A Comparison of Classification and Regression Methods in the Diagnosis of Diabetic Retinopathy

Dominic Chang *

August 18, 2022

Abstract

Diabetic retinopathy is a major cause of blindness. The disease is categorized by 4 stages, with the first being mild and the final proliferate diabetic retinopathy. Because medical screening is often difficult to carry out and rare in rural places, it is imperative that Machine Learning models are trained to be able to detect the disease. With successful Machine Learning models, early detection can prevent later stages of the disease. We can think of this task as regression or classification because we are predicting stages or values that can be continuous values or discrete classes. Therefore, we compared Machine Learning classification and regression models for our task. We utilized K Nearest Neighbors, Linear Models, Decision Trees, and Random Forests as our models. In addition, we evaluated each model's performance with accuracy and Mean Squared Error. Our results show that the classifier models tend to have higher accuracies while regressor models have lower Mean Squared Errors. After trying our models on the original dataset and an augmented dataset of 30,000 images, we found the logistic regression classifier model to be the best with 67% accuracy and 0.47 Mean Squared Error on the augmented dataset. Then, when we tried a modified detection task (0 or 1 labels) with the augmented dataset, we achieved an accuracy of 83% accuracy and 0.13 Mean Squared Error with the linear regression model. Overall, it becomes clear that linear models work best with our task of detecting diabetic retinopathy.

1 Introduction

Diabetic retinopathy is the most widespread source of blindness among adults [nat]. The disease can be contracted by anyone with any type of diabetes because of how high blood sugar affects the eyes [nat]. Too much sugar in the blood will block the vessels that lead to the retina, which is the part of the eyes that detects light and sends the appropriate signals [nat]. This blockage will cause the vessels to leak fluid or even bleed [nat]. While the eyes will grow new blood vessels, these are weaker and will leak and bleed easily [nat]. Diabetic retinopathy starts out as changes in vision but as it worsens, dark spots form as a result of blood vessels leaking into the fluid that fills the eye, called vitreous [nat]. Eventually, diabetic retinopathy can lead to eye conditions like Diabetic Macular Edema, where blood vessels leak into the macula and cause blurry vision, Neovascular Glaucoma, where abnormal blood vessels block draining fluid which can cause vision loss or blindness, and Retinal Detachment, where scars that form in the back of the eye pull the retina away [nat].

Needless to say, early detection of diabetic retinopathy is crucial. Normally, the disease is detected through dilated eye exams or regular eye exams, but not everyone has the privilege. Therefore, our goal is to train a Machine Learning model, a program which "learns" by finding patterns in data, that will be able to make diagnoses based on retina images. By doing this, we can provide wider access to early detection of diabetic retinopathy because Machine Learning models can be run by small devices anywhere like local pharmacies.

Diabetic retinopathy has four stages ranging from mild to moderate to severe nonproliferative retinopathy, and finally proliferative diabetic retinopathy [NN]. In the mild stage, tiny bulges called microaneurysms appear in the retina blood vessels, causing small leaks that do not affect vision [NN]. In the moderate stage, the blood vessels swell and cannot carry blood as well, which can lead to Diabetic Macular Edema [NN]. Next, with the severe stage, blood vessels become more blocked and

*Advised by: Mason McGill

if closed (macular ischemia), dark spots and blurry vision ensue [NN]. Not only this, but scar tissues form and new blood vessels may be created [NN]. Finally, in the proliferative stage, the new blood vessels grow into the retinas and the vitreous (neovascularization), causing retinal detachment and blindness [NN]. When looking at our images, it is crucial that our model is able to discern the features like the microaneurysms and the leaks in order to correctly classify each stage.

2 Data

2.1 Dataset

For our data, we utilized a dataset from a past competition called the Asia Pacific Tele-Ophthalmology Society (APTOS) 2019 Blindness Detection challenge [kag]. The dataset includes images of the fundus, the back of the eye which includes the retina, taken by fundus cameras, which are low power microscopes with an attached camera [dep]. These machines are common in optometry offices and we seek to widen access to their capabilities by installing stations in places like pharmacies to detect any eye condition like diabetic retinopathy early on. The images scale over different conditions and are split into both training and testing sets, and also include annotation files. The training file matches photo id names with diagnosis numbers made by clinicians that range from 0 (No diabetic retinopathy) to 5 (Proliferative Diabetic Retinopathy), while the testing file only contains photo ids.

2.2 Preprocessing

After matching associated input images with diagnosis labels, our next step was to prepare our data to feed into a Machine Learning model. We then used transfer learning, a Machine Learning technique that utilizes another pre-trained model to get useful features out of input data to then be used in the actual model. This method helps us achieve higher accuracy with less time and computational power spent. For our pre-trained model, we used a Resnet18 model [HZRS15] that was trained on ImageNet's dataset [RDS+14] and we pulled out the second to last layer's 512 features. We did so because pre-training on ImageNet is great for image classification tasks and Resnet18 is one of the simplest models with great performance. However, we need to apply preprocessing to our data in order to use the Resnet18 model, which included resizing to 256 by 256 and center cropping to 224 by 224, and normalization by the mean and standard deviation of the ImageNet data color channels red, green, and blue, which are [0.485, 0.456, 0.406] and [0.229, 0.224, 0.225] respectively.

2.3 Data Augmentation

The training set was short with only 3662 images and the testing set 1928 images. However, we were able to add an additional 27,000 images to the original images by writing code to augment our images [BSH17]. Our augmentations happen with a set probability and how much of a transformation is performed is decided randomly given a range. Our data augmentations included transformations like rotations ranging from 10 degrees left to right (0.7 probability), left right flips (0.5 probability), and zooming from 1.1 to 1.5 times (0.5 probability).

2.4 Dealing with Duplicates

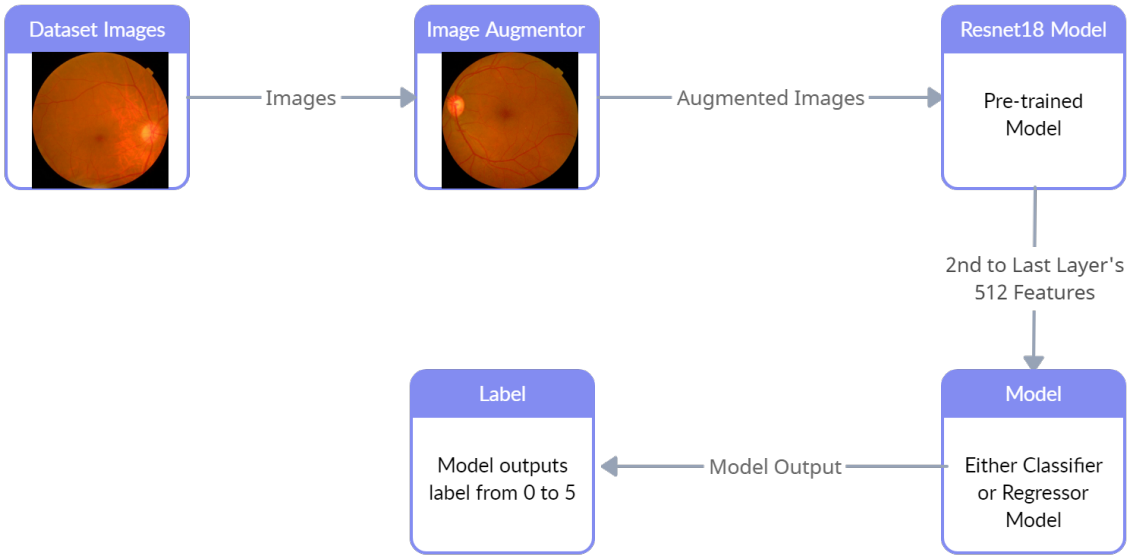
Additionally, after training our model and checking the accuracy on the training data itself, we found that there were some errors: When there should have been 100% accuracy, there was actually slightly less. After searching through our data and feature inputs, we found various duplicate images with different labels. In fact, out of the 3662 images, only 3534 were actually unique. To fix this, we sorted our images alphabetically and only used the first of each duplicate image and its corresponding label.

2.5 Final Data Techniques

The Machine Learning models we talk about later operate on the inputs, the features from Resnet18's second to last layer, and corresponding labels, the 0-5 diagnoses, together. We then split all this data 80-20 into training and validation sets. We do this because we need a lot of data to train our model, learning internal weights and such, but also need testing data to tune our hyperparameters, which are

values passed into our actual models. Finally, the training set is passed into the model and the model is tested on the validation set.

2.6 Data Pipeline Diagram



3 Methods

3.1 Framing our Task

Because the output labels of our task are integers from 0 to 5, we can make our task both classification and regression. This means that we can have our outputs be the diagnosis integers as categories and pick between them (classification) or continuous values from 0 to 5 (regression). From there, we chose some of the less complex yet still popular models from both categories, including K Nearest Neighbors (both), Linear Models(both), Decision Trees (both), and Random Forests (both).

In addition, because this task of predicting diagnoses from 0 to 5 may be complicated and our models may not achieve a sufficient accuracy, we also explored a detection task. This means outputting 0 or 1 to tell if the patient has the disease diabetic retinopathy. Although this will not diagnose stages, our model might only need to let doctors know if patients need advanced screening and help. This way, we can achieve higher accuracies and efficiencies while still working toward our goal.

3.2 K Nearest Neighbors Classifier and Regressor Model

The K Nearest Neighbors Model (KNN for short) classifier will output the "majority vote" label from the nearest K neighbors, the number of which is determined by the hyperparameter K (number of neighbors) and is tuned based on the validation set. Then, the nearest neighbors are just the points closest to the input feature vector the model is looking at. Here, "closest" means the least distance between the two points, represented by a metric with a form of stored training set vector minus input vector. KNN uses the Euclidean Distance which is represented by the equation:

$$\text{Euclidean Distance} = \sqrt{(\sum_{i=1}^{n}(Y_i - X_i)^2)}$$

where:

n = total number of dimensions

i = dimensionality of the vector

Here, Euclidean distance is just the square root of the entire sum of all n distances, which is training set scalar Yi minus input scalar Xi, or the difference in values raised to the 2nd power. The regressor, on the other hand, will just take the average of the nearest K neighbor labels while otherwise being the same.

3.3 Linear Classifier and Regressor Models

The Linear Regression model is a regressor based on drawing a linear function of form y = mx + b to fit data points. On the other hand, the Logistic Regression model, although it sounds like a regressor, is actually a classifier. The model uses a linear function that separates different classes and uses the sigmoid or logistic function to compute outputs (probability estimates). The sigmoid function, represented by σ(x), raises the Euler's constant e to the negative of input value x, adds one, and finds the reciprocal of it, as shown below[log].

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

This function makes it so that infinitely large values get mapped to 1/1 or 1 and infinitely negative values get mapped to 1/infinity or 0 [log]. This function transforms input values into numbers ranging from 0 to 1, representing the probability of the input being of a given class[log]. The highest number class is then outputted as the predicted label. With this model, the most important hyperparameter is the C regularization value. The C value determines how complex the model is, or how nonlinear or complicated the function that maps inputs to outputs is, by changing the coefficient of a penalty in the loss function. The loss function evaluates the model and a higher value, which comes from a higher penalty, means the model does worse [reg].

$$\text{Loss Function} = \text{Loss Metric} + \frac{1}{C}\sum_{i=1}^{n}(W_i^2)$$

The penalty, represented by $\frac{1}{C}\sum_{i=1}^{n}(W_i^2)$, determines how much influence the weights have on the cost function because a lower coefficient will mean that the weights can be greater for the same loss. Then, with greater weights, the "captured complexity" of the model also increases [reg]. By tuning this value, we will be able to prevent overfitting, which is when the model memorizes the training data and performs poorly on outside data, and underfitting, when the model is too general to pick up patterns.

3.4 Decision Tree and Random Forest Models

The Decision Tree and Random Forest models are both models based on logic and decision making. The Decision Tree classifier is a tree where each branch separates classes based on categories, while the regressor does so based on continuous values. These decision points are adjusted with training, but an important hyperparameter is the max depth or maximum number of levels in the decision tree. The deeper a tree is, the more regions or compartments there will be, which increases the possibility of memorizing the training data and thus overfitting. However, with too much data in each compartment, like less neighbors in KNN, there might be generalization. For this reason, there is a golden spot for the max depth that can balance out both.

Next, building on top of the Decision Tree, the Random Forest uses the average of the output of many random Decision Trees in an ensemble to better accuracy and reduce overfitting. Here the hyperparameters of num estimators and max depth are important. The variable num estimators control the number of trees and more trees mean higher accuracy but slower performance. Moreover, the max depth controls the maximum number of levels of each tree like in the Decision Tree classifier. For simplicity, we set the max depth to the best preforming number as shown by our Decision Tree graphs.

3.5 Implementation Details

Our project was implemented in code with Python as it has many libraries and lots of documentation. For our data preprocessing, we used PyTorch [PGM+19] because it had a Resnet18 pre-trained model, functions to apply normalization and other preprocessing methods, and the Augmentor package [BSH17], which allowed us to apply transformations to the images with little code. For our models, we

chose the Scikit-Learn library [PVG+12] as it has all of the needed models and much more information. Lastly, we used Matplotlib's pyplot feature to visualize our results shown down below [Hun07].

4 Results

Evaluating our model's results is important in the process of improving them and checking our progress. In our case, we used two different metrics for scoring: accuracy and distance. Accuracy is simple– just add up all true positives or correct guesses and divide by the total number of predictions. On the other hand, distance, which measures how far the prediction is from the true label, is a little more complicated. Here, as shown by the equation below, we use Mean Squared Error (MSE), where we take the average of the square of differences between predicted and actual label diagnoses for all points 1 to n, which is the length of the dataset. We use MSE in order to eliminate negative values while also penalizing further distances with the exponent.

$$\text{Accuracy} = \frac{\text{True Positives}}{\text{All Predictions}} \qquad \text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(\text{Labels} - \text{Predictions})^2$$

4.1 Visualization

To visualize these two metrics across all models, we plotted them on graphs with two y-axes and one x-axis. The x-axis is the hyperparameter that we changed, while the two y-axes are accuracy and distance. The two different performance measurements were shown with different color lines, with red being accuracy and blue being distance.

4.2 Original Dataset

To start off, we tried our models on our default 3534 image dataset, as shown in Figures 1-4. With the KNN classifier, there were accuracies of up to around 0.6 and distances of 2.0, while the regressor had accuracies above 0.6 and distances as low as 1.1. Then, Logistic Regression had around 0.6 accuracy and 1.75 distance, while Linear Regression, which has no hyperparameters to plot, had an accuracy of around 0.42 and distance of 1.1. Next, Decision Tree classifier had accuracies under 0.6 and distances around 1.7, while the regressor had accuracies around 0.5 and distances around 1.6. Finally, the Random Forest classifier had accuracies above 0.6 and distances around 1.9, while the regressor had accuracies under 0.5 and distances around 1.1. With this dataset, it seems that the classifiers have better accuracy while the regressors have lower distance. However, it does seem clear that the best overall model is the KNN Regressor with num neighbors around 6.

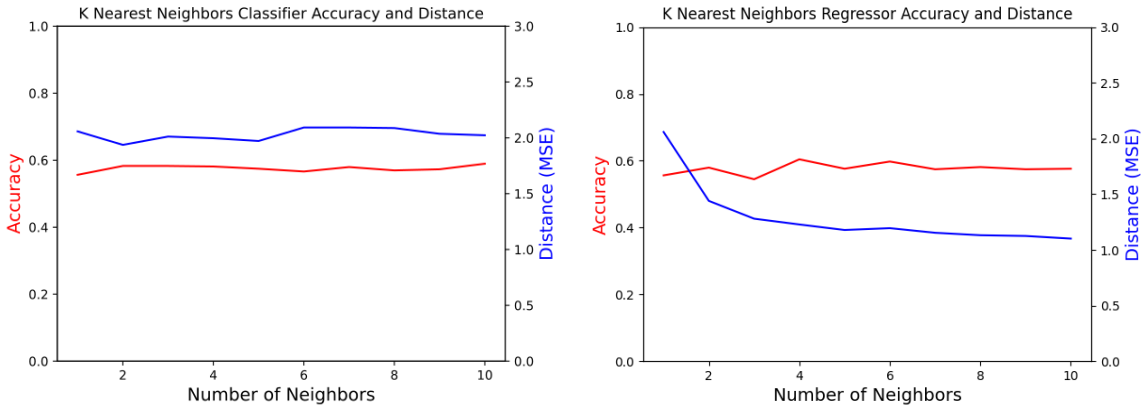


Figure 1: Here we measured the performance of 10 different KNN Classifier and Regressor models on our last layer Resnet18 features extracted from a fundus image while varying the number of neighbors.

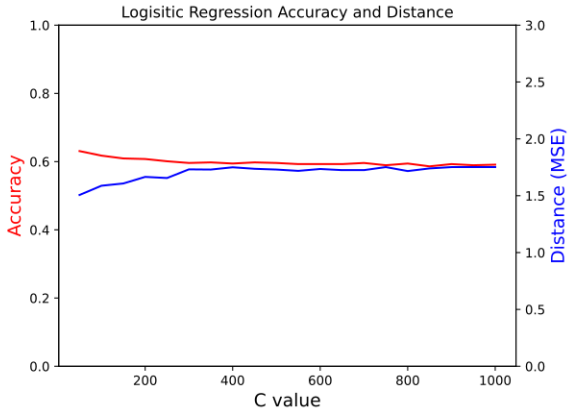


Figure 2: Here we measured the performance of 20 different Logistic Regression Classifier models on our Resnet18 features from fundus images while varying the C regularization hyperparameter.

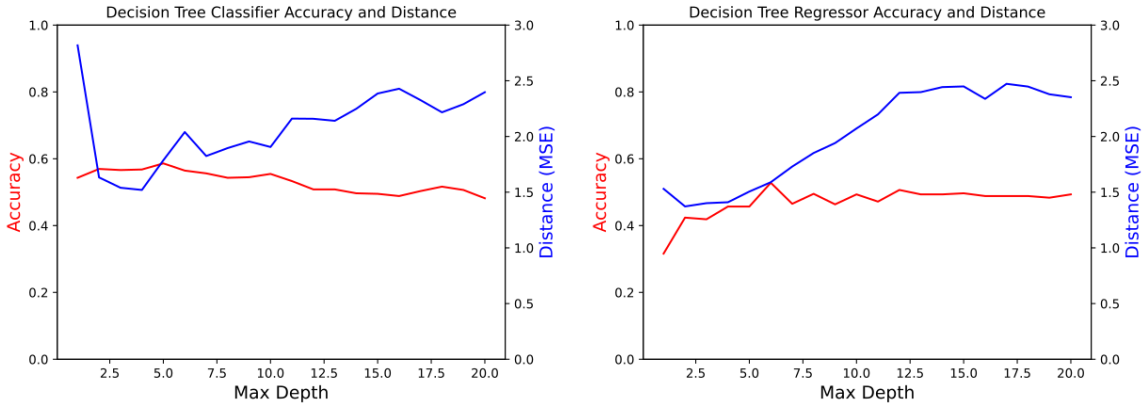


Figure 3: Here we measured the performance of 20 different Decision Tree Classifier and Regressor models on our fundus image features while varying the max depth hyperparameter.

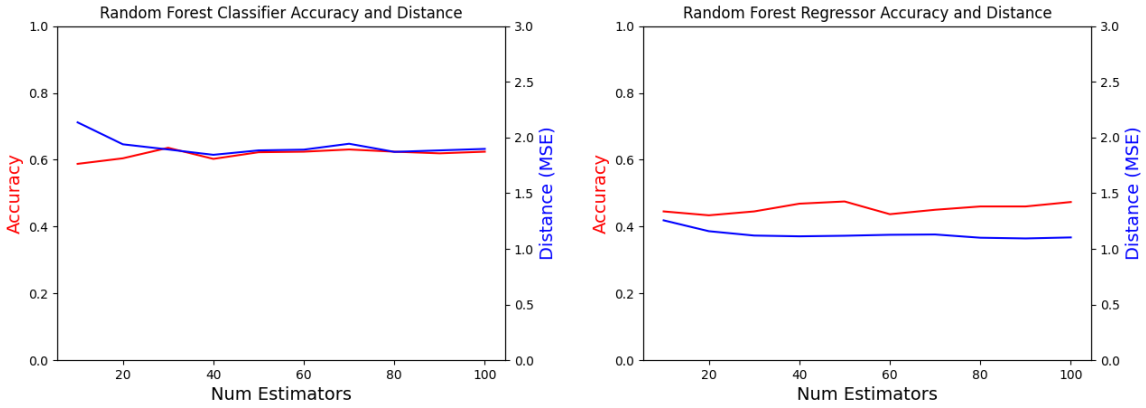


Figure 4: Here we measured the performance of 10 different Random Forest Classifier and Regressor models on our image features while varying the num neighbors hyperparameter.

4.3 Augmented Dataset

Then, we ran our model with the bigger, augmented dataset of around 30,000 images, as shown by Figures 5 through 8. With the KNN classifier, there were accuracies of above 0.6 and distances of 1.75, while the regressor had accuracies above 0.6 and distances as low as 1.1. Then, logistic regression had around 0.67 accuracy and 1.4 distance, while linear regression, which has no hyperparameters, had an accuracy of around 0.46 and distance of 1.04. Next, Decision Tree classifier had accuracies under 0.6 and distances above 2, while the regressor had accuracies under 0.5 and distances above 1.75. Finally, the Random Forest classifier had accuracies around 0.6 and distances under 2.5, while the regressor had accuracies above 0.4 and distances above 1. With this larger augmented dataset, it also seems that the classifiers have better accuracy while the regressors have lower distance. However, logistic regression with the C regularization hyperparameter as 300 is the best model.

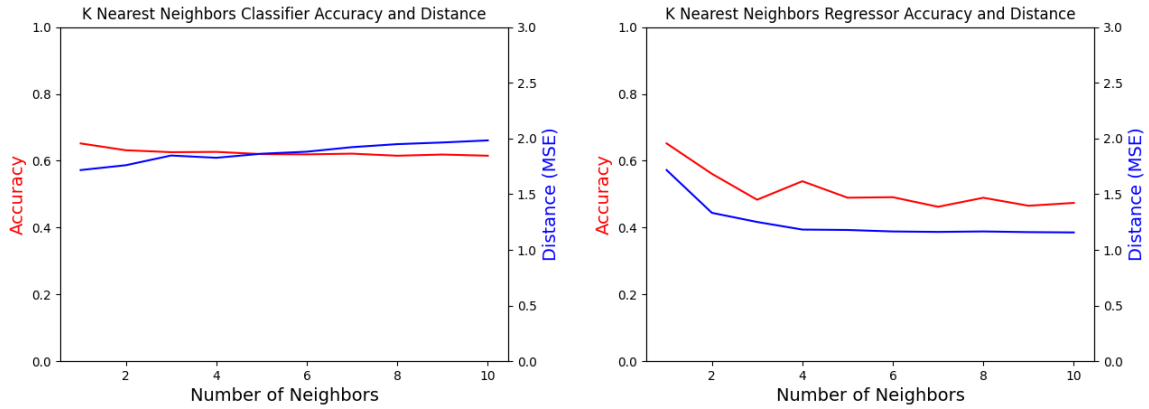


Figure 5: Here we measured the performance of 10 different KNN Classifier and Regressor models on our augmented dataset of 30,000 images with transformations while varying the num neighbors hyperparameter.

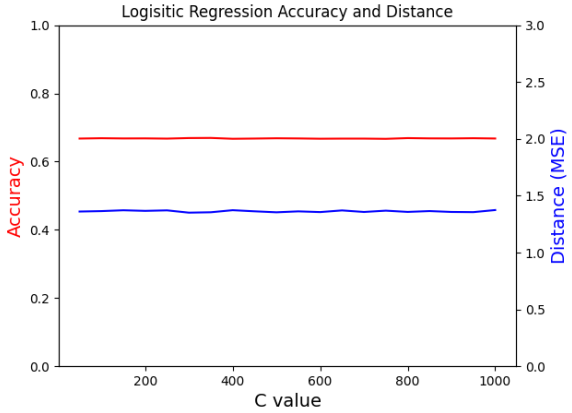


Figure 6: Here we measured the performance of 20 different Logistic Regression Classifier models on our augmented dataset while varying the C regularization hyperparameter.

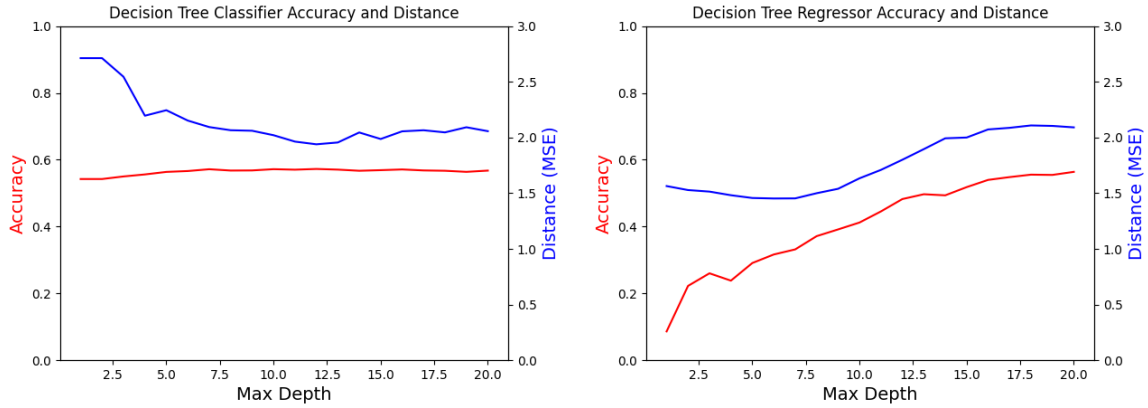


Figure 7: Here we measured the performance of 20 different Decision Tree Classifier and Regressor models on our augmented dataset while varying the max depth hyperparameter.

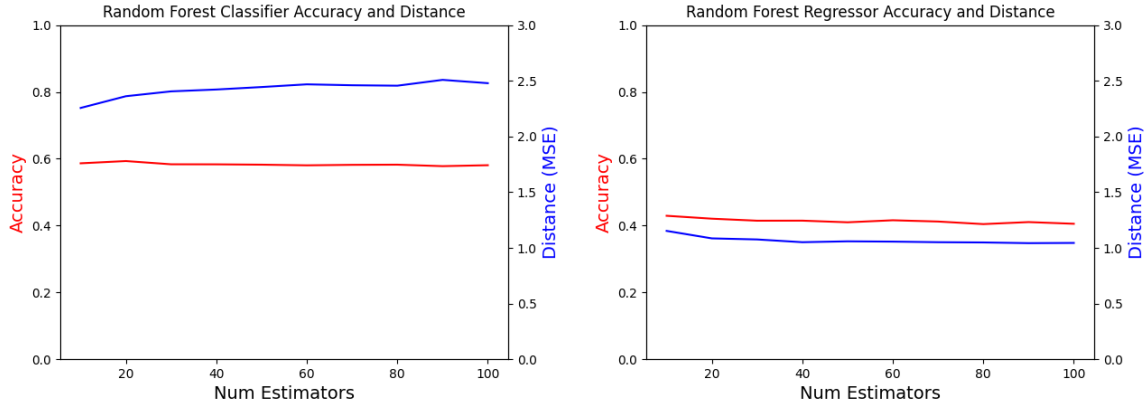


Figure 8: Here we measured the performance of 10 different Random Forest Classifier and Regressor models on our augmented dataset while varying the num estimators hyperparameter.

4.4 Augmented Dataset With Detection Task

Finally, our accuracies and distances were a little far off of what we were looking for, so we transformed our task into a detection one, meaning outputting 0 or 1 for diseased or not. This can be seen in Figures 9 through 12. With the KNN classifier, there were accuracies of under 0.8 and distances of around 0.25, while the regressor had similar accuracies and distances as low as 0.1. Then, logistic regression had around 0.83 accuracy and around 0.2 distance, while linear regression had an accuracy of around 0.83 and distance of 0.13. Next, Decision Tree classifier and regressor had accuracies under 0.7 and distances around 0.3. Finally, the Random Forest classifier had accuracies above 0.8 and distances under 0.3, while the regressor had accuracies around 0.8 and distances under 0.2. With this larger augmented dataset and modified detection task, it also seems that the classifiers and regressors are more similar, with individual models being more divergent. Overall, the best models were the linear models and random forest models.

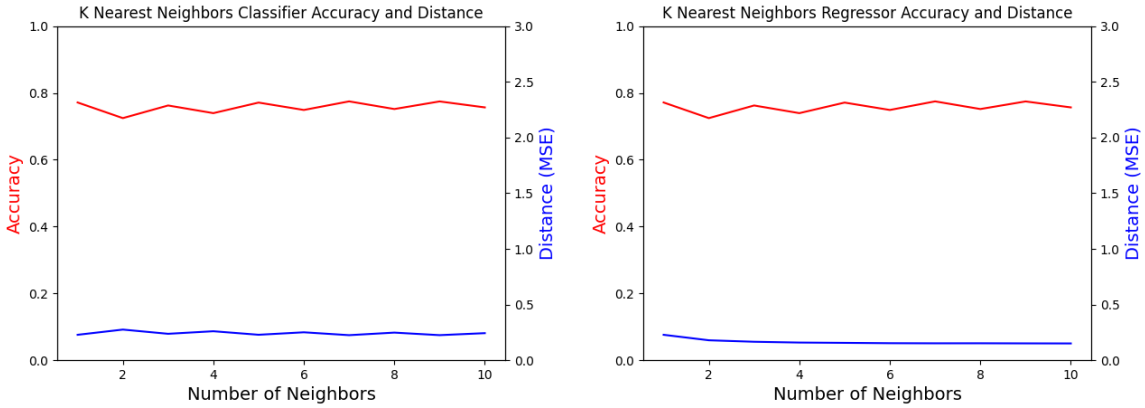


Figure 9: Here we measured the performance of 10 different KNN Classifier and Regressor models on our augmented dataset with the detection task (0 or 1 labels) while varying the num neighbors hyperparameter.

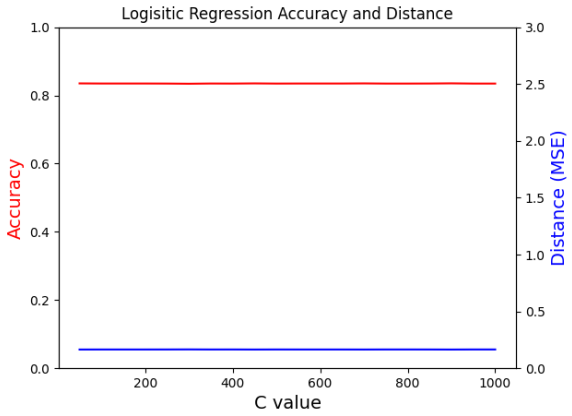


Figure 10: Here we measured the performance of 20 different Logistic Regression Classifier models on our augmented dataset with detection while varying the C regularization hyperparameter.

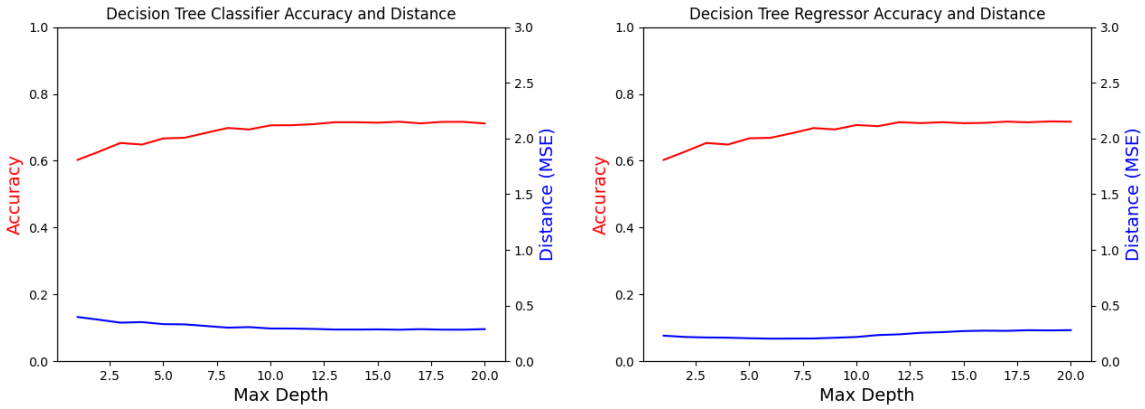


Figure 11: Here we measured the performance of 20 different Decision Tree Classifier and Regressor models on our augmented dataset with detection while varying the max depth hyperparameter.

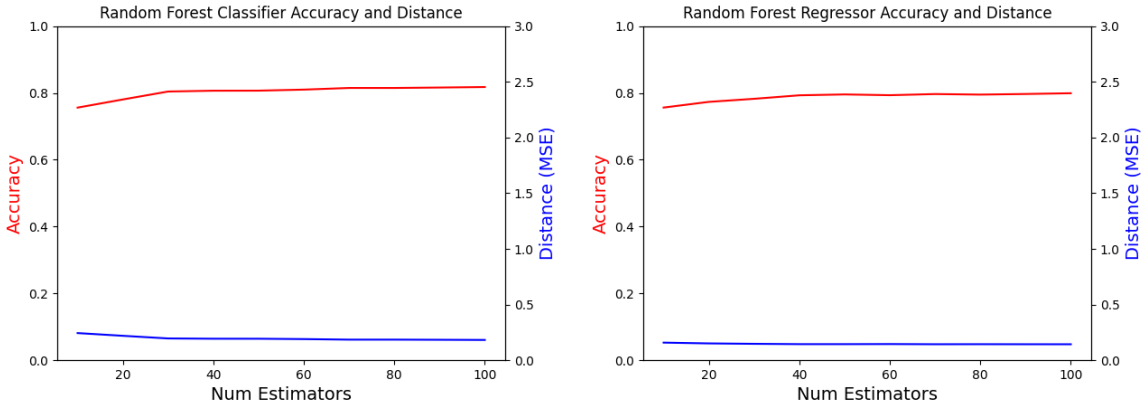


Figure 12: Here we measured the performance of 10 different Random Forest Classifier and Regressor models on our augmented dataset with detection while varying the num estimators hyperparameter.

5 Conclusion

Utilizing several different regression and classifier models, we were able to detect diabetic retinopathy at a relatively high accuracy and low distance. For the augmented dataset, we were able to achieve up to 0.67 accuracy with logistic regression, which is a lot better than the 0.167 accuracy obtained from random guessing from 0 to 5. For our detection task, we were able to obtain 0.83 accuracy with linear regression, but this was similar to just guessing 1 all the time. Either way, our results ultimately show that linear models are the best with our task. In the future, more extensive tests on each model including more extensive data preprocessing, more hyperparameters, a bigger range of hyperparameters, and more performance metrics may be needed to better prove our observations.

References

[BSH17] Marcus D. Bloice, Christof Stocker, and Andreas Holzinger. Augmentor: An image augmentation library for machine learning. *CoRR*, abs/1708.04680, 2017.

[dep] Color fundus photography.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[kag] Aptos 2019 blindness detection.

[log] Logistic regression.

[nat] Diabetic retinopathy.

[NN] Angela Nelson and Brunilda Nazario. Diabetic retinopathy stages.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[PVG+12] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu

Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490, 2012.

[RDS+14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[reg] Regularization.