

Melissa LaHoud

Grand Canyon University

DSC 540: Machine Learning

Dr. Aiman Darwiche

4/14/2021

## Part 2 - Application

The *sinc* function is one of the commonly used datasets for testing nonlinear regression algorithms. This function is given by the following equation:

$$\text{sinc}(x) = \frac{\sin \pi x}{\pi x}$$

Familiarize yourself with the SVM tools in Python, which can be found within your topic materials.

Create a Jupyter notebook and implement the following (in Python):

- (a) Generate 50 data points from this function in the range  $[-3, 3]$ .
- (b) Add Gaussian noise to the data.
- (c) Train an SVM regressor with the data generated in (a). Define (and explain) suitable parameters required for training the regressor.
- (d) Describe the functionality of the regressor.
- (e) Discuss the potential use of the regressor and quantify its accuracy.

After you assess the importance and approach to using the *sinc* function in conjunction with SVM, refer to "A Signal Theory Approach to Support Vector Classification: The Sinc Kernel" within your topic materials. You are not expected to grasp all the concepts and theorems described in the article, but skim through it on a high level, to get some insight into the work it describes. Upon skimming through the article, expand your discussion in (e) above, to include some of the relevant points.

```

import numpy as np
import math
import pandas as pd
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import random

```

```

mu = 0
sigma = 1
x = np.linspace(-3,3,50)
y=np.sinc(x)

```

```

noise = y[[0]] + 0.01*random.gauss(mu, sigma)
for i in range(49):
    noise = np.vstack((noise, y[[i+1]] + 0.01*random.gauss(mu, sigma)))

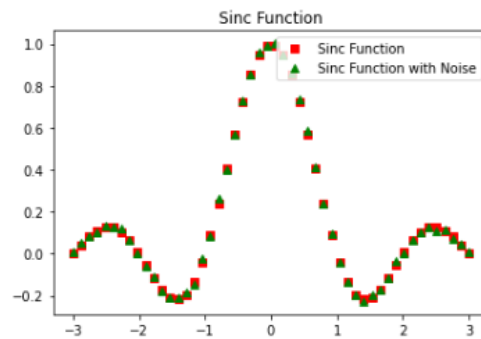
```

```

plt.figure(1)
plt.plot(x,y,'rs', x, noise, 'g^')
plt.title("Sinc Function")
plt.legend(['Sinc Function','Sinc Function with Noise'], loc='upper right')
plt.show

```

35]: <function matplotlib.pyplot.show(close=None, block=None)>



36]:

```

regressor1 = SVR(kernel = 'rbf')
regressor1.fit(x1,Noise)
y_pred1 = regressor1.predict(x1)

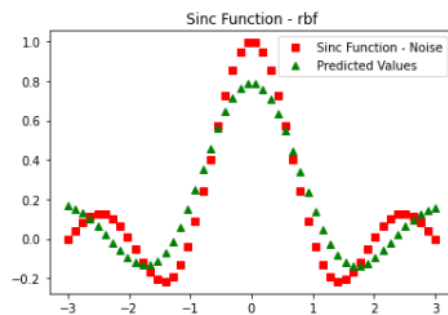
```

```

plt.figure(1)
plt.plot(x,y,'rs', x, y_pred1, 'g^')
plt.title("Sinc Function - rbf")
plt.legend(['Sinc Function - Noise','Predicted Values'], loc='upper right')
plt.show

```

37]: <function matplotlib.pyplot.show(close=None, block=None)>



```

mean_squared_error(Noise, y_pred1)

```

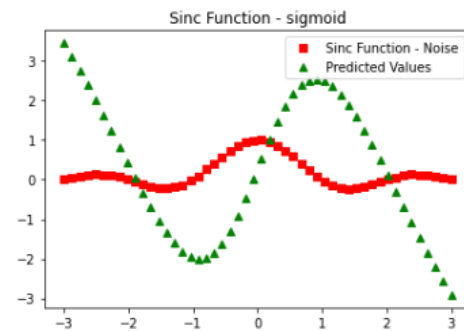
38]: 0.014985992932663677

```
x1=x.reshape(-1,1)
Noise = np.ravel(noise)
```

```
regressor2 = SVR(kernel = 'sigmoid')
regressor2.fit(x1,Noise)
y_pred2 = regressor2.predict(x1)
```

```
plt.figure(1)
plt.plot(x,noise,'rs', x, y_pred2, 'g^')
plt.title("Sinc Function - sigmoid")
plt.legend(['Sinc Function - Noise', 'Predicted Values'], loc='upper right')
plt.show
```

```
17]: <function matplotlib.pyplot.show(close=None, block=None)>
```



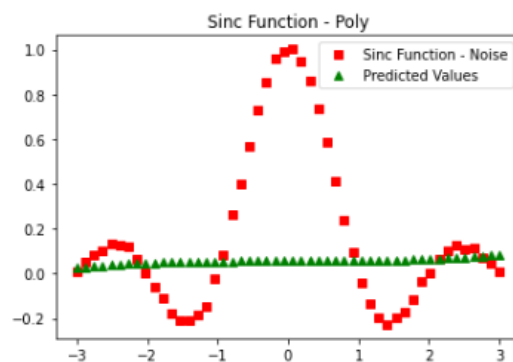
```
mean_squared_error(Noise, y_pred2)
```

```
11]: 3.190352466609899
```

```
regressor3 = SVR(kernel = 'poly')
regressor3.fit(x1,Noise)
y_pred3 = regressor3.predict(x1)
```

```
plt.figure(1)
plt.plot(x,noise,'rs', x, y_pred3, 'g^')
plt.title("Sinc Function - Poly")
plt.legend(['Sinc Function - Noise', 'Predicted Values'], loc='upper right')
plt.show
```

```
.]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
mean_squared_error(Noise, y_pred3)
```

```
.]: 0.1435384937284845
```

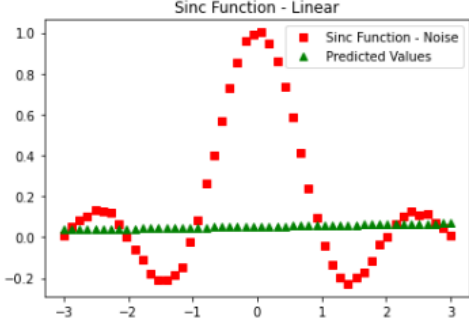
```

In [ ]: regressor4 = SVR(kernel = 'linear')
        regressor4.fit(x1,Noise)
        y_pred4 = regressor4.predict(x1)

In [ ]: plt.figure(1)
        plt.plot(x,noise,'rs', x, y_pred4, 'g^')
        plt.title("Sinc Function - Linear")
        plt.legend(['Sinc Function - Noise', 'Predicted Values'], loc='upper right')
        plt.show

]: <function matplotlib.pyplot.show(close=None, block=None)>

```



```

In [ ]: mean_squared_error(Noise, y_pred4)

]: 0.14416919368915557

```

I tried four different regressors, radial basis function, sigmoid, polynomial, and linear. Radial basis function is a real-valued function whose value depends only on the distance the input and some fixed point. The distance is usually Euclidean distance. A sigmoid function is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point and exactly one inflection point having a characteristic S-shaped curve. Polynomial function is a function that involves only non-negative integer powers of a variable in an equation like the quadratic equation, cubic equation, etc. Linear function attempts to model the relationship between two variables by fitting a linear equation to observed data. Looking at the code in python above, we can see how each regressor behaves with the sinc function. If we first look at the polynomial and linear regressors, they have a very similar output. The predicted values are around 0 when the sinc function is not and the mean squared error in around 0.14 for both. The parameters for the polynomial regressor has a default degree of 3 and when I ran it with different degrees, it did not change so I kept it at the default degree. Looking

at the sigmoid regressor, we can see that the predicted values are in S-shaped curve but looks that it does a very poor job of predicting the sinc function. It has a mean squared error of 3.19 which is very high and would help understand that it is not very accurate. Lastly, the radius basis function is by far the best regressor to predict the sinc function. We can see this is the graph produced in python, along with a low mean squared error of 0.014. I would choose this regressors as it is the most accurate with the sinc function.

After reading, "A Signal Theory Approach to Support Vector Classification: The Sinc Kernel", I thought about the potential use of the regressor and its accuracy and these depending on the parameters used. Using a regressor and its correct parameters has a impact on how accurate the regressor will be. According to Nelson, J., et. al., 2009, "Knowledge pertaining to the structure of the manifold can be used to guide the choice of parameters, and thus the nature and degree of regularization. Such realizations lead to a more considered approach: that is to ascertain, a priori, properties of the space wherein the data lie. Although there may still exist infinitely many solutions, the range of an empirical search could then at least be focused upon subsets of parameters rather than all possible choices of parameters." The parameter set (or hyper-parameter) is chosen by estimating the performance of the SVM for each parameter value. The value of  $\omega_* = \{\omega_*^r\}_1^d$  that yields the best performance is then chosen as the optimal parameter. Therefore, the optimal parameter will help improve the measure SVM performance and/or accuracy. Specifically for the sinc function, there is a way to estimate a search space where the optimal parameter lies and it can be found by following an approach of filter design. This could help with finding the most accurate regressor for the sinc function or a even more accurate radius basis function that I found in python.

## References

Nelson, J., Damper, R., Gunn, S., & Guo, B. (2009, January). A signal theory approach to support vector classification: The sinc kernel. Retrieved from [https://www.sciencedirect-com.lopez.idm.oclc.org/science/article/pii/S0893608008002347?via=ihub](https://www.sciencedirect.com/science/article/pii/S0893608008002347?via=ihub)