CSE 514A Data Mining: Assignment 2

Danni Beaulieu (ID 497567)

Introduction

In the UCI Letter Recognition Data Set, there are 16 features created from “statistical moments and edge counts” of black and white letters of the English alphabet; the characters themselves were derived through randomly distorting 20 different fonts [1]. While the dataset itself is not composed of handwritten characters, it is not far fetched to imagine that it could assist with identifying them. The fonts and their distortions may act as a simulation of different handwriting styles. Machine learning models may train on these if a dearth of handwriting data exists or is not accessible for legal, privacy, or other reasons. Another real life situation these characters may simulate even more closely is one in which characters have been printed onto paper. Applications of the character recognition models may be scanning documents like receipts, recognizing signs for traffic or businesses, or reading lettering from photographs.

To determine the “best” classifier for deciding between two specific letters, say ‘H’ and ‘K’ or ‘M’ and ‘Y’, there are a few metrics to consider. We can use model accuracy or loss, however variance explained (R-squared) does not make sense for classification problems. AUC would be a better way to evaluate than R-squared, especially considering the dataset is relatively balanced. If it were severely imbalanced, it might be better to look at the PR curve. We should also examine complexity and interpretability. For instance, if these models are deployed in real time systems, minimal gains may not be worth the expense of a deep neural network or even a random forest. Similarly, it can be easier to iterate on systems and explain them to others when the type of model matches the task. That is, at least intuitively, as a human, letter recognition does not seem to match with a decision tree since no sequences of choices are consciously made and no chain of consequences come to mind when thinking about identifying one letter versus another. It may seem to match more easily with a Naive Bayes model based on probabilities or an SVM which leverages margins of separability for classification problems. Indeed it is for these reasons, I selected Bayes and SVM for my experiments. KNN may also be a fair choice, given an appropriate distance metric, but again could venture into unintuitive territory and generally requires continual storage of all data, which could limit deployability. As an expanded version of this experiment, I have elected to include KNN.

Another step to consider is dimensionality reduction. This preprocessing step is motivated by desires to reduce complexity of models, reduce overfitting, reduce size of data storage, and reduce training times. In this dataset, we have 20,000 samples which means 320,000 input aspects total to the model. Reducing from 16 features down to, say 4, brings us down to 80,000 input aspects total. This is quite a significant jump. Depending on where the models are deployed, any and all of these motivating goals may come into play. If a model pipeline will need to be run often with new input data, training time becomes very relevant. Similarly, if the model is deployed on a small device, data storage becomes relevant. Finally, while reducing

complexity of the models, depending upon the type of dimensionality reduction chosen, it may not increase the explainability of the models, which is usually one advantage of reduced complexity that is not present in dimensionality reduction.

Without any missing values to remove or impute and without a strong understanding of the features themselves from domain knowledge, simple quality filtering does not seem to be the best choice. Likewise, since this is not a regression problem and I have elected not to use decision trees or a random forest, embedded methods are not well-suited. Wrapper feature selection is a possibility, although it is somewhat of a brute force technique. Lastly, we have filter methods, feature extraction, and a range of options including PCA, LDA, ICA, SVD, NMF, and t-SNE. Originally, LDA seemed like a front-runner since it performs well with classification and does not require standardization, which makes Gaussian assumptions about the data. Unfortunately, the number of features it creates is the number of classes minus one; for binary classification, this would result in only one feature. ICA also does not necessarily seem ideal since it assumes the features are statistically independent and non-Gaussian; in letter classification it seems unlikely the features would be wholly independent of each other. SVD is most popularly applied with sparse data, which is not the case for us here. NMF has been said to be more or less a constrained version of SVD in which between the two, SVD optimizes predictions while NMF optimizes explainability. A benefit of PCA is that since it uses orthogonal transformation which maximizes the variance of the dataset, the resulting extracted features should not be correlated with each other. Finally, t-SNE did not seem like a good fit since it is generally used more for visualization and can be incredibly sensitive to parameter choices, possibly producing misleading results. Given that we know the features will contain correlations, overall, PCA seemed like the most promising technique for this dataset. Additionally, it includes noise reduction and does not require additional parameters. Upon further investigation, however, I discovered that one of the Naive Bayes models I wanted to use (multinomial) required positive values. This limited the options and led me back to examine filter methods. A method that seems to carry some similar benefits to PCA is to select the best features according to mutual information score. This is a supervised filtering method which measures the mutual dependence between variables. In this case, the higher the score, the more likely the feature is connected with the target response variable; it tries to maximize information gain about the target. I chose to normalize the data and then use this technique. I also considered using the unsupervised approach of feature agglomeration, however this did not perform as well in initial explorations. Note, I often use “preprocessing” to refer to all steps before modeling.

Framing the alphabetical classification problem as pairs of binary problems is interesting. It seems like discriminating between ‘H’ and ‘K’ could operate as an intersection detector, whereas discriminating between ‘Y’ and ‘M’ could operate as a detector for additional line segments. I say this because one way to think of the difference between ‘H’ and ‘K’ is where the middle to right line segments meet. That is, do they meet in the middle or on the right-hand side of the letter? One way to think of the difference between ‘Y’ and ‘M’ is whether the left-hand line segment and right-hand line segment are noticed or not and whether the short middle line segment is noticed or not. In this vein, I chose to predict whether letters are ‘O’ or ‘D’ and think of this differentiator as whether roundness is detected or not. I also feel these two letters will be

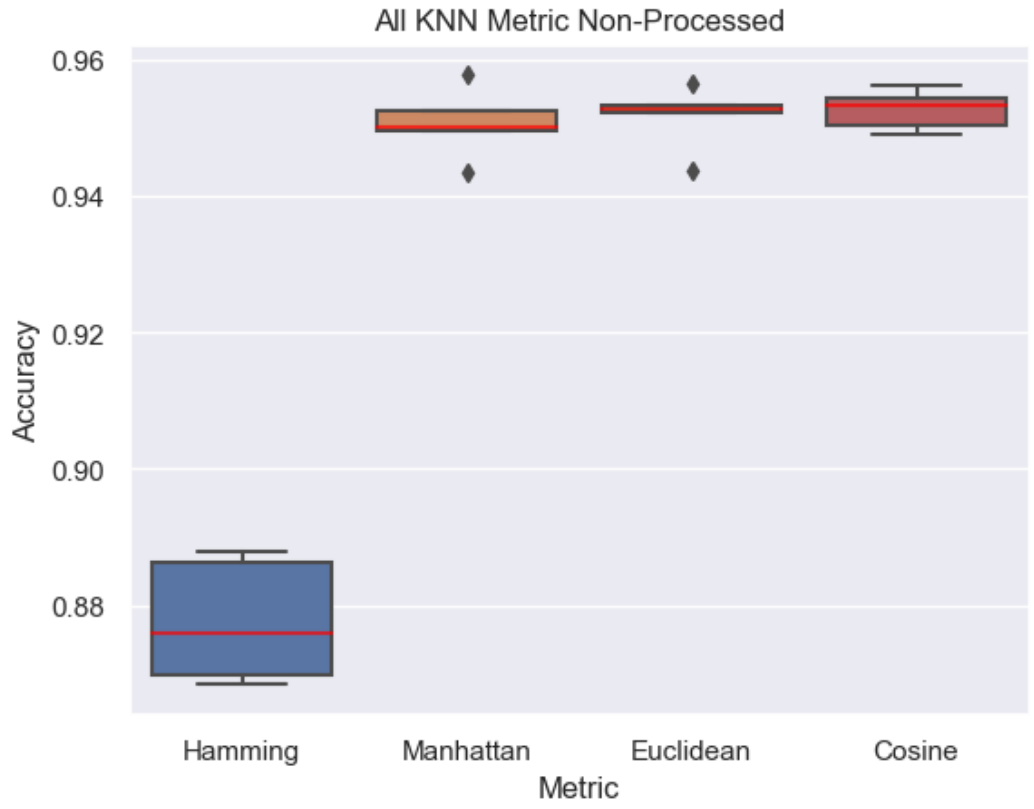
the most difficult to classify against each other and therefore will be a good metric for my models to ensure they can perform well. The easiest pair to classify may be a pair like ‘O’ and ‘X’ wherein the visual disparity between the two seems very high. ‘O’ is a round circle while ‘X’ consists of line segments only. A multi-class classifier significantly compounds problems like we just discussed with the binary classifiers. In some respects, with 26 options, there is a larger spectrum for error, and it is harder to tell when a classifier is “almost there” versus completely wrong. For instance, if it gets ‘O’ and ‘D’ mixed up it might be on the right track and need some fine tuning whereas if it gets ‘O’ and ‘X’ mixed up, perhaps the model should be scrapped in favor of a new one. Usually, this information gets lost to the engineer, though, with so many classes. On the plus side, a model which can classify any letter of the alphabet would have a lot more flexibility in its applications and would be easier to drop into a practical scenario like sorting mail, in which any letter may appear.

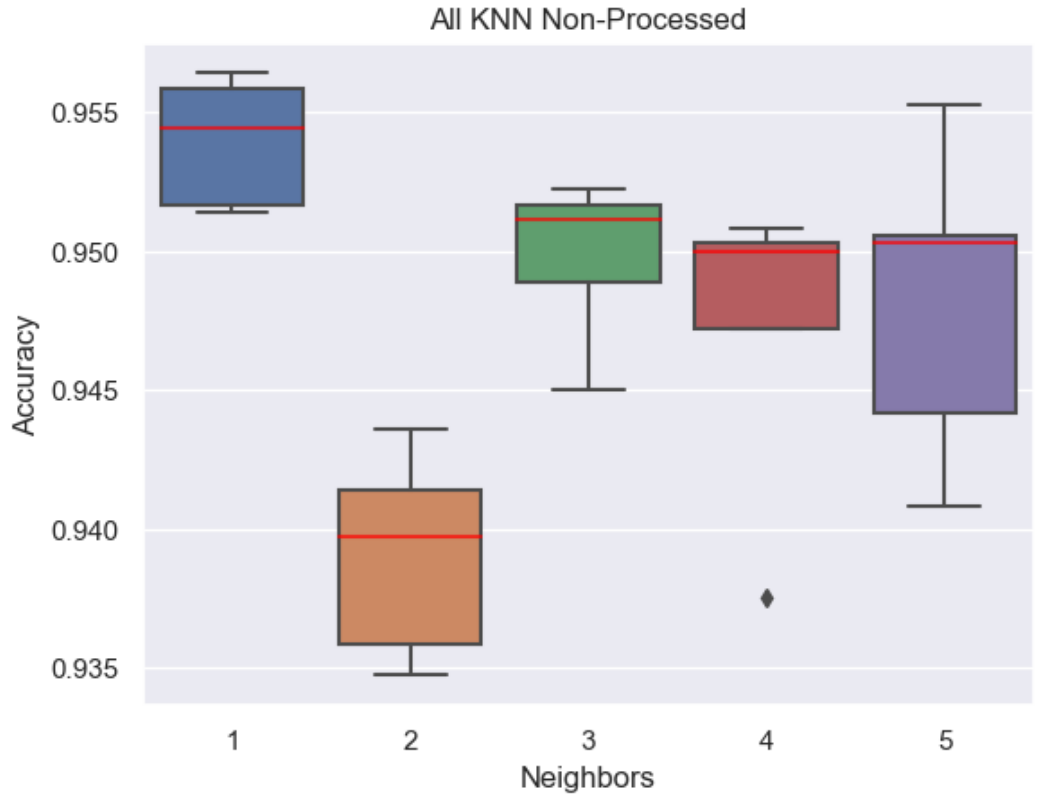
Results

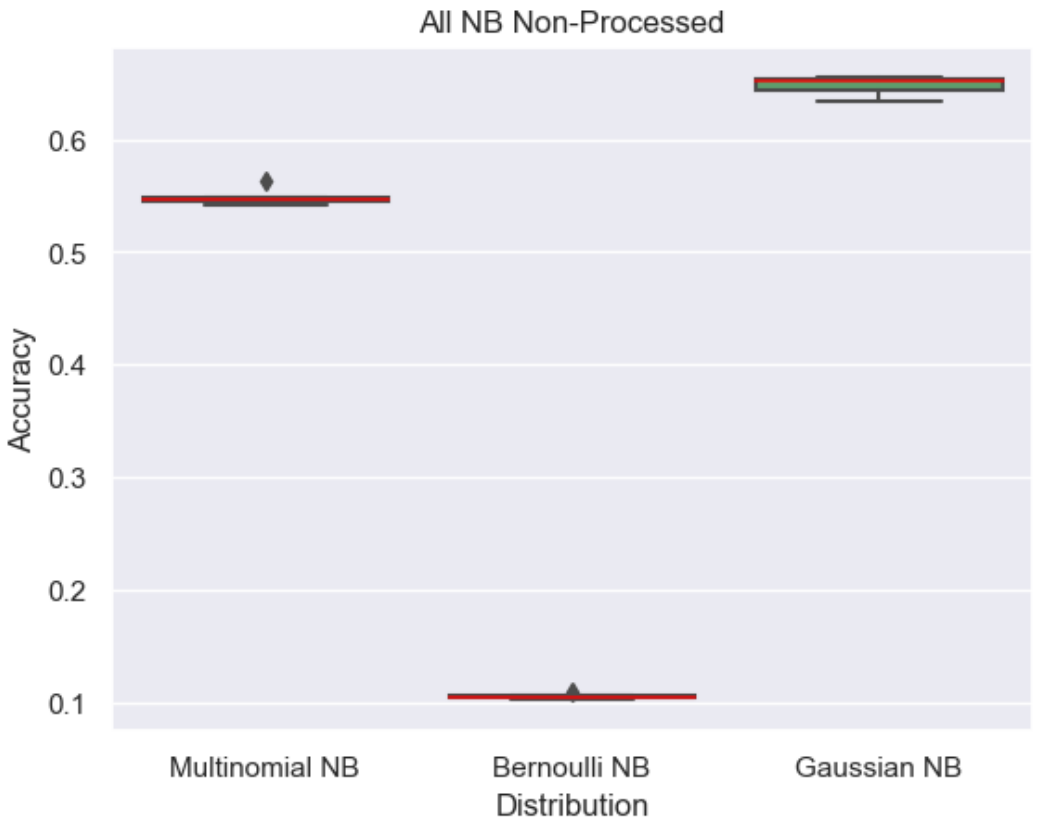
The classifiers chosen have various advantages and disadvantages. Naive Bayes is simple, usually fast, and generally well-suited to classification including multi-class problems. It’s not sensitive to irrelevant features, but it is sensitive to outliers and can be biased toward features with high frequency or an imbalanced majority data class. It favors categorical data but can handle continuous or discrete, although it requires discretizing continuous data which may cause information loss. This model includes an underlying assumption that features are independent and therefore doesn’t capture relationships between them. Additionally, it often needs smoothing and can be limited by the initial parameters (probability tables). SVM, on the other hand, can have long training times due to its large memory requirement; only linear SVM is considered parametric. It is effective in high dimensional spaces but needs clear separation between classes and can’t deal well with noise or overlap. Additionally, while it is stable and has less risk of overfitting, it usually requires the critical choice of kernel. KNN is another simple algorithm good for classification. No training is needed for it but this is tied to its non-parametric nature; this makes it bad with large datasets since it requires keeping all the data. Similarly, it is not good with high dimensionality, is sensitive to noise and missing values, is not good with imbalanced data, and requires choice of k and a distance metric. When it comes to hyperparameters for these models, the choices can be quite impactful. For SVM, a few kernel choices are linear for when the data is linearly separable without transformation, polynomial to try to achieve separability through addition of extra dimensions to the data, and RBF (Gaussian) to try to create similarity features via data landmarks. For Naive Bayes, we have the choice of distribution, Bernoulli for binary or boolean features, Multinomial for discrete values representing frequencies, and Gaussian for normally distributed continuous values. For KNN, two important hyperparameters are the number of neighbors to consider and the distance or similarity metric for assessing which data points are relevant to a prediction, e.g hamming, manhattan, euclidean, cosine similarity, etc.

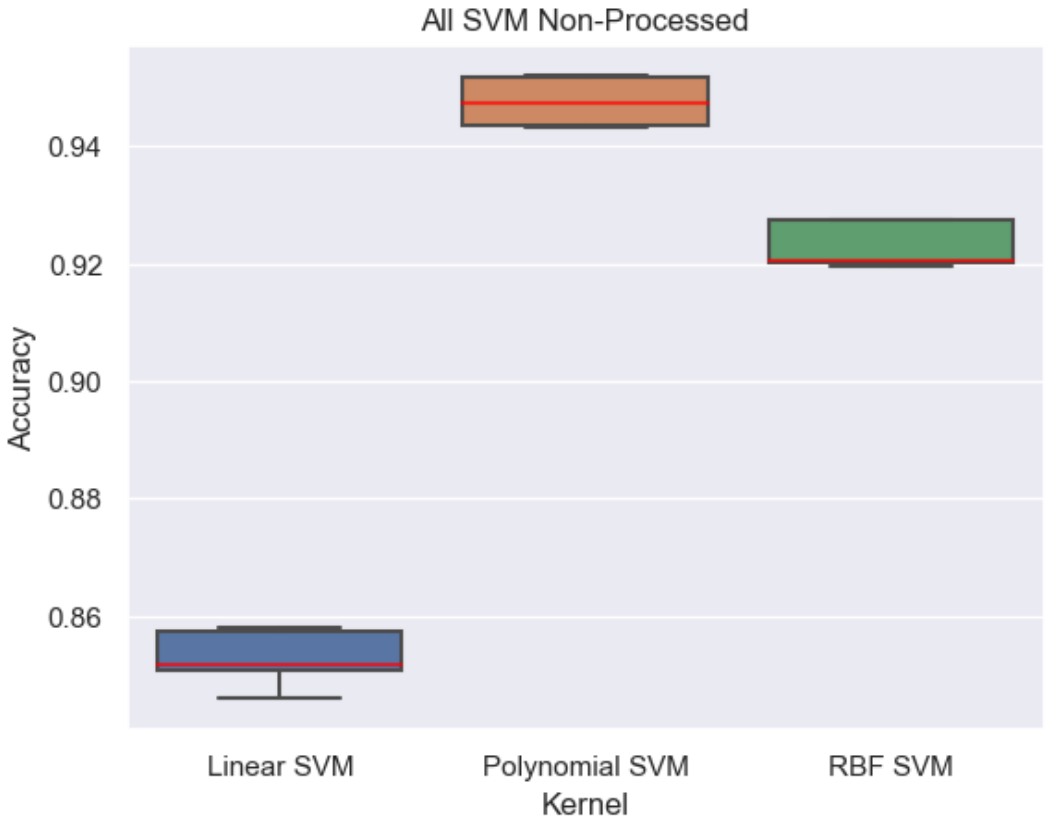
Hyperparameter Cross-Validation: KNN, SVM, Naive Bayes

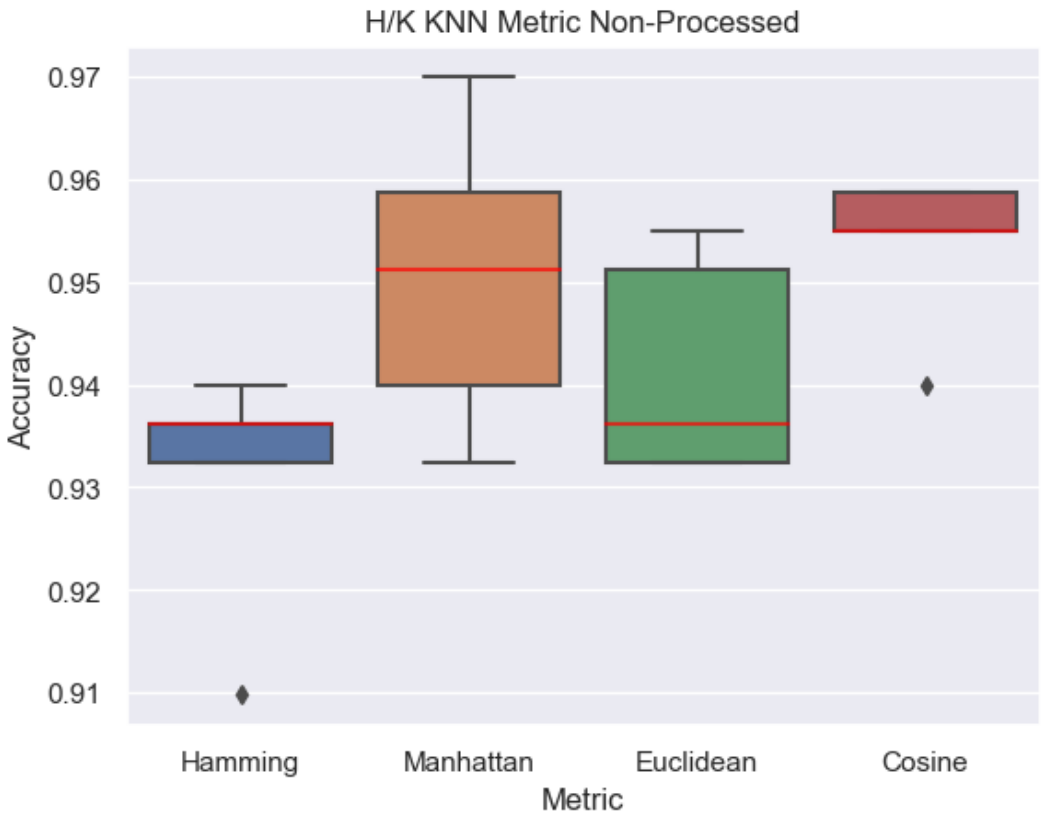
Non-Processed: All Classes, H/K Pair, O/D Pair, Y/M Pair

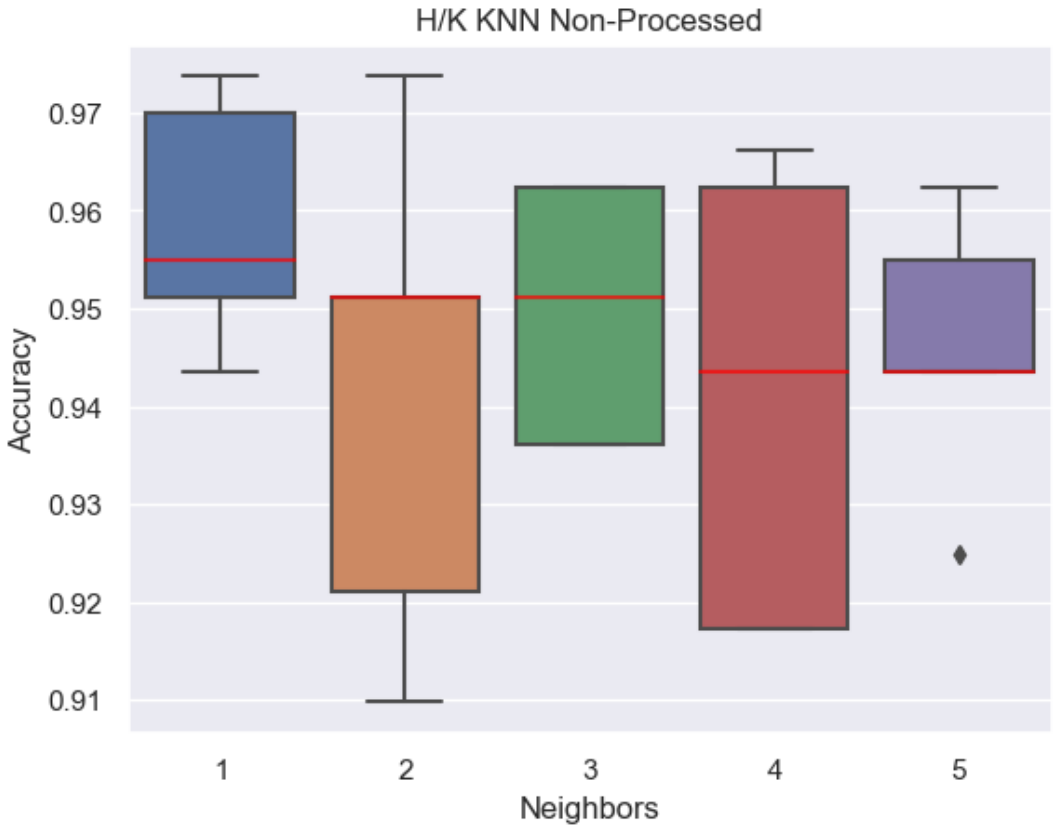


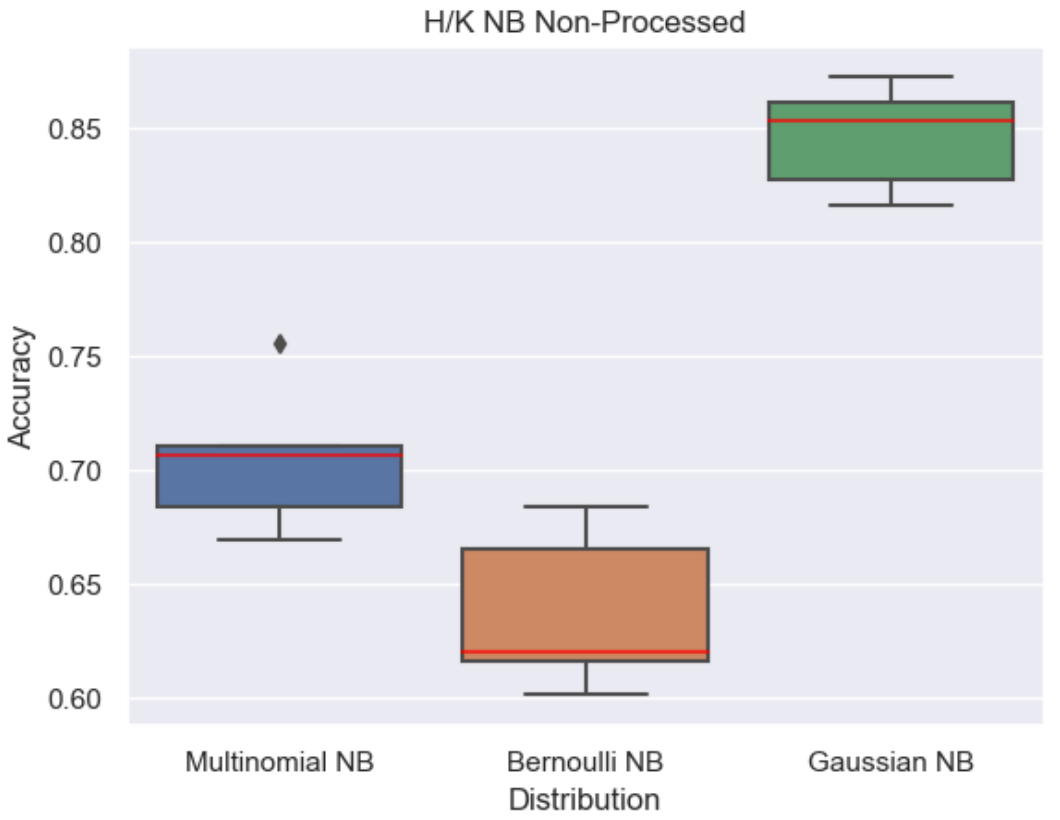


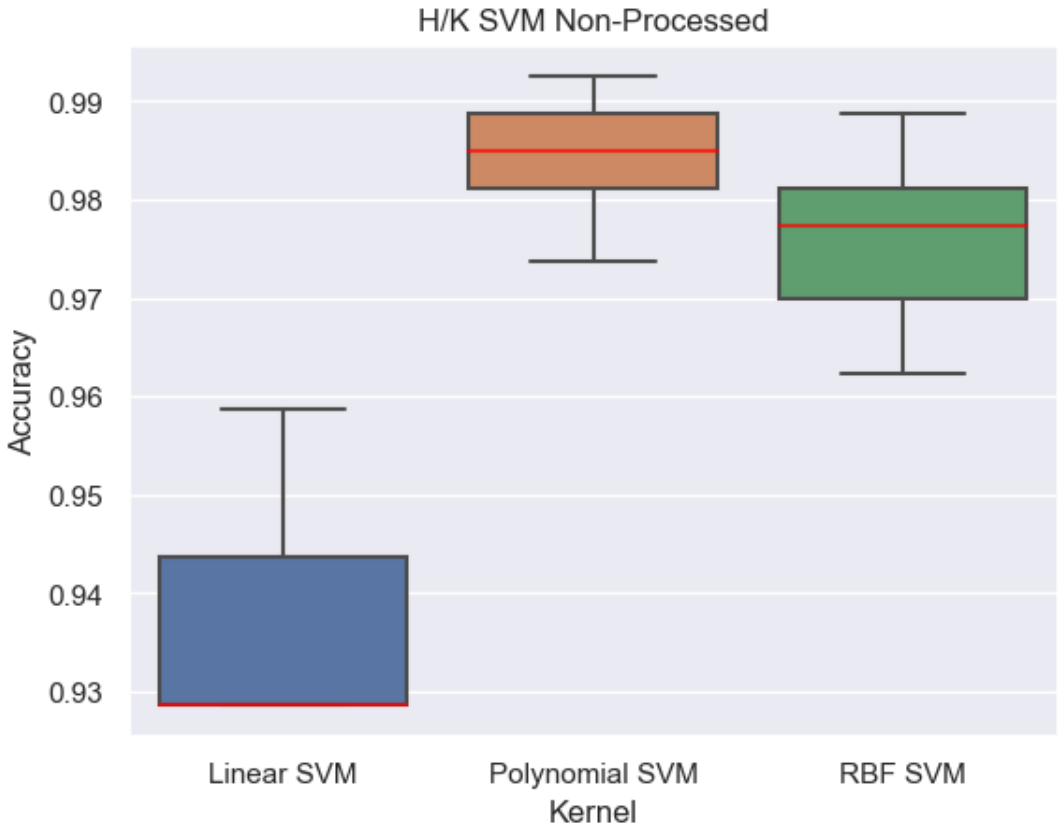

All NB Non-Processed
Accuracy
0.6
0.5
0.4
0.3
0.2
0.1
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution

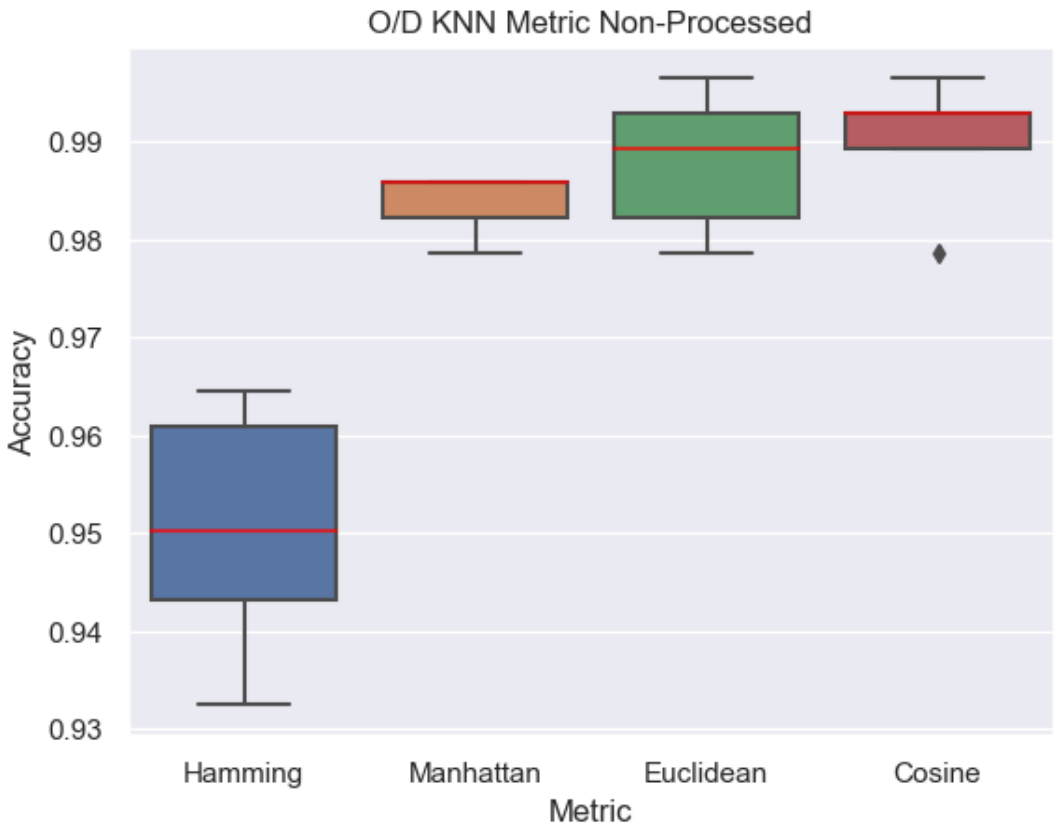

All SVM Non-Processed
Accuracy
0.94
0.92
0.90
0.88
0.86
Linear SVM
Polynomial SVM
RBF SVM
Kernel

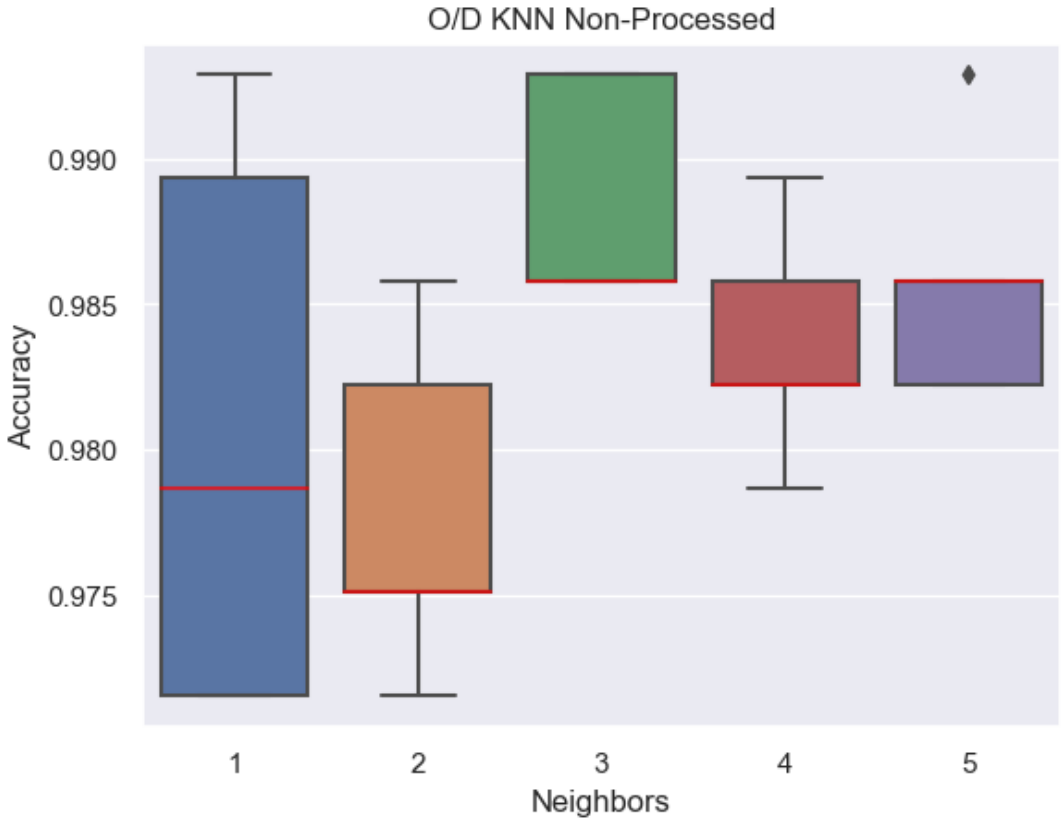

H/K KNN Metric Non-Processed
0.97
0.96
0.95
0.94
0.93
0.92
0.91
Accuracy
Hamming
Manhattan
Euclidean
Cosine
Metric

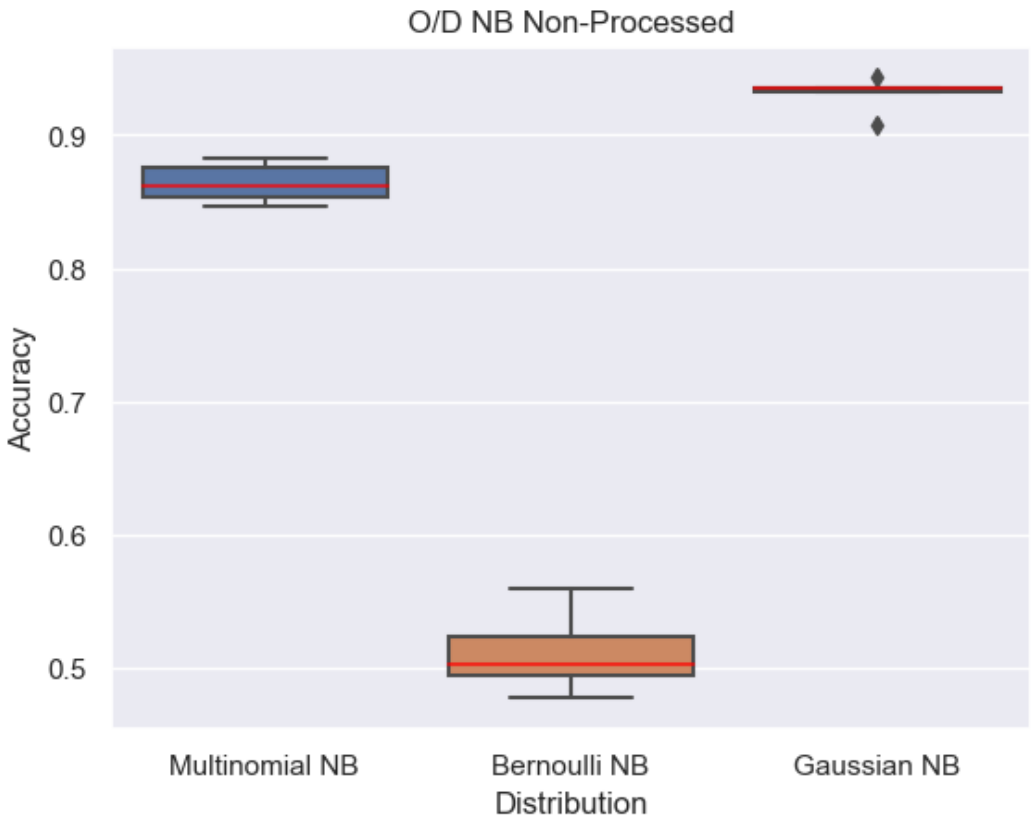

H/K KNN Non-Processed
0.97
0.96
0.95
0.94
0.93
0.92
0.91
Accuracy
1
2
3
4
5
Neighbors

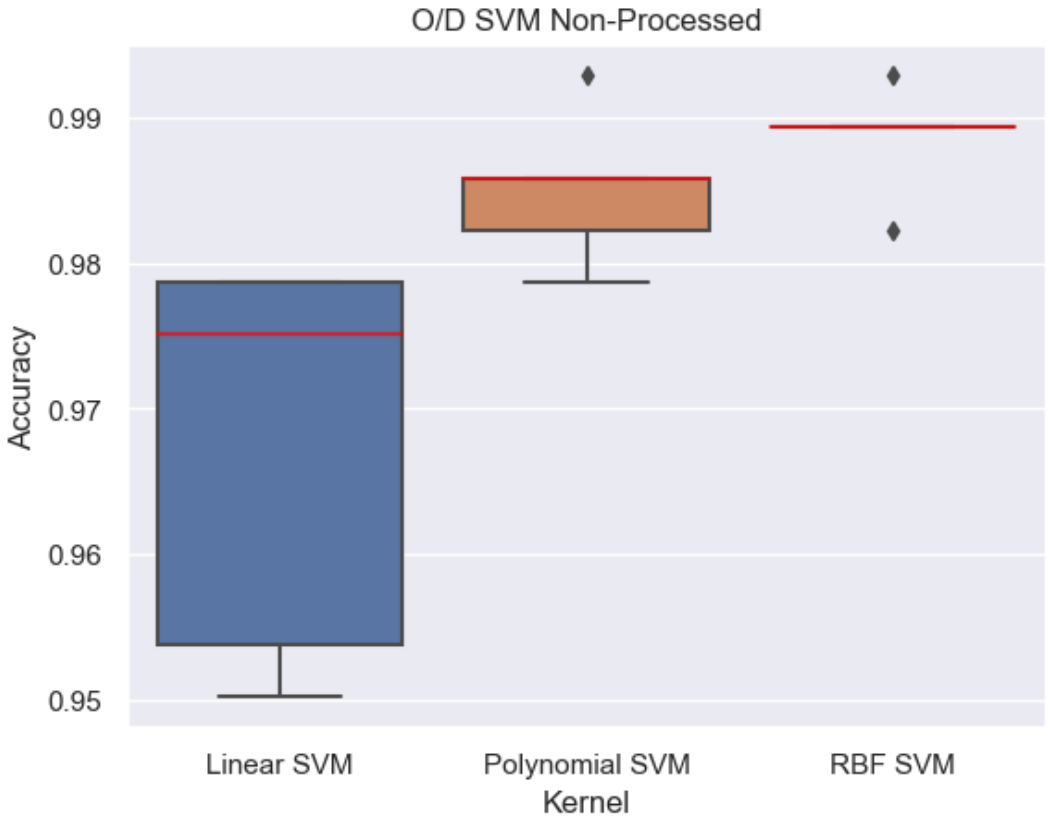

H/K NB Non-Processed
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution

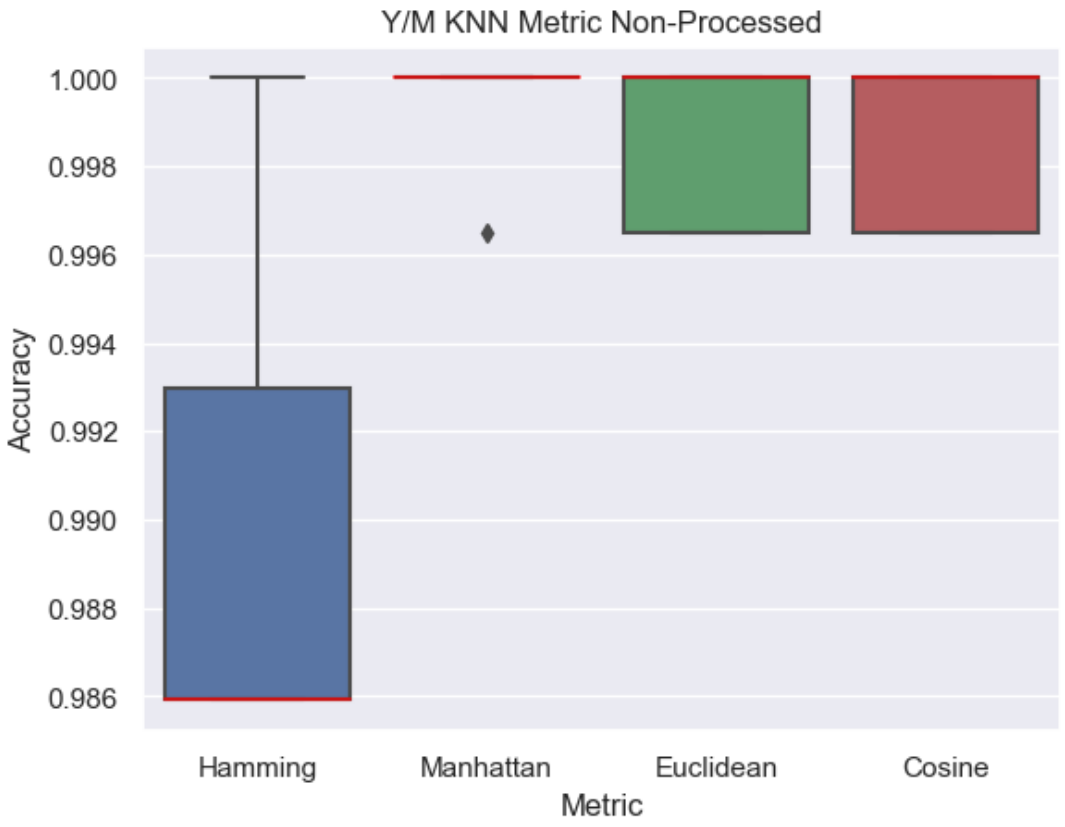

H/K SVM Non-Processed
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
Linear SVM
Polynomial SVM
RBF SVM
Kernel

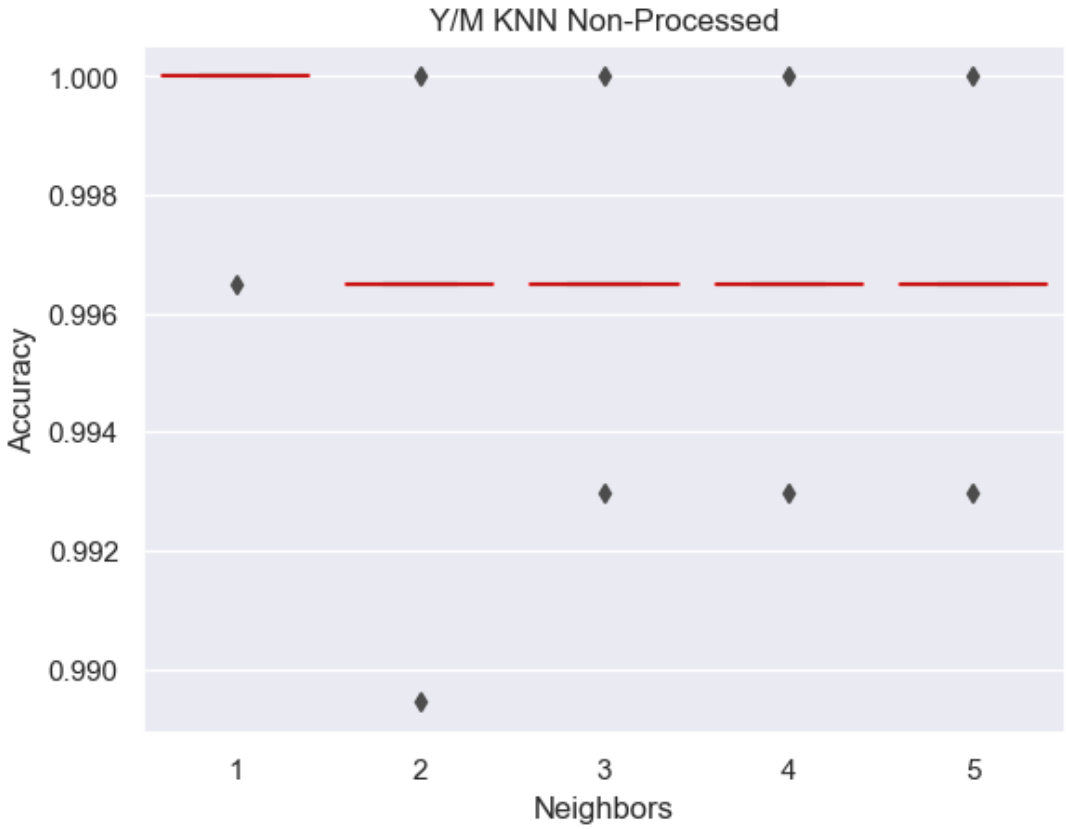

O/D KNN Metric Non-Processed
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
Hamming
Manhattan
Euclidean
Cosine
Metric

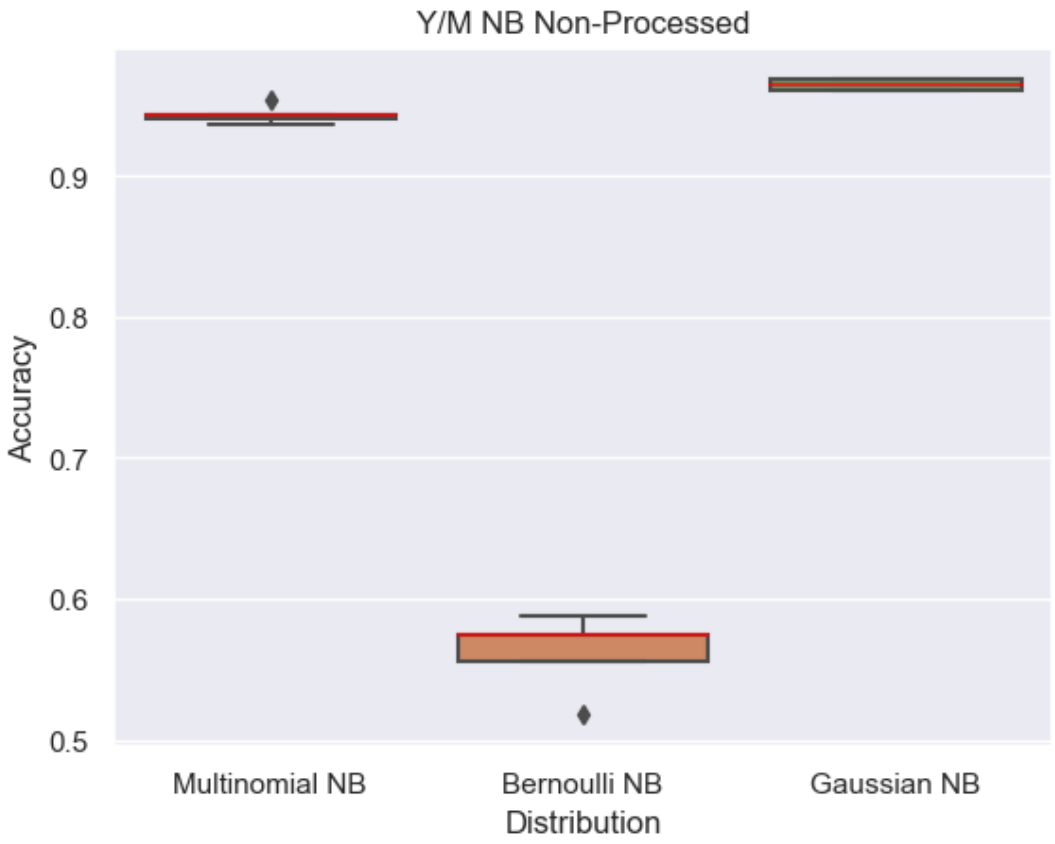

O/D KNN Non-Processed
Accuracy
0.990
0.985
0.980
0.975
1
2
3
4
5
Neighbors

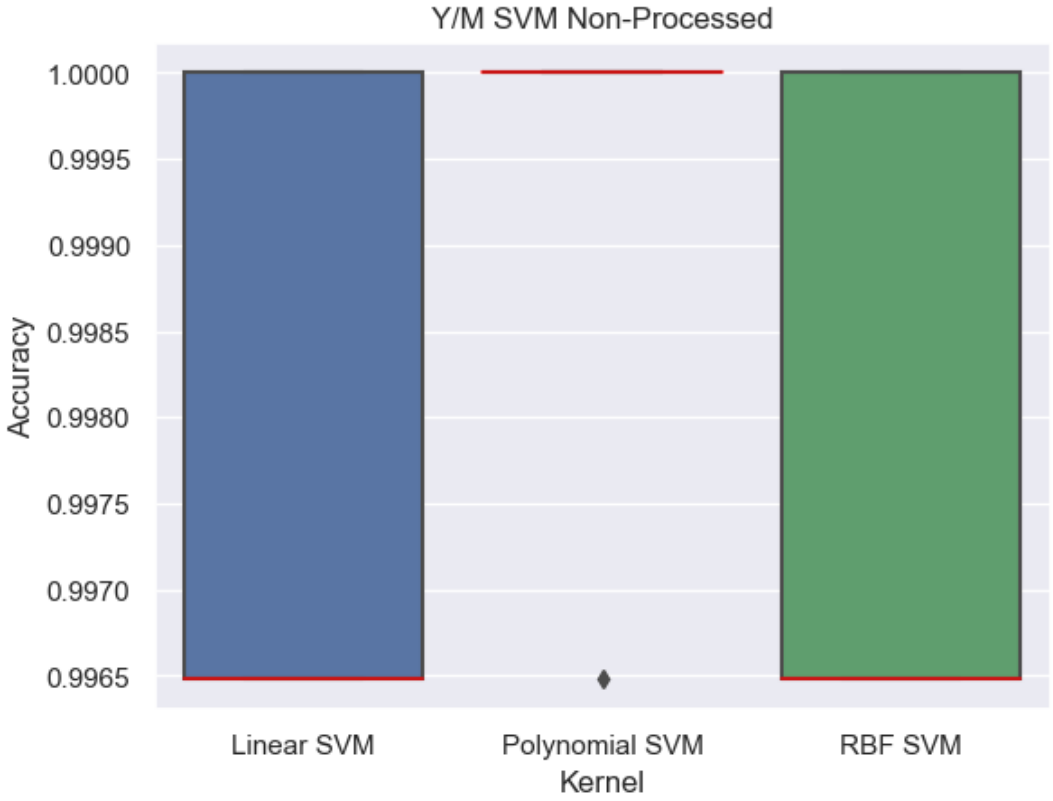

O/D NB Non-Processed
Accuracy
0.9
0.8
0.7
0.6
0.5
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution


O/D SVM Non-Processed
Accuracy
0.99
0.98
0.97
0.96
0.95
Linear SVM
Polynomial SVM
RBF SVM
Kernel

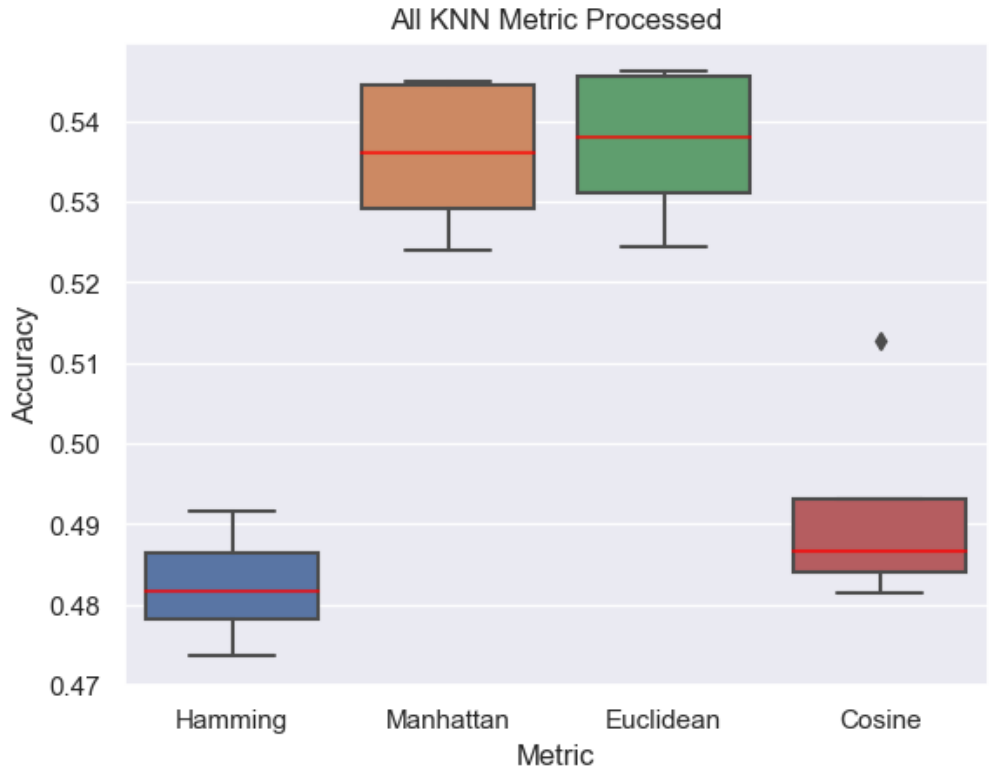

Y/M KNN Metric Non-Processed
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986
Accuracy
Hamming
Manhattan
Euclidean
Cosine
Metric

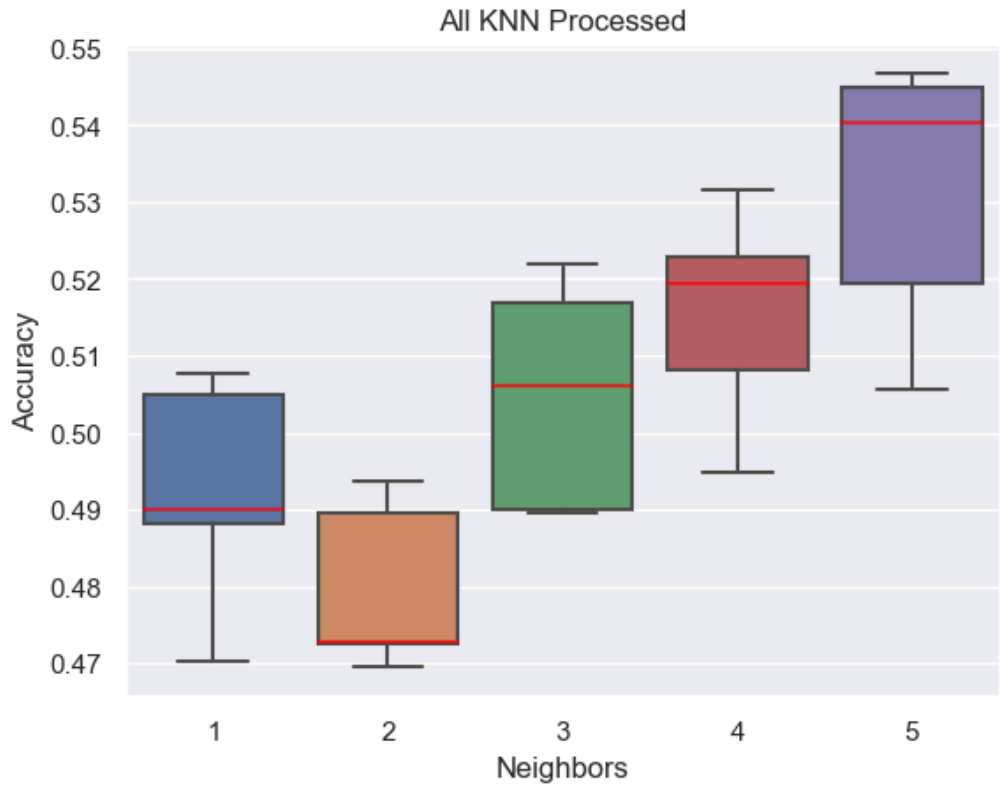

Y/M KNN Non-Processed
1.000
0.998
0.996
0.994
0.992
0.990
Accuracy
1
2
3
4
5
Neighbors

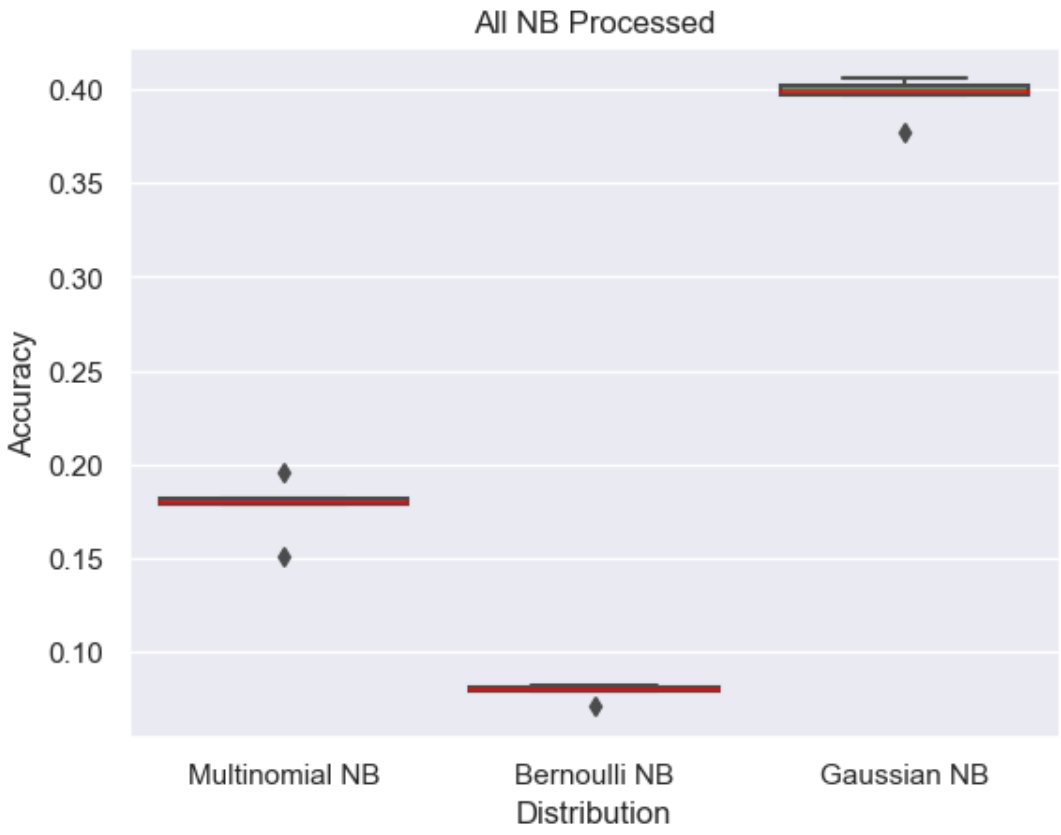

Y/M NB Non-Processed
Accuracy
0.9
0.8
0.7
0.6
0.5
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution

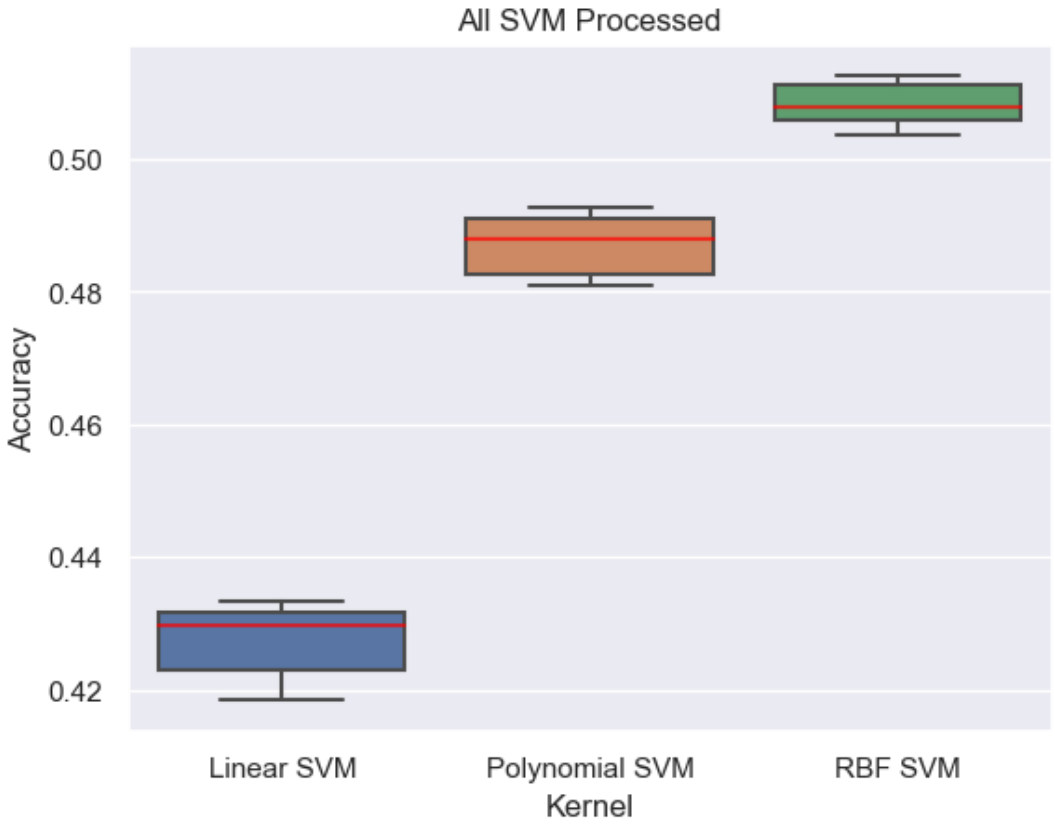

Y/M SVM Non-Processed
Accuracy
1.0000
0.9995
0.9990
0.9985
0.9980
0.9975
0.9970
0.9965
Linear SVM
Polynomial SVM
RBF SVM
Kernel

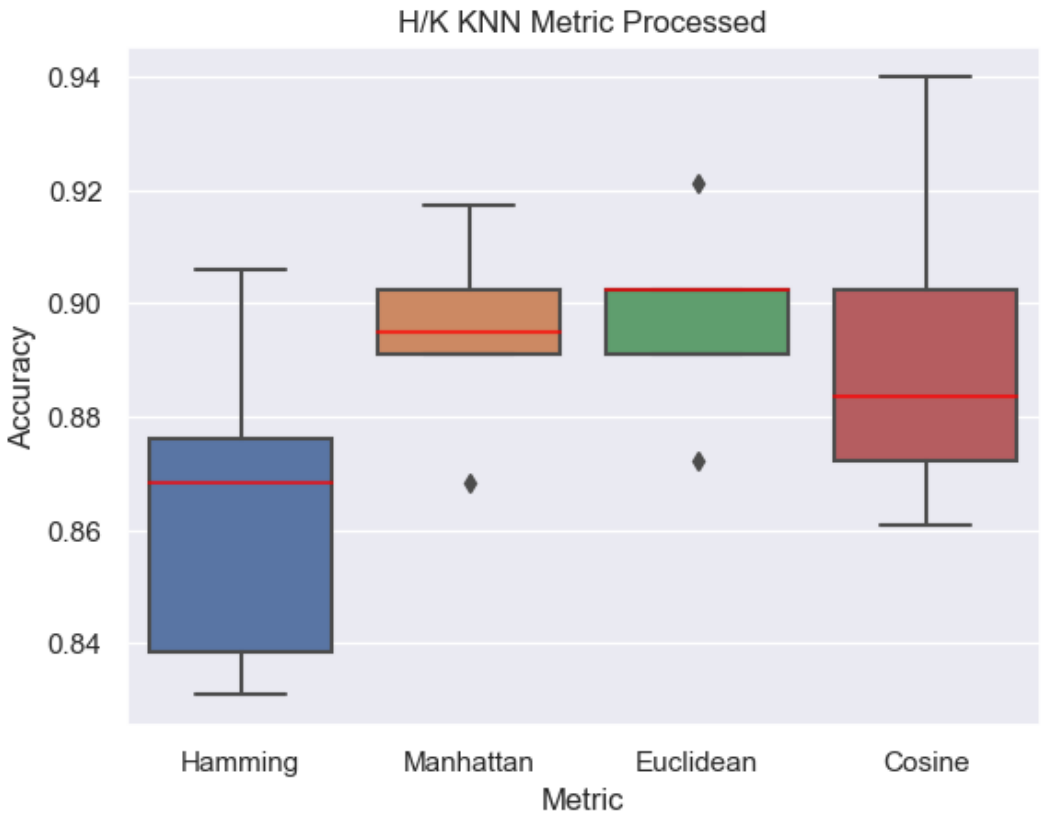
Preprocessed: All Classes, H/K Pair, O/D Pair, Y/M Pair

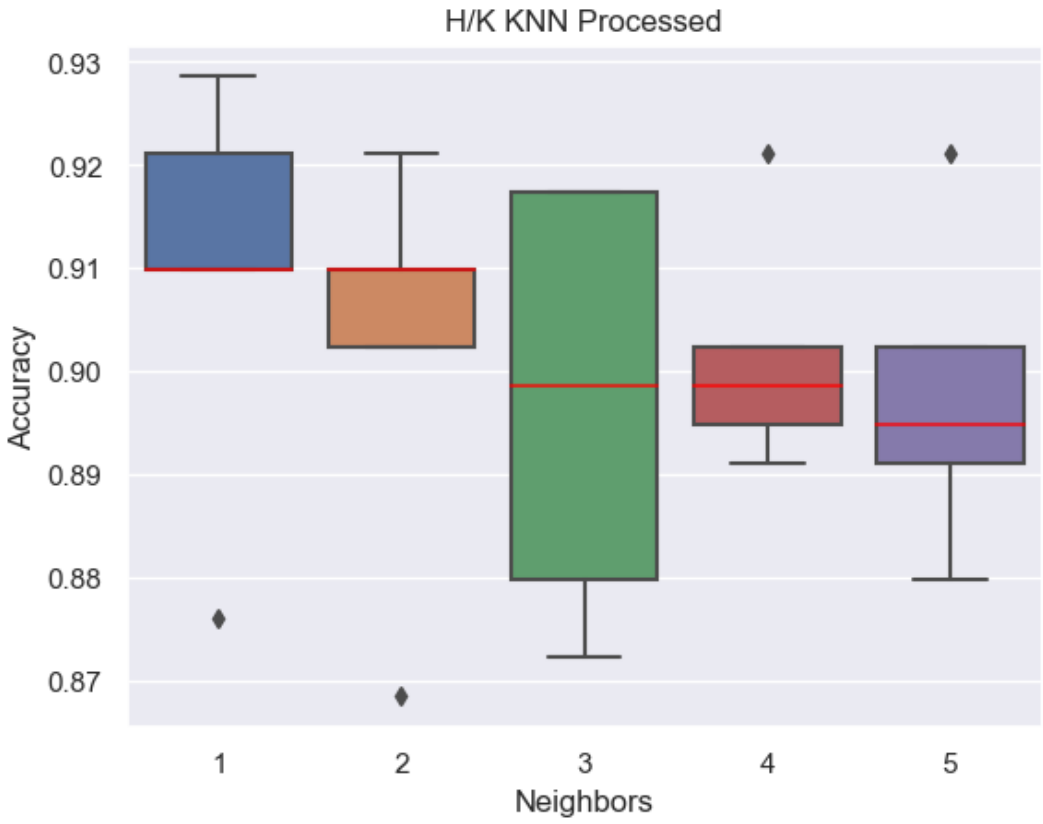


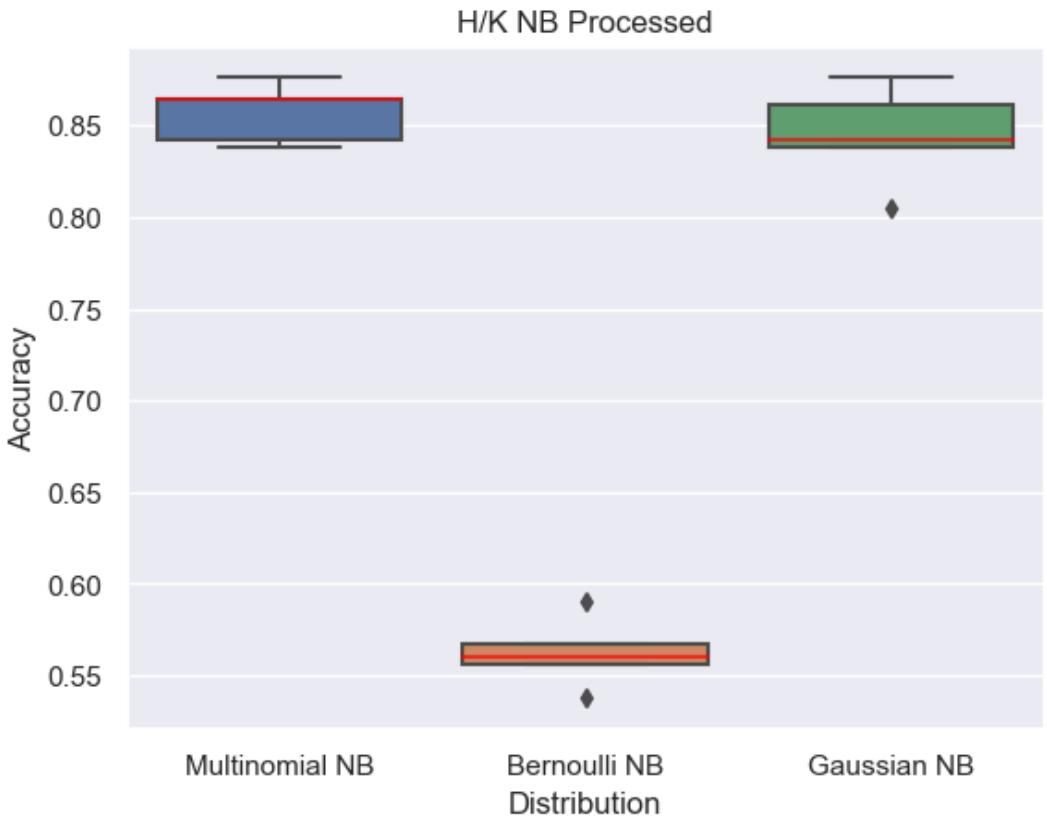


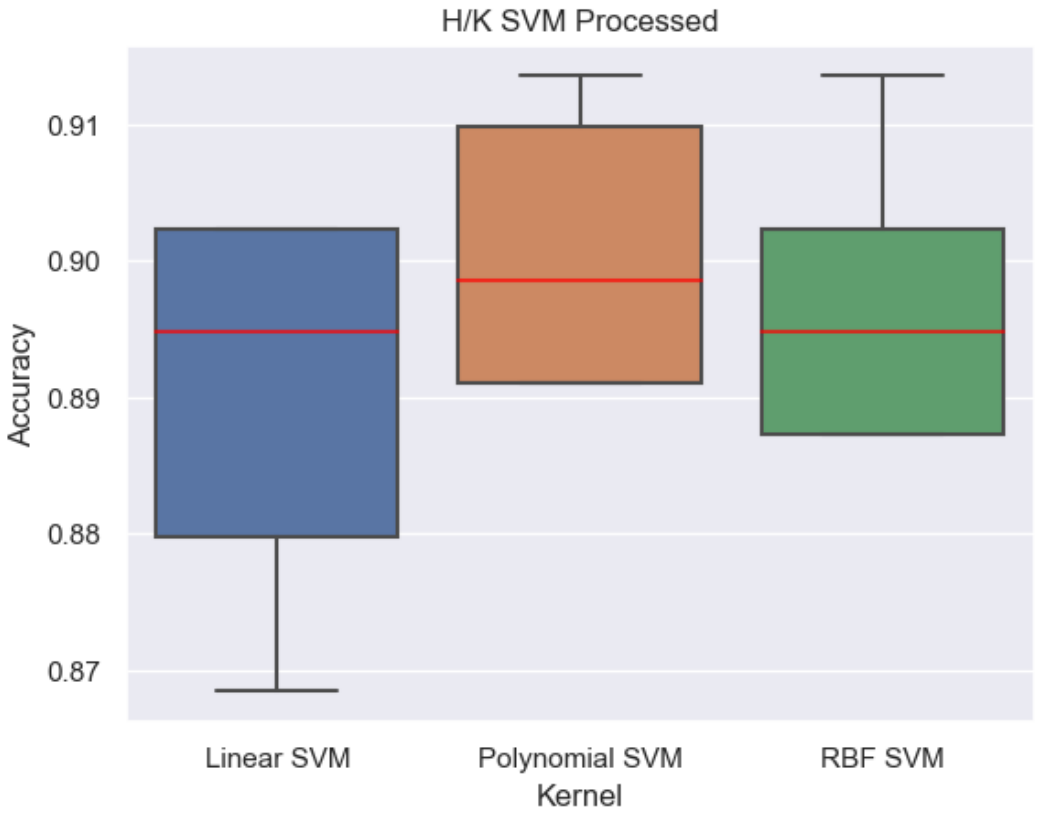

All NB Processed
Accuracy
0.40
0.35
0.30
0.25
0.20
0.15
0.10
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution

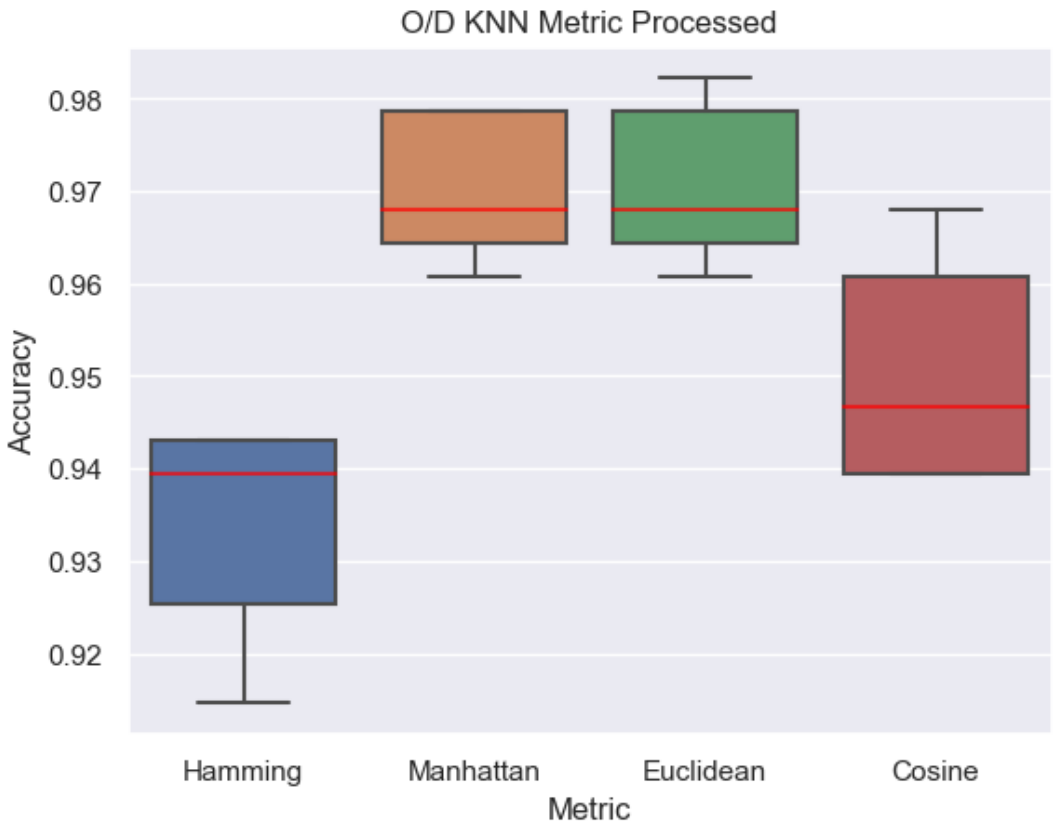

All SVM Processed
Accuracy
0.50
0.48
0.46
0.44
0.42
Linear SVM
Polynomial SVM
RBF SVM
Kernel

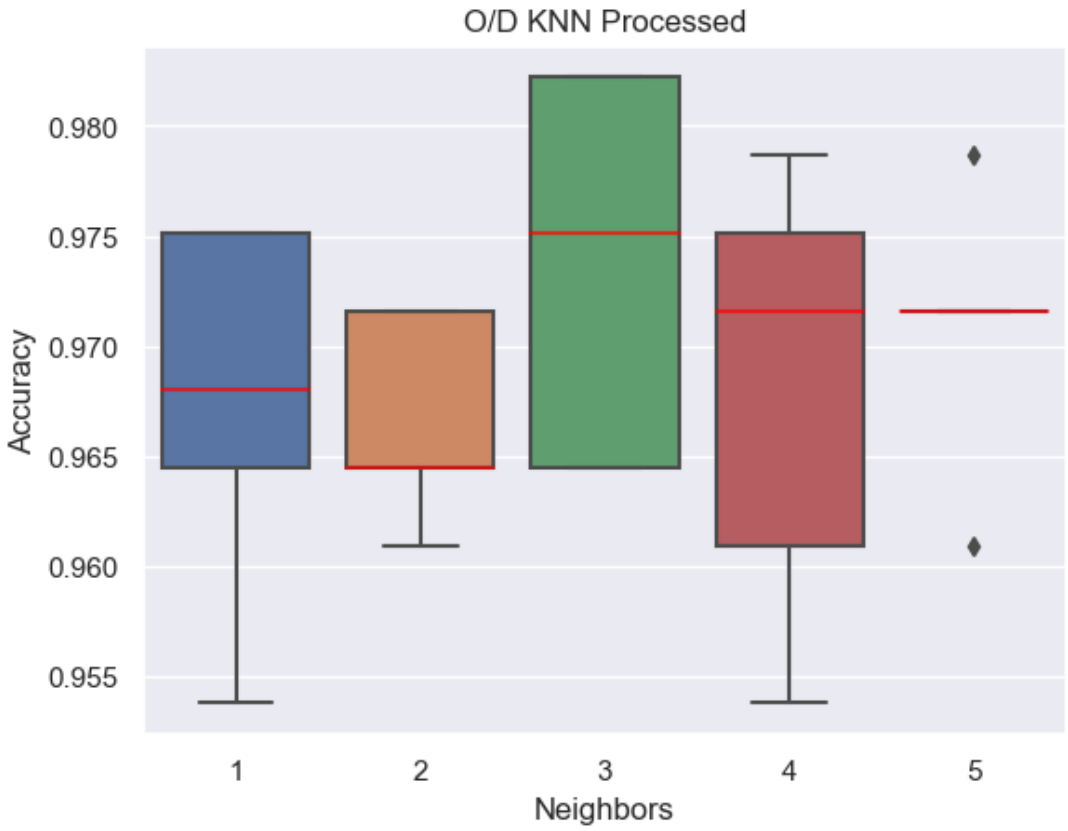

H/K KNN Metric Processed
0.94
0.92
0.90
0.88
0.86
0.84
Accuracy
Hamming
Manhattan
Euclidean
Cosine
Metric

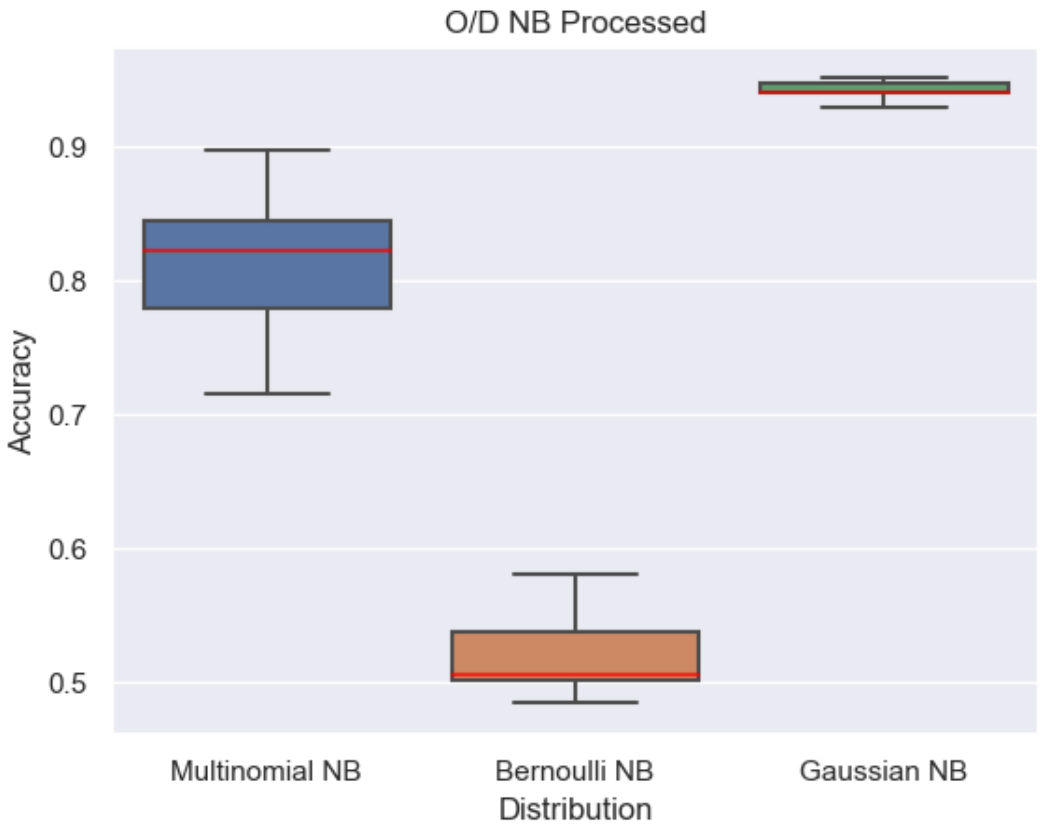

H/K KNN Processed
0.93
0.92
0.91
0.90
0.89
0.88
0.87
Accuracy
1
2
3
4
5
Neighbors

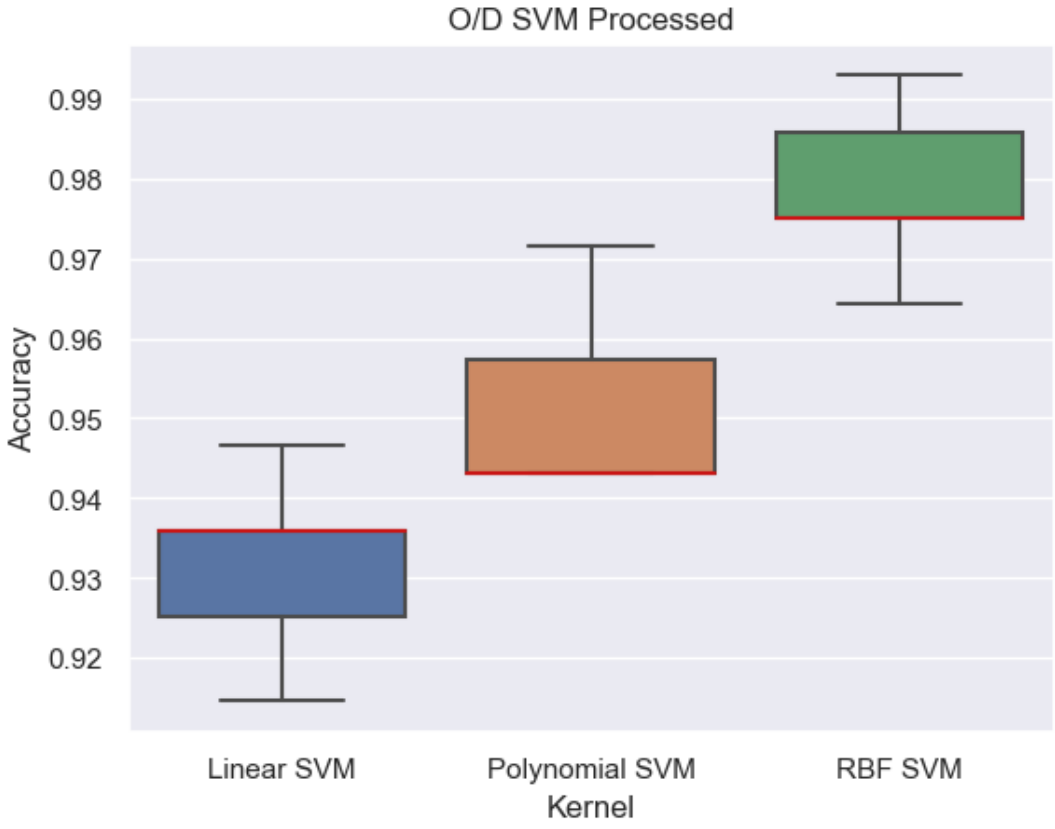

H/K NB Processed
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution

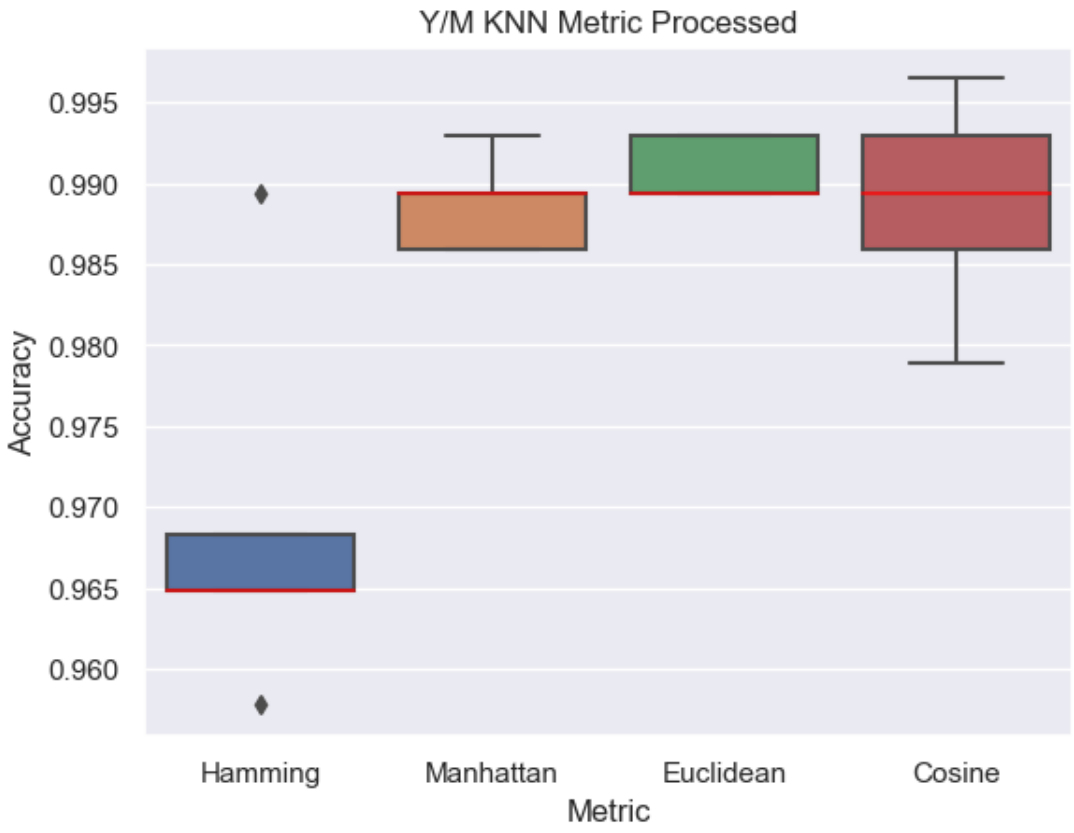

H/K SVM Processed
Accuracy
0.91
0.90
0.89
0.88
0.87
Linear SVM
Polynomial SVM
RBF SVM
Kernel

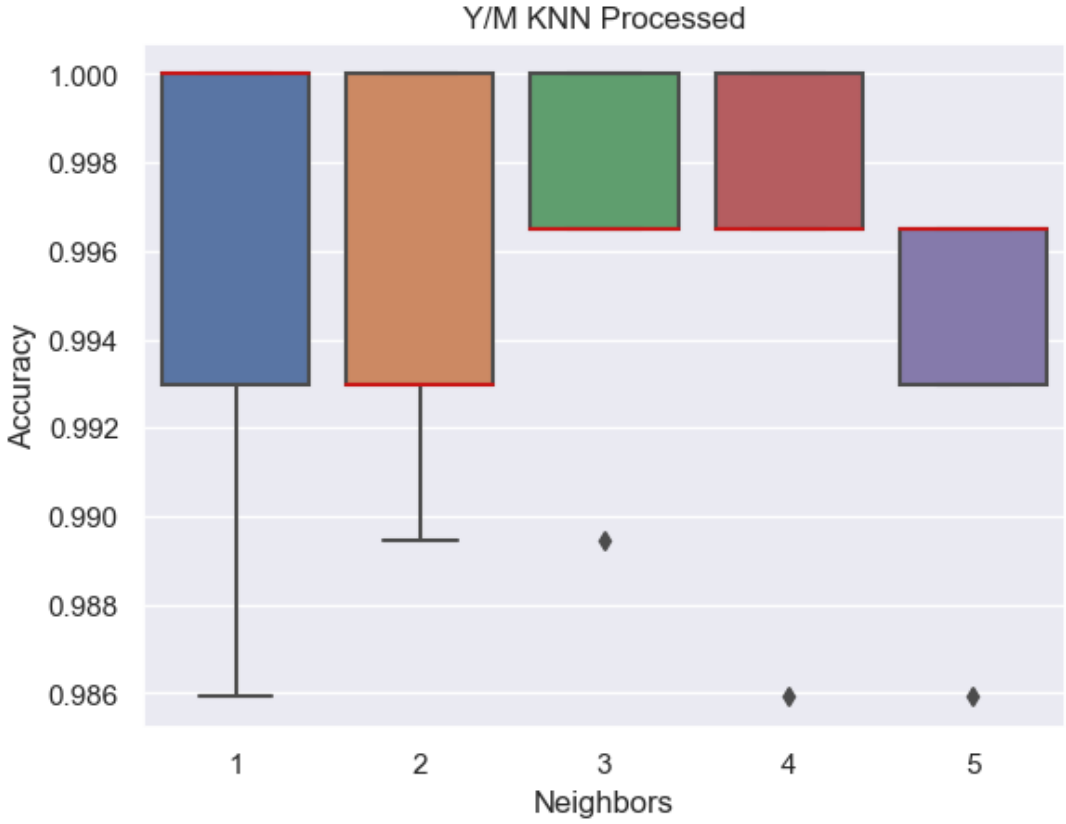

O/D KNN Metric Processed
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Accuracy
Hamming
Manhattan
Euclidean
Cosine
Metric

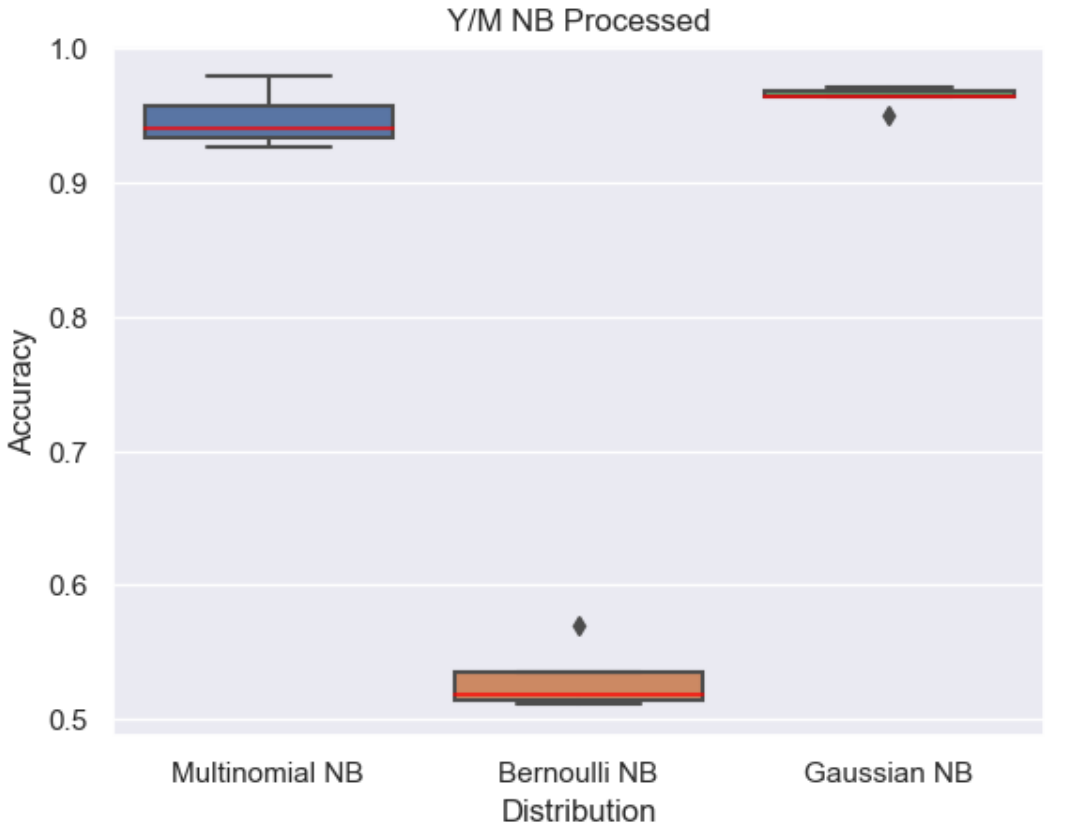

O/D KNN Processed
0.980
0.975
0.970
0.965
0.960
0.955
Accuracy
1
2
3
4
5
Neighbors

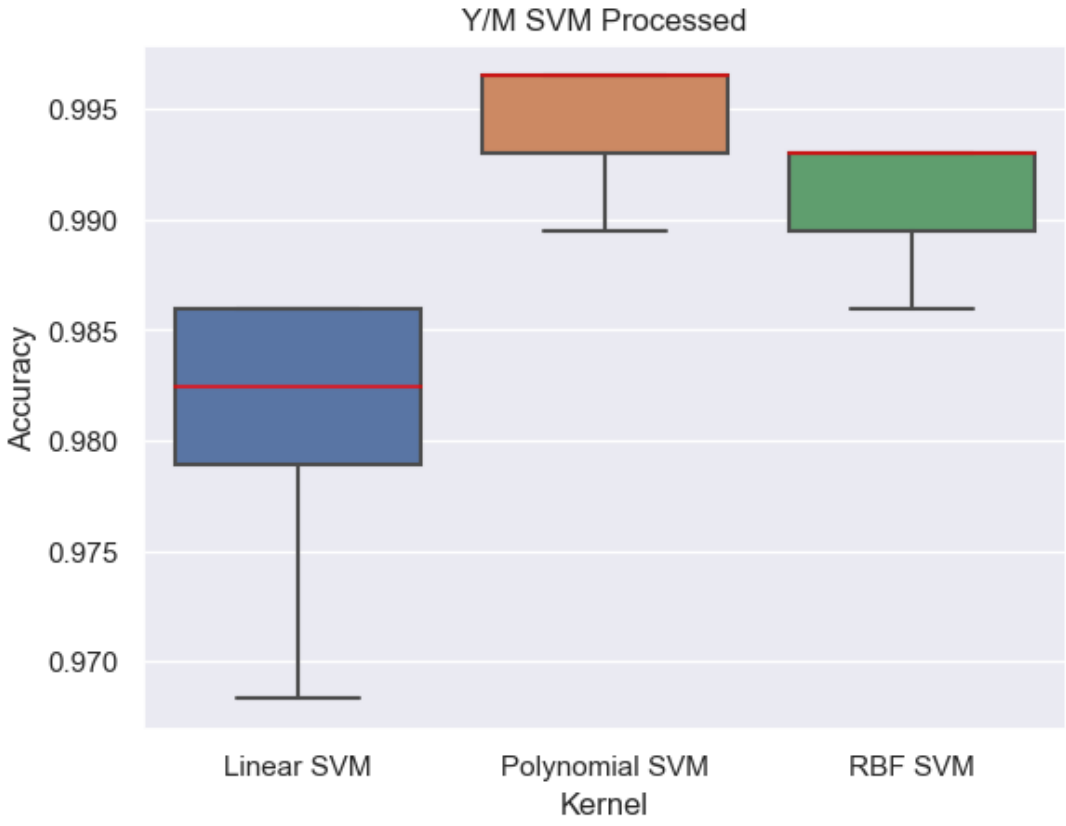

O/D NB Processed
Accuracy
0.9
0.8
0.7
0.6
0.5
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution


O/D SVM Processed
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Linear SVM
Polynomial SVM
RBF SVM
Kernel


Y/M KNN Metric Processed
Accuracy
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0.960
Hamming
Manhattan
Euclidean
Cosine
Metric


Y/M KNN Processed
Accuracy
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986
1
2
3
4
5
Neighbors


Y/M NB Processed
1.0
0.9
0.8
0.7
0.6
0.5
Accuracy
Multinomial NB
Bernoulli NB
Gaussian NB
Distribution


Y/M SVM Processed
0.995
0.990
0.985
0.980
0.975
0.970
Accuracy
Linear SVM
Polynomial SVM
RBF SVM
Kernel

Discussion

Non-Processed Final Models: All Classes, H/K Pair, O/D Pair, Y/M Pair

Naive Bayes	Distribution	Accuracy	Fit Time
H/K	Gaussian	0.8231292517006803	0.0023999999999997357
Y/M	Gaussian	0.9810126582278481	0.002538999999999625
O/D	Gaussian	0.8910256410256411	0.0033399999999996766
All	Gaussian	0.647	0.043287000000001186

SVM	Kernel	Accuracy	Fit Time
H/K	Polynomial	0.9795918367346939	0.0269200000000005
Y/M	Polynomial	1.0	0.00741299999999967
O/D	RBF	0.9871794871794872	0.019095000000000084
All	Polynomial	0.9475	3.2980610000000183

KNN	Neighbors	Metric	Accuracy	Fit Time
H/K	1	Cosine	0.9591836734693877	0.008628999999984899
Y/M	1	Manhattan	1.0	0.008582000000018297
O/D	3	Cosine	0.9807692307692307	0.010252000000008366
All	1	Cosine	0.9555	0.028114999999957035

Preprocessed Final Models: All Classes, H/K Pair, O/D Pair, Y/M Pair

Naive Bayes	Distribution	Accuracy	Fit Time
H/K	Multinomial	0.8299319727891157	0.0031309999999393767
Y/M	Gaussian	0.9746835443037974	0.0017169999999850916
O/D	Gaussian	0.9230769230769231	0.001790000000028158
All	Gaussian	0.393	0.1157780000000912

SVM	Kernel	Accuracy	Fit Time
H/K	Polynomial	0.8843537414965986	0.039533000000005813
Y/M	Polynomial	0.9936708860759493	0.012280999999916276
O/D	RBF	0.9743589743589743	0.01109499999995478
All	RBF	0.506	7.355756000000156

KNN	Neighbors	Metric	Accuracy	Fit Time
H/K	1	Euclidean	0.9115646258503401	0.0014499999999770807
Y/M	3	Euclidean	0.9936708860759493	0.0017380000001594453
O/D	3	Euclidean	0.967948717948718	0.0016879999998309358
All	5	Euclidean	0.5465	0.02214600000002065

Interestingly, the only models that did reasonably well for the multi-class classifiers, which included all the data for all 26 letters, were the non-preprocessed SVM and the non-preprocessed KNN. This implies that the dimensionality reduction technique used was not well suited to the multi-class task, and perhaps also we see the Naive Bayesian preference for categorical features at work in bringing down the accuracy; the features are all integers. It's possible the KNN multi-class model accuracy could have come up higher if it were tuned with higher values of k. Unsurprisingly, the fit time for the multi-class model was significantly higher than for all other models, and this was seen especially strongly with SVM, again as expected. While both fast, KNN was somewhat faster than Naive Bayes for the multi-class models, particularly for the preprocessed data; this is likely due to most of the value of KNN being provided at fitting time with basically no training required. For the pair-wise models, Naive Bayes was generally slightly faster than KNN for the non-preprocessed data and about the same fit time for the processed data. Curiously for the pair-wise models, the H/K pair appears unilaterally more difficult than the O/D pair, which I did not anticipate. As might have been presumed, though, Y/M was the easiest of the three pairs. The accuracies for Naive Bayes with and without dimension reduction were mostly comparable to each other, with results slightly improving for H/K and O/D with dimension reduction. For the other pair-wise models accuracies were also mostly comparable to each other although preprocessed results were slightly lower, with the biggest hit taken in H/K detection for both SVM and KNN. This is another testament to the difficulty and sensitivity involved in classifying the H/K pairing. Overall, KNN and SVM both did exceptionally well.

For the KNN distance metrics, I am surprised how poorly Hamming distance performed considering all features are integers. I am also surprised at the tuning preference for 1 neighbor and cosine similarity in the non-preprocessed data; perhaps these preferences are in part due

to noise in the data which is later removed. In this experiment, preference for the number of neighbors was found first using the default Euclidean distance, and then this number of neighbors was fed into another round of tuning using the various distance/similarity metrics. If I were to do the experiment over again, I would probably do a full grid search with these hyperparameters to ensure no bias is introduced into the selection. For the Naive Bayes distribution, it is unsurprising that Bernoulli overwhelmingly performed the most poorly with accuracies well below 60% for all except the H/K non-preprocessed pairing which was just over. To use a Bernoulli distribution on non-binary features, they must be discretized which, as discussed in the introduction, creates information loss. Multinomial and Gaussian performed mostly comparably to each other with the exception of multi-class, in which Gaussian always performed better, and H/K non-processed and O/D processed, in which Gaussian also performed better. Perhaps the preprocessing made the features for H/K seem more like frequencies and the features for O/D less like frequencies. While the features are integers, it can be seen from some of their descriptions that there may be overlap with the frequency domain, e.g. with some features representing edge counts and number of pixels. It's interesting to see this notion reflected in the results. For SVM, polynomial and RBF kernels were always selected over a linear kernel, however all kernels still resulted in above 85% accuracy with the exception of the preprocessed multi-class SVM model. Noticeably RBF was always chosen for the O/D pairing while a polynomial kernel was always chosen for the H/K and Y/M pairings. Maybe this lends some credence to the initial notion that the curves of these letters make it more complex to classify between them.

If given this same task with a new dataset, one of the primary changes in approach next time would be to apply different dimensionality reduction techniques to each of the different models. While using a filter method based on mutual information made sense for pair-wise classification with Naive Bayes, it's possible KNN and SVM could have performed better with a different method. The multi-class model certainly needed a different reduction technique; for multi-class I would try PCA and either skip Naive Bayes altogether or limit it to a Gaussian model. I think KNN and SVM were good choices as is, and in fact they achieved perfect accuracy for the Y/M pairing in the non-preprocessed data. Which model I would choose would depend on the application. For this dataset, using the non-parametric models KNN and SVM worked well, but if the dataset grows, they could quickly become poor choices. If fit time were an issue, of the two, I would choose KNN, otherwise I would choose SVM for overall robustness and accuracy. With these, I would either elect not to do preprocessing and reduction or to investigate other options. If data storage is a potential issue and this was only needed for pair-wise classification tasks, I would select Gaussian Naive Bayes with the current preprocessing steps. However if data storage is an issue and multi-class prediction is needed, again other options should be investigated to achieve the best trade-offs.

References

[1] http://archive.ics.uci.edu/ml/datasets/letter+recognition
[2] https://github.com/danni-beaulieu/data-mining-hw-2/