Breast Cancer Diagnosis and Prognosis using Neural Networks and SVMs

Andrew Levy
Boston University
8 Saint Mary's St.
levya@bu.edu

Yixuan Xiao
Boston University
8 Saint Mary's St.
yxxiao@bu.edu

Zhe Cai
Boston University
8 Saint Mary's St.
s20525xx@bu.edu

Abstract

In this paper, we evaluate neural networks and support vector machines as possible automated solutions to breast cancer diagnosis and prognosis. The Wisconsin Diagnostic and Prognostic Breast Cancer datasets were used to train and test the classifiers. Neural networks were optimized using grid search to determine structure and regularization parameters. SVMs were optimized using grid search to find optimal regularization and kernel parameters. Top neural network and SVM classification accuracies on the diagnosis data set were 98.1% and 97.7%, respectively. Top neural network and SVM classification accuracies on the prognosis data set were 84.5% and 81.9%.

1. Introduction

There is a need for automated methods in breast cancer diagnosis and prognosis because existing manual methods do not achieve both high accuracy and minimal invasiveness. Worldwide, breast cancer is the second most common cause of cancer deaths in females after lung cancer [1]. About 1 in 8 women in the United States will develop breast cancer over the course of her lifetime [2]. Yet current manual methods struggle to strike an effective balance that maximizes accuracy and minimizes bodily and monetary costs. The below table shows the wide range of sensitivity performance among the three most common manual breast cancer diagnosis techniques [3].

Method Type	Sensitivity Range (%)
Mammogram	68-79
FNA	65-98
Surgical Biopsy	~ 100

Figure 1: Sensitivity ranges of manual diagnosis methods

The imaging technique, mammography, and the biopsy technique, FNA (Fine Needle Aspiration), are both minimally invasive but have wide-ranging sensitivities. On the other hand, the surgical biopsy technique is the most accurate but is also the most invasive and expensive [3]. The purpose of this paper is to assess whether neural networks and support vector machines can serve as potential automated solutions to breast cancer diagnosis and prognosis.

2. Literature Review

Various methodologies have been applied successfully to breast cancer diagnosis. Anagnostopoulos et al. applied a probabilistic neural network (PNN) to the Wisconsin Diagnostic Breast Cancer (WDBC) dataset [4]. Unlike the feedforward neural networks implemented in this paper, PNNs assign label probabilities based on Euclidean distances between the test point and training samples belonging to a particular label. The group achieved a 97.9% CCR. Wolberg et al. applied logistic regression to the WDBC data set and achieved a classification accuracy of 96.2% [5]. Mu and Nandi in [1] used gradient descent to find optimal pairs of C and RBF-sigma in L2-regularized SVMs. The classification accuracy using the WDBC dataset was approximately 98% for these optimal pairings.

Different strategies have also been applied to breast cancer prognosis. Anagnostopoulos et al. applied a regression type neural network to the Wisconsin Prognostic Breast Cancer (WPBC) and classified predictions into different categories [4]. Their classification achieved a 92.3% accuracy. Mangasarian et al. implemented a recurrence surface approximation that determined the linear combination of input features that best approximate the time to cancer recurrence data provided in WPBC [3]. The average error of the algorithm was 24 months.

3. Problem Statement

3.1. Datasets

We used the Wisconsin Diagnostic Breast Cancer (WDBC) dataset to test our diagnosis classifier.

Researchers at the University of Wisconsin obtained the dataset by applying computer vision techniques to isolate healthy and cancerous nuclei from images of FNA biopsies. The following 10 features of the nuclei were then extracted: radius, perimeter, area, compactness, smoothness, concavity, concave points, symmetry, fractal dimension, and texture. The mean, standard error, and the extreme (largest or "worst") value for each feature were computed for each 569 patients, yielding a database of 569 samples x 30 features. The 569 images are composed of 357 benign and 212 malignant cases, which we relabeled "0" and "1", respectively [6].

The Wisconsin Prognostic Breast Cancer (WPBC) dataset was used to train our prognosis classifier. The dataset provides two features in addition to the those in the WDBC dataset: tumor diameter and number of lymph nodes removed. The dataset also provides a label for recurring and non-recurring patients and a time feature that indicates either the number of months post initial treatment when the cancer recurred or the number of disease free months since initial treatment. We labeled the samples with cancer that recurred within 24 months as a "1" and the remaining samples with a "0". Data samples belonging to patients with non-recurring cancer of less than 24 months were removed. The 24-month threshold was chosen so as to keep the training set as large as possible. There were many relatively new non-recurring patients so a longer time line would have forced these data samples to also be removed. Moreover, four of the samples in the WPBC dataset were missing values for the lymph node feature. In our implementation section, we discuss how we handled this degradation. After the initial preprocessing, the WPBC dataset included 198 patients of which 151 and 47 had non-recurring and recurring cancer within a 24-month period, respectively.

3.2. Classification Problems Defined

Given the datasets above, the following classification problems can be defined.

Diagnosis:

The diagnosis data set $D_D = \{(x_i,y_i)\} \in (X_D, Y_D)$ for i = 1,…, n = 569. $X_D = \Re^d$ for d = 30, $Y_D = \{0,1\}$. We seek to find a function Ψ_D: $X_D \Rightarrow Y_D$.

Prognosis:

The prognosis data set $D_P = \{(x_i,y_i)\} \in (X_P, Y_P)$ for i = 1,…, n = 198. $X_P = \Re^d$ for d = 32, $Y_P = \{0,1\}$. We seek to find a function Ψ_P: $X_P \Rightarrow Y_P$.

In this paper, we evaluate neural networks and SVMs as possible solutions for the mapping functions Ψ_D and Ψ_P.

3.3. Proposed Solutions

3.3.1. Neural Networks

Neural networks when combined with certain learning algorithms can be used as a supervised classification technique that uses a structure of interconnected neurons to learn feature detectors. Feature detectors are the combinations of input parameters that can best differentiate between class labels. In the context of our paper, the feature detectors will be the combinations of nuclei properties that can best distinguish between benign/malignant and recurring/nonrecurring cancer.

Neural networks have two attractive properties that can make them effective classifiers: function universality and trainability. Similar to NAND gates in Boolean circuits, neural networks are capable of representing any function in the R^n space [7]. Further, certain learning algorithms can be used to train a neural network to learn any underlying function.

Our diagnosis and prognosis neural network classifiers will use a feedforward structure such as that shown below.

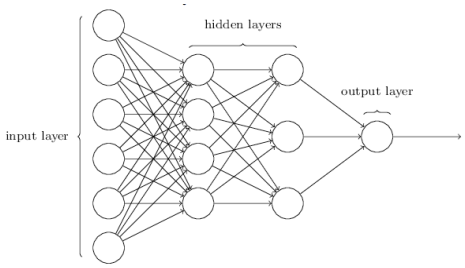

Figure 2: Example of a feedforward neural network structure

Feedforward neural networks generally contain three types of layers: input, output, and hidden. The input layer to a feedforward neural network is a single array of neurons with a length equal to the dimension of the feature set of the training data. Neural networks are a discriminative classification technique so the output layer of the typical feedforward neural network is composed of a "softmax" layer. The output layer thus represents the probability that the training sample belongs to a certain class. The dimension of our feature sets is 30 and 32 for the diagnosis and prognosis sets, respectively. Consequently, the inputs layers for those networks will be of lengths 30 and 32. Further, both classifiers will perform binary classification, and, as a result, the output layer will have one neuron. If the output neuron has a value > 0.5, the predicted label will be either malignant or recurring. Otherwise, the predicted label for the training sample will be benign or non-recurring. We discuss our methodology for optimizing the hidden layers in the implementation section.

The neural network classifiers will use a variant of stochastic gradient descent to learn the internal weights and biases that form the feature detectors. The cost

function that gradient descent will use reflects the error in the actual and predicted labels for each training sample. See the implementation section for more detail on the cost function we chose. A version of gradient descent is used to train neural networks because it would not be practical to find optimal parameters by calculating first derivatives and finding extrema as there are typically too many parameters in a neural network. Gradient descent finds the minimum of the cost function by updating the weights and biases in the direction of the gradient, which is the direction of steepest descent of the cost function. The equations to update the weights and biases are shown below:

(1) $w_k \Rightarrow w_k' = w_k - (\eta/m) \Sigma_j (\partial Cx_j / \partial w_k)$

(2) $b_k \Rightarrow b_k' = b_k - (\eta/m) \Sigma_j (\partial Cx_j / \partial b_k)$

such that x_j is the j-th training sample, η is the learning rate, and m is the batch size. This version of gradient descent is "stochastic" because the gradient is determined by taking the average of a random subset or batch of the training samples. This batch is of size m.

Gradient descent uses a process known as backpropagation to determine the gradients. Backpropagation enables the calculation of the gradient of the cost function with respect to any weight or bias in the network by working backwards from the gradients of the final layer. The gradient of the cost function with respect to any weight or bias can be calculated with backpropagation as follows [8]. Let z^n be the vector of inputs to the activation functions of each neuron in the n-th layer, w^n be the weight vector matrix in which w^n_{kj} is the weight connecting the k-the neuron in the (n-1)-th layer to the j-th neuron in the n-th layer, b^n be the biases in the n-th layer, a^n be the vector of activation outputs in the n-th layer, and $\sigma(z)$ be the activation function for each neuron. Then

(3) $z^n = (w^n)^T a^{n-1} + b^n$

(4) $a^n = \sigma(z^n)$

Also, let δ^n be the gradient vector of the cost function with respect to z^n.

(5) $\delta^n = \nabla_z C$

Then using the chain rule, δ^N of the final layer N is

(6) $\delta^N = \nabla_a C \bullet \sigma'(z^N)$,

in which $\bullet$ is the Hadamard or elementwise product. With some further chain rule algebra, δ^n for any layer n can be determined by working backwards from δ^N of the final layer (see appendix section for proof). Specifically,

(7) $\delta^n = ((w^{n+1})^T \delta^{n+1}) \bullet \sigma'(z^n)$

Given the definition of z from (3), the gradient of the cost function with respect to any weight or bias in the network is then

(8) $\partial C / \partial w^n_{jk} = a_k^{n-1} \delta^n_j$

(9) $\partial C / \partial b^n_j = \delta^n_j$

Gradient descent will update the weights and biases in the direction given by the averages of (8) and (9) for a given batch of training samples.

Gradient descent as described above makes the following two assumptions both of which are related to the cost function. First, the cost function must be a function of the costs of the individual training samples.

(10) $C = (1/n) \Sigma_x C_x$

If the cost function is not a function of the individual training samples costs, $\Sigma_j (\partial Cx_j / \partial w_k)$ and $\Sigma_j (\partial Cx_j / \partial b_k)$ in the (1) and (2) cannot be computed. Second, the cost must be a function of the output predictions of the network.

(11) $C = C(a^L)$

such that a^L is the vector of outputs from the last layer L. If the cost function is not related to the last layer of the network, backpropagation will not be able to determine (6), the gradient of the cost function with respect to the last layer. Gradients in prior layers will then not be able to be calculated.

See the implementation section for how the neural network classifiers were further optimized.

3.3.2. Support Vector Machines

- Basics

Support vector machines are a supervised learning algorithm for solving linear and non-linear classification problems. The main idea of binary SVMs is to construct a hyperplane that maximizes the margin of separation between two classes of points. SVMs then classify test data points based on the side of the decision boundary that the point falls on.

The optimal hyperplane can be found by solving the following constrained optimization problem

(12) $\min_{w,b} (1/2)||w^2||$

s.t. $y_i(w^T x_i + b) \geq 1$

This minimization problem can be rephrased as a dual optimization problem. The resulting decision rule from the dual problem is

(13) $h(x) = \text{sign}(\sum_i \alpha_i (x_i \cdot x) + b)$,

in which α_i represent the support vectors or the points that lie on the margin.

- Regularization

Our datasets are not linearly separable and regularization is needed to penalize slack variables while optimizing margin. The best regularization is L0 norm, but it is a NP-hard problem. Therefore, L1 and L2 regularization can be used as a relaxation of the problem to find an approximate hyperplane. These regularization techniques add a penalty term, C, to the cost function that represents the tradeoff between choosing a hyperplane that maximizes the soft margin and one that maximizes the number of correct training point classifications. The primal form to solve the constrained optimization problem are:

(14) L1: $\min_{w,b,\xi} (1/2)\|w^2\| + C \sum_i^n \xi_I$
s.t. $y_i(w^T x_i + b) \geq 1 - \xi_i \quad \wedge \quad \xi_i > 0$
(15) L2: $\min_{w,b,\xi} (1/2)\|w^2\| + C \sum_i^n (\xi_i)^2$
s.t. $y_i(w^T x_i + b) \geq 1 - \xi_i \quad \wedge \quad \xi_i > 0$

However, it turns out the primal forms are too hard to solve in polynomial time. Hence the dual form is introduced to optimize a new variable α:

(16) L1:
$\max_\alpha \sum_{i=1}^n \alpha_i - (1/2) \sum_{i=1}^n \sum_{i=1}^n y_i y_j \alpha_i \alpha_j K(x_i, x_j)$
s.t. $\sum_{i=1}^n y_i \alpha_i = 0 \wedge 0 \leq \alpha_i \leq C$

(17) L2:
$\max_\alpha \sum_{i=1}^n \alpha_i - (1/2) \sum_{i=1}^n \sum_{i=1}^n y_i y_j \alpha_i \alpha_j (K(x_i, x_j) + 1/\delta_{ij})$
s.t. $\sum_{i=1}^n y_i \alpha_i = 0 \wedge 0 \leq \alpha_i \leq C$

The resulting decision rules are the same as (13), except the support vectors now include all points within or on the soft margins.

- Kernel trick

An important feature of SVMs is that they can be used to solve non-linear classification problems through a kernel function along with regularization. The kernel function maps two classes of data points in a lower dimensional feature space onto a higher dimensional space, where the points are more likely to be linearly separable by a hyperplane. When a kernel function K is used, the decision rule can then be represented as:

(18) $h(x) = \text{sign}(\sum_i \alpha_i K(x_i, x_j) + b)$

In this project, mainly two kernel functions are tested: linear and RBF kernel. Linear kernel is described below:

(19) $K(x_i, x_j) = x_i^T x_j = < x, x' >$,

Not all problems are linear separable in higher dimensions. RBF kernel can be useful in this case:

(20) $K(x_i, x_j) = \exp((x_i - x_j)^T \Sigma^{-1} (x_i - x_j) / 2)$

This is equivalent to:

(21) $C * \exp(x_i - x_j)$, *C is a constant of x itself*

This form can be interpreted as a Tayler expansion that projects x_i and x_j into an infinite dimensional feature space where a hyperplane can then separate the two classes of data points.

4. Implementation

4.1. Data Preprocessing

Min-max normalization was applied to the datasets to normalize the wide ranges of the feature data. The below plot graphs the distribution for each of the features in the WDBC dataset. Per the plot below, the ranges of the features vary widely across the dataset. These wide ranges can be problematic for both our neural network and SVM classifiers.

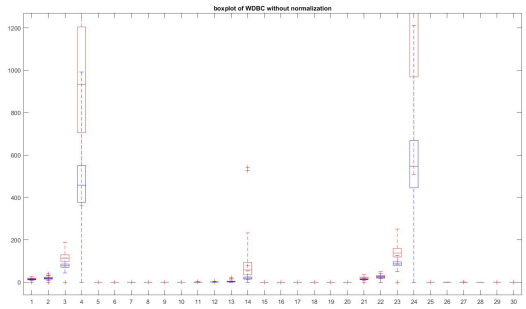

Figure 3: WDBC feature distribution

For neural networks, learning feature detectors becomes more difficult in this scenario because it is challenging to fine-tune a weight that represents a combination of features with very different ranges. For SVMs, the varying ranges of features may produce a hyperplane that is more sensitive to the larger features and thus less robust. Min-max normalization scales the data set into the range between [0,1] and thus can alleviate this issue. Both non-normalized data and min-max normalized datasets were evaluated in our implementation.

PCA was also applied to our datasets in the case that there were redundant or noisy features. Redundant data is likely as the dataset is only composed of 10 unique features from which the mean, worst, and standard error have been recorded. PCA works by filtering out the feature dimensions that have small eigenvalues, which imply that they may have little information or are noise. We choose to test the performance of SVM on 5 features, which yielded only 0.04% approximation error. To make sure the test data is in the same feature space as training data without contaminating the testing data, a transformation matrix was extracted from training data and then applied to testing data. In our implementation, the SVM classifiers performed approximately equally on both the PCA-reduced dataset and the full data set. Consequently, the PCA-reduced dataset was thus ignored for our neural network implementation.

Further, two strategies were used to handle the data degradation in the WPBC dataset. Those samples with missing lymph node features were either removed or the median value for that feature was added. Both produced similar results, so the WPBC results presented used training data in which those samples have been removed.

4.2. Neural Networks

To optimize the performance of our neural network classifiers, we implemented a grid search that focused on fine-tuning four hyper-parameters relating to the regularization and structure of the networks.

Regularization can be an important tool for neural networks as overfitting can be an issue given the large

number of weight and bias parameters in a network. Indeed, for many of the models we tested, there were many more parameters than actual training samples. We used two regularization techniques to improve our classifiers.

The dropout regularization technique was integrated into our neural network to reduce overfitting. Unlike L1 or L2 regularization that add terms to the cost function, dropout modifies the structure of the network. Dropout works by eliminating a percentage of the neurons within the hidden layers during training. Dropout has demonstrated strong empirical success in improving the performance of neural networks [8,9]. There are two popular explanations for the benefits of dropout regularization. Hinton et al. explain in [9] that dropout reduces the co-adaptation of neurons in a multi-layered network. With dropout, neurons can no longer rely on neighboring neurons and thus have to develop robust feature detectors on their own. The resulting network thus contains neurons that are less susceptible to overfitting. In addition, dropout can be interpreted as training multiple neural networks in one as the layout of the network changes for each training sample [8]. The resulting network can be thought of as the average of the collection of trained neural networks. The averaging of networks can put downward pressure on the weights of the network and thereby possibly reduce overfitting.

The second regularization technique was to put actual limits on the weights of the networks. In a similar manner, this would put downward pressure on the internal weights of the network and thereby possibly limit overfitting.

We also attempted to optimize the structure of the neural networks by fine-tuning with the sizes of the hidden layers. Our networks assumed two hidden layers and then tested a range of sizes of each hidden layer.

The four-dimensional grid search that was implemented trained and tested neural network classifiers using every unique group of four parameters in the following ranges:

- Hidden layer 1 Size: [10, 30, 50, 70, 90, 110]
- Hidden layer 2 Size: [10, 0, 30, 40]
- Dropout Percentage: [10, 20, 30, 40, 50, 60, 70, 80, 90]
- Weight Limits: [1, 2, 3, 4, 5]

Moreover, we chose to train the neural network classifiers using the cross-entropy cost function to improve and speed up the parameter optimization. The cross-entropy cost function is

(22) $C = (-1/n) * \sum_x [y \ln a + (1 - y)\ln (1-a)]$,

in which x is the individual training samples, y is the actual output label, and a is the predicted output from the output activation function. The cross-entropy function is useful because the gradient of the cross-entropy cost function with respect to the weight and bias parameters is proportional to the error of the classification ($\sigma(z) - y$)) [8].

(23) $\partial C / \partial w_j = (1/n) * \sum_x x_j(\sigma(z) - y)$

(24) $\partial C / \partial b = (1/n) * \sum_x x_j(\sigma(z) - y)$

This is helpful because, during gradient descent, the parameters of the model will be adjusted by a greater magnitude when there are larger classification errors, which will speed up the optimization. The mean squared error cost function does not have this attractive property. The gradients for the mean squared error cost function are:

(25) $\partial C / \partial w = (a - y)\, \sigma'(z)x$

(26) $\partial C / \partial b = (a - y)\, \sigma'(z)$

The issue with these gradients is the presence of the $\sigma'(z)$ term when there is significant classification error. If the activation function $\sigma(z)$ is the sigmoid function and z is a number with a large magnitude, $\sigma'(z)$ will be small as the slope of the sigmoid function is small at the extremes. Consequently, the gradients will be small and learning times slow if there is large error in the network's classifications.

Our neural network classifiers were implemented using two third-party applications. The structure of the neural network and the parameter optimization were implemented using the Keras deep learning software framework. Additionally, the model was trained using an Amazon AWS GPU (instance type: g2.2xlarge) to reduce training times.

4.3. Support Vector Machines

The SVM methodology can be summarized in the flow chart below.

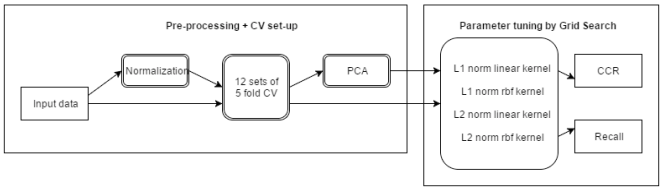

Figure 4: SVM Flow Chart

Two methods of preprocessing are tested along with non-preprocessed data, so that four data sets are fed to SVM model.

To find the optimal SVM classifier we implemented a large grid search to find the best collection of tuning parameters. Due to the non-separable nature of the data, our grid search examined the performance of numerous pairs of cost penalties and kernel functions, which included linear and RBF kernel, under both L1 and L2 regularization.

5. Experimental Results

5.1. Neural Networks

The table below shows a summarized version of the top neural network performances from the four-dimensional grid searches when optimized for either classification accuracy or sensitivity. The tables include the top overall result, the top result using no regularization, and the top result using the raw dataset with no min-max normalization. See appendix for greater detail on the neural network structures that produced these results.

WDBC Top Performances

Model	CCR (%)	Sensitivity (%)
Best	**98.07**	**96.23**
No Reg.	97.12	95.76
No Min-Max	95.25	95.23

WPBC Top Performances

Model	CCR (%)	Sensitivity (%)
Best	**84.52**	**83.87**
No Reg.	84.52	35.48
No Min-Max	84.52	83.23

Overall, the performance of the neural network classifiers surpassed our expectations on the WDBC dataset and was consistent with our expectations on the WPBC dataset. Prior to beginning our implementation, we were unsure how neural networks would perform on a dataset composed of manually chosen features as opposed to an image, for instance, in which there are no organized feature set. Moreover, we did not expect the neural network classifier to perform well on the WPBC dataset due to its small size and unbalanced distribution of classes.

5.2. Support Vector Machines

5.2.1 WDBC

Top average CCRs of each model for the WDBC dataset are listed below. See appendix for sensitivity results.

CCR	Standard	***Normalized***	PCA	Normalized + PCA
L1 norm Linear Kernel	95.40 C=1	***97.70*** ***C=8***	92.77 C=0.25	94.58 C=0.5
L1 norm RBF Kernel	92.49 σ=32 C=128	97.75 σ=1 C=2	92.09 σ=32 C=0.5	94.26 σ=4 C=4
L2 norm Linear Kernel	95.5 C=0.5	97.45 C=8	92.26 C=0.5	94.38 C=0.5
L2 norm RBF Kernel	93.48 σ=32 C=1/16	N/A	N/A	N/A

Per the table above, L1-SVM with linear kernel performs about equal to the L1-SVM with RBF. However, the L1-SVM with linear kernel produces a better sensitivity performance. Most of the L2 norm result is not available because the optimization failed converge within 10 million iterations.

5.2.2 WPBC

Top average CCRs of each model for the WPBC dataset are listed below. See appendix for sensitivity results.

CCR	Standard	Normalized	SVD	***Normalized + SVD***
L1 norm Linear Kernel	80.91 C=1/32	81.94 C=1	81.83 C=1/16	81.94 C=1
L1 norm RBF Kernel	81.94 σ=1 C=1	81.94 σ=1 C=1	81.94 σ=1 C=1	***81.94*** ***σ=1*** ***C=1***
L2 norm Linear Kernel	80.97 C=0.5	81.94 C=8	81.61 C=0.5	81.94 C=0.5
L2 norm RBF Kernel	81.94 σ=1 C=1	81.94 σ=1 C=1	81.94 σ=1 C=1	81.94 σ=1 C=1

The result for WPBC is surprisingly consistent across multiple models. Similar to the neural networks results, min-max normalization appears to improve the classifier.

6. Conclusions

Our primary conclusion from this project is that automated methods such as neural networks and SVMs should likely play a larger role in breast cancer diagnosis. The combination of our results and the strong results from prior literature suggest that automated methods may be better able to achieve the dual goal of maximizing accuracy and minimizing invasiveness than current manual techniques.

Future work should examine the application of convolutional neural networks and other machine learning methods to mammography, the imaging technique that is the other popular, minimally invasive technique for breast cancer diagnosis. If effective, this could further increase the number of less invasive and costly tools that doctors can use for breast cancer classification.

The team gained several new skills from this project. The team learned an entirely new machine learning method in neural networks. The team learned why they are useful, how they work, and how they can be optimized. Additionally, the team gained greater familiarity with regularization techniques such as L1, L2, and dropout and how they can reduce overfitting. The team also learned about certain preprocessing techniques such as min-max normalization and PCA and how they can be used to improve the training of classifiers. Lastly, the group not only learned about the theory behind these concepts, but learned how to implement these and other concepts through code.

7. Description of Individual Effort

7.1. Andrew Levy

Andrew focused on implementing the neural network classifiers. Andrew also helped apply min-max normalization to the datasets.

7.2. Yixuan Xiao

Yixuan focused on implementing the data preprocessing techniques. Yixuan also helped implement the grid search that optimized the neural network classifiers.

7.3. Zhe Cai

Zhe focused on implementing the SVM classifiers. Zhe also helped perform PCA feature reduction on the datasets.

8. References

[1] M. Tingting, and A. K. Nandi, "Breast cancer detection from FNA using SVM with different parameter tuning systems and SOM-RBF classifier," *Journal of the Franklin Institute*, no. 344, pp. 285-311, 2007.

[2] U.S. Breast Cancer Statistics [Online]. Available: http://www.breastcancer.org/symptoms/understand_bc/statistics

[3] Mangasarian et al., "Breast Cancer Diagnosis and Prognosis via Linear Programming," *Mathematic Programming Technical Report*, no. 94-10, 1994.

[4] Anagnostopoulos et al., "The Wisconsin breast cancer problem: Diagnosis and TTR/DFS time prognosis using probabilistic and generalised regression information classifiers," *Oncology Reports*, no. 15, pp. 975-981, 2006.

[5] Wolberg et al., "Computer-Derived Nuclear Features Distinguish Malignant from Benign Breast Cytology," *Human Pathology*, no. 7, pp. 792-796, 1995.

[6] Breast Cancer Wisconsin (Diagnostic) Data Set [Online]. Available: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

[7] Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function," *Math. Control Signals Systems*, no. 2, 1989.

[8] M. Nielsen. (2016). *Neural Networks and Deep Learning* [Online]. Available: http://neuralnetworksanddeeplearning.com/

[9] G. E. Hinton et al., "Improving neural networks by preventing co-adaptation of feature detectors," University of Toronto, Ontario, 2012.

9. Appendix

9.1. Math Derivations

This section proves (7), which is one of the key equations of backpropagation. This equation describes how the gradient of the n-th layer ($\delta^n = \nabla_z C$) can be determined from the gradient of the (n + 1)-st layer, δ^{n+1}. The ability to know $\nabla_z C$ for any layer makes it possible to then determine the gradients with respect to any weight or bias in network, $\partial C / \partial w^n_{jk}$ and $\partial C / \partial b^n_j$. The gradient descent learning algorithm can then be properly implemented when the gradients of the cost function with respect to the weight and base parameters can be calculated. The proof is as follows [8]:

(27) $\delta_j^n = \partial C^n / \partial z_j^n$
(28) $= \sum_k (\partial C^n / \partial z_k^{n+1})(\partial z_k^{n+1} / \partial z_j^n)$
(29) $= \sum_k \delta_k^{n+1} (\partial z_k^{n+1} / \partial z_j^n)$

z_k^{n+1} can be expressed in terms of z_j^n
(30) $z_k^{n+1} = \sum_j w_{kj}^{n+1} a_j^n + b_k^{n+1} = \sum_j w_{kj}^{n+1} \sigma(z_j^n) + b_k^{n+t}$
where w_{kj}^n is the weight from neuron j in the (n-1)-st layer to neuron k in the n-th layer.
(31) $\partial z_k^{n+1} / \partial z_j^n = w_{kj}^{n+1} \sigma'(z_j^n)$

Substituting (31) back into (29)
(32) $\delta_j^n = \sum_k w_{kj}^{n+1} \delta_k^{n+1} \sigma'(z_j^n)$

This is the component form of (7), which is the equation to be proved.

9.2. Detailed Results

9.2.1. Neural Networks

- Top Results WDBC (Accuracy %)

Normalization	Layer 1	Layer 2	Dropout (%)	Max Weight	Accuracy (%)	Std
Min-Max	10	10	40	1	98.07	.01
Min-Max	10	10	None	None	97.12	.01
None	30	30	10	4	95.25	.03

- Top Results WDBC (Sensitivity %)

Normalization	Layer 1	Layer 2	Dropout (%)	Max Weight	Accuracy (%)	Std
Min-Max	90	30	10	2	96.23	.02
Min-Max	10	30	None	None	95.76	.03
None	50	20	0	4	95.23	.02

- Top Results WPBC (Accuracy %)

Normalization	Layer 1	Layer 2	Dropout (%)	Max Weight	Accuracy (%)	Std
Min-Max	30	30	50	2	84.52	.02
Min-Max	70	10	None	None	84.52	.02
None	50	20	50	2	84.52	.02

- Top Results WPBC (Sensitivity %)

Normalization	Layer 1	Layer 2	Dropout (%)	Max Weight	Accuracy (%)	Std
Min-Max	30	30	40	None	83.87	.03
Min-Max	70	20	None	None	35.48	.10
None	50	20	40	None	83.23	.02

9.2.2. Support Vector Machines

- WDBC (Sensitivity %)

Sensitivity	Standard	Normalized	SVD	Normalized + SVD
L1 norm Linear Kernel	97.34	99.23	96.27	97.87
L1 norm RBF Kernel	93.38	98.67	100	93.79
L2 norm Linear Kernel	97.36	99.21	96.38	97.87
L2 norm RBF Kernel	100	N/A	N/A	N/A

- WPBC (Sensitivity %)

Sensitivity	Standard	Normalized	SVD	Normalized + SVD
L1 norm Linear Kernel	98.17	100	99.87	100
L1 norm RBF Kernel	100	100	100	100
L2 norm Linear Kernel	98.29	100	99.61	100
L2 norm RBF Kernel	100	100	100	100

9.3. Source Code

Source code submitted separately. Code can also be found at the below GitHub link.

https://github.com/GordonCai/503-project