Optimizing k-Nearest Neighbor Classification Efficiency and Efficacy

Shreraj Sainani, Olivia Wu

Thomas Jefferson High School for Science and Technology

Dr. Yilmaz Machine Learning 1 Period 6

January 2026

Abstract

The k-Nearest Neighbors (KNN) algorithm is a commonly used supervised classification method, but it comes with a high computational cost that scales with dataset size and has reduced efficiency in high-dimensional settings. Computing with pure theoretical KNN for a single query is exhaustive, with an overall time complexity of *~O(n)*. Additionally, standard distance metrics such as Euclidean distance treat all attributes equally, allowing weaker features to distract similarity calculations. In this work, we identify two approaches to improving KNN: reducing the computation time of finding neighbors and improving distance calculations by favoring informative attributes. We propose a modified KNN framework that is tested on the well-known Diabetes Dataset as well as the well-known MAGIC Gamma Telescope Dataset. We show that algorithm speed and F1 scores are both improved with our framework.

Introduction

When using the KNN algorithm, finding the k-nearest neighbors of a desired testing instance requires the distance from that testing instance out to all points in the training set to be determined. As those distances are being determined, the k-nearest neighbors must be kept track of in a sorted list. If the number of instances in a training set is *n* and the dimension of the dataset is *d*, determining the distance for a single point is *O(d)* and if a point is close enough, then inserting it into the sorted holding the nearest neighbors and appropriately shifting elements is *O(k)*. Often though, points are not close enough in the first place, and very few of these insertions actually happen throughout the algorithm's perusal of the training space. Thus the operation for a single point is *O(d)* in the best case, *~O(d)* in practice, and *O(kd)* in the worst case. Performing this operation for all points results in a time complexity of *O(nd)* in the best case, *~O(nd)* in practice, and *O(nkd)* in the worst case. This makes the KNN algorithm quite slow for larger training datasets with higher dimensionality. If some way could be found to restrict the amount of points that need to be considered in the training dataset (decreasing the value of *n*), or improving the distance calculation, the calculations could be sped up significantly.

Dimensionality reduction would also help with this as well. The KNN algorithm is highly vulnerable to the curse of dimensionality with most distance metrics. While not only increasing the computational resources required to find the k-nearest neighbors, the higher dimension also weakens a KNN model's ability to generalize. While metrics like the cosine similarity aim to build on some of the weaknesses of pure distance metrics like Euclidean or Manhattan distances, cosine similarity causes a KNN model to become blind to distances anyway. For higher dimension datasets with the KNN algorithm, points that are close in a few important attributes but far away in many other unrelated attributes may end up being overtaken by unrelated points that are not super far away in more attributes, but still unrelated nonetheless. If more important and/or impactful attributes could be prioritized, a situation like this could be avoided since the important attributes for which the points were closer would be prioritized more, even though they were far away in terms of the unrelated attributes. KNN thus has fertile grounds for improvement in mitigating its vulnerability to the curse of dimensionality.

We have then identified two specific areas of improvement for the KNN algorithm: 1) improving the time complexity of a k-nearest neighbor determination in any way, 2) mitigating the curse of dimensionality by prioritizing more important attributes in distance calculations, and by reducing dimension in general, but beyond basic dimensionality reduction in data-preprocessing.

The implementation of our solutions to these areas of improvement are discussed in **Methods**, and the experiments ran to evaluate these improvements are described in **Experimental Design**. In general, since we are using a KNN model, the input will be supervised training data with quantitative continuous features and a categorical class along with a desired testing instance, and the output will be a classification of that testing instance.

Related Work

Past work on KNN can be grouped into approaches that exploit feature-weighted methods, optimization approaches, and information gain or contribution weighting schemes. Huang et al. (2018) propose a KNN algorithm that assigns weights based on both feature importance and class contribution. While effective in giving informational attributes more influence, there are many additional parameters that are considered in their work. Karabulut et al. (2019) instead make their own similarity measure, creating a weighted similarity metric that replaces the standard Euclidean distance. Their weights are static, however, and may be underfitted to local data.

Optimization has been targeted through the use of genetic algorithms. Kalaivani and Shunmuganathan (2014) optimize feature weights for KNN with sentiment classification, and they tested their methods on movie and book reviews data, achieving F scores of over 87%. However, genetic algorithms are computationally expensive and may be limited by dataset size. In contrast, Ostrovsky and Rabani (1998) address the curse of dimensionality's effect on computational performance. Their work on approximate nearest neighbor search in high-dimensional spaces inspired our idea of simplifying our dataset into sectors discussed in our methods before running the KNN algorithm.

Additionally, researchers such as Vahedifar et al. have been investigating contribution-based weighting of features, which represent the current state-of-the-art research in this field. In 2025, the team introduced a Shapley-based modified KNN that uses mutual information to quantify each data point's contribution to classification. They utilize cooperative game theory and information theory, strengthening the interpretation of their performance. However, Shapley value estimation is expensive for large datasets.

These papers highlight two approaches we aim to take with improving KNN: reducing the computational complexity of the algorithm and improving the distance metric with feature weighting.

Methods

Sectors Approach: Improving Time Complexity

We aimed to improve the time complexity of finding the k-nearest neighbors when classifying a point using the KNN algorithm by decreasing the amount of points that need to be considered. Our first idea was to split up the *d*-dimensional space of the training data into different sectors and classify instances based only on their constituency in their sector and not the entire training data as a whole. These sectors were to be *d*-cubes, or in other words, cubes of dimension *d* (1-cube: line, 2-cube: square, 3-cube: cube, 4-cube: hypercube/tesseract, etc.). It didn't make sense to make them balls or spheres or any other sort of complex shape, because just partitioning up the space into these *d*-cubes would be the simplest and easiest. Determining which sector a point would be in would also be incredibly easy, since being d-cubes, each of these sectors would just be determined by *d* inequalities (some number < x_1 < some number, …, some number < x_d < some number). The partitioning would just be equal-width (as opposed to equal frequency) to preserve the notion of distance in KNN–performing equal frequency-binning would split up the space by frequency, destroying information related to distance. Thus, the partitioning was to be equal-width splitting along the range of data of an attribute. Along each feature in, the lowest split and highest split were not bounded at the maximum and minimum in the range of the data of the feature, instead extending off into infinity in the appropriate direction to include points that could fall outside of the range of the training data in classification. This itself does not make the model susceptible to outliers, since the points would still be close together, but the bounds would be different. The model may be susceptible to outliers since outliers in data could widen the splits significantly. That however does not change the fact that points on the boundary of a *d*-cube are classified by only points in their sector, even though they might be much closer to points in the neighboring sector. For these reasons, it's important to collect testing metrics rather than just time in assessing the potency of this approach.

However, this approach presents one significant issue. Considering that we perform *s* splits in each dimension, the training space would be split up into s^d *d*-cubes. Even just having *s* as 2 (a halfway split), this creates exponentially increasing amounts of *d*-cubes for dataset with even relatively low dimensionality. This method was revised to be as said before but only split along a certain number of dimensions, denoted d_{splits}. Thus, in a dataset with *d* dimensions and *s* splits along each dimension, there are *s* raised to the power of d_{splits} splits. This results in the sectors of the training space being determined by d_{splits} inequalities. The features that are chosen to be split were the ones with the highest standard deviations, since those would result in more spread-out data and thus tending towards more equal frequencies, despite being equal-width splitting (equal-width splitting was still imperative to preserve distance in the KNN algorithm as mentioned earlier). So, the d_{splits} features out of the total *d* features with the highest standard deviations were split along *s* times.

The following figures illustrate what this splitting mechanism looks like with different combinations of d and d_{splits} with $d \leq 3$ in the d-dimensional training space and $s = 2$ and $s = 3$ for illustration purposes.

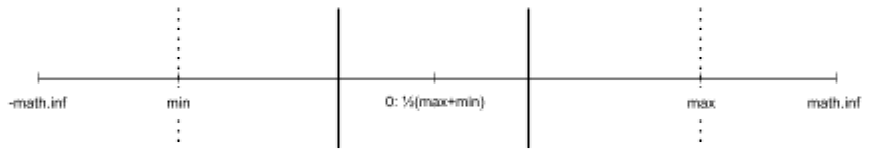


Figure 1: A sectors split with $s = 3$ and $d = d_{splits} = 1$. While illustrating how a dataset with $d = 1$ is split with $s = 3$, it also illustrates how an individual feature in a dataset is split no matter d or d_{splits} with $s = 3$. The large, thicker lines illustrate the boundaries of the d-cubes. The amount of d-cubes is s^(d_{splits}) = 3. For illustration purposes, the minimum value of the feature, min, is chosen to be -max where max, the maximum value of the feature, is > 0. This is not always the case, but it is convenient for this illustration. As described in above, the lower and upper bounds of the lowermost and uppermost d-cubes are abolished in favor of having them extend off to the appropriate infinity. The dotted lines show where those boundaries would be otherwise if they were not abolished.

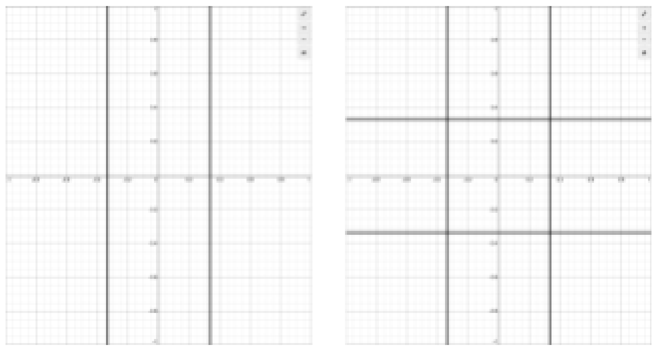


Figure 2: Sectors split with $s = 3$; $d = 2$; and $d_{splits} = 1$ (left) and $d_{splits} = 2$ (right), where the minimum and maximum of both features are arbitrary. The selection of features that are split when $d_{splits} < d$ is arbitrary. The amount of d-cubes is s^(d_{splits}) = 3 and 9 on the left and the right respectively.

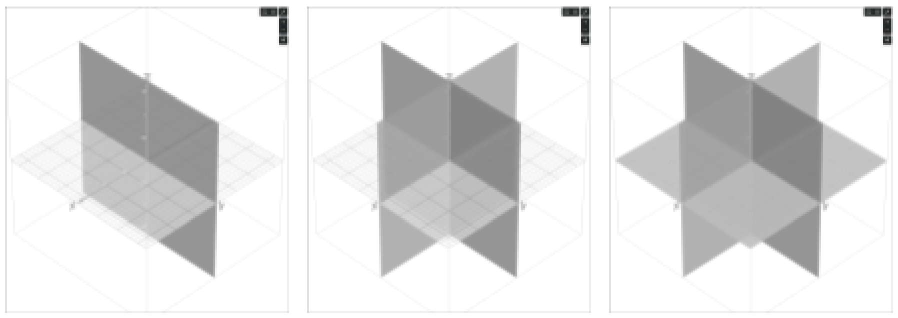


Figure 3: Sectors split with $s = 2$; $d = 3$; and $d_{splits} = 1$, 2, and 3 in left to right order respectively. Here, the minimums and maximums of the features are arbitrary. The selection of features that are split when $d_{splits} < d$ is arbitrary. The amount of d-cubes is s^(d_{splits}) = 2, 4, and 8 going from left to right respectively.

From the above figures, it becomes easy to see how the same geometric patterns apply and continue when d or $d_{splits} > 3$, and s any number. For instance, a training space with $d = 10$, $d_{splits} = 4$, and $s = 3$ would have $s\^(d_{splits}) = 81$ d-cubes in this case.

This Sectors Approach would make the k-nearest neighbor determination of a point in time complexity drop from *~O(nd)* to $\sim O(sd_{splits} + (n/s\^(d_{splits}))d)$. The first term $s*d_{splits}$ comes from classifying a testing instance into its respective cube–for each feature that's split, check all the bounds for that feature and see which one it falls into, finding the right cube. The second obviously comes from the distance calculations (and keeping track of the k-nearest neighbors), where assuming points are somewhat uniformly distributed, we only have to look through the amount of points in the testing instances' *d*-cube. If we split into enough sectors s along enough dimensions *d*, this theoretically should significantly decrease the computational resources required for finding the k-nearest neighbors. The Sectors Approach however incurs an overhead cost of *O(nsd)* for grouping training data into *d*-cubes. This only occurs once through when using the model, and is insignificant when an already initialized model is in use. We aim to see how significant this overhead approach is in practice in testing.

This idea was born from the notion that it seemed silly to calculate distances to points that are very far away from an instance, but to even know that they are super far away would require a distance calculation in itself. By splitting up the d-dimensional space into sectors and partitioning points based on their residency in a sector, it can be known from the start that some points are so far away that they're not even worth considering. If there aren't enough neighbors in the desired instance's sector for classification (points in testing instance's *d*-cube < k), then the entire sector is majority-voted for classification. Finally, if there are no points in the sector (this is rare with realistic *n*, *d*, *s*, and d_{splits}), then class is assigned randomly. In short this is just slicing up the space into sectors and only considering points that are known to be relatively nearer to an instance based on their sector identity, rather than considering *all* points, which seems silly.

It is important to note that the time complexity of the distance calculation could already be improved by discretizing and then using hamming distance as the distance metric without even needing to consider sectors; we test if this improvement incurs penalties in model performance though.

After all, discretizing simply groups instances into sectors by attribute individually anyway, and hamming distance is faster compared to Euclidean or other distance-based metrics. To advocate for sectors over discretizing and using hamming distance we note two potential advantages of this Sectors Approach for speeding up k-nearest neighbor determination. 1) Sectors considers far fewer points than *n* while discretizing and using hamming distance will still search through all *n* points, and 2) Sectors groups things by their overall similarity, like a summary of their closeness in all attributes, while discretizing and using hamming distance would only group things by their similarity across each attribute individually, which may not converge to the former. Thus, while evaluating the Sectors Approach, we must also test its potential speed improvement *and* its performance against discretizing and using hamming distance.

Correlation-Weighted Euclidean Metric: Mitigating Insignificant Attribute Bias

To address KNN's susceptibility to unimportant attributes, we propose weighting distance terms in the distance metric calculation by their correlation or entropy with the class. This would avoid situations like stated earlier where larger distances in unimportant attributes weaken the closeness of a point with high predictive value, being close in important attributes that determine class. For example, imagine if $d = 5$ and x_1 and x_2 had a correlation of 1 with the class while x_3 through x_5 had a correlation of 0 with the class. Now imagine the point we aimed to classify was (0, 0, 0, 0, 0). KNN would consider the point (0.5, 0.5, 100, 100, 100) to be *much* farther (distance ~ $3*10^4$) from the instance we would want to classify than a completely unrelated point, like (10, 10, 2, 7, 3) (distance = 262), even though we would consider the first one to be much more important and related to the point we wanted to classify. One might argue that dimensionality reduction would solve this issue, but instead of just black-and-white choosing to include an attribute or not, we want to make this importance-weighting continuous by weighting the distance terms in the distance calculation by a metric explaining the predictive value of an attribute.

Datasets and Preprocessing

For evaluating the Sectors Approach, the MAGIC Gamma Telescope Dataset from the University of California Irvine Machine Learning Repository was used. The rationale for using this dataset was: 1) All features were quantitative and continuous while the class was categorical, so the KNN algorithm could apply, 2) It was on the order of 10^4 instances, providing enough instances to observe time complexity effects for large n while not being too large to incur significant time penalties during testing, and 3) The binary classification task was relatively simple and thus a good application of KNN. The dataset consisted of 19020 instances of 10 quantitative continuous features and one categorical class variable: pulse (gamma signal or hadron background).

As mentioned earlier, discretizing and using the Hamming distance metric was proposed as an alternate time-saving solution. When testing this method, we used equal-width binning to generate 6 bins for each feature. Equal-width binning was chosen to preserve distance in the context of using KNN.

For evaluating the Correlation-Weighted Euclidean Metric, We used the well-known Diabetes Dataset available from the Waikato Environment for Knowledge Analysis (WEKA) software which consists of 768 instances of 7 quantitative continuous attributes, one quantitative discrete attribute, and one categorical class variable: diabetes outcome (tested positive or negative). When evaluating our weighted-Euclidean distance approach, we used k-fold cross validation with 5 folds to have more datapoints of comparison with the default KNN approach. The tables below describe the features of both datasets.

Features/Class	**Data Type**	**Description**
fLength	Quantitative Continuous Feature	major axis of ellipse (mm)
fWidth	Quantitative Continuous Feature	minor axis of ellipse (mm)
fSize	Quantitative Continuous Feature	log of sum content of pixels (#phot)
fConc	Quantitative Continuous Feature	ratio of sum of two highest pixels over fSize
fConc1	Quantitative Continuous Feature	ratio of highest pixel over fSize
fAsym	Quantitative Continuous Feature	distance from highest pixel to center, projected onto major axis
fM3Long	Quantitative Continuous Feature	3rd root of third moment along major axis (mm)
fM3Trans	Quantitative Continuous Feature	3rd root of third moment along minor axis (mm)
fAlpha	Quantitative Continuous Feature	major-axis ∠ with origin vector (°)
fDist	Quantitative Continuous Feature	distance from origin to center of ellipse (mm)
Class: pulse	Categorical **Class**	gamma signal/hadron background

Table 1: The MAGIC Gamma Telescope Dataset

Features/Class	**Data Type**	**Description**
preg	Quantitative Continuous Feature	Number of times pregnant
plas	Quantitative Continuous Feature	Plasma glucose concentration
pres	Quantitative Continuous Feature	Diastolic blood pressure (mm Hg)
skin	Quantitative Continuous Feature	Triceps skin fold thickness (mm)
insu	Quantitative Continuous Feature	Insulin (mu U/ml)
mass	Quantitative Continuous Feature	Body mass index (kg / m^2)
pedi	Quantitative Continuous Feature	Diabetes pedigree function
age	Quantitative Discrete Feature	Age (years)
Class: diagnosis	Categorical **Class**	Tested positive or negative

Table 2: The Diabetes Dataset

Experimental Design

Both the Sectors Approach and the Correlation-Weighted Euclidean Metric were tested in separate experiments to prevent their interference with each other.

With the MAGIC Gamma Telescope Dataset, the Sectors Approach was tested against a Control KNN and a KNN that used the MAGIC Dataset with 6 equal-width discretized bins across every feature. The k-value used for all the implementations was k = 9, since the control KNN performed best on the MAGIC dataset with k = 9 for k $\in$ {1, …, 20} with the MAGIC Dataset. The *s*-value was 3 and the d_{splits} value was 4 for the Sector Approach, splitting up the training space into $3^4 = 81$ 4-cubes. These values were chosen because for all of the values of *n*, when divided by 81, were on the order of 10^1 - 10^3, splitting the space finely enough to significantly reduce the amount of points but not too finely such that there weren't enough points in each 4-cube.

The MAGIC Dataset was sampled with *n* ranging in 13 increments of 1/20th of the dataset going from (1/20)**len*(MAGIC) to (13/20)**len*(MAGIC). Along with each of these samples of the MAGIC Dataset, a testing set of 100 instances, regardless of the magnitude of *n*, was formed out of the data remaining after the sample (so no testing instances were in the sample) and prepared and coupled with the sample. We stopped at 13 and did not go to (20/20)*n* to save time while testing. For each of these samples, all three models were prepared, and their overhead time was measured based on *n* after being provided the training data. They were then tested on the prepared testing sets of 100 instances and their classification time for the entire testing sets was obtained, and then divided by 100 to obtain the per-instance classification time based on *n*. Finally, their F1-Scores were collected from their classification of 100 instance testing sets. This data then lent itself to a holistic approach that compared the performance of the different models in both efficiency and performance (generalization), seeing how it changed across training set size. The following table summarizes this first Sector Approach evaluation experiment:

Independent Variable: KNN Implementation	Sector Approach - K = 9; s = 3; d_{splits} = 4.	Regular KNN - K = 9.	Discretize/Hamming Regular KNN - K = 9; 6 equal-width bins.
Independent Variable: MAGIC Dataset training sample with size *n*	$n = (i/20)*len(\text{MAGIC})$ for $i \in \{1, 2, \ldots, 13\}$	$n = (i/20)*len(\text{MAGIC})$ for $i \in \{1, 2, \ldots, 13\}$	$n = (i/20)*len(\text{MAGIC})$ for $i \in \{1, 2, \ldots, 13\}$
Supplied Testing Set	100 instances not appearing in training, no matter *n*	100 instances not appearing in training, no matter *n*	100 instances not appearing in training, no matter *n*
Metric/Data Discretized?	Euclidean/No	Euclidean/No	Hamming/Yes
Aim	See if Sector Approach beats out controls in DVs	Base Control	Alternate Idea Control
Dependent Variables Collected	Overhead Time, Per-instance classification time, F1-Score	Overhead Time, Per-instance classification time, F1-Score	Overhead Time, Per-instance classification time, F1-Score

Table 3: The Sector Approach evaluation experiment, summarized. Note *len*(MAGIC) = 19020.

For the evaluation of the Correlation-Weighted Metric implementation, model generalization was the focus of the experiment. As discussed earlier, the Diabetes dataset was used because of its (mostly) quantitative continuous features and categorical class, so KNN applies. It was tested against the performance of a control KNN model in F1-Score as a holistic measure of model generalization abilities. An 80-20 training-testing split was used, and different values of k this time opposed to *n* were used as the independent variable, since covering the bases of generalization abilities rather than time complexity was the focus of this experiment. The following table summarizes the experiment for the Correlation-Weighted Metric implementation.

KNN Implementation	Regular KNN	Regular KNN
Independent Variable: Metric	Correlation-Weighted Euclidean	Euclidean
Independent Variable: K	K $\in$ {1, …, 20}	K $\in$ {1, …, 20}
Dataset/Train-Test Split	Diabetes/k-fold for k=5	Diabetes/k-fold for k=5
Aim	See if model performance is improved	Control
Dependent Variable	F1-Score	F1-Score

Table 4: The Correlation-Weighted Euclidean Metric evaluation experiment, summarized.

Results and Analysis

The results and analysis of the experiment evaluating the Sectors Approach are presented below with accompanying figures:

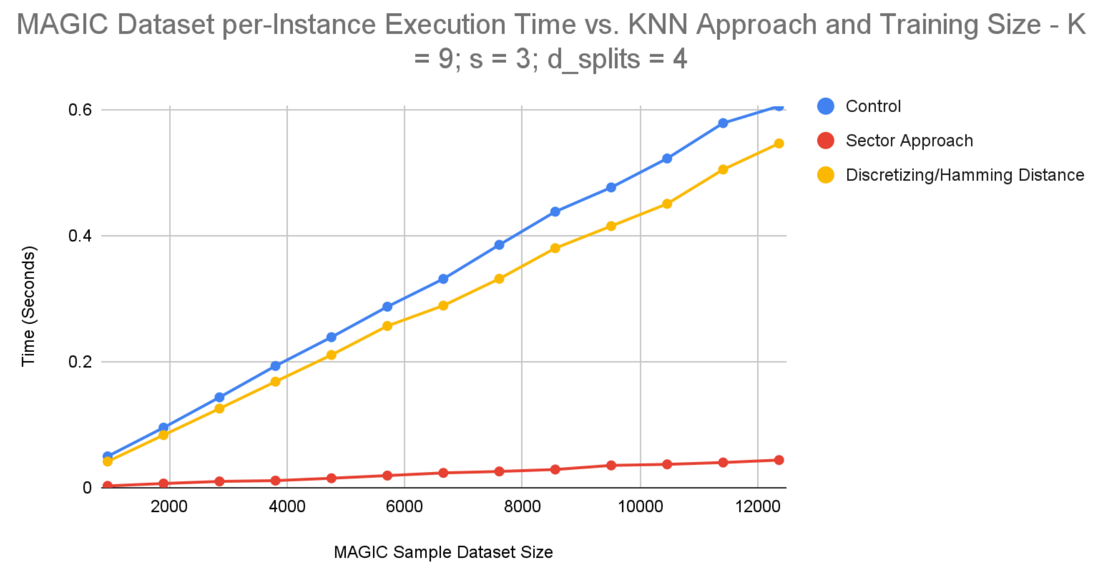


Figure 4: The per-instance execution time for each KNN approach and increasing dataset sizes. Each point represents a MAGIC Dataset sample.

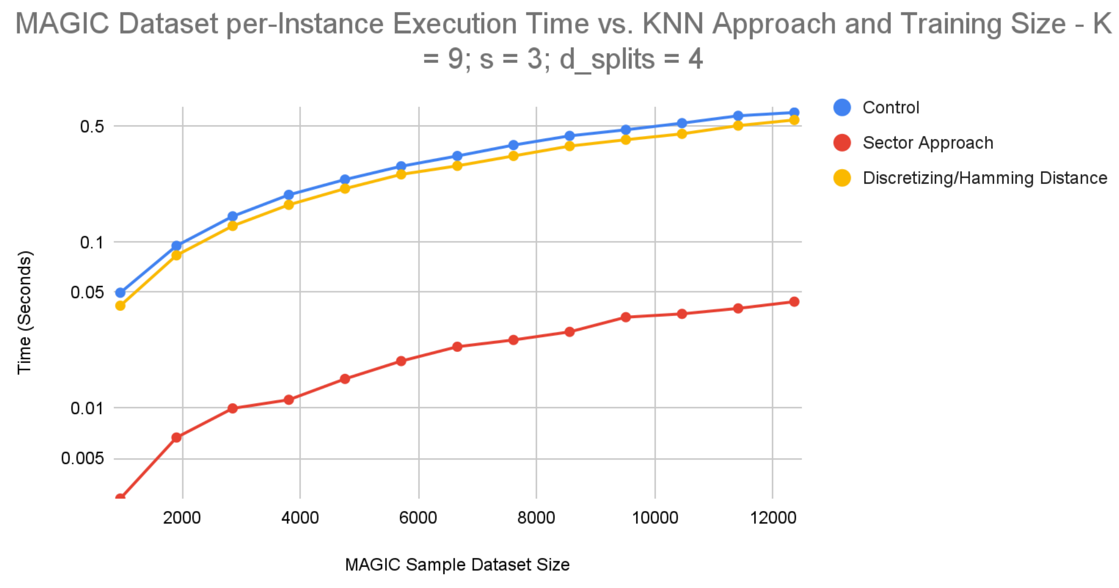


Figure 5: Figure 4 with the y-axis logarithmically scaled for better viewing of the time-complexity of the sectors approach

Figures 4 and **5** above illustrate complete dominance in time-complexity by the Sectors Approach. **Figure 6** below further drives the point home by illustrating how many times longer the Control takes than the Sectors Approach for values of *n*. Discretization with Hamming distance still is a slight improvement over the control, as less complicated operations are required, but going through *d* dimensions and *n* instances is still required anyway. It can be observed that all of these approaches are linear–this lines up with our time complexity analysis in **Methods**. For the control, the per-instance classification time complexity is expected to be *~O(nd)* while for the Sectors Approach it's expected to be $\sim O(sd_{splits} + (n/s^\wedge d_{splits})d)$ (see **Methods** for an explanation of the big-O analysis). Substituting for *s*, *d*, and d_{splits} in both, we obtain *~O(10n)* for the former vs. $\sim O(81 + (n/81)*10) \cong \sim O(0.1235n)$. This means that on average, assuming roughly uniform data split across sectors, the Sectors Approach should be about 81 times faster with these settings of *s*, *d*, and d_{splits}. This theoretical estimation does not come to fruition, rather, the Sectors Approach ends up being about 13.5 - 17.25 times faster (see **Figure 6**). This is obviously because we ignored the *sd* term in the Sector big-O analysis and data is obviously *not* uniformly split, making the estimation of there being $n/(s^\wedge d_{splits})$ points in each sector a naive assumption. 13.5 - 17.25 times faster is still, however, a resounding success.

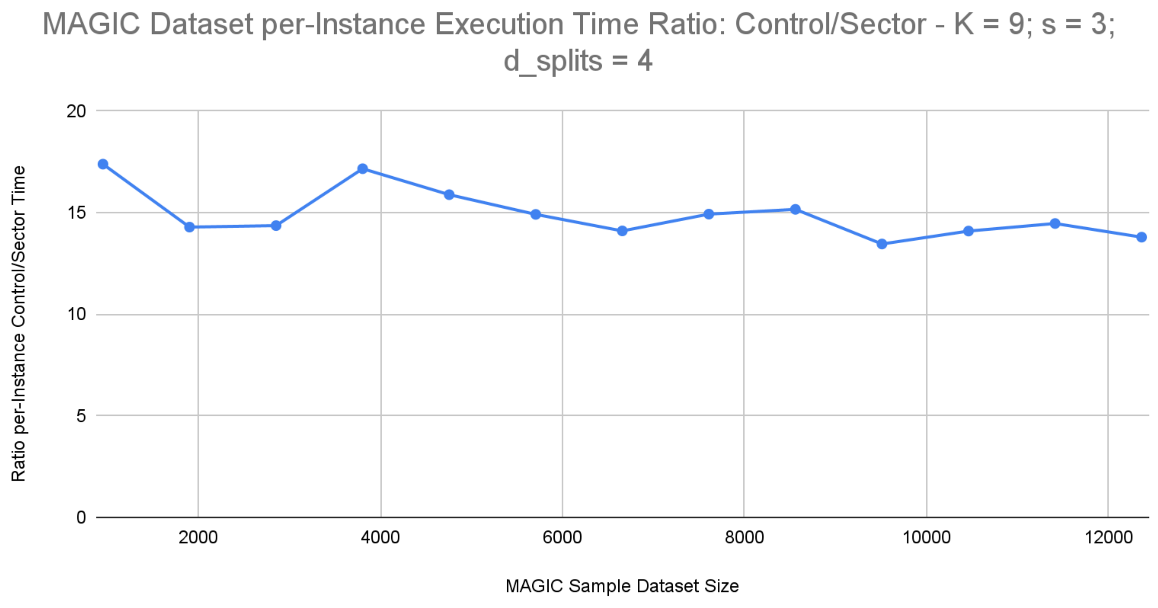


Figure 6: This figure illustrates that the Sector approach with the current configuration (k = 9; *s* = 3; d_{splits} = 4) can be expected to be roughly 13.5 - 17.25 times faster than using regular KNN.

Obviously, it is highly important that the Sectors Approach can generalize just as well as the controls. **Figure 7** below illustrates through F1-Score that Sectors Approach did not disappoint on this front.

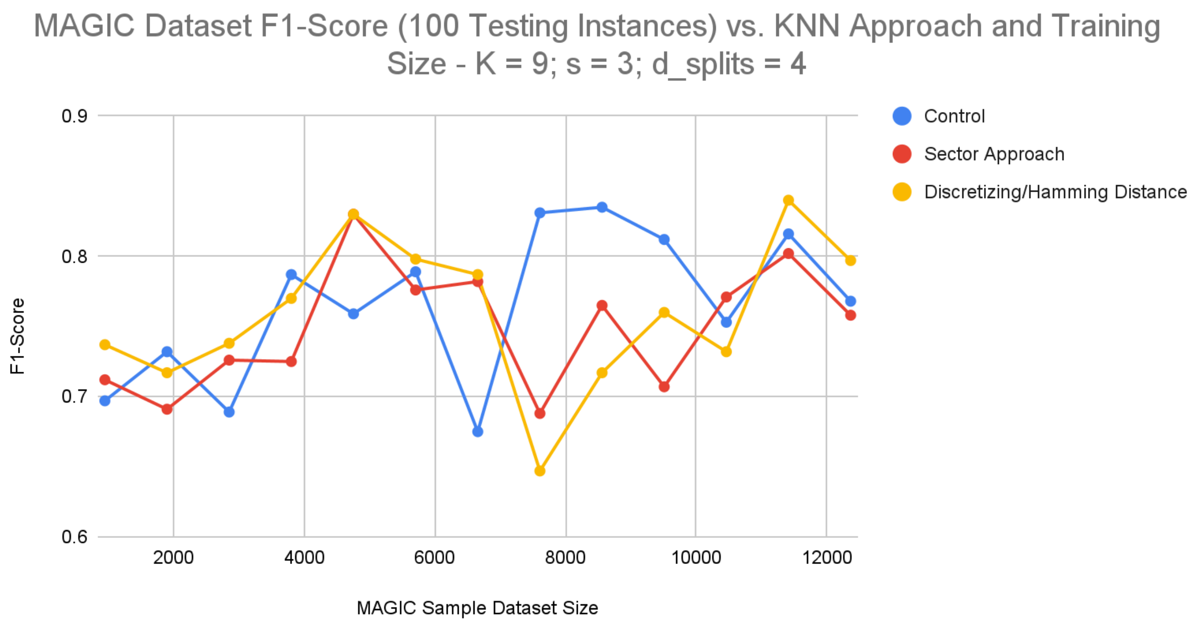


Figure 7: The Sectors Approach consistently has comparable generalization abilities to the Control in terms of F1-Score. The relative time-cost of the Control (or even the discretized Hamming approach) suggests that the Sector Approach is superior.

The astute reader will appreciate the success of the Sectors Approach but also point out that we have not considered the overhead costs of the Sectors Approach. Such an overhead cost is linear with the data estimated to be *O(nsd)* as described in the big-O analysis in **Methods**. This is because the training points have to be grouped into their sectors. **Figure 8** shows the overhead initialization (feeding training data) costs across all KNN approaches based on *n* showing that the grouping of points into sectors is significant. **Figure 9** shows the overhead cost based on *n* compared to the per-instance classification time for the Sectors Approach. The fact that this overhead has to only be done once in practice, before the model begins real-world use makes it near-0.

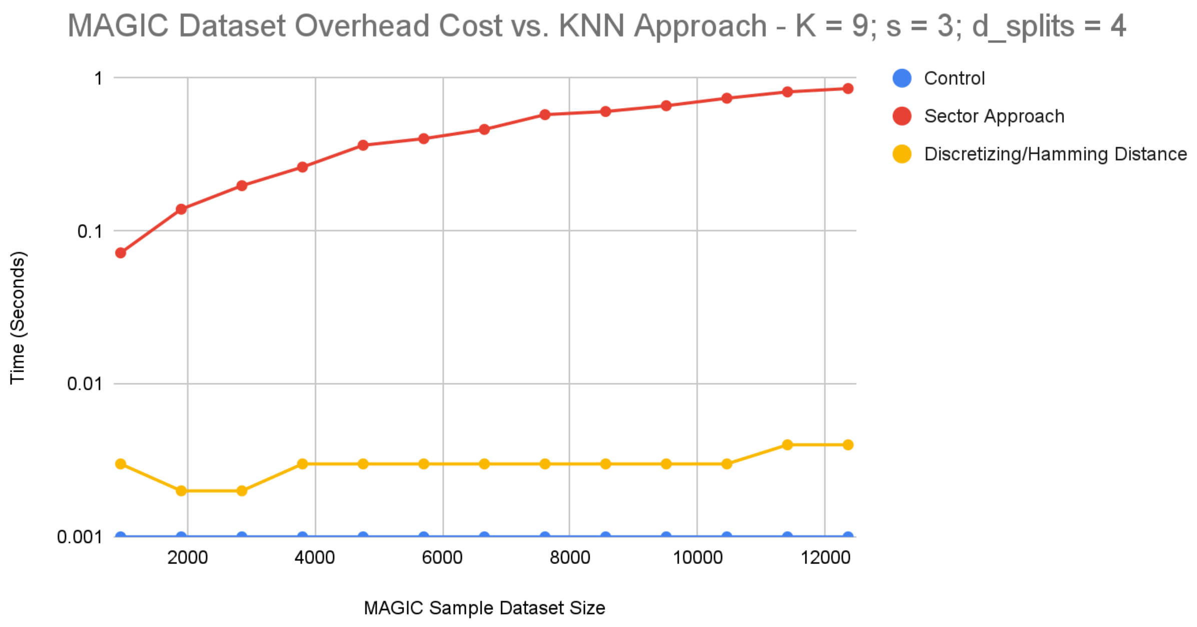


Figure 8: Overhead Costs for all KNN Approaches. The Control requires a simple initialization while the cost of discretizing is included in the overhead of the Discretizing/Hamming distance approach. The y-axis is logarithmically scaled, so the Sector Approach's overhead cost scales linearly ~*O(nsd)* as expected.

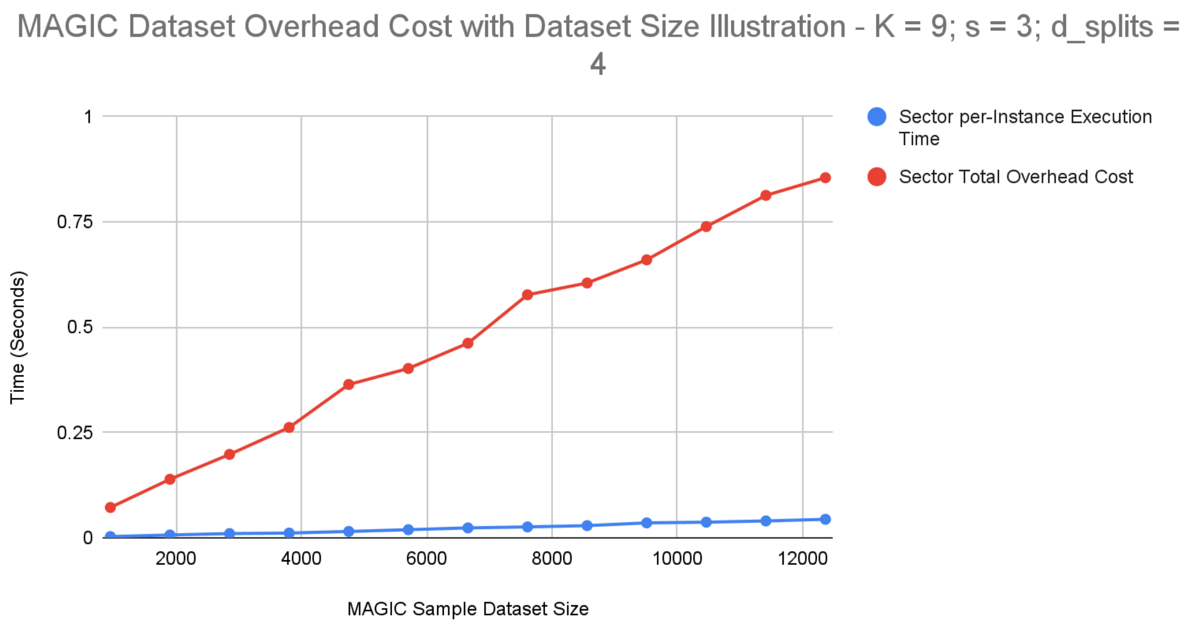


Figure 9: The overhead cost is near-0 in practice, since it only must be done once before the model can be used any number of times. However, the overhead cost still does grow much faster than the per-instance classification time.

For the second experiment evaluating the performance of the Correlation-Weighted Euclidean Metric, F1 score was considered in the modified KNN algorithm on each of the five testing sets from the k-fold cross validation. We then averaged the F1 score across each testing dataset. This was repeated for a range of k values (for the KNN algorithm) in [1, 20]. The results were finally plotted in the following figure for both the normal and our correlation-weighted Euclidean distance metrics.

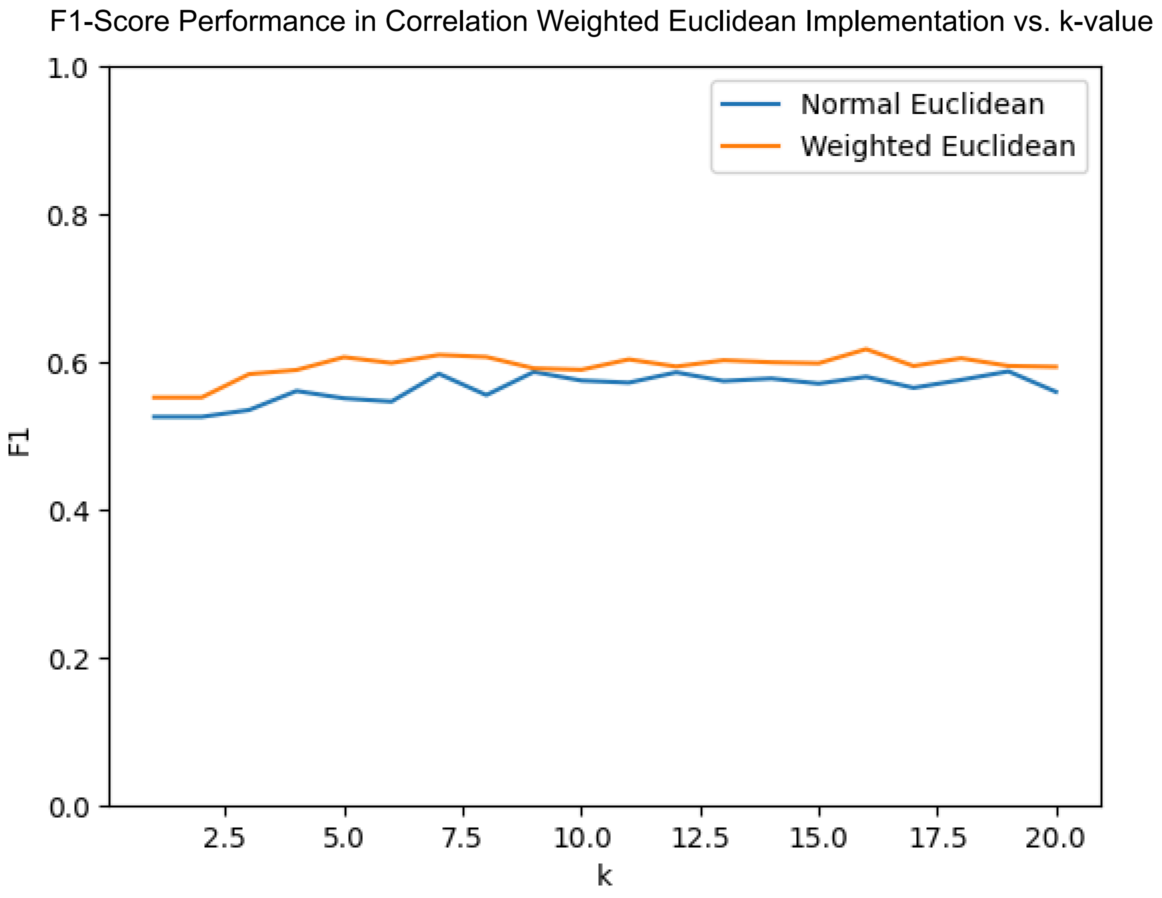
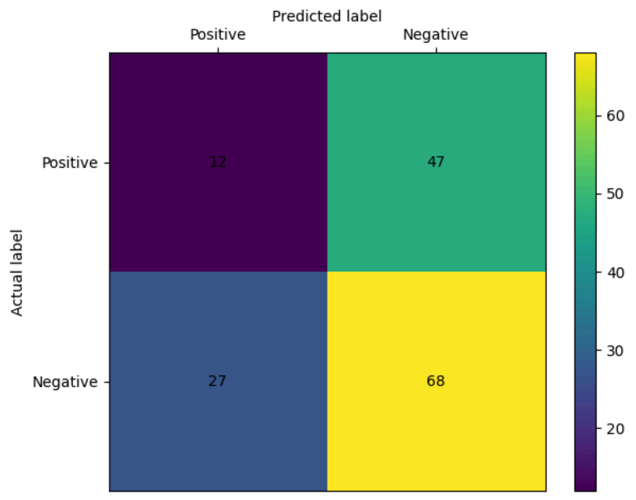


Figure 10: The F1-Score across varying k for the Correlation-Weighed and regular KNN models. The performance of the Correlation Weighted Euclidean Metric implementation was consistently better regardless of k.

Notice that the orange line, which denotes our own distance metric, is consistently higher than the blue line, which denotes the regular Euclidean distance. Since a higher F1 score indicates better classification, this shows that incorporating correlation coefficients into a KNN distance metric is superior regardless of the k-value chosen, indicating all-around improved performance.

Context for our F1-Scores taken is given by an example confusion matrix along with the F1-Score in the figure below:



$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Figure 11: Example confusion matrix as well as the formula for F1 Score used, illustrating its application as a holistic metric of model precision and recall.

Conclusion

Sectors outperformed regular KNN and discretization with hamming distance by almost 13.5 - 17.25 in terms of speed with comparable F1-Score, and the correlation-weighted Euclidean distance metric performed with better F1-Score than the regular Euclidean distance metric regardless of the k-value chosen. The success of both of these improvements in their respective domains of time-complexity and improving model performance suggests that implementing both of these methods together over pure KNN would create a faster and stronger model.

If we had more time, we would test our algorithms on a wider variety of datasets. d, d_{splits}, and s could also be varied in further experiments. Based on the performance of the regular Euclidean distance, the Diabetes dataset was not too well-suited for KNN classification. The improvement from weighting attributes by correlation could be coincidental, and we would want to check if this new metric performs worse on datasets that typically work for normal Euclidean. We would also want to improve our metric by considering the effects of multicollinearity, or the linear dependency across multiple attributes and predictors. This can be done by either checking the Variance Inflation Factor or combining the conflicting attributes into a single attribute. Additionally, other metrics in addition to correlation could be used, such as nonlinear correlations, or the Matthews correlation coefficient for discrete data, to mention a few. These are possibilities for future work.

Contributions

Shreraj focused on creating and evaluating the Sectors Approach, and Olivia implemented the Weighted Euclidean distance KNN while also creating the Discretize/Hamming approach. We wrote our corresponding portions of the Results, Conclusions, and presentation. Shreraj wrote the Introduction and Methods, and Olivia wrote the Related Works and Abstract sections.

References

Huang, J., Wei, Y., Yi, J., & Liu, M. (2018). An improved kNN based on class contribution and feature weighting. 2018 10th International Conference on Measuring Technology and Mechatronics Automation. https://doi.org/10.1109/ icmtma.2018.00083

Kalaivani, P., & Shunmuganathan, K. L. (2014). An improved k-nearest-neighbor algorithm using genetic algorithm for sentiment classification. 2014 International Conference on Circuits, Power and Computing Technologies. https://doi.org/10.1109/iccpct.2014.7054826

Karabulut, B., Arslan, G., & Ünver, H. M. (2019). A weighted similarity measure for k-Nearest neighbors algorithm. *Celal Bayar Üniversitesi Fen Bilimleri Dergisi,* 15(4), 393-400. https://doi.org/10.18466/cbayarfbe.618964

Ostrovsky, R., & Rabani, Y. (1998). Efficient search for approximate nearest neighbor in high dimensional spaces, *Thirteenth annual ACM symposium on Theory of computing.* https://doi.org/10.1145/276698.276877

Vahedifar, M. A., Akhtarshenas, A., RafatPanah, M. M., & Sabbaghian, M. (2025). Shapley-Based data valuation with mutual information: A key to modified k-nearest neighbors. 2025 IEEE 35th International Workshop on Machine Learning for Signal Processing. https://doi.org/10.1109/mlsp62443.2025.11204262