GRA-KNN Algorithm

Siddharth Arya, Raiyan Raad, Vedant Goel

Link to implementation of all algorithms used for results:
https://github.com/sid19arya/GRA-KNN

1 Introduction

With the rampant increase in data being collected and made available to machine learning practitioners, the issue of parsing through redundant information and getting the important pieces of information has become more pressing. Additionally, as the dimensions of our data gets higher, computational costs explode as more information needs to be processed, but more alarmingly, it causes some algorithms to perform worse. This issue is commonly known as the Curse of Dimensionality [1], and it effects many algorithms in Machine Learning.

The K-nearest-neighbours algorithm is a powerful yet simple non-parametric classification model. It compares the Euclidean distance between a new data point and all existing data points, finding the one closest to the new data point, for predicting outcome labels. However when dealing with incredibly high-dimensional spaces, the Euclidean distance between any 2 points becomes somewhat constant [1]; the concept of distance becomes less meaningful as data points are very far apart from each other. As such, when dealing with high-dimensional spaces, the K-nearest-neighbours requires modification to perform well.

Given this background on the curse of dimensionality, and the KNN algorithm, this paper examines different existing methods to dealing with high dimensional data - in the context of the KNN algorithm - and proposes a new modification to this algorithm to tackle this curse.

2 Literature Review

To tackle the curse of dimensionality, we try to find a way to use fewer number of dimensions than the entire data set, and to use these fewer dimensions to still effectively classify the classes for new data points. At its core, all techniques follow these same steps, but they go about doing this in different ways.

2.1 PCA Dimension Reduction

A very common technique to reduce dimensions is Principle Component Features. The intuition behind this technique is to use the covariance of features to extract latent features that summarize the data, and then use these new features going forward. PCA is a provenly effective technique at reducing the features required to represent datapoints, while still keeping high classification accuracy, especially in image classification [2]. Although accompanied with other pre-processing strategies, the crux of feature extraction in this paper is done by the PCA algorithm.

Formally, we can discuss the PCA as the following: first we consider the training data matrix X. We can calculate the covariance matrix of X as follows:

$$Cov(\mathbf{X}) = E[(x-\mu)(x-\mu)^T] \tag{1}$$

As this matrix is symmetric, we know that it has a decomposition in terms of its eigenvalues and vectors:

$$Cov(\mathbf{X}) = QAQ^T \tag{2}$$

where the columns of Q are the eigenvectors of Cov(X) and values of the diagonal matrix A, are the corresponding eigenvalues. As per the spectral decomposition theorem, the eigenvectors of a symmetric matrix are an orthogonal basis for the space of the original matrix. Therefore, now we can represent vector in X, i.e any one of our training points using the new basis of eigen vectors.

To actually reduce dimensions, instead of representing our data points as a linear combination of **all** the eigen vectors, we are going to choose a few of them, specifically those few that correspond with the highest eigen values.

The idea behind this is that if we group feature that have a highest covariance (they occur together the most) and use those to summarize our data points using fewer features.

Highlighting this same process of 'extracting' latent features, the aforementioned paper classified images into plant types and performed incredibly accurately even with much fewer features than the original dataset. This is a common yet effective solution and should be kept in consideration when dealing with possible solutions to the curse of dimensionality.

2.2 Feature Selection by Information Gain

Instead of extracting latent feature and representing our data differently, we can also simple select the important features out the existing ones to lower dimensions. This can be implemented using Information Gain (IG), which is an entropy-based feature evaluation method that measures the relevant information about the target class provided by a feature.

The entropy of a variable X can be calculated by:

$$H(X) = -\sum_{i=1}^{K} P(x_i) \log_2 P(x_i)$$

The IG of a feature X with class labels Y can be computed using:

$$IG(X, Y) = H(X) - H(X|Y)$$
$$H(X|Y) = -\sum_{j} P(y_j) \sum_{i} P(x_i|y_j) \log_2 P(x_i|y_j)$$

Having gotten a measure for the 'importance' of features of a dataset relevant to the classes, we can have an algorithm that selects feature based on their importance. An example of such an algorithm is one that first computes the IG of each feature with respect to the target class and ranks them in descending order. Then it takes a threshold, selects the features for which the IG is more than the threshold, and adds them to an initially empty set of relevant features. The algorithm then compares the computed IGs of the features within the set of relevant features and eliminates the redundant features to produce the set of the most relevant least redundant features.[3]

Information gain based feature selection can reduce the dimensions effectively and improve the quality of the set of selected features.

2.3 Weighted-KNN Algorithm

Now given that we selected the important features, rather than just running KNN on the reduced features, we want to distinguish between the selected features in the order of their importance. Before we do this, we use a modification of the KNN algorithm named "Weighted KNN Algorithm" to motivate the incorporation of weights into the KNN algorithm .[4]

This algorithm's approach is similar to regular KNN, but it assigns a weight value to each of the "k" neighbors. By using an inverse ratio with respect to the distance for our weight formula, we can ensure the point that is closest to our query point within the "k" neighborhood will have a higher weight than a point that is farther away. A very commonly used weight formula is the inverse squared distance.

$$w_i = \frac{1}{(d(x' - x_i))^2}$$

In the above formula x' is our query point and x_i is a point from our "k" neighborhood. For each of these data points, the weight value represents the importance of its corresponding classification. After assigning a weight to each of the "k" points in our k-neighborhood, we will count the total weight for each class and output the label with the highest weight value. Notice that if our query point is the same as one of the data points, the output label will be that data point's label (this is because the weight for that class will be infinite). Although this approach of making some neighbour more important is different than making some features more important, we can still draw some ideas from this for our algorithm as it introduces weights to the regular KNN.

3 Problem and Method

The problem this paper centers on is the high computational cost and potentially poor performance of the KNN algorithm when faced with high dimensional data. Literature Review showed us the existing approaches to deal with this issue. Fewer dimensions have shown to be achieved through **feature extraction**, using **Principle Component Analysis**, or **feature selection** using **Information Gain**. However both of these still hold drawbacks that can be improved upon. Principle component Analysis creates new features that are no longer easily interpretable. On the other hand, while information gain selects existing features, which allows for interpretability, it takes these 'important features' and gives them the same weight in our algorithm. The existing modification of **Weighted KNN** does add some weights between different neighbours, but only between which points are closer or further as a whole. We find that in each of the methods, after getting a subset of 'important feature', they then forego the hierarchy of importance among those features and consider them equally again. The approach we propose is to capitalize on this hierarchy of features and use them to consider distances not between points as a whole, but between the values of features of those points. This brings us to the formal introduction of our proposed KNN variation called **GRA-KNN Algorithm** (Goel-Raad-Arya KNN).

3.1 GRA-KNN Algorithm

First, we define our training set as follows: $\mathbf{X} = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(N)}, t^{(N)})\}$. Where each $x^{(i)} \in \mathbb{R}^d$, and each $t^{(i)} \in L$, where L denotes the set of all the classes our points could belong to. Now suppose that we have a test vector x' that we would like to classify. We use the following steps.

(i) Get the 'most important' features using Information Gain. Using the same method to calculate information gain as above, we simply select d' features with the highest information gain as our selected features. We take our data and modify it, such that it is sorted based on the most important features. We will perform the same modification on any test point.

(ii) Iterate through these features and gather 'votes' towards what the test data point's label should be.

Similar to Vanilla KNN, we are going to iterate through training data points that are 'close' to our test points and take their classes as votes towards what our new label should be, but we follow a different idea of 'closeness'. Our modified train points (and test points) now only include d' features, which are ordered by their feature importance. We will calculate the distance between the test and train points in terms of one feature at a time; we take the votes of the ones with the least absolute distance. We will give more votes when looking at dimensions that are more 'important'.

Formally,

We define V_i as the votes along the i_{th} dimension

$$V_1 = \underset{x^{(i)} \in \mathbf{X}}{\overset{k}{\arg\min}}(|x'_1 - x_1^{(i)}|)$$

$$V_2 = \underset{x^{(i)} \in \mathbf{X}}{\overset{k-1}{\arg\min}}(|x'_2 - x_2^{(i)}|)$$

$$\ldots$$

$$V_{d'} = \underset{x^{(i)} \in \mathbf{X}}{\overset{k-d'}{\arg\min}}(|x'_{d'} - x_{d'}^{(i)}|)$$

where $x_n^{(i)}$ represents the n_{th} dimension of the i_{th} data point $\forall n \in [1, d']$, d' is the number of dimensions, and x'_n is the n_{th} dimension of the test vector.

The k in the superscript of arg min indicates that we are calculating the arg min k times. As a result, the length of V_1 will be k, V_2 will be $k-1$ and so on up until $V_{d'}$ has length $k - d'$. Note that if d' becomes higher than k, then we simply let the vote for that dimension be an empty list.

(iii) Concatenate the voters into one list
Now we simply take the different votes across different dimensions and concatenate them into one large list of votes:

$$V^* = V_1 ^\frown V_2 ^\frown \ldots ^\frown V_{d'}$$

(iv) Now using this list V*, we will determine class label for the test vector by selecting the class that is most voted upon in the V*:

$$y' = \max_{t^{(i)} \in L} \sum_{t^{(j)} \in V^*} \mathbb{1}(t^{(i)} = t^{(j)})$$

where $t^{(i)}$ is the i_{th} class, and $t^{(j)}$ is the target class. The resulting y' is our guess for what the class label for this test point should be!

Using this technique enables us to not forego the hierarchy of features created by using feature selection through information gain, in fact we use this hierarchy to make the more relevant features more important when predicting the class of a test point.

However, we can still do better than this idea. This idea looks at features entirely independently, it assumes that a single feature, in isolation, can give us information about the class. This, however, is not always the case; we may face situations (especially in the domain of image classification) where one feature alone is not enough to give us any significant information, it is a collection of features that are relevant. Therefore, we take this intuition and formalize it, modifying the GRA-KNN algorithm.

3.2 Feature-Batch GRA-KNN Algorithm

We follow the same general format from GRA-KNN except we want to consider more than just 1 feature at a time while still keeping the hierarchy in features based on information gain. Let $i \in [1, d']$ be the feature we want to take the votes for. We define the set D_i that is associated with feature i, it will include all the additional features that we want consider when taking the votes associated with this feature i. We want the features in this set to be closely related to this i_{th} feature, therefore we can use those features that have the highest covariance with the i_{th} feature!

Formally, This is how our modified algorithm will be.

Consider the same training set: $\mathbf{X} = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(N)}, t^{(N)})\}$, and the same test vector x'. The steps are as follows

(i) Get the 'most important' features using Information Gain - same as before.

(ii) We create a covariance matrix between these 'most important' features:

$$Cov(\mathbf{X}) = E[(x - \mu)(x - \mu)^T] \tag{3}$$

For any select feature i, we can then observe the features that have the highest covariance with i (the highest covariance values in the ith row), we select λ (hyperparameter) such features and create D_i with all these features.

(iii) Iterate through these features and gather 'votes' towards what the test data point's label should be: Having create these associated set of features for each feature, we can now take the distance of this collection of features (as euclidean distance) in order to find the 'closest' test points:

$$\begin{gathered}
\text{We define } V_i \text{ as the votes along the } i_{th} \text{ dimension} \\
V_1 = \underset{x^{(i)} \in \mathbf{X}}{\arg\overset{k}{\min}}(||x'_{D_1} - x^{(i)}_{D_1}||_2) \\
V_2 = \underset{x^{(i)} \in \mathbf{X}}{\arg\overset{k-1}{\min}}(||x'_{D_2} - x^{(i)}_{D_2}||_2) \\
\cdots \\
V_{d'} = \underset{x^{(i)} \in \mathbf{X}}{\arg\overset{k-d'}{\min}}(||x'_{D_{d'}} - x^{(i)}_{D_{d'}}||_2)
\end{gathered}$$

The notation $x^{(i)}_{D_1}$ is a vector that includes all the dimensions in D_1 of the ith data point. The notation x'_{D_1} is a vector that includes all the dimensions in D_1 of our test point.

(iv) Concatenate all the votes and determine our prediction label by finding the most voted-for class, identical to before.

With the GRA -algorithm and its modification established, we can now evaluate the accuracy of these algorithms to assess whether there is validity to them.

4 Results

In this paper, we used the MNIST-digit dataset built into the TensorFlow package. This dataset consists of 60,000 training points and 10,000 test points. We split the training points into training and validation sets of size 45,000 and 15,000 respectively. We use the validation sets to find optimal values of the hyperparameters and test our model with the fixed hyperperamters.

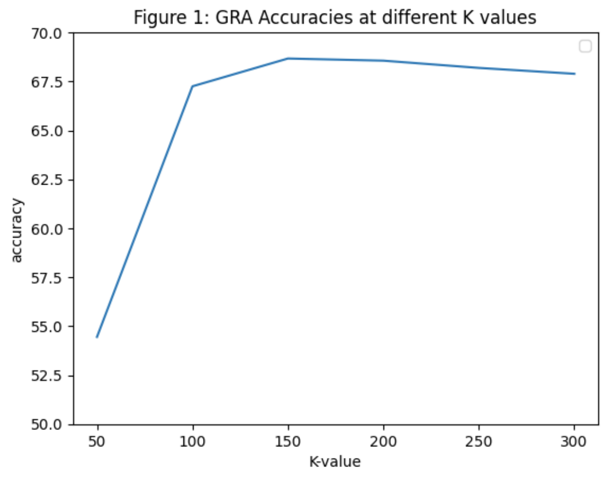
In Table 1, we report the optimal hyperperameters and test accuracies of a Vanilla KNN algorithm, alongside the different models that can be used when trying to reduce dimensions, as discussed in the Literature Review.

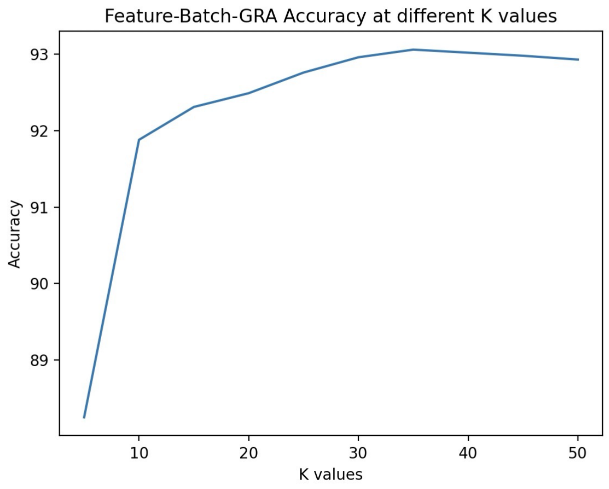
Table 1: Test Accuracies of existing KNN algorithms.

	Vanilla-KNN (784 Features)	PCA-KNN (100 Features)	IG-KNN (100 Features)	IG-Weighted-KNN (100 Features)
k value (found through validation set):	1	1	5	5
Test Accuracy:	96.91%	97.00%	95.78%	95.72%

As we can see PCA-KNN performed the best and IG-weighted-KNN performs the worst, however, the accuracies of all the techniques are very close to one another.

Now, using these accuracies as benchmarks we can evaluate our GRA-KNN algorithm. Figures 1 and 2 show the validation accuracies of the GRA-KNN and the Batch GRA-KNN models respectively, as a function of the hyperparameter k.





The value of k that reports the highest validations for GRA and Batch-GRA are 150 and 35 respectively, these values of the hyperperameter of k are drastically different from our benchmark models. However, that is to be expected, as in GRA this hyperperamter also dictates how many features we assess when gathering votes, while in the benchmark we always look at all the features for data points. Table 2 reports the test accuracies of these models given optimal hyperparameters.

Table 2: Test Accuracies of GRA and Feature-Batch-GRA.

	GRA	Feature Batch GRA
Optimal K value:	150	35
Test Accuracy:	69.32%	93.09%

Our **Feature-Batch-GRA-KNN** algorithm performed with 93.09% accuracy on the test dataset, which is better than the 69.32% accuracy of the **GRA-KNN** algorithm. Using these results, we can dive into a discussion of their relevance and limitations.

5 Discussion

The **GRA-KNN** algorithm achieved an accuracy of **69.32%** and our **Feature-Batch-GRA-KNN** algorithm achieved an accuracy of **93.09%**. Whilst the result of the original GRA-KNN was mediocre, the modified GRA algorithm was highly competitive with the existing dimension reduction techniques - given they are reduced to the same number of dimensions. While it may not yet perform better than those existing algorithms, the validity of this approach has been clearly shown. Our **GRA-KNN** algorithm performs worse than **Feature-Batch-GRA-KNN** algorithm as the comparison is only based on the important features. Looking at just these features might not give us much information when it comes to image classification. This is where **Feature-Batch-GRA-KNN** goes a step further by looking at additional features that are highly correlated with the important features. This modification significantly increased our prediction accuracy to **93.09%**.

That being said, there are still some important limitations of this paper. Firstly, due to the time constraint, we had to fix the number of dimensions we are reducing down to, rather than leaving it as a hyperperameter as we initially wanted. It is important to consider how the GRA and Feature-Batch-GRA will perform given different reductions in dimensions. Another 'hyperperameter' that we had to fix in place was the number of 'associated features' we consider for the important features in the Feature-Batch-GRA algorithm. Exploring further, we may also consider different ways to weigh the important and less important neighbours. The current algorithm simply consider 1 less neighbour when reducing to a lesser important feature as we iterate through the features, however this reduction can be different, infact it can even be made non-linear.

Lastly note that the data we used for this project had 784 features but the regular KNN algorithm already performed really well with an accuracy of **96.91%**. One of the challenging part of this project was to come up with a data-set high high dimensions where the KNN algorithm performed poorly. Such data-set would highlight the importance of dimension reduction and the performance of our algorithm will be more apparent. Additionally, due to time pressure, we were not able to optimize the implementation of our algorithm. This caused the runtime of our GRA-KNN and Feature-Batch-GRA-KNN to be quite high. We can vectorize these algorithms further to reduce the runtime in the future.

6 Conclusion

The curse of dimensionality is a common and troubling problem that machine learning practitioners deal with constantly. As we discussed in this paper, there are already a range of models that tackle this issue. That being said, new ideas, such as the one discussed in this report are important as they provide new insight and techniques to improve the state of our machine-learning models. The paper has evidenced that there is validity to the proposed Feature-Batch-GRA-KNN algorithm, and this new approach should be explored and developed further to add to the ever-growing world of machine learning.

References

[1] Michel Verleysen and Damien François. "The Curse of Dimensionality in Data Mining and Time Series Prediction". In: *Computational Intelligence and Bioinspired Systems*. Ed. by Joan Cabestany, Alberto Prieto, and Francisco Sandoval. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 758–770. DOI: 10.1007/11494669_93.

[2] Rohmat Indra Borman et al. "Implementation of PCA and KNN Algorithms in the Classification of Indonesian Medicinal Plants". In: *2021 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE)*. 2021, pp. 46–50. DOI: 10.1109/ICOMITEE53461.2021.9650176.

[3] Thee Zin Win and Nang Saing Moon Kham. "Information Gain Measured Feature Selection to Reduce High Dimensional Data". In: *ICCA 2019 Proceedings*. Feb. 2019. URL: https://meral.edu.mm/records/3413.

[4] Halil Yigit. "A weighting approach for KNN classifier". In: *2013 International Conference on Electronics, Computer and Computation (ICECCO)*. 2013, pp. 228–231. DOI: 10.1109/ICECCO.2013.6718270.