Project 1: Classification using Bayesian Decision Rule (Parametric Learning vs. Non-Parametric Learning)

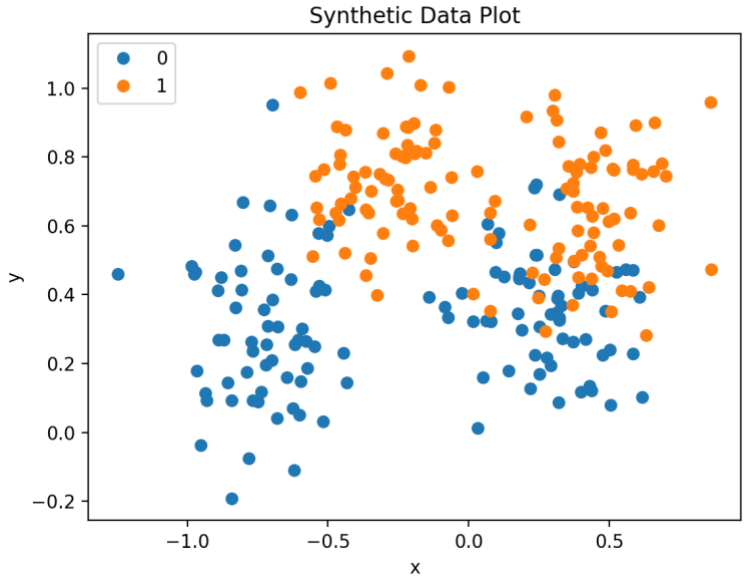
Owen Queen
COSC 522 – Machine Learning

For this project, we were asked to examine two datasets: a 2-feature, balanced dataset containing synthetic data (the *synth* dataset), and a 7-feature, unbalanced dataset containing information about diabetes occurrence in a sample of Pima Indians (the *Pima* dataset). We will examine the use of Bayesian Decision Theory to build several parametric classifiers and nonparametric classifiers for these datasets as well as the corresponding results from the models.

This report is broken down into sections. The two sections correspond to the major bullet points on the writeup for this assignment while the subsections (denoted by letters a, b, c, …) represent the smaller bullet points underneath the sections.

Section 1: Implement parametric learning using the three cases of discriminant functions with Gaussian pdf.

a. A scatterplot of the classes in the synthetic dataset is shown below:



From a visual inspection, we can see that this dataset appears to be bimodal. This is especially evident in the "0" class, where there seems to be a cluster of points centered around (-0.75, 0.3) and another cluster centered around (0.4, 0.3). Therefore, a single-model Gaussian might not be a very good choice of pdf for this model. A 2-modal Gaussian may be more appropriate for being able to model the spread of this data.

b. Results for running these models on two different datasets with equal prior probability is shown below:

Table 1: Synth dataset:

Discriminant Function	**Overall Classification Accuracy**	**Class "0" Accuracy**	**Class "1" Accuracy**	**Run time**
Euclidean (Case 1)	0.71300	0.68000	0.74600	0.21266 sec
Mahalanobis (Case 2)	0.88500	0.89600	0.84700	0.21983 sec
Quadratic (Case 3)	0.89800	0.90800	0.88800	0.37919 sec

Table 2: Pima dataset

Discriminant Function	**Overall Classification Accuracy**	**Class "Yes" Accuracy**	**Class "No" Accuracy**	**Run time**
Euclidean (Case 1)	0.75000	0.72477	0.76233	0.08813 sec
Mahalanobis (Case 2)	0.76205	0.73394	0.77578	0.08435 sec
Quadratic (Case 3)	0.73795	0.61468	0.79821	0.14034 sec

Note: These classification routines were run on my Macbook Pro.

c. Discussion:

The results in these tables show differing levels of accuracy and run time for each of the parametric learning algorithms. In the synth dataset, we see an increase in accuracy as the complexity of the discriminant function increases; however, this increase in accuracy comes at the expense of run time. Conversely in the Pima dataset, we see a higher accuracy rating for the simple discriminant functions while the run time for the more complex function, the Quadratic discriminant, remains relatively higher.

Before we discuss these results in detail, it is worthwhile to establish some expectations for our different discriminant functions. We can think of our different cases for discriminant functions as being of varying degrees of assumptions. For Case 1 (Euclidean), we make the most significant assumption: our features are statistically independent and have the same variance. For Case 2 (Mahalanobis), we make a slightly more relaxed assumption: the covariance matrices for all of our classes are equal. Finally, for Case 3 (Quadratic), we make only the assumption that the distribution of data for every class is Gaussian. Please note that we also make the assumption of a Gaussian distribution across classes in Case 1 and Case 2.

Therefore, Case 1 is our most restrictive assumption, and Case 3 has the most relaxed assumption. Because Case 3 makes no assumptions about the variance of our data, it allows us to model the spread of our data more accurately than either Case 1 or Case 2. Similarly, Case 2 models the spread of the data better than Case 1. However, as the assumptions are relaxed, we would expect an increase in run time because the calculations

required become more complex. Thus, we expect the more complex functions (those with the most relaxed assumptions) to have the higher accuracy and higher run time, and we expect the less complex functions (those with the more restrictive assumptions) to have the lower run time and lower accuracy.

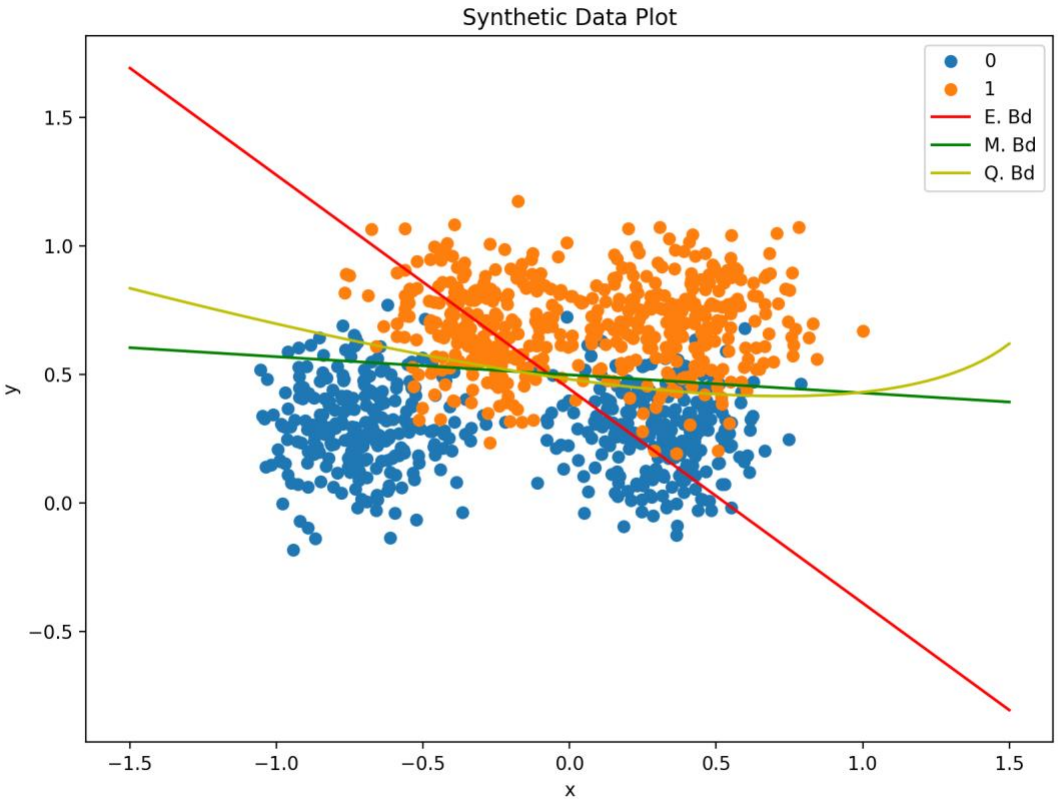
The synth dataset exhibits these expected results. The Quadratic discriminant function performs best on the testing data with almost a 90% overall classification accuracy, an improvement of 19% from the Euclidean function. However, this performance comes at the expense of run time; this algorithm ran at 0.37919 seconds as opposed to only 0.21983 for the Mahalanobis function. This sacrifice in run time might be acceptable if the Quadratic function greatly improved accuracy, but this function only exhibits a roughly 1% increase from the Mahalanobis function. Therefore, we can see that the Mahalanobis function may be the best choice for a discriminant function in this instance due to its high accuracy rating coupled with its relatively short run time.

However, the results from the Pima dataset show different results than for the synth dataset. The Quadratic function has the worst performance on the testing data, with a lower accuracy than the other two discriminant functions while still running slower (almost 2x slower than Mahalanobis). For this testing data, the Mahalanobis function had the highest classification accuracy while achieving the lowest run time. In addition, we can see a lower classification accuracy for the "Yes" class than the "No" class across every algorithm used. This might be explained by the imbalance in classes in both our training and testing sets with both of the Pima datasets containing more "No" individuals than "Yes".

Therefore, it appears as if the Mahalanobis discriminant function was, overall, the optimal function for our classification algorithm on these datasets. It had a low run time while achieving the highest classification accuracy on the Pima dataset and the second-highest classification accuracy on the synth dataset. Its runtime is very close to the Euclidean function; this could be explained by the fact that both algorithms do not require a covariance matrix to be calculated at every iteration, unlike the Quadratic function. In my implementation, I calculate the variance for Euclidean and covariance matrix for Mahalanobis once at the beginning of the iterations. Therefore, the difference in their run time should be equal to the difference between these two front-end operations.

d. Decision Boundaries:

I calculated the decision boundaries for each of the discriminant functions by hand using linear algebra techniques. Then, I implemented these decision boundary equations using numpy in Python, generating a plot of the curves for each of these boundaries. Below is a plot of the decision boundaries for the different discriminant functions used on the synth training data:



These decision boundaries are clearly different for each of the discriminant functions. We see that the Euclidean and Mahalanobis functions produce linear decision boundaries while the Quadratic function produces a quadratic decision boundary.

The Euclidean boundary seems to run between the centroids of the data clouds. If we let the straight line between the centroids of the data clouds of each class be L, then we can almost think of the Euclidean boundary (E) as being the vector that bisects and is orthogonal to L (from a visual analysis).

The Mahalanobis boundary seems to account more for the spread of the two data clouds. Therefore, instead of simply bisecting the centroids of the two clouds, the boundary is weighted due to the spread, making it tend towards the vertical separation of the two classes. Visually, it seems as if the Mahalanobis discriminant partitions the classes across the boundary more accurately than the Euclidean boundary.

The Quadratic boundary seems to be the most accurate boundary; however, it is not much more accurate than the Mahalanobis. As we can see from our numerical results, the Mahalanobis and Quadratic functions have similar classification rates. This can be verified visually in this plot, because the mean distance between the Quadratic boundary and the Mahalanobis boundary is relatively small. This boundary is the only one that takes the form

of a quadratic function, allowing it to more accurately fit this data that does not have a clear linear separation between the two data clouds.

e. In the plots below, we can see the class-0 accuracy on each dataset. For these plots, I was unsure of what to exactly to plot: class-0 accuracy for given predictions or class-0 accuracy for the entire testing data. Therefore, I plotted both of these metrics for each prior probability. The methods will be discussed in detail below

Synth dataset:

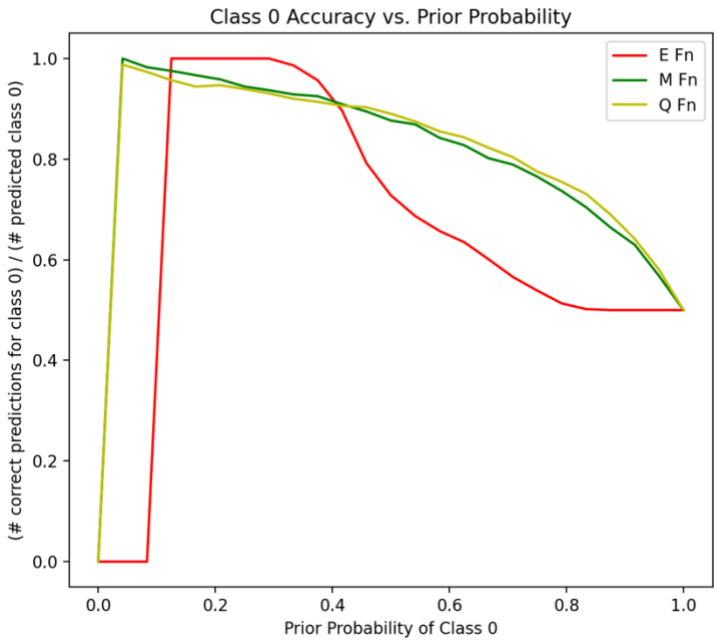


Figure Synth A.

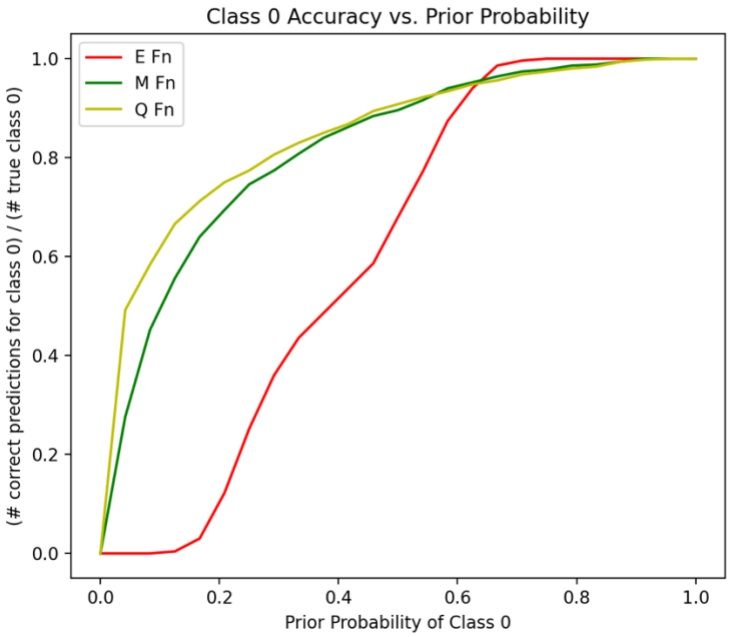
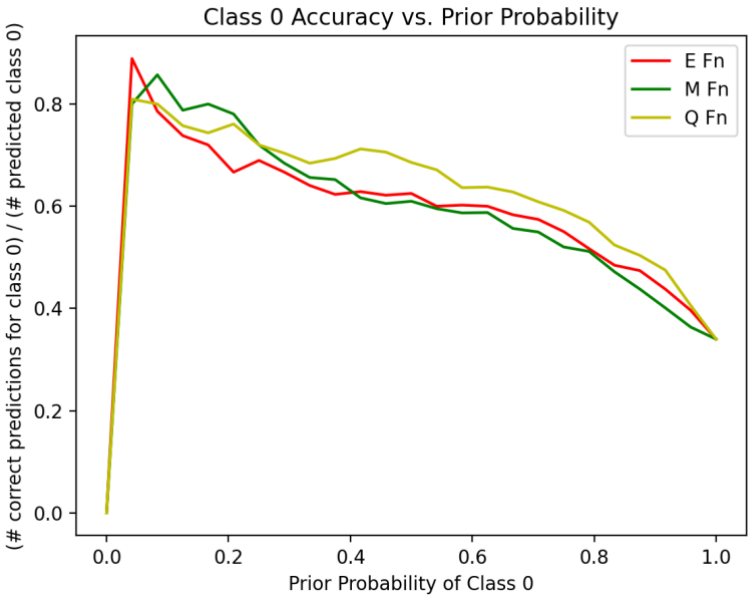
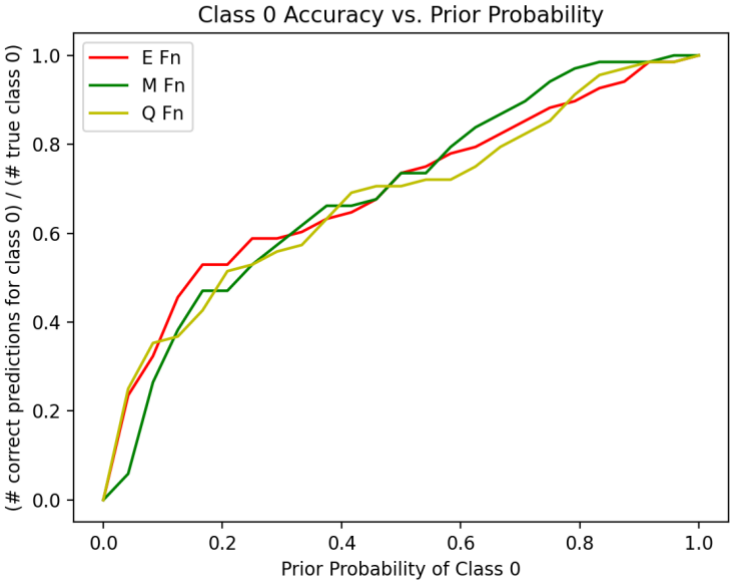


Figure Synth B.

Pima dataset:





For these plots, the y-label indicates the method that I used to check accuracy on class 0 for my predictions. In the plots to the left, I calculated the accuracy of class-0 predictions against the total number of predictions made for class-0. The formula was:

$$\frac{(\#\ correct\ predictions\ for\ class - 0)}{(total\ \#\ predicted\ class - 0)}$$

It should be noted that we would expect this value to be 0 when the prior probability for class-0 is 0; this is because we know that our model is not predicting any testing samples to be of class-0 (because we are adding ln(0) to our discriminant function for our models, which approaches negative infinity). We would also expect to see all of our functions approach the true ratio of class-0 to class-1 testing samples because when the prior probability for class-0 is 1, then class-1's prior probability is zero; thus the model predicts every testing sample as class-0. In this case, the accuracy of the classifier will only be as great as the ratio of class-0 samples in the testing dataset.

In the plots to the right, I calculated the accuracy of predictions against the total number of true class-0 individuals. This was the same metric by which I measured the class accuracies for the different discriminant functions above. Therefore, the formula for this metric was:

$$\frac{(\#\ \text{correct predictions for class } 0)}{(\text{total } \#\ \text{true class} - 0\ \text{testing samples})}$$

We would expect this metric to produce a graph that starts at 0 and ends at 1. When the prior probability for class 0 is 0, then we don't predict any testing samples for this class. However, when the prior probability is 1, we predict all samples as class 0, so this trivial model correctly classifies every training sample that belongs to class 0.

Both of these metrics can be informative as to how the prior probability affected classification accuracy. Prior probabilities for class 0 were varied from 0 to 1 incrementing by 0.04; this subsequently made the prior probability for class 1 equal to 1 – P(class 0).

In the results involving the synth dataset, we can see a distinct difference in how the Euclidean model is affected by the change in prior probability versus how the Quadratic and Mahalanobis models are affected by it. In Figure Synth A., we can see that the graph for accuracy stays near 1 for longer for the Euclidean model while the Quadratic and Mahalanobis models seem to show a smoother downward trend in class-0 accuracy. Their similar behavior may be due to the proximity of the Quadratic and Mahalanobis decision boundaries (as can be seen in the plot above in part D).

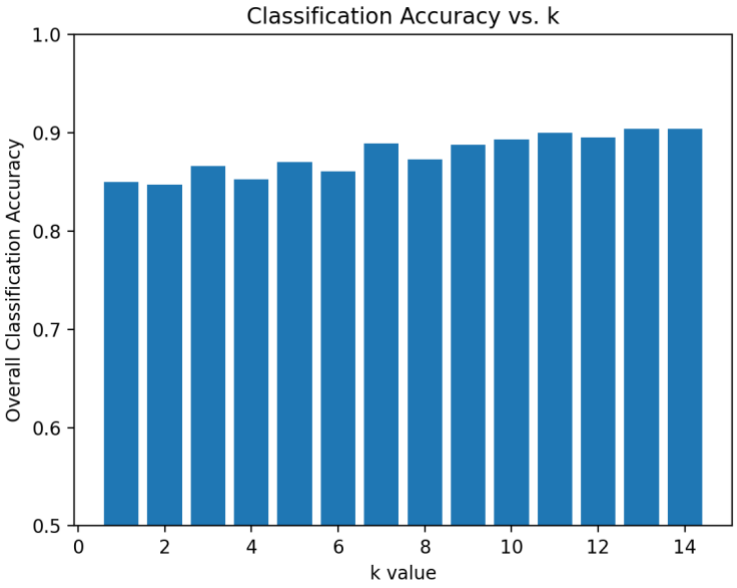
In the results involving the Pima dataset, the results seem very similar for each of the discriminant functions. Considering how close their overall accuracies are on this dataset, we could expect these results to be similar. It can be reasonably assumed that our decision boundaries are very close to one another due to these similar accuracies. Because we assume a Gaussian distribution on our data, our prior probability manifests itself as an

additive term in our final discriminant function. Therefore, this addition affects the discriminant functions in a constant way; in other words, the prior probability does not scale with **x**. Therefore, if the decision boundaries are close to each other, then the prior probability will affect those discriminants in a similar way, no matter the degree of the boundary. These results may be indicative of these similar boundaries within the models.

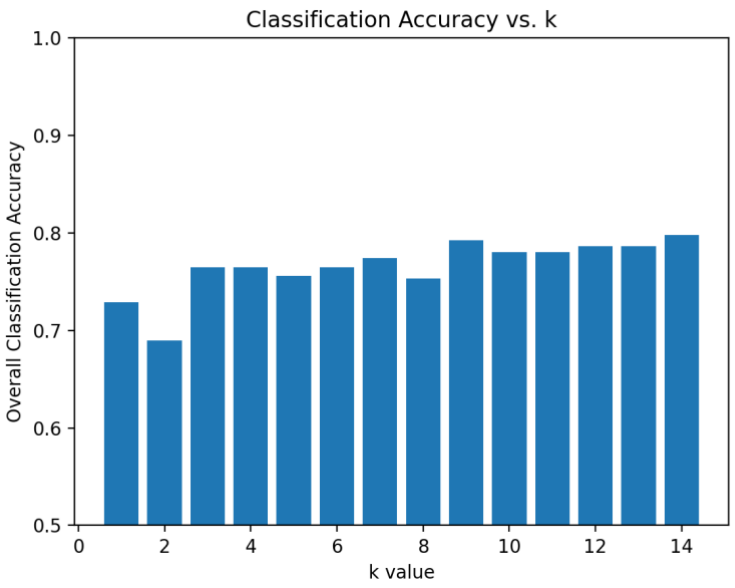
Section 2: Implement nonparametric learning using kNN.

a. Below are two plots of classification accuracies for different *k* values (ranging from 1 – 15). The first plot is for the accuracy on the synth dataset, and the second is for the accuracy on the Pima dataset:

Synth Dataset



Pima Dataset



- The highest classification rate on the synth dataset was achieved by *k = 13* at 0.904 overall accuracy.
- The highest classification rate on the Pima dataset was achieved by *k = 14* at 0.79819 overall accuracy.

Note: For the Pima dataset, I tested larger *k* values to see if we could achieve a higher accuracy because *k = 14* seems to show a much larger accuracy rating. However, these higher *k* values only yielded a marginal increase in accuracy (< 2%), so I decided to use *k = 14* for simplicity in the model.

<u>Observations:</u>

From these plots, we may make the assumption that a larger k-value produces a greater accuracy in our predictions; however, we must be careful in making this

assumption. After running these models for higher k-values, I found that the accuracy levels off, and it eventually begins to decrease. If we think about a scenario where we are using a k-value that is very large (i.e. close to the number of training samples), then our classifier will tend towards classifying every testing sample as the most prevalent label in our dataset. Therefore, large k-values tend to be biased towards the relative frequency of labels in our dataset.

As a result, we want to keep our k-values relatively small to avoid this bias. In these figures, we can actually see that accuracy does not have much variance across different k-values (~3% varying rates in synth dataset, ~8% in the Pima dataset), so choosing a smaller k, even if we do have incremental increases in classification accuracy for larger k-values, will avoid bias towards frequencies in testing data. While we do not want a very large k-value, we also want to avoid very small k-values (< 4). Small k-values will be subject to local variances and errors, allowing local results and potentially outliers to affect the prediction label of a testing sample. In conclusion, choosing a k-value is a balance between these two extremes, and while these plots exhibit increased accuracy at higher k-values, this may not necessarily be the best choice for our model.

b. Results

Table 1: Synth dataset:

Discriminant Function	**Overall Classification Accuracy**	**Class "0" Accuracy**	**Class "1" Accuracy**	**Run time**
Euclidean (Case 1)	0.71300	0.68000	0.74600	0.21266 sec
Mahalanobis (Case 2)	0.88500	0.89600	0.84700	0.21983 sec
Quadratic (Case 3)	0.89800	0.90800	0.88800	0.37919 sec
kNN (k = 13)	0.90400	0.91400	0.89400	55.8491 sec

Table 2: Pima dataset

Discriminant Function	**Overall Classification Accuracy**	**Class "Yes" Accuracy**	**Class "No" Accuracy**	**Run time**
Euclidean (Case 1)	0.75000	0.72477	0.76233	0.08813 sec
Mahalanobis (Case 2)	0.76205	0.73394	0.77578	0.08435 sec
Quadratic (Case 3)	0.73795	0.61468	0.79821	0.14034 sec
kNN (k = 14)	0.79819	0.60550	0.89238	14.8176 sec

c. Discussion about Parametric vs. Nonparametric learning

Parametric and nonparametric learning both have their separate benefits and limitations. Parametric learning offers a fast, streamlined method with parameters that can easily be tweaked; however, because it requires previous knowledge of the dataset, this may be undesirable with datasets in which their structures are unknown. Nonparametric

learning can be more accurate on datasets with unknown structures because these models do not require previous knowledge of the dataset. However, this increased accuracy comes at the expense of runtime. Both methods, including their results in predicting the synth and Pima datasets, will be analyzed in this discussion.

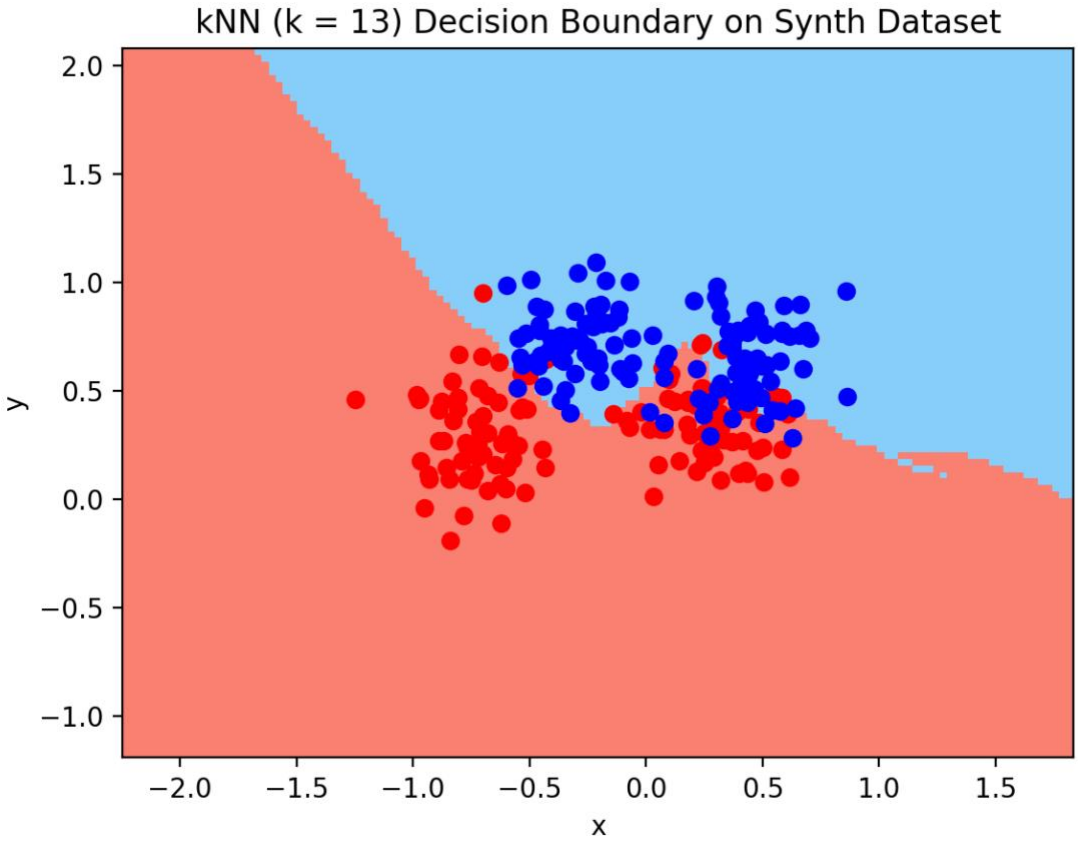
As we can see from the tables, nonparametric learning increases accuracy over parametric learning. We can observe only a slight improvement ($\sim$0.006) in accuracy on the synth dataset for kNN versus the most accurate parametric algorithm (Quadratic). However, in the Pima dataset, there is almost a total 4% improvement in total accuracy, and we can observe higher class-wise accuracies in both classes. This higher improvement in the Pima dataset may be explained by the unusual structure of the dataset. This dataset seems to be less structured than the synth dataset; this means that our assumptions for parametric learning may not necessarily hold. Thus, nonparametric learning, where we avoid these assumptions, may be the better choice in order to capture the unstructured nature of these data. This is one of the benefits of nonparametric learning; however, this benefit does not come without costs.

One of the main limitations of nonparametric is the slow runtime compared to parametric learning. Since kNN, our nonparametric learning algorithm, uses an iterative method, exhaustive search, as opposed to MLE optimization in parametric learning, it takes longer to generate a prediction for testing data. Training time for parametric learning is slightly longer: depending on the method, we must calculate the mean vectors and possibly covariance matrices. In contrast, kNN's training phase only involves storing the training data. However, testing time is much slower for kNN. While parametric learning only involves calculating your discriminant function for each class, kNN requires you to calculate distance from the testing sample to all other training samples. This would make kNN disadvantageous in situations where we require fast testing times, such as search features on a website or real-time traffic analysis. While we do see an increase in accuracy from kNN, this comes at the cost of runtime. Therefore, when deciding which algorithm to use, we must decide whether the increase in accuracy is worth the increase in runtime.

d. Decision boundary for kNN

With the parametric learning algorithms, we were able to draw explicit functions with decision boundaries for each of the different discriminant functions. However, kNN involves more localized analysis in order to predict a testing sample. Therefore, we cannot draw an explicit function because this function would need to be based on localized results instead of means and covariances – continuous properties of the entire training dataset – that we used in parametric learning.

In order to plot the decision boundary, we will have to use an iterative approach. If we take a mesh of the area containing the training data points, we can generate a prediction for each point in that mesh. Then, we can assign a color to that point (effectively a pixel), and we can plot this mesh alongside the training sample points. The results of doing this is shown below:



This plot illustrates which class the points on the graph, should they be a testing sample, would be assigned to. The red points and area represent the "0" class, and the blue points and area represent the "1" class on the synth dataset.

Extra Credit

1. Is there any way to change the prior probability of kNN? Implement it and compare performance.

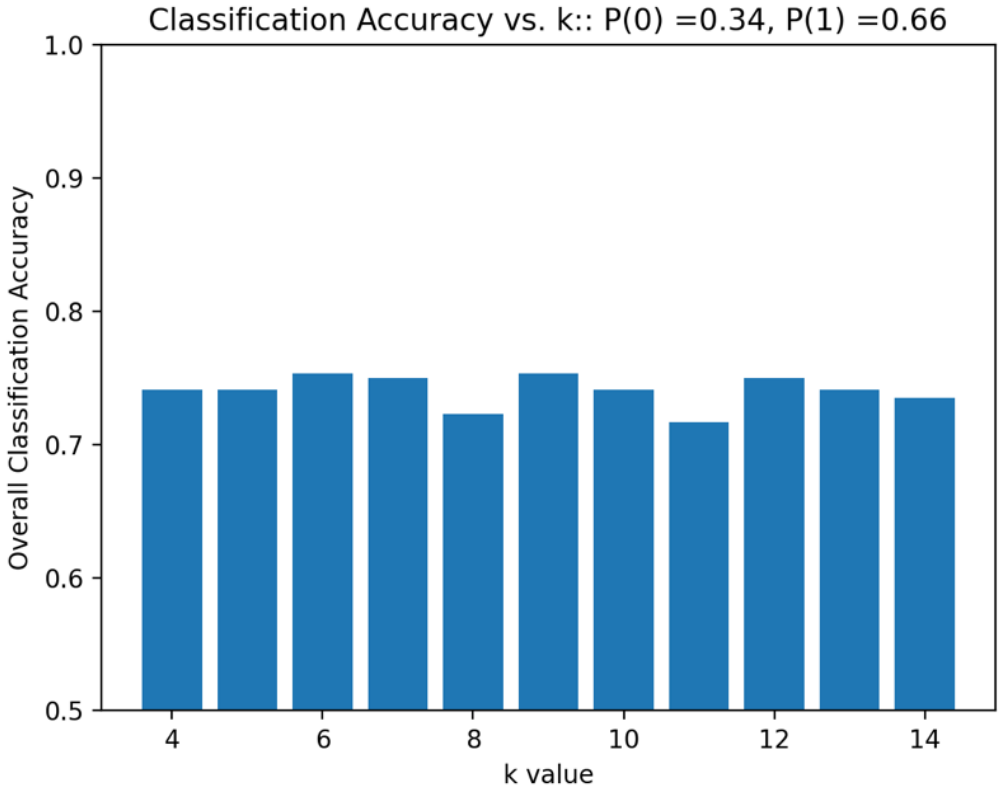
Changing the prior probability is equivalent to weighting the classification based on prior knowledge of our dataset. Implementing prior probability in parametric learning was very apparent; however, prior probability in kNN will have to be different.

One way that we could think about applying this idea is by weighting the votes of each of the neighbors during our classification. The kNN algorithm uses a discriminant function that is equivalent to k_m/k where m denotes the class index and k denotes the number of nearest neighbors (training samples with smallest distance from our testing

sample) to consider. By the maximum posterior probability principle, we choose the class that has the largest probability when evaluated against our testing sample in this discriminant function. Since k remains constant with respect to each calculation of this posterior probability, then we effectively choose the class that has the maximum k_m value.

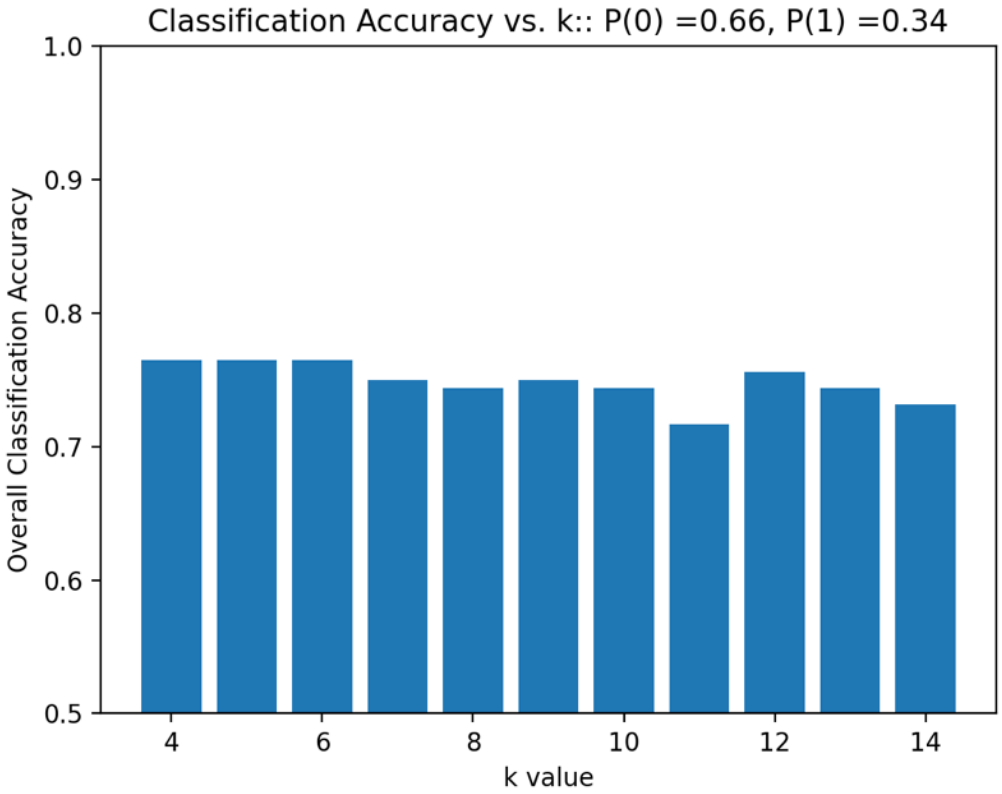
Instead of diving into the derivation of this discriminant function, we can simply assume that our weighting procedure is equivalent to multiplying our prior probability by some constant. Therefore, we can multiply the k_m/k value at the end of our discriminant function to implement this constant in our function. The question then becomes: what constant should we use? This is what we will answer with some analyses below:

First, I decided to test using a prior probability that reflected the proportions of certain labels in the dataset. I calculated the frequency of class 0 in the Pima testing data and used this as my prior probability for class 0. Note that the frequency of labels in the synth training data were equal, so I only ran these first analyses on the Pima dataset.



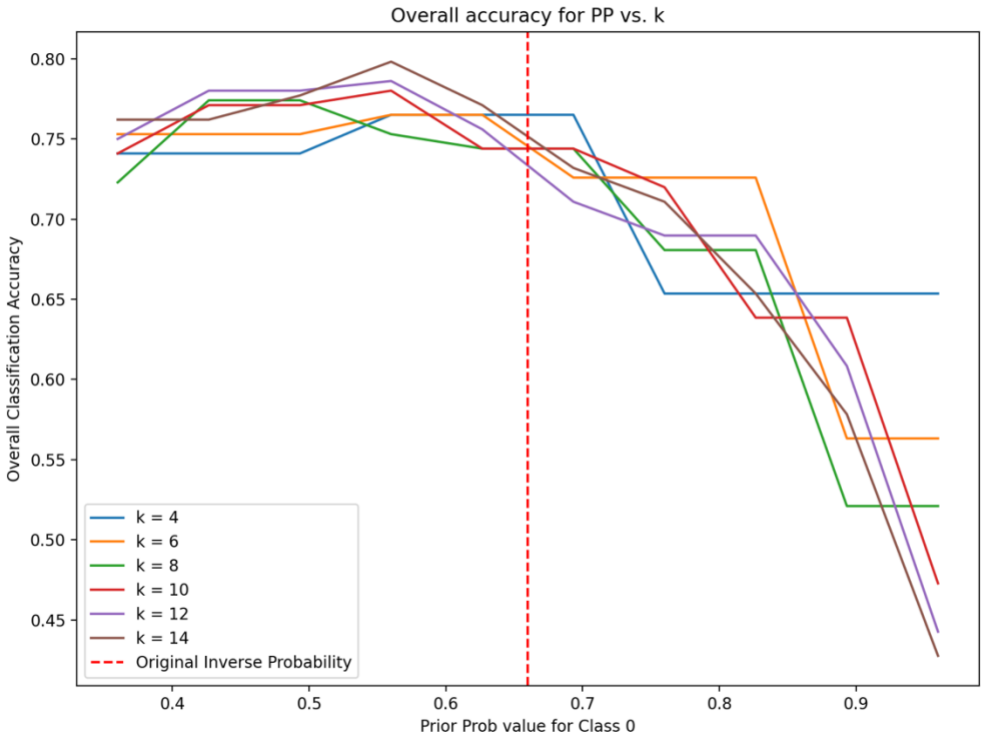
The k values that yielded the highest accuracies were k = 6 and k = 9, both yielding an overall classification accuracy of 0.75301. However, their class-wise accuracies for class 0 were much lower than with equal prior probabilities: these accuracies varied from between ~0.27 - 0.40.

Since we achieved lower accuracy results on average using these prior probabilities, I decided to flip the probabilities. I used the frequency of class-0 labels to be the prior probability for class-1 and vice versa. This was effectively implementing the inverse of the proportions of each class in the testing dataset. The hope is that this would act as a normalization procedure, accounting for imbalances in the original proportion of labels in the training data. The results for this are shown below:

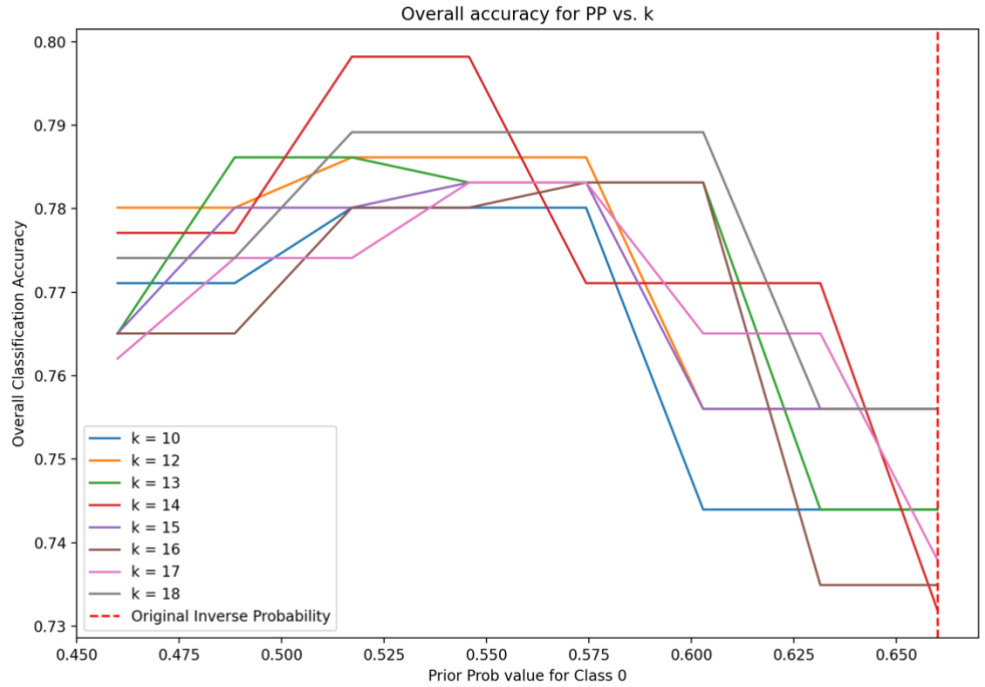


These accuracies were higher than for the directly-proportional prior probabilities, but they were still lower than the original numbers assuming equal prior probability. The highest accuracies were achieved at lower k values: 4, 5, and 6 had an overall accuracy of 0.76506. However, the class-wise accuracies were higher than in our original trial (see Section 2, Part B, Table 2) , with both class accuracies ranging from ~0.65 – 0.8.

Since both of these models had a lower overall classification accuracy than our original model, I decided that a more exploratory approach might help us find better accuracy in our classification. It is likely that our training data will not contain the same proportions as our testing data; however, it is reasonable to assume that the true proportions of labels within the testing data are close to those in the training data. Therefore, I wrote a procedure that varied the prior probabilities from in a 0.3 radius around our inverse proportions. Therefore, with P(1) = 1 – P(0), we would test P(0) values from [0.66 – 0.3, 0.66 + 0.3]. I decided to vary the k value in order to explore all possibilities in boosting our classification accuracy. The results are shown below:

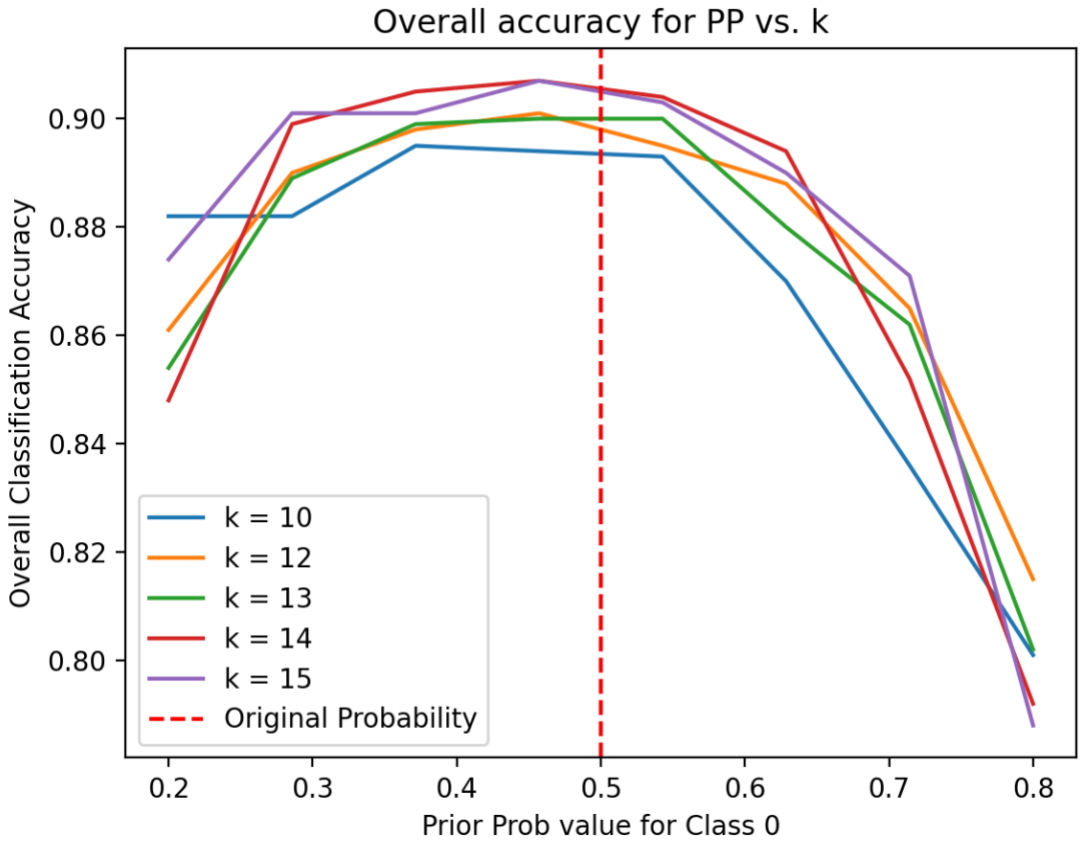


The highest accuracy of 0.79819 was achieved by k = 14 at P(0) = 0.51714. Note that this value is the exact same as the accuracy achieved by k = 14 at P(0) = 0.5 (the results shown in Section 2, part B, Table 2); this is most likely due to the similarity of these two prior probability values. I tried to test larger k values around the prior probability values shown to yield the highest results here (0.4 – 0.6), but the k = 14 model still seemed to have consistently-greater accuracy. This plot is shown below:



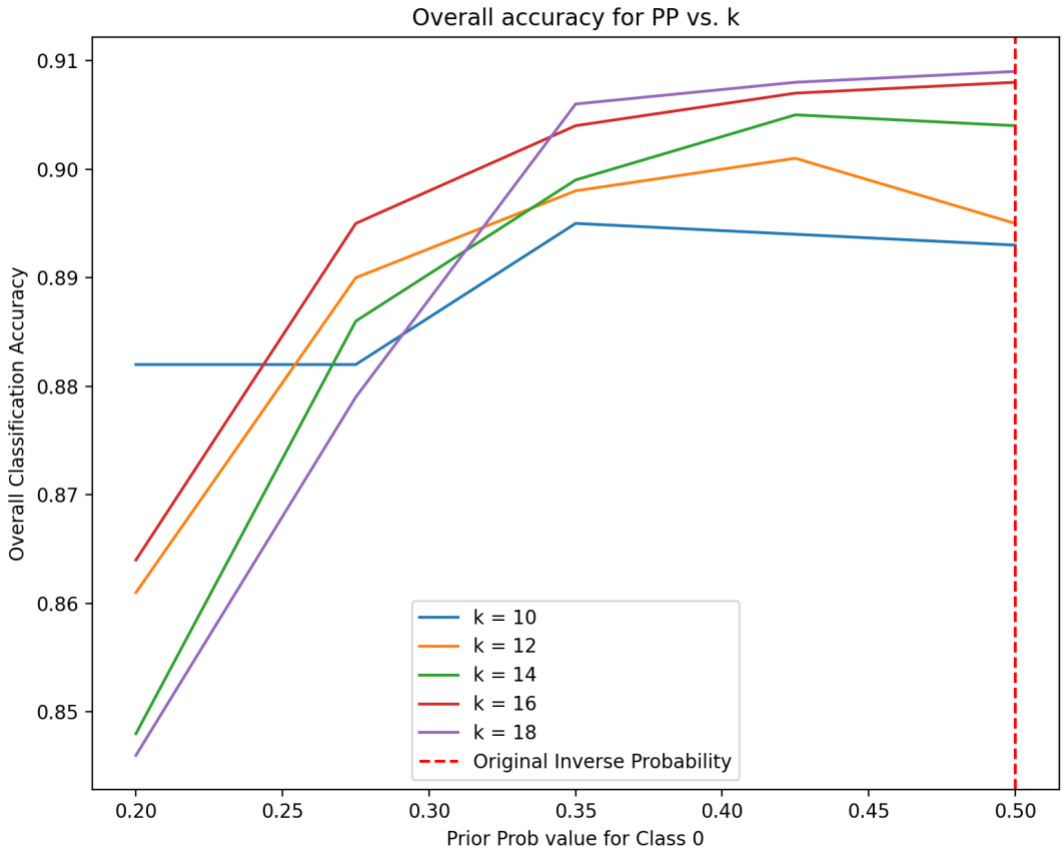
Knowing this, I decided to test only the k = 14 model around the region of 0.5 (0.45 – 0.55) to see if we could get a better accuracy. The greatest accuracy shown was still at a prior probability of 0.5. Therefore, I was not able to find a prior probability that increased the accuracy of our classifier. From my ad hoc methods, I can say with reasonable confidence that varying the prior probabilities to be different than 0.5 for both classes does not yield higher classification accuracy on the Pima dataset.

I also tried to see if I could achieve greater results with the synth dataset using this iterative approach to explore different prior probabilities. I hypothesized that this method would not work on the synth dataset because it is more balanced that the Pima dataset; therefore, I thought that keeping the prior probability at 0.5 for each class would accurately model the frequency of each class in the testing data. Nevertheless, I went forward with this trial, and the results are shown below:



Please not that since the exact value 0.5 was not tested during this process, we observe the k = 13 model to have lower accuracy than it actually does when the prior probability is 0.5 (see Section 2, part B, Table 2). We can see that the k = 14 model and k = 15 models have the highest accuracies, but their values are still around the same as the k = 13 model with prior probability being 0.5 (0.904). Seeing that these models seemed to achieve higher probabilities at-and-below the 0.5 prior probability mark and that models

with large k's seemed to perform better, I decided to test some larger k models that I did not previously test on this dataset at values below 0.5. The results from this trial are shown below:



From this plot, we can observe that our accuracies in this range were generally less-than-or-equal-to the accuracies achieved at a prior probability of 0.5 for each respective model. However, I found that the k = 18 and k = 16 models seemed to have marginally higher classification accuracies than the k = 13 model. Therefore, I can conclude that similar to the Pima dataset, I could not find a prior probability that produced a model yielding higher accuracy than the model that assumed equal prior probability.

I do not believe that these findings are generalizable to the kNN method as a whole; the effectiveness of varying the prior probability values hinges entirely on the structure of the training and testing data. In situations where the frequencies of labels in the training set is vastly different than the frequencies in the testing set, it may be better to normalize the proportions of your dataset in order to eliminate bias in predictions. This normalization could come from using inverse proportions (such as those that I used with the Pima dataset) to weight the votes of the neighbors. This could be used as a tool to fight intrinsic bias in machine learning algorithms – currently a popular topic of study in the

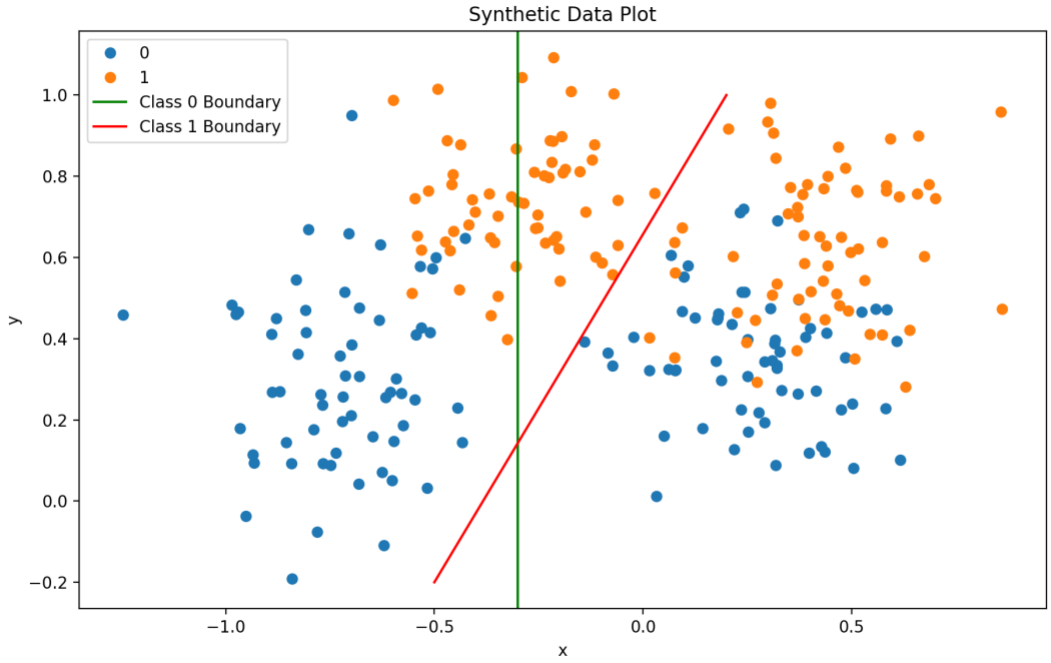
machine learning field. However, with the structure of our data, using these techniques are not advantageous in obtaining more accurate models.

Extra Credit – Part 2

2. For the synthetic dataset, if single-modal Gaussian is not the best model for the pdf, revise the model and report the performance.

As I mentioned back in Section 1, Part A, both of the classes in our synth dataset appear to have bimodal distributions. If we could model this bimodal nature, this may yield higher accuracies in our parametric learning model. Therefore, I set forth to try and define bimodal pdf's for each class so that I could implement it in the classifier.

First, I needed to determine the parameters for the bimodal distributions. I decided that we needed to create two sub-classes within the current classes; these subclasses would define our clusters of samples that we can observe in the scatterplot of this dataset. Upon visual inspection, I chose a separation boundary for both of our classes. The plot of these separation boundaries is shown below:



The boundary for class 0 was set as a vertical line at -0.3 (plotted above in green). For the boundary for class 1, I needed a diagonal linear boundary, separating the two clusters observed in the orange points. Therefore, I picked two points – (-0.5, -0.2) and (0.2, 1) – and used point-slope form to derive an equation for the line between them. This line is plotted above in red, and it seems to separate the two clusters in class 1. These boundaries would be used to split the data into clusters; for class 0 samples, if their x value was <-0.3,

they were put into cluster 0A, and if not, they were put into cluster 0B (denoted 0_0 and 0_1 respectively in the code). Then, for class 1, samples to the left of the boundary shown in red above were put into cluster 1A and samples to the right of the boundary were put into cluster 1B (denoted 1_0 and 1_1 respectively in the code).

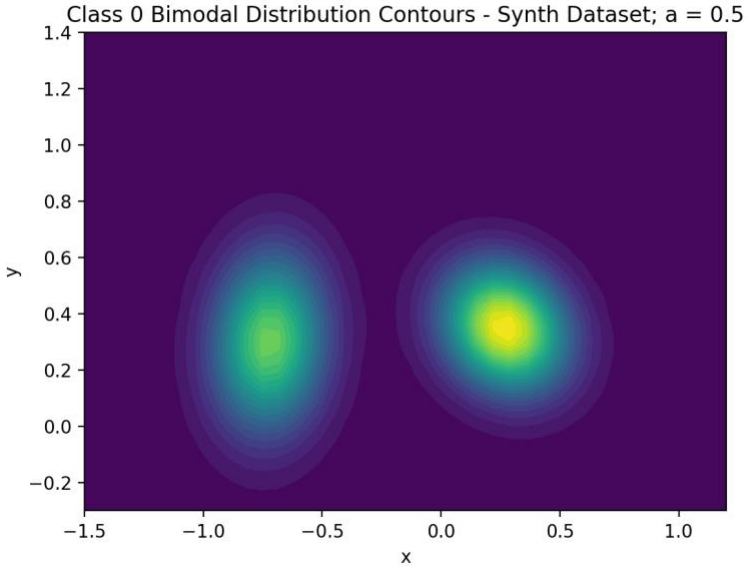
After I decided on these boundaries, we then needed to calculate parameters for Gaussian distributions centered on all four clusters. By doing this, we can define two separate Gaussian distributions for each class (corresponding to each cluster), and then we can add these distributions in order to produce a bimodal Gaussian distribution consisting of two separate Gaussian distributions centered around each cluster. This process involved parsing the data into the separate clusters (see the method in the previous paragraph) and then calculating the mean vector and covariance matrix for each separate distribution. With these parameters, we could then define separate multivariate Gaussians and add them as previously mentioned.

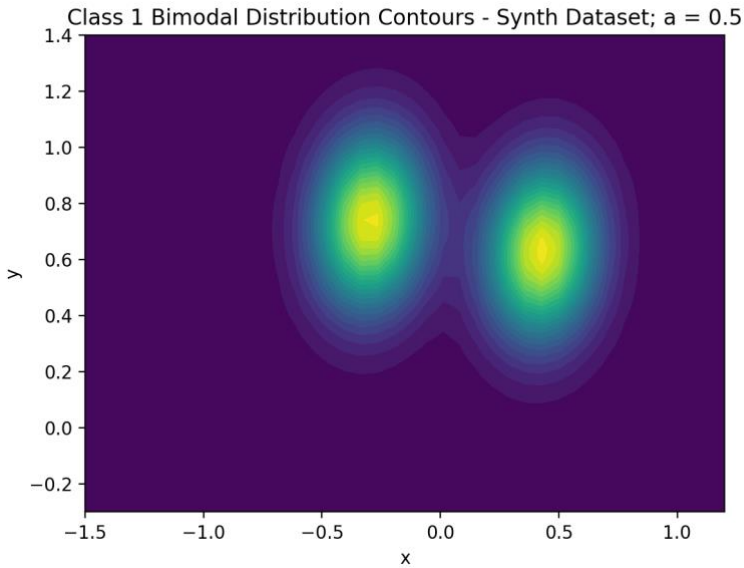
In order to preserve the pdf properties (area under the curve = 1), we also had to introduce another parameter: a. This parameter would be a value between 0-1, and it would define the weighting factor between the two separate distributions making up the bimodal distributions in the separate classes. For example, if we let N_a, N_b be the Gaussian distribution functions for clusters A and B, respectively, then our bimodal distribution would be defined as:

$$N_{bimodal} = (a)N_a + (1 - a)N_b$$

This *a* value served as a normalization factor for our separate Gaussian distributions, forcing the area under the curve for $N_{bimodal}$ to be equal to 1, consistent with the properties of pdf's. For initial trials, I set the *a* value equal to 0.5, representing an equal weighting of the distributions for separate clusters.

To verify my construction of these bimodal distributions, I created contour plots of each of the distributions for the separate datasets. My results are shown below for each class in the dataset:





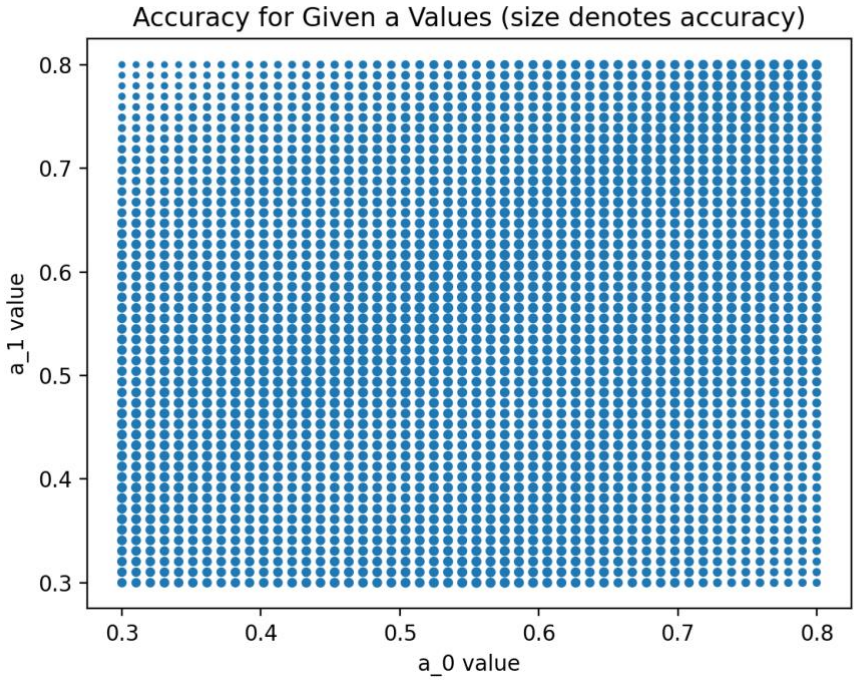
Upon examining our original scatterplot, we can see that these contour plots seem to match the distributions of the clusters in our data. Therefore, I decided to go forward with these bimodal distributions in order to see if we could achieve higher classification accuracy on this dataset. The accuracy of this method on the synth dataset is shown below (along with the other parametric algorithms for reference):

Results on Synth Dataset:

Discriminant Function	**Overall Classification Accuracy**	**Class "0" Accuracy**	**Class "1" Accuracy**	**Run time**
Euclidean (Case 1)	0.71300	0.68000	0.74600	0.21266 sec
Mahalanobis (Case 2)	0.88500	0.89600	0.84700	0.21983 sec
Quadratic (Case 3)	0.89800	0.90800	0.88800	0.37919 sec
Bimodal Dist.	0.90500	0.91000	0.90000	1.28975 sec

We can already see that the bimodal distribution has increased our classification accuracy over the other three discriminant functions. However, the runtime of the bimodal distribution function is much slower than the other three functions. This may be because, in addition to the function requiring a similar setup to the Quadratic function, the function call for each iteration is more robust.

I then decided to try and see if we could increase this accuracy by varying the value of a. For these calculations, I kept the prior probability equal because equal prior probability on this dataset proved to be very effective in the case of kNN (see Extra Credit 1). After running 2500 different combinations of a0 and a1 values, I found that I only improved my accuracy by 0.5%, from 0.905 to 0.91. This was achieved using an a0 value of 0.565 and an a1 value of 0.5245. These numbers were incredibly close to the original a0 and a1 values of 0.5 and 0.5; most likely, this increase in accuracy can be attributed to noise in the data. In fact, we can observe that varying the a value did not significantly affect the classification accuracy from this plot shown below:



The size of each dot represents the accuracy for those given a0 and a1 values (scaled to the power of 10 for that given value). We can see that even when varying the a0 and a1 values, we did not achieve more significant results than for our original value of 0.5 for each.

Therefore, the bimodal distribution did seem to provide an increase in accuracy over the other parametric learning algorithms, but even by varying the a values, we did not achieve much better results. In conclusion, there is evidence that the bimodal distribution offers a better model of the distribution of this dataset, and it can be used to achieve higher accuracies than the other parametric learning algorithms and the kNN algorithm.

Final Discussion

In this project, we saw examinations of both parametric and nonparametric learning algorithms on two different sets of data. For parametric learning, we analyzed the differences between three discriminant functions: Euclidean, Mahalanobis, and Quadratic. These three functions made different assumptions about the underlying distribution of our data, and we analyzed these differences and how each function performed on the different datasets. For the synth dataset, we saw that increasing complexity in our discriminant function yielded higher accuracy; this was not the case with the Pima dataset, where we observed the highest classification accuracy in the model using the Mahalanobis function. The more complex functions had slower run times. From my analysis, it seems as if the Mahalanobis function is very useful in parametric learning due to its high accuracy rating and fast run time. I also examined the decision boundaries of each discriminant function, where it was apparent that the Quadratic boundary and Mahalanobis boundary were very close to one another. In addition, we examined how prior probability affected these different discriminant functions in predicting class-0 labels in the synth dataset.

For nonparametric learning (for our purposes, the k-nearest neighbors algorithm) we examined k values that yielded the highest classification accuracy, and my results indicate that the larger k value is not necessarily the one that leads to higher accuracy. We saw that kNN performed very well on these datasets – better than parametric learning. It was concluded that nonparametric learning would generally be expected to yield higher classification accuracies because it avoids making any assumptions about the underlying distribution of the data. I compared and contrasted parametric and nonparametric learning and how parametric learning makes sacrifices in accuracy compared to nonparametric learning in order to achieve faster run times. Also, we looked at the decision boundary for kNN, and we saw that this boundary cannot be described by an explicit function.

My biggest takeaway from this project is that while avoiding assumptions about our data yields higher accuracies, sometimes we can make shortcuts and assumptions about our data and achieve marginally lower accuracies at much less expense. While kNN did have a higher classification accuracy than our parametric algorithms, this came with a substantial increase in run time. Moreover, this classification accuracy only improved by around 4 – 5% at best. Therefore, we could use the parametric algorithms, and therefore much less computational power, in order to achieve a comparable accuracy. When applying

machine learning to real-world problems, these are the factors that we must consider. Using a more streamlined approach that makes sacrifices in terms of accuracy may be acceptable, even preferable, in cases where our predictions do not carry much weight. Predicting what movie a user would like to watch does not carry much weight; predicting whether a patient has cancer or not does carry a lot of weight. Therefore, as data scientists, we are the ones who need to make these decisions, analyzing the context of our problem in order to understand it as much as possible. This human element of machine learning is why I want to pursue a career in this field, and I really enjoyed this project, examining all the variables that go into these analyses.