Classifying Breast Tumors using Support Vector Machines

Erik Carlson and Will Fletcher

Abstract

This paper discusses the usage of support vector machines (SVMs) to classify breast tumors as malignant or benign using data from the nuclei of the tumor cells. We begin with a discussion of what an SVM is. We then set out to experiment with different hyperparameters and kernels to optimize for our purpose. In particular, we create models using polynomial and radial kernels, and get testing accuracies of 92.3% and 98.6% respectively.

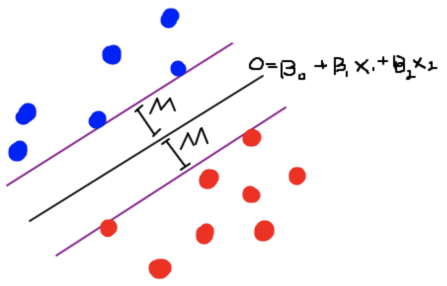
In this paper, we aim to learn about and implement support vector machines (SVMs), and use them to classify breast tumors as malignant or benign based on measurements of the nuclei of the tumor cells. A malignant tumor is one that is cancerous and a benign tumor is one that is not. While benign tumors can be harmful, malignant ones are particularly dangerous, so it is important to be able to differentiate the two. We will be using a dataset from the University of Wisconsin[1] containing information about the cells from breast tumors. For each patient, the researchers conducted a non-invasive, fine needle aspirate (FNA) of the breast mass.[2] For each cell in the sample, 10 real-valued features are computed such as radius and texture. The researchers then computed the mean, standard error, and maximum value for each of these features for a total of 30 features in the dataset. In total, the data contains 569 rows, of which 357 are benign and 212 are malignant. We normalized the data so that each feature has a mean of 0 and standard deviation 1. We then split the data into 50% training, 25% validation, and 25% test data. We used all features included in the dataset, and did not create any additional features.

There has been a good amount of research done to try and classify tumor cells using machine learning algorithms. On this dataset in particular, people have attempted to use logistic regression[3] and linear programming-based machine learning models.[4] Both of these papers had strong results with cross-validation accuracies of 96.2% and 97% respectively. The first of these papers introduced a 3rd classification which gave an unsure classification to data points that it found had a 30-70% chance of being malignant. This allows potential real-world users of this model to reconsider patients who the model has a hard time evaluating.

The model we are using, the support vector machine, is a supervised learning model. They are typically used for binary classification, although can be modified to classify between more classes. We begin by discussing a simplified variety of SVMs called a maximum margin classifier. Suppose we have N training data points with d dimensions. A maximum margin classifier attempts to completely separate the training data based on their labels using a hyperplane. A hyperplane is the set of solutions, x, to the equation $\beta_0+\beta_1(x_1)+...+\beta_d(x_d) = 0$ where the β_i are real valued parameters of the hyperplane. We assume that not all β_i equal 0 and often the hyperplane is normalized by insisting that $\Sigma_{i=1}\beta_i^2 = 1$. If d is one, two, or three, then a hyperplane is a point, line, or plane respectively. Since there can be infinitely many hyperplanes

[1] Wolberg, William, et al. "Breast Cancer Wisconsin (Diagnostic) Data Set."
[2] Ibid
[3] Wolberg, W H. "Computer-Derived Nuclear Features Distinguish Malignant from Benign Breast Cytology."
[4] Mangasarian, Olvi L., et al. "Breast Cancer Diagnosis and Prognosis Via Linear Programming."

that separate the data, the SVM chooses the one that maximizes the minimum distance from the hyperplane to any of the data points. This distance, denoted M, is known as the margin. Choosing a separating plane this way is called using a hard margin. Once we have found the optimal β_i we can define a function $f(x) = \beta_0+\beta_1(x_1)+\ldots+\beta_d(x_d)$. Then, to classify a new point, x^*, we simply plug it into f. If $f(x^*)$ is greater than 0, then we classify it as 1 and if it is less than 0, we classify it as -1. At exactly 0 we must choose one or the other.

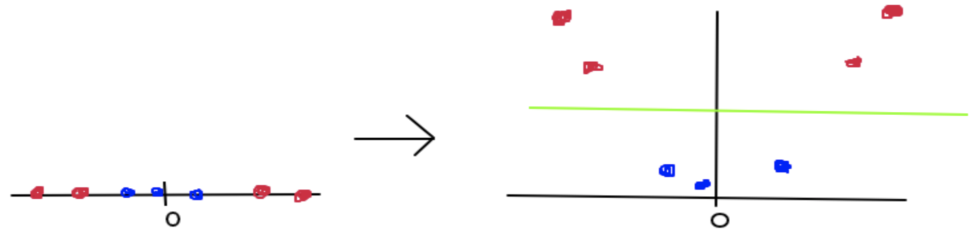


The reason for choosing the classifier with the largest margin is that it separates the buffer zone between the data equally. However, there are some downsides to using a hard margin. There will often not exist any hyperplane that perfectly separates the data, making a hard margin impossible to use. Also, the optimal hyperplane only depends on a small subset of the training data. Data points that are on the margin are called support vectors. Moving a non-support vector will not change the optimal hyperplane unless it is moved to within the margin. The over-reliance on a small amount of data points leads to high variance when using a hard margin.

To help address these issues, SVMs often use a soft margin which allows some of the training data to cross the margins. For each data point, x_i, we introduce a slack variable, ϵ_i, that says how much that data point can violate the margin. If ϵ_i equals zero, then the ith data point is on the correct side of the margin. The slack variables must all be non-negative and sum to no more than C, where C is a hyperparameter called the regularization parameter. A soft margin with C = 0 is equivalent to a hard margin. The higher the value of C, the more that the data can violate the margin. Data points on or past the margin are the support vectors. The SVM uses the hyperplane with the largest margin while respecting all the necessary conditions. Finding the correct hyperplane is equivalent to solving the following optimization problem:

$$\begin{aligned}
&\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} \quad M \\
&\text{subject to } \sum_{j=1}^{p}\beta_j^2 = 1, \\
&y_i(\beta_0+\beta_1x_{i1}+\beta_2x_{i2}+\ldots+\beta_px_{ip}) \geq M(1-\epsilon_i), \\
&\epsilon_i \geq 0, \ \sum_{i=1}^{n}\epsilon_i \leq C,
\end{aligned}$$

The SVMs that we have described up to this point are linear classifiers. However, not all data can be reasonably classified by a hyperplane. For example, consider the data on the left side of the figure below. There is no single point that can separate the red and blue data points. However, if we add a second feature, the original feature squared, we get data on the right side



which is separated by the green line. When we pull that classification back to the original feature space, it is no longer linear and is able to classify the data.

In general, by transforming the feature space by adding new features based on the old ones, we are able to get a non-linear classifier in our original feature space even though we use a hyperplane in the transformed space. However, training SVMs is computationally intensive and so adding more features is often not practical. Additionally, there are some geometric properties that we might be interested in that can not be described by a finite number of new features. To get around these issues, we use a kernel function which allows us to do computations in our original feature space but use the geometry of a transformed feature space.

To be able to use kernel functions, instead of using an f of the form $f(x^*) = \beta_0+\beta_1(x^*_1)+...+\beta_n(x^*_n)$ we express it as $f(x^*) = \beta_0+\Sigma_{j=1}^N\alpha_j(x^*,x_j)$ where x_j is the jth training data point and (x^*,x_i) is the dot product. The α_j are called Lagrange multipliers and they sum to 0. If we expand the second version of f(x) we can see that it is a linear combination of $x^*_1, \dots, x^*_n$ and thus we can express the β_i in terms of the α_i and vice versa. A kernel function is a generalization of the dot product and is written as k(x,y). To classify a point using a kernel function, we simply replace the dot product with the kernel function in f to get $f(x^*) = \beta_0+\Sigma_{j=1}^N\alpha_jk(x^*,x_j)$. Sometimes, the label is factored out of the Lagrange multiplier and we get $f(x^*) = \beta_0+\Sigma_{j=1}^N y^i\alpha_jk(x^*,x_j)$ where the y^i equals one or negative one. In the second case, the Lagrange multipliers are all non-negative. In this way, we can view the sign of α_j as determining how the jth training datapoint influences the classification given by f. If $k(x^*,x_j)$ is positive, then x_j contributes to classifying as y^i and if it is negative then x_j contributes to classifying as the opposite of y^i.

The idea behind a kernel function is that it is equivalent to transforming two data points to a higher dimensional feature space and then taking the dot product there. One important example of a kernel function is the polynomial kernel of degree p which is defined as $k(x,y) = (1 + (x,y))^p$. Note that the polynomial kernel of degree 1 is equivalent to using the dot product in the original feature space. For degrees p greater than 1, it is equivalent to adding a new feature for each product with multiplicity p or less of original features. These features are multiplied by a constant so that when the dot products are taken in the transformed feature space, it simplifies to $(1 + (x,y))^p$ as desired. Another important kernel function is the radial basis function kernel, also called the radial kernel or rbf kernel. The radial kernel depends on a parameter, $\gamma > 0$, and is defined as $k(x,y) = \exp(-\gamma\, L_2(x,y)^2)$ where $L_2(x,y)$ is the L-2 norm. When classifying a point, $k(x^*,x_j)$ decreases exponentially the farther x^* is from x_j. The larger γ is, the less influence far away points have on classification. In this way, the radial kernel considers points close to the point it is trying to classify and classifies it based on the local behavior of the training data. The radial kernel is an example of a transformed feature space that can't be expressed using finite additional features.

When we train an SVM our goal is to find the optimal Lagrange multipliers. In practice, training an SVM is usually done by solving the following optimization problem subject to the constraints:

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} y_i y_j k(\vec{x}_i,\vec{x}_j)\alpha_i\alpha_j,$$
$$0 \le \alpha_i \le C, \quad \forall i,$$
$$\sum_{i=1}^{\ell} y_i\alpha_i = 0.$$

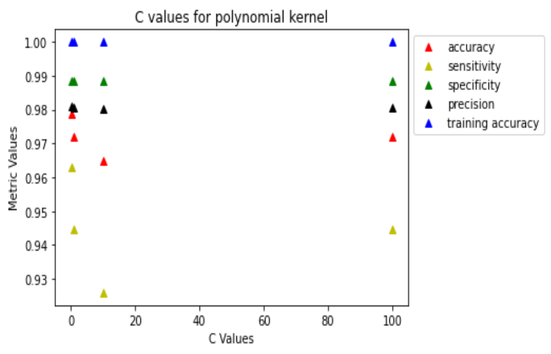
The computational difficulty of this problem grows very quickly as the number of data points increases. For our SVM, we implemented a simplified SMO algorithm[5] which aims to improve the training runtime by breaking up this optimization problem into much smaller problems. Platt, the creator of the SMO algorithm, argues that conducting these small optimizations are faster than trying to adjust the Lagrange multipliers via an analysis of the original optimization equation.[6] To conduct these smaller optimizations, the algorithm intelligently selects pairs of training data, and updates their corresponding Lagrange multipliers based on the error between them. The simplified version of the model we implemented takes a simpler approach to choosing data points. It loops through all of the data points, and for any data point with too much error it randomly chooses another data point and optimizes their Lagrange multipliers together. Therefore, during each pass, the simplified algorithm looks at a maximum of N pairs of Lagrange multipliers out of the total N(N-1) possible pairings. Since we don't examine each pair during each pass, we don't stop training until we have completed multiple passes without changing any multipliers. The exact number of such passes can be chosen as a hyperparameter. While the simplified algorithm is easier to implement than the full SMO algorithm, there are other tradeoffs. Because we choose data points randomly instead of purposely, it takes longer to train and is not guaranteed to converge to a global maximum.

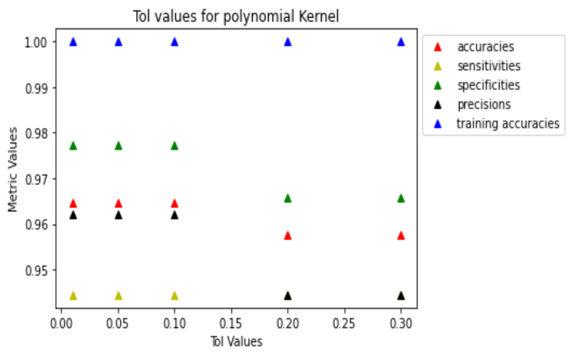
For our experiments, we decided to build two types of SVM models, the first being with polynomial kernels and the second having radial kernels. Training with either of these kernels requires a selection for the regularization parameter, C, max passes without change for the SMO algorithm, and a tolerance for the SMO algorithm. We decided to set max passes to two for all of our experiments. Increasing this value would give more confidence that we have reached an optimal solution at the cost of increasing runtime. With this value set, we conducted experiments to determine the best values for the C and tolerance hyperparameters for both of our kernels. Additionally, we experiment with possible degree values for our polynomial kernel as well as gamma values for our radial kernel. For every test that we do, we calculate the accuracy, sensitivity, specificity, and precision for identifying malignant tumors in our validation set. We will also record the accuracy of our training data. For our experiments, we focus on the overall accuracy and sensitivity. Having high sensitivity is important, because in a real world application, it is detrimental for a patient to not receive treatment if our model falsely classifies their tumor as benign.

For our polynomial kernel, training runtime was a serious issue, so for the purposes of finding optimal hyperparameters we compared models trained on only 20 data points. We first ran tests not shown here to determine a reasonable range of values for each of these hyperparameters. We then tested values over several orders of magnitude within these ranges. For C, we tried values of 0.0001, 0.1, 1, 10, and 100 with a set degree of 1 and tolerance of .1. For our tolerance values, we examined values of 0.01, 0.05, .1, .2, and .3 with a set degree of 1 and C of 1 based on the results from the first experiment. Here are graphs of those results:

[5] "CS 229, Autumn 2009 The Simplified SMO Algorithm."

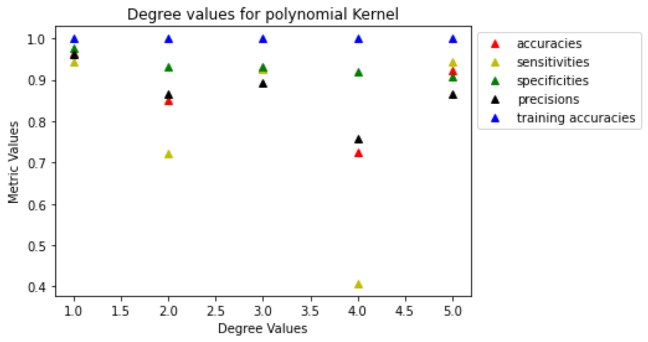
[6] Platt, John. "Fast Training of Support Vector Machines Using Sequential Minimal Optimization, in Advances in Kernel Methods."





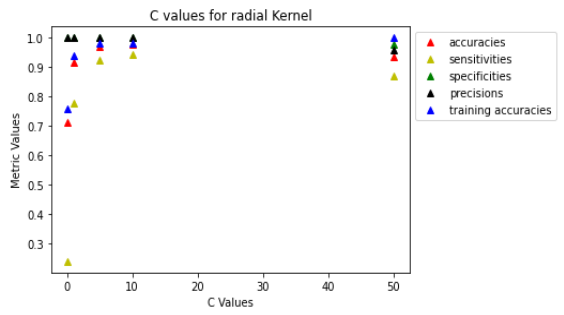
We did not display the results for C = .0001 because it performed very poorly and had sensitivity of 0. Otherwise, our model was able to achieve high validation accuracy and sensitivity given relatively small C and tolerance values which implies that our data is reasonably easy to separate. C = .1 was tied for the highest accuracy with C = 100, but had a higher sensitivity. Even though such a low value worked nicely, we chose a slightly larger value for C of 1, to avoid potential overfitting. It is also important to note that C represents the total error we allow across all of our training data points, so larger amounts of training data should have a larger value for C. Since our final polynomial model will be trained on more points than we used in these experiments, this is another reason to use a C that is on the larger side. Regarding tolerance, all of the lower values performed similarly, so we chose to use the highest tolerance value from this selection of .1, to make it faster to train.

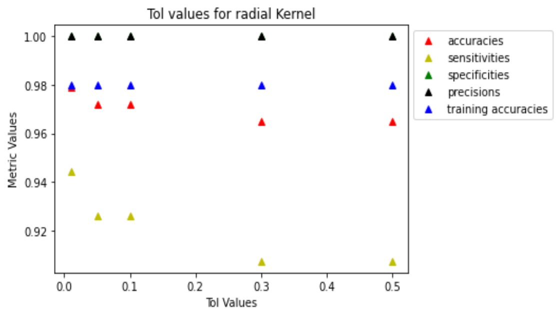
Next, we tested different degree values for our polynomial kernel function. Increasing the degree of the kernel increases the size of our hypothesis space. Therefore, we expect that increasing the degree will decrease bias while adding variance. We set C and tolerance values based on the previous experiments to 1 and .1 respectively. We then tested degree values of 1,2,3,4, and 5. This is the graph of the results:



The training accuracy was 100% for all degree selections including 1, suggesting that the training data has a linear decision boundary. For all other metrics, our model performed the best with a degree1 kernel, which gave a validation accuracy of 96.5%. The results of this experiment imply that the rest of our data also has a linear or nearly linear decision boundary, so allowing degrees beyond 1 allows our algorithm to overfit to the training data without learning meaningful structure. It's also interesting that our model seemed to perform better with odd degree polynomials than with even degree ones. In particular, even degree functions have really low sensitivity, which means it is giving a lot of false negatives. Perhaps this is caused by odd degree polynomials more easily approximating the hyperplane. It is also worth noting that we optimized our other hyperparameters using a linear kernel, which may have contributed to the degree one kernel's strong performance.

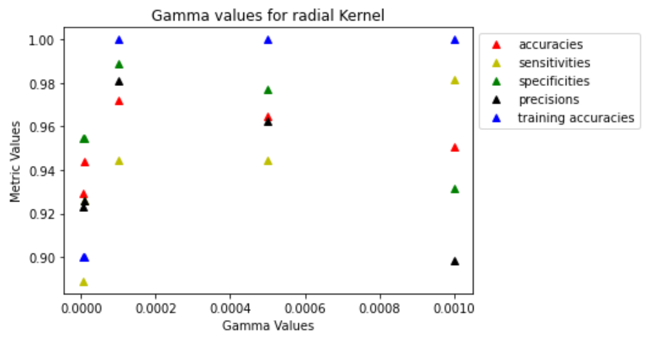
For our radial kernel, we conducted a similar process to find ideal values for C and tolerance. Once again, we first ran tests not shown here to determine a reasonable range of values for each of these hyperparameters. We then tested values over several orders of magnitude within these ranges. For C values, we tested .1, 1, 5, 10, 50 with a set gamma of .005 and tolerance of .1. We then tested tolerance values of .01, .05, .1, .3, .5 with a set gamma of .005 and C value of 10 based on the results of the first experiment. Here are the graphs of our results:





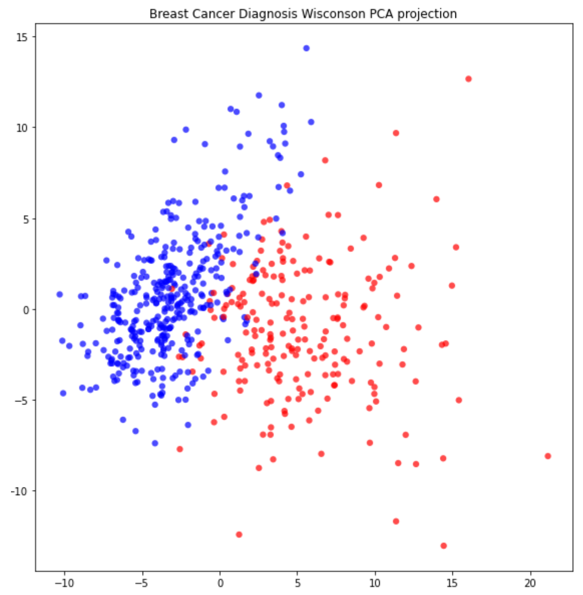
With radial kernels we needed a larger regularization term, C, for the algorithm to train well. In particular, C = 10 had the highest validation accuracy and sensitivity of our C values tested. If C was too small, then the algorithm classified everything as benign giving a sensitivity of 0. Similar to before, we found that the data is able to classify well with small tolerance values. We ultimately decided to use a tolerance of .1 to help with training speed even though a smaller tolerance value performed marginally better in terms of accuracy and sensitivity.

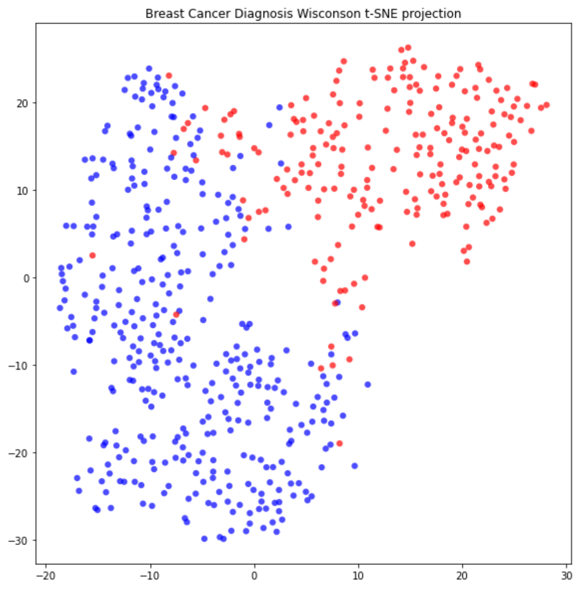
Next, we experimented with our gamma values. Based on our previous experiments, we set C = 10 and tolerance to .1 and then tried gamma values of .000005, .00001, .0001, .0005, and .001. Here is the graph of these results:



A gamma value of .001 gave a lower accuracy and specificity than the smaller gamma values, but had a higher sensitivity. In general, it seemed the optimal gamma value for classifying malignant data was slightly larger than the optimal gamma value for classifying benign data. One reason for this might be that there our data has more benign data points than malignant ones. When gamma is really small, the distance between the data point we are classifying and the other data is less important. Thus, for points near the boundary, the larger number of benign data points will influence the model to classify as benign. This results in lower sensitivity since we classify less points as malignant but higher accuracy because the majority of points are benign. For our final model, we decided to use a gamma value of .001, since this had a strong sensitivity without sacrificing much accuracy.

With the results of these experiments in mind, we decided to examine our data through PCA and t-SNE projections where red represents malignant data and blue represents benign:





Looking at the PCA projection, it is clear that even when projecting our features onto the plane, it is reasonable to separate our data linearly. In higher dimensions, it will be as easy or easier to separate it with a hyperplane. This explains why the linear kernel was able to classify the data successfully and why higher degree polynomials were unnecessary. Additionally, looking at the t-SNE projection, we can see that most data points are surrounded by other points of the same class. t-SNE does a good job of preserving local behavior, which is what the radial kernel is measuring, so it is not a surprise that the radial kernel also performs well. Validation data points in the middle that have points of both classes that are near them are likely the ones impacted by changing gamma values.

Throughout these experiments, we found that bad hyperparameters often lead to the model not converging or classifying every point the exact same way. However, with good hyperparameters our models tended to classify most points well. This makes sense in light of how well separated the data is. The points our model misclassified were likely statistical outliers and points that lived near the other class.

Combining our best hyperparameters, we trained the polynomial model on 30 training data points and trained the radial model on all of our training data points. We got the following metrics on the test data for our polynomial and radial models:

```
Polynomial:
{'accuracy': 0.923, 'sensitivity': 0.909,
'specificity': 0.929, 'precision': 0.851}

Radial:
{'accuracy': 0.986, 'sensitivity': 0.977,
'specificity': 0.990, 'precision': 0.977}
```

In both of these cases, the sensitivity is quite high. In particular, our radial model had very high sensitivity and accuracy, performing better than the other papers we saw that worked on this dataset. We believe a model like this is usable in real world applications where it is very important to not classify false positives. This model has been successful in doing that while maintaining a high accuracy. While we would recommend the radial kernel be used due to its better performance, we think it is important to recognize how well the linear degree, polynomial model performed on this model. This shows the data can be separated quite nicely by just a hyperplane. Perhaps the performance of the polynomial model would have been better if we had time to train it on all of our training data.

Further work in this subject could explore which features in our data made it so easy to distinguish between the two classes. With our model performing so well, the biggest issue for transforming this into a real world application might be data collection. By performing a feature analysis we would be able to narrow down which features are necessary in order to achieve confident classifications. If data collection could be made easier by only using the most important features, then this model could become usable in a real world setting. Feature analysis is not only important from a real world application perspective, but this analysis might help scientists better understand the differences between malignant and benign breast tumors. Another thing to explore, is just how to handle cases near the classifier. Perhaps, to further make this usable in the real world, features that are close to the classifier can be given professional attention or even classified positively to further decrease our false positive rate.

References

"CS 229, Autumn 2009 The Simplified SMO Algorithm." *Stanford*, 2009, cs229.stanford.edu/materials/smo.pdf.

Gareth, James, et al. *An Introduction to Statistical Learning*. Springer, 2013.

Platt, John. "Fast Training of Support Vector Machines Using Sequential Minimal Optimization, in Advances in Kernel Methods." *MIT Press*, 1998.

Wolberg, William, et al. "Breast Cancer Wisconsin (Diagnostic) Data Set." *UCI Machine Learning Repository*, 1995, archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29.

Mangasarian, Olvi L., et al. "Breast Cancer Diagnosis and Prognosis Via Linear Programming." *Operations Research*, vol. 43, no. 4, 1995, pp. 570–577., doi:10.1287/opre.43.4.570, https://pubsonline.informs.org/doi/abs/10.1287/opre.43.4.570.

Wolberg, W H. "Computer-Derived Nuclear Features Distinguish Malignant from Benign Breast Cytology." *Elsevier*, 1995, https://www.sciencedirect.com/science/article/pii/0046817795902295?via%3Dihub.

Laskov, Pavel, and Blaine Nelson. "Theory of Kernel Functions." Universität Tübingen. 2012, www.ra.cs.uni-tuebingen.de/lehre/ss12/advanced_ml/lecture3.pdf.