Optimizing k-Nearest Neighbor Classification Efficiency and Efficacy

By Shreraj Sainani, Olivia Wu

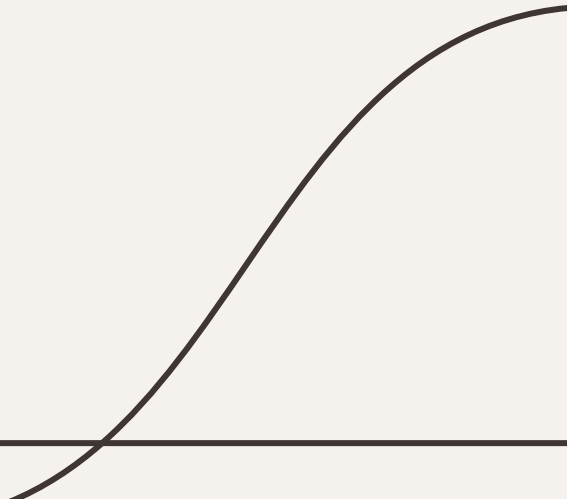
2 Pure-KNN Problems we Address

- Improving KNN
- 1. Time Complexity for classification ~O(nd) (every point). Slow for large datasets/high dim
- 2. KNN can’t account for insignificant features. Larger distances in unimportant attributes can weaken the closeness of points.

Solutions - Improving Time Complexity

- Seems silly to consider all points when finding the k-nearest neighbors, some are just clearly far away.

- Optimizing finding the k-nearest neighbors by splitting up the training space in to “sectors”



Sectors Approach

- Dataset has *n* instances with dimension *d*.
- Split up training data into sectors and classify instances based only on their constituency in their sector
- If less than k points in sector, consider all points in sector. If no points in sector, choose random label (both highly unlikely with proper setup)
- Sectors are *d*-cubes, cubes of dimension *d*: (1-cube: line, 2-cube: square, 3-cube: cube, 4-cube: hypercube/tesseract, etc.).

Sectors Approach

- Not balls or spheres or any other sort of complex shape–partitioning up the space into d-cubes was simplest
- Split each dimension *s* times.
- Determine sector a point falls into: *d* inequalities (some number < x1 < some number, ..., some number < xd < some number).
- Equal-width to preserve distance in KNN context. Min/max abolished.

Sectors Approach - Issue/Solution

- Even for small *s* and *d*, you get exponentially increasing amounts of *d*-cubes with higher *d*.
- Solution: only split along some dimensions, notated *d_splits*.
- These dimensions that are to be split are the ones with the highest SD, since splits that more evenly/uniformly split up the data are preferred
- (don’t want to have a *d*-cube with most of the training data and others with few)

Illustration: $s = 3$; $d = d\_splits = 1$

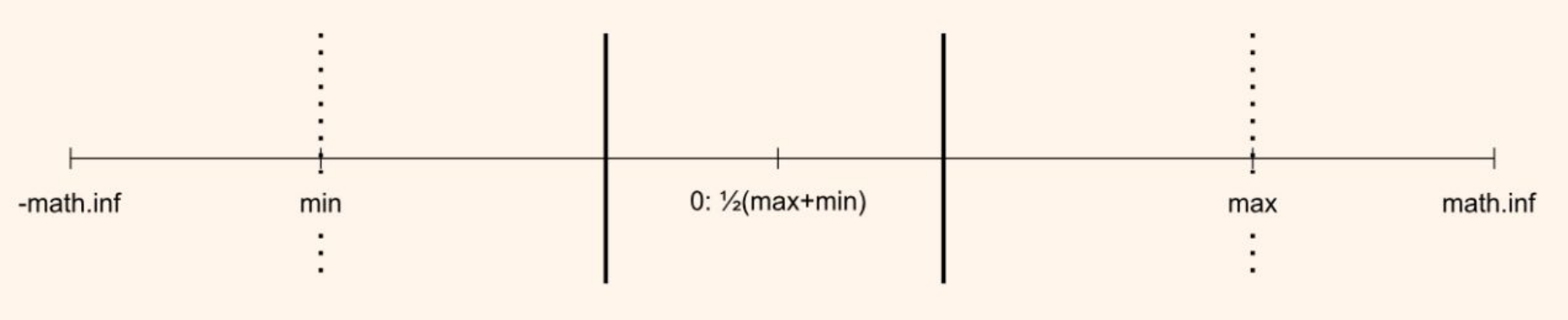


Illustration: $s = 3$; $d = 2$; $d\_splits = 1, 2$

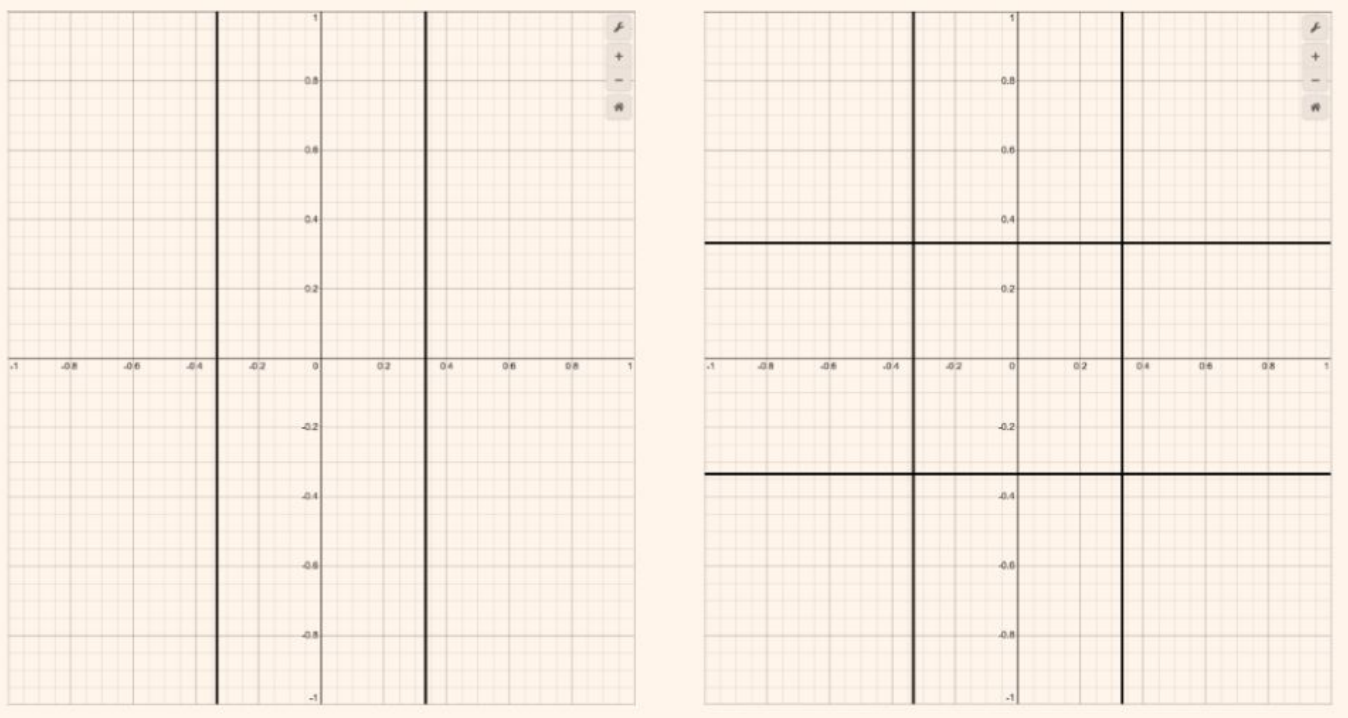
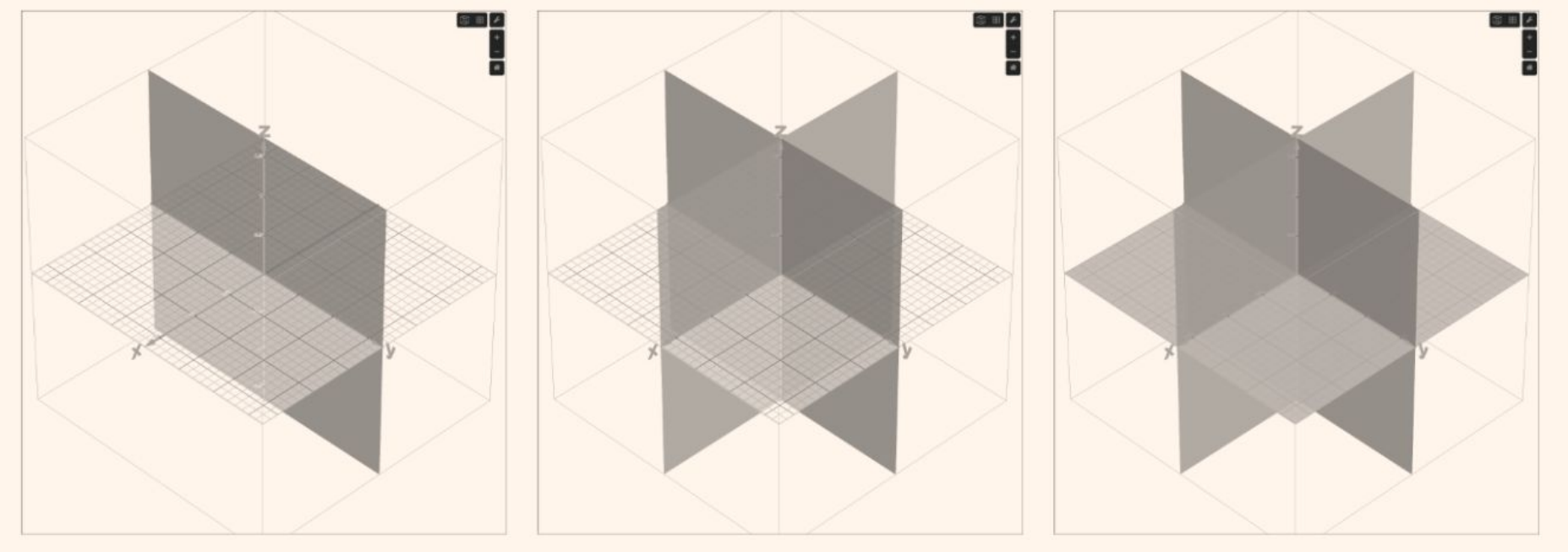


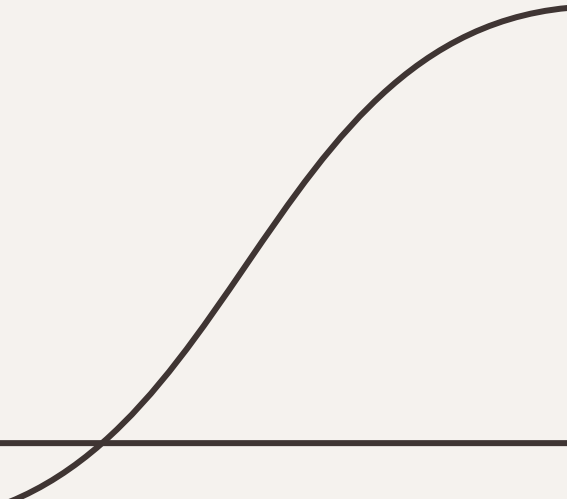
Illustration: $s = 2$; $d = 3$; $d\_splits = 1, 2, 3$



Pattern: *s*^(*d_splits*) *d*-cube sectors extend for higher dimensions

How is this different than discretizing?

- Holistic summary in terms of attributes
- Preserves quantitative nature
- Discretizing would still require you to look at all points and do Hamming distance (still ~O(nd))



Theoretical Improvement

$$\sim O(nd) \text{ to } \sim O(sd_{splits} + (n/s\text{^}(d_{splits}))d)$$

- The first term *s*d_splits* comes from classifying a testing instance into its respective *d*-cube–for each feature that's split, check all the bounds for that feature.

- The second comes from the distance calculations , where assuming points are **somewhat uniformly distributed**

Solutions - Mitigating Bias Towards Unimportant Attributes - Correlation-Weighted Euclidean

$$D = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

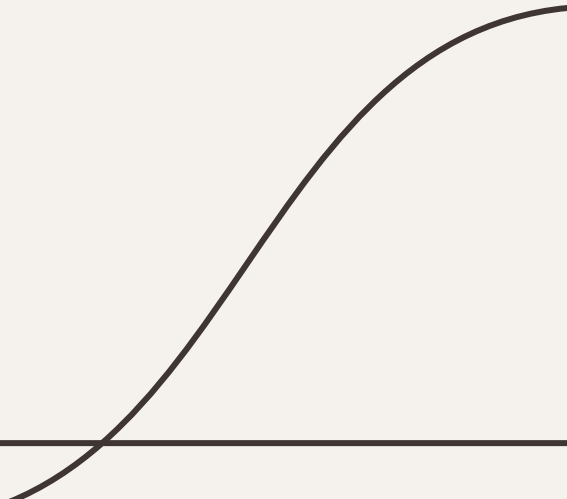
$$D = \sqrt{\sum_{i=1}^{n} |c_i(x_i - y_i)|^2}$$

- c_i is the correlation coefficient between the i-th attribute with the class variable
- We square c_i because we want the weight value to lie in the interval [0, 1]
- This also reflects the purpose of the coefficient of determination, which indicates how much variation in the class is explained by the i-th attribute

How is this different from data preprocessing? (dim reduction)

- One might argue that dimensionality reduction would solve this issue

- Instead of just black-and-white choosing to include an attribute or not, we want to make this importance-weighting continuous.



Related Works

- Karabulut et al. (2019) made their own weighted similarity measure, but their weights were static and prone to underfitting
- Optimization through the use of genetic algorithms is computationally expensive
- Ostrovsky and Rabani (1998) work on approximate nearest neighbor search in high-dimensional spaces

Sectors Evaluation Dataset - MAGIC

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 10e4 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.

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

Corr. W. Euclidian Evaluation Dataset - Diabetes

- 768 instances of 7 quantitative continuous attributes, one quantitative discrete attribute, and one categorical class variable: diabetes outcome (tested positive or negative) → application of KNN

- K-fold cross validation

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

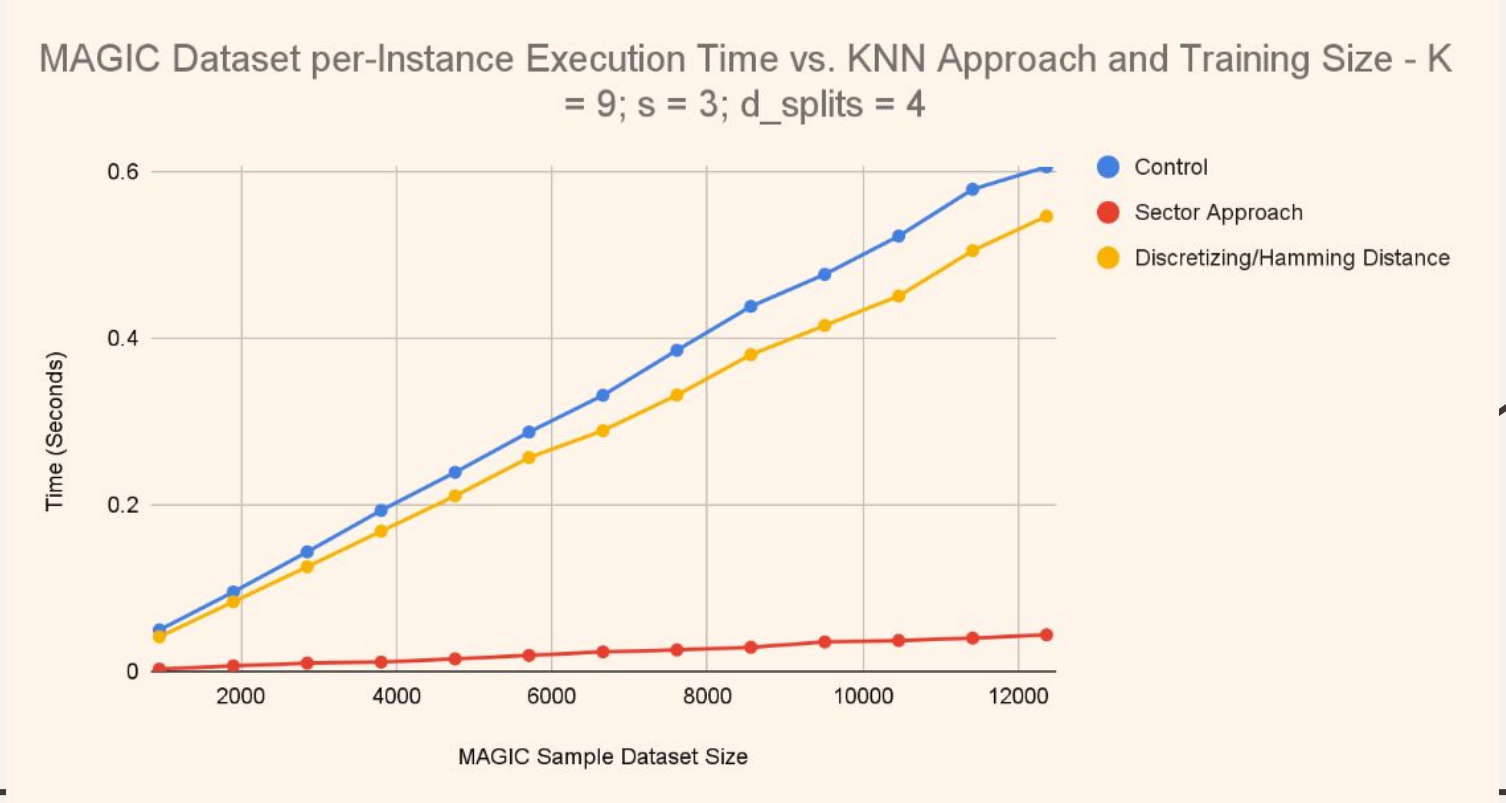
Experimental Design - Sectors

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

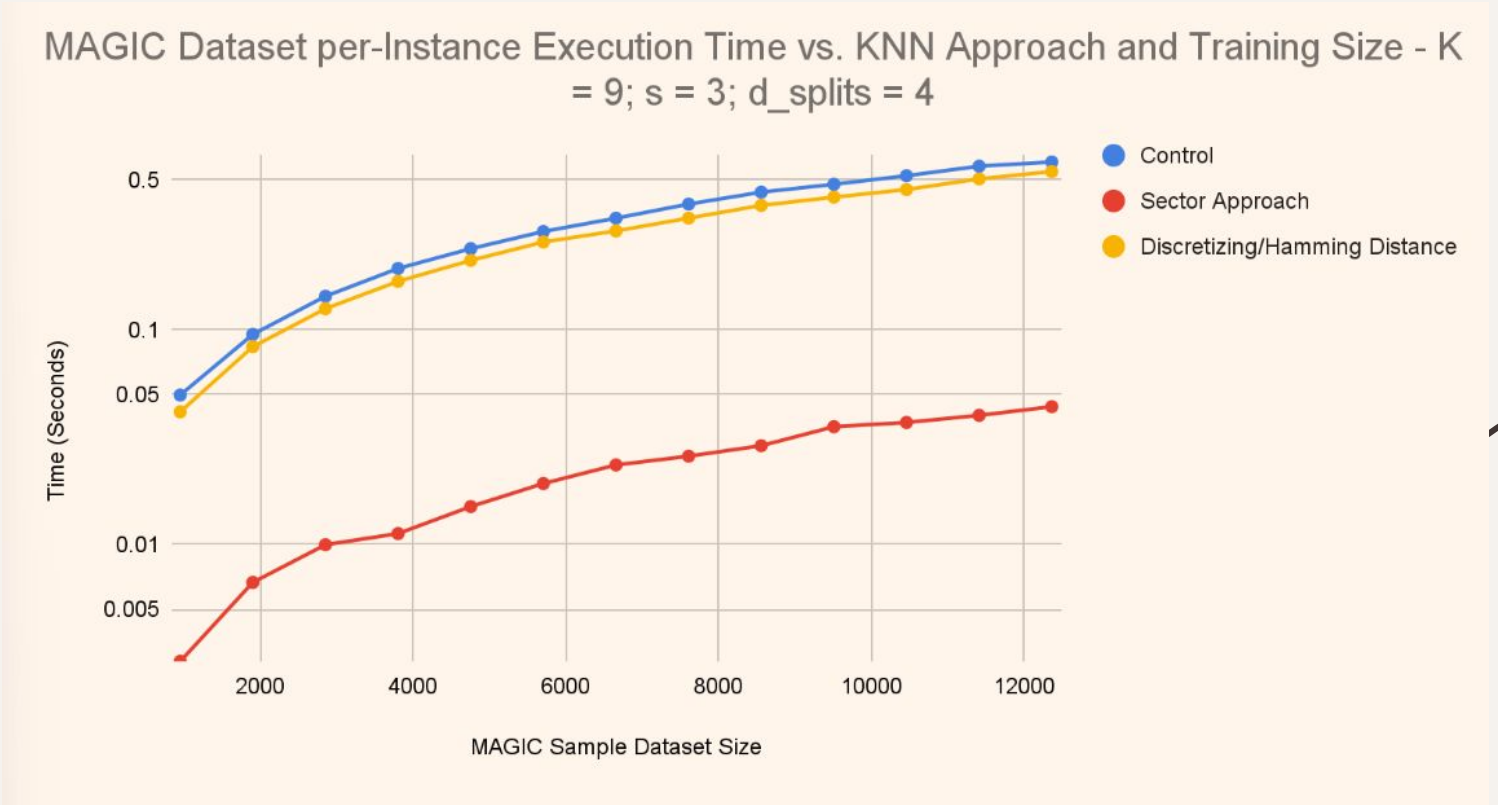
Experimental Design - Corr. W. Euclidian

KNN Implementation	Regular KNN	Regular KNN
Independent Variable: Metric	Correlation-Weighted Euclidean	Euclidean
Independent Variable: K	$K \in \{1, \ldots, 20\}$	$K \in \{1, \ldots, 20\}$
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

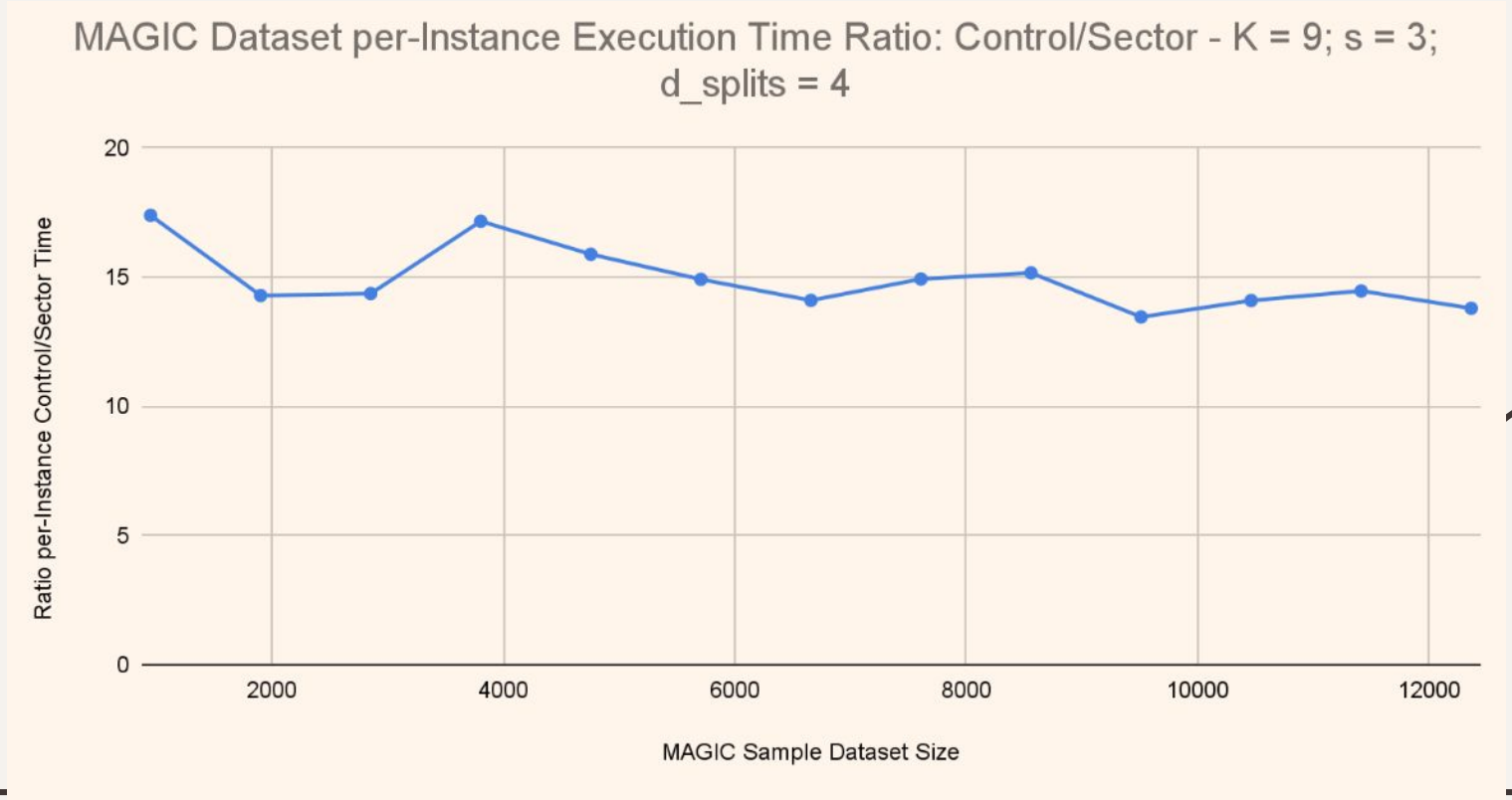
Results


MAGIC Dataset per-Instance Execution Time vs. KNN Approach and Training Size - K = 9; s = 3; d_splits = 4
Time (Seconds)
0.6
0.4
0.2
0
2000
4000
6000
8000
10000
12000
MAGIC Sample Dataset Size
Control
Sector Approach
Discretizing/Hamming Distance

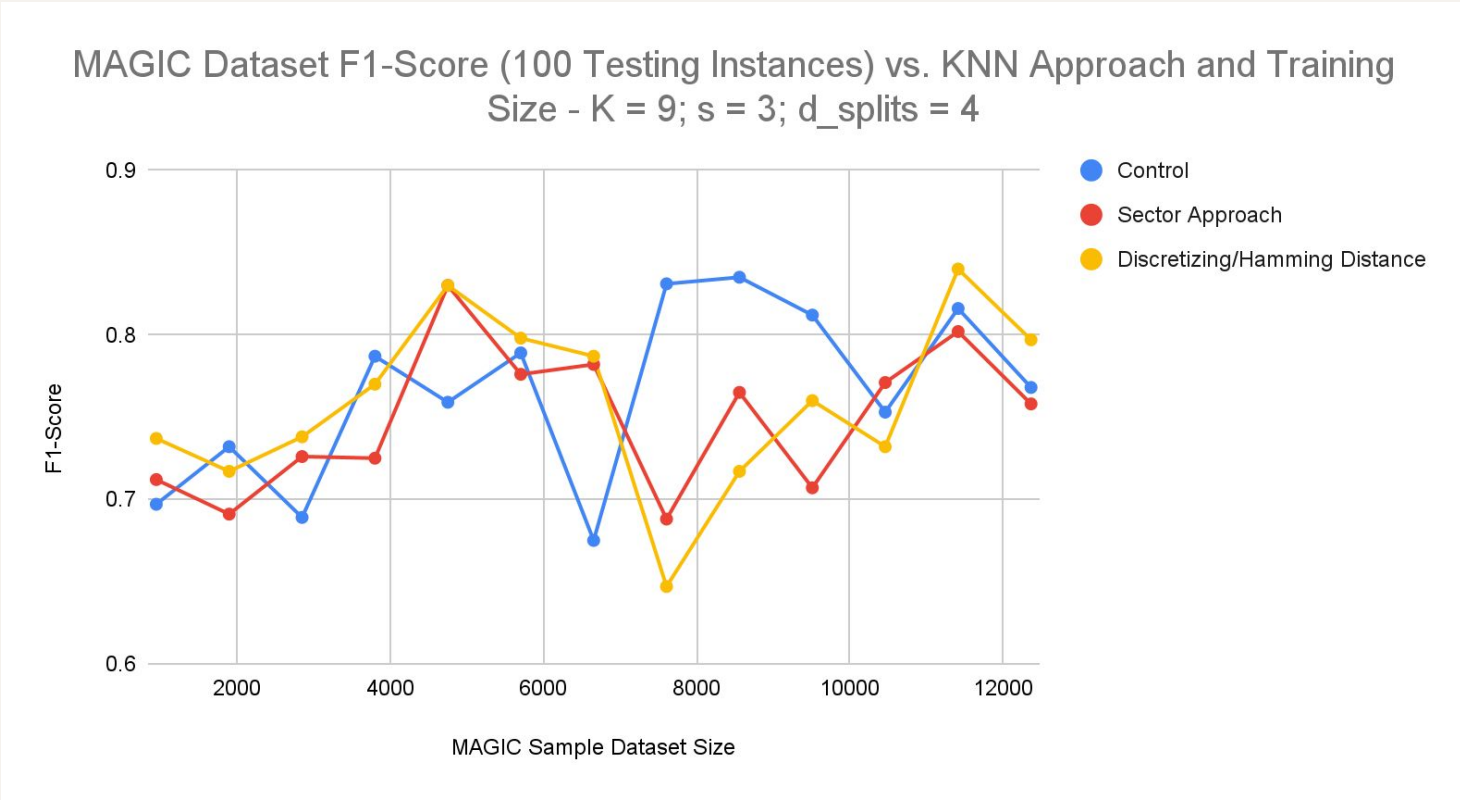
Results


MAGIC Dataset per-Instance Execution Time vs. KNN Approach and Training Size - K = 9; s = 3; d_splits = 4
Time (Seconds)
0.5
0.1
0.05
0.01
0.005
2000
4000
6000
8000
10000
12000
MAGIC Sample Dataset Size
Control
Sector Approach
Discretizing/Hamming Distance

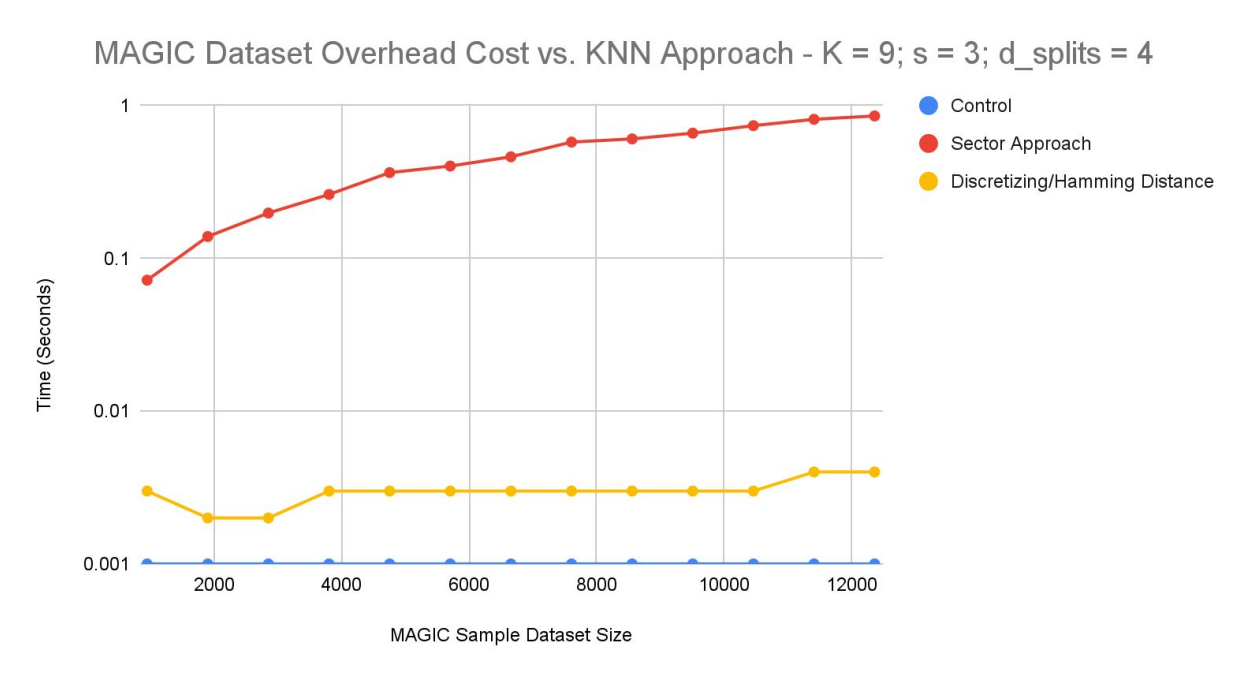
Results - Expectation (81) vs. Reality (13.5 - 17.25)



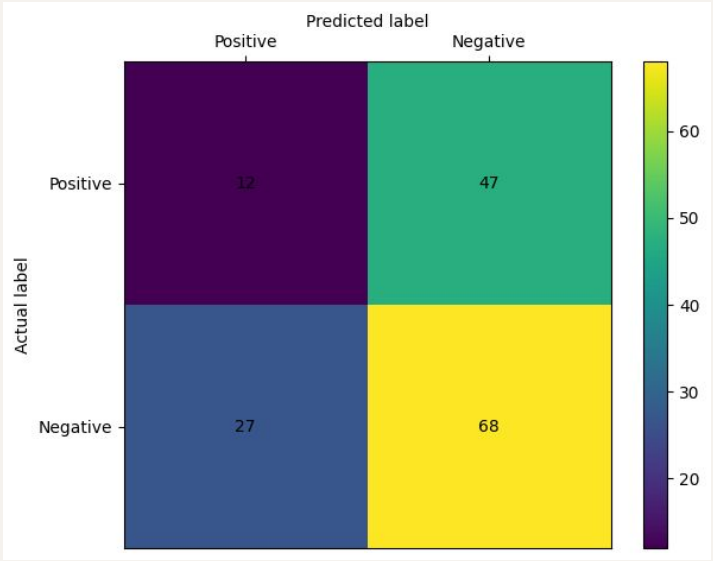
Results - Sectors is Competent

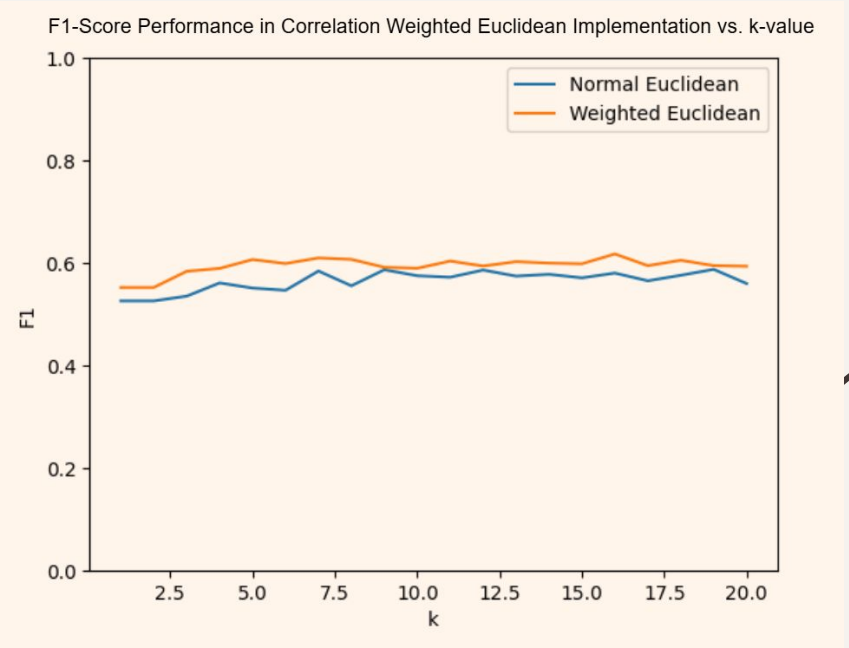


Results - Overhead ~O(ns * d_splits)



Results - Correlation W. Metric





Conclusion

- Sectors outperformed regular KNN and discretization with hamming distance by almost 13.5 - 17.25 in terms of speed with comparable F1-Score,
- Correlation-weighted Euclidean distance metric performed with better F1-Score
- The success of both of these improvements in their respective domains of time-complexity and improving model performance suggest they should be implemented in KNN.
- Consider checking Variance Influence Factor to account for potential linear dependencies across attributes → reduce effects of multicollinearity
- Create a metric that will work for attributes that are not linearly correlated to the class variable → what about quadratic relationships and such
- Test our algorithms on even more datasets than just Diabetes