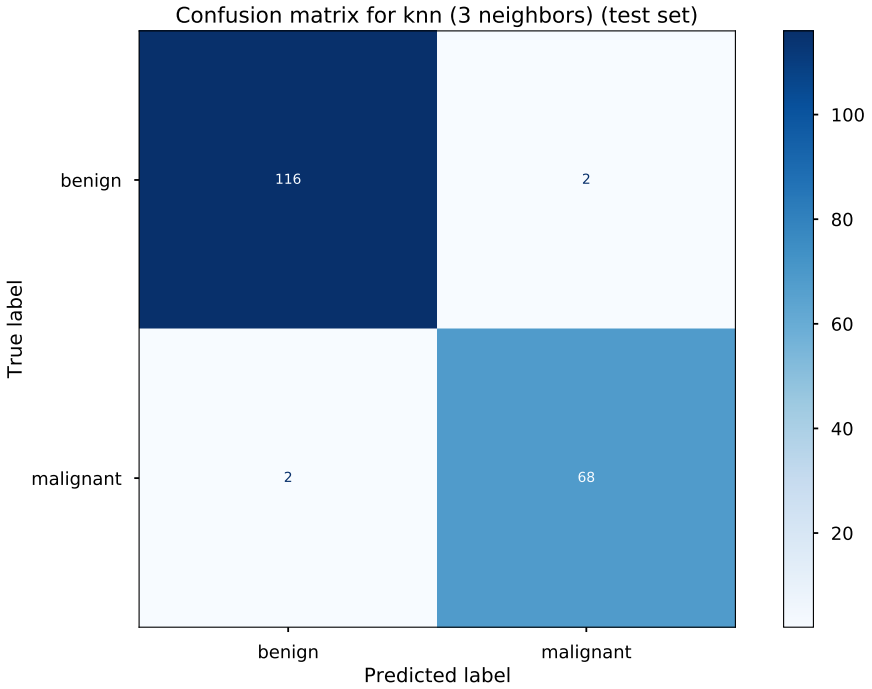

Figure 1: *Confusion matrix for the* 3*-nearest neighbor classifier.*

Variable	**Unit**
Accuracy	0.89
Precision	0.81
Recall	0.93
F1	0.86

Table 4: *Results for the decision tree classifier with a maximum depth of* 8*.*

The results for this decision tree classifier are given in table 4. The confusion matrix is shown in Fig. 6.

We note that optimizing the depth does not improve the classifier. This indicates that it might still be overfitted.

(g) The results for the Naive Bayes, LDA and Random forest classifiers are shown in tables 5–7. The corresponding confusion matrices are shown in Fig. 7–9. For the Random forest classifier, the variable importance is shown in Fig. 10.

(h) The results for the classifiers are shown in Fig. 11. Here, the largest recall is attained for the k-nearest neighbor classifier with 3 neighbors. Further, this classifier also performs well for the other metrics and we also note that the number of false negatives (see

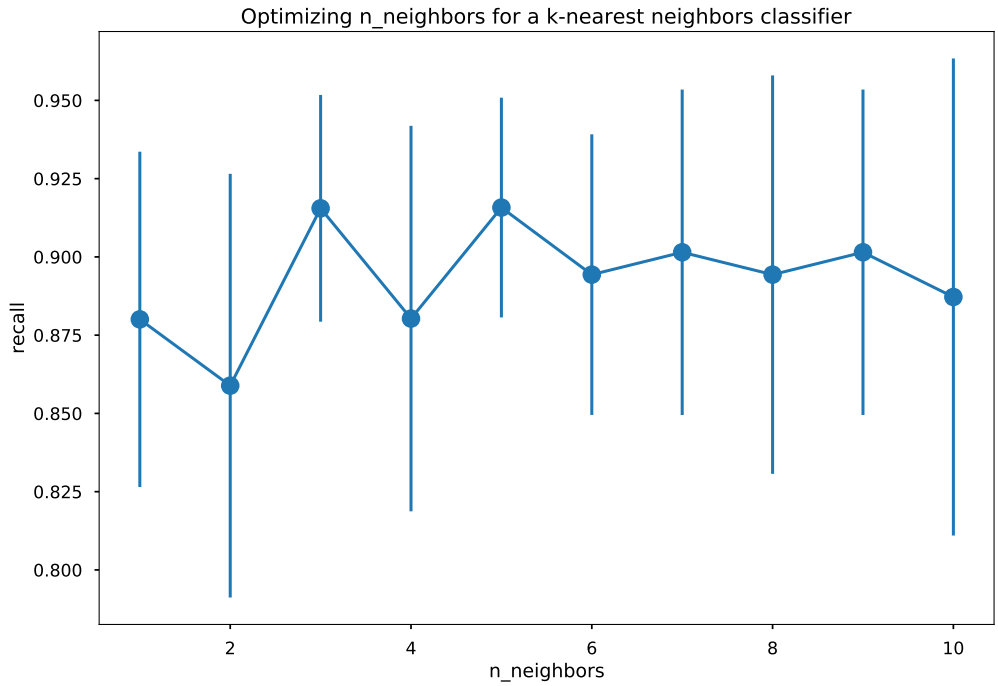

Figure 2: *The average recall as a function of the number of nearest neighbors.*

Variable	**Unit**
Accuracy	0.94
Precision	0.88
Recall	0.96
F1	0.92

Table 5: *Results for the Naive Bayes classifier.*

Variable	**Unit**
Accuracy	0.95
Precision	0.97
Recall	0.90
F1	0.93

Table 6: *Results for the LDA classifier.*

Fig. 1) is smaller than for the other classifiers. We would therefore select this classifier.

(i) The decision tree is shown in Fig.12. The SVM classifier attains a higher recall than

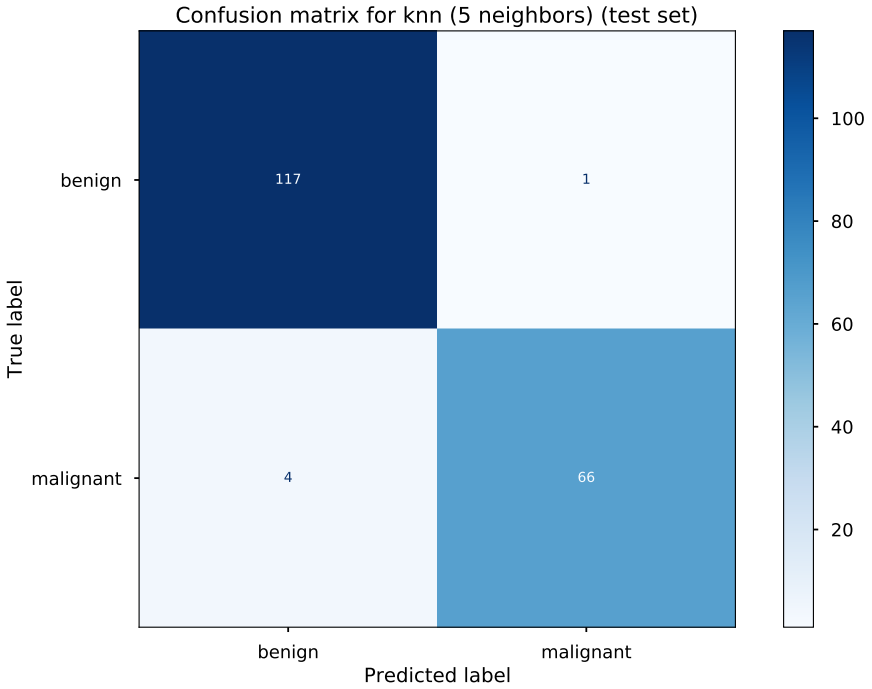

Figure 3: *Confusion matrix for the* 5*-nearest neighbor classifier.*

Variable	**Unit**
Accuracy	0.96
Precision	0.94
Recall	0.96
F1	0.95

Table 7: *Results for the Random forest classifier.*

the other classifiers, see Fig. 13. The confusion matrix for a optimized SVM classifier is shown in Fig. 14. We see here that the SVM classifier outperforms the other classifiers (for the recall metric) we have tried, and we would then prefer this one in a real-life setting.

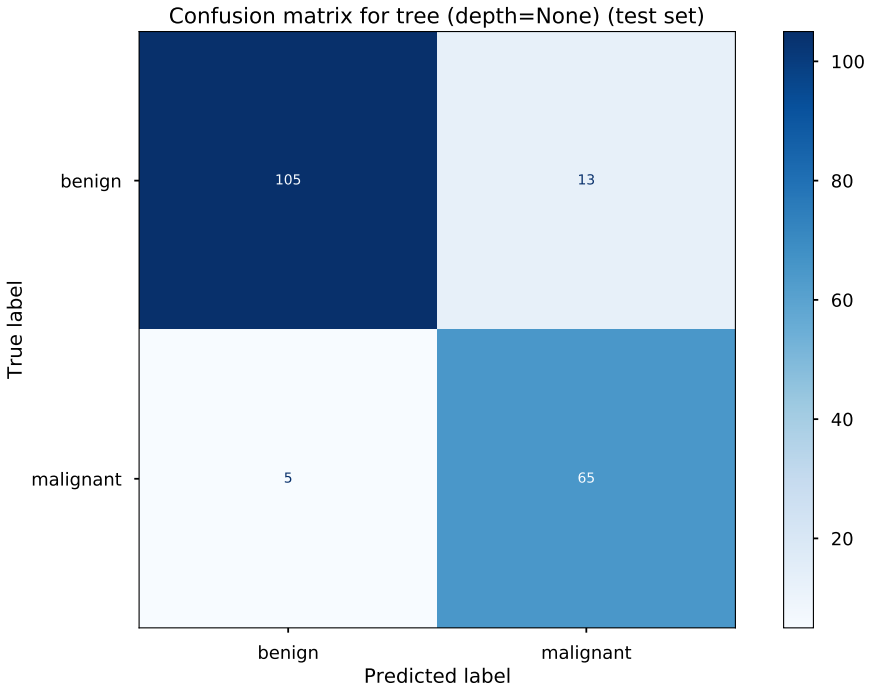

Figure 4: *Confusion matrix for the decision tree classifier.*

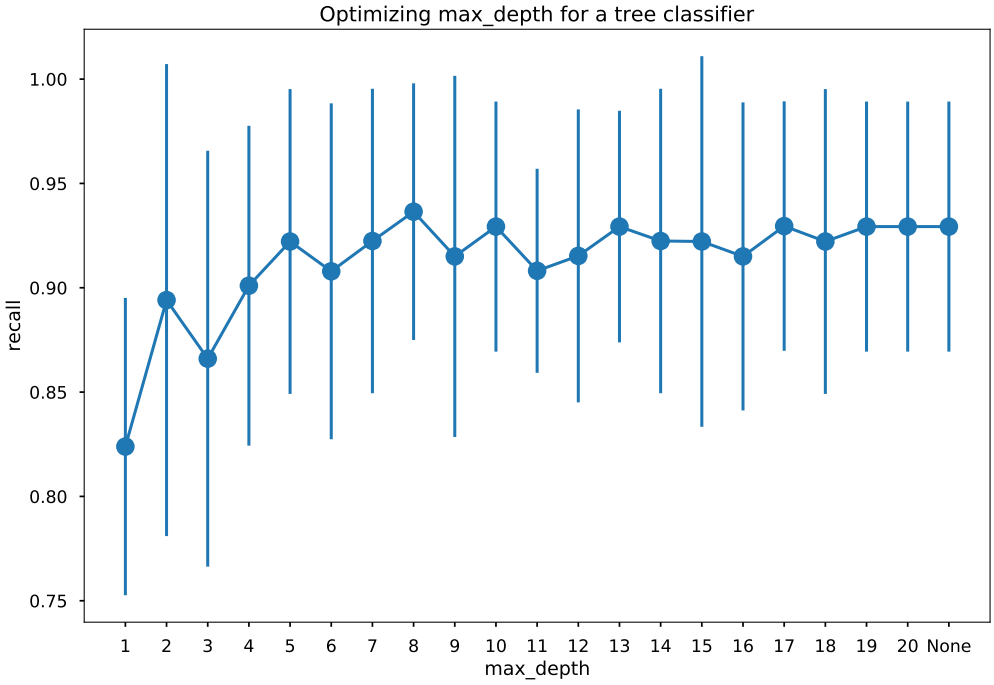

Figure 5: *The average recall as a function of the depth of the decision tree.*

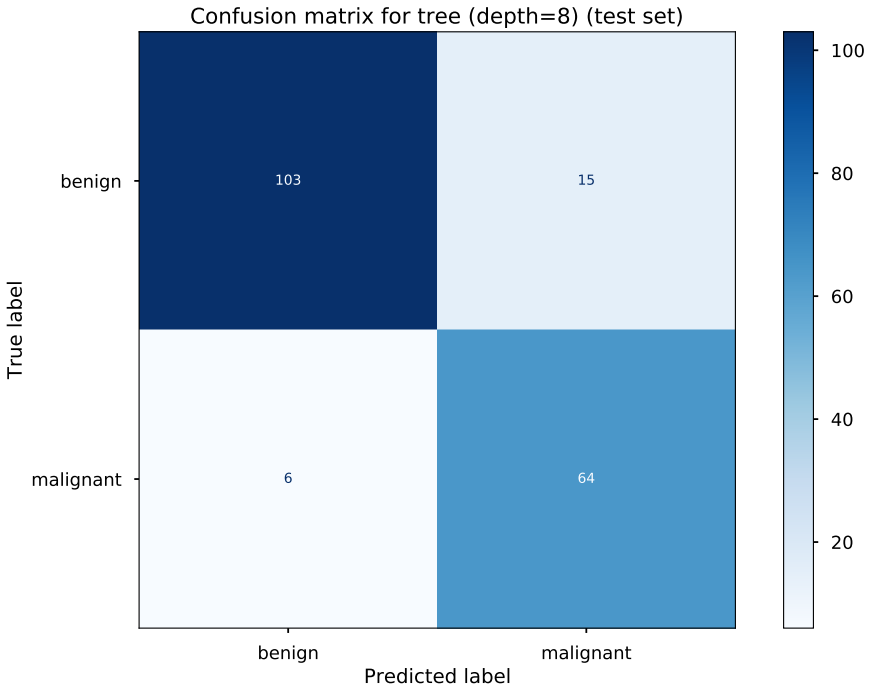

Figure 6: *Confusion matrix for the decision tree classifier with a maximum depth of* 8.

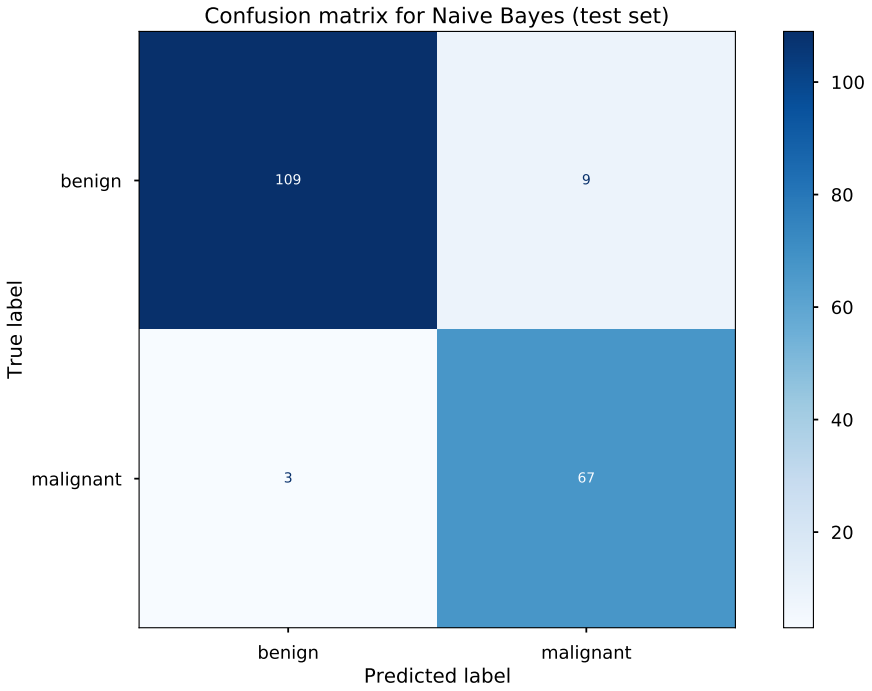

Figure 7: *Confusion matrix for the Naive Bayes classifier.*

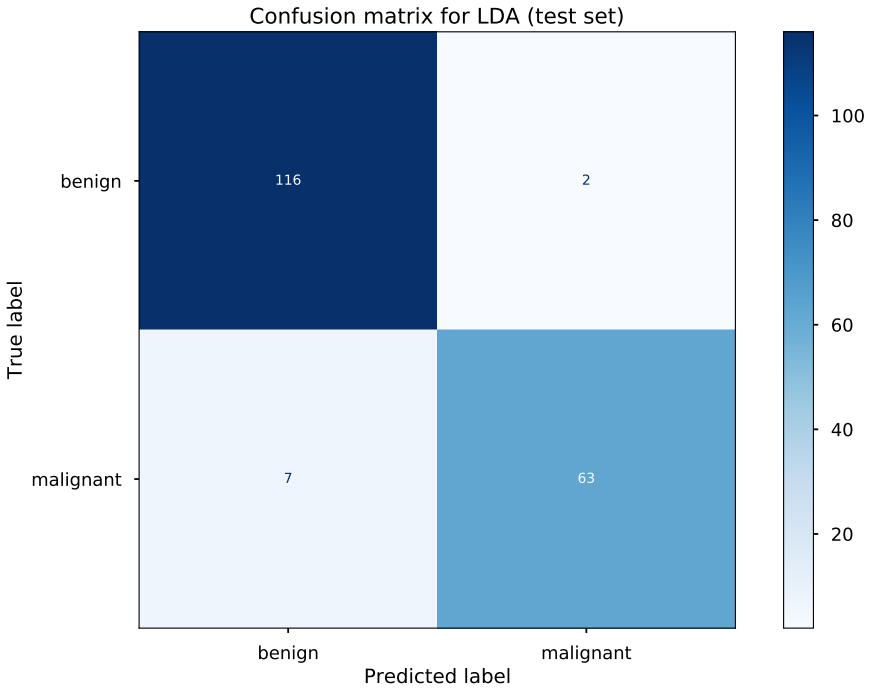

Figure 8: *Confusion matrix for the LDA classifier.*

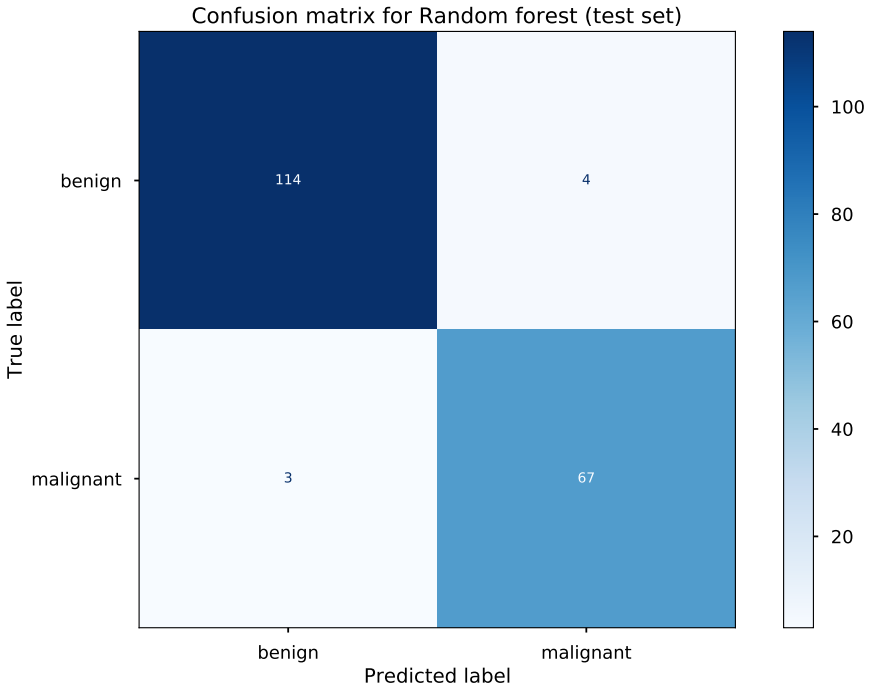

Figure 9: *Confusion matrix for the Random forest classifier.*

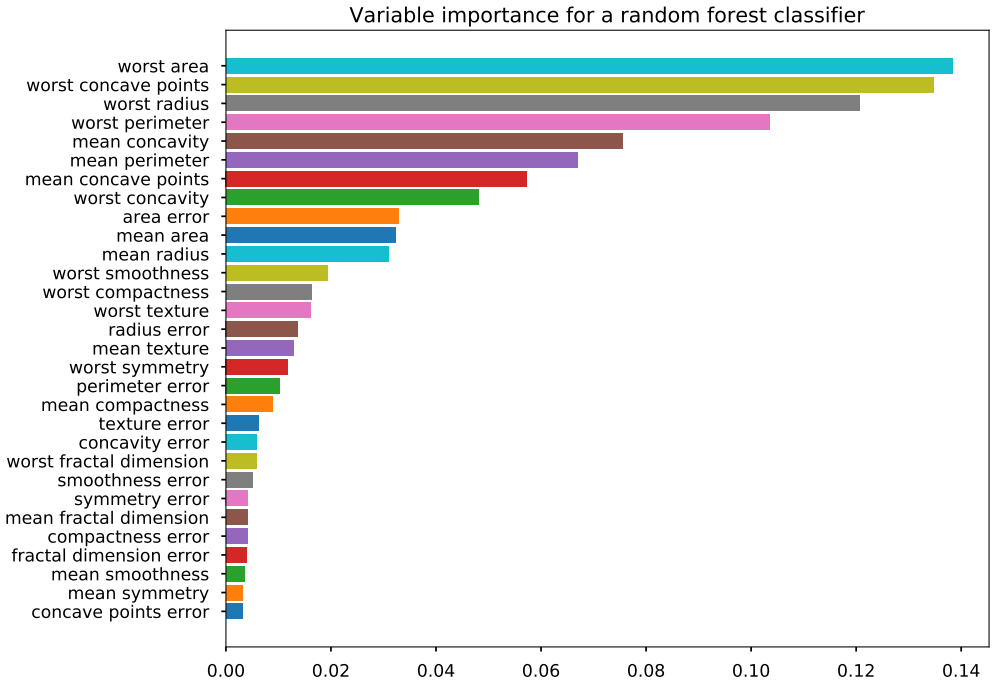

Figure 10: *Variable importance for the Random forest classifier.*

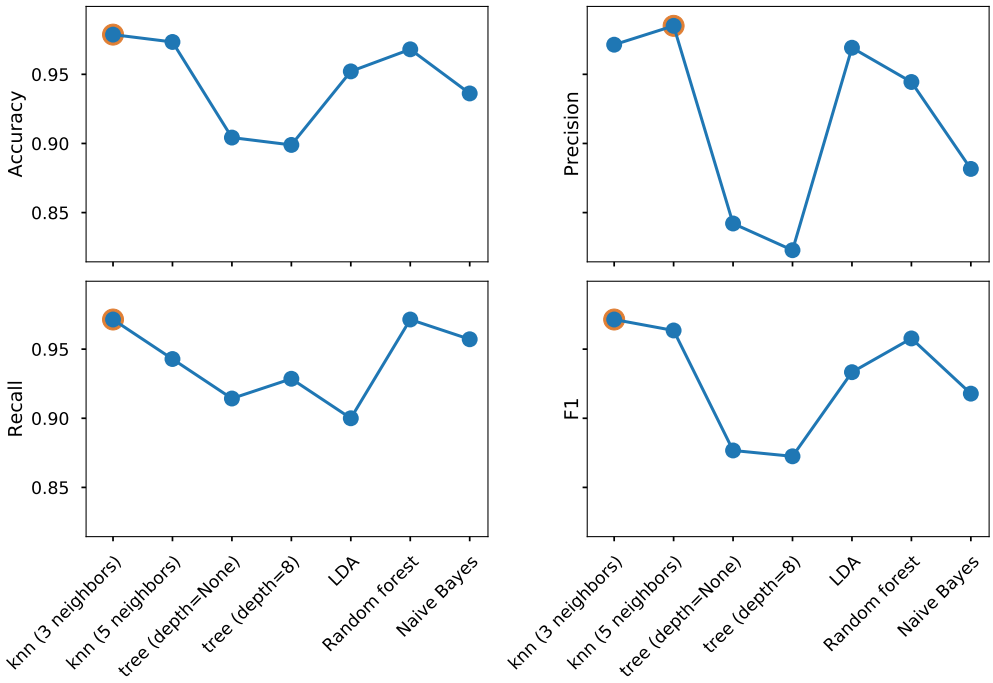

Figure 11: *Comparison of the different classifiers. The maximum score for each metric is marked with a orange circle.*

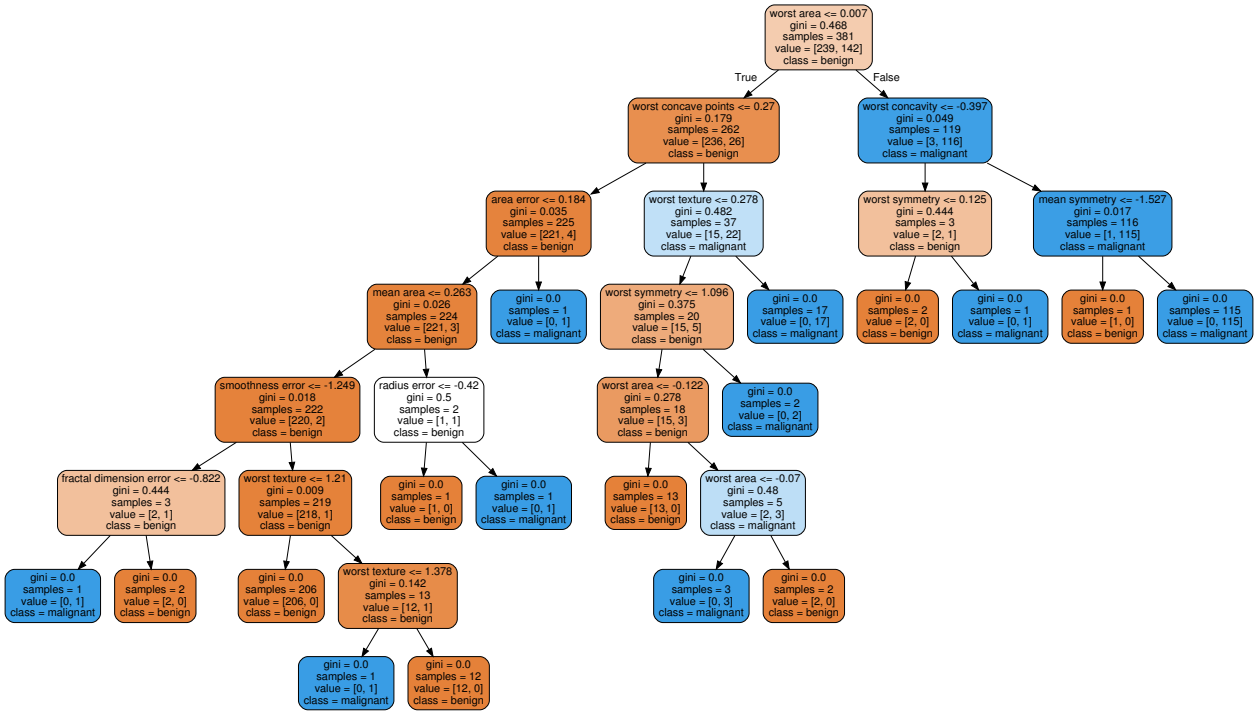

Figure 12: *The decision tree with a depth of 8.*

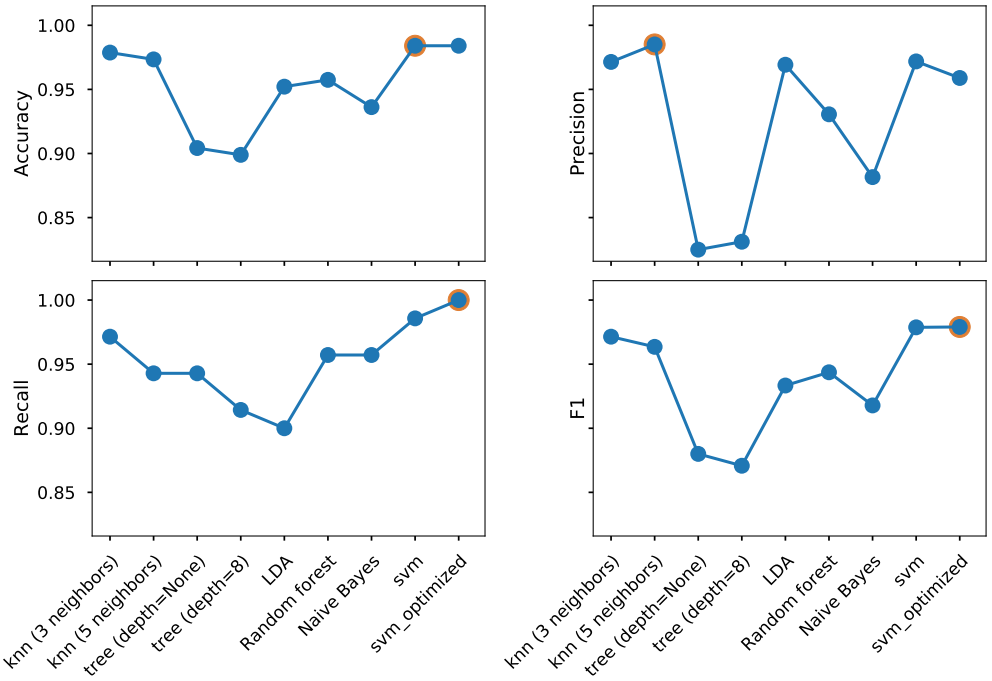

Figure 13: *Comparison of the different classifiers.*

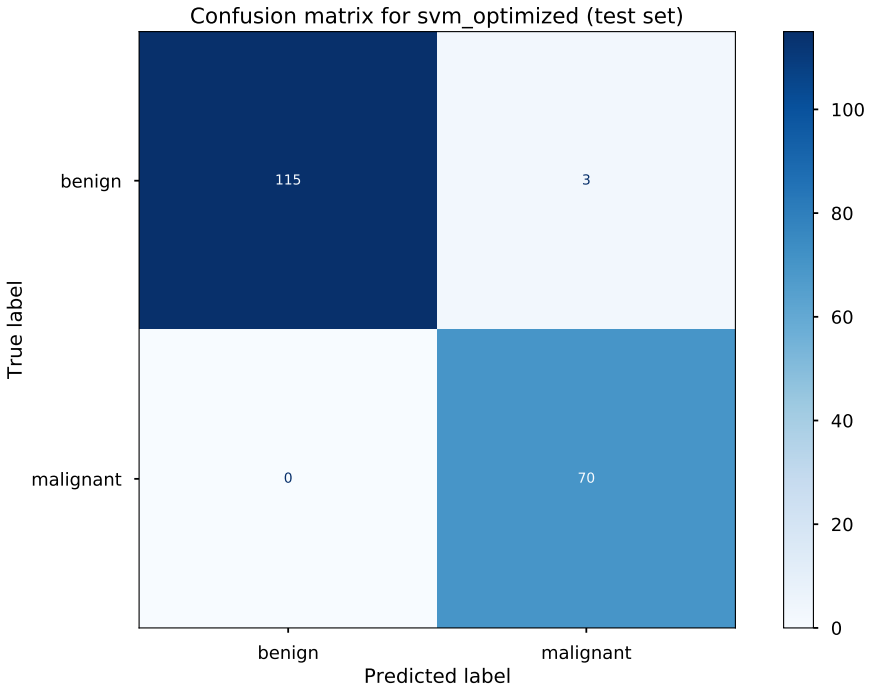

Figure 14: *Confusion matrix for a optimized SVM classifier.*

Python solutions

```
"""Solution to exercise 8."""
import random
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import (
    train_test_split,
    GridSearchCV,
)
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    confusion_matrix,
    precision_score,
    recall_score,
    plot_confusion_matrix,
)
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz, plot_tree
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from slugify import slugify  # For making safe file names.
plt.style.use('seaborn-talk')


def count_items(items):
    """Count items (as fractions) in a dictionary."""
    counts = {}
    for val in items:
        if val not in counts:
            counts[val] = 0
        counts[val] += 1
    for key, val in counts.items():
        counts[key] = val / len(items)
    for key in sorted(counts):
        print('{}: {}'.format(key, counts[key]))


def classify_knn(X_train, y_train):
    """Run classification with k-nearest neighbors."""
    clf = KNeighborsClassifier()
    parameters = [{'n_neighbors': range(1, 11)}]
    grid = GridSearchCV(
        clf,
```

```
        parameters,
        cv=5,
        scoring='recall',
        return_train_score=True,
    )
    grid.fit(X_train, y_train)
    print('Best parameters for knn:', grid.best_params_)
    # Plot the scores:
    fig1, ax1 = plt.subplots(constrained_layout=True)
    ax1.errorbar(
        parameters[0]['n_neighbors'],
        grid.cv_results_['mean_test_score'],
        yerr=grid.cv_results_['std_test_score'],
        marker='o', markersize=14
    )
    ax1.set(xlabel='n_neighbors', ylabel='recall')
    ax1.set_title(
        'Optimizing n_neighbors for a k-nearest neighbors classifier'
    )
    fig1.savefig('optknn.pdf', bbox_inches='tight')
    return grid.best_estimator_, grid.best_params_['n_neighbors']


def classify_decision(X_train, y_train, feature_names=None, class_names=None):
    """Run classification with a decision tree."""
    clf = DecisionTreeClassifier()
    parameters = [{'max_depth': list(range(1, 21)) + [None]}]
    grid = GridSearchCV(
        clf,
        parameters,
        cv=5,
        scoring='recall',
        return_train_score=True
    )
    grid.fit(X_train, y_train)
    print('Best parameters for Decision tree:', grid.best_params_)
    clf_best = grid.best_estimator_
    # Plot the tree:
    export_graphviz(clf_best, out_file='tree.dot',
                    feature_names=feature_names,
                    class_names=class_names,
                    rounded=True,
                    filled=True)
    fig0, ax0 = plt.subplots(figsize=(10, 10))
    plot_tree(clf_best, feature_names=feature_names,
              class_names=class_names, rounded=True, filled=True, ax=ax0)
    fig0.savefig('tree.png', bbox_inches='tight')
    # Plot the scores:
    fig1, ax1 = plt.subplots(constrained_layout=True)
    depth = list(range(1, 21)) + [21]
```

```
    ax1.errorbar(depth, grid.cv_results_['mean_test_score'],
                 yerr=grid.cv_results_['std_test_score'],
                 marker='o', markersize=14)
    ax1.set(xlabel='max_depth', ylabel='recall')
    ax1.set_xticks(depth)
    ax1.set_xticklabels(list(range(1, 21)) + ['None'])
    ax1.set_title('Optimizing max_depth for a tree classifier')
    fig1.savefig('optdecision.pdf', bbox_inches='tight')
    return grid.best_estimator_, grid.best_params_['max_depth']


def classify_svc(X_train, y_train):
    """Run classification with a decision tree."""
    clf = SVC()
    parameters = [
        {
            'C': [0.01, 0.1, 1.0, 10.0],
            'degree': [1, 2, 3, 4, 5],
            'gamma': ['scale', 'auto'],
        }
    ]
    grid = GridSearchCV(
        clf,
        parameters,
        cv=10,
        scoring='recall',
        return_train_score=True
    )
    grid.fit(X_train, y_train)
    print('Best parameters for SVC tree:', grid.best_params_)
    return grid.best_estimator_, grid.best_params_


def score(clf, X, y_true, name='Classifier', class_names=None):
    """Test a classifier."""
    y_hat = clf.predict(X)
    accuracy = accuracy_score(y_true, y_hat)
    precision = precision_score(y_true, y_hat)
    recall = recall_score(y_true, y_hat)
    f1 = f1_score(y_true, y_hat)
    conf = confusion_matrix(y_true, y_hat)
    print('\nScores for:', name)
    print('Accuracy:', accuracy)
    print('Precision:', precision)
    print('Recall:', recall)
    print('F1', f1)
    print('Confusion matrix:', conf)
    if class_names is not None:
        fig = plot_confusion_matrix(
            clf, X, y_true,
```

```
            display_labels=class_names,
            cmap=plt.cm.Blues,
            values_format='d',
        )
        fig.ax_.set_title('Confusion matrix for {}'.format(name))
        fig.figure_.tight_layout()
        fig.figure_.savefig('confmat{}.pdf'.format(slugify(name)),
                            bbox_inches='tight')
    result = {'accuracy': accuracy, 'precision': precision, 'recall': recall,
              'f1': f1, 'confusion': conf}
    return result


def plot_variable_importance(variables, clf):
    """Plot variable importance for a random forest."""
    figi, axi = plt.subplots(constrained_layout=True)
    importance = clf.feature_importances_
    idx = np.argsort(importance)
    y_pos = []
    y_label = []
    for i, idxi in enumerate(idx):
        y_pos.append(i)
        y_label.append(variables[idxi])
        axi.barh(i, importance[idxi], align='center')
    axi.set_yticks(y_pos)
    axi.set_yticklabels(y_label)
    axi.set_title('Variable importance for a random forest classifier')
    figi.savefig('variableimportance.pdf', bbox_inches='tight')
    return figi, axi


def score_classifiers(clfs, names, X_test, y_test, class_names):
    """Score several classifiers."""
    results = {}
    for name, clf in zip(names, clfs):
        result = score(clf, X_test, y_test,
                       name='{} (test set)'.format(name),
                       class_names=class_names)
        for key in result:
            if key not in results:
                results[key] = []
            results[key].append(result[key])
    return results


def plot_results(results, names):
    """Plot the results from scoring different classifiers."""
    fig, axes = plt.subplots(constrained_layout=True, nrows=2, ncols=2,
                             sharex=True, sharey=True)
    axes = axes.flatten()
```

```
    for axi, scorer in zip(axes, ('accuracy', 'precision', 'recall', 'f1')):
        scores = results[scorer]
        xpos = range(len(scores))
        idx = np.argmax(scores)
        axi.plot(xpos, scores, marker='o', markersize=12)
        axi.scatter(xpos[idx], scores[idx], marker='o',
                    s=250, alpha=0.8, color='#d95f02')
        axi.set_xticks(xpos)
        axi.set_xticklabels(names, rotation=45,
                            rotation_mode='anchor', ha='right')
        axi.set(ylabel=scorer.title())
    fig.savefig('comparison.pdf', bbox_inches='tight')
    return fig, axes


def load_data():
    """Load and preprocess data."""
    data_set = load_breast_cancer()
    x_data = pd.DataFrame(data=data_set['data'],
                          columns=data_set['feature_names'])
    scaler = StandardScaler()
    scaler.fit(x_data)

    X = scaler.transform(x_data)
    # Rename the target so that 0 is benign (negative) and
    # 1 is malignant (positive).
    y = [0 if i else 1 for i in data_set['target']]
    class_names = ['benign', 'malignant']
    return X, y, class_names, data_set['feature_names']


def main():
    """Load the data set and run classification."""
    # First, set global random states to get reproducible results:
    np.random.seed(4)
    random.seed(4)
    # Load the data and make the training/test sets:
    X, y, class_names, variables = load_data()
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.33, random_state=10, stratify=y
    )
    print('Fraction of items in classes:')
    count_items(y)
    print('Fraction of items in classes in the training set:')
    count_items(y_train)
    print('Fraction of items in classes in the test set:')
    count_items(y_test)

    knn_3 = KNeighborsClassifier(n_neighbors=3)
    tree0 = DecisionTreeClassifier()
```

```
    lda = LinearDiscriminantAnalysis()
    rnd_tree = RandomForestClassifier()
    bayes = GaussianNB()
    svc = SVC()

    for clf in (knn_3, tree0, lda, rnd_tree, bayes, svc):
        clf.fit(X_train, y_train)

    # Do the optimizations for knn and the tree:
    knn, k = classify_knn(X_train, y_train)
    tree, depth = classify_decision(X_train, y_train,
                                    feature_names=variables,
                                    class_names=class_names)
    # And a SVC for fun:
    svc_opt, _ = classify_svc(X_train, y_train)

    # Assess the different classifiers
    clfs = [knn_3, knn, tree0, tree, lda, rnd_tree, bayes,
            svc, svc_opt]
    names = [
        'knn (3 neighbors)',
        'knn ({} neighbors)'.format(k),
        'tree (depth=None)',
        'tree (depth={})'.format(depth),
        'LDA',
        'Random forest',
        'Naive Bayes',
        'svm',
        'svm_optimized',
    ]
    results = score_classifiers(clfs, names, X_test, y_test, class_names)

    # Plot the comparison:
    plot_results(results, names)

    # Plot the variable importance from the random forest:
    plot_variable_importance(variables, rnd_tree)
    # Show all plots:
    plt.show()


if __name__ == '__main__':
    main()
```

Listing 1: *Python code for exercise 8.*