Brenden Hein

CSE 847

Dr. Zhou

Diabetes Prediction Final Report

The code for the project can be found at: https://github.com/Brenhein/Project_847

I. Introduction

As the amount of sugar in food continues to increase, diabetes has been steadily on the rise. This issue can become a major problem for a significant portion of those with the disease, thus making early detection and prediction important. If one were to know they may develop diabetes, that person is able to make the changes necessary to help hinder the disease before it's too late. The Diabetes Prediction project aims to generate insights for those providing a subset of their medical record on whether or not they are susceptible to developing diabetes.

This project starts with a dataset, where every sample has 8 features and 1 ground truth. Below, all 9 of these attributes are described in better detail.

- *Pregnancies – The number of pregnancies a person has* ***(INT)***
- *Glucose – The glucose level in the blood for a person* ***(INT)***
- *Blood Pressure – The diastolic blood pressure of a person* ***(INT)***
- *Skin Thickness – The thickness of a person's skin* ***(INT)***
- *Insulin – The insulin level of person* ***(INT)***
- *Diabetes Pedigree – Likelihood a patient will develop diabetes based on family history* ***(FLOAT)***
- *BMI – The Body Mass Index of a person* ***(FLOAT)***
- *Age – The age of a person* ***(INT)***
- *Outcome (ground truth) – Whether or not a patient will develop diabetes* ***(BOOL)***

The first 8 features will be used to predict whether a patient will develop diabetes or not, and the last feature is the ground truth, i.e., whether they actually develop diabetes or not. These will be used to test a multitude of machine learning models in attempt to figure out which model works the best. For this project, 4 models will be used: K-Nearest Neighbors, Logistic Regression, Gaussian Parzen Windows, and Nonlinear SVM. The methods for the setup of the program and the data-preprocessing are described below, followed by the methods and results for each of the four models.

II. Data Preprocessing

To initialize the project, the data set (found in a single csv file) was parsed and brought into program memory; the logic for this takes place in the file, data_preprocessing.py. The get_data function takes in 1 parameter, fname (the name of the file), and returns the training samples, testing samples, training ground truth, and testing ground truth.

Next, as the dataset contains no distinction on which samples are for training and which samples are for testing, cross-validation was used to minimize any biases that may come with how the training and testing data is split. As the whole dataset is only 768 samples, it was decided that, after some investigative testing, 6 folds was the best way to evenly split the data. Thus, for each fold being tested, 128 testing samples will be evaluated with a model that is trained with the remaining 640 samples. As in the intermediate report K-NN was not yielding the best results, this will hopefully help slightly improve the classification accuracy, and it will be used instead of a simple split for all models in this project.

Also, there were quite a few zeros in the dataset. As most of these zeros are assumed to be missing values, this can throw off the dataset. Therefore, a simple algorithm of filling in the missing values with the average of the corresponding column the zero is in was used. While there are more advanced algorithms, such as the EM algorithm, which makes assumptions on the underlying distribution of the data, that can be used, this was a simple way to lower the error rate as these outliers can throw of predictions. However, zeros were left in for the pregnancy feature, as it is completely reasonable that people have had 0 pregnancies, while for features such as blood pressure, it doesn't make sense for them to be zero.

Lastly, the data matrix was normalized using the normalize function provided by the sklearn library. By normalizing the dataset, the hope is to improve the classification accuracy by fixing the varying ranges of each feature into a set range for all features. For example, the range for age and blood pressure are drastically different, but through normalization, their ranges will be similar, both being centered around zero and divided by their standard deviations. Finally, an outlier removal preprocessing technique using the inner quartile range was attempted, as well, but the technique was scrapped as it noticeably decreased the classification accuracy.

III. K-NN

The first and simplest algorithm implemented and tested was K-NN. The way in which K-NN works is by selecting a K which represents the number of neighbors (training points) that will be used to classify a single test point. Thus, for every test point, K-NN will find the K nearest training points of that test point based on the Euclidean distance, and the most prevalent class of those K neighbors will be the class that the single test point will be assigned to. An example of K-NN can be seen below in [1].

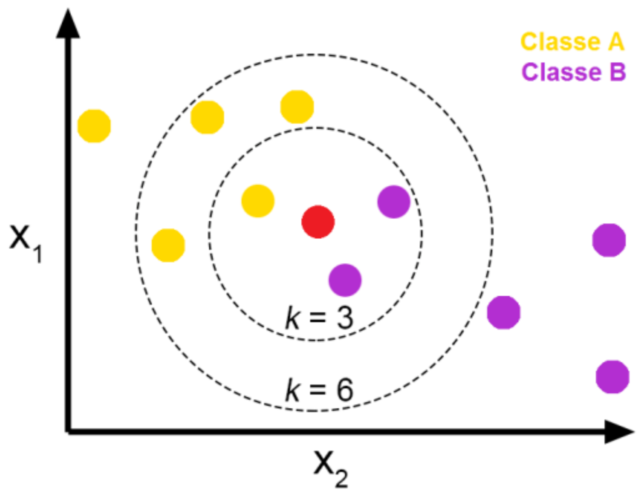

Fig [1]. The error rate for K-NN on the diabetes dataset using a K of 1, 2, …, 49, 50

In this diagram, if the three nearest training neighbors are selected, two of them will belong to class B and one will belong to class A; thus, the test point is assigned to class B. However, in the outer ring, if the six nearest neighbors are selected, two of them will belong to class B and four will belong to class A; now, the test point is assigned to class A, as the majority of its neighbors are from class A.

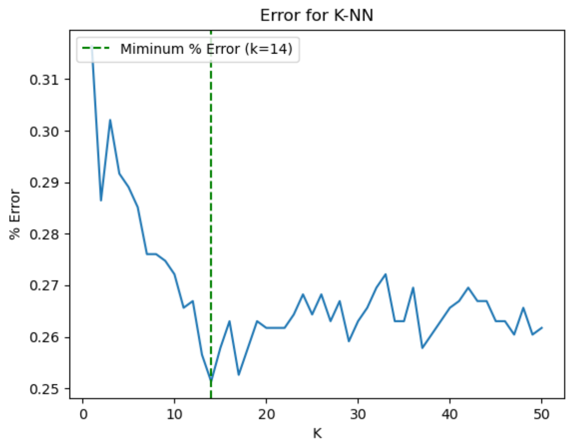

Fig [2]. The error rate for K-NN on the diabetes dataset using a K of 1, 2, …, 49, 50

Table [1]. The confusion matrix for a K of 14, as it yielded the minimum error rate

	Diabetic	Not Diabetic
Predicted Diabetic	447	140
Predicted Not Diabetic	53	128

In this experiment, K values ranging from 1 to 50, increased by increments of 1, were used to test this model. This was changed from a range of 1 to 30 in the intermediate report in order to see if the optimal K was larger than initially expected. Through this, it was discovered that given a K of 14, the error rate was the lowest at about 25.1%, as seen in fig [1]. In table [1], the confusion matrix can be found for K-NN, using a K of 14, on the dataset. Interestingly, it appears that using 6-fold cross validation, along with filling in the missing values with the column's average and normalization, seemed to barely change the results. However, both methods lead to more accurate, not necessarily better, results, and the difference was negligible, so these results are acceptable. However, the results are still in a poorer territory, and without further data-preprocessing, will remain as such. K-NN, while it can serve as a good base classifier, is a rather simple machine learning algorithm, returning worse results than those more advanced models. Testing K-NN still provided some good insight on the dataset. Next, this paper turns its attention to logistic regression.

IV. Logistic Regression

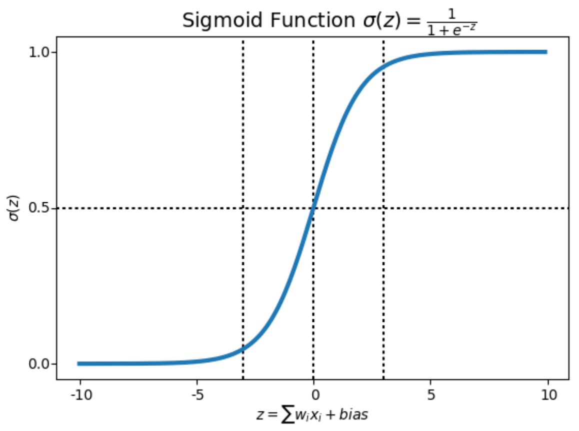

Fig [3]. A graph of the shape of the sigmoid function, adapted from [2]

The next algorithm analyzed was logistic regression. Logistic regression is another classification algorithm. The model uses the sigmoid function, which can be seen above in fig [3]. Here, the values range from 0 to 1. If the value of z, which is the weighted sum of features added together with a bias, approaches infinity, the result of the sigmoid function approaches one; this confidently assigns the sample to one of the two classes. If the value of z approaches negative infinity, the result of the sigmoid function approaches 0; this confidently assigns the sample to the other class. This is the discriminant function used for classification.

The logistic regression function has been implemented for this project, and it uses non-stochastic first-order gradient descent. As the total number of data points was small enough with a data matrix of size 768 by 8, it made sense to load all of it into memory, which is why a non-stochastic model was used. During the runtime of the model, the gradient descent algorithm weights are iteratively updated through the negative gradient of the error function for logistic regression, multiplied by its step size. After intense testing, it was discovered that an optimal step size was 1.5.

During testing, it was discovered that the optimal value of the hyperparameter, n, was around 1.5. This was chosen as the error rate of logistic regression tended to converge towards its minimum error rate here. Anything above this and anything below this both yielded worse results, with the error rate increasing the farther away the chosen n is from the optimal n. With this value for the hyperparameter found, logistic regression was then evaluated.

Table [2]. The confusion matrix for logistic regression, using a step size of 1.5

	Diabetic	Not Diabetic
Predicted Diabetic	421	146
Predicted Not Diabetic	79	122

The results of logistic regression were very disappointing. Through rigorous testing and preprocessing, the optimal error was stuck at **29.3%**. The confusion matrix for that result can be seen above in table [2]. It appears that logistic regression is not a good classifier for this dataset, since even a simple model like K-NN outperformed it. The issue is most likely due to the struggle that logistic regression has when finding a global minimum: local optima. It appears that, while logistic regression found a decent optimum, it is most likely not the absolute minimum, which is why the error rate may be so high. However, it is not the only model being tested, so we will focus on Gaussian Parzen windows next.

V. Gaussian Parzen Windows

A third model looked at was a non-parametric density estimation model called Parzen Windows. Similar to K-NN, it uses all training points to classify a test point. But while K-NN looks at all the nearest training point neighbors, Parzen windows create Euclidean structure, where depending on if the points lie inside or outside of the structure, will be classified as different classes. This can potentially work great for diabetes classification, as the underlying distribution of the data is unknown. However, a kernel function is still required to represent the data

For this project, it was decided the best window to select was a Gaussian Kernel Function. Many kernel functions are binary. For example, if the kernel is a hypercube, the testing point either falls inside or outside the cube; there is no in-between. While this has the potential to work well if the data's distribution conforms well to the hypercubes, it is not guaranteed. This is where the Gaussian kernel function comes into play. While this window obviously works best when the underlying distribution is Gaussian, its major benefit in this case comes from the fact that the results are not binary. Instead of a point being inside or outside a kernel, its distance to the center of the kernel is used. Whatever kernel the testing point has the maximum probability density for will be the class it is assigned to [9].

Through testing the normalized dataset, it was determined that a window size fell somewhere between 0.02 and 0.01, so a range of .02 to .01 was tested, decrementing by .001 for each test. What was discovered that the optimal window size seemed to be around 0.0137, with an error rate of **26.2%** The confusion matrix for this window size using 6-fold cross validation can be seen below in table [3].

Table [3]. The confusion matrix for Gaussian Parzen Windows, using a window size of 0.0137

	Diabetic	Not Diabetic
Predicted Diabetic	464	165
Predicted Not Diabetic	36	103

In [5], the error rates for all window sizes can be seen. Again, similar to K-NN, the optimal window size is represented by a vertically dashed green line. The error rate seems to reduce until about .014, before sharply increasing again. With this, there is fair certainty that the optimal window size for a Gaussian Parzen Window is around 0.0137, which is then used to evaluate the classifier.

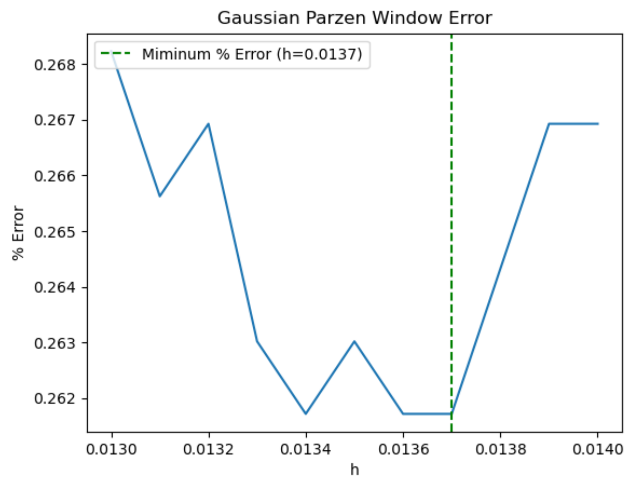

Fig [4]. The error rates for a Gaussian Parzen Window for window sizes of .02 to .01, decremented by .001 for each test

Once again, the error rate is not the best. Gaussian Parzen windows seem to perform similarly to K-NN, and thus offer no real benefit outside of that. So, while Parzen windows performed better than logistic regression, it seems to still be a subpar classifier with mediocre results, and when deciding between using it or K-NN, there aren't any noticeable classification benefits for either.

VI. Nonlinear SVM with a Gaussian Radial Basis Function

Support vector machines are another form of binary classification, which uses these data points called "support vectors" to determine a margin. This margin will be the maximum margin that separates the two classes. If classes are linearly separable, this works great. An example of a decision boundary with the SVM margin held up by its support vectors can be seen below in [6].

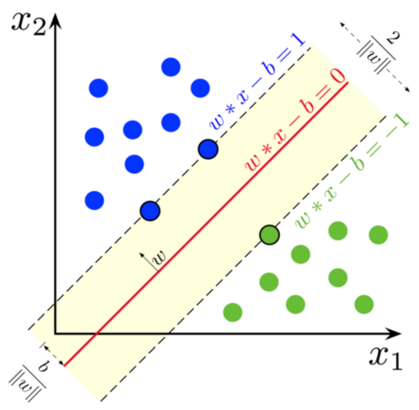

Fig [5]. An example of hard-margin SVM with linearly separable data, adapted from [1]

However, what if the data is not linearly separable? SVM still works great for classification on nonlinearly separable data, but it must be done through use of a kernel function. These similarity functions, which replace the dot product in normal SVM, transform the data set into higher dimensions, where the new features may actually be linearly separable. A situation where kernel SVM works well is demonstrated in [7], where the feature set is transformed from 2 to 3 dimensions, with the new feature seeming to be some inverse of $x_1^2 + x_2^2$. By adding this third feature, the new feature space becomes linearly separable.

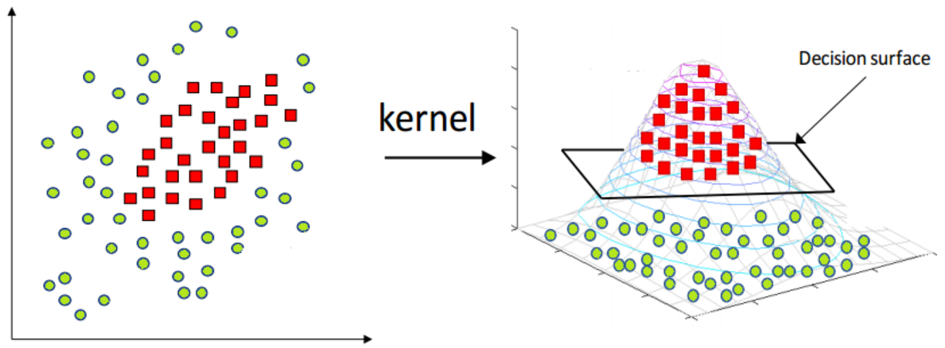

Fig [6]. An example of hard-margin SVM with non-linearly separable data, adapted from [1]

As there is no guarantee that the diabetes dataset is linearly separable, leaving 2 options left to classify the data: soft margin SVM and nonlinear SVM. For this project, nonlinear SVM was chosen, in hopes of providing better results than K-NN and logistic regression. The kernel function that will be used is a Gaussian Radial Basis Function. The reason this kernel function was chosen was two-fold. Firstly, this function works well, even if the underlying distribution of the data is unknown, as in this case. The other reason is that it's Gaussian, meaning that points are penalized differently based on their location. The way this model works is by using the function seen below in [8].

$$K(x_1, x_2) = e^{\frac{||x_1 - x_2||^2}{2\sigma^2}}$$

Fig [7]. The equation used for the Gaussian Radial Basis Function, where x1 and x2 are sample vectors, adapted from [7]

This function is based around the similarity between two sample vectors, x1 and x2. The distance being calculated in the numerator, as it is taking a vector norm, is how close or far away the two points are from each other. In denominator, there is one unknown, sigma. This sigma is a hyperparameter and represents the variance of the model. Through some testing and finetuning, a good value for sigma can be discovered, which will then be used in this project. The result of this function is a value in the range [0, 1]. The closer the value of this function is to 0, the farther a way the points are, and thus the more dissimilar they are. Conversely, the closer the function's return is to 1, the more similar the two vectors are. If the function returns 1, the points are in the exact same location [3]. This mathematical foundation will transform the dataset using the GRBF (Gaussian Radial Basis Function) to a new dimensional space, where SVM can then be applied, which will be implemented below.

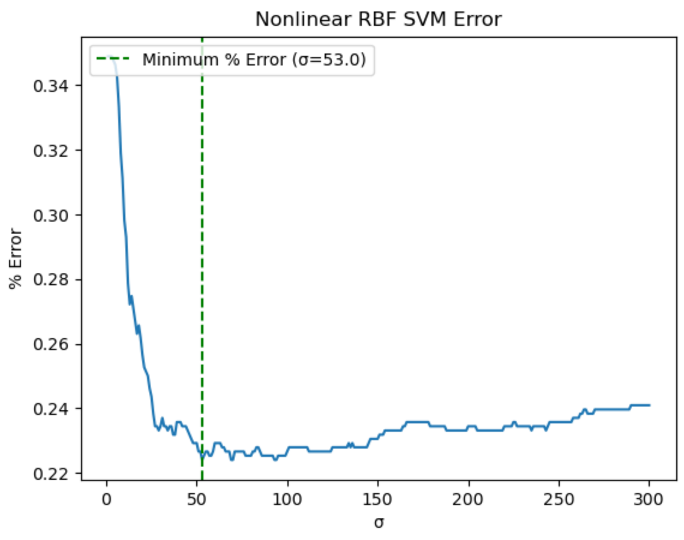

Fig [8]. The average error rates of nonlinear SVM using 6-fold cross validation on the dataset from σ ranging from 1 to 300 with a step size of 1

Above in [9], the average error rate between the 6 folds is plotted. Again, through some trial and error, the best error rates appeared to be in the low hundreds. And add to the fact that nonlinear SVM was very quick to run, many hyperparameters could be tested at once. Thus, 300 different values were tested from 1 to 300. The optimal value, represented by the green, dashed, vertical line, is the optimal parameter that led to the lowest error.

Table [4]. The confusion matrix for nonlinear SWM, using a σ of 53

	Diabetic	Not Diabetic
Predicted Diabetic	454	126
Predicted Not Diabetic	46	142

Above in table [4], the confusion matrix for a σ of 53 can be seen. Like with the other 3 models, nonlinear support vector machines with the RBF kernel seem to do a decent job of classification. But what sets nonlinear SVM apart is that it managed to achieve the lowest error rate of 22.4%. However, even though it is better, it is less so than initially hoped for. Interestingly, this model seemed to predict more correctly that non-diabetic patients were actually non-diabetic, something the other 3 models weren't as good at, potentially due to a more significant variance in the diabetic class than the non-diabetic class. However, this was not tested in the scope of this project.

VII. Conclusion

The results of these four models, while still decent, are worse than initially anticipated. Nonlinear SVM performed the best of the models, but by a smaller margin than hoped for. K-NN and Gaussian Parzen windows were next as two best models, both yielding very similar results. As they are both non-parametric density estimation methods with a somewhat similar structure, these results aren't too surprising. Lastly, logistic regression performed the worst, by far. While its error rate was not terrible at around 29%, it is still not good and would not be considered a viable classifier. As discussed previously, the theory behind this result is that gradient descent is getting stuck in local optima instead of the global minimum.

When it comes to selecting the best model, nonlinear SVM with an RBF kernel seems to be the ideal option. Other kernel functions were tested with the model as well, such as a polynomial and sigmoid kernel, but they yielded worse results than that of the RBF kernel function. Despite nonlinear SVM appearing to be the best, it is not guaranteed. Many values of the hyperparameter for each model was tested to finetune them, but even with high certainty that they are the best, it is not guaranteed. Also, there are still a multitude of data preprocessing techniques that can be applied, too, especially if more research was done into the initial structure and measures of the dataset. Finally, as it seems that these machine learning techniques tend to cap at about 80% classification accuracy, deep learning techniques, such as an artificial neural network, could be looked at as a good alternative in a future version of this project, as they tend to outperform most other models.

Overall, while the results are worse than expected, after looking into how other models performed written by others on Kaggle in [4], the results obtained in this project are very similar to those other results across many different types of machine learning algorithms. Thus, they dataset is most likely not so easy to work with and separate than initially thought. Thus, this project is considered a fair success, and even though there is plenty of room for more testing, that is something that must be delved into in a future version of this project.

Sources

[1] Alzubaidi, L., Zhang, J., Humaidi, A.J. *et al.* Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data* **8,** 53 (2021). https://doi.org/10.1186/s40537-021-00444-8. [Accessed: 15-Mar-2022].

[2] A. Pant, "Introduction to Logistic Regression," *Introduction to Logistic Regression*, 22-Jan-2019. [Online]. Available: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148. [Accessed: 21-Mar-2022].

[3] S. Sreenivasa, "Radial basis function (RBF) kernel: The go-to kernel," *Medium*, 12-Oct-2020. [Online]. Available: https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a. [Accessed: 17-Apr-2022].

[4] M. Akturk, *Diabetes Dataset*, 2020. [Dataset]. Available: https://www.kaggle.com/datasets/mathchi/diabetes-data-set. [Accessed: February 14, 2022].

[5] A. Chakure, "K-Nearest Neighbors (KNN) algorithm," *Medium*, 06-Jun-2021. [Online]. Available: https://medium.datadriveninvestor.com/k-nearest-neighbors-knn-algorithm-bd375d14eec7. [Accessed: 20-Apr-2022].

[6] S. Raschka, "Kernel density estimation via the parzen-rosenblatt window method," *Dr. Sebastian Raschka*, 19-Jun-2014. [Online]. Available: https://sebastianraschka.com/Articles/2014_kernel_density_est.html. [Accessed: 09-Apr-2022].

[7] "1.4. Support Vector Machines," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed: 23-Apr-2022].

[8] "Support-Vector Machine," *Wikipedia*, 25-Mar-2022. [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine. [Accessed: 22-Apr-2022].