Lab 1 Solution

1 Lab1: Decision Trees

Author: David Boetius
Date: 5.11.2020 Group: 3

[1]:
```
import pandas as pd
from matplotlib import pyplot
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

1.0.1 Data

Import the data sets and split them into label and attribute dataframes

[2]:
```
diabetes = pd.read_csv('diabetes.csv')
glass = pd.read_csv('glass.csv')

X_diabetes = diabetes.drop(['class'], axis=1)
y_diabetes = diabetes['class']

X_glass = glass.drop(['class'], axis=1)
y_glass = glass['class']
```

Create training and test sets for both data sets

[3]:
```
X_diabetes_train, X_diabetes_test, y_diabetes_train, y_diabetes_test = \
    train_test_split(X_diabetes, y_diabetes, train_size=0.66,␣
 ↪random_state=0xC0FFEE)
X_glass_train, X_glass_test, y_glass_train, y_glass_test = \
    train_test_split(X_glass, y_glass, train_size=0.66, random_state=0x1CE)
```

1.0.2 Task C - Training

Now that we have the data set up, we can start training. We will create four classifiers, one one level decision tree for each data set and one multilevel decision tree for each data set.

[4]:
```
diabetes_one_level = tree.DecisionTreeClassifier(criterion='entropy',␣
↪max_depth=1)
diabetes_one_level.fit(X_diabetes_train, y_diabetes_train)

glass_one_level = tree.DecisionTreeClassifier(criterion='entropy', max_depth=1)
glass_one_level.fit(X_glass_train, y_glass_train)

diabetes_multi_level = tree.DecisionTreeClassifier(criterion='entropy',␣
↪max_depth=None)
diabetes_multi_level.fit(X_diabetes_train, y_diabetes_train)

glass_multi_level = tree.DecisionTreeClassifier(criterion='entropy',␣
↪max_depth=None)
glass_multi_level.fit(X_glass_train, y_glass_train)
```

[4]:
```
DecisionTreeClassifier(criterion='entropy')
```

Having the classifiers trained we can now compare their performance. For each classifier we will calculate the accuracy on the training data and the corresponding test set.

[5]:
```
def assess_accuracy(classifier, X, true_y):
    predicted_y = classifier.predict(X)
    return accuracy_score(true_y, predicted_y)

accuracies = [
    [
        'diabetes one-level',
        assess_accuracy(diabetes_one_level, X_diabetes_train, y_diabetes_train),
        assess_accuracy(diabetes_one_level, X_diabetes_test, y_diabetes_test)
    ], [
        'diabetes multi-level',
        assess_accuracy(diabetes_multi_level, X_diabetes_train,␣
↪y_diabetes_train),
        assess_accuracy(diabetes_multi_level, X_diabetes_test, y_diabetes_test)
    ], [
        'glass one-level',
        assess_accuracy(glass_one_level, X_glass_train, y_glass_train),
        assess_accuracy(glass_one_level, X_glass_test, y_glass_test)
    ], [
        'glass multi-level',
        assess_accuracy(glass_multi_level, X_glass_train, y_glass_train),
        assess_accuracy(glass_multi_level, X_glass_test, y_glass_test)
    ]
]
pd.DataFrame(accuracies, columns=['name', 'train accuracy', 'test accuracy'])
```

[5]:

	name	train accuracy	test accuracy
0	diabetes one-level	0.687747	0.610687
1	diabetes multi-level	1.000000	0.679389
2	glass one-level	0.475177	0.397260
3	glass multi-level	1.000000	0.589041

Task C - Discussion of results One thing that we able to observe directly is that the multi-level classifiers both were perfectly fitted to the training data, archiving an accuracy 1.0, meaning that the classifiers are able to predict the class labels on these data sets without any error. The one-level classifiers, using only a single attribute of the data, are not able to archive this accuracy on the training data. Their structural limitation does not allow them to perfectly fit to the data set.

Both multi-level decision trees perform significantly better on the training data set, than on the test data set, so both classifiers are clearly over-fitted. For the one-level trees we can observe similar behavior, however the discrepancy between the data sets is way smaller. While those classifiers theoretically also fulfill the definition of over-fitting, they only slightly over-fit and the difference in the accuracies is not very significant.

To compare the overall accuracy rates on the data sets, we can have a look at the performance of the majority voting classifier. The most common class in the diabetes data set is 0, respectively `tested_negative`. 500 of the instances carry this class label, giving this classifier an accuracy of around 65%.

The most common class in the glass data set is `building_windows_non_float_processed`, with 76 instances, giving the majority voting classifier an accuracy of about 35%.

Classifier	accuracy
diabetes majority voting	65%
glass majority voting	35%

Based on these values we can conclude that the one-level decision tree for the diabetes data set is of rather poor quality as it archives about the same accuracy on both the test and the training set as the majority voting classifier would be expected to. The multi-level decision tree performs a little bit better in the test data than the majority voting classifier, but the improvement is very small.

For the glass data set, the situation is a bit different. Both the one-level classifier and the multi-level classifier outperform the majority voting classifier on both training and test data. The multi-level classifier performs significantly better than the one-level classifier on the data set.

The accuracy differences between training and test data stem from the fact, that the classifier has never seen the test data, however it has seen and tried to most accurately predict the training data. Naturally the classifiers therefore perform better on the training data than the test data.

The differences in the accuracies between the one-level and multi-level trees on the test data indicate that although the multi-level trees are way more over-fitted to the test data, they still perform in general better than the structurally very limited one-level trees.

Explainability One level decision trees, basing their decision on only a single attribute of the data are very easy to explain. Multi-level decision trees require longer explanations as they contain more branches, however the overall principle of operation is the same.

The overall concept of decision trees is rather transparent to humans. Whether they consist of only a single decision on one attribute or on multiple decisions organised in a tree has an influence on the complexity of the explanation needed to describe the behaviour if a decision tree, but the overall complexity needed to explain a multi level tree is still rather low.

Task D For this task we will train a number of classifiers, exploring the impact of pre-pruning with different values for the minimum amount of instances that need to be available in each child of a node to add those children to the decision tree. If there are less instances than the choosen minimum value available for a child, the node that has this child will be a leaf node in the decision tree, none if it's children will be added.

We start by providing a few functions that we will use first and generate data for the two datasets.

[6]:
```
def get_classifier_min_samples_leaf(X_train, y_train, min_samples_leaf):
    classifier = tree.DecisionTreeClassifier(criterion='entropy',␣
 ↪min_samples_leaf=min_samples_leaf)
    classifier.fit(X_train, y_train)
    return classifier

def get_accuracies_for_min_samples_per_leaf(X_train, y_train, X_test, y_test,␣
 ↪step_size = 5):
    accuracies = []
    for min_samples_leaf in range(1, len(X_train), step_size):
        decision_tree = get_classifier_min_samples_leaf(X_train, y_train,␣
 ↪min_samples_leaf)
        accuracies.append([
            min_samples_leaf,
            assess_accuracy(decision_tree, X_train, y_train),
            assess_accuracy(decision_tree, X_test, y_test)
        ])
    return accuracies

accuracies_diabetes = get_accuracies_for_min_samples_per_leaf(X_diabetes_train,␣
 ↪y_diabetes_train, X_diabetes_test, y_diabetes_test)
accuracies_glass = get_accuracies_for_min_samples_per_leaf(X_glass_train,␣
 ↪y_glass_train, X_glass_test, y_glass_test)
```

Having calculated the accuracies for the different parameters now, we can continue with plotting the results. We start with the results for the diabetes data set:

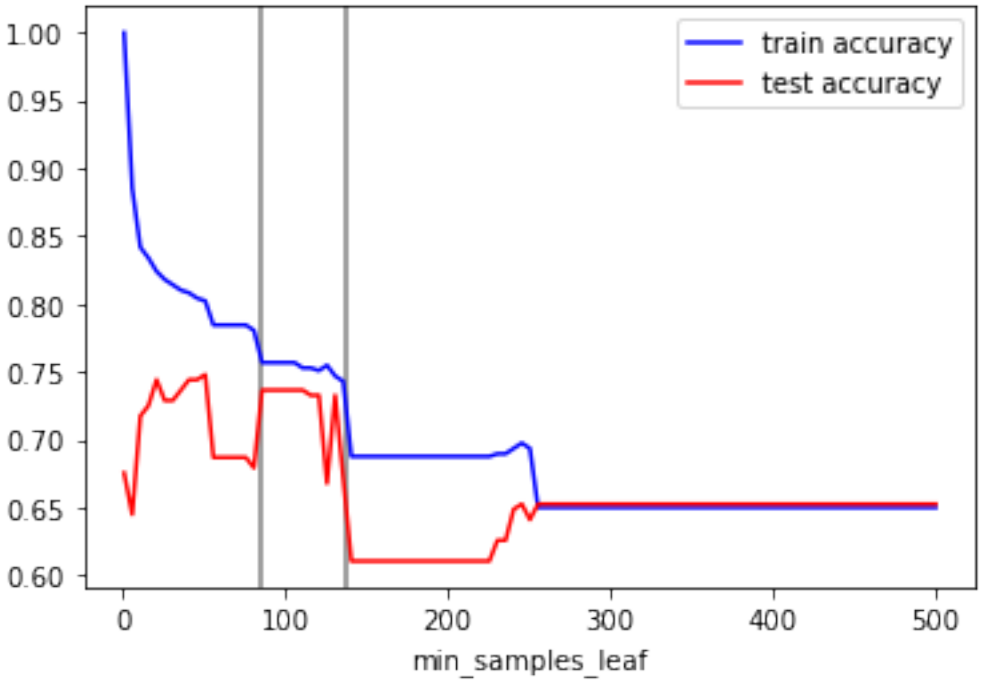
[7]:
```
accuracies_diabetes = pd.DataFrame(accuracies_diabetes,␣
 ↪columns=['min_samples_leaf', 'train accuracy', 'test accuracy'])

%matplotlib inline
```

```
# print lines indicating overfitting and underfitting areas
pyplot.axvline(x=85, color='gray')
pyplot.axvline(x=138, color='gray')

ax = pyplot.gca()
accuracies_diabetes.plot(kind='line', x='min_samples_leaf', y='train accuracy',␣
↪color='blue', ax=ax)
accuracies_diabetes.plot(kind='line', x='min_samples_leaf', y='test accuracy',␣
↪color='red', ax=ax)

pyplot.show()
```



This plot we should read from right to left. High values of min_samples_leaf do not allow the tree to grow much in depth, therefore on the right, when the whole data set is required to continue training on a node, not even a single node with children can be generated and we obtain the majority voting classifier. Below roughly 260 we obtain the one-level decision tree that we examined before. On the other end of the spectrum, at `min_samples_leaf = 1`, we find the multi-layer decision tree that we trained initially.

What we can see here is that coming from the right, initially the classification quality is poor, both on the training data and the test data. Having too simple models in this area (e.g. the majority voting classifier or the one-level decision tree), the models underfit the data since they do not allow enough complexity.

Looking at the other spectrum, at the left, we see that classification accuracy on the training set

is much higher than on the test set. This indicates overfitting. The model integrated too many details of the training data into itself to provide generalisation. Since it is fitted to closely to the training data, it no longer reflects the actually relationship given in the whole data and hence the accuracy on the test set is much lower.

The gray lines shown in the plot indicate the region of optimal fit. Here, the overall accuracy is not too bad and the accuracy on the training set does not differ too much from the accuracy on the test set. We can assume that the model generalised well from the given data set and actually approximates the true relationship in the data. Inside the optimal region we want the model with the highest accuracy, so we would tend to choose a model with the min_samples_leaf parameter close to the left gray line at `min_samples_leaf = 85`.

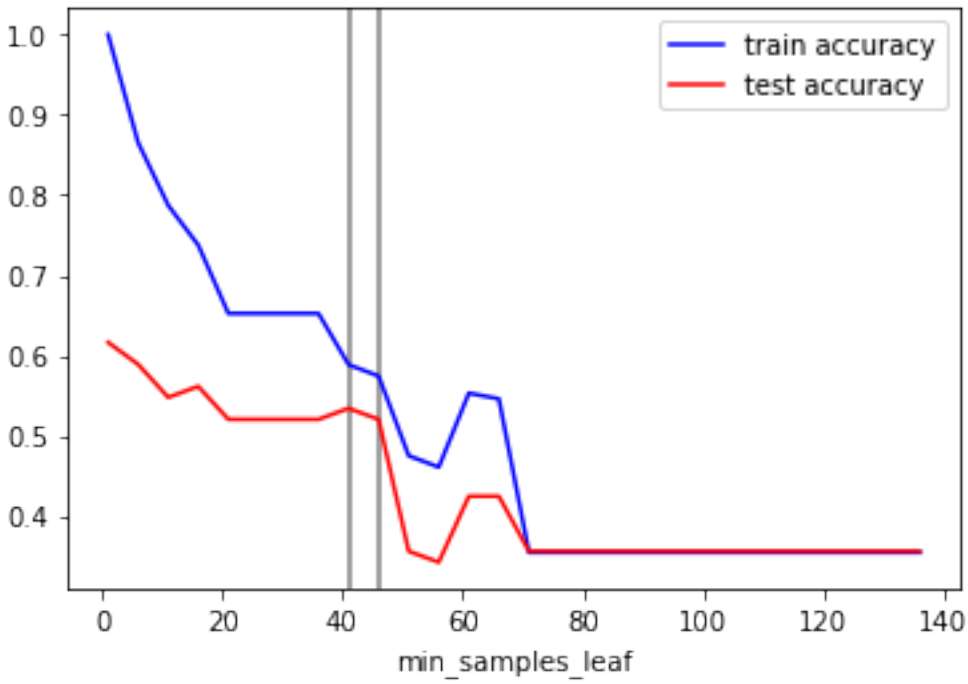
Now we continue with the results for the glass data set:

[8]:
```
accuracies_glass = pd.DataFrame(accuracies_glass, columns=['min_samples_leaf',␣
↪'train accuracy', 'test accuracy'])

%matplotlib inline
ax = pyplot.gca()

pyplot.axvline(x=41, color='gray')
pyplot.axvline(x=46, color='gray')

accuracies_glass.plot(kind='line', x='min_samples_leaf', y='train accuracy',␣
↪color='blue', ax=ax)
accuracies_glass.plot(kind='line', x='min_samples_leaf', y='test accuracy',␣
↪color='red', ax=ax)
pyplot.show()
```



Overall this plot shows us a similar story. On the right we find the majority voting classifier, on the left the multi-level tree from task C. In the same way, the overfitted region is left of the first gray line at `min_samples_leaf = 41`, with the degree of overfitting increasing to the left. On the right we find the region of underfitting. The overall accuracy in this region is rather low.

This plot has a much smaller region of optimality, between `min_samples_leaf = 41` and `min_samples_leaf = 46`. This means that it is even harder to find the right parameters for training the decision tree for this model, than it is for the diabetes data set.

If we examined the classifiers trained for each possible value of `min_samples_leaf` and not just every fifth, we can see that the optimal region is actually a bit larger:

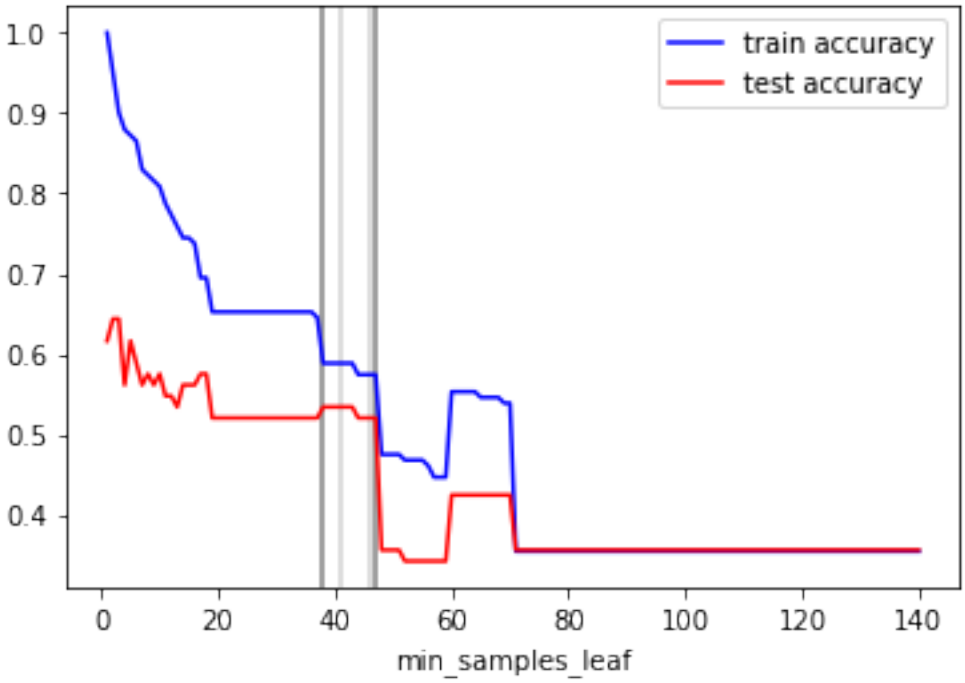
[9]:
```
accuracies_glass_2 = get_accuracies_for_min_samples_per_leaf(X_glass_train,␣
↪y_glass_train, X_glass_test, y_glass_test, step_size=1)
```

[10]:
```
accuracies_glass_2 = pd.DataFrame(accuracies_glass_2,␣
↪columns=['min_samples_leaf', 'train accuracy', 'test accuracy'])

%matplotlib inline
ax = pyplot.gca()

pyplot.axvline(x=41, color='lightgray')
pyplot.axvline(x=46, color='lightgray')
pyplot.axvline(x=38, color='gray')
pyplot.axvline(x=47, color='gray')

accuracies_glass_2.plot(kind='line', x='min_samples_leaf', y='train accuracy',␣
↪color='blue', ax=ax)
accuracies_glass_2.plot(kind='line', x='min_samples_leaf', y='test accuracy',␣
↪color='red', ax=ax)
pyplot.show()
```


train accuracy
test accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120
140
min_samples_leaf