Booklet_1_Code

August 8, 2020

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
import time
import seaborn as sns
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
import eli5
from eli5.sklearn import PermutationImportance

from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel

import category_encoders as ce
```

1 Booklet Teil 1

1.1 Aufgabe 1: Feature Engineering

[18]:
```
def filter_years(df):
    # filter Jahre 2013 and 2018 fuer unsere Gruppe
    df['Date'] = pd.to_datetime(df['Date'])
    df = df[df['Date'].dt.year.isin([2013, 2018])]
    return df

weather = pd.read_csv("weatherAUS.csv")  # lese CSV Daten und schreibe sie in␣
 ↪pandas Data Frame
weather = filter_years(weather)
weather['Date'].dt.year.value_counts()
```

[18]:
```
2018    17821
2013    16415
Name: Date, dtype: int64
```

```
[19]: # Spalten mit fehlender Zielvariable "RainTomorrow" rausschmeißen.
# Man könnte diese Auch als Testvariablen nehmen, dann wären diese aber nicht␣
↪zufällig ausgewählt...
weather = weather[weather['RainTomorrow'].notna()]
```

```
[20]: # Spalten, in denen mehr als 40% der Variablen fehlen rausscheißen.
# Zeilen, in denen mehr als 50% der Variablen fehlen rausschmeißen.

weather = weather[weather.columns[weather.isnull().mean() < 0.4]]
weather = weather.loc[weather.isnull().mean(axis=1) < 0.5]
```

1.1.1 Null Accuracy

```
[ ]: ax = sns.countplot(x="RainTomorrow", data=weather)

ax.set_xlabel("RainTomorrow", fontsize=18)
ax.set_ylabel('Count', fontsize=18)

plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

plt.savefig('distribution_target_variable.png')
```

1.1.2 Feate creation

Erstelle neue Merkmale anhand bereits bestehender Merkmale

```
[22]: weather['Year'] = weather['Date'].dt.year  # get year
weather['Month'] = weather['Date'].dt.month  # get month
weather['Day'] = weather['Date'].dt.day  # get day

weather['MinMaxDiff'] = weather['MaxTemp'] - weather['MinTemp']
weather['PressureDiff'] = weather['Pressure3pm'] - weather['Pressure9am']
weather['WindSpeedDiff'] = weather['Pressure3pm'] - weather['WindSpeed9am']
weather['HumidityDiff'] = weather['Humidity3pm'] - weather['Humidity9am']
```

1.1.3 Feature Binning

Diskretisiere bestehende Merkmale

```
[23]: def encode_season(month):
    if month >= 9 and month <= 11:
        return 'Spring'
    if month == 12 or month <= 2:
        return 'Summer'
    if month >= 3 and month <= 5:
        return 'Autumn'
```

```
    if month >= 6 and month <= 8:
        return 'Winter'

weather['Season'] = weather['Month'].apply(encode_season)
```

[24]:
```
# Ist quasy "Target Encoding". Nur halt manuell...

def encode_rainly_month(month):
    rainy_month = [5,6, 7,8,11]
    if month in rainy_month:
        return 1
    return 0

weather['RainyMonth'] = weather['Month'].apply(encode_rainly_month)
```

1.1.4 Train Test split

Wichtig: Bevor weiteres Feature Engineering betrieben wird, müssen die Daten in eine Test- und eine Trainingsmenge aufgeteilt werden, um *Test Train Leakage* zu vermeiden. Die Testmenge darf die Trainingsdaten nicht beeinflussen. Sie gilt als unbekannt.

[25]:
```
# Zunächst wird noch nicht die Zielvariable "abgespalten"
# Warum? Wenn die Zielvariable noch im gleichen DataFrame ist, kann man␣
↪leichter Outlier rausschmeißen.
train, test = train_test_split(weather, test_size=0.2, random_state = 0)
```

1.1.5 Outlier detection (Optional)

-> Verschlechtert das Ergebnis!

[ ]:
```
plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)
fig = train.boxplot(column='Rainfall')
fig.set_title('')
fig.set_ylabel('Rainfall')

plt.subplot(2, 2, 2)
fig = train.boxplot(column='WindGustSpeed')
fig.set_title('')
fig.set_ylabel('WindGustSpeed')
```

[ ]:
```
higher_lim = train['Rainfall'].quantile(0.995)
train = train[train['Rainfall'] < higher_lim]
higher_lim = train['WindGustSpeed'].quantile(0.995)
train = train[train['WindGustSpeed'] < higher_lim]
```

1.1.6 Train Test Split part 2

Hier wird jetzt die Zielvariable abgespalten

[26]:
```
X_train = train.drop(['RainTomorrow'], axis=1)
y_train = train['RainTomorrow']

X_test = test.drop(['RainTomorrow'], axis=1)
y_test = test['RainTomorrow']
```

1.1.7 Impute missing Data (Univariat)

[27]:
```
# Impute values the naiive approache without considering the locations or other␣
↪stuff like season

for dataset in [X_train, X_test]:

    colums_containing_nan = dataset.columns[dataset.isnull().any()]

    numerical_containing_nan = [col for col in colums_containing_nan if␣
↪dataset[col].dtypes != 'O']
    categorial_containing_nan = [col for col in colums_containing_nan if␣
↪dataset[col].dtypes == 'O']

    for col in numerical_containing_nan:
        col_median=X_train[col].median() #always use median from Train data !␣
↪Never impute based on Test Data ! we have to assume we dont know it.
        dataset[col] = dataset[col].fillna(col_median)

    for col in categorial_containing_nan:
        col_most_occuring = X_train[col].mode()[0]
        dataset[col] = dataset[col].fillna(col_most_occuring)
```

1.1.8 Encoding Categorial Variables

[28]:
```
X_train['RainToday'] = X_train["RainToday"].replace({'No':0, 'Yes':1})
X_test['RainToday'] = X_test["RainToday"].replace({'No':0, 'Yes':1})

y_train = y_train.replace({'No':0, 'Yes':1})
y_test = y_test.replace({'No':0, 'Yes':1})
```

1.1.9 Target encoding

Beim target Encoding kodieren wir die Variable als Einfluss auf die Zielvariable. Wenn es also in Perth zu 20% geregnet hat, dann wird Perth mit 0.2 kodiert.

[29]:
```
# Create the encoder
cat_features=['Location','WindGustDir',"WindDir9am", "WindDir3pm"]
for feature in [cat_features]:
    target_enc = ce.TargetEncoder(cols=feature)
    target_enc.fit(X_train[feature], y_train)

    # Transform the features, rename the columns with _target suffix, and join␣
↪to dataframe
    X_train = X_train.join(target_enc.transform(X_train[feature]).
↪add_suffix('_target'))
    X_test = X_test.join(target_enc.transform(X_test[feature]).
↪add_suffix('_target'))
```

1.1.10 One Hot Encoding

[30]:
```
# apply One Hot encoding

for col in ["Season"]:
    encoded_columns = pd.get_dummies(X_train[col], prefix=col, drop_first=True)
    X_train = X_train.join(encoded_columns).drop(col, axis=1)

    encoded_columns = pd.get_dummies(X_test[col], prefix=col, drop_first=True)
    X_test = X_test.join(encoded_columns).drop(col, axis=1)
```

1.1.11 Remove Features

Einige Merkmale wurde in andere Merkmale transformiert, sodass die originalen Merkmale verworfen werden können.

[31]:
```
columns_to_drop = ['Date', 'Location','WindGustDir',"WindDir9am", "WindDir3pm"]
X_train.drop(labels=columns_to_drop, axis=1, inplace=True)
X_test.drop(labels=columns_to_drop, axis=1, inplace=True)
```

1.1.12 Scale numerical features

Bemerkung: Dieser Schritt ist nicht notwendig für Entscheidungsbäume. Jedoch können diese auch mit skalierten Daten arbeiten, sodass an dieser Stelle bereits skaliert wird. Die Skalierung wird benötigt, falls eine Variablenselektion durchgeführt wird. Zudem arbeiten zum Beispiel neuronale Netze besser mit skalierten Daten.

[32]:
```
scaler = MinMaxScaler()
cols = X_train.columns

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=cols)
```

```
X_test = pd.DataFrame(np.array(X_test), columns=cols)
```

1.1.13 Feature Selection (Optional)

Bemerkung: wurde im finalen Stand nicht ausgeführt

[33]:
```
selected_columns = weather.columns
```

Univariat

[ ]:
```
# zeige correlations matrix
train = X_train
train["RainTomorrow"] = y_train.values
train.corr()
```

[ ]:
```
# zeige heatmap
plt.imshow(train.corr(), cmap='hot', interpolation='nearest')
plt.show()
```

[ ]:
```
NUM_FEATURES_TO_SELECT = 5

selector = SelectKBest(f_classif, k=NUM_FEATURES_TO_SELECT)

X_new = selector.fit_transform(X_train, y_train)

selected_features = pd.DataFrame(selector.inverse_transform(X_new),
                                 index=X_train.index,
                                 columns=X_train.columns)

selected_columns = selected_features.columns[selected_features.var() != 0]

X_train[selected_columns].head()
```

Multivariat

[ ]:
```
# Regularisation strengh is set by regularization parameter C.
# Note: the lower C, the higher the regularization
logistic = LogisticRegression(C=0.01, penalty="l1", solver='liblinear',␣
 ↪max_iter=10000, random_state=7).fit(X_train, y_train)
model = SelectFromModel(logistic, prefit=True)

X_new = model.transform(X_train)

selected_features = pd.DataFrame(model.inverse_transform(X_new),
                                 index=X_train.index,
                                 columns=X_train.columns)

# Dropped columns have values of all 0s, keep other columns
```

```
selected_columns = selected_features.columns[selected_features.var() != 0]
selected_columns
```

[ ]:
```
# getting accuracy for a logistic regression model
logistic = LogisticRegression().fit(X_train[selected_columns], y_train)

test_predictions = logistic.predict(X_test[selected_columns]).round().
 ↪astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)
```

1.2 Aufgabe 2 Entscheidungsbäume

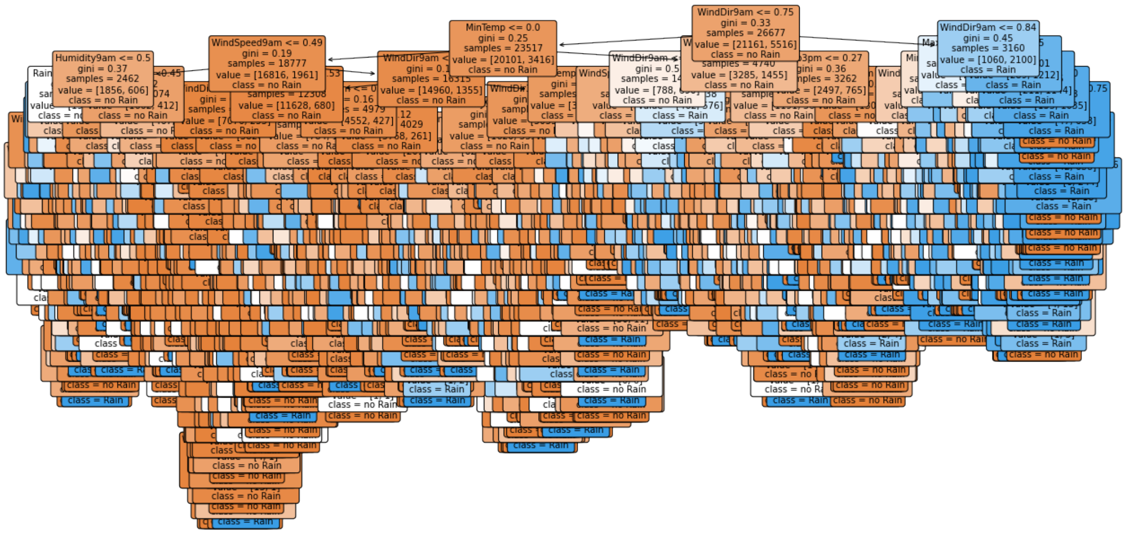
[34]:
```
# Default-Einstellungen
model = tree.DecisionTreeClassifier()
model.fit(X_train, y_train)

# Errechne Genauigkeit und Testfehler
test_predictions = model.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)
```

```
0.7860569715142429
```

[34]:
```
0.2139430284857571
```

[35]:
```
fig, ax = plt.subplots(figsize = (20,10))
tree.plot_tree(model, ax=ax,
              feature_names=selected_columns,
              filled = True,
              rounded = True,
              precision =2,
              fontsize = 10,
              class_names=["no Rain", "Rain"]
              );
```



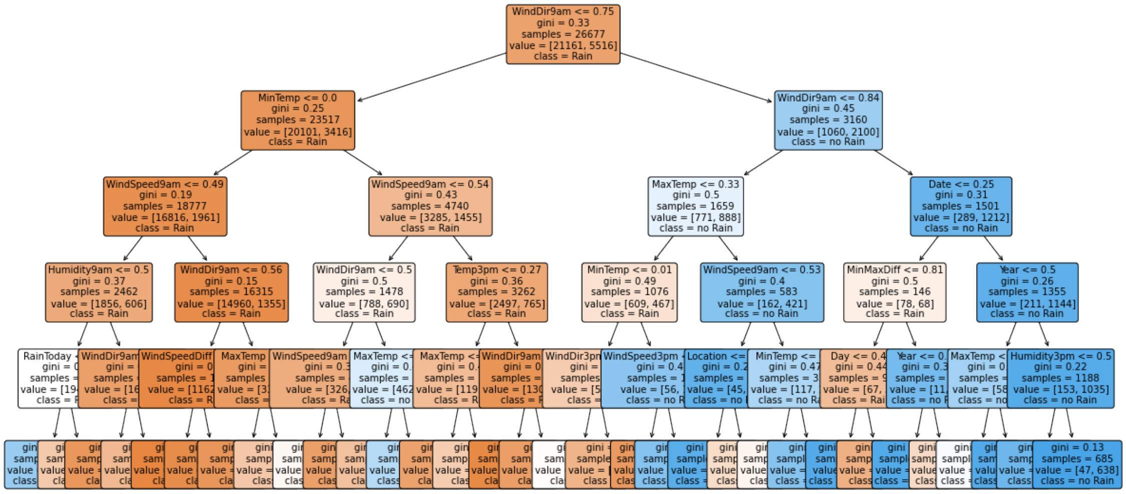
[38]:
```python
# max_depth Variation
model1 = tree.DecisionTreeClassifier(max_depth=5)
model1.fit(X_train, y_train)

test_predictions1 = model1.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions1))
mean_absolute_error(y_test, test_predictions1)
```

```
0.8446776611694153
```

[38]:
```
0.15532233883058472
```

[39]:
```python
fig, ax = plt.subplots(figsize = (20,10))
tree.plot_tree(model1, ax=ax,
               feature_names=selected_columns,
               class_names= ["Rain", "no Rain"],
               filled = True,
               rounded = True,
               precision = 2,
               fontsize = 10);
```



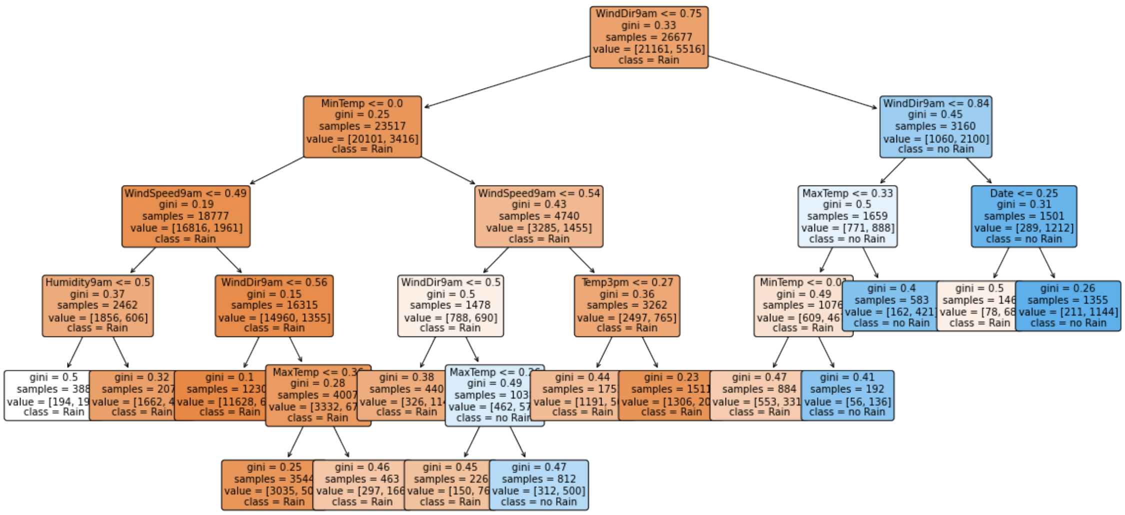
[40]:
```python
# min_impurity_increase variation
model2 = tree.DecisionTreeClassifier(min_impurity_decrease=0.001)
model2.fit(X_train, y_train)

test_predictions2 = model2.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions2))
mean_absolute_error(y_test, test_predictions2)
```

0.8452773613193403

[40]: 0.15472263868065966

[41]:
```python
fig, ax = plt.subplots(figsize = (20,10))
tree.plot_tree(model2, ax=ax,
               feature_names=selected_columns,
               class_names= ["Rain", "no Rain"],
               filled = True,
               rounded = True,
               precision = 2,
               fontsize = 10);
```



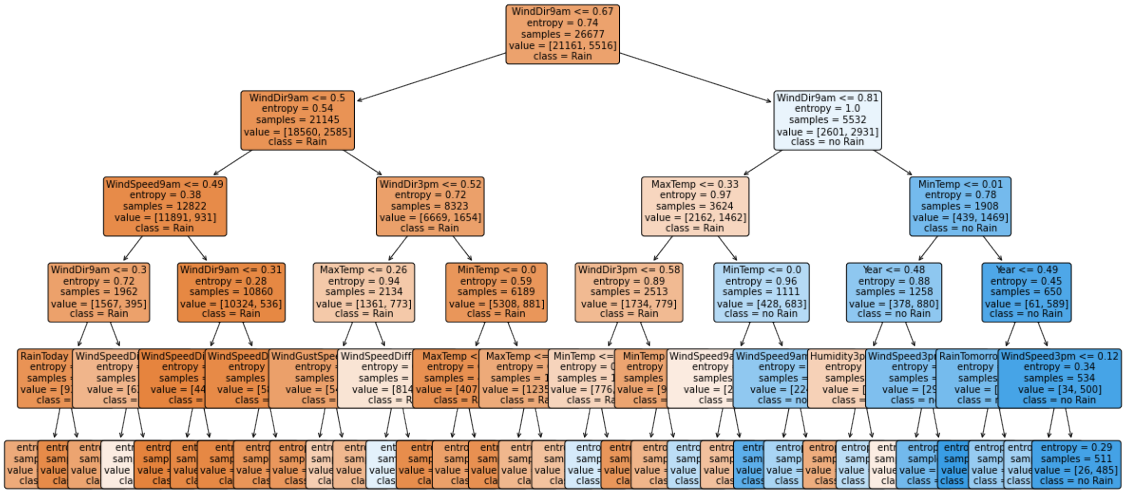

```
[42]: # criterion variation
model3 = tree.DecisionTreeClassifier(criterion="entropy", max_depth=5)
model3.fit(X_train, y_train)

test_predictions3 = model3.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions3))
mean_absolute_error(y_test, test_predictions3)
```

```
0.843928035982009
```

```
[42]: 0.156071964017991
```

```
[43]: fig, ax = plt.subplots(figsize = (20,10))
tree.plot_tree(model3, ax=ax,
               feature_names=selected_columns,
               class_names= ["Rain", "no Rain"],
               filled = True,
               rounded = True,
               precision = 2,
               fontsize = 10);
```



1.2.1 Analysis

[44]:
```
perm = PermutationImportance(model, random_state=1).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

[44]: <IPython.core.display.HTML object>

1.2.2 Cost-Complexity Pruning

Führe mit mind. 3 Bäumen unterschiedlicher Tiefe aus Aufgabenteil c) ein Minimal Cost Complexity Pruning durch. Wie verändern dich die Bäume bei Variation des Prunings? Welche Auswirkung auf die Modellgüte hat das?

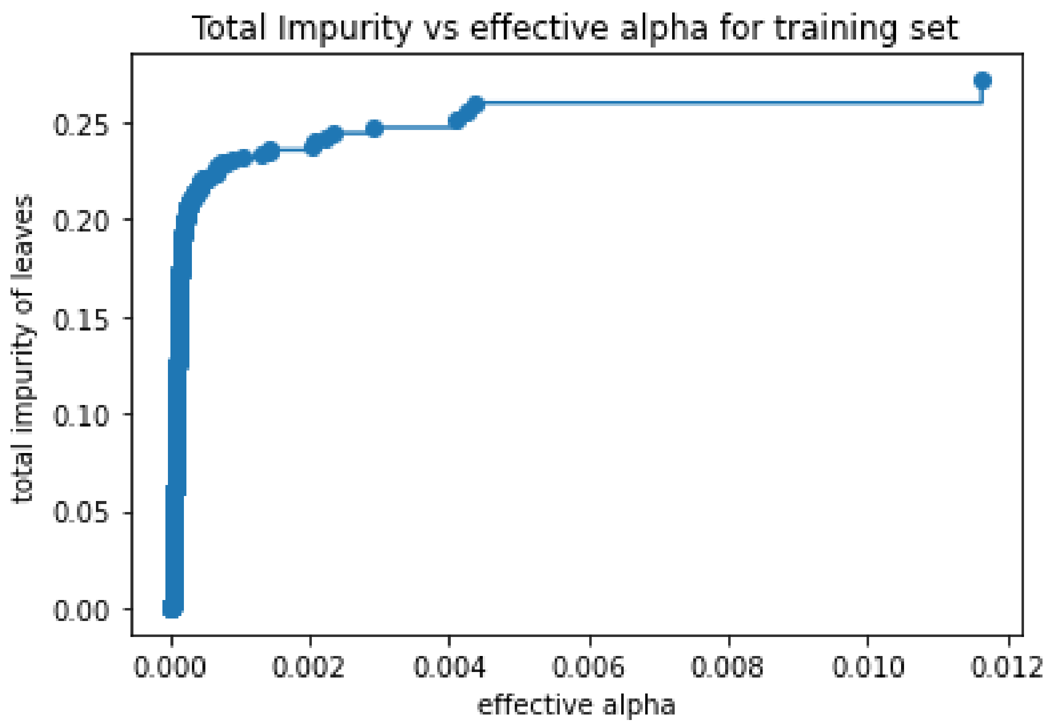
[45]:
```
model = tree.DecisionTreeClassifier()
path = model.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
ccp_alphas.size
```

[45]: 1344

[46]:
```
# nur einen Teil der Ergebnisse nutzen zur Beschleunigung
ccp_alphas_part = ccp_alphas[[0, 68, 136, 204, 272, 340, 408, 476, 544, 612,␣
 ↪680, 748, 816, 884, 952, 1020, 1088, 1156, 1224, 1292, (ccp_alphas.size-20)]]
```

[47]:
```
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
```

[47]: Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')



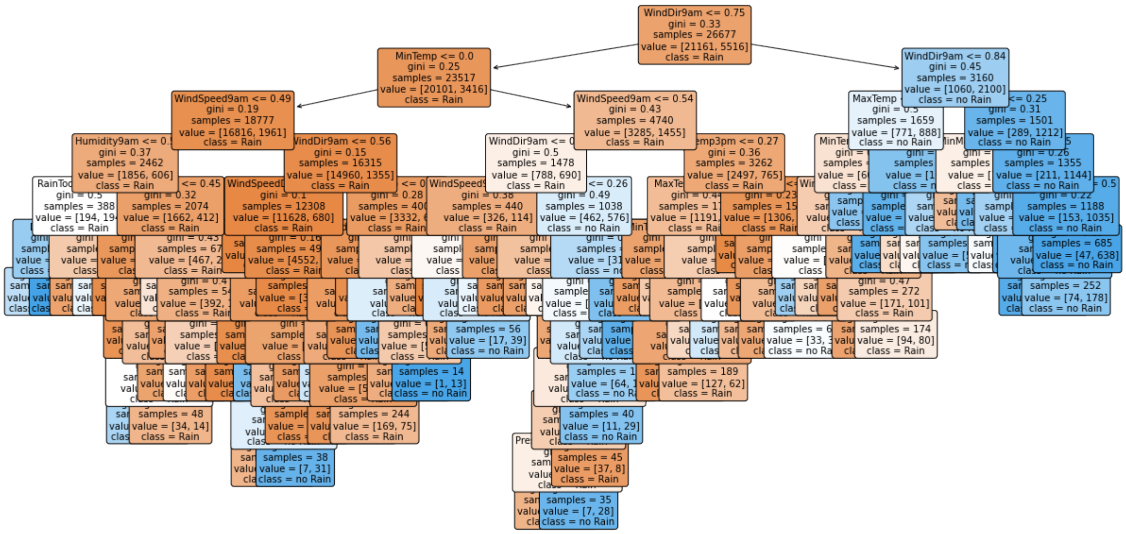
[53]:
```
model = tree.DecisionTreeClassifier(ccp_alpha=0.00025)
model.fit(X_train, y_train)

test_predictions = model.predict(X_test).round().astype(int)
print(accuracy_score(y_test, test_predictions))
mean_absolute_error(y_test, test_predictions)
```

```
0.8506746626686656
```

[53]: 0.14932533733133432

[54]:
```
fig, ax = plt.subplots(figsize = (20,10))
tree.plot_tree(model, ax=ax,
               feature_names=selected_columns,
               class_names= ["Rain", "no Rain"],
               filled = True,
               rounded = True,
               precision = 2,
               fontsize = 10);
```



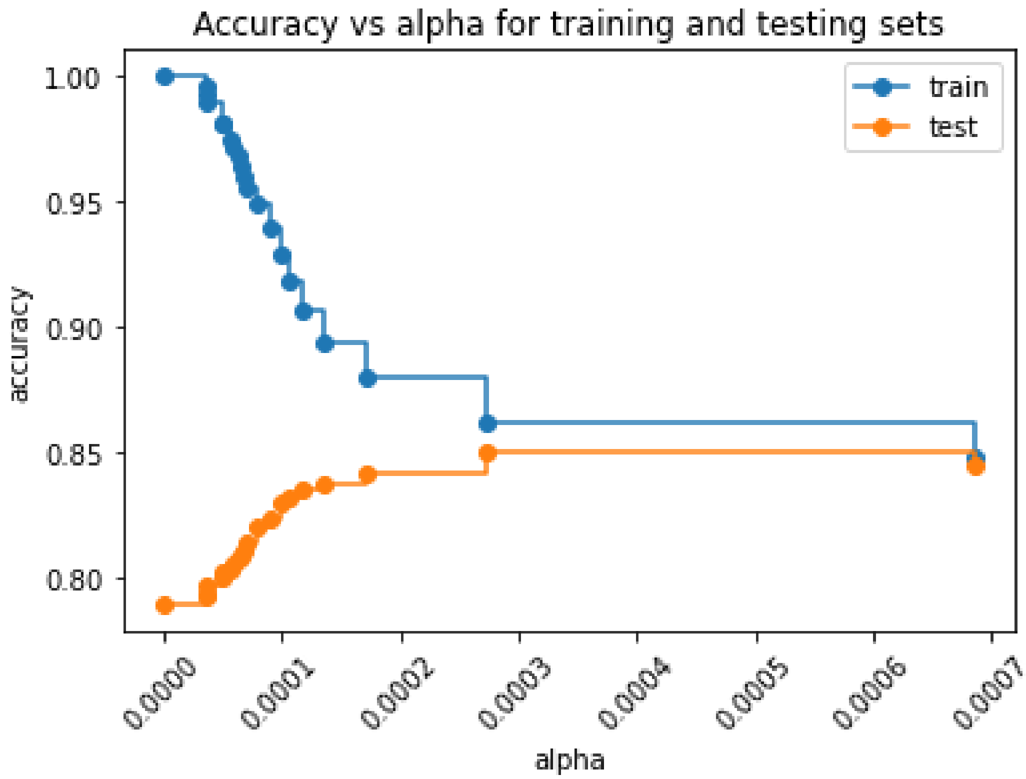
[55]:
```python
clfs = []
for ccp_alpha in ccp_alphas_part:
    clf = tree.DecisionTreeClassifier(ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of nodes in the last tree is: 39 with ccp_alpha: 0.05631994633917342
```

[56]:
```python
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas_part, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas_part, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)

plt.show()
```


Accuracy vs alpha for training and testing sets
train
test
accuracy
1.00
0.95
0.90
0.85
0.80
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
alpha