2013

A Taxonomy of Massive Data for Optimal Predictive Machine Learning and Data Mining

Ernest Fokoue

A Taxonomy of Massive Data for Optimal Predictive Machine Learning and Data Mining

Ernest Fokoué
Center for Quality and Applied Statistics
Rochester Institute of Technology
98 Lomb Memorial Drive, Rochester, NY 14623, USA
ernest.fokoue@rit.edu

Abstract

Massive data, also known as big data, come in various ways, types, shapes, forms and sizes. In this paper, we propose a rough idea of a possible taxonomy of massive data, along with some of the most commonly used tools for handling each particular category of massiveness. The dimensionality p of the input space and the sample size n are usually the main ingredients in the characterization of data massiveness. The specific statistical machine learning technique used to handle a particular massive data set will depend on which category it falls in within the massiveness taxonomy. Large p small n data sets for instance require a different set of tools from the large n small p variety. Among other tools, we discuss Preprocessing, Standardization, Imputation, Projection, Regularization, Penalization, Compression, Reduction, Selection, Kernelization, Hybridization, Parallelization, Aggregation, Randomization, Replication, Sequentialization. Indeed, it is important to emphasize right away that the so-called no free lunch theorem applies here, in the sense that there is no universally superior method that outperforms all other methods on all categories of massiveness. It is also important to stress the fact that simplicity in the sense of Ockham's razor non plurality principle of parsimony tends to reign supreme when it comes to massive data. We conclude with a comparison of the predictive performance of some of the most commonly used methods on a few data sets.

Keywords: *Massive Data, Taxonomy, Parsimony, Sparsity, Regularization, Penalization, Compression, Reduction, Selection, Kernelization, Hybridization, Parallelization, Aggregation, Randomization, Sequentialization, Cross Validation, Subsampling, Bias-Variance Trade-off, Generalization, Prediction Error.*

I. Introduction

We consider a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i^\top \equiv (\mathrm{x}_{i1}, \mathrm{x}_{i2}, \cdots, \mathrm{x}_{ip})$ denotes the p-dimensional vector of characteristics of the input space $\mathcal{X}$, and y_i represents the corresponding categorical response value from the output space $\mathcal{Y} = \{1, \cdots, g\}$. Typically, one of the most basic ingredients in statistical data mining is the data matrix $\boldsymbol{X}$ given by

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \ddots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}. \tag{1}$$

Five aspects of the matrix $\boldsymbol{X}$ that are crucial to a taxonomy of massive data include: (i) The dimension p of the input space $\mathcal{X}$, which simply represents the number of explanatory variables measured; (ii) The sample size n, which represents the number of observations (sites) at which the variables were measured/collected; (iii) The relationship between n and p, namely the ratio n/p; (iv) The type of variables measured (categorical, ordinal, interval, count or real valued), and

the indication of the scales/units of measurement; (v) The relationships among the columns of $\boldsymbol{X}$, namely whether or not the columns are correlated (nearly linearly dependent). Indeed, as we will make clear later, massive data, also known as big data, come in various ways, types, shapes, forms and sizes. Different scenarios of massive data call upon tools and methods that can be drastically different at times. The rest of this paper is organized as follows: Section 2 presents our suggested taxonomy for massive data based on a wide variety of scenarios. Section 3 presents a summary of the fundamental statistical learning theory along with some of the most commonly used statistical learning methods and their application in the context of massive data; Section 4 presents a comparison of predictive performances of some popular statistical learning methods on a variety of massive data sets. Section 5 presents our discussion and conclusion, along with some of the ideas we are planning to explore along the lines of the present paper.

II. On the Diversity of Massive Data Sets

Categorization of massiveness as a function of the input space dimensionality p

Our idea of a basic ingredient for a taxonomy for massive data comes from a simple reasoning. Consider the traditional multiple linear regression (MLR) setting with p predictor variables under the Gaussian noise. In a typical model space search needed in variable selection, the best subsets approach fits $2^p - 1$ models and submodels. If $p = 20$, the space of linear models is of size 1 *million*. Yes indeed, one theoretically has to search a space of 1 *million* models when $p = 20$. Now, if we have $p = 30$, the size of that space goes up to 1 *billion*, and if $p = 40$, the size of the model space goes up to 1 *trillion*, and so on. *Our simple rule is that any problem with an input of more than* 50 *variables is a massive data problem, because computationally searching a thousand trillion is clearly a huge/massive task for modern day computers.* Clearly, those who earn their keep analyzing inordinately large input spaces like the ones inherent in microarray data (p for such data is in the thousands) will find this taxonomy somewhat naive, but it makes sense to us based on the computational insights underlying it. Besides, if the problem at hand requires the estimation of covariance matrices and their inverses through many iterations, the $O(p^3)$ computational complexity of matrix inversion would then require roughly 125000 complexity at every single iteration which could be quickly untenable computationally. Now, it's clear that no one in their right mind decides to exhaustively search a space of a thousand trillion models. However, this threshold gives us somewhat of a point to operate from. From now on, any problem with more than 50 predictor variables will be a big data problem, and any problem with p exceeding 100 will be referred to as a massive data problem.

Categorization of massiveness as a function of the sample size n

When it comes to ideas about determining how many observations one needs, common sense will have it that the more the merrier. After all, the more observations we have to close we are to the law of large numbers, and indeed, as the sample size grows, so does the precision of our estimation. However, some important machine learning methods like Gaussian Process Classifiers, Gaussian Process Regression Estimators, the Relevance Vector Machine (RVM), Support Vector Machine (SVM) and just about all other kernel methods operate in dual space and are therefore heavily dependent on the sample size n. The computational and statistical complexity of such methods is driven by the same size n. Some of these methods like Gaussian Processes and the Relevance Vector Machine require the inversion of $n \times n$ matrices. As a result, such methods could easily be computationally bogged down by too large a sample size n. Now, how large is

too large? Well, it takes $O(n^3)$ operations to invert an $n \times n$ matrix. Anyone who works with matrices quickly realizes that with modern-day computers, messing around with more that a few hundreds in matrix inversion is not very smart. These methods can become excruciatingly (impractically) slow or even unusable when n gets ever larger. For the purposes of our categorization, we set the cut-off at 1000 and define as *observation-massive* any dataset that have $n > 1000$. Again, we derive this categorization based on our observations on the computational complexity of matrix inversion and its impact on some of the state-of-the-art data mining techniques.

Categorization of massiveness as a function of the ratio n/p

From the previous argumentation, we could say that when $p > 50$ or $n > 1000$, we are computationally in the presence of massive data. It turns out however that the ratio n/p is even more important to massive and learnability than n and p taken separately. From experience, it's our view that for each explanatory variable under study, a minimum of 10 observations is needed to have a decent analysis from both accuracy and precision perspectives. Put in simple terms, the number of rows must be at least 10 times the number of columns, specifically $n > 10p$. Using this simple idea and the fact that information is an increasing function of n, we suggest the following taxonomy as a continuum of n/p.

	$\frac{n}{p} < 1$ Information Poverty $(n \lll p)$	$1 \leq \frac{n}{p} < 10$ Information Scarcity	$\frac{n}{p} \geq 10$ Information Abundance $(n \ggg p)$
$n > 1000$	Large p, Large n A	Smaller p, Large n B	Much smaller p, Large n C
$n \leq 1000$	Large p, Smaller n D	Smaller p, Smaller n E	Much smaller p, Small n F

Table 1: *In this taxonomy,* A *and* D *pose a lot of challenges.*

III. Methods and Tools for Handling Massive Data

Batch data vs incremental data production

When it comes to the way in which the data is acquired or gathered, the traditionally assumed way is the so-called batch, where all the data needed is available all at once. In state-of-the-art data mining however, there are multiple scenarios where the data is produced/delivered in a sequential/incremental manner. This has prompted the surge in the so-called online learning methods. As a matter of fact, the perceptron learning rule, arguably the first algorithm that launched the whole field of machine learning, is an online learning algorithm. Online algorithms have the distinct advantage that the data does not have to be stored in memory. All that is required in the storage of the built model at time t. In the sense the stored model is assumed to have accumulated the structure of the underlying model. Because of that distinct feature, one may think of using online algorithms even when the whole data available. Indeed, when the sample size n is so large that the data cannot fit in the computer memory, one can consider building a learning method that receives the data sequentially/incrementally rather than trying to load the whole data set into memory. We shall refer to this aspect of massive data as sequentialization or

incrementalization. *Sequentialization* is therefore useful for both streaming data and massive data that is too large to be loaded into memory all at once.

Missing Values and Imputation Schemes

In most scenarios of massive data analytics, it is very common to be faced with missing values. The literature on missing values is very large, and we will herein simply mention very general guidelines. One of the first thing one needs to consider with missing values is whether they are missing systematically or missing at random. The second important aspect is the rate of missingness. Clearly, when we have abundance of data, the number of missing values is viewed differently. Three approaches are often used to address missingness: (a) Deletion, which consists of deleting all the rows that contain any missingness; (b) Central imputation, which consists of filling the missing cells of the data matrix with central tendencies like mode, median or mean; (c) model-based imputation using various adaptation of the ubiquitous Expectation-Maximization (EM) algorithm.

Inherent lack of structure and importance of preprocessing

Sentiment analysis based on social media data from facebook and twitter, topic modelling based on a wide variety of textual data, classification of tourist documents or even to be more general the whole field of text mining and text categorization require the manipulation of inherently unstructured data. All these machine learning problems are of great interest to end-users, statistical machine learning practitioners and theorists, but cannot be solve without sometimes huge amounts of extensive pre-processing. The analysis of a text corpus for instance never starts with a data matrix like the $\boldsymbol{X}$ defined in Equation (1). With these inherently unstructured data like text data, the pre-processing often leads to data matrices whose entries are frequencies of terms. It's important to mention that term frequency matrices tend to contain many zeroes, because a term deemed important for a handful of documents will tend not to appear in many other documents. This content-sparsity can be a source of a variety of modelling problems.

Homogeneous vs Heterogeneous input space

There are indeed many scenarios of massive data where the input space is homogeneous, i.e. where all the variables are of the same type. Audio processing, image processing and video processing all belong to a class of massive data where all the variables are of the same type. There are however many other massive data scenarios where the input space is made up of variables of various different types. Such heterogeneous input spaces arise in fields like business, marketing, social sciences, psychology, etc ... where one can have categorical, ordinal, interval, count, and real valued variables gathered on the same entity. Such scenarios call for hybridization, which may take the form of combining two or more data-type-specific methods in order to handle the heterogeneity of the input space. In kernel methods for instance, if one has both `textual` inputs and `real valued` inputs, then one could simply use a kernel $\mathcal{K} = \alpha\mathcal{K}_1 + (1-\alpha)\mathcal{K}_2$ that is the convex combination of two data-type-specific kernels, namely a string kernel $\mathcal{K}_1$ and a real valued kernel $\mathcal{K}_2$. *Hybridization* can also be used directly in modelling through the use of combination of models.

Difference in measurement scale and the importance of transformation

Even when the input space is homogeneous, it is almost always the case that the variables are measured on different scales. This difference in scales can be the source of many modelling difficulties. A simple way to address this *scale heterogeneity* is to perform straightforward transformations that project all the variables onto the same scale.

Standardization: The most commonly used transformation is *standardization* which leads to all the variables having zero mean and unit variance. Indeed, if X_j is one of the variables in $\mathcal{X}$ and we have n observations $X_{1j}, X_{2j}, \cdots, X_{nj}$, then the standardized version of X_{ij} is

$$\tilde{X}_{ij} = \frac{X_{ij} - \bar{X}_j}{\sqrt{\sum_{i=1}^{n} (X_{ij} - \bar{X}_j)^2}}, \quad \text{where} \quad n\bar{X}_j = \sum_{i=1}^{n} X_{ij}$$

Unitization: is another form of transformation commonly used. *Unitization* simply consists of transformation the variables such that all take values in the unit interval $[0, 1]$. The resulting input space is therefore the *unit p-dimensional hypercube*, namely $[0, 1]^p$. With *unitization*, if X_j is one of the variables in $\mathcal{X}$ and we have n observations $X_{1j}, X_{2j}, \cdots, X_{nj}$, then the unitized version of X_{ij} is given by

$$\tilde{X}_{ij} = \frac{X_{ij} - \min X_j}{\max X_j - \min X_j}.$$

Dimensionality reduction and feature extraction

Learning, especially statistical machine learning, is synonymous with dimensionality reduction. Indeed, after data is gathered, especially massive data, nothing can be garnered in terms of insights until some dimensionality reduction is performed to provide meaningful summaries revealing the patterns underlying the data. Typically, when people speak of dimensionality reduction, they have in mind the determination of some *intrinsic dimensionality* q of the input space, where $q \lll p$. There are many motivations for dimensionality reduction (a) achieve orthogonality in the input space (b) eliminate redundant and noise variables, and as a result perform the learning in a lower dimensional and orthogonal input space with the benefit of variance reduction in the estimator. In practice, *lossy data compression techniques* like principal component analysis (PCA) and singular value decomposition (SVD) are the methods of choice for dimensionality reduction. However, when $n \lll p$, most of these techniques cannot be directly used in their generic forms.

Kernelization and the Power of Mapping to Feature Spaces

In some applications, like signal processing, it is always the case that $n \lll p$ in time domain. A ten seconds audio track at a $44100Hz$ sampling rate, generates a vector of dimension $p = 441000$ in time domain, and one typically has only few hundreds or maybe a thousand tracks for the whole analysis. Typically image processing problems are similar in terms of dimensionality with a simple face of size 640×512 generating a $p = 327680$ dimensional input space. In both these cases, it's impossible to use basic PCA or SVD, because $n \lll p$, making it impossible to estimate the covariance structure needed in eigenvalue decomposition. One of the solution to this problem is the use of the methods that operate in dual space, like kernel methods. In recent years, *kernelization* has been widely applied and with tremendous success to PCA, Canonical Correlation Analysis (CCA), Regression, Logistic Regression, k-Means clustering just to name a few. Given a dataset with n input vectors $\mathbf{x}_i \in \mathcal{X}$ from some p dimensional space, the main

ingredient in kernelization is a bivariate function $\mathcal{K}(\cdot,\cdot)$ defined on $\mathcal{X}\times\mathcal{X}$ and with values in $\mathbb{R}$, and the corresponding matrix of similarities $\boldsymbol{K}$ known as the Gram matrix and defined as

$$\boldsymbol{K}=\begin{bmatrix}\mathcal{K}(\boldsymbol{x}_1,\boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_1,\boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_1,\boldsymbol{x}_n)\\ \mathcal{K}(\boldsymbol{x}_2,\boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_2,\boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_2,\boldsymbol{x}_n)\\ \vdots & \ddots & \cdots & \vdots\\ \mathcal{K}(\boldsymbol{x}_n,\boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_n,\boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_n,\boldsymbol{x}_n)\end{bmatrix}.$$

Crucial to most operations like kernel PCA is the centered version of the Gram matrix given by

$$\tilde{\boldsymbol{K}}=(\boldsymbol{I}_n-\mathbb{I}_n)\boldsymbol{K}(\boldsymbol{I}_n-\mathbb{I}_n)=\boldsymbol{K}-\mathbb{I}_n\boldsymbol{K}-\boldsymbol{K}\mathbb{I}_n+\mathbb{I}_n\boldsymbol{K}\mathbb{I}_n,$$

where $\boldsymbol{I}_n\in\mathbb{R}^{n\times n}$ and $\mathbb{I}_n\in\mathbb{R}^{n\times n}$ are both $n\times n$ matrices defined as

$$\mathbb{I}_n=\frac{1}{n}\begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\\ \vdots & \ddots & \cdots & \vdots\\ 1 & 1 & \cdots & 1\end{bmatrix}\quad\text{and}\quad\boldsymbol{I}_n=\begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \ddots & \cdots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix}.$$

The next step is to solve the eigenvalue problem

$$\frac{1}{n}\tilde{K}\boldsymbol{v}_i=\lambda_i\boldsymbol{v}_i,$$

where $\boldsymbol{v}_i\in\mathbb{R}^n$ and $\lambda_i\in\mathbb{R}$ for $i=1,\cdots,n$. In matrix form, the eigenvalue problem is

$$\frac{1}{n}\tilde{K}=\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top.$$

In fact, basic PCA can be formulated in kernel form using the Euclidean inner product kernel $\mathcal{K}(\mathbf{x}_i,\mathbf{x}_j)=\langle\mathbf{x}_i,\mathbf{x}_j\rangle=\boldsymbol{x}_i^\top\boldsymbol{x}_j$, sometimes referred to as the vanilla kernel. If we center the data, i.e, such that $\sum_{i=1}^n\mathbf{x}_{ij}=0$, then the Gram matrix is

$$\boldsymbol{K}=\begin{bmatrix}\boldsymbol{x}_1^\top\boldsymbol{x}_1 & \boldsymbol{x}_1^\top\boldsymbol{x}_2 & \cdots & \boldsymbol{x}_1^\top\boldsymbol{x}_n\\ \boldsymbol{x}_2^\top\boldsymbol{x}_1 & \boldsymbol{x}_2^\top\boldsymbol{x}_2 & \cdots & \boldsymbol{x}_2^\top\boldsymbol{x}_n\\ \vdots & \ddots & \cdots & \vdots\\ \boldsymbol{x}_n^\top\boldsymbol{x}_1 & \boldsymbol{x}_n^\top\boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n^\top\boldsymbol{x}_n\end{bmatrix}=\boldsymbol{X}\boldsymbol{X}^\top.$$

Now, the covariance matrix is $\boldsymbol{C}=\frac{1}{n}\sum_{i=1}^n\boldsymbol{x}_i\boldsymbol{x}_i^\top=\frac{1}{n}\boldsymbol{X}^\top\boldsymbol{X}$, and PCA based on the covariance is simply $\frac{1}{n}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w}_j=\lambda_j\boldsymbol{w}_j$ for $j=1,\cdots,p$ with $\boldsymbol{w}_j\in\mathbb{R}^p$ and $\lambda_j\in\mathbb{R}$.

Aggregation and the Appeal of Ensemble Learning

It is often common in massive data that selecting a single model does not lead the optimal prediction. For instance, in the presence of multicollinearity which is almost inevitable when p is very large, function estimators are typically unstable and of large variance. The now popular *bootstrap aggregating* also referred to as *bagging* offers one way to reduce the variance of the estimator by creating an aggregation of bootstrapped versions of the base estimator. This is an example of ensemble learning, with the aggregation/combination formed from equally weighted

base learners.

Bagging Regressors: Let $\hat{\mathtt{g}}^{(\mathtt{b})}(\cdot)$ be the bth bootstrap replication of the estimated base regression function $\hat{\mathtt{g}}(\cdot)$. Then the *bagged* version of the estimator is given by

$$\hat{\mathtt{g}}^{(\mathtt{bagging})}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{\mathtt{g}}^{(\mathtt{b})}(\mathbf{x}).$$

If the base learner is a multiple linear regression model estimator $\hat{\mathtt{g}}(\mathbf{x}) = \hat{\beta}_0 + \mathbf{x}^\top\hat{\boldsymbol{\beta}}$, then the bth bootstrapped replicate is $\hat{\mathtt{g}}^{(\mathtt{b})}(\mathbf{x}) = \hat{\beta}_0^{(b)} + \mathbf{x}^\top\hat{\boldsymbol{\beta}}^{(b)}$, and the bagged version is

$$\hat{\mathtt{g}}^{(\mathtt{bagging})}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{\beta}_0^{(b)} + \mathbf{x}^\top\hat{\boldsymbol{\beta}}^{(b)}\right)$$

Bagging classifiers: Consider a multi-class classification task with labels y coming from $\mathcal{Y} = \{1, 2, \cdots, m\}$ and predictor variables $\mathbf{x} = (\mathtt{x}_1, \mathtt{x}_2, \cdots, \mathtt{x}_q)^\top$ coming from a q-dimensional space $\mathcal{X}$. Let $\hat{\mathtt{g}}^{(\mathtt{b})}(\cdot)$ be the bth bootstrap replication of the estimated base classifier $\hat{\mathtt{g}}(\cdot)$, such that $(\hat{\mathtt{y}})^{(b)} = \hat{\mathtt{g}}^{(\mathtt{b})}(\mathbf{x})$ is the bth bootstrap estimated class of $\mathbf{x}$. The estimated response by bagging is obtained using the majority vote rule, which means the most frequent label throughout the B bootstrap replications. Namely, $\hat{\mathtt{g}}^{(\mathtt{bagging})}(\mathbf{x}) = \mathtt{Most\ frequent\ label\ in}\ \hat{\mathtt{C}}^{(\mathtt{B})}(\mathbf{x})$, where

$$\hat{\mathtt{C}}^{(\mathtt{B})}(\mathbf{x}) = \left\{\hat{\mathtt{g}}^{(1)}(\mathbf{x}), \hat{\mathtt{g}}^{(2)}(\mathbf{x}), \cdots, \hat{\mathtt{g}}^{(\mathtt{B})}(\mathbf{x})\right\}.$$

Succinctly, we can write the bagged label of $\mathbf{x}$ as

$$\hat{\mathtt{g}}^{(\mathtt{bagging})}(\mathbf{x}) = \arg\max_{\mathtt{y}\in\mathcal{Y}}\left\{\mathtt{freq}_{\hat{\mathtt{C}}^{(\mathtt{B})}(\mathbf{x})}(\mathtt{y})\right\} = \arg\max_{\mathtt{y}\in\mathcal{Y}}\left\{\sum_{b=1}^{B}\left(\mathbf{1}_{\{\mathtt{y}=\hat{\mathtt{g}}^{(\mathtt{b})}(\mathbf{x})\}}\right)\right\}.$$

It must be emphasized that in general, ensembles do not assign equal weights to base learners in the aggregation. The general formulation in the context of regression for instance is

$$\hat{\mathtt{g}}^{(\mathtt{agg})}(\mathbf{x}) = \sum_{b=1}^{B}\alpha^{(\mathtt{b})}\hat{\mathtt{g}}^{(\mathtt{b})}(\mathbf{x}).$$

where the aggregation is often convex, i.e. $\sum_{b=1}^{B}\alpha^{(\mathtt{b})} = 1$.

Parallelization

When the computational complexity for building the base learner is high, using ensemble learning techniques like bagging becomes very inefficient, sometimes to the point of being impractical. One way around this difficulty is the use of parallel computation. In recent years, both R and Matlab have offered the capacity to parallelize operations. Big data analytics will increasingly need parallelization as a way to speed up computations or sometimes just make it possible to handle massive data that cannot fit into a single computer memory.

Regularization and the Power of Prior Information

All statistical machine learning problems are inherently inverse problems, in the sense that learning methods seek to optimally estimate an unknown generating function using empirical observations assumed to be generated by it. As a result statistical machine learning problems are inherently *ill-posed*, in the sense that they typically violate at least one of Hadamard's three *well-posedness* conditions. For clarity, according to Hadamard a problem is *well-posed* if it fulfills the following three conditions: *(a) A solution exists; (b) The solution is unique; (c) The solution is stable, i.e does not change drastically under small perturbations.* For many machine learning problems, the first condition of well-posedness, namely existence, is fulfilled. However, the solution is either not unique or not stable. With large p small n for instance, not only is there a *multiplicity* of solutions but also the instability thereof, due to the singularities resulting from the fact that $n \lll p$. Typically, the regularization framework is used to isolate a feasible and optimal (in some sense) solution. *Tikhonov*'s regularization is the one most commonly resorted to, and typically amounts to a Lagrangian formulation of a constrained version of the initial problem, the constraints being the devices/objects used to isolate a unique and stable solution.

IV. Statistical Machine Learning Methods for Massive Data

We consider the traditional supervised learning task of pattern recognition with the goal of estimating a function f that maps an input space $\mathcal{X}$ to a set of labels $\mathcal{Y}$. We consider the symmetric zero-loss $\ell(Y, f(X)) = 1_{\{Y \neq f(X)\}}$, and the corresponding theoretical risk function

$$R(f) = \mathbb{E}[\ell(Y, f(X))] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, f(\mathbf{x}))dP(\mathbf{x}, y) = \Pr[Y \neq f(X)].$$

Ideally, one would like to find the universally best classifier f^* that minimizes the rate $R(f)$ of misclassification, i.e.,

$$f^* = \underset{f}{\texttt{argmin}}\Big\{\mathbb{E}[\ell(Y, f(X))]\Big\} = \underset{f}{\texttt{argmin}}\Big\{\Pr[Y \neq f(X)]\Big\}.$$

It is impossible in practice to find f^*, because that would require knowing the joint distribution of (X, Y) which is usually unknown. In a sense, $R(f)$, the theoretical risk, serves as a standard only and helps establish some important theoretical results in pattern recognition. For instance, although in most practical problems one cannot effectively compute it, it has been shown theoretically that the universally best classifier f^* is the so-called Bayes classifier, the one obtained through the Bayes' formula by computing the posterior probability of class membership as the discriminant function, namely,

$$f^*(\mathbf{x}) = \texttt{class}^*(\mathbf{x}) = \underset{j\in\{1,\cdots,g\}}{\texttt{argmax}}\{\Pr[Y = j|\mathbf{x}]\} = \underset{j\in\{1,\cdots,g\}}{\texttt{argmax}}\left\{\frac{\pi_j p(\mathbf{x}|y = j)}{p(\mathbf{x})}\right\}.$$

Assuming multivariate Gaussian class conditional densities with common covariance matrix Σ and mean vectors $\boldsymbol{\mu}_0$ and $\boldsymbol{\mu}_1$, the Bayes Risk, that is the risk associated to the Bayes classifier, is given by $R(f^*) = R^* = \Phi(-\sqrt{\Delta}/2)$ where $\Phi(\cdot)$ is the standard normal cdf and

$$\Delta = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0).$$

Once again, it is important to recall that this R^* is not knowable in practice, and what typically happens is that, instead of seeking to minimize the theoretical risk $R(f)$, experimenters focus on minimizing its empirical counterpart, known as the empirical risk. Given an i.i.d sample $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2) \cdots, (\mathbf{x}_n, y_n)\}$, the corresponding empirical risk is given by

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} 1_{\{y_i \neq f(\mathbf{x}_i)\}},$$

which is simply the observed (empirical) misclassification rate. It is re-assuring to know that fundamental results in statistical learning theory (See Vapnik (2000)) establish that as the sample size goes to infinity, the empirical risk mimics the theoretical risk

$$\lim_{n \to \infty} \Pr[|\hat{R}(f) - R(f)| < \epsilon] = 1.$$

From a practical perspective, this means that the empirical risk provides a tangible way to search the space of possible classifiers. Another crucial point is the emphasis on the fact that even with this empirical risk, we still cannot feasibly search the universally best function, for such a space would be formidably large. That's where the need to choose a particular function class arises. In other words, instead of seeking an elusive universally best classifier, one simply proposes a plausible classifier, possibly based on aspects of the data, then finds the empirical risk minimizer in that space, and then, if the need arises maybe theoretically find out how the associated risk compares to the Bayes risk. One of the fundamental results in statistical learning theory has to do with the fact the minimizer of the empirical risk could turn out to be overly optimistic, and lead to poor generalization performance. It is indeed the case, that by making our estimated classifier very complex, it can adapt too well to the data at hand, meaning very low in-sample error rate, but yield very high out of sample error rates, due to overfitting, the estimated classifier having learned both the signal and the noise. In technical terms, this is referred to as the bias-variance dilemma, in the sense that by increasing the complexity of the estimated classifier, the bias is reduced (good fit all the way to the point of overfitting) but the variance of that estimator is increased. On the other hand, considering much simpler estimators, leads to less variance but higher bias (due to underfitting, model not rich enough to fit the data well). This phenomenon of bias variance dilemma, is particularly potent with massive data when the number of predictor variables p is much larger than the sample size n. One of the main tools in the modern machine learning arsenal for dealing with this is the so-called regularization framework whereby instead of using the empirical risk alone, a constrained version of it, also known as the regularized or penalized version is used.

$$\hat{R}_{\texttt{reg}}(f) = \hat{R}(f) + \lambda \|f\|_{\mathcal{H}} = \frac{1}{n} \sum_{i=1}^{n} 1_{\{y_i \neq f(\mathbf{x}_i)\}} + \lambda \|f\|_{\mathcal{H}},$$

where λ is referred to as the tuning (regularization) parameter, and $\|f\|_{\mathcal{H}}$ is some measure of the complexity of f with the class $\mathcal{H}$ from which it is chosen. It makes sense that choosing a function f with a smaller value of $\|f\|_{\mathcal{H}}$ helps avoid overfitting. The value of $\lambda \in [0, +\infty)$, controls the trade-off between bias (goodness of fit), and function complexity (which is responsibility for variance). Practically though, it may still be hard to even explore the theoretical properties of a given classifier and compare it to the Bayes risk, precisely because, methods typically do not directly act on the zero-one loss function, but instead use at best surrogates of it. Indeed, within a selected class $\mathcal{H}$ of potential classifiers, one typically chooses some loss function $\ell(\cdot, \cdot)$ with some

desirable properties like smoothness and/or convexity (this is because one needs at least to be able to build the desired classifier), and then finds the minimizer of its regularized version, i.e.,

$$\hat{R}_{\texttt{reg}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_{\mathcal{H}}.$$

Note that λ stills controls the bias-variance trade-off as before. Now, since the loss function typically chosen is not the zero-one loss on which the Bayes classifier (universally best) is based, there is no guarantee that the best in the selected class $\mathcal{H}$ under the chosen loss function $\ell(\cdot, \cdot)$ will mimic f^*. As a matter of fact, each optimal classifier from a given class $\mathcal{H}$ will typically perform well if the data at hand and the generator from which ut came, somewhat accord with the properties of the space $\mathcal{H}$. This remark is probably what prompted the famous so-called no free lunch theorem, herein stated informally.

Theorem 1. *(`No Free Lunch`) There is no learning method that is universally superior to all other methods on all datasets. In other words, if a learning method is presented with a data set whose inherent patterns violate its assumptions, then that learning method will under-perform.*

The above no free lunch theorem basically says that there is no such thing as a universally superior learning method that outperforms all other methods on all possible data, no matter how sophisticated the method may appear to be. Indeed, it is very humbling to see that some of the methods deemed somewhat simple sometimes hugely outperform the most sophisticated ones when compared on the basis of average out of sample (test) error. It is common practice in data mining and machine learning in general, to compare methods based on benchmark data, and empirical counterparts of the theoretical predictive measures, often computed using some form of re-sampling tool like the bootstrap or cross-validation. Massive data, also known as big data, come in various types, shapes, forms and sizes. The specific statistical machine learning technique used to handle a particular massive data set depends on which aspect of the taxonomy it falls in. Indeed, it is important to emphasize that the no free lunch theorem applies more potently here, in the sense that there is no panacea that universally applies to all massive data sets. It is important however to quickly stress the fact that simplicity in the sense of Ockham's razor non plurality principle of parsimony tends to reign supreme when it comes to massive data. In this paper, we propose a rough idea of a possible taxomony of massive data, along with some of the most commonly used tools for handling each particular class of massiveness. In this paper, we consider a few datasets of different types of massiveness, and we demonstrate through computational results that the no free lunch applies as a stronger as ever. We typically consider some of the most commonly used pattern recognition techniques, from those that are most simple and intuitive to some that are considered sophisticated and state-of-the-art, and we show that the performances vary sometimes drastically from data to data. It turns out, as we will show, that depending on the type of massiveness, some methods cannot even be used. We also provide our taxonomy of massiveness along with different approaches to dealing with each case. See Vapnik (2000) and Guo et al. (2005)

Linear Discriminant Analysis

Under the assumption of multivariate normality of the class conditional densities with equal covariance matrices, namely $(\mathbf{x}|y = j) \sim \texttt{MVN}(\boldsymbol{\mu}_j, \Sigma)$, or specifically,

$$p(\mathbf{x}|y = j) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^\top \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)\right\},$$

a classifier with excellent desirable properties is Linear Discriminant Analysis (LDA) classifier, which, given any new point $\mathbf{x}$, will estimate the corresponding class Y as

$$\hat{Y}_{\texttt{LDA}} = \hat{f}_{\texttt{LDA}}(\mathbf{x}) = 1_{\left\{\hat{\beta}_0 + \hat{\boldsymbol{\beta}}^\top \mathbf{x} > 0\right\}} = 1_{\{\delta_1(\mathbf{x}) > \delta_0(\mathbf{x})\}}$$

where

$$\delta_j(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j - \frac{1}{2} \boldsymbol{\mu}_j^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j + \log \pi_j$$

or

$$\hat{\boldsymbol{\beta}} = \hat{\Sigma}^{-1}(\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_0) \quad \text{and} \quad \hat{\beta}_0 = -\frac{1}{2}(\hat{\boldsymbol{\mu}}_1 + \hat{\boldsymbol{\mu}}_0)^\top \hat{\Sigma}^{-1}(\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_0) + \log \frac{\hat{\pi}_1}{\hat{\pi}_0}$$

where using the indicators $z_{ij} = I(y_i = j) = 1_{\{y_i = j\}}$, the estimated prior probabilities of class membership is given by $\hat{\pi}_j = (n_j/n) = (1/n)\sum_{i=1}^n z_{ij}$, the sample mean vectors $\boldsymbol{\mu}_j$ are given by $\hat{\boldsymbol{\mu}}_j = (1/n_j)\sum_{i=1}^n z_{ij}\mathbf{x}_i$ and the sample covariance matrices $S_j = (1/(n_j - 1))\sum_{i=1}^n z_{ij}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)^\top$, so that the sample pooled covariance

$$\hat{\Sigma} = (1/n)\sum_{j=0}^{1}\sum_{i=1}^{n} z_{ij}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)^\top$$

The estimation of Σ is clearly central to the use of Linear Discriminant Analysis. However, when $n \lll p$, $\hat{\Sigma}$ is ill-defined at best. One of the approaches to dealing with this drawback is the use of regularization which essentially consists of adding a jitter to the diagonal of $\hat{\Sigma}$. For $\alpha \in (0,1)$ or alternatively for $\lambda \in (0,\infty)$, the regularized version $\hat{\Sigma}$, namely $\hat{\Sigma}_{\texttt{reg}}$ can be obtained $\hat{\Sigma}_{\texttt{reg}} = (1-\alpha)\hat{\Sigma} + \alpha I$ or $\Sigma_{\texttt{reg}} = \hat{\Sigma} + \lambda I$ or even $\Sigma_{\texttt{reg}} = \lambda\hat{\Sigma} + I$. Friedman (1989) provides one of the earliest full treatment of this approach to singularity in LDA. Guo et al. (2005) give a very compelling application of this kind of regularization to microarray data.

Nearest Neighbors Methods

The so-called k-Nearest Neighbors approach to learning computes the estimated response $\hat{Y}^*_{\texttt{kNN}}$ of a new point $\mathbf{x}^*$ as follows: first set the neighborhood size k; then choose a distance measure $d(\cdot,\cdot)$ defined on $\mathcal{X} \times \mathcal{X}$; then compute $d_i^* = d(\mathbf{x}^*, \mathbf{x}_i)$, $i = 1, \cdots, n$, then rank all the distances d_i^* in increasing order $d^*_{(1)} \leq d^*_{(2)} \leq \cdots \leq d^*_{(k)} \leq d^*_{(k+1)} \leq \cdots \leq d^*_{(n)}$. Then specify $\mathcal{V}_k(\mathbf{x}^*) = \left\{\mathbf{x}_i : d(\mathbf{x}^*, \mathbf{x}_i) \leq d^*_{(k)}\right\}$, the size k neighborhood of $\mathbf{x}^*$, made up of the k points in $\mathcal{D}$ that are closest to $\mathbf{x}^*$ according to $d(\cdot,\cdot)$. The estimate response is the *Most frequent label in* $\mathcal{V}_k(\mathbf{x}^*)$, specifically

$$\hat{Y}^*_{\texttt{kNN}} = \hat{f}_{\texttt{kNN}}(\mathbf{x}^*) = \underset{j \in \{1,\cdots,g\}}{\texttt{argmax}} \left\{p_j^{(k)}(\mathbf{x}^*)\right\} \tag{2}$$

where

$$p_j^{(k)}(\mathbf{x}^*) = \frac{1}{k}\sum_{i=1}^{n} I(Y_i = j) I(\mathbf{x}_i \in \mathcal{V}_k(\mathbf{x}^*)) \tag{3}$$

estimates the probability that $\mathbf{x}^*$ belongs to class j based on $\mathcal{V}_k(\mathbf{x}^*)$. Indeed, $p_j^{(k)}(\mathbf{x}^*)$ can be thought of as a rough estimate of $\pi_j(\mathbf{x}^*) = \Pr[Y^* = j|\mathbf{x}^*]$, the posterior probability of class membership of $\mathbf{x}^*$,ie $p_j^{(k)}(\mathbf{x}^*) =\approx \widehat{\pi_j(\mathbf{x}^*)}$. See Hastie et al. (2001) and Clarke et al. (2009). kNearest Neighbors (kNN) essentially performs classification by voting for the most popular re-

sponse among the k nearest neighbors of $\mathbf{x}^*$. In a sense, `kNN` provides the most basic form of nonparametric classification Thanks to the fact that the estimated response $\hat{Y}^*_{\texttt{kNN}}$ for $\mathbf{x}^*$ is - at least - a crude nonparametric estimator of Bayes classifier's response, which somewhat justifies (or at least explains) the great interest in kNN. Typical distances used in practice include the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2 = \left(\sum_{\ell=1}^{p} (\mathtt{x}_{i\ell} - \mathtt{x}_{j\ell})^2\right)^{1/2}$, and the Manhattan distance $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{\ell=1}^{p} |\mathtt{x}_{i\ell} - \mathtt{x}_{j\ell}|$. One of the appeals of the k-Nearest Neighbors approach in both classification and regression lies in the fact it seamlessly applied to data of any type, as long as a valid and appropriate distance can be defined. For instance, with binary data where each $\mathbf{x}_i^\top = (\mathtt{x}_{i1}, \cdots, \mathtt{x}_{ip}) \in \{0,1\}^p$ is a p-dimensional vector of binary (indicator) values, one could use the Hamming distance or the very useful Jaccard distance defined as $d_J(\mathbf{x}_i, \mathbf{x}_j) = 1 - J(\mathbf{x}_i, \mathbf{x}_j)$, where, $J(\mathbf{x}_i, \mathbf{x}_j)$ is the Jaccard similarity index, defined by

$$J(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^\top \mathbf{x}_j}{\mathbf{x}_i^\top \mathbf{x}_i + \mathbf{x}_j^\top \mathbf{x}_j - \mathbf{x}_i^\top \mathbf{x}_j}.$$

Clearly, $|J(\mathbf{x}_i, \mathbf{x}_j)| \leq 1$, with maximum value of 1 attained at $J(\mathbf{x}_i, \mathbf{x}_i)$ and minimum value of 0 corresponding to two vectors with no matching 1 values. Since the fundamental building block of `kNN` is the distance measure, one can easily perform classification beyond the traditional setting where the predictors are numeric. For instance, classification with kNN can be readily performed on indicator attributes $\mathbf{x}_i = (\mathtt{x}_{i1}, \cdots, \mathtt{x}_{ip})^\top \in \{0,1\}^p$. `kNN` classifiers are inherently naturally multiclass, and are used extensively in applications such as image processing, character recognition and general pattern recognition tasks.

When it so happens as it often does that the input space $\mathcal{X}$ is nonhomogeneous, a typical approach to nearest neighbors pattern recognition consists of defining a distance that is the direct or convex sum of the type-specific variables. For instance, if there is a group of numeric variables and a group of categorical variables, one could use the distance $d_1(\cdot, \cdot)$ for the first group, and $d_1(\cdot, \cdot)$ for the second group, and then either use the direct sum $d(\mathbf{x}_i, \mathbf{x}_j) = d_1(\mathbf{x}_i, \mathbf{x}_j) + d_2(\mathbf{x}_i, \mathbf{x}_j)$ or the convex sum $d(\mathbf{x}_i, \mathbf{x}_j) = \alpha d_1(\mathbf{x}_i, \mathbf{x}_j) + (1 - \alpha) d_2(\mathbf{x}_i, \mathbf{x}_j)$ for some $\alpha \in (0, 1)$.

Nearest Neighbors Algorithms are extremely popular in Statistical Data Mining and Machine Learning probably due to the fact that they offer the simplest and most flexible form of nonparametric predictive analytics. In this paper, we explore various aspects of predictive analytics in regression and pattern recognition (classification) using kNearest Neighbors. The complexity of the underlying pattern in k-Nearest Neighbors Analysis is controlled by the size k of the neighborhood. We show how Cross validation and other forms of re-sampling can be used to determine the optimal value k that helps achieve bias-variance trade-off and thereby good generalization (low prediction error).

Support Vector Machines

Consider a response variable taking values in $\{-1, +1\}$ and a regularized empirical risk functional given by

$$\hat{R}_{\texttt{reg}}(\beta_0, \boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} (1 - y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i))_+ + \frac{\lambda}{2} \sum_{j=0}^{p} |\beta_j|^2,$$

where $\lambda \in \mathbb{R}_+^*$ is the *regularization* (tuning) hyperparameter, and the *hinge loss* $(u)_+$ is defined as

$$(u)_+ = \max(0, u) = \begin{cases} 0 & \text{if } u < 0 \\ u & \text{if } u \geq 0 \end{cases}$$

Solving the optimization problem

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}^\top)^\top = \underset{(\beta_0, \boldsymbol{\beta}^\top)^\top \in \boldsymbol{R}^{p+1}}{\texttt{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} (1 - y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i))_+ + \frac{\lambda}{2} \sum_{j=0}^{p} |\beta_j|^2 \right\}$$

yields the linear support vector machine (SVM) binary classifier whose predicted response is

$$\hat{f}_{\texttt{svm}}(\mathbf{x}) = \mathsf{sign}\left(\hat{\beta}_0 + \hat{\boldsymbol{\beta}}^\top \mathbf{x}\right)$$

The above solution assumes that the decision boundary between the two classes is a hyperplane. However, it often happens that the decision boundary is not linear. *Kernelization* is the standard approach to handling the nonlinearity of the decision boundary in support vector machine classification. The kernel version of the SVM classifier is given by

$$\hat{f}_{\texttt{svm}}(\mathbf{x}) = \mathsf{sign}\left(\sum_{i=1}^{n} y_i \hat{\alpha}_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i) + \hat{\beta}_0\right),$$

where $\mathcal{K}(\cdot, \cdot)$ is a bivariate function called *kernel* defined on $\mathcal{X} \times \mathcal{X}$, and used to measure the similarity between two points in observation space. The $\hat{\alpha}_i$'s are determined via quadratic programming optimization, with the nonzero $\hat{\alpha}_i$'s corresponding to the so-called support vectors. One of the most general and most commonly used kernel in the context of pattern recognition is the Gaussian Radial Basis Function kernel given by

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\tau^2}\right).$$

The second most commonly used kernel is the Laplace radial basis function kernel defined as

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_1}{\tau}\right).$$

There are many other kernels like the polynomial kernel $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = (\texttt{scale}\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \texttt{offset})^{\texttt{degree}}$, and others, see Clarke et al. (2009), Bousquet (2003), Tipping (2001), Vapnik (2000) among others. When it so happens that the input space is nonhomogeneous, one could remedy by defined a *hybrid* that is a linear convex combination of other dat-type specific kernels, namely

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \alpha \mathcal{K}_1(\mathbf{x}_i, \mathbf{x}_j) + (1 - \alpha)\mathcal{K}_2(\mathbf{x}_i, \mathbf{x}_j)$$

It can therefore be said that the Support Vector Machine has the potential to use massive data tools like Regularization, Kernelization, Hybridization. In fact, we will see later that even *aggregation/combination* will be using in both the *bagging* and *boosting* SVM classifiers.

Logistic Regression

Arguably one of the most widely used pattern recognition machines, logistic regression is indeed used in the context of massive data with both regularization and kernelization frequently resorted to in a variety of scenarios. Using the traditional $\{0, 1\}$ indicator labelling, the empirical risk for

the binary linear logistic regression model is given by

$$\hat{R}(\beta_0, \boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\left\{y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) - \log\left[1 + \exp\left\{(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)\right\}\right]\right\}.$$

However, if we use the labelling $\{-1, +1\}$ as with SVM, the empirical risk for the linear logistic regression model is given by

$$\hat{R}(\beta_0, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}\log\left[1 + \exp\left\{-y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)\right\}\right].$$

One of the most common extensions of this basic formulation of logistic regression is the use of the regularized version of the empirical risk with ℓ_1 norm on the space of the coefficients β_j's. This so-called LASSO logistic regression achieves both shrinkage and variable selection in situations where the data contains redundant and/or noise variables.

$$\hat{R}_{\texttt{reg}}(\beta_0, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}\log\left[1 + \exp\left\{-y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)\right\}\right] + \lambda\sum_{j=0}^{p}|\beta_j|$$

An even more powerful extension of the logistic regression model comes with the use of kernels to capture the nonlinearity of underlying decision boundary of the classifier. Using kernels as defined earlier, the regularized empirical risk for the kernel logistic regression is given by

$$\hat{R}_{\texttt{reg}}(g) = \frac{1}{n}\sum_{i=1}^{n}\log\left[1 + \exp\left\{-y_i g(\mathbf{x}_i)\right\}\right] + \frac{\lambda}{2}\|g\|^2_{\mathcal{H}_\mathcal{K}}$$

where

$$g(\mathbf{x}_i) = v + \sum_{j=1}^{n} w_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j).$$

with $g = v + h$, $v \in \mathbb{R}$ and $h \in \mathcal{H}_\mathcal{K}$. Here, $\mathcal{H}_\mathcal{K}$ is the Reproducing Kernel Hilbert Space (RKHS) engendered by the kernel $\mathcal{K}$. The predicted class (label) of $\mathbf{x}$ via KLR is given by

$$\hat{f}_{\texttt{KLR}}(\mathbf{x}) = \texttt{sign}\left(\frac{1}{1 + \exp\{-\hat{g}(\mathbf{x})\}} - \frac{1}{2}\right).$$

One of the main advantages of KLR over SVM lies in the fact that KLR unlike SVM provides both the predicted label (hard classification) and the probability (soft classification) thereof, while SVM is inherently built to provide only the predicted label. KLR also extends naturally to multi-class while SVM requires more complicated modelling to extend beyond binary classification.

Classification and Regression Trees Learning

Understanding trees is indeed straightforward as they are intuitively appealing piecewise functions operating on a partitioning of the input space. Given $\mathcal{D} = \left\{(\mathbf{x}_1, Y_1), \cdots, (\mathbf{x}_n, Y_n)\right\}$, with $\mathbf{x}_i \in \mathcal{X}$, $Y_i \in \{1, \cdots, g\}$. If T denotes the tree represented by the partitioning of $\mathcal{X}$ into q regions $R_1, R_2, \cdots, R_q$ such that $\mathcal{X} = \cup_{\ell=1}^{q} R_\ell$, then, all the observations in a given terminal node (region)

will be assigned the same label, namely

$$c_\ell = \underset{j\in\{1,\cdots,\mathcal{G}\}}{\texttt{argmax}} \left\{ \frac{1}{|R_\ell|} \sum_{\mathbf{x}_i \in R_\ell} I(Y_i = j) \right\}$$

As a result, for a new point $\mathbf{x}$, its predicted class is given by

$$\hat{Y}_{\texttt{Tree}} = \hat{f}_{\texttt{Tree}}(\mathbf{x}) = \sum_{\ell=1}^{q} c_\ell \mathcal{I}_\ell(\mathbf{x}),$$

where $\mathcal{I}_\ell(\cdot)$ is the indicator function of R_ℓ, i.e. $\mathcal{I}_\ell(\mathbf{x}) = 1$ if $\mathbf{x} \in R_\ell$ and $\mathcal{I}_\ell(\mathbf{x}) = 0$ if $\mathbf{x} \notin R_\ell$. Trees are known to be notoriously unstable. Methods like bagging described earlier are often applied to trees to help reduce the variance of the tree estimator.

V. Predictive Performance Comparison of Learning Machines on some Massive Datasets

When we are given a benchmark test set $\{(\mathbf{x}_1, \mathrm{y}_1), (\mathbf{x}_2, \mathrm{y}_2), \cdots, (\mathbf{x}_m, \mathrm{y}_m)\}$, we can assess the generalizability (predictive strength) of a regression function f using the Empirical Prediction Error (EPE) defined as

$$\texttt{EPE}(f) = \frac{1}{m} \sum_{j=1}^{m} \ell(\mathrm{y}_j, f(\mathbf{x}_j)).$$

In the k-Neatest Neighbors context, we consider $S_k = \{k_{\texttt{min}}, \cdots, k_{\texttt{max}}\}$, the set of possible values of k, and the optimal k can then be estimated as

$$\hat{k}^{(\texttt{opt})} = \underset{k\in S_k}{\texttt{argmin}} \Big\{ \texttt{EPE}(\hat{f}_{\texttt{kNN}}) \Big\}.$$

It's often the case in practice that the training set is the only dataset available. In such cases, the training set is subsampled. Typically, the data set is split into training set and test set, and many realizations of the EPE are computed over many replications of the split. Let $\hat{f}_j$ be a regression estimator and let $\hat{f}_j^{(r)}$ be its r-th replication based on the rth split of the data into training and test sets. Now, let

$$E_r = \texttt{EPE}(\hat{f}_j^{(r)})$$

be the rth replication of the test Empirical Predictive Mean Squared Error based on the test portion of the split. Then we have $E_1, E_2, \cdots, E_R$, and can perform all kinds of statistical analyses on these numbers in order to gain deeper insights in the predictive strength of $\hat{f}_j$. For instance, given different competing estimators $\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_s$, we can plot comparative boxplots to assess virtually which of the estimators has the best performance. We could also simply compute measures of central tendency and measures of spread on these empirical predictive measures. Clearly, this gives us a potent framework that can be used for determining the predictively optimal value of k when using the kNearest Neighbors Algorithm. Indeed, if we consider each value of k as defining a different regression estimator, we can then compare them using EPE and indeed choose the value of k at which EPE is minimized.

A more commonly used approach involving a systematic subsampling as opposed to random (stochastic) subsampling is provided by the ubiquitous cross validation tool. To choose the opti-

mal number of neighbors by *leave one out cross validation (LOOCV)*, one computes

$$\hat{k}^{(\texttt{opt})} = \underset{k \in S_k}{\texttt{argmin}} \Big\{ \texttt{CV}(\hat{f}_{\texttt{kNN}}) \Big\},$$

where $S_k = \{k_{\texttt{min}}, \cdots, k_{\texttt{max}}\}$ is the set of possible values of k, and

$$\texttt{CV}(\hat{f}_{\texttt{kNN}}) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{f}_{\texttt{kNN}}^{(-i)}(\mathbf{x}_i)) = \frac{1}{n} \sum_{i=1}^{n} 1_{\{y_i \neq \hat{f}_{\texttt{kNN}}^{(-i)}(\mathbf{x}_i)\}}$$

with $\hat{f}_{\texttt{kNN}}^{(-i)}(\cdot)$ denoting an instance of the estimator $\hat{f}_{\texttt{kNN}}$ obtained without the ith observation. Once the "optimal" value of k is estimated using one of the above approach, we compare the predictive performance of the k-Nearest Neighbors estimates to the estimates produced by other function estimators.
Our first data set the *Musk* data set, with $n = 476$ and $p = 166$ which will fall in category *E* of our taxonomy, since $n < 1000$ and $1 < n/p < 10$.

	LDA	SVM	CART	rForest	GaussPR	kNN	adaBoost	NeuralNet	Logistic
`Musk`	0.2227	0.1184	0.2450	0.1152	0.1511	0.1922	0.1375	0.1479	0.2408
`Pima`	0.2193	0.2362	0.2507	0.2304	0.2304	0.3094	0.2243	0.2570	0.2186
`Crabs`	0.0452	0.0677	0.1970	0.1097	0.0702	0.0938	0.1208	0.0350	0.0363

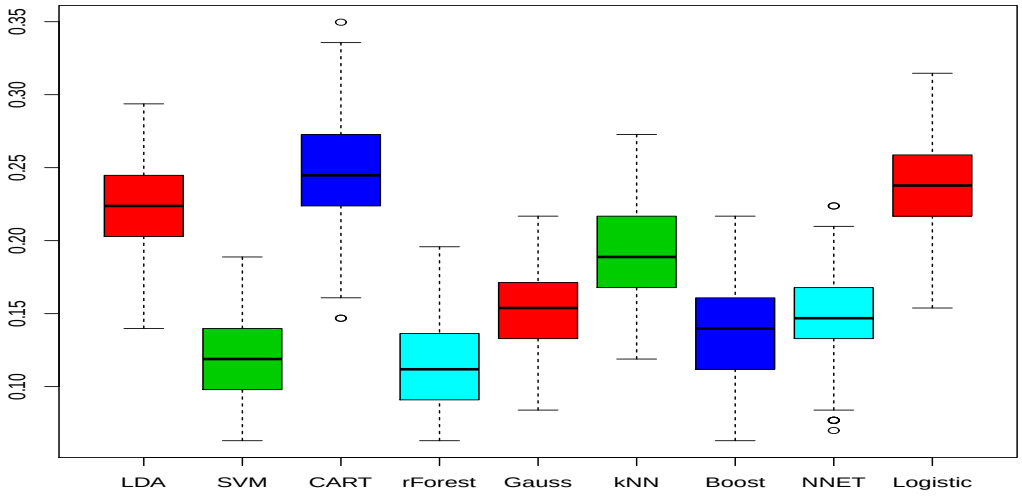

Figure 1: *Comparison of the average prediction error over $R = 100$ replications on the Musk data*

VI. Conclusion and discussion

We have shown in this paper that despite their simplicity and tremendous intuitive appeal, k-Nearest Neighbors can be implemented quite straightforwardly in the `R` environment. The ubiqui-

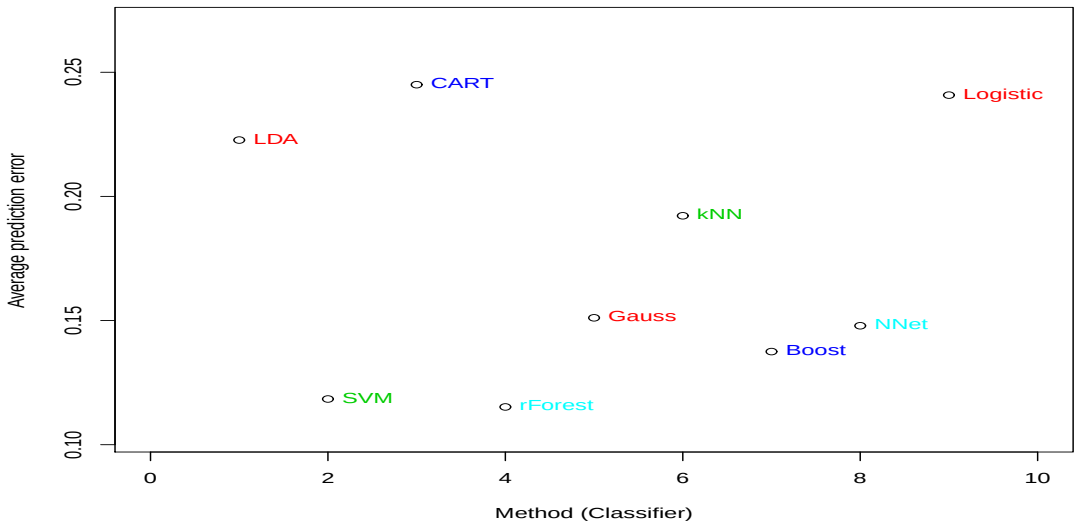

Figure 2: *Comparison of the average prediction error over $R = 100$ replications on the Musk data*

tous cross validation tool is also readily implemented and does provide an easily understandable approach to estimating the true generalization error of any estimating function, including the k-Nearest Neighbors estimator. To guarantee the completeness of our analysis, we have provided a comparison of predictive strength between optimally tuned k-Nearest neighbors estimates and support vector machines regression estimates.

References

Bousquet, O. (2003, August). Statistical learning theory. Machine Learning Summer School, Tuebigen, Germany.

Clarke, B., E. Fokoue, and H. Zhang (2009). *Principles and Theory for Data Mining and Machine Learning* (First ed.). Springer Texts in Statistics. Springer-Verlag.

Friedman, J. H. (1989). Regularized discriminant analysis. *Journal of the American Statistical Association 84*, 165–175.

Guo, Y., T. Hastie, and R. Tibshirani (2005). Regularized linear discriminant analysis and its application in microarrays. *Biostatistics 1*(1), 1–18.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *Elements of Statistical Learning*. Springer Texts in Statistics. Springer-Verlag.

Tipping, M. E. (2001). Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research 1*, 211–244.

Vapnik, V. N. (2000). *The Nature of Statistical Learning Theory*. Springer.