242ProjectCoding

November 22, 2021

1 Data (Preprocessing & Feature Engineering)

1.1 Import Data

[1]:
```
# import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

[2]:
```
# import raw data
weather = pd.read_csv("weatherAUS5000.csv",index_col=0)
weather.head()
```

[2]:
```
         Date Location  MinTemp  MaxTemp  Rainfall  Evaporation  Sunshine  \
0  2015-03-24  Adelaide     12.3     19.3       0.0          5.0       NaN
1  2011-07-12  Adelaide      7.9     11.4       0.0          1.0       0.5
2  2010-02-08  Adelaide     24.0     38.1       0.0         23.4      13.0
3  2016-09-19  Adelaide      6.7     16.4       0.4          NaN       NaN
4  2014-03-05  Adelaide     16.7     24.8       0.0          6.6      11.7

  WindGustDir  WindGustSpeed WindDir9am  … WindSpeed3pm  Humidity9am  \
0           S           39.0          S  …         19.0         59.0
1           N           20.0        NNE  …          7.0         70.0
2          SE           39.0        NNE  …         19.0         36.0
3           N           31.0          N  …         15.0         65.0
4           S           37.0          S  …         24.0         61.0

   Humidity3pm  Pressure9am  Pressure3pm  Cloud9am  Cloud3pm  Temp9am  \
0         47.0       1022.2       1021.4       NaN       NaN     15.1
1         59.0       1028.7       1025.7       NaN       NaN      8.4
2         24.0       1018.0       1016.0       NaN       NaN     32.4
3         40.0       1014.4       1010.0       NaN       NaN     11.2
4         48.0       1019.3       1018.9       NaN       NaN     20.8
```

```
   Temp3pm  RainTomorrow
0     17.7            No
1     11.3            No
2     37.4            No
3     15.9            No
4     23.7            No

[5 rows x 22 columns]
```

1.2 Train/Test Split

[3]:
```
# split the dependent/independent variables
X = weather.iloc[:,:-1]
Y = weather.iloc[:,-1]
```

[4]:
```
Y.isnull().sum() # no missing value in label
```

[4]:
```
0
```

[5]:
```
#explore the label classes
np.unique(Y) #binary classification
```

[5]:
```
array(['No', 'Yes'], dtype=object)
```

[6]:
```
# split train/test set
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,Y,test_size=0.
 ↪3,random_state=420) #solidify the random state
```

[7]:
```
# reset the indexes of each data set
for i in [Xtrain, Xtest, Ytrain, Ytest]:
    i.index = range(i.shape[0])
```

[8]:
```
# encode the label
#Yes->1, No->2
from sklearn.preprocessing import LabelEncoder
encorder = LabelEncoder().fit(Ytrain)
# use training set to train the encoder, and apply the encoder on both training/
 ↪testing set
Ytrain = pd.DataFrame(encorder.transform(Ytrain))
Ytest = pd.DataFrame(encorder.transform(Ytest))
```

1.3 Preprocessing 1: Outliers

[9]:
```
# use method 'describe' to explore outliers
Xtrain.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.99]).T
```

[9]:
```
                count         mean        std    min       1%       5%  \
MinTemp        3486.0    12.225645   6.396243   -6.5   -1.715    1.800
MaxTemp        3489.0    23.245543   7.201839   -3.7    8.888   12.840
Rainfall       3467.0     2.487049   7.949686    0.0    0.000    0.000
Evaporation    1983.0     5.619163   4.383098    0.0    0.400    0.800
Sunshine       1790.0     7.508659   3.805841    0.0    0.000    0.345
WindGustSpeed  3263.0    39.858413  13.219607    9.0   15.000   20.000
WindSpeed9am   3466.0    14.046163   8.670472    0.0    0.000    0.000
WindSpeed3pm   3437.0    18.553390   8.611818    0.0    2.000    6.000
Humidity9am    3459.0    69.069095  18.787698    2.0   18.000   35.000
Humidity3pm    3408.0    51.651995  20.697872    2.0    9.000   17.000
Pressure9am    3154.0  1017.622067   7.065236  985.1 1000.506 1006.100
Pressure3pm    3154.0  1015.227077   7.032531  980.2  998.000 1004.000
Cloud9am       2171.0     4.491939   2.858781    0.0    0.000    0.000
Cloud3pm       2095.0     4.603819   2.655765    0.0    0.000    0.000
Temp9am        3481.0    16.989859   6.537552   -5.2    2.400    7.000
Temp3pm        3431.0    21.719003   7.031199   -4.1    7.460   11.500

                  10%     25%     50%     75%     90%       99%     max
MinTemp           4.1     7.7    12.0    16.7    20.9    25.900    29.0
MaxTemp          14.5    18.0    22.5    28.4    33.0    40.400    46.4
Rainfall          0.0     0.0     0.0     0.8     6.6    41.272   115.8
Evaporation       1.4     2.6     4.8     7.4    10.2    20.600    56.0
Sunshine          1.4     4.6     8.3    10.6    12.0    13.300    13.9
WindGustSpeed    24.0    31.0    39.0    48.0    57.0    76.000   117.0
WindSpeed9am      4.0     7.0    13.0    19.0    26.0    37.000    65.0
WindSpeed3pm      7.0    13.0    19.0    24.0    30.0    43.000    65.0
Humidity9am      45.0    57.0    70.0    83.0    94.0   100.000   100.0
Humidity3pm      23.0    37.0    52.0    66.0    79.0    98.000   100.0
Pressure9am    1008.9  1012.8  1017.6  1022.3  1027.0  1033.247  1038.1
Pressure3pm    1006.5  1010.3  1015.2  1020.0  1024.4  1030.800  1036.0
Cloud9am          1.0     1.0     5.0     7.0     8.0     8.000     8.0
Cloud3pm          1.0     2.0     5.0     7.0     8.0     8.000     8.0
Temp9am           9.0    12.2    16.6    21.6    26.0    31.000    38.0
Temp3pm          13.3    16.6    21.0    26.6    31.4    38.600    45.9
```

[10]:
```python
# use method 'describe' to explore outliers
Xtest.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.99]).T
```

[10]:
```
                count         mean        std    min       1%       5%  \
MinTemp        1493.0    11.916812   6.375377   -8.5   -2.024    1.600
MaxTemp        1498.0    22.906809   6.986043   -0.8    9.134   13.000
Rainfall       1483.0     2.241807   7.988822    0.0    0.000    0.000
Evaporation     858.0     5.657809   4.105762    0.0    0.400    1.000
Sunshine        781.0     7.677465   3.862294    0.0    0.000    0.300
WindGustSpeed  1406.0    40.044097  14.027052    9.0   15.000   20.000
WindSpeed9am   1483.0    13.986514   9.124337    0.0    0.000    0.000
```

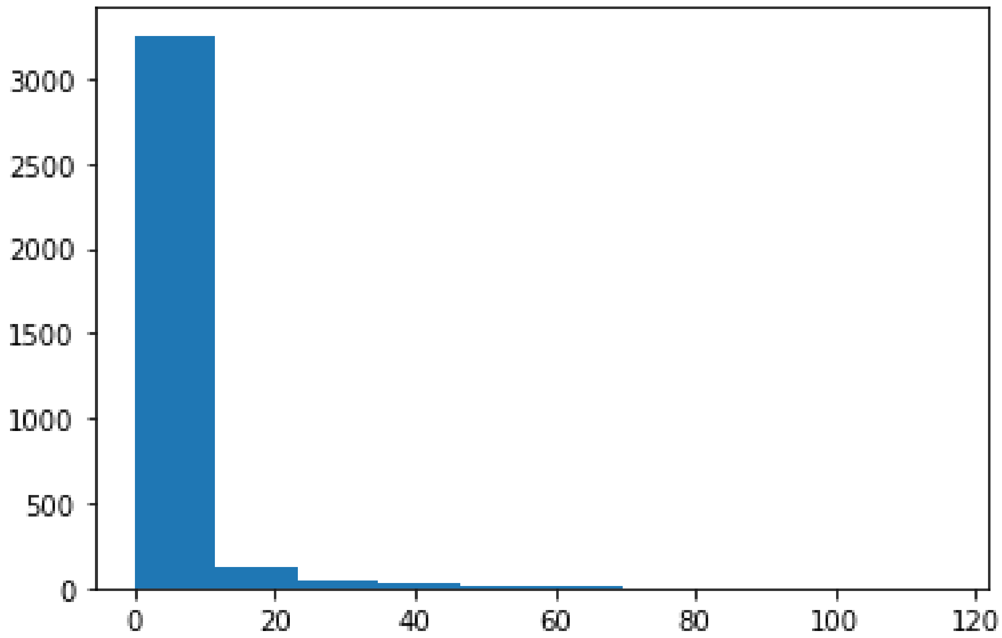
WindSpeed3pm	1482.0	18.601215	8.850446	0.0	2.000	6.000
Humidity9am	1477.0	68.688558	18.876448	4.0	20.000	36.000
Humidity3pm	1472.0	51.431386	20.459957	2.0	8.710	18.000
Pressure9am	1352.0	1017.763536	6.910275	988.5	1000.900	1006.255
Pressure3pm	1350.0	1015.397926	6.916976	986.2	999.198	1003.900
Cloud9am	940.0	4.494681	2.870468	0.0	0.000	0.000
Cloud3pm	917.0	4.403490	2.731969	0.0	0.000	0.000
Temp9am	1486.0	16.751817	6.339816	-5.3	2.370	6.725
Temp3pm	1481.0	21.483660	6.770567	-1.2	8.540	11.800

	10%	25%	50%	75%	90%	99%	max
MinTemp	3.70	7.3	11.8	16.5	20.48	25.316	28.3
MaxTemp	14.50	17.8	22.4	27.8	32.60	38.303	45.1
Rainfall	0.00	0.0	0.0	0.8	5.20	35.372	108.2
Evaporation	1.60	2.8	4.8	7.6	10.40	19.458	38.8
Sunshine	1.50	4.7	8.6	10.7	12.20	13.400	13.9
WindGustSpeed	24.00	30.0	39.0	48.0	57.00	78.000	122.0
WindSpeed9am	4.00	7.0	13.0	20.0	26.00	39.360	72.0
WindSpeed3pm	7.00	13.0	19.0	24.0	31.00	43.000	56.0
Humidity9am	44.00	57.0	69.0	82.0	95.00	100.000	100.0
Humidity3pm	23.00	37.0	52.0	66.0	78.00	96.290	100.0
Pressure9am	1008.61	1013.2	1017.8	1022.3	1026.50	1033.449	1038.2
Pressure3pm	1006.49	1010.9	1015.4	1020.0	1024.20	1031.151	1036.9
Cloud9am	1.00	1.0	5.0	7.0	8.00	8.000	8.0
Cloud3pm	1.00	2.0	5.0	7.0	8.00	8.000	8.0
Temp9am	9.00	12.1	16.5	21.3	25.45	30.200	35.1
Temp3pm	13.30	16.5	20.9	26.2	30.90	37.400	42.9

Conclusion: No outlier in the dataset

1.4 Feature Engineering 1: Feature Creation-"Rainfall"

[11]:
```
plt.hist(Xtrain.Rainfall)
plt.show()
```



[12]:
```
#33 missing values
Xtrain["Rainfall"].isnull().sum()
```

[12]: 33

[13]:
```
# create a new col "RainToday"
Xtrain.loc[Xtrain["Rainfall"] >= 1,"RainToday"] = "Yes"
Xtrain.loc[Xtrain["Rainfall"] < 1,"RainToday"] = "No"
Xtrain.loc[Xtrain["Rainfall"] == np.nan,"RainToday"] = np.nan

# operate similarly on test set
Xtest.loc[Xtest["Rainfall"] >= 1,"RainToday"] = "Yes"
Xtest.loc[Xtest["Rainfall"] < 1,"RainToday"] = "No"
Xtest.loc[Xtest["Rainfall"] == np.nan,"RainToday"] = np.nan
```

[14]:
```
Xtrain.loc[:,"RainToday"].value_counts()
```

[14]:
```
No     2642
Yes     825
Name: RainToday, dtype: int64
```

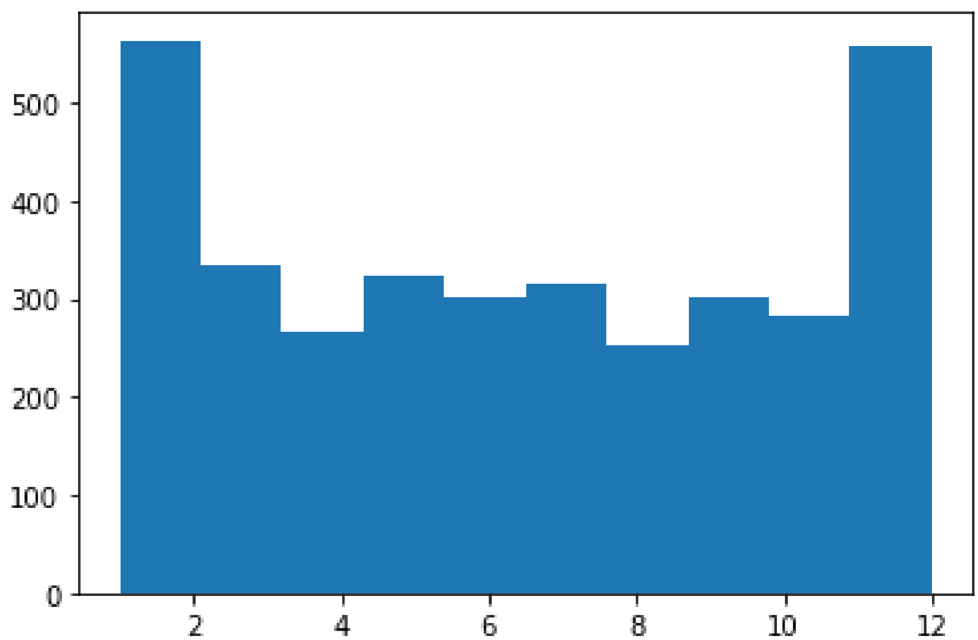
1.5 Feature Engineering 2: Feature Extraction-“Month”

[15]:
```
# we cannot treat it as categorical variables because there are 2141 difference classes
Xtrain.iloc[:,0].value_counts().count()
```

[15]:
```
2141
```

[16]:
```
# extract the month
Xtrain["Date"] = Xtrain["Date"].apply(lambda x:int(x.split("-")[1]))
# rename the column
Xtrain = Xtrain.rename(columns={"Date":"Month"})
```

[17]:
```
plt.hist(Xtrain.loc[:,"Month"])
plt.show()
```



[18]:
```
# operate similarly on test set
Xtest["Date"] = Xtest["Date"].apply(lambda x:int(x.split("-")[1]))
Xtest = Xtest.rename(columns={"Date":"Month"})
```

1.6 Feature Engineering 3: Feature Creation-“Climate”

[19]:
```
# 49 distinct cities in training set, too many for one-hot encoding
Xtrain.loc[:,"Location"].value_counts().count()
```

[19]:
```
49
```

1.6.1 web crawler

[20]:
```
# web crawler

# before running this code, a list, cityname, should be predefined

# import time
# from selenium import webdriver
# import pandas as pd
# import numpy as np

# df = pd.DataFrame(index = range(len(cityname)))
# driver = webdriver.Chrome()
# time0 = time.time() # tik

# for num, city in enumerate(cityname):    # go through all the cities in␣
 ↪cityname
#     driver.get('https://google.com')    # open google first
#     time.sleep(0.3)   # stop for 0.3 second
#     serach_box = driver.find_element_by_name('q') # google search box
#     search_box.send_keys('%s Australia latitude and longitude' %city)   #␣
 ↪input this sentence in the google search box
#     search_box.submit()     # press enter
#     result = driver.find_element_by_xpath('//div[@class="Z0Lcw"]').text  #␣
 ↪get the result (it may vary from computer to computer)
#     resultsplit = result.split(' ')  # split the result
#     df.loc[num, 'City'] = city # the first col is city
#     df.loc[num, 'Latitude'] = resultsplit[0] # latitude
#     df.loc[num, 'Longitude'] = resultsplit[2] # longitude
#     df.loc[num, 'Latitudedir'] = resultsplit[1] # latitude direction
#     df.loc[num, 'Longitudedir'] = resultsplit[3] # longitude direction
#     print('%i webcrawler successful for city %s' %(num, city))   # monitor␣
 ↪the progress

# time.sleep(1) # stop for 1 second
# driver.quit() # close the chrome
# print(time.time() - time0) # print the total running time
```

1.6.2 Display the result of Web Crawler

[21]:
```
# import the result of web crawler
cityll = pd.read_csv("cityll.csv",index_col=0)
city_climate = pd.read_csv("Cityclimate.csv")
```

[22]:
```
# the latitude and longitude of the cities listing on the climate graph made by␣
 ↪Australian Bureau of Meteorology
cityll.head()
```

[22]:
```
           City  Latitude  Longitude Latitudedir Longitudedir
0      Adelaide  34.9285°  138.6007°          S,            E
1        Albany  35.0275°  117.8840°          S,            E
2        Albury  36.0737°  146.9135°          S,            E
3       Wodonga  36.1241°  146.8818°          S,            E
4  AliceSprings  23.6980°  133.8807°          S,            E
```

[23]:
```
# the climate of the cities made by Australian Bureau of Meteorology
city_climate.head()
```

[23]:
```
           City                      Climate
0      Adelaide             Warm temperate
1        Albany             Mild temperate
2        Albury  Hot dry summer, cool winter
3       Wodonga  Hot dry summer, cool winter
4  AliceSprings  Hot dry summer, warm winter
```

1.6.3 Combine cityll & city_climate

[24]:
```
# remove the degree sign
cityll["Latitudenum"] = cityll["Latitude"].apply(lambda x:float(x[:-1]))
cityll["Longitudenum"] = cityll["Longitude"].apply(lambda x:float(x[:-1]))
# all the cities are in the Eastern Hemisphere and Southern Hemisphere
citylld = cityll.iloc[:,[0,5,6]]
citylld
```

[24]:
```
            City  Latitudenum  Longitudenum
0       Adelaide      34.9285      138.6007
1         Albany      35.0275      117.8840
2         Albury      36.0737      146.9135
3        Wodonga      36.1241      146.8818
4   AliceSprings      23.6980      133.8807
..           …            …             …
95    Wollongong      34.4278      150.8931
96       Wyndham      15.4825      128.1228
97        Yalgoo      28.3445      116.6851
98        Yulara      25.2335      130.9849
99         Uluru      25.3444      131.0369

[100 rows x 3 columns]
```

[25]:
```
# the climate of the cities made by Australian Bureau of Meteorology
city_climate.head()
```

[25]:
```
           City                      Climate
0      Adelaide             Warm temperate
1        Albany             Mild temperate
```

```
2        Albury  Hot dry summer, cool winter
3       Wodonga  Hot dry summer, cool winter
4  AliceSprings  Hot dry summer, warm winter
```

[26]:
```
# add column "climate" to citylld
citylld["climate"] = city_climate.iloc[:,-1]
citylld.head()
```

[26]:
```
           City  Latitudenum  Longitudenum                      climate
0      Adelaide      34.9285      138.6007            Warm temperate
1        Albany      35.0275      117.8840            Mild temperate
2        Albury      36.0737      146.9135  Hot dry summer, cool winter
3       Wodonga      36.1241      146.8818  Hot dry summer, cool winter
4  AliceSprings      23.6980      133.8807  Hot dry summer, warm winter
```

1.6.4 Calculate the climate of cities in training/testing set (sampelecities)

[27]:
```
# samplecity stores the latitude and longitude of cities listed in the training/
↪testing set
samplecity = pd.read_csv("samplecity.csv",index_col=0)
samplecity.head()
```

[27]:
```
       City  Latitude  Longitude Latitudedir Longitudedir
0  Canberra  35.2809°  149.1300°          S,            E
1    Sydney  33.8688°  151.2093°          S,            E
2     Perth  31.9505°  115.8605°          S,            E
3    Darwin  12.4634°  130.8456°          S,            E
4    Hobart  42.8821°  147.3272°          S,            E
```

[28]:
```
# operate it similarly
samplecity["Latitudenum"] = samplecity["Latitude"].apply(lambda x:float(x[:-1]))
samplecity["Longitudenum"] = samplecity["Longitude"].apply(lambda x:float(x[:
↪-1]))
samplecityd = samplecity.iloc[:,[0,5,6]]
samplecityd.head()
```

[28]:
```
       City  Latitudenum  Longitudenum
0  Canberra      35.2809      149.1300
1    Sydney      33.8688      151.2093
2     Perth      31.9505      115.8605
3    Darwin      12.4634      130.8456
4    Hobart      42.8821      147.3272
```

[29]:
```
# calulate the distance
# convert the angle to radian
from math import radians, sin, cos, acos
citylld.loc[:,"slat"] = citylld.iloc[:,1].apply(lambda x : radians(x))
```

```
citylld.loc[:,"slon"] = citylld.iloc[:,2].apply(lambda x : radians(x))
samplecityd.loc[:,"elat"] = samplecityd.iloc[:,1].apply(lambda x : radians(x))
samplecityd.loc[:,"elon"] = samplecityd.iloc[:,2].apply(lambda x : radians(x))

# add the climate of sample cities
import sys
for i in range(samplecityd.shape[0]):
    slat = citylld.loc[:,"slat"]
    slon = citylld.loc[:,"slon"]
    elat = samplecityd.loc[i,"elat"]
    elon = samplecityd.loc[i,"elon"]
    dist = 6371.01 * np.arccos(np.sin(slat)*np.sin(elat) +
                     np.cos(slat)*np.cos(elat)*np.cos(slon.values - elon))
    city_index = np.argsort(dist)[0]
    # use the climate of the nearest city as its climate
    samplecityd.loc[i,"closest_city"] = citylld.loc[city_index,"City"]
    samplecityd.loc[i,"climate"] = citylld.loc[city_index,"climate"]
```

[30]:
```
samplecityd.head(5)
```

[30]:
```
       City  Latitudenum  Longitudenum      elat      elon closest_city  \
0  Canberra      35.2809      149.1300  0.615768  2.602810     Canberra
1    Sydney      33.8688      151.2093  0.591122  2.639100       Sydney
2     Perth      31.9505      115.8605  0.557641  2.022147        Perth
3    Darwin      12.4634      130.8456  0.217527  2.283687       Darwin
4    Hobart      42.8821      147.3272  0.748434  2.571345       Hobart

                            climate
0                    Cool temperate
1                    Warm temperate
2                    Warm temperate
3  High humidity summer, warm winter
4                    Cool temperate
```

[31]:
```
# solidify the result in a new dataframe
locafinal = samplecityd.iloc[:,[0,-1]]
locafinal.columns = ["Location","Climate"]
locafinal = locafinal.set_index(keys="Location")   # set location as its index
locafinal.head()
```

[31]:
```
                                    Climate
Location
Canberra                     Cool temperate
Sydney                       Warm temperate
Perth                        Warm temperate
Darwin    High humidity summer, warm winter
Hobart                       Cool temperate
```

[32]:
```python
# save it
locafinal.to_csv("samplelocation.csv")
```

1.6.5 Replace "location" with "Climate"

[33]:
```python
# use the climate to represent the specific location because the distinct␣
↪number of climate is much fewer than locations

import re

#replace location with climate, and then regulize the string
Xtrain["Location"] = Xtrain["Location"].map(locafinal.iloc[:,0]).apply(lambda x:
↪re.sub(",","",x.strip()))
Xtest["Location"] = Xtest["Location"].map(locafinal.iloc[:,0]).apply(lambda x:
↪re.sub(",","",x.strip()))

# rename
Xtrain = Xtrain.rename(columns={"Location":"Climate"})
Xtest = Xtest.rename(columns={"Location":"Climate"})
```

[34]:
```python
Xtrain.head(3)
```

[34]:
```
   Month                         Climate  MinTemp  MaxTemp  Rainfall  \
0      8  High humidity summer warm winter     17.5     36.0       0.0
1     12                  Cool temperate      9.5     25.0       0.0
2      4                  Mild temperate     13.0     22.6       0.0

   Evaporation  Sunshine WindGustDir  WindGustSpeed WindDir9am  …  \
0          8.8       NaN         ESE           26.0        NNW  …
1          NaN       NaN         NNW           33.0         NE  …
2          3.8      10.4         NaN            NaN         NE  …

  WindSpeed3pm  Humidity9am  Humidity3pm  Pressure9am  Pressure3pm  Cloud9am  \
0         15.0         57.0          NaN       1016.8       1012.2       0.0
1         17.0         59.0         31.0       1020.4       1017.5       NaN
2         31.0         79.0         68.0       1020.3       1015.7       1.0

   Cloud3pm  Temp9am  Temp3pm  RainToday
0       NaN     27.5      NaN         No
1       NaN     14.6     23.6         No
2       3.0     17.5     20.8         No

[3 rows x 22 columns]
```

1.7 Preprocessing 2: Missing Values

[35]:
```
# look up for the missing values
Xtrain.isnull().mean()
```

[35]:
```
Month            0.000000
Climate          0.000000
MinTemp          0.004000
MaxTemp          0.003143
Rainfall         0.009429
Evaporation      0.433429
Sunshine         0.488571
WindGustDir      0.067714
WindGustSpeed    0.067714
WindDir9am       0.067429
WindDir3pm       0.024286
WindSpeed9am     0.009714
WindSpeed3pm     0.018000
Humidity9am      0.011714
Humidity3pm      0.026286
Pressure9am      0.098857
Pressure3pm      0.098857
Cloud9am         0.379714
Cloud3pm         0.401429
Temp9am          0.005429
Temp3pm          0.019714
RainToday        0.009429
dtype: float64
```

[36]:
```
Xtrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3500 entries, 0 to 3499
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Month           3500 non-null   int64
 1   Climate         3500 non-null   object
 2   MinTemp         3486 non-null   float64
 3   MaxTemp         3489 non-null   float64
 4   Rainfall        3467 non-null   float64
 5   Evaporation     1983 non-null   float64
 6   Sunshine        1790 non-null   float64
 7   WindGustDir     3263 non-null   object
 8   WindGustSpeed   3263 non-null   float64
 9   WindDir9am      3264 non-null   object
 10  WindDir3pm      3415 non-null   object
 11  WindSpeed9am    3466 non-null   float64
```

```
 12  WindSpeed3pm   3437 non-null   float64
 13  Humidity9am    3459 non-null   float64
 14  Humidity3pm    3408 non-null   float64
 15  Pressure9am    3154 non-null   float64
 16  Pressure3pm    3154 non-null   float64
 17  Cloud9am       2171 non-null   float64
 18  Cloud3pm       2095 non-null   float64
 19  Temp9am        3481 non-null   float64
 20  Temp3pm        3431 non-null   float64
 21  RainToday      3467 non-null   object
dtypes: float64(16), int64(1), object(5)
memory usage: 601.7+ KB
```

1.8 Preprocessing 2.1: Missing Values in Categorical Variables

[37]:
```
# missing values in categorical variables
cate = Xtrain.columns[Xtrain.dtypes == "object"].tolist()
#"Cloud9am","Cloud3pm" is also categorical variable indeed because it only␣
↪takes integer 0-8 to represent the cloud
cloud = ["Cloud9am","Cloud3pm"]
cate = cate + cloud

# use mode to fill in the missing values on categorical variables
from sklearn.impute import SimpleImputer

si = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
# use the training set to fit the imputer
si.fit(Xtrain.loc[:,cate])
# apply the imputer to training/testing set
Xtrain.loc[:,cate] = si.transform(Xtrain.loc[:,cate])
Xtest.loc[:,cate] = si.transform(Xtest.loc[:,cate])
```

[38]:
```
# make sure there is no missing value in categorical variables
Xtrain.loc[:,cate].isnull().mean()
```

[38]:
```
Climate        0.0
WindGustDir    0.0
WindDir9am     0.0
WindDir3pm     0.0
RainToday      0.0
Cloud9am       0.0
Cloud3pm       0.0
dtype: float64
```

[39]:
```
# make sure there is no missing value in categorical variables
Xtest.loc[:,cate].isnull().mean()
```

[39]:
```
Climate        0.0
WindGustDir    0.0
WindDir9am     0.0
WindDir3pm     0.0
RainToday      0.0
Cloud9am       0.0
Cloud3pm       0.0
dtype: float64
```

1.9 Preprocessing 2.2: Missing Values in Numerical Variables

[40]:
```python
col = [x for x in Xtrain.columns if x not in cate]

# use mean to impute the numerical variables
impmean = SimpleImputer(missing_values=np.nan,strategy = "mean")
impmean = impmean.fit(Xtrain.loc[:,col])
Xtrain.loc[:,col] = impmean.transform(Xtrain.loc[:,col])
Xtest.loc[:,col] = impmean.transform(Xtest.loc[:,col])
```

[41]:
```python
# no missing value in training/testing set
Xtrain.isnull().mean()
Xtest.isnull().mean()
```

[41]:
```
Month            0.0
Climate          0.0
MinTemp          0.0
MaxTemp          0.0
Rainfall         0.0
Evaporation      0.0
Sunshine         0.0
WindGustDir      0.0
WindGustSpeed    0.0
WindDir9am       0.0
WindDir3pm       0.0
WindSpeed9am     0.0
WindSpeed3pm     0.0
Humidity9am      0.0
Humidity3pm      0.0
Pressure9am      0.0
Pressure3pm      0.0
Cloud9am         0.0
Cloud3pm         0.0
Temp9am          0.0
Temp3pm          0.0
RainToday        0.0
dtype: float64
```

1.10 Preprocessing 3: Encoding Categorical Variables

[42]:
```
# encode the categorical variables
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
oe = oe.fit(Xtrain.loc[:,cate])
Xtrain.loc[:,cate] = oe.transform(Xtrain.loc[:,cate])
Xtest.loc[:,cate] = oe.transform(Xtest.loc[:,cate])
```

1.11 Preprocessing 4: Normalization

[43]:
```
# use StandarScaler to accelerate the convergence especially in unit-sensitive␣
↪models like SVM
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
col.remove("Month")
ss = ss.fit(Xtrain.loc[:,col])
Xtrain.loc[:,col] = ss.transform(Xtrain.loc[:,col])
Xtest.loc[:,col] = ss.transform(Xtest.loc[:,col])
```

1.12 Now we have finished all the preprocessing and feature engineering part

[44]:
```
Xtrain.head()
```

[44]:
```
   Month  Climate   MinTemp   MaxTemp  Rainfall  Evaporation  Sunshine  \
0    8.0      1.0  0.826375  1.774044 -0.314379     0.964367  0.000000
1   12.0      0.0 -0.427048  0.244031 -0.314379     0.000000  0.000000
2    4.0      4.0  0.121324 -0.089790 -0.314379    -0.551534  1.062619
3   11.0      4.0  0.262334  0.911673 -0.314379     0.054826 -0.885225
4    4.0      2.0 -0.975421  0.035393 -0.314379    -0.854715  0.401087

   WindGustDir  WindGustSpeed  WindDir9am  …  WindSpeed3pm  Humidity9am  \
0          2.0  -1.085893e+00         6.0  …     -0.416443    -0.646283
1          6.0  -5.373993e-01         4.0  …     -0.182051    -0.539186
2         13.0  -1.113509e-15         4.0  …      1.458692     0.531786
3          8.0  -2.239744e-01         3.0  …      1.107105     0.692432
4          5.0  -1.242605e+00         0.0  …     -0.416443    -0.592734

   Humidity3pm  Pressure9am  Pressure3pm  Cloud9am  Cloud3pm   Temp9am  \
0     0.000000    -0.122589    -0.453507       0.0       7.0  1.612270
1    -1.011310     0.414254     0.340522       7.0       7.0 -0.366608
2     0.800547     0.399342     0.070852       1.0       3.0  0.078256
3    -0.374711    -0.763819    -1.397352       6.0       6.0  0.231658
4    -0.815433     0.324780    -0.168855       2.0       4.0 -0.704091

    Temp3pm  RainToday
0  0.000000        0.0
```

```
1  0.270238        0.0
2 -0.132031        0.0
3  0.830540        0.0
4  0.097837        0.0

[5 rows x 22 columns]
```

[45]:
```
Xtrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3500 entries, 0 to 3499
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Month          3500 non-null   float64
 1   Climate        3500 non-null   float64
 2   MinTemp        3500 non-null   float64
 3   MaxTemp        3500 non-null   float64
 4   Rainfall       3500 non-null   float64
 5   Evaporation    3500 non-null   float64
 6   Sunshine       3500 non-null   float64
 7   WindGustDir    3500 non-null   float64
 8   WindGustSpeed  3500 non-null   float64
 9   WindDir9am     3500 non-null   float64
 10  WindDir3pm     3500 non-null   float64
 11  WindSpeed9am   3500 non-null   float64
 12  WindSpeed3pm   3500 non-null   float64
 13  Humidity9am    3500 non-null   float64
 14  Humidity3pm    3500 non-null   float64
 15  Pressure9am    3500 non-null   float64
 16  Pressure3pm    3500 non-null   float64
 17  Cloud9am       3500 non-null   float64
 18  Cloud3pm       3500 non-null   float64
 19  Temp9am        3500 non-null   float64
 20  Temp3pm        3500 non-null   float64
 21  RainToday      3500 non-null   float64
dtypes: float64(22)
memory usage: 601.7 KB
```

[46]:
```
Xtest.head()
```

[46]:
```
   Month  Climate   MinTemp   MaxTemp  Rainfall  Evaporation  Sunshine  \
0    1.0      0.0  1.531425  0.633489  2.871067     0.000000  0.000000
1    3.0      4.0 -0.035354 -0.646158 -0.036285    -0.794079  0.107073
2    3.0      0.0 -0.489720 -1.383346  0.000000     0.000000  0.000000
3   10.0      6.0  0.136992 -0.409702 -0.314379     0.000000  0.000000
4   11.0      4.0 -0.004018 -0.451429 -0.263817     0.000000  0.000000
```

```
   WindGustDir  WindGustSpeed  WindDir9am  …  WindSpeed3pm  Humidity9am  \
0         11.0       1.343150         8.0  …  2.161868e+00     1.174369
1         12.0       0.951369        12.0  …  1.107105e+00     1.013723
2          4.0       0.089450         3.0  … -4.163637e-16     0.000000
3         12.0      -0.537399        13.0  …  6.383207e-01    -1.556609
4          0.0      -0.537399        12.0  …  5.234093e-02     1.227917

   Humidity3pm  Pressure9am  Pressure3pm  Cloud9am  Cloud3pm   Temp9am  \
0     1.681991    -1.643646    -1.067755       7.0       7.0  1.412848
1     0.506733     0.384430     0.700082       8.0       7.0 -0.335927
2     0.000000     0.000000     0.000000       7.0       7.0  0.000000
3    -0.031928     0.548465     0.640155       7.0       7.0 -0.029125
4     0.849516    -0.301537    -0.303690       8.0       4.0 -0.520009

    Temp3pm  RainToday
0  0.198404        1.0
1 -0.606132        1.0
2  0.000000        0.0
3 -0.304431        0.0
4 -0.390632        0.0

[5 rows x 22 columns]
```

[47]:
```
Xtest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Month          1500 non-null   float64
 1   Climate        1500 non-null   float64
 2   MinTemp        1500 non-null   float64
 3   MaxTemp        1500 non-null   float64
 4   Rainfall       1500 non-null   float64
 5   Evaporation    1500 non-null   float64
 6   Sunshine       1500 non-null   float64
 7   WindGustDir    1500 non-null   float64
 8   WindGustSpeed  1500 non-null   float64
 9   WindDir9am     1500 non-null   float64
 10  WindDir3pm     1500 non-null   float64
 11  WindSpeed9am   1500 non-null   float64
 12  WindSpeed3pm   1500 non-null   float64
 13  Humidity9am    1500 non-null   float64
 14  Humidity3pm    1500 non-null   float64
 15  Pressure9am    1500 non-null   float64
 16  Pressure3pm    1500 non-null   float64
 17  Cloud9am       1500 non-null   float64
```

```
 18  Cloud3pm       1500 non-null   float64
 19  Temp9am        1500 non-null   float64
 20  Temp3pm        1500 non-null   float64
 21  RainToday      1500 non-null   float64
dtypes: float64(22)
memory usage: 257.9 KB
```

2 Analytics Models

Result: We applied 9 models to this dataset

- 'XGBoost' is the most powerful model with the highest accuracy
- 'SVM-Poly Kernel' has the advantage of balancing accuracy and time consumption

Model	Accuracy	Time(s)
Base Model	0.771	0
SVM-Linear Kernel	0.844	480
SVM-Poly Kernel	0.851	25
SVM-RBF Kernel	0.842	25
Logistic Regression	0.842	22
CART	0.824	22
Random Forest	0.852	182
Gradient Boosting Tree	0.852	284
XGBoost	0.858	681

[48]:
```
Ytrain = Ytrain.iloc[:,0].ravel()
Ytest = Ytest.iloc[:,0].ravel()
```

[49]:
```
from time import time
time_start = time()
```

2.1 Base Model

[50]:
```
sum(Ytest == 0)/len(Ytest)
```

[50]: 0.7713333333333333

2.2 SVM

[51]:
```
from time import time
import datetime
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score, recall_score
```

2.2.1 select kernel

[52]:
```
times = time()
for kernel in ["linear","poly","rbf","sigmoid"]:
    clf = SVC(kernel = kernel
              ,gamma="auto"
              ,degree = 1
              ,cache_size = 5000
             ).fit(Xtrain, Ytrain)
    result = clf.predict(Xtest)
    score = clf.score(Xtest,Ytest)
    recall = recall_score(Ytest, result)
    auc = roc_auc_score(Ytest,clf.decision_function(Xtest))
    print("\n%s 's testing accuracy %f, recall is %f', auc is %f" %␣
 ↪(kernel,score,recall,auc))
    print(datetime.datetime.fromtimestamp(time()-times).strftime("%M:%S:%f"))
```

```
linear 's testing accuracy 0.844000, recall is 0.469388', auc is 0.869029
00:03:514385

poly 's testing accuracy 0.840667, recall is 0.457726', auc is 0.868157
00:04:458924

rbf 's testing accuracy 0.813333, recall is 0.306122', auc is 0.814873
00:07:794253

sigmoid 's testing accuracy 0.655333, recall is 0.154519', auc is 0.437308
00:08:767875
```

Conclusion: sigmoid kernel should be exclued.

2.3 SVM-Linear Kernel

[53]:
```
# 4 min
import time
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

grid_values = {'C': np.linspace(0.01,20,20)}

tic = time.time()
clf = SVC(kernel = 'linear')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
```

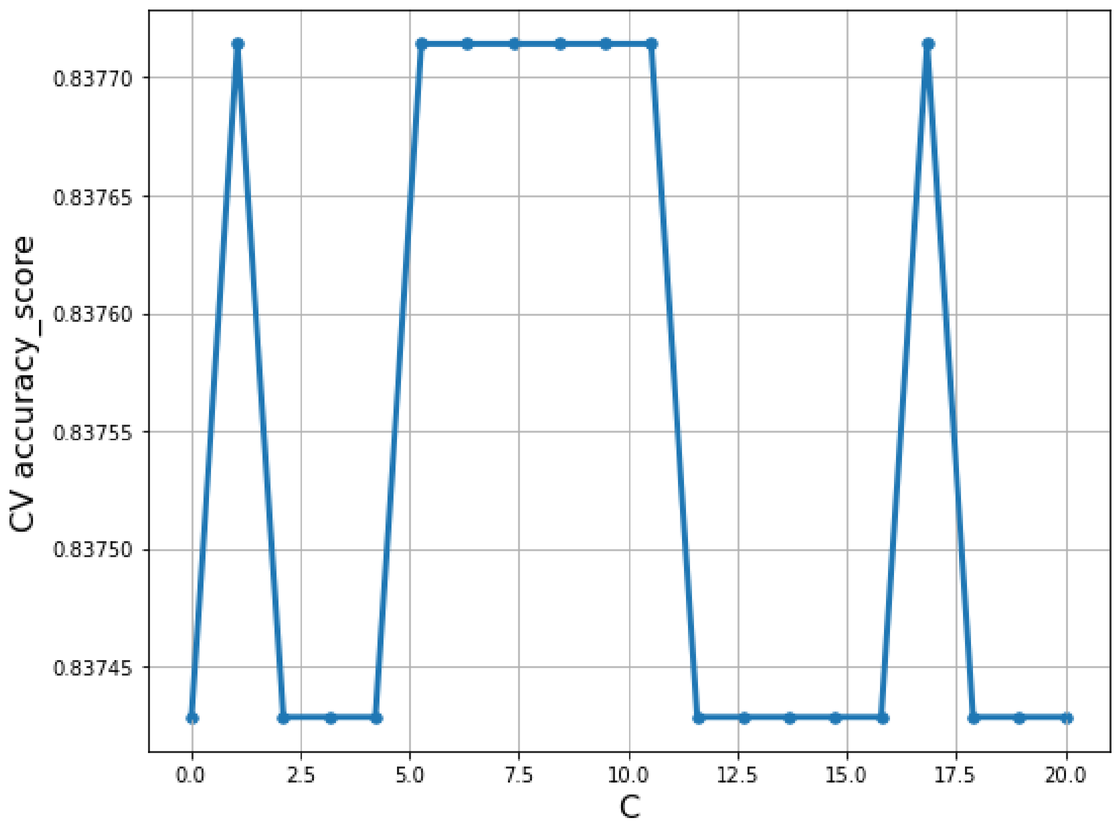
time: 480 s

[54]:
```
C = clf_cv.cv_results_['param_C'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('C', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(C, scores, s=30)
plt.plot(C, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best C', clf_cv.best_params_)
```




```
Best C {'C': 1.0621052631578947}
```

[55]:
```python
from sklearn.metrics import accuracy_score
clf = SVC(kernel = 'linear', C = 1.0621052631578947).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[55]: 0.844

2.4 SVM-Poly Kernel

[56]:
```python
import time
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

grid_values = {'C': np.linspace(0.01,20,20)}

tic = time.time()
clf = SVC(kernel = 'poly')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
```

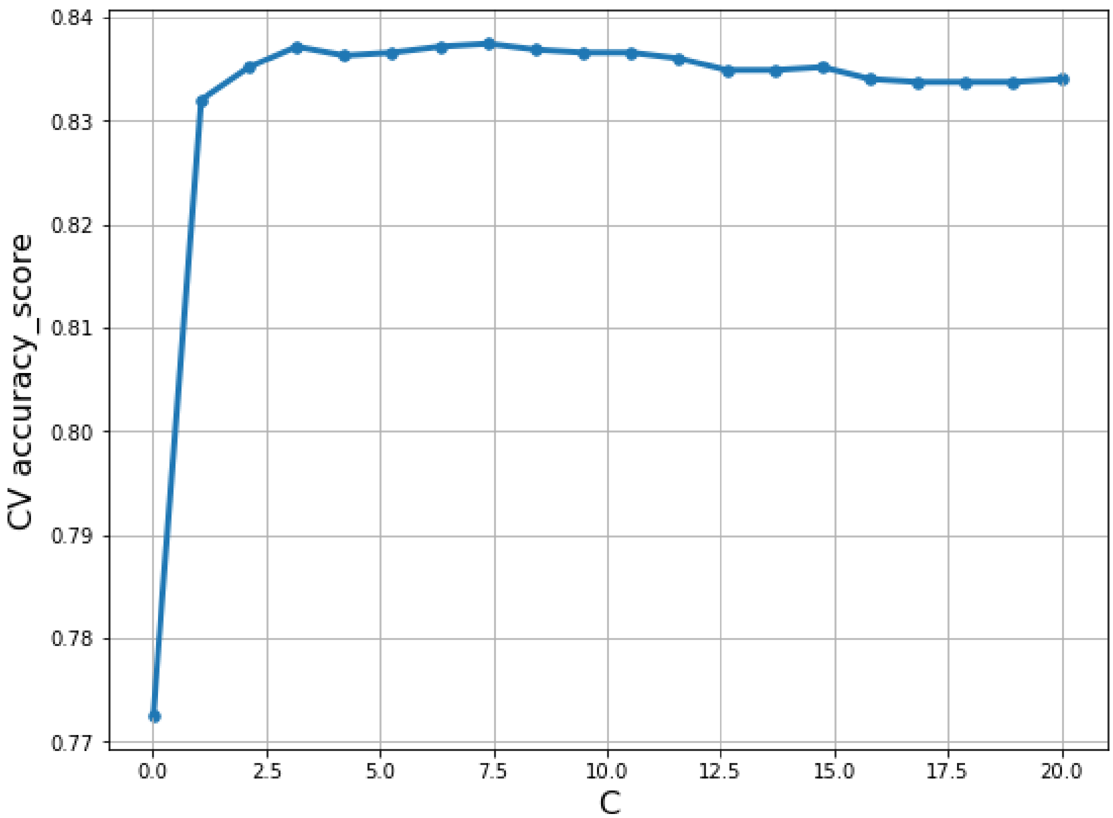
```
time: 41 s
```

[57]:
```python
C = clf_cv.cv_results_['param_C'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('C', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(C, scores, s=30)
plt.plot(C, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best C', clf_cv.best_params_)
```




```
Best C {'C': 7.374736842105262}
```

[58]:
```python
from sklearn.metrics import accuracy_score
clf = SVC(kernel = 'poly', C = 7.374736842105262).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[58]:
```
0.8506666666666667
```

2.5 SVM-RBF Kernel

[59]:
```python
import time
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

grid_values = {'C': np.linspace(0.01,20,20)}

tic = time.time()
clf = SVC(kernel = 'rbf')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
```

```
toc = time.time()
print('time:', round(toc-tic),'s')
```

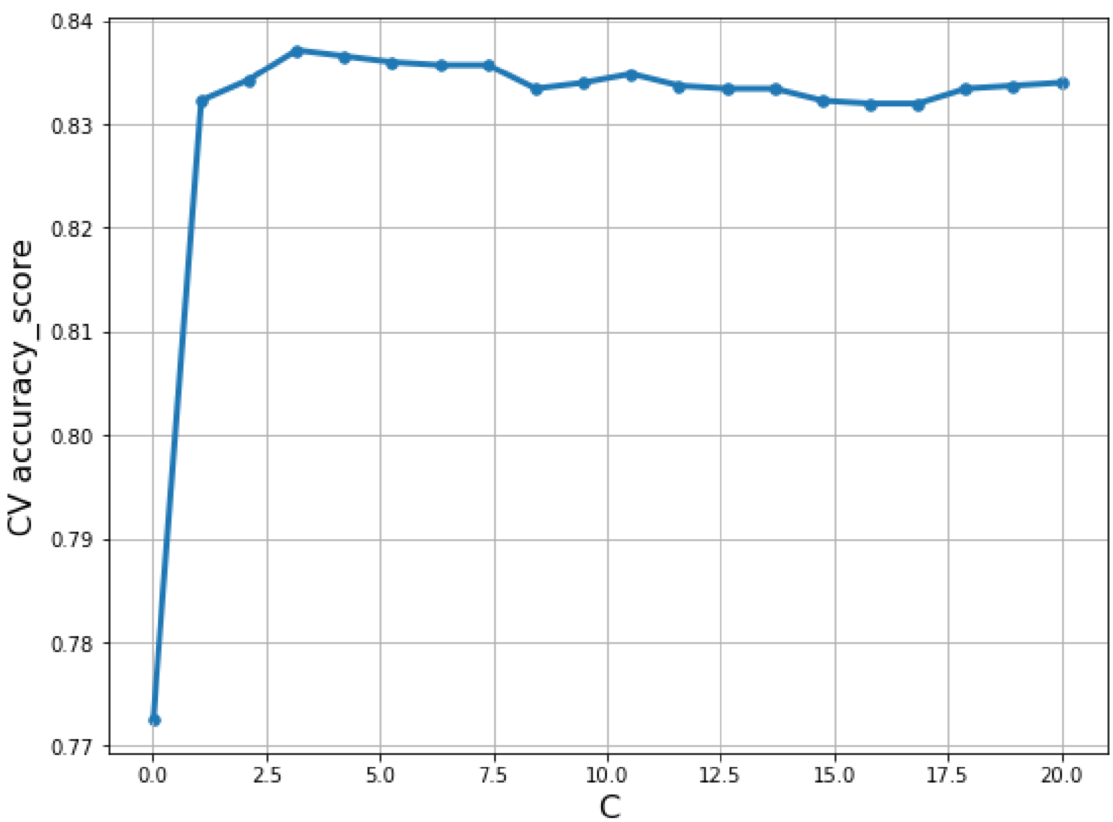
```
time: 43 s
```

[60]:
```
C = clf_cv.cv_results_['param_C'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('C', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(C, scores, s=30)
plt.plot(C, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best C', clf_cv.best_params_)
```



```
Best C {'C': 3.1663157894736838}
```

[61]:
```python
from sklearn.metrics import accuracy_score
clf = SVC(kernel = 'rbf', C = 3.1663157894736838).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[61]:
```
0.842
```

2.6 Logistic Regression

[62]:
```python
import time
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

grid_values = {'C': np.linspace(0, 1, 101)}

tic = time.time()
clf = LogisticRegression()
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
```

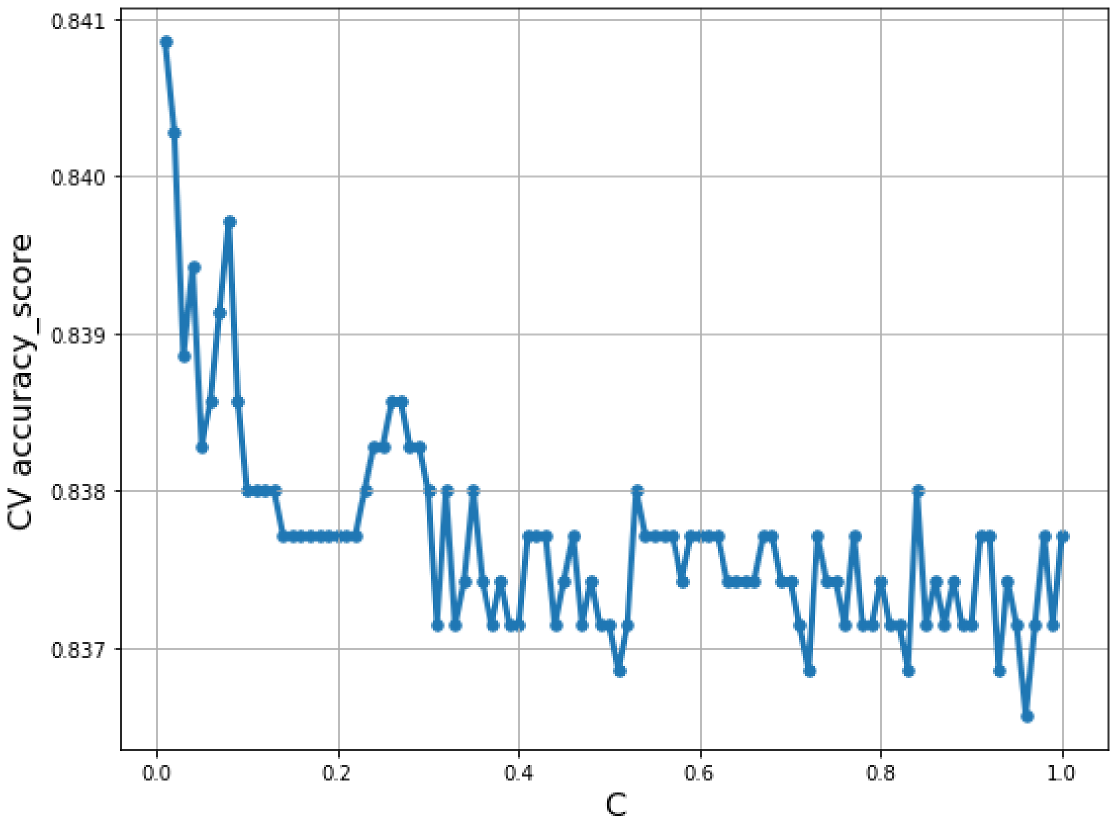
```
time: 21 s
```

[63]:
```python
C = clf_cv.cv_results_['param_C'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('C', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(C, scores, s=30)
plt.plot(C, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best C', clf_cv.best_params_)
```




```
Best C {'C': 0.01}
```

[64]:
```
from sklearn.metrics import accuracy_score
clf = LogisticRegression(C = 0.01).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[64]:
```
0.842
```

2.7 CART

[65]:
```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

grid_values = {'ccp_alpha': np.linspace(0, 0.01, 101)}

tic = time.time()
clf = DecisionTreeClassifier()
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
```

```
toc = time.time()
print('time:', round(toc-tic),'s')
```

```
time: 22 s
```

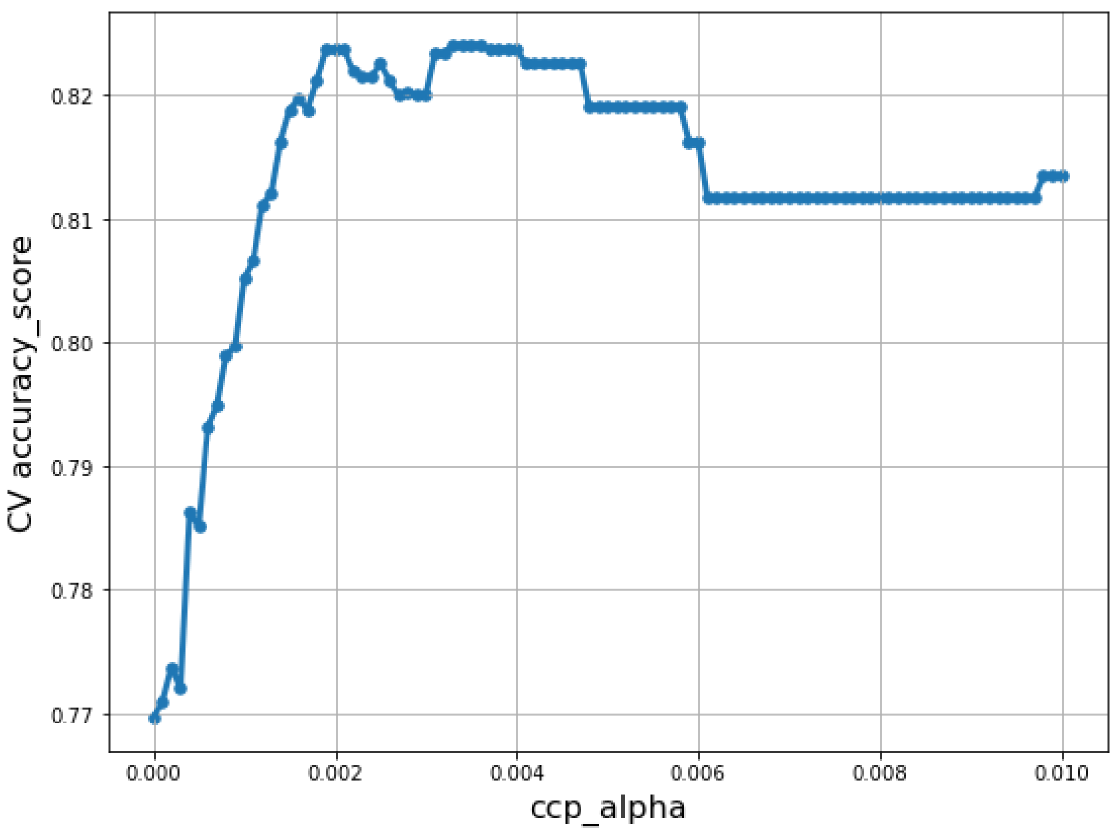
[66]:

```
ccp_alpha = clf_cv.cv_results_['param_ccp_alpha'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(ccp_alpha, scores, s=30)
plt.plot(ccp_alpha, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best ccp_alpha', clf_cv.best_params_)
```



```
Best ccp_alpha {'ccp_alpha': 0.0033}
```

[67]:
```python
from sklearn.metrics import accuracy_score
clf = DecisionTreeClassifier(ccp_alpha = 0.0033).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[67]:
```
0.824
```

2.8 Random Forest

[68]:
```python
import time
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

grid_values = {'ccp_alpha': np.linspace(0, 0.01, 10)}

tic = time.time()
clf = RandomForestClassifier(n_estimators=500)
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
```

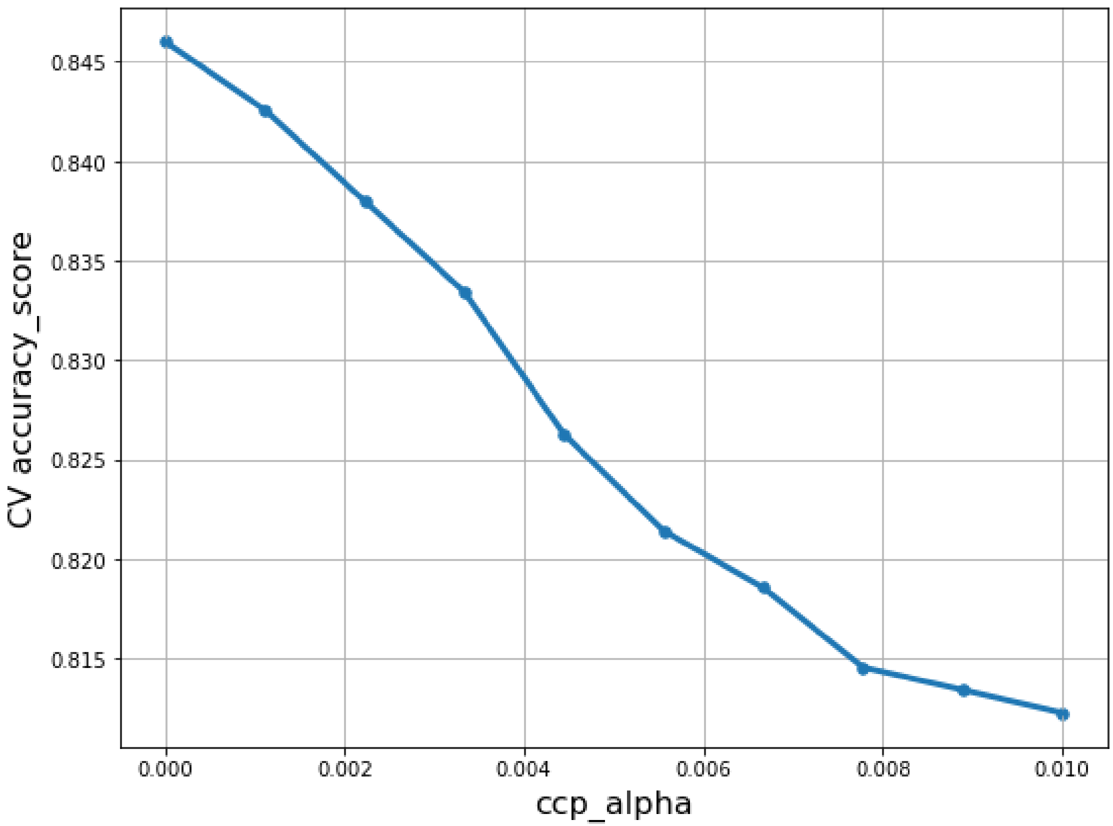
```
time: 182 s
```

[69]:
```python
ccp_alpha = clf_cv.cv_results_['param_ccp_alpha'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(ccp_alpha, scores, s=30)
plt.plot(ccp_alpha, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best ccp_alpha', clf_cv.best_params_)
```




```
Best ccp_alpha {'ccp_alpha': 0.0}
```

[70]:
```
from sklearn.metrics import accuracy_score
clf = RandomForestClassifier(ccp_alpha = 0.0).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[70]:
```
0.852
```

2.9 Gradient Boosting Tree

[71]:
```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier

grid_values = {'ccp_alpha': np.linspace(0, 0.01, 10)}

tic = time.time()
clf = GradientBoostingClassifier(n_estimators=500)
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
```

```
toc = time.time()
print('time:', round(toc-tic),'s')
```

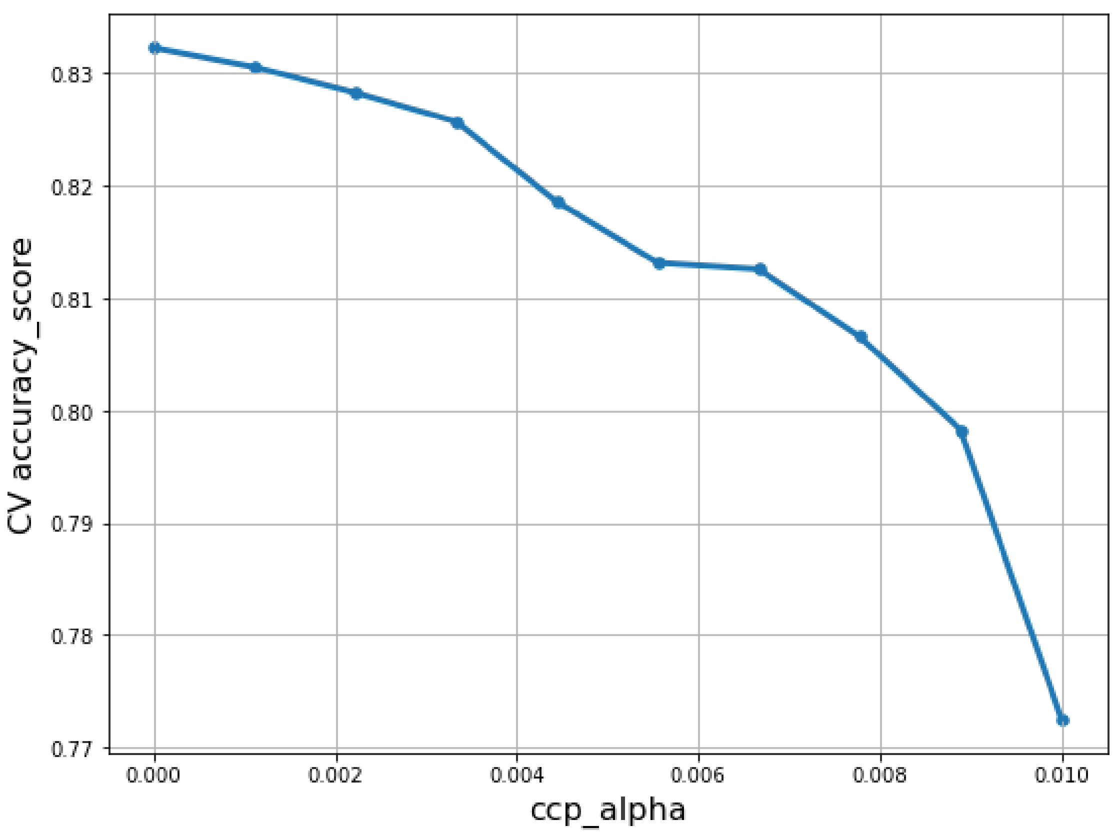
```
time: 284 s
```

[72]:
```
ccp_alpha = clf_cv.cv_results_['param_ccp_alpha'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(ccp_alpha, scores, s=30)
plt.plot(ccp_alpha, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best ccp_alpha', clf_cv.best_params_)
```



Best ccp_alpha {'ccp_alpha': 0.0}

[73]:
```python
from sklearn.metrics import accuracy_score
clf = GradientBoostingClassifier(ccp_alpha = 0.0).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[73]: 0.852

2.10 XGBoost

[74]:
```python
# learning rate
import time
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

grid_values = {'learning_rate': np.logspace(-2, 0, 10)}

tic = time.time()
clf = XGBClassifier(n_estimators=500, eval_metric = 'logloss')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
```

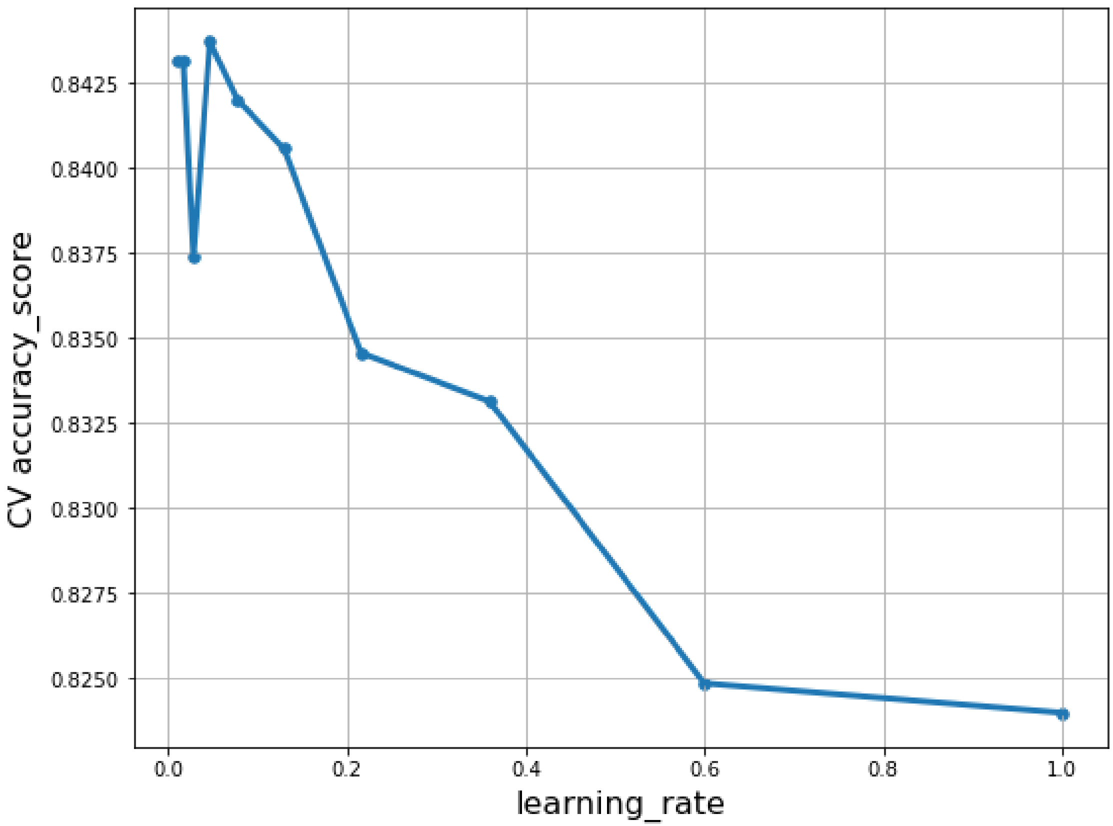
time: 39 s

[75]:
```python
learning_rate = clf_cv.cv_results_['param_learning_rate'].data
scores = clf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('learning_rate', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(learning_rate, scores, s=30)
plt.plot(learning_rate, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best learning_rate', clf_cv.best_params_)
```




```
Best learning_rate {'learning_rate': 0.046415888336127774}
```

[76]:

```
# max_depth
import time
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

grid_values = {'max_depth': range(1,20)}

tic = time.time()
clf = XGBClassifier(n_estimators=500, eval_metric = 'logloss')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
```

```
time: 80 s
```

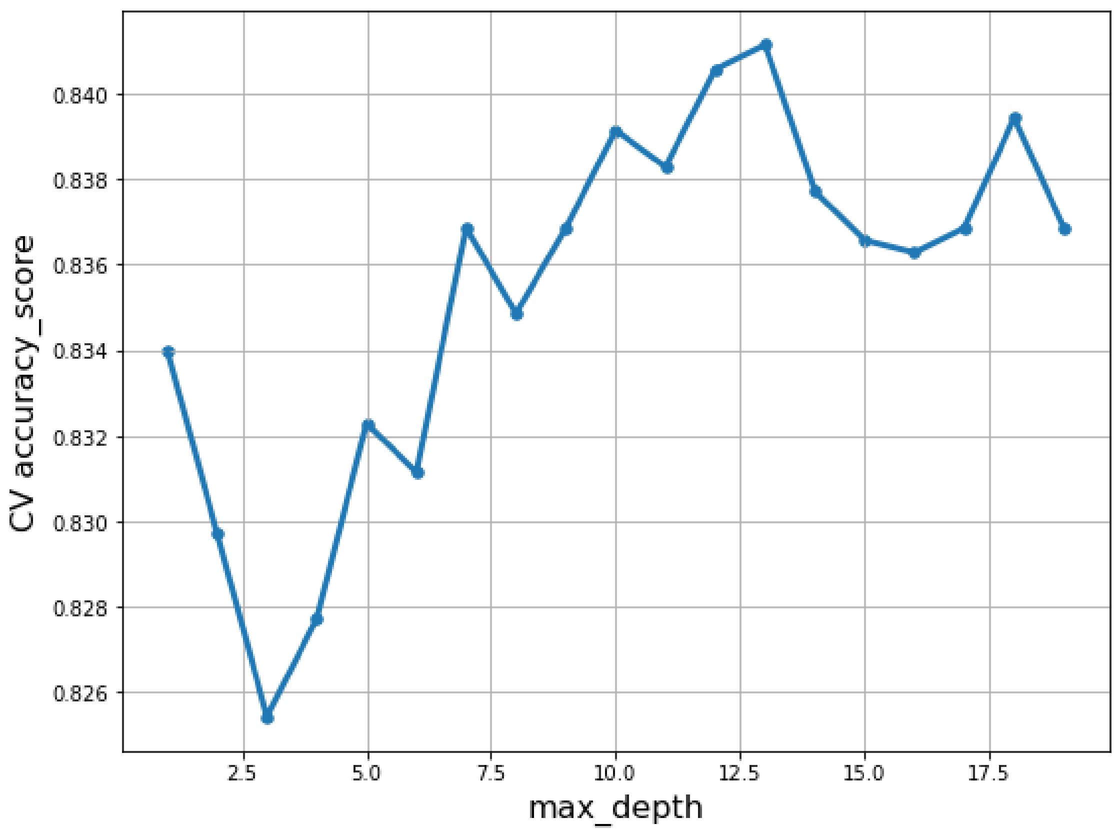
[77]:

```
max_depth = clf_cv.cv_results_['param_max_depth'].data
scores = clf_cv.cv_results_['mean_test_score']
```

```
plt.figure(figsize=(8, 6))
plt.xlabel('max_depth', fontsize=16)
plt.ylabel('CV accuracy_score', fontsize=16)
plt.scatter(max_depth, scores, s=30)
plt.plot(max_depth, scores, linewidth=3)
plt.grid(True, which='both')
# plt.xlim([0, 0.01])
# plt.ylim([0.8, 1])

plt.tight_layout()
plt.show()

print('Best max_depth', clf_cv.best_params_)
```




```
Best max_depth {'max_depth': 13}
```

[78]:
```
import time
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

```
grid_values = {'learning_rate': np.logspace(-2, 0, 20)
               ,'max_depth': range(1,10)
               }

tic = time.time()
clf = XGBClassifier(n_estimators=500, eval_metric = 'logloss')
clf_cv = GridSearchCV(clf, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=0)
clf_cv.fit(Xtrain, Ytrain)
toc = time.time()
print('time:', round(toc-tic),'s')
print('Best learning_rate', clf_cv.best_params_)
```

```
time: 674 s
Best learning_rate {'learning_rate': 0.016237767391887217, 'max_depth': 5}
```

[79]:

```
from sklearn.metrics import accuracy_score
clf = XGBClassifier(n_estimators=500
                    ,eval_metric = 'logloss'
                    ,learning_rate = 0.016237767391887217
                    ,max_depth = 4).fit(Xtrain, Ytrain)
accuracy_score(Ytest, clf.predict(Xtest))
```

[79]: 0.858

[82]:

```
print('time:', round(time.time()-time_start),'s')
```

```
time: 1993 s
```

[ ]:

```
```