# Heart Disease Classification

Cassidy Klement Nicholas McGinty Nayef Aloraini
*Drexel University, Philadelphia, PA, USA*
*College of Computing and Informatics*
{cck47, nm3249, nma332}@drexel.edu

***Abstract*—Heart disease-related deaths account for nearly $\frac{1}{3}$ of all deaths worldwide each year. Being able to predict the presence of heart disease in patients well in advance will allow doctors to adapt their diagnoses and treatment plans, potentially saving millions of lives. We propose a system, encompassing several machine learning models, that will classify medical samples as being positive or negative for the presence heart disease. We will evaluate the performance of each model and consider the value of using an ensemble learning approach to improve the system's classification performance, as well as utilizing parallelism to improve the system's computational performance.**

***Keywords*—** machine learning, heart disease, ensemble learning, pycuda, recursive feature elimination, decision tree, logistic regression, support vector machine, k-nearest neighbors

## I. Introduction

Cardiovascular disease (CVD), also known as "heart disease," refers to a number of heart conditions that affect the heart or blood vessels [1]. According to the Centers for Disease Control and Prevention (CDC) [2] and the World Health Organization (WHO) [1], heart disease is the number one cause of death globally. The CDC has estimated that approximately 17.9 million people around the world die every year from heart disease, which accounts for 31% of all deaths. 85% of those CVD-related deaths are caused by stroke or heart attack. Strokes and heart attacks are typically caused by an obstruction in the blood vessel, stopping the blood flow to the heart or brain. These events are most commonly caused by a build-up of fatty deposits in arteries, or atherosclerosis [1].

Not only does the prevalence of heart disease have a widespread impact health-wise, but also the treatment of both symptoms and effects of heart disease has a significant financial impact. As reported by the American Heart Association (AHA), the United States' average estimated direct and indirect cost of heart disease treatment in 2014-2015 was $351.2 billion [3]. Both of these impacts could be greatly reduced by focusing on the prevention of heart disease so that care providers can avoid the treatment and aftermath of both fatal and nonfatal cardiovascular events.

There is a multitude of factors that contribute towards an increased risk of heart disease, including behavioral risk factors, existing health conditions, age, and family history. While some of these risk factors are uncontrollable, both the CDC and the WHO recognize the three key risk factors for heart disease being high blood pressure, high cholesterol, and smoking. Whether present risk factors are controllable or not, evidence has shown that preventing heart disease is even more important than treating it [4].

With the amount of data collected regarding patient health care, the problem of being able to acutely assess the risk of heart disease for a given patient well in advance, so that a physician may adapt their diagnoses and/or treatment plans, is one that is well-suited to be accomplished using machine learning techniques. We explore and compare several machine learning classification methods to consider the efficiency and performance of each method.

## II. Related Work

Machine learning can be a valuable asset in providing insight to doctors with regard to advanced predictions in heart disease diagnoses. As a result, many experiments have been conducted to compare various machine learning methods' ability to correctly classify heart disease samples, with the hope that these methods could be applied to real patient data.

One major concern broached by several authors is the issue of the high-dimensionality of heart disease data. Work done by Gárate-Escamila et al [5] poses that data relevant to heart disease prediction suffers from an abundance of irrelevant features, and in order to address this problem and avoid the likelihood of expensive memory computations and overfitting, they have experimented with using principal component analysis (PCA) to reduce the number of features consumed by the classifier.

Additional work by Li et al [6] addresses the concern of feature selection by first utilizing techniques in data pre-processing and standard feature selection algorithms: Relief, minimum redundancy maximum relevance (mRMR), LASSO, and local learning-based feature selection (LLBFS). They then propose a fast conditional mutual information (FCMIM) feature reduction method to reduce dimensionality and decrease computation time. Li et al used several classifiers in their research, including naive Bayes, support vector machines, logistic regression, decision trees, and k-nearest neighbors, as well as an artificial neural network.

Similar to the aforementioned works, we propose a system to analyze the performances of various machine learning models on heart disease classification. The models we have chosen to analyze are k-nearest neighbors with several distance functions, support vector machines, decision trees, and logistic regression. We propose the inclusion of an ensemble learning approach, as well, in the hopes that the overall performance yielded from the ensemble method will surpass the performance of any single classification model. To address the high-dimensionality concern raised by previous work, we explore using recursive feature elimination (RFE) to reduce the number of features our models will work with. To address any potential computational concerns, we investigate the idea of parallelizing the k-nearest neighbors algorithm.

## III. Dataset and Features

The datasets used in this experiment are both the processed and raw heart disease datasets provided in the University of California Irvine (UCI) Machine Learning Repository[1]. The raw dataset contains 75 features, with the target attribute, angiographic disease status (*num*), containing a value from 0 (indicating the absence of heart disease in a patient to 4. This experiment focuses on predicting the presence (1) or absence (0) of heart disease in a given sample.

$$X = \{\text{age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal}\}$$

The processed dataset contains a subset of 13 out of the 75 features available in the raw dataset, with a target value of 0 or 1, indicating

[1]https://archive.ics.uci.edu/ml/datasets/heart+disease

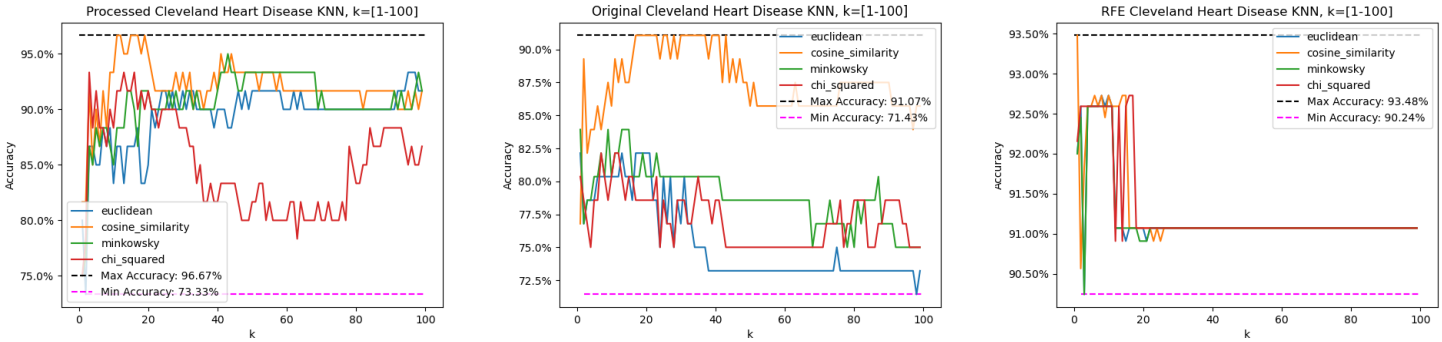



Figure 1: Accuracy for four KNN distance functions across all three datasets

the presence of heart disease or not. The processed dataset consists of 303 samples from patients, including information such as gender, age, type of chest pain experienced, resting blood pressure, blood sugar, and cholesterol levels. There is no further pre-processing needed for this experiment on the processed dataset given by UCI.

To prepare the raw dataset, several corrupted samples were stripped out, leaving a remaining 283 usable samples. The column *name*, which held dummy string values, and the column *id*, which held an ordered identification number, were both removed. The target values, ranging from 0-4, were converted to binary values, with the value 0 (no presence) remaining 0 and the values 1-4 (variable presence) translating to 1.

To better understand if the features that were pre-selected in the processed dataset were the most optimal ones, we processed a third version of the heart disease dataset using RFE to eliminate unnecessary features. We computed the calculated probability (p-values) of each feature and set a significance level of 0.0001 to reduce to the smallest possible feature space since we started with 73 features. We selected the maximum p-value and if that p-value was greater than the significance level, the corresponding feature was removed from the set of possible features. This was done until there were no remaining features with a p-value greater than the significance level. We were left with six features.

X = {cyr, ladprox, laddist, cxmain, rcaprox, rcadist}

We use all three variations of the heart disease dataset to conduct this experiment.

## IV. Methods

Heart disease prediction is a complicated task to address. In order to scrutinize the impact of different classification models, four machine learning classification algorithms were selected to solve this complex problem. Across all three datasets, raw, RFE-reduced, and processed, we implemented the k-nearest neighbors algorithm (validating with several values of $k$), the decision tree algorithm, the support vector machine algorithm, and the logistic regression algorithm.

Our approach entailed training a model for each algorithm against the same training dataset, $\mathbf{W}^{train}$, and validating against the same validation dataset, $\mathbf{W}^{test}$. We computed the following statistics to evaluate the various models' performance which we will describe in the *Evaluation* section: precision, recall, f1-score, and accuracy.

After we implemented and trained all four models, we then utilized ensemble learning to investigate if our prediction performance would improve by combining the predictions from all models to serve as input for our ensemble approach.

Finally, we investigated utilizing parallelism to allow us to increase the computational performance of running our KNN implementation across many values of $k$ and multiple distance functions.

### A. k-Nearest Neighbors

Our k-nearest neighbor implementation leverages four commonly used distance functions: Cosine Similarity, Minkowski, Euclidean and Chi-squared functions. These functions are defined as follows, where X is a feature vector denoted as $X = \{x_1, x_2, x_3 \dots x_D\}$ and Y is an observation to classify, $Y = \{y_1, y_2, y_3 \dots y_D\}$. D is the number of features in the observation.

**Cosine Similarity:**

$$dis(X,Y) = \frac{X \cdot Y}{\|\mathbf{X}\|\|\mathbf{Y}\|} \tag{1}$$

**Minkowski Distance:**

$$dis(X,Y) = (\sum_{i=1}^{D} |x_i - y_i|^p)^{\frac{1}{p}} \tag{2}$$

**Euclidean Distance (L2):**

$$dis(X,Y) = \sqrt{\frac{1}{D} \cdot \sum_{i=1}^{D} (x_i - y_i)^2} \tag{3}$$

**Chi-square Distance:**

$$dis(X,Y) = \frac{1}{2} \cdot \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{(x_i + y_i)} \tag{4}$$

We ran our KNN implementation across the three datasets of interest for this investigation, with our $k$ range being: $\{1, 2, \dots 100\}$ for each of the four distance functions. With 100 $k$ values and four distance functions, we were concerned that inference would be expensive with regards to computation time since KNN is required to retain the entire dataset in memory for each observation classification. This challenge led our team to consider leveraging a GPU where we could increase throughput for the $100(kvalues) * 4(distfuncs) = 400$ model configurations.

Our initial investigation led us to utilize PyCUDA, which is a Python API that allows one to interface with Nvidia's popular heterogeneous parallel GPU library: CUDA. CUDA, at its essence, is a language extension of C/C++ where developers can write functions that run on the GPU. A function that executes on the GPU is known as a **kernel**. Kernels can take advantage of the massive number of cores available on modern GPUs, resulting in significant speed boosts for heavy arithmetic-based operations. For this project, our team had access to an Nvidia RTX 2080 Ti

which boasts 4352 CUDA cores and 11GB of DDR6 memory. Our attempt at writing a Euclidean distance kernel can be seen below.

```
#include <math.h>
__global__
void euclidean_distance(
        float *dest, float *a,
        float *b, size_t N
        )
{
  const int threadId = threadIdx.x;
  const int index = threadIdx.x +
        blockDim.x * blockIdx.x;
  const int stride = blockDim.x * gridDim.x;
  for (int i = index; i < N; i += stride) {
      dest[threadId] +=
            pow((a[i] - b[i]), 2);
  }
  dest[threadId] = sqrt(dest[threadId];
}
```

The following Python code would be used to interface with the CUDA kernel above.

```
# ... other imports
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
# Imagine the CUDA Kernel from above was
# dropped into this "kernel" Python string
kernel = """ kernel code here"""
mod = SourceModule(kernel)
dim = 4
a = np.random.randn(dim, dim).astype(np.float32)
b = np.random.randn(dim, 1).astype(np.float32)
N = np.int32(np.product(a.shape))
thread_size = 256
block_size = int((N+thread_size-1) / thread_size)
ed = mod.get_function("euclidean_distance")
dest = np.zeros(a.shape[0])
ed(cuda.Out(dest), cuda.In(a), cuda.In(b),
N, block=(thread_size,1,1), grid=(block_size,1))
```

The one challenge we overlooked when pursuing this approach was the overhead cost of transferring the data from the host (CPU) to the device (GPU). This was a concern expressed multiple times on the proposal comments we received. However, the cost associated with this transfer was trivial, clocking in around 2ms. For larger datasets, this is a risk that would likely outweigh the benefit of making the host to device memory transfer without some additional instruction-level optimization.

Currently, our kernel only works when $a$ and $b$ are row vectors. Future work here would be to implement the kernel to compute the distance for a single observation across an entire $mxD$ feature vector, where $m$ is the number of observations and $D$ is the number of features.

A secondary approach we took to compute the 400 model configurations was leveraging a ProcessPoolExecutor. A Python ProcessPool lends itself well to the large number of jobs we had to run. A job could run over our validation dataset with a unique $k$ and distance function. Running the 400 KNN model configurations for classification in serial took approximately 57s, whereas running these configurations through a ProcessPoolExecutor took a speedy 4.9s. Our team was satisfied with this improvement in performance. However, one thing to note is that this technique does not influence the model classification to run any faster. It simply allows multiple classifications to run in parallel. Any instruction-level optimization such as parallelizing the distance function to work on slices of the feature would have to be done with a CUDA kernel. Without any instruction-level optimization, inference on larger datasets will raise compute times.

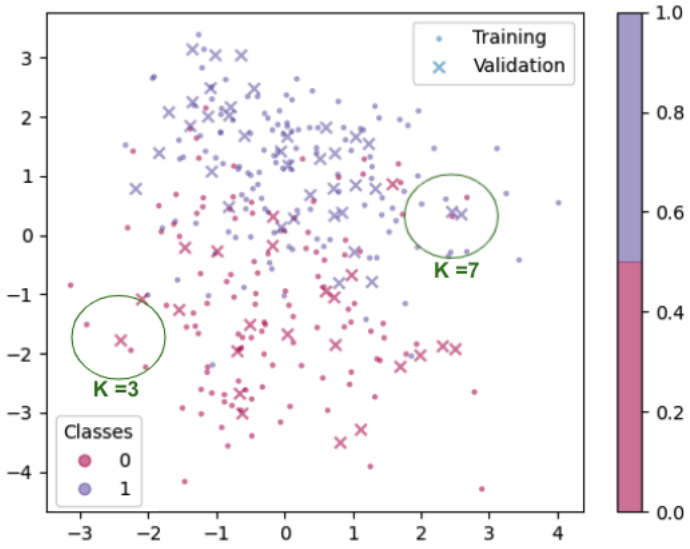



Figure 2: KNN Model Distribution

### B. Decision Tree

The decision tree algorithm requires that all features be binary. Since our datasets included both binary features and continuously-valued features, we first converted all continuous features to binary. This was achieved by computing the mean value of each feature, treated as the threshold for converting to binary, and if the sample's value for that feature was greater than or equal to the mean, the sample's value was treated as positive (1). If the sample's value for that feature was less than the mean, the sample's value was treated as negative (0).

We then recursively built the decision tree from the training data using Algorithm 1, with $\mathcal{E}$ being the weighted average entropy of a given feature.

**Algorithm 1** Decision tree algorithm

**procedure** DT($X, Y, labels, default$)
  $t = \{\}$
  **if** $X.length = 0$ **then** $return\ default$
  **if** $y_i = y_j\ for\ y_i, y_j \in Y$ **then** $return\ y_i$
  **if** $labels.length = 0$ **then** $return\ Mode(Y)$
  $\mathcal{E} = Entropy(X)$
  $\epsilon = Min(\mathcal{E}).index$
  $t[feature] = \epsilon$
  $labels.delete(\epsilon)$
  $t[true] = DT(X.true,\ Y.true,\ labels,\ default)$
  $t[false] = DT(X.false,\ Y.false,\ labels,\ default)$
  $returnt$

### C. Support Vector Machine

The support vector machine algorithm was used to distinguish whether or not a sample should be classified as having heart disease or

not by finding a hyperplane in the dimensional space of the datasets. In our case, linear separation was applied. To achieve our goal of maximizing the margin, where the margin had to be $2x$ the distance of the closest sample to the hyperplane, support vector points were laid into the margin. We used the below linear discriminant function:

$$W = X^T \times diag(Y) \times \alpha \tag{5}$$

The SVM model predicted positive or negative signs, with a positive sign indicating the prediction of the positive class (has heart disease) and a negative sign indicating the prediction of the negative class (does not have heart disease). The diagram below illustrates our SVM model generalization.

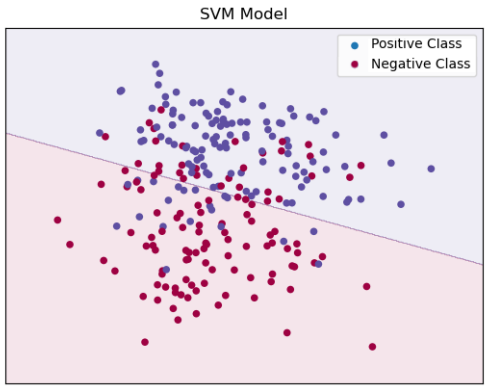



Figure 3: SVM Model Distribution

### D. Logistic Regression

The main reason in choosing logistics regression as one of the classifiers was that the target labels fall into discrete categories as having heart disease or not. Having free hyperparameters enables us to avoid overfitting in our model. Our initial investigation experiments led us to tune our hyperparameters by setting the learning rate, $\alpha$, to $10^{-3}$ and weights were initialized randomly in the range of $\pm\alpha$. Since logistic regression yields a probability, a threshold was applied as follows: if $\hat{Y} \geq 0.5$, then classify the sample as 1, otherwise classify the sample as 0.
As expected, our logistic regression model had normal convergence and achieved high scores on all three datasets. To outline the impact of selecting different threshold values on positive prediction with regards to true positive prediction, the graph below represents the Precision-Recall curve.

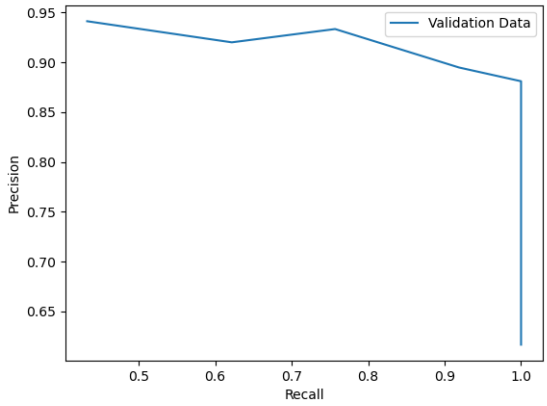



Figure 4: Precision Recall Graph

### E. Ensemble Learning

As outlined in tables 1, 2, and 3, the performance for each model varies considerably. To increase and stabilize the performance, we implemented ensemble learning. The mode, minimum, and maximum strategies have been selected as ensemble methods since most of the models predict 1's and 0's. The highest performing strategy was the majority (mode) voting strategy, using the equation:

$$P^k = mode(P^t k) \tag{6}$$

The second-highest performing strategy, on average, was the maximum voting strategy. This strategy selected the vote that had the highest value (in this case, class 1 would be selected if there was at least one vote for the positive class, otherwise class 0 would be selected), using the equation:

$$P^k = max(P^t k) \tag{7}$$

Finally, the minimum voting strategy selected the vote that had the lowest value (in this case, class 0 would be selected if there was at least one vote for the negative class, otherwise class 1 would be selected), using the equation:

$$P^k = min(P^t k) \tag{8}$$

While the mode ensemble strategy performed considerably well, one possible future improvement would be to weigh the vote of each classification model according to its training accuracy.

## V. Evaluation

To evaluate the models, the accuracy, precision, recall, and f1-score were considered. Focusing on the accuracies, the k-nearest neighbors utilizing the cosine similarities function was the best-performing classification model across all three datasets, closely followed by the mode ensemble method.

Table I: Compares the performance of each classification model on the processed dataset containing 13 features

| **Results of classification on processed heart disease dataset** | | | | |
|---|---|---|---|---|
| Approach | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
| KNN (Euc) K=95 | 90.24 | 100 | 94.87 | 93.33 |
| KNN (Cos) K=11 | 97.30 | 97.30 | 97.30 | 96.67 |
| KNN (Min) K=43 | 92.50 | 100 | 96.10 | 95 |
| KNN (Chi) K=3 | 92.31 | 97.30 | 94.74 | 93.33 |
| SVM | 92.31 | 97.30 | 94.74 | 93.33 |
| DT | 91.43 | 86.49 | 88.89 | 86.67 |
| LR | 88.10 | 100 | 93.67 | 91.67 |
| Ensemble (Mode) | 92.50 | 100 | 96.10 | 95 |
| Ensemble (Max) | 74 | 100 | 85.06 | 78.33 |
| Ensemble (Min) | 100 | 51.35 | 67.86 | 70 |

Table II: Compares the performance of each classification model on the raw dataset containing 73 features

| **Results of classification on raw heart disease dataset** | | | | |
|---|---|---|---|---|
| Approach | Precision (%) | Recall (%) | F-Measure (%) | Accuracy (%) |
| KNN (Euclidean); K=1 | 86.67 | 61.90 | 72.22 | 82.14 |
| KNN (Cosine); K=17 | 94.44 | 80.95 | 87.18 | 91.07 |
| KNN (Minkowski); K=1 | 87.50 | 66.67 | 75.68 | 83.93 |
| KNN (Chi-Squared); K=7 | 82.35 | 66.67 | 73.68 | 82.14 |
| SVM | 68 | 80.95 | 73.91 | 78.71 |
| DT | 100 | 100 | 100 | 100 |
| LR | 100 | 100 | 100 | 100 |
| Ensemble (Mode) | 100 | 80.95 | 89.47 | 92.86 |
| Ensemble (Max) | 44.68 | 100 | 61.76 | 53.57 |
| Ensemble (Min) | 100 | 23.80 | 38.46 | 71.43 |

Table III: Compares the performance of each classification model on the original dataset pruned with RFE containing 6 features

| Results of classification on RFE heart disease dataset | | | | |
|---|---|---|---|---|
| Approach | Precision (%) | Recall (%) | F-Measure (%) | Accuracy (%) |
| KNN (Euclidean); K=8 | 92.73 | 100 | 96.23 | 92.73 |
| KNN (Cosine); K=1 | 93.48 | 100 | 96.63 | 93.48 |
| KNN (Minkowski); K=4 | 92.59 | 100 | 96.15 | 92.59 |
| KNN (Chi-Squared); K=16 | 92.73 | 100 | 96.23 | 92.73 |
| SVM | 90.32 | 52.83 | 66.67 | 50 |
| DT | 92.31 | 100 | 96 | 92.31 |
| LR | 91.07 | 100 | 95.33 | 91.07 |
| Ensemble (Mode) | 92.73 | 100 | 96.23 | 92.73 |
| Ensemble (Max) | 93.18 | 100 | 96.47 | 93.18 |
| Ensemble (Min) | 90.32 | 52.83 | 66.67 | 50 |

On the raw heart disease dataset, both the decision tree and logistic regression models achieved 100% accuracy. We suspect that this is likely due to the noise introduced by dummy features induced during the UCI cleaning process. There were several features in the raw dataset that appeared to have the same dummy values for each sample, as well as "junk" ones that were not assigned any meaning. The remaining models (with the exception of the ensemble approaches, likely due to the influence of the decision tree and logistic regression models) all decreased in performance. During our pre-processing of the raw data, we intended to retain as many of the original features as possible, removing just the identification number column and the name column, which contained a dummy string value. Noting the performance of the decision tree and logistic regression models, we will be looking to complete a more thorough cleaning process on the raw dataset in the future.

## VI. Conclusion

In summary, we explored several machine learning methods, including k-nearest neighbors, decision tree, logistic regression, and support vector machines, to evaluate how different models performed on a dataset containing samples to be classified as having heart disease or not. We consider the classification predictions in several ensemble learning models, utilizing mode, maximum, and minimum strategies. We proposed two solutions to two separate concerns present in heart disease classification. To address the problem of high-dimensionality, we employ recursive feature elimination to reduce the feature space of the heart disease dataset. We also implement k-nearest-neighbors to run across four distance functions and values of $k = \{1..100\}$ in parallel to reduce computation time, which would be a significant concern when working with a very large dataset. Our evaluation revealed that k-nearest neighbors utilizing the cosine similarity method was the most effective in classifying heart disease samples, with mode ensemble learning proving to be just as effective. We also identified an additional area of concern when working with heart disease data. Based on the models' performances on the raw heart disease data, more sophisticated qualitative data analysis may be a prerequisite to working with raw medical data.

## VII. Future Work

Having a variety of algorithms encourages us to surpass our current outcomes by incorporating larger datasets into our next experiments to evaluate our systems. Furthermore, we would like to include artificial neural networks along with Naive Bayes in our next experiments. Artificial neural networks do not impose any restrictions with regards to the inputs, and naive Bayes would be suitable for lack of training data. There are additional optimization techniques our team can explore with respect to heterogeneous programming. One of those being to preserve the data feature set on the device across each KNN model configuration change. The cost of transferring the data was trivial for a single run. However, with the significant number of configurations our KNN model took on, the host-to-device memory allocation and transfer could become quite costly for 400+ calls. Making the host-to-device transfer once and processing that same data across varying KNN *k* values and distance functions is a simple technique that could improve throughput.

## References


[1] "Cardiovascular diseases (cvds)," https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[2] "About heart disease," https://www.cdc.gov/heartdisease/about.htm.

[3] "Heart disease and stroke statistics-2019 update: A report from the american heart association," *Circulation*, 2019.

[4] C. SG, "A new paradigm in the treatment of the cardiovascular disease continuum: focus on prevention," *Hippokratia*, 2011.

[5] A. K. Gárate-Escamila, A. Hajjam El Hassani, and E. Andrès, "Classification models for heart disease prediction using feature selection and pca," *Informatics in Medicine Unlocked*, vol. 19.

[6] J. P. Li, A. U. Haq, S. U. Din, J. Khan, A. Khan, and A. Saboor, "Heart disease identification method using machine learning classification in e-healthcare," *IEEE Access*, vol. 8, 2020.