Cross Validation and Prediction Using Regression Tree

Concept Session

Demo 5.1: Decision Tree Regressor

Import libraries

In [ ]:
```
# data manipulation
import numpy as np
import pandas as pd

# modeling utilities
from sklearn import metrics
from sklearn import preprocessing
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, cross_val_score, cross_val_predict, train_test_split
from sklearn import tree

# plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore')
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

Loading Data

In [ ]:
```
hour_df = pd.read_csv("/content/drive/MyDrive/ML/DS2_C5_S5_BikeSharing_Data_Concept.csv")
hour_df.head()
```

1. Data Exploration

In [ ]:
```
hour_df.info()
```

In [ ]:
```
hour_df.describe()
```

2. Preprocessing

In [ ]:
```
# Renaming columns names to more readable names
hour_df.rename(columns={'instant':'rec_id',
                        'dteday':'datetime',
                        'holiday':'is_holiday',
                        'workingday':'is_workingday',
                        'weathersit':'weather_condition',
                        'hum':'humidity',
                        'mnth':'month',
                        'cnt':'total_count',
                        'hr':'hour',
                        'yr':'year'},inplace=True)

########################
# Setting proper data types
########################
# date time conversion
hour_df['datetime'] = pd.to_datetime(hour_df.datetime)

# categorical variables
hour_df['season'] = hour_df.season.astype('category')
hour_df['is_holiday'] = hour_df.is_holiday.astype('category')
hour_df['weekday'] = hour_df.weekday.astype('category')
hour_df['weather_condition'] = hour_df.weather_condition.astype('category')
hour_df['is_workingday'] = hour_df.is_workingday.astype('category')
hour_df['month'] = hour_df.month.astype('category')
hour_df['year'] = hour_df.year.astype('category')
hour_df['hour'] = hour_df.hour.astype('category')
```

In [ ]:
```
# check missing values in train data
hour_df[hour_df.isnull().any(axis=1)]
```

In [ ]:
```
# detect correlations
column_correlation = hour_df.corr()
column_correlation
```

In [ ]:
```
import seaborn as sns

sns.heatmap(column_correlation);
plt.show()
```

3. Data Preparation - Feature Engineering

Creation of dummy variables: Convert categorical data to numbers

In [ ]:
```
# Encoding all the categorical features
cat_attr_list = ['season','is_holiday',
                 'weather_condition','is_workingday',
                 'hour','weekday','month','year']
# though we have transformed all categoricals into their one-hot encodings, note that ordinal
# attributes such as hour, weekday, and so on do not require such encoding.
numeric_feature_cols = ['temp','humidity','windspeed',
                        'hour','weekday','month','year']
subset_cat_features =  ['season','is_holiday','weather_condition','is_workingday']
```

In [ ]:
```
#Encoding season
season_dummies = pd.get_dummies(hour_df.season, prefix="season")
df_with_dummies = pd.concat([hour_df,season_dummies],axis='columns')
df_with_dummies.drop('season',axis='columns',inplace=True)
# is_holiday
is_holiday_dummies = pd.get_dummies(hour_df.is_holiday, prefix="is_holiday")
df_with_dummies = pd.concat([df_with_dummies,is_holiday_dummies],axis='columns')
df_with_dummies.drop('is_holiday',axis='columns',inplace=True)
# weather_condition
weather_condition_dummies = pd.get_dummies(hour_df.weather_condition, prefix="weather_condition")
df_with_dummies = pd.concat([df_with_dummies,weather_condition_dummies],axis='columns')
df_with_dummies.drop('weather_condition',axis='columns',inplace=True)
# is_workingday
is_workingday_dummies = pd.get_dummies(hour_df.is_workingday, prefix="is_workingday")
df_with_dummies = pd.concat([df_with_dummies,is_workingday_dummies],axis='columns')
df_with_dummies.drop('is_workingday',axis='columns',inplace=True)
# hour
hour_dummies = pd.get_dummies(hour_df.hour, prefix="hour")
df_with_dummies = pd.concat([df_with_dummies,hour_dummies],axis='columns')
df_with_dummies.drop('hour',axis='columns',inplace=True)
# weekday
weekday_dummies = pd.get_dummies(hour_df.weekday, prefix="weekday")
df_with_dummies = pd.concat([df_with_dummies,weekday_dummies],axis='columns')
df_with_dummies.drop('weekday',axis='columns',inplace=True)
# month
month_dummies = pd.get_dummies(hour_df.month, prefix="month")
df_with_dummies = pd.concat([df_with_dummies,month_dummies],axis='columns')
df_with_dummies.drop('month',axis='columns',inplace=True)
# year
year_dummies = pd.get_dummies(hour_df.year, prefix="year")
df_with_dummies = pd.concat([df_with_dummies,year_dummies],axis='columns')
df_with_dummies.drop('year',axis='columns',inplace=True)
df_with_dummies.head()
```

In [ ]:
```
X = df_with_dummies
X.head()
y=hour_df.total_count
```

In [ ]:
```
X.drop('datetime', axis='columns', inplace=True)
```

Splitting the dataset into training and test datasets

In [ ]:
```
# Divide the dataset into training and testing sets
df_train, df_test = train_test_split(X, train_size=0.7)
print('Size of training dataset: ', df_train.shape)
print('Size of test dataset: ', df_test.shape)
```

In [ ]:
```
X_train = df_train.drop(columns='total_count', axis =1)
y_train = df_train['total_count']
X_test = df_test.drop(columns='total_count', axis =1)
y_test = df_test['total_count']
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

4. Modeling

In [ ]:
```
dtm = DecisionTreeRegressor(max_depth=4,
                            min_samples_split=5,
                            max_leaf_nodes=10)

dtm.fit(X_train,y_train)
print("R-Squared on train dataset={}".format(dtm.score(X_test,y_test)))

dtm.fit(X_test,y_test)
print("R-Squaredon test dataset={}".format(dtm.score(X_test,y_test)))
```

5. Visualization

In [ ]:
```
# Setting dpi = 300 to make image clearer than default
fig, axes = plt.subplots(dpi=300)

tree.plot_tree(dtm,
               filled = True);

#save figure in current directory as png
fig.savefig('Bike_regression_decisiontree.png')
```

Demo 5.2: Cross Validation

Hold Out Method

In [ ]:
```
from sklearn.model_selection import train_test_split

X = [9,19,29,39,49,59,69,79,89,99]
train, test= train_test_split(X,test_size=0.25, random_state=1)
print("Train:",train,"Test:" ,test)
```

Leave One Out Method

In [ ]:
```
from sklearn.model_selection import LeaveOneOut

X = [9,19,29,39,49,59,69,79,89,99]
LOO = LeaveOneOut()

for train, test in LOO.split(X):
    print("%s %s"% (train,test))
```

K-Fold Method

In [ ]:
```
from sklearn.model_selection import KFold

X = [9,19,29,39,49,59,69,79,89,99]

kf = KFold(n_splits=3, shuffle=False, random_state=None)

for train, test in kf.split(X):
    print("Train data",train,"Test data",test)
```

Stratified K-Fold Method

In [ ]:
```
import numpy as np
from sklearn.model_selection import StratifiedKFold


X = np.array([[9,19],[29,39],[49,59],[69,79],[89,99],[109,119]])
y= np.array([0,0,1,0,1,1])

skf = StratifiedKFold(n_splits=3,random_state=None,shuffle=False)

for train_index,test_index in skf.split(X,y):
    print("Train:",train_index,'Test:',test_index)
    X_train,X_test = X[train_index], X[test_index]
    y_train,y_test = y[train_index], y[test_index]
```

Demo 5.3: Find Optimum Depth Level of Regressor Tree Using Stratified K-Fold Cross Validation method

In [ ]:
```
kf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

cnt = 1
# split()  method generate indices to split data into training and test set.
for train_index, test_index in kf.split(X, y):
    print(f'Fold:{cnt}, Train set: {len(train_index)}, Test set:{len(test_index)}')
    cnt+=1
```

In [ ]:
```
def rmse(score):
    rmse = np.sqrt(-score)
    print(f'rmse= {":.2f".format(rmse)}')
```

In [ ]:
```
# We will try with max depth starting from 1 to 15 and depending on the final 'rmse' score choose the v
alue of max_depth.
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn import  tree

max_depth = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

for val in max_depth:
    score = cross_val_score(tree.DecisionTreeRegressor(max_depth= val, random_state= 42), X, y, cv= kf,
 scoring="neg_mean_squared_error")
    print(f'For max depth: {val}')
    rmse(score.mean())
```