Assignment 4

Sophia J Quinton

Grand Canyon University

DSC-540: Machine Learning

Dr. Aiman Darwiche

24 November 2021

Assignment 4

Part 1

Question A

The misclassification error for binary problems, $y^i \in [0,1], and\ h(w, x^i) =$ $y^\iota \in [0,1]; \widehat{\ y} = 1, \dots, N$, is given by:

$Misclassification\ error = \frac{Number\ of\ data\ points\ for\ which\ (y^i - \widehat{y^\iota}) \neq 0}{N}$ (Gopal, 2019).

To choose a misclassification error as an accuracy measure works for class tuples when they are "more or less evenly distributed" (Gopal, 2019). If a class is rare, misclassification error is not a good measure of accuracy.

In a situation where the margin is maximized, and the misclassification performance is taken into consideration would best be solved with a soft SVM. A Soft SVM is a type of SVM that allows the points to fall into the decision boundary, but not cross the margin. Any data point that is located in the decision boundary has the loss calculated (Ritvikmath, 2020) (Gopal, 2019). The benefit of this is that the in real world problems, there is not always a clear and hard line between two classes. This type of algorithm allows the machine learning engineer to take into consideration the different classes but still account for the errors. There is a balance for classification mistakes and the large margin. "When c [number of mistakes] is small, classification mistakes are given less importance and focus is more on maximizing the margin, whereas when c is large, the focus is more on avoiding misclassification at the expense of keeping the margin small" (Misra, 2020). This balance is dependent on how far the misclassified points lie. The further from the correct class, the greater the error.

$L = \frac{1}{2}||w||^2 + C(\#\ of\ mistakes)$ w represents the margin width and c represents the number of mistakes. "Here the hyperparameter that decides the trade-off between maximizing the margin and minimizing the mistakes" (Misra, 2020). This equation can be expanded out to take into consideration the location of the misclassified data points. $L = \frac{1}{2}||w||^2 + C\sum_i \xi_i + \sum_i \lambda_i(y_i(\vec{w} \cdot \vec{x_i} + b - 1 + \xi_i)$ (Misra, 2020). This takes into consideration the loss function. This works by this simple explanation:

- If the correct data points are classified outside the support vectors, then their loss is zero. Max(0, 1± $y_i(\vec{w} \cdot \vec{x_i} + b)) = 0$ (Ritvikmath, 2020)
- If the data is located inside the decision boundary, is given a slight loss: Max(0, 1± $y_i(\vec{w} \cdot \vec{x_i} + b)) = (between\ zero\ and$ 1) (Ritvikmath, 2020)
- If the wrong data points are classified outside the support vectors, then their loss is a lot larger. Max(0, 1± $y_i(\vec{w} \cdot \vec{x_i} + b)) > 1$ (Ritvikmath, 2020)

Overall, the decision depends on how the data is separated. The goal of a soft SVM is to minimize the misclassification but keep the margin large. It is dependent on the spread of the misclassified points. It is a balance between these two decision points. If one were to solely focus on misclassification, then this might cause the testing dataset to not behave as accurately. The margins can allow for better classification but can be misclassified. A way to balance these two issues is to use the soft SVM.

Question 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.

This question is similar to question A, but it takes into consideration the two methods as different algorithms. The Perceptron algorithm is a type of neural network algorithm that "takes a vector of real-valued inputs, calculates a linear combination of these inputs; then outputs +1 if the result is greater than the threshold and -1 otherwise" (Gopal, 2019). Some of the downfalls of the perceptron algorithm is that is strictly for linear problems, and many questions are nonlinear. There is one way to overcome the nonlinearity of the perceptron algorithm, and that is to change the way the coordinates are assigned. An example would be from going to polar coordinates from cartesian coordinates (Ritvikmath, 2019).

The issue of nonlinear data makes the perceptron function difficult, so the SVM is better in this case. The SVM can transform the data using the kernel trick (Zvornicanin, 2021). This algorithm and technique are shown to help reduce computational input and help assign data better. In terms of probability and prediction, "SVM model doesn't output probability natively," but can through Platt scaling, and the perceptron "directly predicts ot probability for each class" (Zvornicanin, 2021). It all depends on what the machine learning data scientist is investigating.

For binary data, SVMs have the margin that can be useful for better accuracy of predictions. The perceptron will classify data accordingly to a single line: $g(x) = w^T x + w_0$ (Gopal, 2019). It does not utilize any margins to help classify the data. The issue appears when the separation of the classes is not as clean. Soft SVMs will be able to take loss into consideration if this is the case because of the margin. Another issue is if the classes are far apart. For the perceptron, there are many different options, but with an SVM, the margins help choose a more successful separation (Kharel, 2020). See figure one for the visual

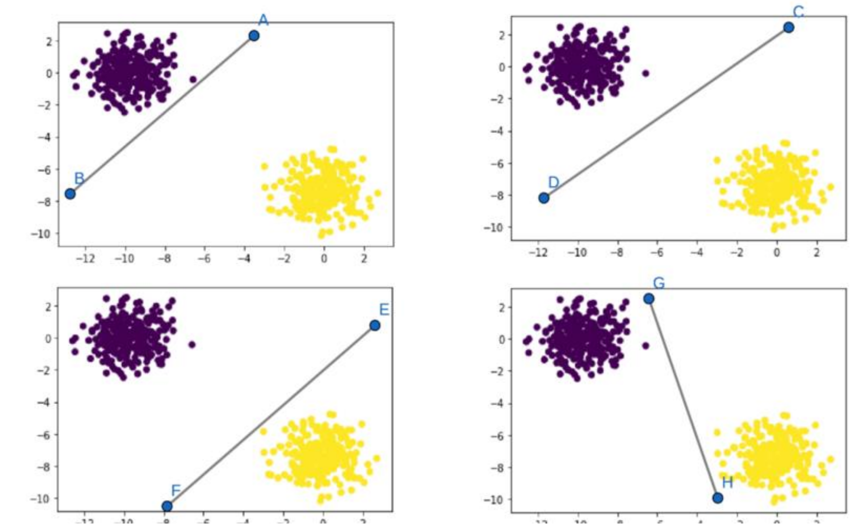


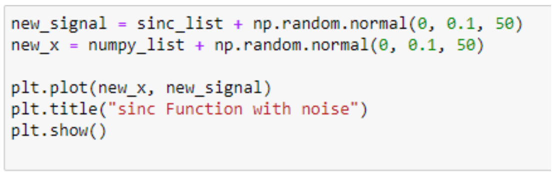
Figure 1: (Kharel, 2020)

comparison. This margin aspect is useful in terms of the accuracy of the testing data. Using figure one, if AB is chosen, then the testing dataset might accidently class to the purple group. Overall, SVM limits the different types of linear classifiers because it focuses on maximizing the margins while limiting the error. Perceptron relies on initial weights, while SVM is more robust to the real boundary (Kharel, 2020).

In conclusion, SVMs are the superior choice when it comes to real life data situations because of the kernel trick, and that it can help balance the margin to misclassification error. Perceptron is useful as a portion of the neural network algorithms. It has been heavily upgraded o different forms to help with neural network classification (Gopal, 2019). It is also useful when probability of prediction is needed.

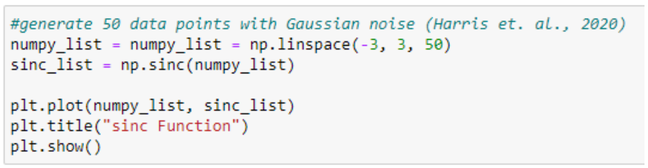
Part 2

Generate 50 data points and then add gaussian noise



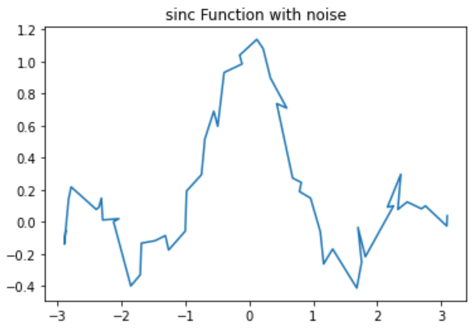
```
new_signal = sinc_list + np.random.normal(0, 0.1, 50)
new_x = numpy_list + np.random.normal(0, 0.1, 50)

plt.plot(new_x, new_signal)
plt.title("sinc Function with noise")
plt.show()
```



```
#generate 50 data points with Gaussian noise (Harris et. al., 2020)
numpy_list = numpy_list = np.linspace(-3, 3, 50)
sinc_list = np.sinc(numpy_list)

plt.plot(numpy_list, sinc_list)
plt.title("sinc Function")
plt.show()
```



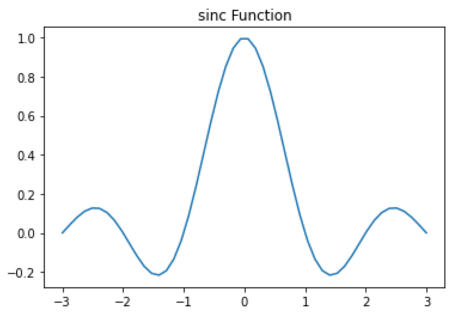


Figure 2: generated the two datasets

To setup the data, I used the numpy library (Harris et. al., 2020). The sinc function is already defined in the numpy library, so I used that dataset. Then I added Gaussian noise by following the instruction in the numpy library. I added the noise to both the x and y arrays.

Going forward, per the instructions, I used the data with the Gaussian noise for the training set, then I used the stand sinc function to prove whether the data is accurate for the different regressor models.

Train SVM regressor

Four different models were used to show the different models and to prove which model is the more accurate model. The models were the RBF kernel, the RBF kernel with a different gamma, a polynomial kernel, and a linear kernel. The data with Gaussian was used for the training of the model in each of these instances. The x data was used to predict and show the accuracy of each of the models.

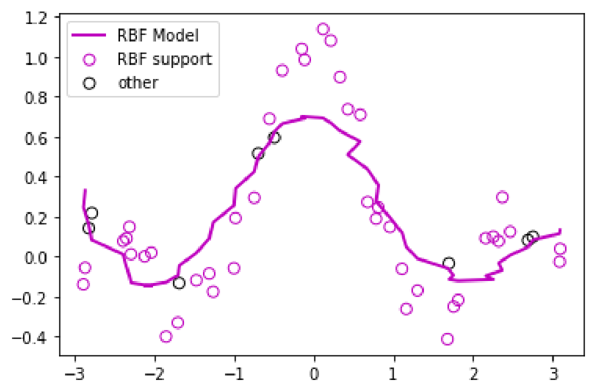


Figure 4: SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)

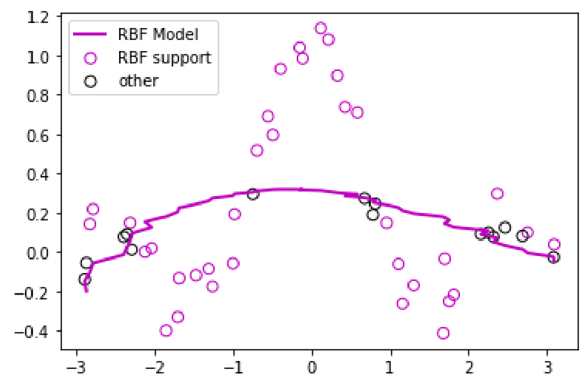


Figure 3: SVR(kernel="poly", C=100, gamma="auto", degree=3, epsilon=0.1, coef0=1)

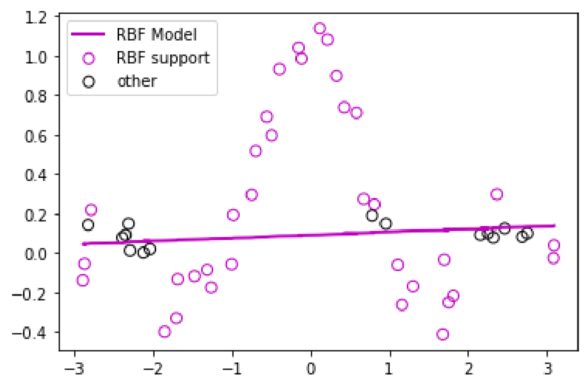


Figure 5: SVR(kernel="linear", C=100, gamma="auto")

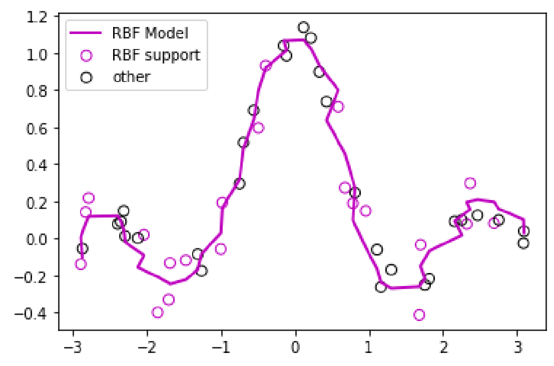


Figure 6: SVR(kernel="rbf", C=50, gamma=0.5, epsilon=0.1)

Model	**MAE**	**Root MSE**
RBF	0.1582	0.1816
Poly	0.2622	0.3218
Linear	0.2566	0.3678
RBF - 2	0.05403	0.0609

The functionality of each of the regressors is based on the structure of the data. Looking at the four graphs of the different models, the linear and polynomial functions are not successful at predicting the data. The linear regressor kernel is based off of the equation: $K(x1, X2) = X1 \cdot X2$ (Pedamkar, 2021). The linear function would not fit the structure of this data and would poorly predict the data. The accuracy of this regressor is shown in the table. The linear has both a high MAE and a root MSE. A successful model will have MAE and root MSEs closest to zero where the data and the prediction is the same or the closes (Larose & Larose, 2019). The polynomial kernel is also an unsuccessful regressor for the sinc function as it is not close to prediction the function according to the graph. The polynomial kernel equation is $K(X1, X2) = (X1 \cdot X2 + 1)d, where\ di\ is\ the\ degrees$ (Pedamkar, 2021). Here the function is not quite polynomial, so it does not fit this type of regressor. The accuracy is shown in the table, where it is still too large. The next two functions are the radial basis function. The equation is $K(x1, X2) = exp(\frac{||X1 - X2||^2}{2\sigma^2})$ (Sreenivasa, 2021). The RBF kernel works by computing “the similarity or how close they are to each other” (Sreenivasa, 2021). A simple RBF kernel looks similar to a gaussian normal curve. This is very similar to the sinc function. The changes within the function are C and gamma. C is used to control error and the gamma or 1/sigma is used to control variance of the data (Kumar, 2018). If C is high there will be many errors. If gamma is too high, overfitting occurs. In the above graphs the second RBF has too high of a gamma, and there is overfitting. Even though this function is extremely accurate with a very low MAE and root MSE, it would overfit the data. Overall, the first RBF is the most successful.

It is accurate without overfitting the data. RBFs are extremely versatile to the function they are trying to fit.

To confirm the results, I switched what I had labeled as the training data above as the testing data. The results here also prove that there was a disconnect with mostly all the models except the RBF one that was not overfitted.

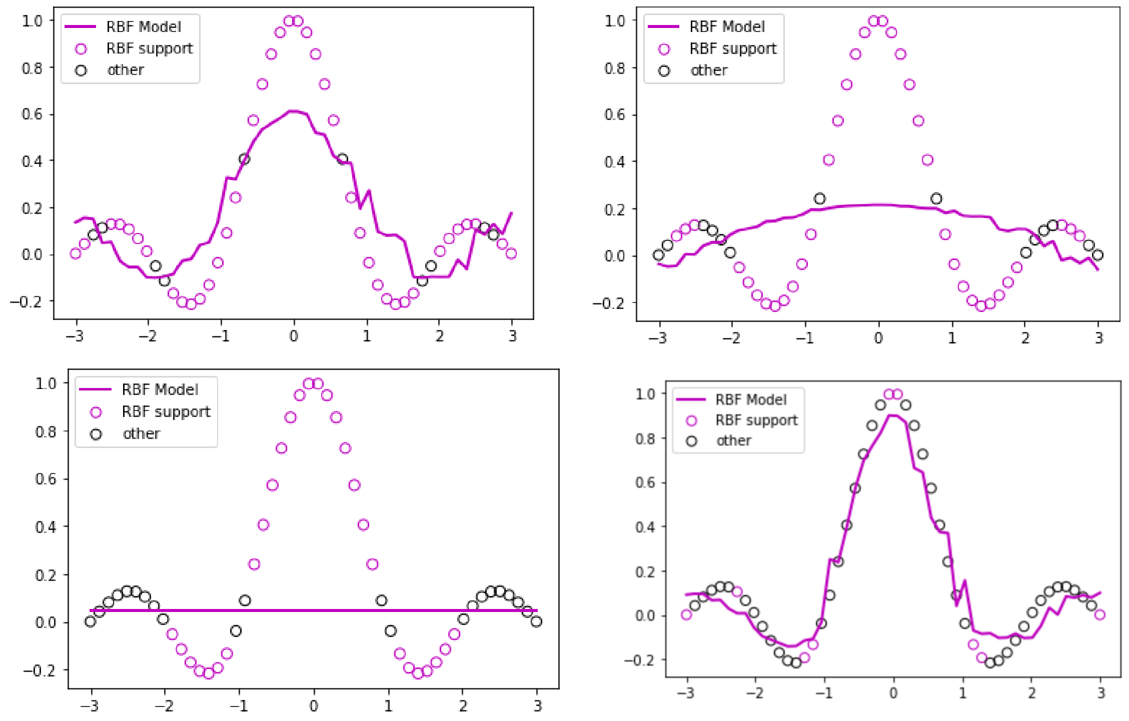


Figure 7: Reverse testing of previous figure

Model	MAE	Root MSE
RBF	0.1901	0.2296
Poly	0.2786	0.3611
Linear	0.2826	0.4055
RBF - 2	0.1244	0.1464

Here there is a comparison between the two reports. Looking at the MAE and the root MSE, there is a stronger relationship with the RBF function.

Article Discussion

The article has a defined sinc kernel that it is used to helps with SVMs of different parameters. The problem with kernels is that they often need to be tested to find the correct variation of the data. The authors show that the sinc function “is shown that a principled and finite kernel hyper-parameter search space can be discerned, a priori” (Nelson et. al., 2009). The article outlines the results by comparing different functions to with sonic hertz data (Nelson et. al., 2009). As the harmonics change, there is a change in amplitude. The sinc kernel can take into consideration the variations in harmonics. The sinc kernel is a Mercer kernel (Nelson et. al., 2009). The kernel functions that are defined above are based off of the Mercer kernel. The Mercer’s Theorem “provides a test whether a function $k(x^i,x^k)$ constitutes a valid kernel without having to construct the function” (Gopal, 2019). So just like the above functions, the sinc kernel follows this rule.

This relates to the analysis in the earlier parts of this report because to find an accurate kernel, several different variations needed to be tested to come across the most accurate kernel (Nelson et. al., 2009). The authors create a kernel based on the function used above.

References

Gopal, M. (2019). *Applied machine learning*. McGraw-Hill Education.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. doi:10.1038/s41586-020-2649-2

Kharel, S. (2020, May 13). *Perceptron vs SVM: A quick comparison*. Medium. Retrieved November 21, 2021, from https://medium.com/@subashkharel/perceptron-vs-svm-a-quick-comparison-6b5d6b5d64f.

Kumar, A. M. (2018, December 17). *C and gamma in SVM*. Medium. Retrieved November 21, 2021, from https://medium.com/@myselfaman12345/c-and-gamma-in-svm-e6cee48626be.

Larose, C. D., & Larose, D. T. (2019). *Data science using Python and R*. Wiley.

Misra, R. (2020, June 7). *Support vector machines‐soft margin formulation and kernel trick*. Medium. Retrieved November 21, 2021, from https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe.

Nelson, J. D. B., Damper, R. I., Gunn, S. R., & Guo, B. (2009). A signal theory approach to support vector classification: The SINC Kernel. *Neural Networks*, *22*(1), 49–57. https://doi.org/10.1016/j.neunet.2008.09.016

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pedamkar, P. (2021, March 3). *Kernel methods: Need and types of kernel in machine learning*. EDUCBA. Retrieved November 22, 2021, from https://www.educba.com/kernel-methods/.

Ritvikmath. (2020 Nov. 30). *Soft Margin SVM: Data Science Concepts* [Video]. YouTube. https://www.youtube.com/watch?v=IjSfa7Q8ngs&t=643s

Ritvikmath. (2019 Jan 20). *Perceptron* [Video]. YouTube. https://www.youtube.com/watch?v=4Gac5I64LM4

Sreenivasa, S. (2020, October 12). *Radial basis function (RBF) kernel: The go-to kernel*. Medium. Retrieved November 22, 2021, from https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a.

Zvornicanin, E. (2021, October 17). *Difference between a SVM and a Perceptron*. Baeldung on Computer Science. Retrieved November 21, 2021, from https://www.baeldung.com/cs/svm-vs-perceptron.

Github link: https://github.com/squinton-gcu/Data-Science/tree/main/DSC-540/Assignment4