Characterising the importance of genetic relatedness in phenotype prediction

Eljas Röllin
Supervised by Lucie Bourguignon, Dr. Michael Adamer
Research Group of Prof. Dr. Karsten M. Borgwardt
Department of Biosystems Science and Engineering
Eidgenössische Technische Hochschule Zürich
Zürich/Basel
`roelline@student.ethz.ch`

Abstract

The effect of single nucleotide polymorphisms (SNPs) on complex phenotypes is often modelled assuming a linear relation of SNPs and the phenotype. In this work, we challenge this assumption and investigate the potential importance of genetic relatedness. We investigate whether individuals with similar SNPs have a similiar phenotype, for which we consider adult human height. Considering similarity in high dimensions, we discuss the curse of dimensionality and approaches to reduce its effects. We focus on developing a software tool which implements these approaches and is able to deal with large datasets. The resulting software tool implements the k-Nearest Neighbor (kNN) rule, and features batched data loading and usage of Graphics Processing Unit (GPU) acceleration. We find that genetic relatedness by usage of the kNN rule has no eminent importance in adult human height. The developed software tool offers flexibility on many levels of the kNN rule, and outperforms its corresponding `scikit-learn` implementation in terms of execution time and memory requirements.

1 Introduction

1.1 Genetic Relatedness in phenotype prediction

Genetic mutations are a causes and risk factors of many diseases. To associate genetic markers with a disease, or the risk of developing it, is an important step in reducing the burden of such diseases. Common methods that aim to associate phenotypes with genetic markers rely on linearly regressing Single Nucleotide Polymorphisms (SNPs) against the phenotype, and use the regression coefficients as proxies for effect size.(1) The result of such methods is a Polygenic Risk Score (PRS), combining multiple SNPs into a single number. This approach, although applied successfully in the past (2), relies on a model assumption of linear relation of the SNPs and phenotype.
In this work, this assumption is challenged, and we focus on the use of genetic relatedness. Rather than assuming linear relation of SNPs and phenotype, genetic relatedness evaluates the potential of similarity of individuals at selected SNPs. An interpretable score in analogy to a PRS can then be computed based on known phenotypes of similar individuals.
Instead of a disease and risk states, we use adult human height as readily observable complex phenotype. This allows to formulate the task of phenotype association with genetic markers as a regression problem. We refer to this as phenotype prediction, which is in our case human height prediction. The dataset used is a subset of the UK Biobank, a biomedical database.(3)

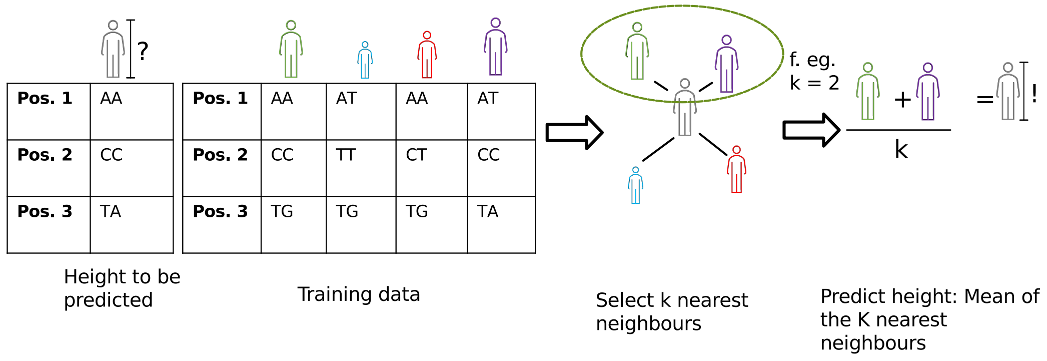

Figure 1: k-Nearest Neighbor Regression of human height from genetic relatedness.

1.2 k-Nearest Neighbor Regression

For prediction of human height based on genetic relatedness, we use the k-Nearest Neighbor (kNN) rule. The kNN rule assigns to an unclassified sample point the classification of the nearest of a set of size k of previously classified points.(4) The kNN rule is an omnipresent technique in data mining due to its simplicity. Here, the regression instance of this rule is applied. Figure 1 illustrates the concept of kNN regression. Prediction of height of a test individual is based on the mean of its k most similar individuals from the training set based on a set of SNPs.
The kNN regression offers several settings and hyperparameters to be tuned:

1. The set of SNPs considered. This is a step of feature selection ahead of the actual kNN regression.
2. The weight of the SNPs. This can be uniform, or according to prior beliefs of the importance of the considered SNPs.
3. The number of neighbors considered, k.
4. The metric used to quantify similarity, that is quantify genetic relatedness. We introduce some metrics in the subsection Distance Metrics. Some of them have parameters themselves.
5. An optional step of dimensionality reduction on the selected SNPs.
6. Using a weighted or unweighted mean of the k nearest neighbors, using the inverse distance as weight.

Besides being straightforward to compute, the kNN rule comes with challenges in terms of memory and computational complexity for large datasets. To examine this, assume we consider a set of d SNPs. Then each individual is represented as a point in d-dimensional space. Figure 2 outlines the brute force computation scheme for finding the k nearest neighbors of m test points from n training points. This computation scheme involves computing the m×n distance matrix, and m times sorting a vector of length n. Computation of the distance matrix requires $\mathcal{O}(nmd)$ multiplications for the n×n distances. The complexity of this computation scheme is problematic: $\mathcal{O}(nmd)$ for the $m \times n$ distances computed and $\mathcal{O}(nmd)$ for the m sorting processes. To reduce the computational

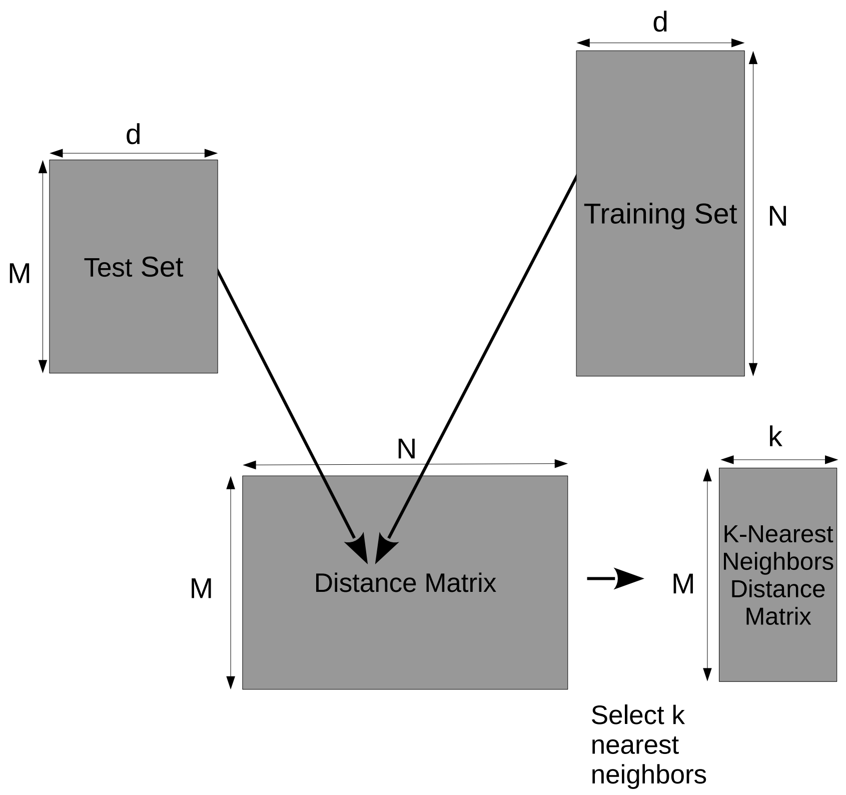
d
Training Set
N
d
Test Set
M
N
k
M
Distance Matrix
M
K-Nearest Neighbors Distance Matrix
Select k nearest neighbors

Figure 2: Naive brute force computation scheme of the kNN rule. Each data point is represented in d dimensions. The aim is to find for each data point in the test set its k nearest neighbors among the training set.

complexity by avoiding computation and sorting of the entire distance matrix, several methods have been proposed such as using locality sensitive hashing(5) or the use of k-d trees(6). An overview of several methods can be found here(7). Another approach to improve the execution time is to retain the brute force computation scheme, but using Graphics Processing Unit (GPU) acceleration.(8; 9) However, in addition to the computational complexity, the required memory of the kNN rule is also unfortunate: $\mathcal{O}(d(M{+}N) + MN)$ for storing the test set, training set and distance matrix in the computer memory. For large datasets not fitting into computer memory, this becomes problematic. For the GPU acceleration, this becomes additionally critical because of GPU's small memory compared to commonly used Central Processing Units (CPUs). In this work, we make use of GPU acceleration to reduce the execution time of kNN regression, and implement a batchwise computation scheme similar to a solution proposed by (9) to uncouple memory requirements from dataset size.

1.3 The Curse of Dimensionality

When considering d SNPs, we represent each individual as a point in d-dimensional space. High dimensional data, while potentially offering a lot of information, comes with issues that do not arise in low-dimensional data. In a high dimensional space, the data becomes sparse, and traditional indexing and algorithmic techniques fail.(10) For the particular aim of the kNN rule, it has been shown that under broad conditions, as dimensionality increases, the distance to the nearest neighbor approaches the distance of the farthest neighbor.(11) One approach towards such issues is dimensionality reduction. In this work, Principal Component Analysis (PCA) is implemented as dimensionality reduction technique and its effect examined. Another approach considered involves choosing the metric and its parameters. It has been shown that the Minkowski distance (or L_k norm) is susceptible to the dimensionality curse, and choosing Minkowski parameters smaller than one could potentially improve the contrast from nearest to furthest neighbor in high-dimensional spaces.(10) The authors refer to such Minkowski-like distance metrics with Minkowski parameter smaller than one as "fractional distance metrics", which term is used thereafter.

1.4 Distance Metrics

In this project, different distance metrics are used. Here, the used distance metrics are briefly recapitulated.
Equation 1 shows the Minkowski distance of two vectors, also known as Minkowski norm, L_k norm, or p-Norm.

$$\mathrm{d}_{\mathrm{mink}}(\mathbf{x},\mathbf{y}) = (\sum_{i=1}^{d} |\mathbf{x}-\mathbf{y}|^{p})^{1/p}, p \geq 1. \tag{1}$$

For $p = 2$ and $p = 1$, the Euclidean distance and Manhattan distance are recovered, respectively.
Equation 2 shows the cosine similarity of two vectors, which is their L_2 normalized dot product.

$$\mathrm{d}_{\cos}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^{\mathrm{T}}\mathbf{y}}{||\mathbf{x}|| \cdot ||\mathbf{y}||} \tag{2}$$

The covariance matrix S can be estimated from the training data in practice.
Equation 3 shows the the fractional distance metric with parameter $p < 1$. This is a distance metric, in contrast to the Minkowski distance with parameter $p < 1$.

$$\mathrm{d}_{\mathrm{frac}}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} |\mathbf{x}-\mathbf{y}|^{p}, p < 1. \tag{3}$$

1.5 Outline

The rest of this work is structured as follows: In the methods section, the data and developed software are described. In the results section, first the execution time of the custom kNN regression implementation are compared to `scikit-learn`'s kNN regression implementation. Second, the prediction accuracy of adult human height of kNN regression is shown. In the discussion section, these findings are wrapped up and discussed along with limitations of this approach. The conclusion finally summarizes the major outcomes of this work, that is the performance and flexibility of our custom kNN regression implementation and the importance of genetic relatedness in phenotype prediction.

2 Methods

2.1 Data

The UK Biobank is a large-scale biomedical database and research resource, containing in-depth genetic and health information from half a million UK participants.(3) The data had been preprocessed as described in (12). Briefly, data from white Caucasians was used (450'945 individuals), and SNPs were filtered for minor allele frequency and missing rate criteria. Mitochondrial and sex chromosome SNPs were also removed. We split this dataset into a training set of 360'753 individuals (80%) and a test set of 90'192 (20%) individuals. The 10k SNPs most associated with the phenotype in a GWAS on the train set were retained. Both male and female subgroup's heights were Z-scored, and the data pooled.
For testing purposes, a sample dataset and two synthetic datasets were created. The sample dataset consists of a small subsection of the dataset described above. One synthetic dataset consists of train and test data from well-separated clusters in two dimensions. The other synthetic dataset consists of train and test data from well-separated clusters in two dimensions, to which 18 non-meaningful noisy dimensions were added.

2.2 kNN Regressor

We developed a batch-wise GPU accelerated kNN Regressor class in Python 3.8. The two main challenges when dealing with large datasets are the computational complexity, in terms of operations to be executed, and the memory storage. It is described here how we used batch-wise data processing

to deal with memory issues, and how this enables us to speed up the computation by using GPU acceleration.
The dataset of interest does not fit into GPU memory. This is why a workaround is required that uncouples memory requirements from dataset size. Our implemented solution to this challenge is illustrated in figure 3, where the batch-wise distance matrix computation is outlined. This figure further highlights how this is different from the naive computation illustrated in figure 3. Instead of trying to store matrices with memory requirements of $\mathcal{O}(Md)$, $\mathcal{O}(Nd)$ and $\mathcal{O}(MN)$, this approach allows to batch-wise store matrices with memory requirements of $\mathcal{O}(md)$, $\mathcal{O}(nd)$ and $\mathcal{O}(nm)$, respectively. The train and test batches are moved to GPU memory for distance computations. After this, the distance matrix part is moved back to CPU memory where the kNN distance matrix is assembled. The execution time of this algorithm is hence the result of moving data between memory, which is slow, and matrix operations on a GPU, which are fast.
This custom kNN regressor features all distance metrics outlined in the Distance Metrices section. Further, the custom kNN regressor allows for SNP weights for the Minkowski, Cosine and Fractional distance metrics.

2.3 Dimensionality Reduction

As to being able to cope with large datasets, we implemented a batch-wise PCA class in Python 3.8. This class computes a PCA on the input data, of which the principal components can be used by our custom KNN regressor class.

2.4 Testing

Two types of tests were done. First, we tested that our custom kNN regressor and PCA classes match the results of the corresponding `scikit-learn`(13) Version 1.0 classes on the sample dataset. Second, the custom KNN regressor and PCA class were tested on the two synthetic datasets, where predictable outcomes were to be achieved.

2.5 Prediction Evaluation

In kNN regression, first the Z-score of an individual from the test set was computed from its k nearest neighbors' Z-scores. In a second step, the Z-score was reconverted to a height in centimeters, inverting the operation described in the Data section. This involved the corresponding Z-scoring coefficients, and the age of the test individual and corresponding linear regression coefficients. This recomputed height in centimeters is what we refer to as the prediction. For evaluating these predictions of the kNN regression, we used four evaluation metrics: The mean squared error (MSE), mean absolute error (MAE), Pearson's correlation coefficient r, and the coefficient of determination r^2.

2.6 Hardware

The experiments were performed on the research group's computation cluster, which is based on the Slurm Workload Manager. This cluster features GPU nodes of 2x 8 core Intel Xeon CPU E5-2620 v4 at 2.10GHz with 128 GB Ram, 8x NVIDIA GTX 1080. The CPU node is a 4x 18 core Intel(R) Xeon(R) Gold 6154 CPU at 3.00GHz with 1024 GB Ram. Per kNN regression, a GPU and 3 CPUs à 10 GB Ram were requested.

2.7 Data and Code availability

The UKBiobank data are available for all bona fide researchers and can be acquired by applying at http://www.ukbiobank.ac.uk/register-apply/. The code of the custom kNN regressor class and the custom PCA class are available on GitHub (YY-INSERT-LINK-YY). The tests as well as the synthetic datasets used are also available there.

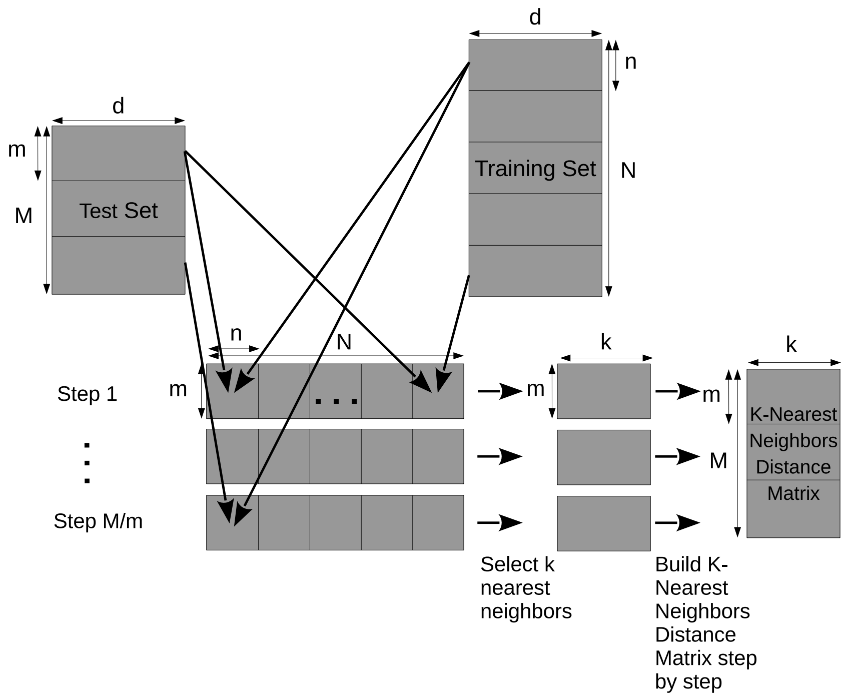

Figure 3: Batch-wise brute force computation scheme of the kNN rule. Each data point is represented in *d* dimensions. The aim is to find for each data point in the test set its k nearest neighbors among the training set. The parameters *m* and *n* describe the batch sizes of the test and training set, respectively.

3 Results

We performed two types of experiments. In the following first part, we show the results of experiments investigating *execution time*. In the second part, we show the results of our investigation of the potential of examining different hyperparameters for *prediction accuracy*.

3.1 Execution time comparison

We compared the execution time of our custom kNN regression implementation with `scikit-learn`'s KNeighborsRegressor(13). `scikit-learn` is a well-known and widespread Python module for machine learning, and is taken as the gold standard. We did not find a Python module using GPU support that is flexible enough for our questions of interest. For these execution time experiments, we used the dataset split into train and test set as described in the methods section. In more detail, for the execution of one `scikit-learn` kNN regression job we allocated 3 CPUs, each with 10 GB memory, except for the cosine distance metric, for which 14 GB were used. 10 GB, respectively 14 GB for the cosine distance, per CPU was the minimum memory allocation required to execute one `scikit-learn` kNN regression job, requiring to load the full dataset into memory. For one custom kNN regressor job, also 3 CPUs, each with 10 GB memory, were allocated, in order not to exceed memory requirements compared to the `scikit-learn` implementation, and a GPU. The batch-size was empirically maximized in order to reach the best performance given the allocated memory. Figure 4 shows the required execution time for different unweighted distance metrics. Figure 5 shows the required execution time for different weighted distance metrics.

3.2 Predictions on adult Human height

As the kNN rule is well-known, we omit characterization of its capability for classification and regression here. Examples on two synthetic datasets can however be found in the testing section which is available on GitHub (see methods). The critical reader can find a demonstration on how

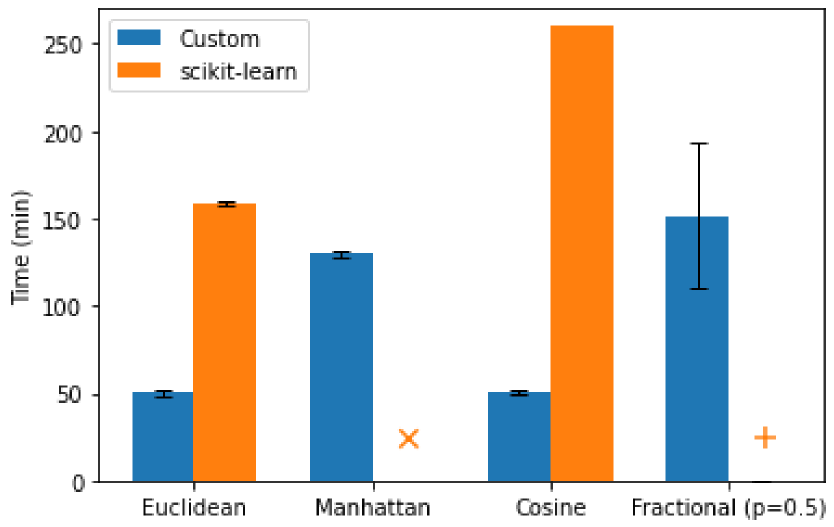

Figure 4: Execution time comparison of custom kNN-Regressor and `scikit-learn` kNN-Regressor. Bars represent the mean over 6 runs, errorbars represent the standard deviation. $\times$ this implementation did not finish within observed time (over 1000 minutes). $+$ this distance metric is not implemented in `scikit-learn`.

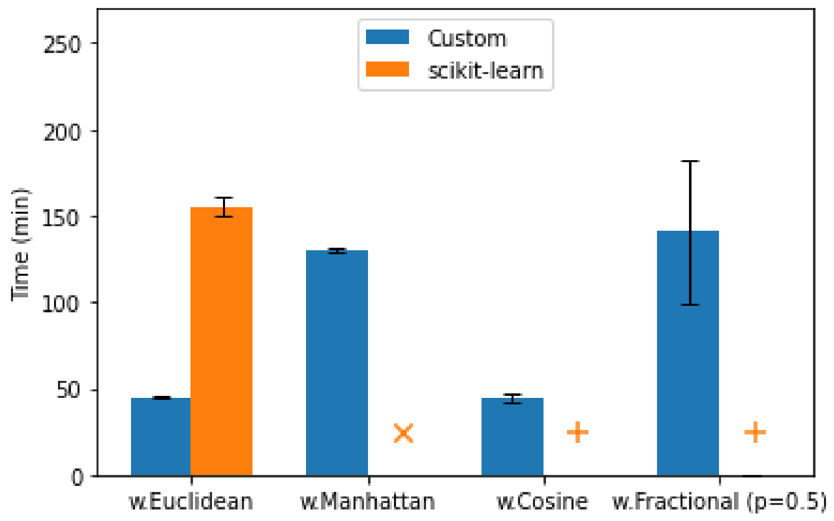

Figure 5: Execution time comparison of custom kNN-Regressor and `scikit-learn` kNN-Regressor. Bars represent the mean over 6 experiments, errorbars represent the standard deviation. w. means that the distance metric is used with weighting the SNPs. $\times$ this implementation did not finish within observed time (over 1000 minutes). $+$ this weighted distance metric is not implemented in `scikit-learn`.

Table 1: Prediction accuracy metrics of kNN regression of adult human height. The unweighted mean of the 1000 Nearest Neighbors was used. The 10k SNPs most associated with height based on a GWAS performed on the train set were used. CV: 5-fold cross-validation on training data. Full: using the train-test split on the entire dataset. Mean Prediction (BL): Baseline, for each cross-validation split, the mean of the training split was used as prediction for all points of the test split. MSE: Mean Squared Error. MAE: Mean Absolute Error. r: Pearson's correlation coefficient. r^2: Coefficient of determination.

Data	Metric	SNP Weights	MSE	MAE	r	r^2
CV	Mean Prediction (BL)	None	41.639 (0.178)	5.118 (0.007)	0.516 (0.002)	0.516 (0.002)
CV	Euclidean	Beta	39.384 (0.181)	4.979 (0.007)	0.543 (0.002)	0.543 (0.002)
CV	Manhattan	Beta	39.493 (0.188)	4.986 (0.007)	0.541 (0.002)	0.541 (0.002)
CV	Cosine	Beta	39.236 (0.187)	4.969 (0.008)	0.544 (0.002)	0.544 (0.002)
Full	Euclidean	None	39.905	5.009	0.535	0.535
Full	Euclidean	Beta	41.965	5.135	0.511	0.511
Full	Manhattan	None	40.024	5.016	0.534	0.534
Full	Manhattan	Beta	39.793	5.002	0.537	0.537
Full	Cosine	None	39.803	5.001	0.537	0.537
Full	Cosine	Beta	41.965	5.135	0.511	0.511
Full	Fractional (p=0.5)	None	40.084	5.019	0.533	0.533
Full	Fractional (p=0.5)	Beta	39.867	5.006	0.536	0.536

Table 2: Prediction accuracy metrics of kNN regression of adult human height. Results from 5-fold cross-validation on training data. The SNPs were not weighted. The unweighted mean of the 1000 Nearest Neighbors was used. Principal Component Analysis was done on the top 10k SNPs. "Principal Components" describes how many top Principal Components were used. MSE: Mean Squared Error. MAE: Mean Absolute Error. r: Pearson's correlation coefficient. r^2: Coefficient of determination.

Principal Components	MSE	MAE	r	r2
1	41.555 (0.148)	5.114 (0.006)	0.517 (0.001)	0.517 (0.001)
2	41.513 (0.156)	5.111 (0.006)	0.518 (0.001)	0.518 (0.001)
5	41.071 (0.171)	5.086 (0.006)	0.523 (0.001)	0.523 (0.001)
10	40.521 (0.198)	5.052 (0.008)	0.529 (0.002)	0.529 (0.002)
20	40.393 (0.190)	5.043 (0.008)	0.531 (0.002)	0.531 (0.002)
50	40.110 (0.190)	5.025 (0.008)	0.534 (0.002)	0.534 (0.002)
100	39.833 (0.189)	5.008 (0.008)	0.537 (0.002)	0.537 (0.002)
1000	39.337 (0.183)	4.975 (0.008)	0.543 (0.002)	0.543 (0.002)

dimensionality reduction with PCA can improve kNN regression's predictive accuracy in case of a few meaningful and many noisy features in the testing section on GitHub.

The potential of combining different settings for predicting adult human height within this dataset were investigated. We refer to the simple mean prediction as baseline. For this, 5-fold cross validation (CVs) on the training dataset as described in the methods section was done. We further show results from kNN regression for the full training and test dataset.

Multiple metrics were examined, as introduced in the distance metrics subsection. Further, the SNP weight was chosen between None, meaning uniform SNP weight, or Beta, meaning each SNP's weight was proportional to its Beta statistic in the GWAS. In addition, a PCA was performed on the top 10'000 SNPs, and different numbers of Principal Components were considered. Here, a selection of such setting and hyperparameter combinations is shown. Table 1 lists predictive performance results for different settings of kNN regression. Further, table 2 lists the predictive performance for a fixed setting of kNN regression, but where a PCA was performed prior to KNN regression.

Further experiments for predictive performance were executed. There, the number of nearest neighbors, and a weighted mean prediction were considered in addition to the aforementioned settings. These results are available on GitHub.

4 Discussion

kNN regression was used in order to evaluate the importance of genetic relatedness in phenotype prediction. For this, a custom made, GPU accelerated, Python KNN regressor class was created to meet the required memory, performance and flexibility needs.
As a comment on memory requirement, we notice that the allocated memory for the *scikit-learn* implementation is dictated by the size of the dataset, as the entire dataset inevitably will be loaded at once. For our custom kNN regression implementation, its memory requirement is flexible and depends on the tunable batch-size parameters n and m.
The execution time for different settings was compared to `scikit-learn`'s KNeighborsRegressor where possible. `scikit-learn` is a well-known and widespread Python module for machine learning, and was chosen as reference for this reason. The presented GPU accelerated kNN regressor outperformed the reference execution time by a factor of 3.2 and 5.1 for the unweighted euclidean distance and cosine distance, respectively. It outperformed the reference execution time by a factor of 3.5 for the weighted euclidean distance. For the manhattan distance, the reference was not able to finish within reasonable timeframe, what was not investigated further. The reference further did neither offer a weighted cosine metric, nor a fractional distance metric at all. Undoubtedly, there are implementations of the kNN rule in other programming languages certainly outperforming the presented implementation.(8; 9) No extensive comparison was performed since this would be beyond the scope of this small report. To our knowledge, no readily available GPU accelerated Python implementation of the kNN rule with the required flexibility would have been available for reference.
As to evaluate the importance of genetic relatedness in phenotype prediction, a major aim was to allow the investigation of kNN regression settings at many different levels. Tables 1 and 2 list prediction performance measures of some selections of such investigated settings. More such results can be found on GitHub. It can be noticed that the different settings show no identifiable patterns of superiority to the mean prediction baseline. In general, a slight trend of outperforming the simple mean-prediction baseline can be observed, at least for unweighted SNPs. Going below 100 nearest neighbors reduces predictive performance, whereas no improving effects with the number of nearest neighbors could be observed (data on GitHub).
As a first comment on the rather large deviances of the predictions, it might be stated that we did not perform height outlier filtering. This step was not deemed necessary for comparing predictive performance compared to the mean prediction baseline. Rather it would have had more of a cosmetic effect of improving the overall predictive performance reported, including that of the baseline.
Further, a straightforward comparison of the intuitive and easy to understand Mean Absolute Error with other authors' work is unfortunately not possible, as this quantity is often not reported. A LASSO regression approach reported that "the actual heights of most individuals are within $\tilde{3}$cm of the predicted value", but we are not sure how to understand this.
It appears that under broad settings, kNN regression for phenotype prediction based on genetic relatedness does not perform well. The reasons for this could be diverse. First of all, the curse of dimensionality persists in such attempts of comparing long vector's similarities, what genetic relatedness is an instance of. Although we tried different methods to mitigate its effects, these did not show significant predictive advantages. Second, we notice the choice of the SNPs selected. These SNPs were selected based on their *single* association with human height in a GWAS. Epistasis(14) was hence ignored in the SNP selection process. Hence although there could exist patterns among SNPs that kNN regression is able to catch, the SNPs themselves producing such patterns were selected in a pattern-insensitive manner. Irrelevant (groups of) SNPs could be present as much as interesting (groups of) SNPs could be absent in the present 10'000 selected SNPs. The problem of GWAS failing to account for epistasis is well known(14), and may be less of an issue in the future with improved algorithms, computer architecture and hardware.
The choice of pooling male and female individuals under Z-scoring showed no significant difference on the MAE to treating these as separate groups for height prediction in internal trials.
We hence characterize the importance of genetic relatedness in phenotype prediction by kNN regression to be minor under broad settings. Other methods capturing genetic relatedness such as Gaussian processes and support vector machines might be considered for improved predictive performance. The large dataset size and search for meaningful metrics pose challenges for these methods, too. Recent research already picked up deep learning methods for the problem of phenotype prediction from genetics.(12) The future will show whether methods focusing on genetic relatedness or deep learning methods eventually significantly outperform linear models in phenotype prediction.

5 Conclusion

In this work, we challenged the assumption of linear relation of SNPs and their respective phenotypes, and focused on the importance of genetic relatedness by using kNN regression. For this, we developed a new software tool in Python able to exploit GPU acceleration. This tool allows using the kNN rule on large datasets by overcoming memory issues with batched data processing. It further outperforms the widespread `scikit-learn` in execution time. The flexibility of this tool enabled us to investigate settings of the kNN rule at many levels, and to study multiple approaches towards high-dimensional data, such as specialized distance metrics and dimensionality reduction.

Acknowledgments

I would like to thank my supervisors Lucie Bourguignon and Dr. Michael Adamer for their great effort in supporting me through this work, and Prof. Dr. Karsten M. Borgwardt for hosting me in his research group for this project. Both Lucie and Michael were very supportive and helpful from the beginning, and available for any interesting discussion considering this work and surrounding topics. Lucie and Michael being this open and communicative was truly a great help for working on a student research project in times like these.

References

[1] F. Dudbridge, "Power and predictive accuracy of polygenic risk scores," *PLoS genetics*, vol. 9, no. 3, p. e1003348, 2013.

[2] N. Mavaddat, K. Michailidou, J. Dennis, M. Lush, L. Fachal, A. Lee, J. P. Tyrer, T.-H. Chen, Q. Wang, M. K. Bolla, *et al.*, "Polygenic risk scores for prediction of breast cancer and breast cancer subtypes," *The American Journal of Human Genetics*, vol. 104, no. 1, pp. 21–34, 2019.

[3] C. Sudlow, J. Gallacher, N. Allen, V. Beral, P. Burton, J. Danesh, P. Downey, P. Elliott, J. Green, M. Landray, *et al.*, "Uk biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age," *PLoS medicine*, vol. 12, no. 3, p. e1001779, 2015.

[4] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[5] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[6] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, no. 1, pp. 579–589, 1991.

[7] N. Bhatia *et al.*, "Survey of nearest neighbor techniques," *arXiv preprint arXiv:1007.0085*, 2010.

[8] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using gpu," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–6, IEEE, 2008.

[9] P. D. Gutiérrez, M. Lastra, J. Bacardit, J. M. Benítez, and F. Herrera, "Gpu-sme-knn: Scalable and memory efficient knn and lazy learning using gpus," *Information Sciences*, vol. 373, pp. 165–182, 2016.

[10] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *International conference on database theory*, pp. 420–434, Springer, 2001.

[11] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *International conference on database theory*, pp. 217–235, Springer, 1999.

[12] P. Bellot, G. de Los Campos, and M. Pérez-Enciso, "Can deep learning improve genomic prediction of complex human traits?," *Genetics*, vol. 210, no. 3, pp. 809–819, 2018.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] H. J. Cordell and D. G. Clayton, "Genetic association studies," *The Lancet*, vol. 366, no. 9491, pp. 1121–1131, 2005.