Part 1

1. Suppose we have an arbitrary binary classification task, and we wish to decide whether to use the number of misclassifications or the margin between the two classes as our performance measure. Both of these are have many situations where their application is appropriate, and they even overlap at many points.

We shall begin by diving into the first measure, total misclassifications. We recall that a general confusion matrix is defined as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Predicted | | | | | |  |
|  |  |  | 0 |  | | 1 |  | Total |
| Actual: | 0 | True Negatives | | | False Positives | | | Total Negative |
| 1 | False Negatives | | | True Positives | | | Total Positive |
|  | Total Negative | | | Total Negative | | | **Grand Total** |

\begin{figure}\begin{pspicture}(0,1)(9,9)
\psset{arrowscale=2,dotsize=6pt}
\psli...
...\\ maximized}}}
\pnode(5.5,3.5){C}
\ncline{->}{D}{C}
\end{pspicture}\end{figure}Where we traditionally abbreviate the four central boxes as . Therefore, the number of misclassifications is simply the sum of the false results, or . It is easy then to see why this measure would be of interest in terms of optimization: we merely want to of iterations where the classifier is . These are known in statistics as Type I () and Type II () errors. Therefore, this goal can be restated as saying we want to minimize the error rate of our classifier. This can be utilized in numerous cases, since it is a general use method of estimating the predictive power of the model.

Now consider the measure of maximizing the margin between classes as how we evaluate the performance of a classifier. In the binary case, this means we are searching for the according to some criterion. The Support Vector Machine (SVM) algorithm, in particular, defines this criterion to be looking for a decision surface that is maximally distant from any one data point. For example, consider the graph to the right, where the solid line represents this optimal separator:

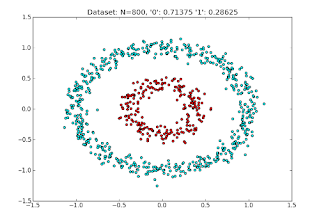
This line is called a *linear discriminant function*, defined as , categorizes the -dimensional space for the input into two regions, and mathematically finds the maximum distance between these regions. It is therefore an optimization problem, since there exist many lines between the data but we would like the “best” one.

Determining which of these to choose for our problem depends on the nature of our data and the task at hand. We are only given that the classification problem is binary, but are given no further information on the sizes of the groups. If the class sizes are similar, error reduction can be a great measure of power. If class A is much more prevalent than class B, then reduction in error rate may not be our best goal in mind. This is because the class imbalance causes a very skewed confusion matrix, meaning that either precision or recall would be better suited for the problem. For example, suppose the problem is screening for a rare type of cancer. Since the cancer is rare, and being placed into class A represents a negative test, then we would expect the size of class A to be far higher than that of class B, the positive tests. In this case, the actual goal at hand would likely be to reduce the occurrences of false negatives, i.e., the cases where someone is diagnosed as healthy when they actually require treatment. While accuracy is a good measure, relying on reducing the number of misclassifications alone would be inappropriate here.

On the other hand, using maximized class margins as the performance measure can be utilized to improve the robustness of the model. This is especially true for classifying extreme values, as a perfectly linear separable model can be very useful. However, there are cases were cautioned should be used. In particular, if the data are not linearly separable, then maximized margins could cause problems of overfitting. SVMs often utilize kernel functions to avoid this, but they need to be carefully picked. In addition, for data with high numbers of parameters relative to the size of the data, maximizing the margin can lead to very high error in the testing set because the volume is increasing fast enough for the data to become sparse, leading to difficulty detecting patterns across the high numbers of dimensions. In cases like this, where parameters can reach into the thousands and the number of data points may even be lower than this, maximizing margins should be used cautiously or even not at all.

1. Now suppose we have our arbitrary binary classification task, but instead we wish to decide whether to use linear SVM or the perceptron algorithm as our performance measure. To recall, perceptron algorithm tries to find some line that will classify every data point correctly. It begins by choosing a learning rate (usually small values, like 0.1 are chosen) and some initial weights (), then proceeding to pick observations that have positive error (meaning they have been misclassified) and update each time like so:

Where is the radius of the hypersphere of the training data, centered on the origin, and sets the step size. This method is used to try to find the assumed line that perfectly splits the data into its two classes. SVMs as defined in part A use the method of trying to maximize the distance between the two classes as its criterion function.   
  
One key restriction of both these methods is that it requires that the data be linearly separable. That is, data such as in the graph below would need to be modelled using another method:



One key difference between these two is their stoppage criterions. Perceptron will stop after it has found one line that serves as a perfect separator, whereas SVM will continue until it has found the of these lines. In the example below, the **blue** line might represent the SVM, since it maximizes the distance between the two classes, but the **orange** perceptron lines are created simply by stopping once any line, not necessarily the one maximally distant, is created.

However, this does not directly imply that SVM will always be the optimal method. In particular, perceptrons are computationally cheap and can be trained online and in real time as examples arrive, whereas SVMs are more expensive and cannot be regenerated as data comes in. So if the binary classification task at hand is based on data that is constantly changing, like customer interaction, churn prediction and advertising displays, then the perceptron algorithm would be preferable. If the data is static and does not need to be tracked and updated constantly, such as spam email detection, loan applications or insurance risk, then the more optimal SVM would likely be the preferred method.

**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:

The regressor (implemented in the Jupyter notebook corresponding to this report) implements the training of random points generated from this above function. The regressor itself estimates a hyperplane that can be used for regression, much like the simple linear regression format. Much like in standard vector machines, in this type of regression a hyper-region is created to mimic the “soft margin” where errors are not penalized in order to account for extreme values carrying undue weight on the decision surface. In this example, 50 points from the range were created and the supervised machine learning model carried out using the svm.SVR method from the sklearn package. This function requires a few parameters to be defined before it can be carried out.

We require first and foremost to define a kernel function. To recall, the kernel “trick” as it is sometimes called manipulates a decision surface that is not linearly separable into a linear equation in more dimensions so that it can be used by an SVM. There are many types of kernel functions, but in our example, we define it to be the radial basis function (RBF), or . Here, is a float that can be passed to the gamma argument. This value, which is by default set to (with being the number of features), is the free parameter and is inversely related to the variance and influence of the support vector. If is small, this implies large variance and vice-versa. The notebook implements two graphs to show the impact that has on the accuracy of the regressor, showing that in this case resembles the plotted points much more closely than that of , even with the relatively small sample size (.

We also use a C parameter in the SVR function. This is called the regularization parameter and arises from the need to create a “soft margin” on SVM’s to account for extreme values affecting the decision surface. Therefore, C is the parameter regulating how much influence each individual support vector has, helping to make the margin more stable. Whereas the range of is constrained to , the range for C is far wider and the scikit-learn manual demonstrates an example range of ( where the value is usually tweaked by powers of 10 (i.e., ). Examples in the Jupyter notebook demonstrate how a lower C value decreases the accuracy of fit, whereas a large enough C appears to offset the small choice.

Both of these variables must be chosen carefully to minimize the risks of making the model too strong or weak. A or C too large will make the regressor prone to overfitting, whereas if it is too small it will risk underfitting.

The nature of the sinc function is that, due to its calculation, it highly resembles a sinusoid wave but experiences very high frequency as in both directions. It is used in many engineering applications, such as electrical signal processing. Our regressor found a mean squared error of , which is appropriate given the small sample size and we wish to avoid overfitting. In the article by Nelson et al. (2009), the authors define the sinc kernel rigorously through the use of Fourier transformations and Hilbert spaces in engineering signal theory, then demonstrate its use for finding parameters in specially defined search spaces. They use the sinc kernel as their example to show that signal theory can be used to find the nature of the information model in SVM classification They also find an error percentage of about 4% for the sinc function in a four class problem, and about 11% for a sixteen class problem, thus comparing favorably with our simple model given the smaller sample size.

References:

Afonja, T. (2018, June 17). *Accuracy Paradox - Towards Data Science*. Medium. https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b

Brownlee, J. (2019, June 20). *Classification Accuracy is Not Enough: More Performance Measures You Can Use*. Machine Learning Mastery. https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

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

Nelson, J. D., 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

*sklearn.svm.SVR — scikit-learn 0.24.1 documentation*. (n.d.). Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Zhai, S. (2013). *Direct 0–1 Loss Minimization and Margin Maximization with Boosting*. Wright State University Department of Computer Science and Engineering. https://papers.nips.cc/paper/2013/file/9461cce28ebe3e76fb4b931c35a169b0-Paper.pdf

Zhang, J. (2013). *Perceptron and Linear SVM*. University of Purdue. https://www.stat.purdue.edu/~jianzhan/STAT598Y/NOTES/slt10.pdf