Proceedings of Machine Learning Research 77:311–326, 2017 ACML 2017

Using Deep Neural Networks to Automate Large Scale Statistical Analysis for Big Data Applications

Rongrong Zhang ZHAN1602@PURDUE.EDU
Department of Statistics
Purdue University
West Lafayette, Indiana, USA
Wei Deng DENG106@PURDUE.EDU
Department of Mathematics
Purdue University
West Lafayette, Indiana, USA
Michael Yu Zhu* YUZHU@PURDUE.EDU
Department of Statistics
Purdue University
West Lafayette, Indiana, USA
Center for Statistical Science, Department of Industrial Engineering
Tsinghua University
Beijing, China

Editors: Yung-Kyun Noh and Min-Ling Zhang

Abstract

Statistical analysis (SA) is a complex process to deduce population properties from analysis of data. It usually takes a well-trained analyst to successfully perform SA, and it becomes extremely challenging to apply SA to big data applications. We propose to use deep neural networks to automate the SA process. In particular, we propose to construct convolutional neural networks (CNNs) to perform automatic model selection and parameter estimation, two most important SA tasks. We refer to the resulting CNNs as the neural model selector and the neural model estimator, respectively, which can be properly trained using labeled data systematically generated from candidate models. Simulation study shows that both the selector and estimator demonstrate excellent performances. The idea and proposed framework can be further extended to automate the entire SA process and have the potential to revolutionize how SA is performed in big data analytics.

Keywords: statistical analysis, deep network, model selection, parameter estimation, convolutional neural network, big data.

1. Introduction

According to the definitions of Gartner Release (2014) and De Maro et al. Mauro et al. (2016), big data refer to information assets with characteristics high volume, high velocity, and/or high variety, and the transformation from big data to value requires specific analytical methods. Currently, machine learning methods are used as the main tools for big data

* Corresponding should be sent to Michael Yu Zhu: yuzhu@purdue.edu

analytics, which emphasize algorithms instead of statistical analysis. Statistical Analysis (SA) is the process of deducing population properties and draw conclusions by analysis of data. Typically, the SA process consists of exploratory data analysis (EDA), model building, parameter estimation, hypothesis testing, interval estimation, prediction, and model diagnostics. Not only does the process depend on available computing system and software, it also heavily depends on the analyst. It takes a well-trained and experienced analyst to successfully conduct SA for data sets of conventional size. It is extremely challenging to do so for large scale data analysis. This is one of the primary reasons that SA has not been widely used for analyzing big data.

We use multiple regression analysis as an illustrative example of the SA process. In particular, we focus on the first three steps including EDA, model building, and parameter estimation. Suppose there is a data set of one response variable Y and p explanatory variables $X_1, X_2, \ldots, X_p$, and it is of interest to investigate the relationship between Y and $X_1, \ldots, X_p$. In the EDA step, summary statistics, empirical distributions, pairwise correlations, and scatter plots will be calculated or generated. The analyst will then need to check the values and graphics for patterns, relations, and suspicious data points. Based on the EDA results, the analyst will propose a proper regression model. After that, statistical principles such as the least squares principle or the maximum likelihood principle are used to compute the estimates of the model parameters.

Deep neural networks (DNNs) have achieved remarkable performances in a variety of applications including competitive game play, object recognition, machine translation, and speech recognition in recent years, and promise to deliver artificial intelligence (AI) in almost every aspect of human life and society. We believe that DNNs can also be used to automate the SA process, bring AI to large scale statistical analysis, and make SA popular for big data analytics.

Due to the complexity of the SA process and space limitation, we will narrow down to two most important SA procedures, which are model building and parameter estimation, and propose to use convolutional neural networks (CNN) LeCun et al. (1998) for their automation. Conventionally, model building is either done by the analyst based on the results of exploratory data analysis and his/her own knowledge, or it is done via model selection using statistical principles and criteria. Both of these two approaches are however difficult to automate. Instead, we first reformulate model building as a general model selection problem, and further propose to construct a CNN to perform model selection. We refer to the resulting CNN as the *neural model selector*. Data systematically simulated from candidate models will be used to train the neural model selector.

Similarly, conventional statistical approaches for parameter estimation such as least squares estimation and maximum likelihood estimation are also difficult to automate. In this paper, we again propose to construct a CNN to perform parameter estimation. Data systematically simulated from the underlying distribution are used to train this CNN. We refer to the resulting CNN as the *neural parameter estimator*. The idea of using neural networks to estimate parameters in a stochastic model is not entirely new, but it is neither well-known nor a common practice in the statistical community. We will give a brief literature review of this idea in the next section.

We further explore the possible interplay between the neural model selector and the neural parameter estimator. As will be discussed in the Proposed Approaches section, the two CNNs can be entirely separate, almost identical, or partially joint, leading to different performances in training as well as in application. We carry out extensive simulation studies, and show that the proposed neural model selector and parameter estimator can be properly trained, and the trained CNNs demonstrate excellent performance in test data and a real data application. The idea and proposed framework can be further extended to the entire SA process with the potential to change how SA is done in conventional data analysis and big data analytics.

2. Related work

There exists an extensive statistical literature on model selection Bozdogan (1987); Burnham and Anderson (2003, 2004). Numerous model selection methods have been proposed. Some of these methods are not applicable to the setting we consider in this paper, while others, though applicable, may run into various difficulties; see discussions in Section 3 for details. To our best knowledge, there is no prior work about redefining the model selection problem as a machine learning classification problem and training CNN to learn and perform model selection with labeled simulated data.

There also exist a variety of statistical methods for parameter estimation in the literature; see Casella and Berger (2002); Huber et al. (1964); Norton et al. (2010). Most statistical methods rely on full or partial knowledge of the model and are based on statistical principles. After conducting intensive literature search, we only found one paper Xie et al. (2007), in which the authors proposed to use artificial neural networks and simulated data to construct estimates for parameters of a stochastic differential equation. However, the idea of using CNNs and simulated data to automate parameter estimation and model selection and bring AI to the general SA process appears to be novel to our best knowledge.

3. Proposed approach

As discussed in the Introduction, we first reformulate model building and parameter estimation as a machine learning problem. Suppose $\mathcal{M} = \{M_k : 1 \leq k \leq K\}$ be a collection of K prespecified models/distributions. Let $f(y|\theta_k, M_k)$ be the density function of model M_k, where $\theta_k \in \Theta_k$ is the scalar parameter of the density function. Assume that we have a random sample from one of the models, which is $\{y_j\}_{1\leq j\leq N}$, but we do not know the data-generating model and its parameter. The goal of statistical analysis is to identify the model and further estimate the model parameter.

To achieve the analysis goal stated above, conventionally, the statistician will employ various model selection methods together with some estimation methods. Here, we will briefly discuss several representative approaches, which include the Kolmogorov-Smirnov (KS) distance Chakravarti et al. (1967), Bayesian Information Criterion (BIC) Schwarz (1978), and Bayes factor Kass and Raftery (1995). The KS distance method calculates the distance between the population Cumulative Distribution Function (CDF) and the empirical CDF based on the sample $\{y_j\}$ for each model. The model that achieves the minimum distance will be selected as the true model. The BIC criterion calculates the BIC score for each model as follows:

$$\mathrm{BIC}(M_k) = -2\log L(\hat{\theta}_k) + \log(n)p$$

where $L(\cdot)$ is the likelihood function, $\hat{\theta}_k$ is the maximum likelihood estimate, and p is the number of parameters in the model M_k. Note that for the scenario considered here, $p = 1$. The model that achieves the minimum BIC score will be selected as the true model.

The Bayes factor method will impose a prior distribution to the models, $\pi(M_k)$, and further impose a prior distribution to the parameter $\pi(\theta_k)$. Then, given the sample, the posterior distribution for each model can be calculated, which is denoted as $\pi(M_k|\{y_j\})$. The Bayes factor between any two models, M_{k_1} and M_{k_2}, can be calculated as $\mathrm{BF}(M_{k_1}, M_{k_2}) = \pi(M_{k_1}|\{y_j\})/\pi(M_{k_2}|\{y_j\})$, which can be used to discriminate between the two models. The model the BF scores support the most will be selected as the true model.

Our criticism for the conventional statistical approaches discussed above is two-fold. First, for the goal of automating model selection, the model set $\mathcal{M}$ usually consists of a large number of candidate models, and the models are of huge variety. The conventional statistical methods such as the KS distance and BIC only work for selection between nested or other well-structured models. Second, for a given sample, all the conventional methods will have to calculate a score for each of the candidate models, and then compare them to pick the winner. This can become computationally intensive or even intractable, especially for the Bayes factor approach. Similar discussion and criticism can be applied to using conventional statistical methods for automating parameter estimation, which we omit them due to space limitation.

In this section, we instead propose to use CNNs and machine learning to automate model selection and parameter estimation. Our main idea is that the procedures for model selection and parameter estimation can be considered mappings from the sample to a model and a value of the model parameter, that is,

$$G : \{y_j\} \to \begin{pmatrix} G_1(\{y_j\}) \\ G_2(\{y_j\}) \end{pmatrix} \in \mathcal{M} \times \Theta$$

where $G = (G_1, G_2)$ consists of the model selection mapping G_1 and the parameter estimation mapping G_2, and Θ is the parameter space. Instead of using statistical principles to derive G_1 and G_2, we propose to use CNNs to approximate them. From here on in the rest of the paper, we refer to G_1 as the neural model selector, and G_2 the neural parameter estimator, as discussed in the Introduction.

3.1. Labeled data and loss functions

As an analogy, the sample $\{y_j\}$ can be considered an image of an object, G_1 the classifier for object recognition, and G_2 the regression procedure for object localization in image processing Girshick et al. (2014). In order to train G_1 and G_2, just like in image processing, labeled data must be available. We propose to generate the labeled data as follows.

Let N be a prespecified sample size. For each model M_k, we first place an equally space grid over the parameter space Θ_k, which is $\{\theta_{k,1}, \theta_{k,2}, \cdots, \theta_{k,n_k}\}$. For each value of the grid $\theta_{k,l}$, we generate D samples of size N from $f(y|\theta_{k,l}, M_k)$. We denote the samples as $\{y_{(k,l,d)j}\}_{1\le j\le N}$ for $1 \le k \le K$, $1 \le l \le n_k$, and $1 \le d \le D$. In total, we have the following collection of labeled data $Y = \left\{ \left(\{y_{(k,l,d)j}\}_{1\le j\le N}, M_k, \theta_{k,l} \right)_{k,l,d} \right\}$, and Y will be used to train and validate both G_1 and G_2.

In order to train G_1 and G_2, we need to choose proper loss functions. As mentioned previously, the neural model selector is essentially a classifier and similar to the classifier for object recognition. Therefore, we choose the commonly used softmax loss function for training G_1. For details of the softmax loss function, the reader is referred to Bishop (2006). The neural parameter estimator G_2 is essentially a regression function and similar to the regression CNN for localizing an object in an image. For object localization, the L_2 loss is typically used, which is $L(G_2, \theta) = \|G_2 - \theta\|_2^2$. The L_2 loss function works well for image processing. For the neural parameter estimator, the L_2 loss is sensitive to extreme observations generated from models with long tails in $\mathcal{M}$, which makes the training process unstable. Resolve this issue, the Huber loss Huber et al. (1964) is employed in training of neural estimator to improve the robustness against outliers. The Huber loss is defined as follows:

$$L_\delta(\theta, \hat{\theta}) = \begin{cases} \frac{1}{2}(\theta - \hat{\theta})^2 & \text{for } |\theta - \hat{\theta}| \le \delta, \\ \delta(|\theta - \hat{\theta}| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases}$$

3.2. Two types of architectures

The last important issue is about the architectures of the neural model selector and parameter estimator. There are two different types of architectures involved. The first type is regarding the architectures of the CNNs, which are about the numbers and sizes of the covolutional and fully connected layers. We refer to this type of architecture as the CNN architecture. The second type of architectures is regarding the interplay between the model selector G_1 and the parameter estimator G_2. Because this type of architecture directly affects how the overall analysis performed, we refer to it as the SA architecture.

There are three possible SA architectures. The first SA architecture uses two separate CNNs for the model selector and the parameter estimator, respectively, which we refer to as the Non-Shared Architecture (NSA). The second SA architecture uses one single CNN for both G_1 and G_2, and they part their ways only at the output layer. We refer to this architecture as the Fully Shared Architecture (FSA). The third architecture uses two partially joint CNNs for G_1 and G_2, respectively. The two CNNs can share from one to all common convolutional and fully connected layers. We refer to this architecture as the Partially Shared Architecture (PSA). The PSA sharing l layers is denoted as PSA-l. The three SA architectures NSA, FSA, and PSA are illustrated in Figure 1.

NSA, FSA, and PSA have their own advantages and disadvantages. Using again the analogy of object recognition and localization, the convolutional layers are used to learn features that can be efficiently and effectively used for identifying the true model and its parameter. When NSA is used, the two separate CNNs are learning the features for model selection and parameter estimation, separately, and this simple SA architecture allows easy implementation of the training algorithms. The disadvantage of NSA is that it uses only the marginal distribution of model label and true parameter value separately, instead of the entire information in their joint distribution.

When FSA is used, the single CNN is trying to learn the same set of features, and hope that they can be used to not only select the model correctly but also estimate the parameter accurately. This is based on the assumption that such a set of common features exists. This assumption holds for distributions that belong to the Exponential family, and

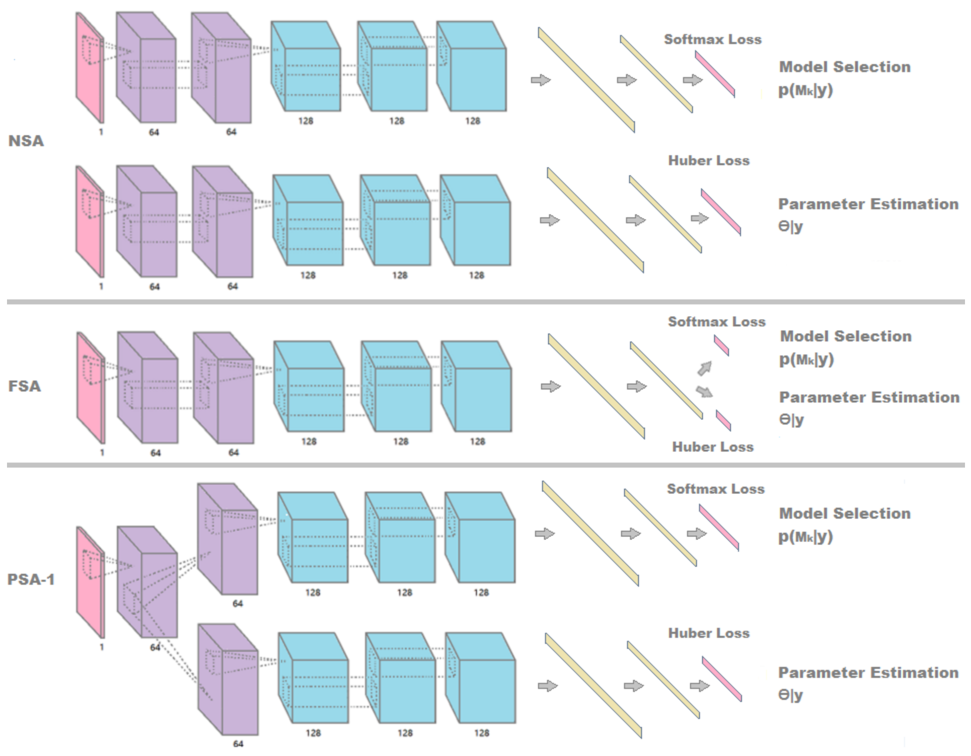


Figure 1: SA Architecture, from top to bottom: NSA, FSA, PSA-1

minimal sufficient statistics can serve as the set of common features. This assumption however may not hold in general. Therefore, FSA is expected to work well under one set of candidate models, but may fail under another set of candidate models.

The most promising architecture is PSA. The intuition underlying PSA is that the early convolutional layers will produce low-level features that are common for both model selection and parameter estimation, and information in the true model label and parameter values can be shared. Because model selection and parameter estimation are two different tasks, we should not expect they would be relaying on the same set of high-level features. Our simulation studies reported in later sections support this intuition. In terms of training, PSA is more demanding than the other two architectures. Furthermore, PSA leads to another important issue, that is, how many convolutional layers should be shared by G_1 and G_2. We will investigate this issue in the next section.

4. Simulation results

We conduct simulation studies to demonstrate the properties and performance of the proposed model selector and parameter estimator in this section, and further compare them with several conventional statistical methods. Due to space limitation, we will emphasize on results regarding model selection instead of parameter estimation. The latter can be found in the Supplementary Document.

4.1. Setup and datasets

The difficulty of model selection and parameter estimation depends on the number of candidate models (K) in $\mathcal{M}$. We consider three levels of K, which are $5, 20$ and 50. We take 50 probability distributions from the textbook Casella and Berger (2002) and some R packages Hennig (2012); Johnstone et al. (2014); Statisticat and LLC. (2016); Swihart and Lindsey (2016); Yee (2017). The 50 distributions are listed in Table S1 in the Supplementary Document. When $K = 5$, the set of candidate models $\mathcal{M}$ includes the top five distributions in the list; when $K = 20$, $\mathcal{M}$ includes the top 20 distributions in the list; and when $K = 50$, $\mathcal{M}$ includes all the 50 distributions.

The performance of the proposed model selector and parameter estimator depends on the sample size. We consider three levels of the sample size N, which are $100, 400$, and 900. Each sample will be resized to a square matrix to feed into CNNs. According to the data-generating scheme described in Section 3.1, the total amount of training samples further depends on the number of parameter values on the grid (n_k) and the number of replicated samples (D). We specify $D = 1000$, and grid size n_k is set to be between 10 and 12. For each distribution, if the parameter space is bounded, like probability p in Bernoulli distribution, we will place the grid on the original space. If the parameter space is not bounded, like μ in Normal distribution, we will set the parameter space to be a bounded interval, $[0, 12]$. Following the scheme of Section 3.1, we generate all the labeled data, 80% of which is used for training, and the other 20% is used for validation. Note that we use the definitions given in Ripley (2007): a training set is used for learning, a validation set is used to tune the network parameters, and a test set is used only to assess the performance of the network.

We further generate the test data as follows. For each model, we first randomly sample 100 parameter values from the parameter space, those values are just in the same range as parameters in the training set, but not the same; and second, for each sampled parameter value, we generate 10 random samples of size N. We test the trained model selector and parameter estimator using the test datasets. Counting both the labeled data and test data, in total, we have generated roughly 400 thousand training samples, 100 thousand validation samples, and 50 thousand test samples for the 50 models.

4.2. Architecture setup and training

In Section 3.2, we discussed the three possible SA architectures (NSA, FSA, and PSA), but have not discussed the CNN architectures. In our simulation studies, we employ three different sizes of CNNs, which are referred to as small, medium, and large, respectively. The small CNN architecture consists of three convolutional layers with 64, 128 and 128 filters, respectively, which are followed by two fully-connected layers with 512 and 256 neurons, respectively. The medium CNN architecture consists of five convolutional layers with 64 filters each, which are followed by two fully connected layers each with 64 neurons. The large CNN architecture consists of five convolutional layers with 64, 64, 128, 128 and 128 filters, respectively, which are followed by two fully connected layers with 1024 and 512 neurons, respectively. In all three CNN architectures, convolutional filters are connected to a 5×5 region of their input, 2×2 max pooling and 2×2 average pooling are performed between some consecutive convolutional layers. The same CNN architecture is used for both the neural model selector and parameter estimator except for the output layers.

Under each combination of SA architectures (NSA, FSA, PSA), CNN architectures (small, medium, large), number of candidate models ($K = 5, 20, 50$), sample sizes ($N = 100, 400, 900$), we use the generated labeled data to train, validate, and test the proposed neural selector and parameter estimator. For PSA, we further vary the number of shared layers (l) in training. Each training run is replicated six times to assess the stability of the training procedure and results. All training is performed using the Caffe implementation Jia et al. (2014) on one GTX-1080 GPU. The running time each training run takes ranges from five minutes to one hour depending on the values of K and N.

4.3. Performance of neural selector and estimator

Due to space limitation, we report more detailed results of our simulation studies in the Supplementary Document and only select part of them to report in this section. Overall, the trained model selector and parameter estimator demonstrate excellent performances.

Accuracy of the model selector Table 1 presents the performance of the model selector on the test dataset under all the combinations of SA architecture, CNN architecture, number of candidate models K, and sample size N. The selection accuracy together with standard deviation in parentheses based on repeated six runs are reported, the better result between NSA and PSA SA architectures is denoted as bold. For PSA, we report the best results based on layer analysis, they are PSA-3, PSA-2, and PSA-5 for small, medium, and large CNN architectures, respectively.

The table shows that the selection accuracy decreases as K increases under fixed N, and the accuracy increases as we have larger sample size. When we have a moderate sample size, 400, the partially shared CNN architecture can achieve more than 90% selection accuracy. In order to maintain high accuracy for large number of candidate models, large sample sizes should be used.

Figure 5 in Appendix shows the confusion matrix based on large CNN and PSA-5 neural model selector on test dataset with $K = 20$ distributions. The performance of the parameter estimator on the test dataset under different scenarios is reported in Table 3 in Appendix. Figure S1, S5-S9 in Supplementary Document show the scatter plots of true parameter values and predicted values estimated by parameter estimator under different architectures. Overall, the large CNN PSA-5 parameter estimator performs the best.

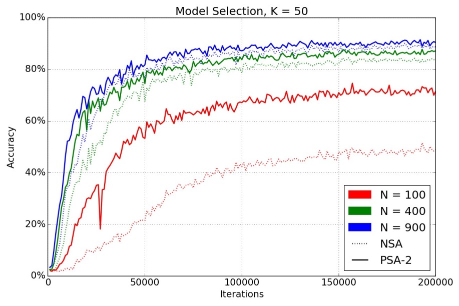
Impact of SA architectures on learning rate Figure 2 is used to compare the impacts of the NSA and PSA-l SA architectures on the learning rates of the model selector and the parameter estimator, respectively. The medium and large CNN architectures are used, and all the three sample sizes are considered. The upper panel is for medium CNN architecture while the lower panel is for large CNN architecture. The upper left panel of Figure 2 plots the accuracy of the model selector evaluated on the validation dataset against the number of iterations during the training process, whereas the upper right panel plots the log Huber loss of the parameter estimator. Solid curves are for the PSA-2 SA architecture, and dotted curves are for the NSA architectures. It is clear from the plots that the learning rate under PSA-2 is faster than that under NSA, indicating that sharing convolutional layers between

Table 1: Model selection results under all the combinations of SA architecture, CNN architecture, number of candidate models K, and sample size N.

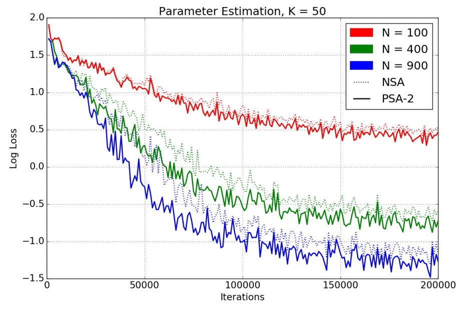
	CNN architecture	$N = 100$		$N = 400$		$N = 900$	
		NSA	PSA	NSA	PSA	NSA	PSA
$K = 5$	small	96.88% (0.12%)	**96.92%** (0.11%)	97.68% (0.19%)	**97.78%** (0.13%)	97.98% (0.13%)	**98.01%** (0.07%)
	medium	96.06% (0.29%)	**96.62%** (0.21%)	97.68% (0.25%)	**97.93%** (0.16%)	97.85% (0.08%)	**97.90%** (0.16%)
	large	**97.30%** (0.19%)	97.13% (0.13%)	**97.88%** (0.08%)	97.77% (0.08%)	98.01% (0.13%)	98.01% (0.17%)
$K = 20$	small	90.76% (0.17%)	**91.44%** (0.20%)	96.34% (0.33%)	**96.59%** (0.27%)	97.79% (0.20%)	**97.81%** (0.24%)
	medium	67.73% (3.09%)	**88.98%** (0.86%)	95.11% (0.47%)	**96.07%** (0.46%)	97.78% (0.14%)	**98.03%** (0.08%)
	large	92.18% (0.23%)	**92.53%** (0.35%)	97.09% (0.23%)	**97.19%** (0.32%)	98.37% (0.29%)	**98.47%** (0.14%)
$K = 50$	small	73.33% (0.54%)	**74.00%** (0.88%)	86.34% (0.29%)	**86.52%** (0.59%)	**90.10%** (0.59%)	90.00% (0.35%)
	medium	48.93% (1.92%)	**71.61%** (0.66%)	83.86% (0.37%)	**87.21%** (0.41%)	88.72% (0.25%)	**90.38%** (0.34%)
	large	75.77% (0.64%)	**78.19%** (0.48%)	88.58% (0.50%)	**88.98%** (0.31%)	91.08% (0.28%)	**91.11%** (0.42%)

the model selector and parameter estimator can expedite their training rates. Similar patterns could be found in other scenarios as showed in Figure S2, S3, S4 in Supplementary Document.

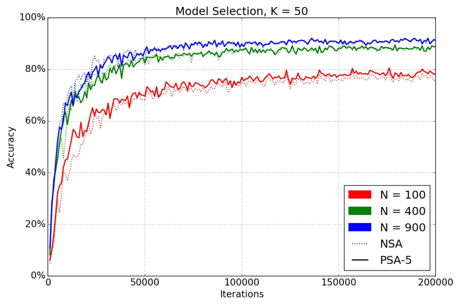
How many layers should be shared? Figure 3 shows the impact of the number of shared layers between the model selector and parameter estimator on their performances. We consider the scenario with $K = 50$, $N = 100$, and the medium and large CNN architectures, and vary the SA architectures from NSA to FSA. The left panel of Figure 3 presents the boxplots of accuracy of the model selector under various SA architectures, whereas the right panel presents the boxplots of the Huber loss of the parameter estimator. In terms of model selection accuracy, for medium CNN architecture, PSA-1 shows significant improvement over NSA, and PSA-2 further improves upon PSA-1, though the amount of improvement from PSA-1 to PSA-2 is small. PSA-3 performs almost the same as PSA-2, and further increasing the number of shared layers leads to slight decrease in selection accuracy. In terms of estimation accuracy (i.e. Huber loss), we can observe similar patterns as the



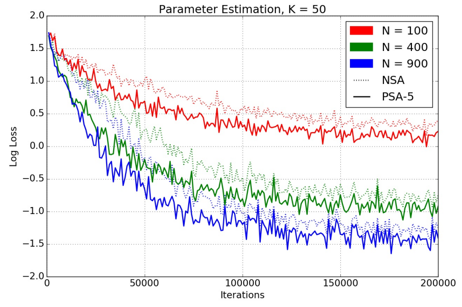
(a)



(b)



(c)



(d)

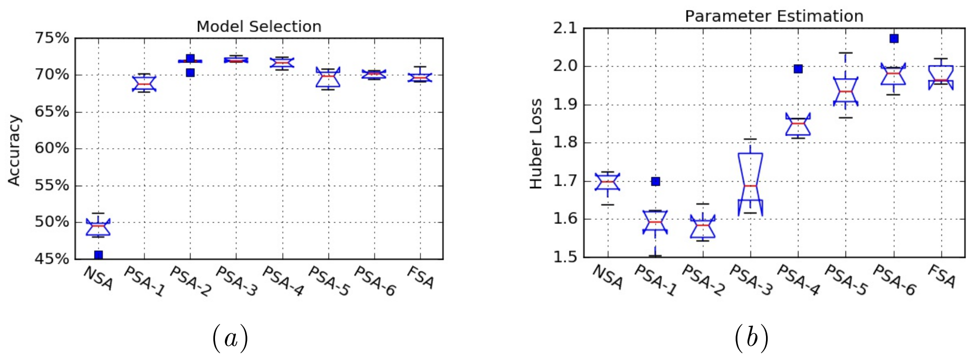
Figure 2: Comparison between NSA and PSA-l neural model selector and parameter estimator, different colours denote for different sample sizes, upper panel is for medium CNN architecture and lower panel is for large CNN architecture.

selection accuracy for NSA, PSA-1 and PSA-2. As the number of shared layers further increases, the estimation accuracy declines fairly fast. The results suggest that the PSA-2 SA architecture is optimal for both of the model selector and parameter estimator for the medium CNN architecture. For the large CNN architecture, the optimal SA architecture turns out to be PSA-5 instead.

4.4. Comparison with conventional methods

We apply the three conventional model selection methods, the KS distance, BIC, and Bayes factor to the test datasets under the scenario with $K = 20$, and compare their performances with that of the trained neural model selector. Table 2 reports the accuracy of the three statistical methods as well as the trained neural model selector under various sample sizes. From the table, it is clear that the neural model selector outperforms the three statistical methods by a significant margin.

In terms of accuracy in parameter estimation, conventional statistical estimators and the proposed neural estimator are not directly comparable. The former is based on the knowledge of the model, whereas the latter does not assume the underlying model is known. If the model is known, then statistical methods such as the maximum likelihood estimation (MLE) method are shown to enjoy certain optimality. For example, MLEs are asymptotically most efficient under some regularity conditions. If the model is unknown, then most



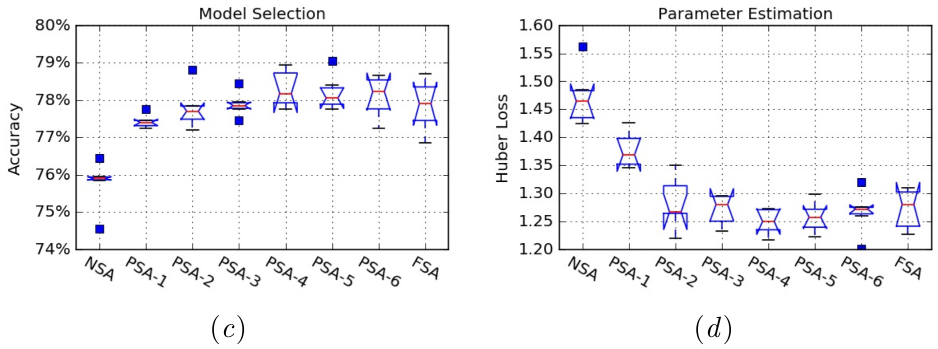


Figure 3: Information sharing comparison for medium and large CNN architectures, $K = 50$ and $N = 100$. The upper panel is for medium CNN architecture and the lower panel is for large CNN architecture.

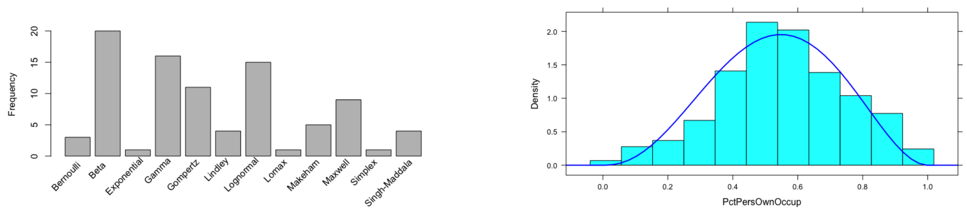
conventional statistical methods are not applicable, but the neural parameter estimator can still work well.

Table 2: Comparison of model selection methods on model set with $K = 20$.

	$N = 100$		$N = 400$		$N = 900$	
	Top-1	Top-2	Top-1	Top-2	Top-1	Top-2
KS distance	72.5%	83.2%	83.3%	85.0%	84.7%	85.0%
BIC	69.9%	74.6%	74.7%	75.0%	75.0%	75.0%
Bayes factor	75.5%	84.8%	77.8%	83.3%	70.0%	75.0%
Neural selector	92.1%	99.2%	96.4%	99.7%	97.9%	99.7%

5. Neural selector for models with covariates

In the Introduction, we propose to use AI powered by deep neural networks to automate the SA process. In the previous sections, we develop the neural model selector and parameter estimator targeting only univariate models with single parameter. Our idea and proposed



(a) (b)

Figure 4: Real data application

framework can be extended to handle more sophisticated models. In this section, we extend the neural selector to a group of commonly used simple regression models.

Let the model set $\mathcal{M}$ include the following seven regression models: simple linear regression model, Poisson regressoin model, Logistic regression model, Negative Binomial regression model, Lognormal regression model, Loglinear regression model, and multinormial regression model. Let $\{(y_j, x_j)\}_{1 \le j \le N}$ be a sample generated from one of the seven model. As before, the neural model selector is a CNN-based classifier that maps the sample to its generating model, and we will use labeled data systematically generated from the seven models to train this neural model selector.

The labeled data are generated as follows. For each regression model, we place an evenly spaced grid over its parameter space. For each vector of the parameter values on the grid, 1000 samples with sample size N are randomly drawn from the model. The generated data are further partitioned into 70% for training, 20% for validation, and 10% for test. We use the medium CNN architecture, employ the Caffe to train the model selector, and further test the performance of the trained selector on the test dataset. The results show that the trained model selector can achieve 87.86% in accuracy when the sample size is 100, and can achieve 97.86% in accuracy when the sample size is 400.

6. Real data example

To demonstrate the applicability of our proposed methods, we use the neural model selector trained under the setting of $K = 50$, $N = 900$, large CNN, and PSA-5 to explore a real data set called the Communities and Crime Data Set from UC Irvine's machine learning repository (see http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime). The data set originally contains 1994 instances and 128 attributes. After we exclude the categorical variables and the variables with missing values, there are 90 real-valued predictor variables left. For each of those 90 variables, we randomly draw a sub-sample of 900 observations, apply the trained neural model selector to the sub-sample, and produce the result. We summarize the model selection results in the left panel of Figure 4 for all 90 variables. We demonstrate the estimation result for the variable named PctPersOwnOccup (percent of people in owner occupied households) in the right panel of Figure 4. The blue line is the density function with the estimated parameter value, while the histogram is based on the observations.

7. Conclusion and future work

In this paper, we have proposed and further developed the neural model selector and parameter estimator to automate model selection and parameter estimation, which are two major tasks in the SA process. Simulation study shows that the neural selector and estimator can be properly trained with simulated labeled data, and further demonstrate excellent performance. We consider this work a demonstration of the validity of our grand proposal that is to use DNNs to automate the entire SA process. There remains a lot of work we need to do before the grand proposal can be finally materialized.

First, we will extend the neural model selector and parameter estimator to models with multiple parameters as well as regression models involving a large number of explanatory variables. Second, we will investigate how CNNs or other DNNs can be used to automate other tasks such as hypotheses testing and diagnostics of the SA process in the near future. Our ultimate goal to develop AI systems or software that can conduct principled SA for big data analytics without the need of human interventions.

References

Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

Hamparsum Bozdogan. Model selection and akaike's information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, 1987.

Kenneth P Burnham and David R Anderson. *Model selection and multimodel inference: a practical information-theoretic approach.* Springer Science & Business Media, 2003.

Kenneth P Burnham and David R Anderson. Multimodel inference understanding aic and bic in model selection. *Sociological methods & research*, 33(2):261–304, 2004.

George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

I.M. Chakravarti, R.G. Laha, and J. Roy. *Handbook of methods of applied statistics*. Number v. 1 in Wiley series in probability and mathematical statistics. Wiley, 1967. URL https://books.google.com.hk/books?id=vtI-AAAAIAAJ.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Christian Hennig. *smoothmest: Smoothed M-estimators for 1-dimensional location*, 2012. URL https://CRAN.R-project.org/package=smoothmest. R package version 0.1-2.

Peter J Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

Iain M. Johnstone, Zongming Ma, Patrick O. Perry, and Morteza Shahram. *RMTstat: Distributions, Statistics and Tests derived from Random Matrix Theory*, 2014. R package version 0.3.

Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Andrea De Mauro, Marco Greco, and Michele Grimaldi. A formal definition of big data based on its essential features. *Library Review*, 65(3):122–135, 2016. doi: 10.1108/LR-06-2015-0061. URL http://dx.doi.org/10.1108/LR-06-2015-0061.

J. Norton, E. Walter, and L. Pronzato. *Identification of Parametric Models: from Experimental Data*. Communications and Control Engineering. Springer London, 2010. ISBN 9781849969963. URL https://books.google.com.hk/books?id=BFmkcQAACAAJ.

Gartner Press Release. Gartner says the internet of things will transform the data center. *Retrieved from http://www.gartner.com/newsroom/id/2684616*, 2014.

Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

Gideon Schwarz. Estimating the dimension of a model. *Ann. Statist*, 6(2):461–464, 03 1978. doi: 10.1214/aos/1176344136. URL http://dx.doi.org/10.1214/aos/1176344136.

Statisticat and LLC. *LaplacesDemon: Complete Environment for Bayesian Inference*, 2016. URL https://web.archive.org/web/20150206004624/http://www.bayesian-inference.com/software. R package version 16.0.1.

Bruce Swihart and Jim Lindsey. *rmutil: Utilities for Nonlinear Regression and Repeated Measurements Models*, 2016. URL https://CRAN.R-project.org/package=rmutil. R package version 1.1.0.

Z Xie, D Kulasiri, S Samarasinghe, and C Rajanayaka. The estimation of parameters for stochastic differential equations using neural networks. *Inverse Problems in Science and Engineering*, 15(6):629–641, 2007.

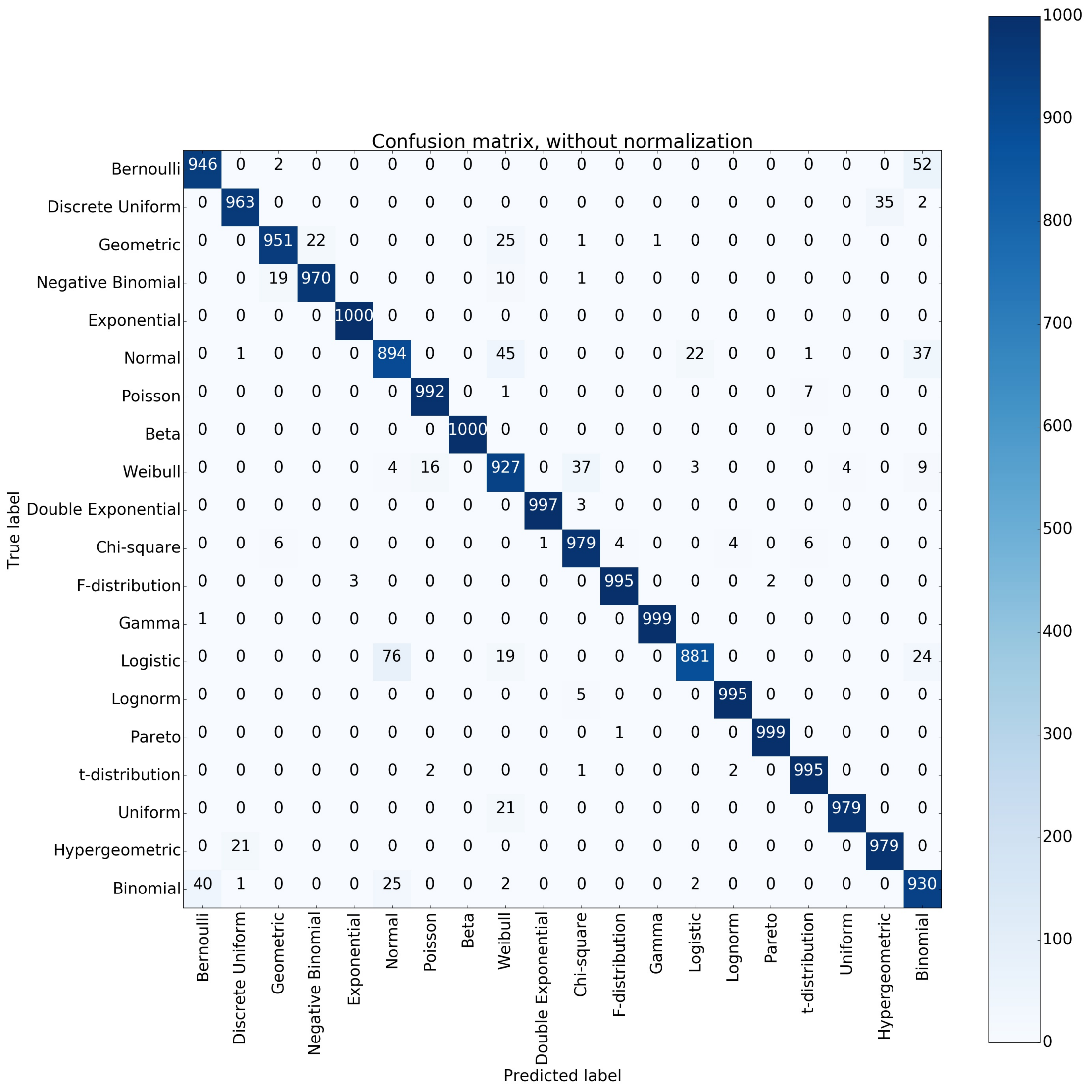
Thomas W. Yee. *VGAM: Vector Generalized Linear and Additive Models*, 2017. URL https://CRAN.R-project.org/package=VGAM. R package version 1.0-3.

Appendix A.

Table 3: Parameter estimation results under all the combinations of SA architecture, CNN architecture, number of candidate models K, and sample size N. The Huber Loss with standard deviation in parentheses based on six repeated runs are reported, the better result between NSA and PSA SA architectures is denoted as bold. For PSA, we report the best results based on layer analysis, they are PSA-3, PSA-2, and PSA-5 for small, medium, and large CNN architectures respectively. We can see that PSA performs better than NSA in most cases.

	CNN architecture	$N = 100$		$N = 400$		$N = 900$	
		NSA	PSA	NSA	PSA	NSA	PSA
$K = 5$	small	0.074 (0.0024)	**0.072** (0.0041)	0.022 (0.0008)	0.022 (0.0004)	**0.014** (0.0007)	0.015 (0.0003)
	medium	**0.071** (0.0011)	0.072 (0.0032)	0.020 (0.0003)	0.020 (0.0003)	0.012 (0.0003)	0.012 (0.0002)
	large	0.068 (0.0022)	**0.067** (0.0020)	0.019 (0.0002)	**0.018** (0.0003)	**0.011** (0.0002)	0.012 (0.0001)
$K = 20$	small	0.887 (0.0310)	**0.830** (0.0261)	0.285 (0.0097)	**0.243** (0.0090)	0.154 (0.0106)	**0.130** (0.0077)
	medium	**0.851** (0.0274)	0.864 (0.0216)	0.223 (0.0105)	**0.207** (0.0103)	0.111 (0.0043)	**0.102** (0.0032)
	large	0.752 (0.0472)	**0.732** (0.0431)	0.203 (0.0076)	**0.187** (0.0043)	0.110 (0.0045)	**0.099** (0.0039)
$K = 50$	small	2.056 (0.0724)	**1.734** (0.0344)	0.91 (0.1323)	**0.636** (0.0137)	0.654 (0.0483)	**0.428** (0.0176)
	medium	1.691 (0.0314)	**1.596** (0.0521)	0.561 (0.0169)	**0.484** (0.0153)	0.313 (0.0049)	**0.282** (0.0132)
	large	1.472 (0.0505)	**1.258** (0.0273)	0.480 (0.0114)	**0.399** (0.0062)	0.288 (0.0313)	**0.246** (0.0051)

Appendix B.



	Bernoulli	Discrete Uniform	Geometric	Negative Binomial	Exponential	Normal	Poisson	Beta	Weibull	Double Exponential	Chi-square	F-distribution	Gamma	Logistic	Lognorm	Pareto	t-distribution	Uniform	Hypergeometric	Binomial
Bernoulli	946	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52
Discrete Uniform	0	963	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	2
Geometric	0	0	951	22	0	0	0	0	25	0	1	0	1	0	0	0	0	0	0	0
Negative Binomial	0	0	19	970	0	0	0	0	10	0	1	0	0	0	0	0	0	0	0	0
Exponential	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Normal	0	1	0	0	0	894	0	0	45	0	0	0	0	22	0	0	1	0	0	37
Poisson	0	0	0	0	0	0	992	0	1	0	0	0	0	0	0	0	7	0	0	0
Beta	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0
Weibull	0	0	0	0	0	4	16	0	927	0	37	0	0	3	0	0	0	4	0	9
Double Exponential	0	0	0	0	0	0	0	0	0	997	3	0	0	0	0	0	0	0	0	0
Chi-square	0	0	6	0	0	0	0	0	0	1	979	4	0	0	4	0	6	0	0	0
F-distribution	0	0	0	0	3	0	0	0	0	0	0	995	0	0	0	2	0	0	0	0
Gamma	1	0	0	0	0	0	0	0	0	0	0	0	999	0	0	0	0	0	0	0
Logistic	0	0	0	0	0	76	0	0	19	0	0	0	0	881	0	0	0	0	0	24
Lognorm	0	0	0	0	0	0	0	0	0	0	5	0	0	0	995	0	0	0	0	0
Pareto	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	999	0	0	0	0
t-distribution	0	0	0	0	0	0	2	0	0	0	1	0	0	0	2	0	995	0	0	0
Uniform	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0	979	0	0
Hypergeometric	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	979	0
Binomial	40	1	0	0	0	25	0	0	2	0	0	0	0	2	0	0	0	0	0	930

Figure 5: Confusion matrix based on large CNN and PSA-5 neural model selector on test dataset with $K = 20$