test

titlepage

Hussain Sajwani
(100045495@ku.ac.ae, gmail)

Faculty Adviser: SFF

"Supervised dimensionality reduction with simulated genetic data towards predicting disease"... or something liek that.

me srp

100045495

March 2021

Contents

use giant margins. don't use fullpage when drafting.

look at various techniques (i.e. variational/supervised autoencoders, ...)

Introduction doesn't have to be called "Introduction". section headings can be a little more interesting, but this is never wrong.

Find a figure.

1 Introduction

Height, weight, IQ, eye color, and many others are all traits of any human being. What determines these traits is a mixture of our genetic information and external factors. Besides controlling traits, genetic information and other external factors can also determine whether someone has a disease or not. A natural question arises: given someone's genetic information, can we predict their traits or a disease they might have? To answer this we need access to genetic data. Given how expensive sequencing is and privacy issues, obtaining genetic data is often a hurdle. Even when access to genetic data is obtained, it is often the case that the sample size, n, is ~~relatively~~ much smaller than the dimensionality, d. Classical statistical methods often fail in this high-dimensional setting [1]. Thus if we wanted to use such methods, we will need to find a way of extracting the most important features in our data. In this study we look at various techniques and evaluate their performance on simulated genetic data. In particular we will extract or construct features that we think are more strongly associated with a particular trait. What we mean here is simulating base positions (locus) that mutate in at least 1% of the population. These loci are called single-nucleotide polymorphisms (SNPs.)

also

For a given simulated dataset, we simulate the genotype and the phenotype of n individuals. The genotype of an individual i is a d-dimensional vector representing d SNPs. Let x_{ij} be a random variable denoting the value of the j^{th} SNP of the i^{th} individual. Each x_{ij} takes values in $\{0, 1, 2\}$ representing how much of the major allele of the j^{th} SNP the i^{th} individual has. Let $X \in \mathbb{R}^{n \times d}$ be the matrix containing the x_{ij}'s mentioned above as its elements. We will call X the *genotype matrix*. The phenotype is then simulated as some combination of genetic effect (genotype), environmental factors, and observational noise [2]. The degree to which each of the three components affects the final phenotype can be varied. The phenotype is an n-dimensional vector $y \in \mathbb{R}^n$ where the i^{th} entry represents a random variable measuring of some trait. The constraint of not having enough data typically means we have $d \gg n$ i.e. we have very few samples compared to how many SNPs we have. This presents a problem as the classical theory of "large n, fixed d" fails to give predictable results. The classical theory breaks down in higher dimensions.

footnote: recall humans are diploid so at each location they have two base-pairs. Each of these basepairs can be equal (or not) to the major allele, which is coded as 0, 1, or 2.

2 literature review

This idea of using supervised dimensionality reduction is not novel at all. In [3], the authors utilize p-values assessing the goodness of a fit of a logistic model between each SNP and polygenic obesity to judge a SNP's effect in predicting polygenic obesity. The authors then pick a threshold p_0 and use SNPs with $p < p_0$ to predict polygenic obesity using a deep neural network. [3] claims an AUC of 0.99. Another example is the approach in [4], where regions of DNA, called promoters, are passed through a convolutional neural network. The promoters with highest classification accuracy are then passed into a bigger convolutional neural network. [4] claims an accuracy of 0.769. The authors in [5] propose a novel method of training neural networks in high dimensional regimes. Using some insight from functional analysis, they propose a way of regularizing the weights with an l_p norm by looking at the l_q, with $\frac{1}{p} + \frac{1}{q} = 1$ norm of the neural network's gradients. When this regularization is applied, the authors claim an AUC of 0.926 on a dataset with $n = 85$ and $d = 22,283$.

3 methodology

transition from last section. previous methods don't do simulation -- ground truth tricky, so we simulate. Simulate, model/fit, evaluate performance (we can do this better because of simualtion).

3.1 PhenotypeSimulator

We will use the PhenotypeSimulator (PS) R package [2] to simulate genotype-phenotype relations. PS can either take the genotype as input or simulate its own relatively simple genotype. Since we do not have access to data, we decided to go with the option of simulating the genotypes using PS. The simulated genotypes are very simple. The genotype matrix X is simulated as

$$\text{vec(X)} \sim \text{Multi}(nd, (p_0, p_1, p_2)).$$

Where each p_k represents the allele frequency. e.g. p_0 is the frequency of there being 0 of the major allele. The phenotype is simulated to be the output of the genotype and some external factors. Mathematically speaking, the phenotype is some convex combination of genetic factors and external factors. In this study we constrain ourselves to only looking at the genetic factors. In all simulations we let external factors only contribute 5% to the simulated phenotype. The genotype contribution is further broken down to

and this effect size is denoted by h2b

and their effect size on the phenotype h2s. A second component...

two components. The first controls how much a select group of SNPs contributes to the phenotype. This select group of SNPs are called causal SNPs. typically, this is a very small amount of SNPs. We will call the number of causal SNPs d_c. The second component controls how much the polygenic effect of all SNPs on the final phenotype. This is different than the first component in that there is no select few SNPs that control the phenotype but rather all SNPs contribute. ~~The effect of each SNP adds up to the final expression. PS calls these two components h_2^s and h_2^b, repectively.~~ Both of which are nonnegative real numbers such that $h_2^s + h_2^b = 1$. We will denote each simulated dataset by a with tuple

4-tuple

$$(n, d, d_c, h_2^s).$$

These will represent the parameters of interest to us in PS. The phenotype generated is then some vector $\hat{y} \in \mathbb{R}^n$ which we convert into a binary vector $y \in \{0,1\}^n$ by thresholding $\hat{y}$. Simply speaking PS works as follows

$$\text{Phenotype} = 0.95 \times [h_2^s \times \text{Causal SNPs} + h_2^b \times \text{Polygenic effect}] + 0.05 \times \text{External factors}$$

dimensionality...

3.2 Reduction methods

In this study, four dimensionality reduction algorithms will be tested. The first two algorithms will simply extract k features the algorithm thinks are relevant and discard the rest. The other two will extract k new features built from the original d features.

... and supervised autoencoders

Figure

3.2.1 Neural networks and autoencoders

A typical binary classifier deep neural, DNN : $\mathbb{R}^d \longrightarrow [0,1]$, is a model made of a sequence of affine transformations that are passed, elementwise, through some nonlinearties. i.e. given an observation $x \in \mathbb{R}^d$, we apply some affine transformation represented by a matrix $W^{(1)} \in \mathbb{R}^{(d^{(2)}+1)\times(d+1)}$, called the weight matrix, (the +1 is added to represent the bias term of the affine model. Every observation x is padded with a constant variable to reflect this bias) and then pass it elementwise into some nonlinear function, called the activation function, $g^{(1)}$. This process is repeated L times with the l^{th} time represented with an affine transformation with matrix $W^{(l)} \in \mathbb{R}^{(d^{(l)}+1)\times(d^{(l-1)}+1)}$ and a nonlinear function $g^{(l)}$. The l^{th} repetition is called the l^{th} layer of the neural network. $d^{(l)}$ is called the width of the l^{th} layer and the elements of $W^{(l)}$ are called the weights of the layer. The final layer, the L^{th}, has output size $d^{(L)} = 2$.The output here can be interpreted as two probabilites of the sample having $y = 0$ or $y = 1$, respectively. Common choices of $g^{(l)}$ are

$$\text{ReLU}(x) = \max\{0, x\}$$

and

$$\sigma(x) = (1 + e^{-x})^{-1}$$

The choice of L, $d^{(l)}$, and $g^{(l)}$ for any $l = 1, \ldots, L$ is made by the designer of a neural network. These parameters which are to up to the designer are called the hyperparameters of the neural network. The choice of $W^{(l)}$ for all $l = 1, ..., L$ though is determined by an optimization algorithm. For a set of weight matrices $W = (W^{(1)}, \ldots, W^{(L)})$ we evaluate the performance of the neural network by comparing its prediction $\hat{y} = \text{DNN}(x; W)$ with the actual value of y. Remember that $\hat{y}$ is a two dimensional vector. Its second element $\hat{y_1}$ is the probability the neural network "thinks" that $y = 1$. In this study we use cross-entropy [6] as our objective function to evaluate the performance of a DNN. cross-entropy is defined as

$$\begin{aligned} J(W) &= -\mathbb{E}_{x,y\sim p_{\text{observed}}} \log \text{DNN}(x; W)_1 \\ &= -\log \hat{y_1}. \end{aligned}$$

stochastic gradient descent. specify parameters.

The goal now will be to find W such that $J(W)$ is minimized. First we intialize all the weights of all of the neural network randomly. Then we pass an observation x_i through the network and observe some $\hat{y_i}$. With this, we can evaluate $J(W)$ to compare y_i and $\hat{y_i}$. We can use this information to judge how good the weights W are. complete this with discussion on gradient descent A logistic regression model can be modeled as a one-layer neural network with $g^{(1)}(x) = (1 + e^{-x})^{-1}$. and $W \in \mathbb{R}^{2\times(d+1)}$.

Autoencoders are a special type of deep neural network where the final layer is not some binary variable but rather a d-dimensional vector. Given an input to the neural network $x \in \mathbb{R}^d$, the goal of an autoencoder is to first compress x into some smaller vector $x_k \in \mathbb{R}^k$ with $k < d$ and then using

figure for autoencoder

x_k to reproduce, to its best ability, the input vector x. Then let the first $\lfloor L/2 \rfloor$ layers have widths $d^{(1)} > d^{(2)} > ... > d^{(\lfloor L/2 \rfloor)}$ with $k := d^{(\lfloor L/2 \rfloor)}$. This part of the encoder is called the encoder. The other layers are to be mirrored around the middle layer. i.e. $d^{(1)} = d^{(L)}$, $d^{(2)} = d^{(L-1)}, \ldots$, and so on. This does not mean these layers have the same weights. We are only imposing that the widths of the layers be tied together. Inspired by PCA, there has been some work in the literature exploring weight-tied autoencoders. See [7] for a discussion on the matter. This part of the encoder is called the decoder. The final layer then produces a d-dimensional vector $\hat{x}$ which we interpret as the reconstruction of x. Cross-entropy does not work well here. So instead we use mean squared error (MSE) as an objective function to evaluate performance of the autoencoder. For a vector x that is approximated with $\hat{x}$ by the autoencoder, MSE is defined as

$$\text{MSE}(W) = (x - \hat{x})^2$$

We use a similar procedure to minimize this. The compressed representation then comes from taking the output of the middle layer. Notice that these features are extracted features; they might not have a biological interpretation. This is one downside to dimensionality reduction methods that extract features rather than select them. Also notice that nowhere in the above description do we use the information from the phenotype y. We are simply looking at a lower dimensional representation of the genotype without taking into account the response of the phenotype. This could be an issue as the autoencoder might not learn to emphasize the presence of a causal SNP. For this reason we use supervised autoencoders.

We combine these two tasks of classification and reconstruction into a neural network structure called supervised autoencoders. Let

CITE

$$\begin{aligned} \text{enc} &: \mathbb{R}^d \longrightarrow \mathbb{R}^k \\ \text{dec} &: \mathbb{R}^k \longrightarrow \mathbb{R}^d \\ \text{clf} &: \mathbb{R}^k \longrightarrow [0, 1]. \end{aligned}$$

These functions represent the encoder part of an autoencoder, the decoder part, and a classifier that takes in a k dimensional representation of a d-dimensional vector. Assume that all of these functions are parts of a neural network (each represents a sequence of affine transformations passed through some nonlinear functions.) An autoencoder, AE is then formed by composotion

$$AE = \text{dec} \circ \text{enc}.$$

And the supervised part, S, of a supervised autoencoder is formed as

$$S = \text{clf} \circ \text{enc}.$$

We train both of these two parts concurrently. The objective function that we use will be some convex combination of the two previously defined objective functions. As always let W and let $\rho \in [0, 1]$ be the weights of the supervised neural network autoencoder (all three parts of it) then the objective function is defined as

$$J(W) = \rho J_{\text{AE}}(W) + (1 - \rho) J_S(W).$$

We will call the variable ρ the reconstruction weight. It determines how much emphasis the learning algorithm should put on the reconstruction objective. $1 - \rho$ is the classification weight. It determines the emphasis on the classification objective. This approach to learning is called multitask learning.

As is the case with a classifier deep neural network, the choice of depth and widths is up to the designer to determine. Here we also add the choice of what classifier to use. We look at various choices for these hyperparameters. Unless stated otherwise, assume that clf is a logistic regression model.

3.2.2 Principal components analysis

add (PCA)

Principal components analysis (PCA) is the second reduction method we consider that considers complex features as its extracted features. PCA is an unsupervised method. PCA works by finding a set of orthogonal basis for $\mathbb{R}^d$ such that the variance of the data along each of the basis vectors is maximized. One can think of this as taking a straight line parallel to one of the orthogonal vectors and projecting all of the dataset on it. The variance we try to maximize is the variance of the projected dataset onto the straight line. The theory here is that more variance implies more interesting information. The basis

vectors are then sorted by how much variance the dataset has in a vector's direction and we take the top k vectors. Finally we project the data onto the linear subspace formed by the top k vectors. The projected data is the reduced representation of the original dataset X.

The set of orthogonal basis is found by studying the covariance matrix of X,

$$\frac{1}{n}XX^T$$

(without loss of generality, assume the variables have mean zero.) Note that this representation of the covariance matrix is made with respect to the canonical basis of $\mathbb{R}^d$. To maximize variance means to maximize the diagonal of the covariance matrix. This means that we need to diagonalize the matrix to find our sought after representation. This can be done since covariance matrices are, by definition, symmetric and hence are always diagonalizable and have orthogonal eigenvectors. Finally, the basis we want is the set of eigenvectors of the covariance matrix.

It can be shown that training a two layers autoencoder with linear activation on both layers (i.e. $g^{(1)}(x) = g^{(2)}(x) = x$) and $d^{(2)} = k$ is equivalent to finding the subspace spanned by the top $k$eigenvectors found above [8].

p-value thresholding

3.2.3 Thresholding

We will look at two thresholding methods. The first one is called p-value thresholding. It is a commonly used method in the field of bioinformatics to determine the strength of the association between a certain SNP and a certain phenotype. The method fits d logistic regression models between each of the d SNPs and the phenotype y. ~~Then a p-value is calculated to measure the goodness of the fit of logistic model.~~ That p-value is then used to quantify the strength of the association between a particular SNP and the phenotype. Then we select k SNPs with the k highest value of $-\log_{10} p$. This is called thresholding because typically, one would select some value of $p_0 \in (0, 1)$, called the threshold, and simply select all SNPs with p-values such that $-\log_{10} p \geq -\log_{10} p_0$. However, since we are interested in comparing this method to other methods, we choose a top-k approach.

new subsection: "Autoencoder weight thresholding"

Another approach we use to select features is based on the autoencoder. After training a supervised autoencoder, we look at the weights of the first layer $W^{(1)} \in \mathbb{R}^{(d^{(2)}+1)\times(d+1)}$ and form a vector,

$$a = \sum_{j=1}^{d^{(2)}+1} \left(W_{:j}^{(1)}\right)^2$$

for the i^{th} SNP. The theory here is that if a SNP is highly associated with the phenotype, then its weights on the first layer should be higher in magnitude. In other words, a_i should be higher. From this we pick the top k SNPs corresponding to the highest k values of a.

3.3 Evaluation

Since our overarching goal is to predict a phenotype, we will use performance of classifiers on reduced data to measure the efficacy of our reduction techniques in extracting relevant features. We will use three different classifiers: a support vector machine [9] with a radial basis function kernel, a logistic regression model, and a deep neural network. add description of NN

Area under the curve (AUC)

3.3.1 AUC

To measure the performance of the classifiers, we will use a metric called area under the curve (AUC.) The curve in question here is the receiver operating characteristic curve (ROC.) The ROC curve can be thought of as a parameteric curve with respect to a thresholding parameter $t \in [0, 1]$. The three methods we test output a number which we interpret as a probability of an observation falling into one of two classes. For any binary classification task, we set some threshold t on the probabilities by which we determine what to label an observation. For certain cases we require very high certainty so we only label an observation if it our models give it a very high probability of being that label so we set a relatively high threshold t. For any choice of threshold, we calculate two parameters. The true positive rate (TPR(t)) and the false positive rate (FPR(t)). Given a labeled dataset, TPR(t) determines what proportion of observations with positive label ($y = 1$ in our case) were assigned to have a positive label by the model under consideration and a threshold t. The FPR(t) is the proportion of observations with negative label

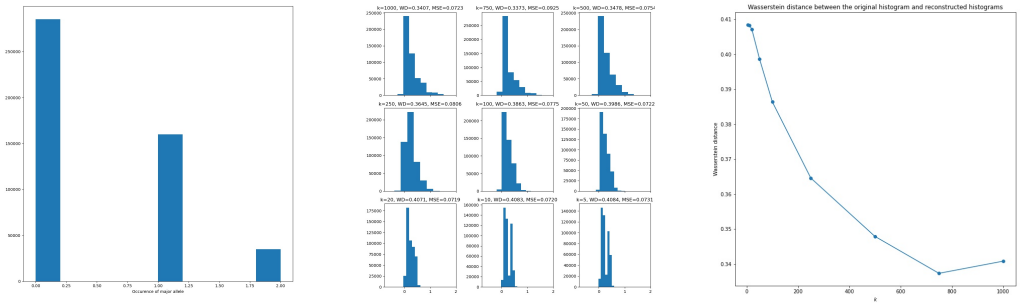

Figure 1: *Left:* Histogram of the occurence of the major allele in a simulated dataset with parameters. *Middle:* histogram of the supervised autoencoder's reconstruction of the dataset with various k values and $\rho = 0.8$ $(480, 1000, 10, 0.5)$ *Right:* Wasserstein distance from original histogram to reconstructed ones from autoencoders with size k. TODO: reproduce these plots to make them more readable and add text

($y = 0$ in our case) were assigned to have a positive label by the model and a threshold t. The ROC curve then is given by

$$\{(\text{FPR}(t), \text{TPR}(t)) \,|\, t \in [0, 1].\}$$

A perfect classifier will assign all positive observations a positive label and not assign any negative observations a positive label. This means that the ROC curve will hug the top left corner of the unit square. If we choose our classifier in such a way that it labels each observation based on the flip of a fair coin, then the ROC curve will be the diagonal line TPR = FPR. This motivates the definition of AUC. The AUC is defined to be the area under the curve of the ROC curve and above the FPR axis. For a perfect classifier where the curve hugs the top left corner of the unit square, the area under the curve is 1. The fair coin classifier will have AUC 0.5.

We will use AUC to measure the performance of the classifiers.

figure to help explain ROC/AUC

3.3.2 Precision and Recall

To understand how well our thresholding is doing, we will use two quantities called precision and recall that give us an idea of how well our thresholding is separating causal SNPs from non-causal ones. Precision and recall are typically used to assess classifiers. We treat thresholding as a classifier that assigns a positive label to quantities above (or below) the threshold and negative labels to quantities below (or above) the threshold. As was the case with the true positive rate and the false positive rate, precision and recall are functions of the thresholding parameter t. Precision is defined as

$$\text{Prec}(t) = \frac{\text{TPR}(t)}{\text{TPR}(t) + \text{FPR}(t)}.$$

It is interpreted as the proportion of all observations that have been assigned a positive label that are actually positive. Recall is defined as

$$\text{Rec}(t) = \frac{\text{TPR}(t)}{\text{TPR}(t) + \text{FNR}(t)}.$$

Where $\text{FNR}(t)$ is the rate of positive observations that are assigned a negative label. Recall is interpreted as the proportion of all positive observations that have been labeled correctly.

Describe the computations... The above methods were carried out on euler (describe the hardware, OS), python, keras versions, etc. (and give citations when needed).

4 Results

4.1 Architecture of autoencoder

move some of this to discussion First thing we want to determine in an autoencoder is how small can we make the middle layer while still preserving the integrity of the reconstructed dataset. We will first need to understand how the simulated datasets look like. On the left of figure 1 we see a histogram

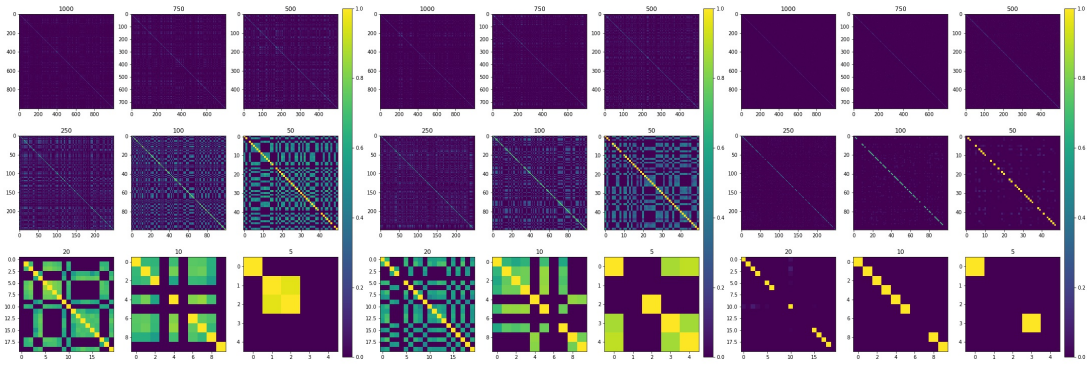

Figure 2: TODO: remake these plots correlation matrices of extracted features of ReLU activated SAE with $rho = 0.2, 0.6, and 1$ for each panel, respectively. Each panel contains eight subpanels which show the correlation matrices for values of k

of how much each possible occurence of the major allele is present at any SNP. We see that most loci (location on DNA) have 0 occurences of the major allele. Fewer loci have 1 occurence, and even fewer loci have 2 occurences. Ideally, we would want our autoencoder to recreate this structure when attempting to reconstruct the dataset. We simulate a dataset with parameters $(480, 1000, 10, 0.5)$ and train 9 supervised autoencoders with $\rho = 0.8$ and values of k ranging from 1000 and 5. It can be seen from the histograms that as k decreases (i.e. the bottleneck tightens), the autoencoder learns to just focus on reconstructing as many zeros as possible.

To quantify the observed effect, we utilize the Wasserstein distance. The Wasserstein distance is a way to measure distances between two probability distributions. It can be shown that the Wasserstein measure defines a metric on the set of Borel measures on some metric space [10] This allows us to measure the distance between the original histogram of the dataset and the autoencoder's attempt at reconstructing it. The rightmost panel of figure 1 shows the Wasserstein distance between the original histogram and the reconstruction. This confirms what we see in the middle panel. As the bottleneck size k decreases, the reconstruction becomes harder to learn.

To further understand the learned SAE, we look at the extraced features. Specifically, we look at the covariances of the extraced features and how that covariance varies with k and ρ. For this, we train a few ReLU activated SAEs with $\rho \in \{0.2, 0.6, 1\}$ and different k values on a dataset. The results shown in figure 2 show the correlation matrices of the extracted features. It can be see that as ρ approaches 1 i.e. the SAE approaches an AE, the correlation matrix tends to become more diagonal. If the SAE has learnt extract a constant feature, then the vairance of that feature will be zero and hence correlation will be undefined. When this happens we let the correlation be 0.

spell it out again (SAE)

4.2 Thresholding

To compare the two thresholding methods we look at how well they capture causal SNPs in various settings of h_2^s. To do this we utilize the precision and recall quantities defined above. We simulate 125 datasets with parameters $(480, 1000, 10, h_2^s)$ for $h_2^s \in \{0.05, 0.25, 0.5, 0.75, 1\}$ and perform both types of thresholding on each of the 25 datasets. The SAE used has $k = 300$ and $\rho = 0.3$. The k used for thresholding is taken to be $k \in \{5, 25, 100, 300\}$. Figure 4 shows an example of manhattan plots where the x axis is all of the SNPs and the y represents how strongly associated the SNP is to the phenotype. Figure 4.2 shows violin plots of precision and recall over all 25 datasets for each value of h_2^s. The straight lines mean that there was no variance in the quantity being measured.

4.3 Classification

Here we benchmark the dimensionality reduction methods on classifiers. We use the same 125 simulated datasets in section 4.2 and apply the four reduction methods discussed. We reduce the dataset to k features with $k \in \{5, 25, 100, 300\}$. Then we train three classifier models on each of the reduced representations for every k. The classifier models to be used are the same ones mentioned in 3.3

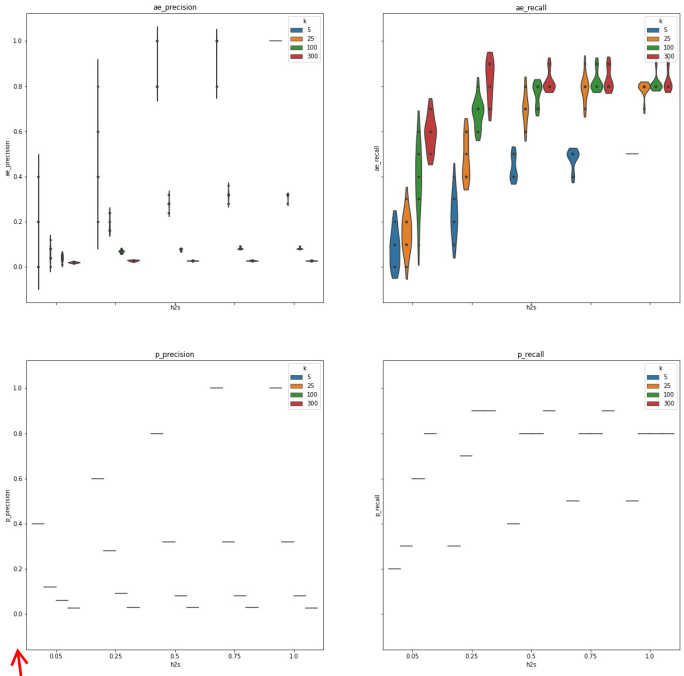

Figure 3: Violin plots of precision and recall of both thresholding methods over all 25 datasets for each value of h_2^s and k

make subtitles/labels nice and big.

expand.

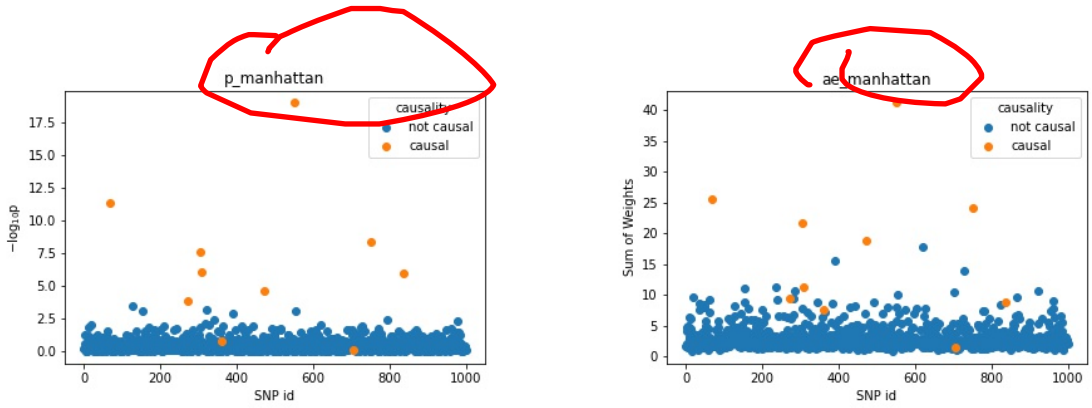

Figure 4: Example thresholding on a single dataset with $h_2^s = 0.75$ and both thresholding methods

expand

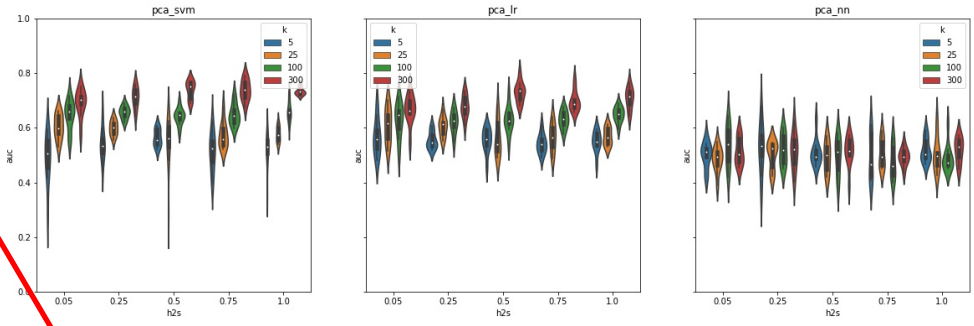

Figure 5: Classification results of three models trained after reducing dimensionality using PCA

4.3.1 Classification with PCA reduced dataset

Figure 4.3 shows the classification results of the three models trained on the projected dataset with different values of h_2^s. The SVM and logistic model seem to outperform the deep neural network. The AUC does not exceed 0.9 in all of the models. With respect to each model, the performance of the model does not change with h_2^s. Increasing the value of k improves performance.

4.3.2 Classification with SAE reduced dataset

Figure 4.3.2 shows the results of passing through the representation a SAE learns in its middle layer to a classification algorithm. SAE seems to be more sensitive to h_2^s than PCA. Each color, on average rises up as h_2^s is increased. Furthermore, in lower k's, SAE's results seem to outperform PCA's. Even though the supervised branch of is made of a logistic regression model, the seperately trained logistic model on the learnt data representation seem to have different outcome. The seperately trained logistic model has high variance in lower k's.

4.3.3 Classification with thresholded dataset

The results of thresholding are shown in figure 4.3.3. The left column shows results of p-value thresholding while the right column shows the results of the SAE thresholded one. Clearly, p-value thresholding blows both in bias (AUC) and variance.

References

[1] M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge University Press, 02 2019.

[2] H. V. Meyer and E. Birney, "PhenotypeSimulator: A comprehensive framework for simulating multi-trait, multi-locus genotype to phenotype relationships," *Bioinformatics*, vol. 34, pp. 2951–2956, 03 2018.

[3] C. A. C. Montañez, P. Fergus, A. C. Montañez, A. Hussain, D. Al-Jumeily, and C. Chalmers, "Deep learning classification of polygenic obesity using genome wide association study snps," *CoRR*, vol. abs/1804.03198, 2018.

[4] B. Yin, M. Balvert, R. A. A. van der Spek, B. E. Dutilh, S. Bohté, J. Veldink, and A. Schönhuth, "Using the structure of genome data in the design of deep neural networks for predicting amyotrophic lateral sclerosis from genotype," *Bioinformatics*, vol. 35, pp. i538–i547, 07 2019.

[5] B. Liu, Y. Wei, Y. Zhang, and Q. Yang, "Deep neural networks for high dimension, low sample size data," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, p. 2287–2293, AAAI Press, 2017.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, p. 3371–3408, Dec. 2010.

[8] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks*, vol. 2, no. 1, pp. 53–58, 1989.

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics, Springer New York, 2013.

[10] P. Clement and W. Desch, "An elementary proof of the triangle inequality for the wasserstein metric," *Proceedings of the American Mathematical Society*, vol. 136, no. 1, pp. 333–339, 2008.

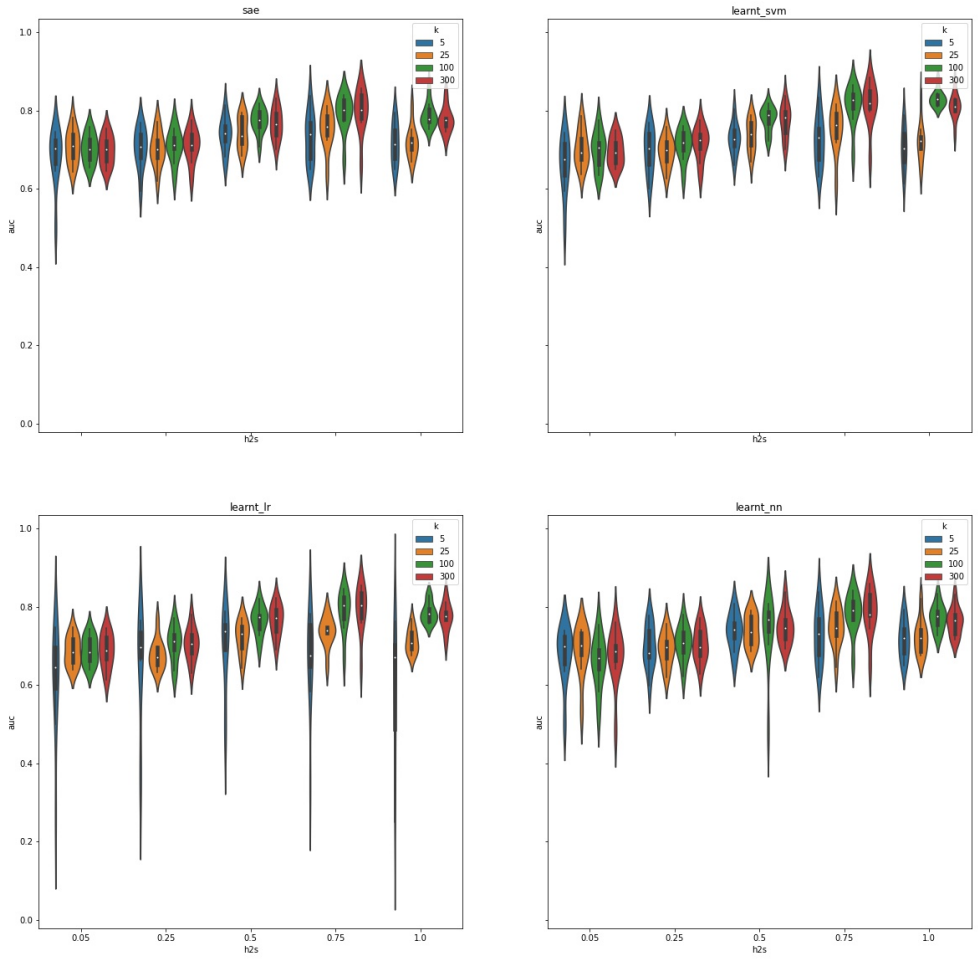

Figure 6: Classification results of three models trained after reducing dimensionality using SAE and classification results of S part of SAE.

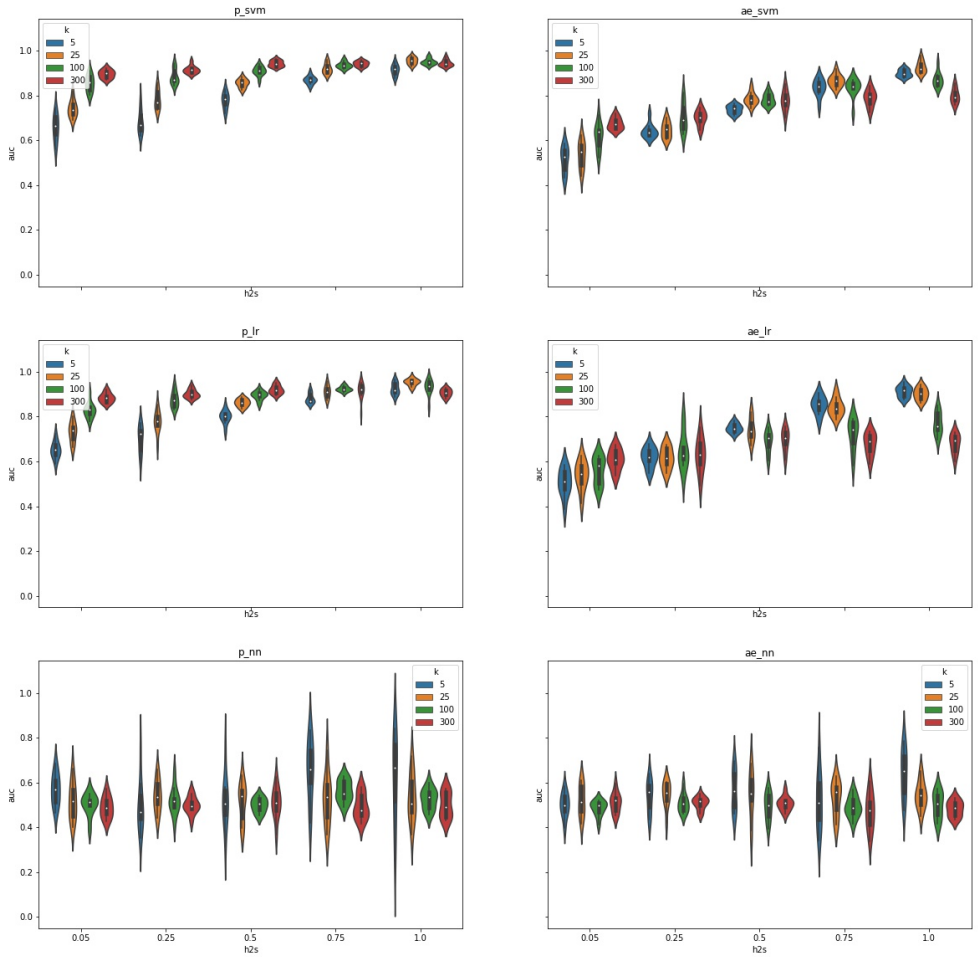

Figure 7: Classification results of three models trained after reducing dimensionality using thresholding.