Martinez-assn-4

September 29, 2020

1 Martinez Assn 4

1.1 Support Vector Machines

1.1.1 Part 1 - Theory

1.1.2 a. You have a choice of handling a binary classification task using number of misclassifications as the performance measure and maximizing the margin between the two classes as the performance measure. On what factors does your decision depend? Provide a formal explanation, supported by theorems and ideas presented in the readings associated with this topic.

A main consideration of leveraging this particular function is whether or not the data is linearly separable. This is crucial because the criterion function is a stepwise constant which makes it a weak candidate for gradient search (Gopal, 2020). In other words, it may not be possible to properly minimize loss and identify proper weights when the decision boundary is nonlinear.

This then leads us into the second option, maximizing the margin. Since functions such as the perceptron algorithm do not have a particular set of boundaries, there are several ways that a linear boundary can be constructed(Gopal, 2020). While this may sound useful, it has two main issues:

- The problem must be NP-Complete. That is, the separation could be determined by an exhaustive brute-force search(Dahlke, 2020).
- Even in the event that the former is achieved, the model will not extrapolate well to new data in the event of points near the boundary because there will be many functions that meet the criteria.

With the addition of a margin, we can then use a function such as the hinge loss function to maximize our boundaries and penalize misclassified points(Narkhede, 2018). This is ideal because maximizing the boundary is the same as minimizing the loss.

That is,

$$\max(0, 1 - y_i[w^T x_i + b]) = 0$$
$$\Rightarrow y_i[w^T x_i + b] = 1$$

$$\mathbf{x}^+ \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} - \mathbf{x}^- \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{1-b}{\|\mathbf{w}\|} - \frac{-b-1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

which leads us to the minimizing function(Narkhede, 2018):

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$

So, in nonlinear cases this would be a more ideal situation since it penalizes misclassifications, maximizes the boundaries so as to extrapolate better, and is ideal for higher dimensional data.

1.1.3 b. You have a choice of handling a binary classification task using (i) linear SVM, and (ii) perceptron algorithm. On what factors does your decision depend? Provide a formal explanation, supported by theorems and ideas presented in the readings associated with this topic.

The linear SVM is as follows:

$$\min \|w\|_2 + C\sum_{i=1}^{n}(1 - y_i(wx_i + w_0))_+$$

and the Perceptron is:

$$\min \sum_{i=1}^{n}(-y_i(wx_i + w_0))_+$$

so it's clear that the goal of them is similar from a mathematical perspective(xxx222, 2018).

The main difference is the addition of a regularization term and more efficient optimization for the SVM.

Some of the main considerations include

- Is the data linearly separable? This would apply to both, but would be important to see if either were a viable option from the start. This is the most critical point as datasets in the real world are seldom linearly separable(Gopal, 2020). Turns out that the perceptron algorithm fails here, hence it is seldom used in practice.
- How distant are the classes? Even if a dataset is linearly separable, the margins may be incredibly small. (Oh, n.d.).
- Do we need some lenience? Say we had one point that was super close to the decision boundary and within the margins, but it may have been an outlier. The perceptron algorithm may never converge if it cannot find a good boundary.

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

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 parame
(d) Describe the functionality of the regressor.
(e) Discuss the potential use of the regressor and quantify its accuracy.
```

[2]:
```
import numpy as np # for the linspace and sinc functions
import pandas as pd # cus dataframes are cooler than arrays
from sklearn.svm import SVR # Support Vector Regressor
from matplotlib import pyplot as plt # to look at the data
from sklearn import metrics
%matplotlib inline
```

[4]:
```
# We can use the linspace function to generate some easy data
# This will generate evenly spaced data between two endpoints. We can use this␣
↪as input for the sinc function.
X = np.linspace(-3, 3, 50)
```

So the sinc function is defined as

$$\frac{\sin(\pi x)}{\pi x}$$

Which is useful in Fourier transform signal theory(Weisstein, 2020). It's short for sine cardinal function, also called the 'sampling function.'

[5]:
```
# Let's use our data to generate the sinc data using the built-in numpy␣
↪function:
y = np.sinc(X)
print(y)
```

```
[ 3.89817183e-17  4.15114091e-02  8.03754744e-02  1.10560173e-01
  1.26741226e-01  1.25106539e-01  1.04020292e-01  6.44510924e-02
  1.00940913e-02 -5.28473225e-02 -1.16213079e-01 -1.70721641e-01
 -2.07001488e-01 -2.16703178e-01 -1.93561426e-01 -1.34277406e-01
 -3.91083073e-02  8.79177183e-02  2.39201032e-01  4.04176345e-01
  5.70268753e-01  7.24101450e-01  8.52825194e-01  9.45423163e-01
  9.93845462e-01  9.93845462e-01  9.45423163e-01  8.52825194e-01
  7.24101450e-01  5.70268753e-01  4.04176345e-01  2.39201032e-01
  8.79177183e-02 -3.91083073e-02 -1.34277406e-01 -1.93561426e-01
 -2.16703178e-01 -2.07001488e-01 -1.70721641e-01 -1.16213079e-01
 -5.28473225e-02  1.00940913e-02  6.44510924e-02  1.04020292e-01
  1.25106539e-01  1.26741226e-01  1.10560173e-01  8.03754744e-02
  4.15114091e-02  3.89817183e-17]
```

[6]:
```
# Now we need to generate some Gaussian noise to dirty up the signal.
mu = 0
sigma = 0.1
```

```python
# The data needs to be the same shape as the signal data
noise = np.random.normal(mu, sigma, [50, 1])

print(noise)
```

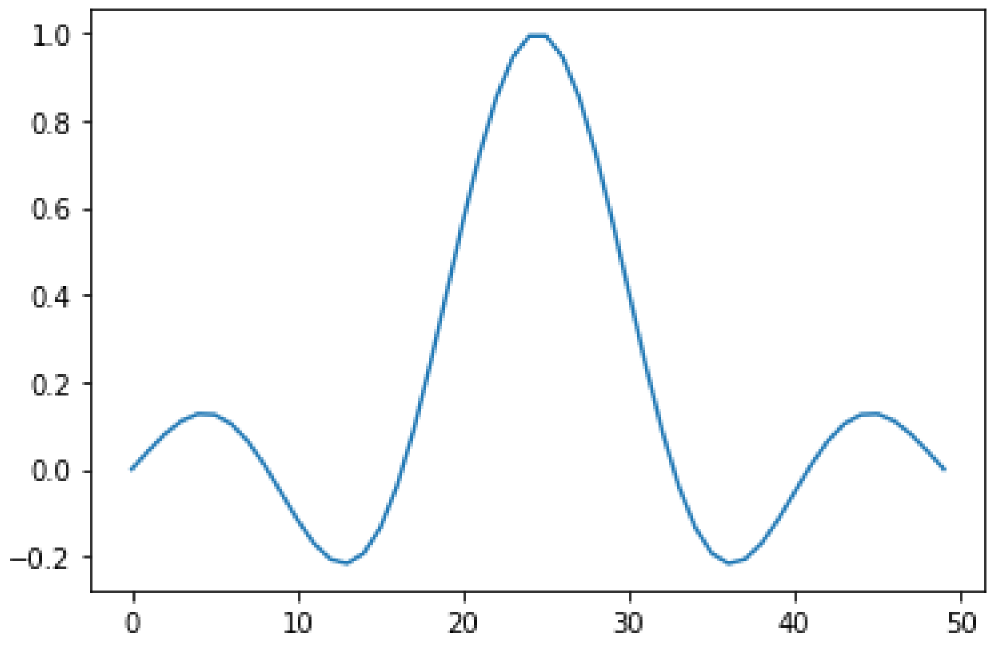
```
[[ 0.07409175]
 [-0.08493496]
 [-0.04550208]
 [-0.07889859]
 [ 0.11598632]
 [ 0.18964409]
 [ 0.10800038]
 [ 0.01926231]
 [ 0.01630129]
 [-0.10540829]
 [ 0.10272037]
 [ 0.22280725]
 [-0.02033732]
 [ 0.10196161]
 [-0.07604723]
 [ 0.06620133]
 [-0.05738235]
 [ 0.08030428]
 [-0.04578446]
 [-0.01205431]
 [-0.08108028]
 [ 0.12279236]
 [-0.03105089]
 [ 0.06472172]
 [ 0.04654086]
 [ 0.09019587]
 [-0.27975135]
 [-0.08120539]
 [-0.04973274]
 [-0.06031299]
 [-0.05941408]
 [ 0.06389396]
 [-0.00087846]
 [-0.043926  ]
 [-0.02509692]
 [-0.14424321]
 [ 0.04559541]
 [-0.02916041]
 [-0.04577316]
 [-0.00061845]
 [-0.01846713]
 [-0.17091548]
 [ 0.00627444]
 [-0.0267165 ]
 [ 0.0575752 ]
 [-0.15643914]
 [ 0.0819316 ]
 [ 0.00394676]
 [ 0.05980855]
 [ 0.18698279]]
```

[7]:
```
signal = pd.DataFrame(y)
```

[8]:
```
# Here is the data prior to the Gaussian noise added:

plt.plot(y)
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f611017bb10>]
```

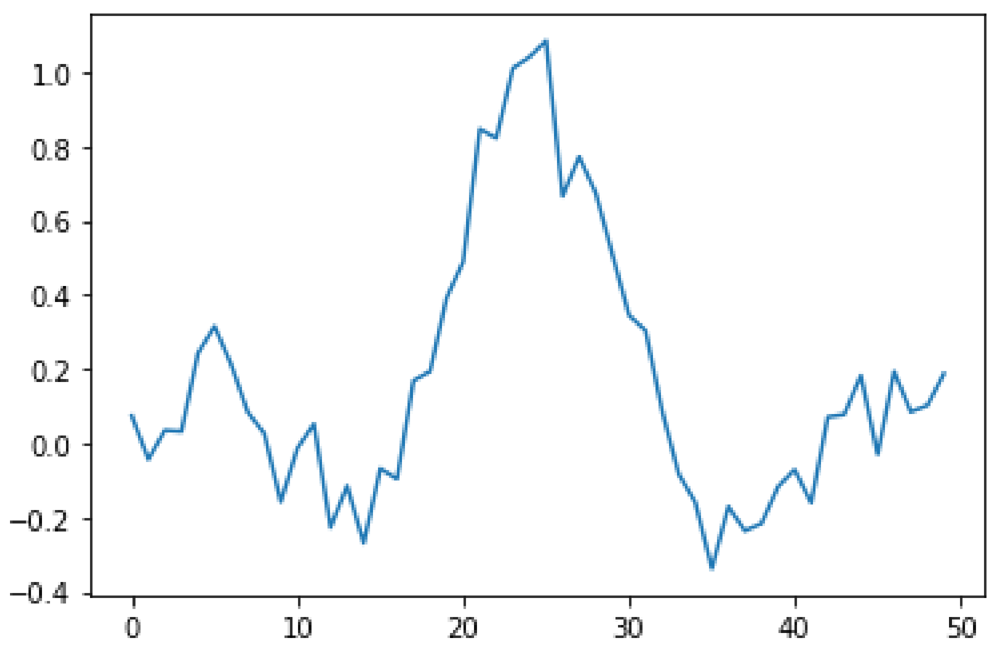


[9]:
```
# Add the noise to the signal
signal = signal + noise
```

[11]:
```
# Let's look at the data after the noise has been added:

plt.plot(signal)
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7f60b18a6ed0>]
```



```
[16]: X = X.reshape(-1, 1)
      np.shape(X)
```

```
[16]: (50, 1)
```

```
[20]: #signal.loc[0:,'X']

      signal.shape
```

```
[20]: (50, 1)
```

```
[14]: z.shape
```

```
            ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-14-dfe50837e4ae> in <module>
    ----> 1 z.shape


        NameError: name 'z' is not defined
```

The Support Vector Regression class has three main parameters, and one additional specific to the polynomial kernel.

SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line (or hyperplane in higher dimensions) to fit the data (Sharp, 2020).

The first value, "kernel", determines which kernel that we set. There are three main ones that are tried below. A kernel is a function that assists in moving the input space to a higher dimension to assist in creating a a better line fit or a hyperplane for discriminant analysis. There are three that are tried here:

- RBF. This is the default option for SVR. This kernel is defined as:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Which uses the squared Euclidean space and increases with distance and ranges between zero (Wikipedia, 2019).

- The next kernel is the linear kernel. Mathematically,

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$

with respect to the constraints

$$|y_i - w_i x_i| \leq \epsilon$$

- Poly Kernel: This particular kernel is interesting because it allows us to map our input space using combinations of our features as well as polynomials of them individually (Wikipedia, 2019).

$$K(x, y) = (x^T y + c)^d$$

where d is the degree of polynomial.

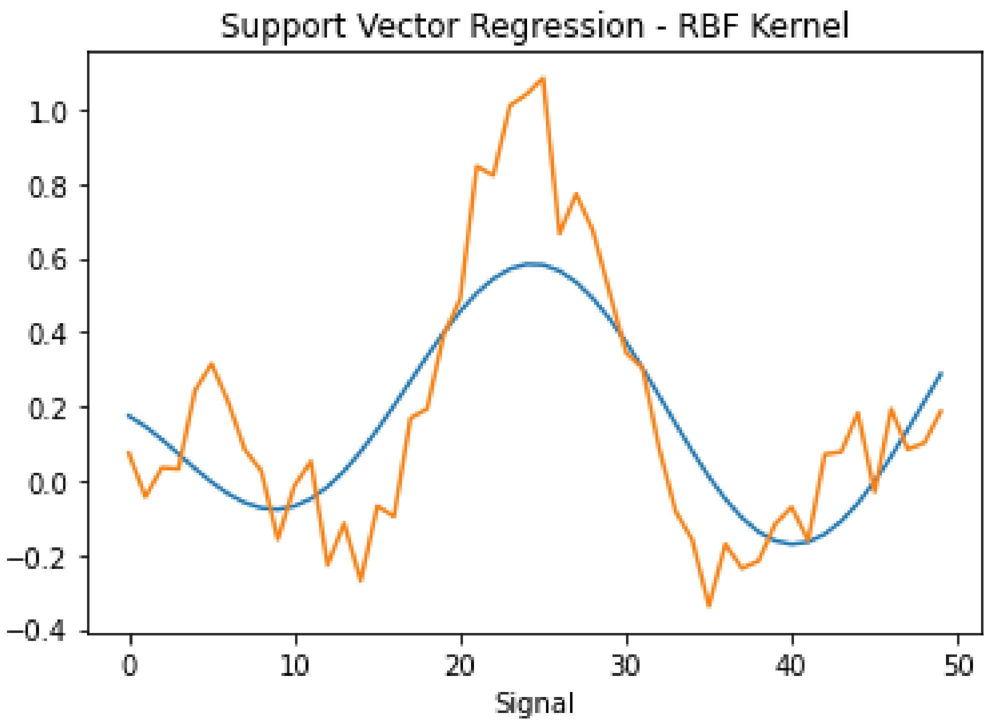
- Epsilon: This value is used to handle our maximum allowable error. The main benefit of the SVR model is that you can set the amount of error that is allowed. As opposed to OLS where the idea is to minimize error through coefficients, here it is a parameter.
- C parameter is also known as tolerance. As C increases, our tolerance for points outside of also increases. As C approaches 0, the tolerance approaches 0 (Sharp, 2020).
- gamma: Chris Albon (2017) puts the definition of this nicely: "gamma is a parameter of the RBF kernel and can be thought of as the 'spread' of the kernel and therefore the decision region. When gamma is low, the 'curve' of the decision boundary is very low and thus the decision region is very broad. When gamma is high, the 'curve' of the decision boundary is high, which creates islands of decision-boundaries around data points."

```
[27]: svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1).fit(X, signal[0])
```

```
y_hat = svr_rbf.predict(X)

plt.title('Support Vector Regression - RBF Kernel')
plt.xlabel('Signal')
plt.plot(y_hat)
plt.plot(signal[0])
```

```
[27]: [<matplotlib.lines.Line2D at 0x7f60b141ab50>]
```

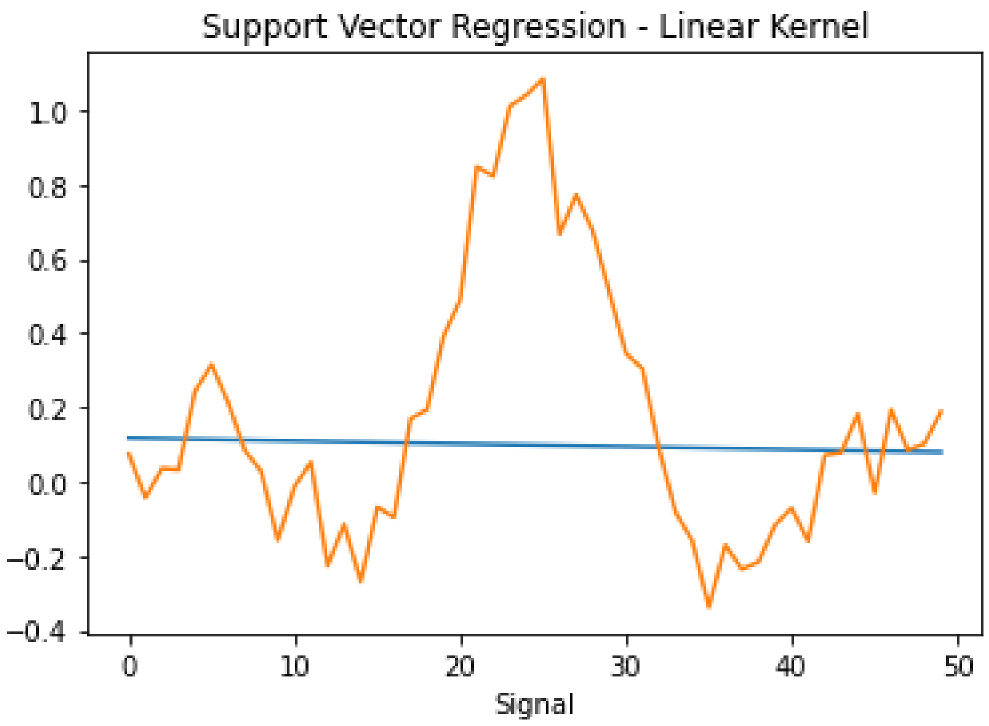


```
[26]: lin_rbf = SVR(kernel='linear', C=100, gamma=0.1, epsilon=.1).fit(X, signal[0])

      l_preds = lin_rbf.predict(X)

      plt.title('Support Vector Regression - Linear Kernel')
      plt.xlabel('Signal')
      plt.plot(l_preds)
      plt.plot(signal[0])
```

```
[26]: [<matplotlib.lines.Line2D at 0x7f60b13b3c50>]
```

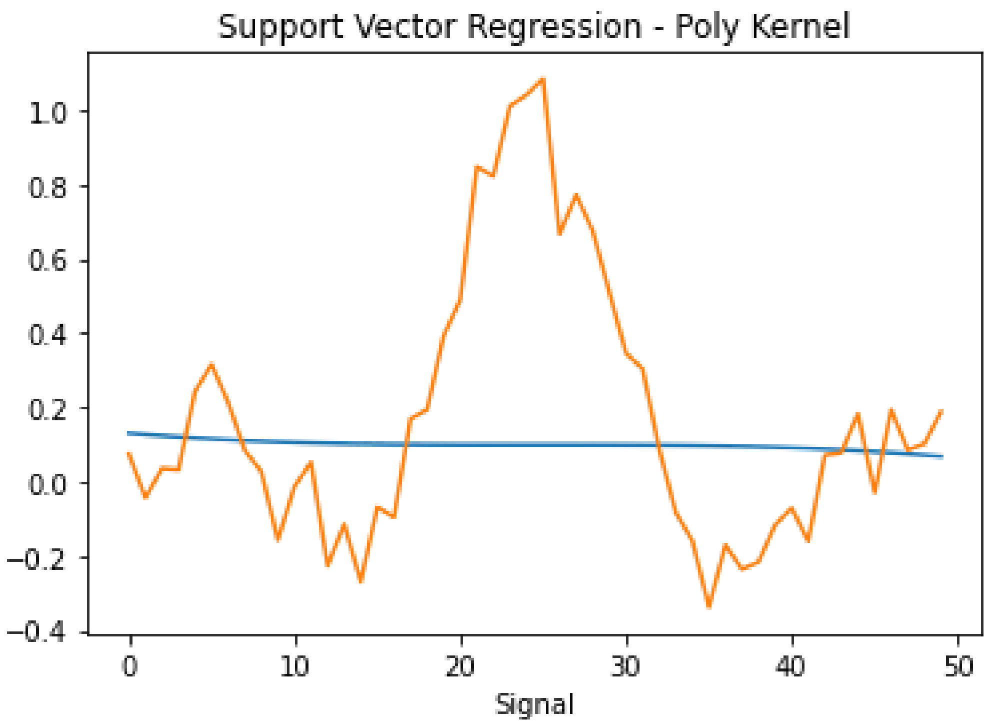


```
[25]: lin_rbf = SVR(kernel='poly', C=100, gamma=0.1, epsilon=.1, degree=3).fit(X,␣
       ↪signal[0])

      p_preds = lin_rbf.predict(X)

      plt.title('Support Vector Regression - Poly Kernel')
      plt.xlabel('Signal')
      plt.plot(p_preds)
      plt.plot(signal[0])
```

```
[25]: [<matplotlib.lines.Line2D at 0x7f60b1441f50>]
```



[35]:
```
print(f' RBF MSE:', metrics.mean_squared_error(signal, y_hat))

print(f' Linear MSE: ', metrics.mean_squared_error(signal, l_preds))

print(f' Poly MSE: ', metrics.mean_squared_error(signal, p_preds))
```

```
 RBF MSE: 0.04202061997234624
 Linear MSE:  0.1341288334747976
 Poly MSE:  0.13433581560667002
```

1.2 Conclusion

The RBF kernel was by far the best one used. This is due to the free parameter sigma, that allows high variance and lets us track the sinc function better than the other two. The default parameters were used and created the most acceptable function. With alterations to the parameters, this could certainly be made to a better fit.

The linear kernel was terrible, which was expected. The data is clearly nonlinear, so this was merely for example. It had an MSE of 0.13.

Lastly the Poly kernel was about the same as the linear kernel. I made several changes to the respective parmeters and poly degrees, but could not come up with anything close to RBF.

Overall the RBF is clearly the winner for a nonlinear SVR model.

References

Gopal, M. Applied Machine Learning. New York. McGraw Hill.

Dahlke, K. “NP-complete problems”. Math Reference Project. Retrieved 2020-09-27.

Narkhede, S. (2018). Understanding Confusion Matrix. Towards Data Science. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Liu, C. (2020). A Top Machine Learning Algorithm Explained: Support Vector Machines (SVM). KDNuggets. https://www.kdnuggets.com/2020/03/machine-learning-algorithm-svm-explained.html

xxx222 (https://stats.stackexchange.com/users/90896/xxx222) (2018). Difference between a SVM and a perceptron, URL (version: 2018-09-30). Cross Validated. https://stats.stackexchange.com/q/369480

Oh, S. (n.d.). Perceptron and SVM. University of Washington. https://courses.cs.washington.edu/courses/cse446/19au/7perceptron_annotated.pdf

Weisstein, Eric W. (accessed September, 2020). “Sinc Function.” From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/SincFunction.html

Hsia, J., Lin, C. (n.d.). Parameter Selection for Linear Support Vector Regression. National Taiwan University. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/warm-start/svr-param.pdf

Sharp, T. (2020). An Introduction to Support Vector Regression (SVR). Towards Data Science. https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2

Albon, C. (2017). SVC Parameters When Using RBF Kernel. Data Science & Machine Learning. https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/

Polynomial Kernel. (2019). In Wikipedia. https://en.wikipedia.org/wiki/Polynomial_kernel

Radial Basis Function Kernel. (2019). In Wikipedia. https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Loukas, F. (2020). Support Vector Machines (SVM) clearly explained: A python tutorial for classification problems with 3D plots. Towards Data Science. https://towardsdatascience.com/support-vector-machines-svm-clearly-explained-a-python-tutorial-for-classification-problems-29c539f3ad8

Crammer, K., Singer, Y. (2001). On the Algorithmic Implementation ofMulticlass Kernel-based Vector Machines. Journal of Machine Learning Research. https://jmlr.csail.mit.edu/papers/volume2/crammer01a/crammer01a.pdf

Loss Function. (2020). In Wikipedia. https://en.wikipedia.org/wiki/Loss_function

[ ]: