Solar Power Generation Forecast Project

Taylor Boyd - CS 458

I. INTRODUCTION

The goal of the project is to use data from three solar power plants located in Australia in order to build models that can predict solar power generation 24 hours ahead. The project is split into three parts: 1. Split data into training and testing datasets 2. Build a 24 hr ahead solar power generation forecast model 3. Evaluate the models through Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) measures.

II. LITERATURE REVIEW

All of the information for the project has been attained by using class resources and the Scikit-learn website. More specifically, I heavily referenced the Scikit-learn Support Vector Regression (SVR) page [1] in order to figure out how to use SVR model. As for class resources, I used lecture slides and information from previous projects.

III. METHODOLOGY

For data preprocessing, I first needed to split the data into training and testing datasets. In order to do this, I first casted the timestamp into different int parts. That way, it could be checked if the timestamp of any given line of data is smaller (earlier) or greater (later) than 20130701 00:00. If the timestamp is earlier than or equal to 20130701 00:00, the line of data is written to the training dataset file; otherwise, the line of data is written to the testing dataset file. Then, the data from each file was split into data and label numpy arrays for each solar plant.

Next, I considered how to build classifiers for each solar plant and make 24hr window predictions; I decided on using SVRs because of their multi-class capabilities using kernels. Before building the SVRs, I first decided to split the possible power generation into 5 different ranges (0-0.2, 0.2-0.4, 0.4-0.6, 0.6-0.8, 0.8-1.0). Using the newly defined training labels, I built a SVR for each solar plant.

Next, I made a function that makes the predictions for 24hrs ahead. It takes in a classifier, test samples, and some index and basically looks at the last week of data (if it's available) in order to make the prediction. The index is the index of a sample with the same date to be predicted. While this would make it difficult for a user to put in a date and get a prediction, I took the predict on "a rolling basis" project description as instruction to go ahead and make predictions for every day of the year for the MAE and RMSE calculations so it doesn't really matter that it may be difficult for a user. The power generation predictions from the last 7 days are averaged in order to come up with the 24hr prediction as output.

For my first submission of this project, I was using SVM models. In trying to improve my results, I switched from SVM models to SVR models. I thought about changing or taking out the splitting of the labels into 5 ranges but it seemed my overall results stayed the same or even got a bit worse when I made attempts.

IV. RESULTS

My original results, using SVMs:

ZONEID	1	2	3	OVERALL
MAE	0.18876	0.18850	0.19681	0.19136
RMSE	0.2409	0.23652	0.25143	0.24298

My final results:

ZONEID	1	2	3	OVERALL
MAE	0.17811	0.18713	0.18999	0.18508
RMSE	0.22121	0.23172	0.23445	0.22913

RMSE results were slightly worse, but overall both look pretty good. I think it's interesting that the error is higher for plant 3 predictions in both calculations; plant 3 may need some more specialized parameters. Comparing the two tables, you can see that after I made changes to my original approach, I ended up with better results. The MAE average went down by 0.00628 and the RMSE average went down by 0.01385.

V. CONCLUSIONS

I think I did data preprocessing just fine. If given more time, I would have liked to store all the timestamps somewhere so that I could have made this project into a more user friendly one. I would have made it so that a user could put any day/month combo in and gotten a single prediction of how much power would be generated in the next 24 hours after that inputted date.

Switching from using SVMs to SVRs was definitely a good decision which improved my results. Splitting the possible power generation into different ranges is still something I wish I could have gotten rid of. I think that given more time, I would try to better optimize the SVR parameters so that my results would not worsen if I were to handle the labels differently.

Overall, I liked this project because of its real-world applications and the different opportunities or approaches it allowed for.

References

[1] Scikit-learn, ”Sklearn.svm.SVR,” https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

CS458_SolarProj

December 12, 2020

1 Solar Power Generation Forecast Report

Taylor Boyd

CS458

1.1 Part 1 - Splitting Data

Split "solar.csv" into training dataset "solar_training.csv" and test dataset "solar_test.csv".

[6]:
```python
f = open("solar.csv", 'r')
f2 = open("solar_training.csv", 'w')
f3 = open("solar_test.csv", 'w')

ctr = 0 # variable that tracks what line of the file we're on

# go through each line of the file
for line in f:
    if ctr != 0:
        a_str = line
        a_str = a_str.strip().split(',')
        time = a_str[1]
        time = time.split(' ')
        hr = time[1].split(":")
        hr = int(hr[0])
        # if timestamp is 20130701 00:00 or earlier, write data to training␣
 ↪dataset file
        if int(time[0]) < 20130701:
            f2.write(line)
        elif (int(time[0]) == 20130701 and hr == 0):
            f2.write(line)
        # else, write data to testing dataset file
        else:
            f3.write(line)
    ctr +=1
f.close()
f2.close()
f3.close()
```

In order to split the data, I casted the timestamp into different int parts. That way, it could be checked if the timestamp of any given line of data is smaller (earlier) or greater (later) than 20130701 00:00. If the timestamp was earlier than or equal to 20130701 00:00, the line of data is written to the training dataset file; otherwise, the line of data is written to the testing dataset file.

1.2 Part 2 - Model Building

Build a 24 hr ahead solar power generation forecast model.

[18]:
```python
import numpy as np
import math
import pandas as pd
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline

f2 = open("solar_training.csv", 'r')
f3 = open("solar_test.csv", 'r')

p1_trainingX = [] # plant 1 training data
p1_trainingY = []
p2_trainingX = [] # plant 2 training data
p2_trainingY = []
p3_trainingX = [] # plant 3 training data
p3_trainingY = []

p1_testX = [] # plant 1 test data
p1_testY = []
p2_testX = [] # plant 2 test data
p2_testY = []
p3_testX = [] # plant 3 test data
p3_testY = []

# helper function for separating data and labels
def store_data(dataX, dataY, a_str):
    a_str.pop(0)
    a_str.pop(0)
    temp = []
    for item in range(len(a_str)-1):
        temp.append(float(a_str[item]))
    dataX.append(temp)
    dataY.append(float(a_str[len(a_str)-1]))
    return 0

# go through each line of training dataset file
```

```python
for line in f2:
    line = line.strip().split(',')
    a_str = line
    plant = int(a_str[0])
    if plant == 1:
        store_data(p1_trainingX, p1_trainingY, a_str)
    elif plant == 2:
        store_data(p2_trainingX, p2_trainingY, a_str)
    else:
        store_data(p3_trainingX, p3_trainingY, a_str)

# go through each line of testing dataset file
for line in f3:
    line = line.strip().split(',')
    a_str = line
    plant = int(a_str[0])
    if plant == 1:
        store_data(p1_testX, p1_testY, a_str)
    elif plant == 2:
        store_data(p2_testX, p2_testY, a_str)
    else:
        store_data(p3_testX, p3_testY, a_str)

f2.close()
f3.close()

# convert lists to np arrays
p1_trainingX = np.array(p1_trainingX)
p1_trainingY = np.array(p1_trainingY)
p2_trainingX = np.array(p2_trainingX)
p2_trainingY = np.array(p2_trainingY)
p3_trainingX = np.array(p3_trainingX)
p3_trainingY = np.array(p3_trainingY)
p1_testX = np.array(p1_testX)
p1_testY = np.array(p1_testY)
p2_testX = np.array(p2_testX)
p2_testY = np.array(p2_testY)
p3_testX = np.array(p3_testX)
p3_testY = np.array(p3_testY)

# function that splits labels into 5 different classes
def split_classes(arr):
    new_arr = np.zeros(arr.shape[0])
    for i in range(arr.shape[0]):
        if arr[i] < 0.2:
            new_arr[i] = 0
        elif arr[i] < 0.4:
            new_arr[i] = 1
        elif arr[i] < 0.6:
            new_arr[i] = 2
        elif arr[i] < 0.8:
            new_arr[i] = 3
        else:
            new_arr[i] = 4
    return new_arr

# variables for newly defined labels
p1train = np.array(split_classes(p1_trainingY))
p2train = np.array(split_classes(p2_trainingY))
p3train = np.array(split_classes(p3_trainingY))

# build svr classifier for each plant
regr1 = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
regr1.fit(p1_trainingX, p1train)
regr2 = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
regr2.fit(p2_trainingX, p2train)
regr3 = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
regr3.fit(p3_trainingX, p3train)

# function for predicting power generation 24hrs after givin timestamp
# takes in n which is the index for the sample with targeted timestamp
def predict_tmrw(clf, testX, n):
    window = []
    # if there is a week of history, use that to make prediction
    for i in range(7):
        if (n-i) >= 0:
            window.append(testX[n-i])
    window = np.array(window)
    pred = 0
    preds = 0
    for i in range(window.shape[0]):
        preds += clf.predict(window)
    for i in range(preds.size):
        pred += preds[i]
    pred = pred / preds.size
    pred = (pred / window.shape[0]) / 5 + 0.1
    return pred
```

In order to build classifiers for each solar plant and make 24hr window predictions, I first decided to split the possible power generation into 5 different ranges (0-0.2, 0.2-0.4, 0.4-0.6, 0.6-0.8, 0.8-1.0). Using the newly defined training labels, I built a classifier for each solar plant. Next, I made a function that makes the predictions for 24hrs ahead. It takes in a classifier, test samples, and some index and basically looks at the last week of data (if it's available) in order to make the prediction. The power generation predictions from the last 7 days are averaged in order to come up with the

24hr prediction as output.

1.3 Part 3 - Model Evaluation

Use MAE and RMSE measures to evaluate the model.

[19]:
```python
# takes in trained model and test data, outputs mae
def mae_function(model, testX, testY):
    mae = 0
    for i in range(testX.shape[0]):
        pred = predict_tmrw(model, testX, i)
        mae += abs(testY[i] - pred)
    mae = mae / testX.shape[0]
    return mae

# takes in trained model and test data, outputs rmse
def rmse_function(model, testX, testY):
    rmse = 0
    for i in range(testX.shape[0]):
        pred = predict_tmrw(model, testX, i)
        rmse += (abs(testY[i] - pred)) * (abs(testY[i] - pred))
    rmse = rmse / testX.shape[0]
    rmse = math.sqrt(rmse)
    return rmse

# calculate and print mae of each plant's model
p1_mae = mae_function(regr1, p1_testX, p1_testY)
p2_mae = mae_function(regr2, p2_testX, p2_testY)
p3_mae = mae_function(regr3, p3_testX, p3_testY)
print("MAE")
print(p1_mae)
print(p2_mae)
print(p3_mae)

# calculate and print rmse of each plant's model
p1_rmse = rmse_function(regr1, p1_testX, p1_testY)
p2_rmse = rmse_function(regr2, p2_testX, p2_testY)
p3_rmse = rmse_function(regr3, p3_testX, p3_testY)
print("RMSE")
print(p1_rmse)
print(p2_rmse)
print(p3_rmse)
```

```
MAE
0.1781144352180035
0.1871333440553051
0.18999626716860712
RMSE
```

```
0.22121105569287766
0.23172321897512396
0.23444825402235855
```

RMSE results are a little worse than MAE results but both are pretty good overall. I think it's interesting that the error is higher for plant 3 predictions in both calculations.