MACHINE LEARNING IN RFI

POSSIBILITIES IN RELATION TO SOCIAL INNOVATION AND IMPACT EVALUATIONS

Anders Grøn, Fall 2019

ROCKWOOL FONDEN

TABLE OF CONTENTS

"Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering."

Andrew Ng

"The best minds of my generation are thinking about how to make people click ads… That sucks."

Jeff Hammerbacher

"If you torture the data long enough it will confess to anything."

Ronald Coase

"The problem with socialism is that you eventually run out of other people's money."

Margaret Thatcher

ABSTRACT

In this paper, machine learning is presented in four primary chapters. The presentation focuses on which possibilities machine learning brings to our work in the ROCKWOOL Foundation Intervention Unit. In Chapter 1, I present the main concepts and algorithms within machine learning in order to clarify important considerations as well as establish a common ground for understanding the following chapters. In Chapter 2, I examine the latest breakthroughs related to using machine learning for social innovation and developing interventions. In Chapter 3, I present possibilities for using machine learning techniques as (part of) the actual intervention, and in Chapter 4, I delve into the most relevant algorithms and methods in relation to evaluation processes, particularly randomized controlled trials. In general, I note practical guidelines for using the different algorithms as well as construct off-the-shelf applicable STATA-dofiles and R-scripts for usage across projects. These are placed in the shared project-folder on our remote desktop at Statistics Denmark. See list below for paths. Finally, a list of references highlights sources and possibilities for further reading.

It is perfectly reasonable to add new content to this document or alter what I have already written. Try your utmost to use the formatting regime already in place.

LIST OF DO-FILES AND SCRIPTS

Initials	Method	Date	Path
ABG	SVM		
ABG	LASSO		
ABG	Het. Effects		
ABG	Seq. analysis		

1.0 INTRODUCTION TO MACHINE LEARNING

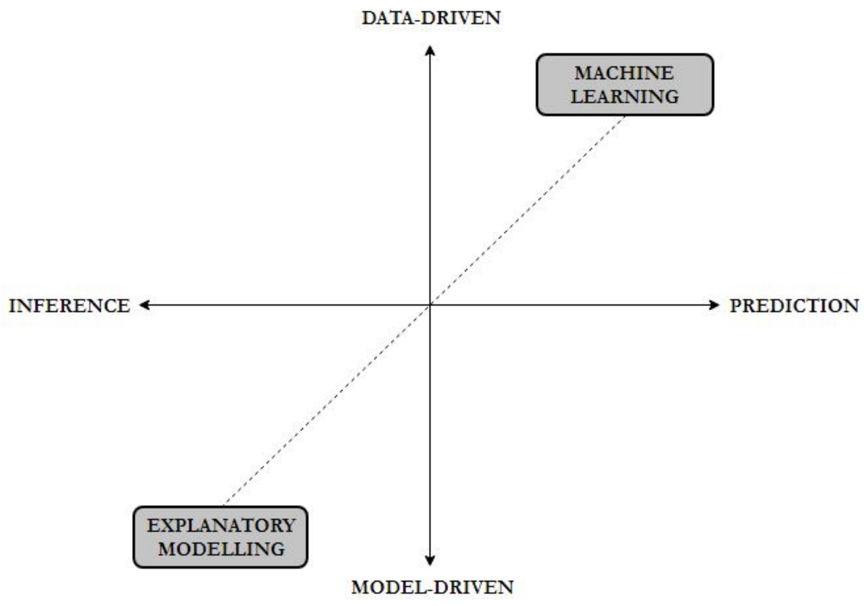
Machine Learning (ML) is an omnipresent notion in modern day data analysis. On a conceptual level, ML is rather ambiguous and often conflated with terms like data science, data mining and statistical learning. In this paper, I draw on all these different analytical branches interchangeably and understand ML as an umbrella term for algorithms and methods that to a greater or lesser extent automates analytical model building. In broader terms, ML constitutes an analysis-focused subbranch of artificial intelligence assuming that systems can learn from data, identify patterns and make enlightened decisions (or educated guesses) with minimal human intervention, i.e. analytical approaches that are data-driven and focused on generalization through either prediction or clarification of ex ante unperceivable patterns in data[1].

The distinction between prediction and clarification is central in understanding ML. Predictive modelling is known as supervised learning, while unsupervised learning covers clarification of patterns. This analytical distinction – and the content of both approaches – is elaborated in section 1.1. Before exemplifying further how ML intuitively can be understood and used, it is worthwhile clarifying some terminology. Observations in ML data sets are oftentimes called instances or examples. Independent variables are, collectively, called the input data (or the input/feature space) and consists of features (each feature being one predictor/independent variable). The outcome variable, which is not included when speaking of the input data, is usually referred to as the target, response, or outcome.

For now, and to figuratively emphasize the basis for understanding ML as data-driven and focused on the ability to predict, it is useful to compare supervised machine learning to explanatory statistical modelling (EM) – the type of modelling most social science statisticians is familiar with. The comparison emanates from Figure 1.1 below which depicts two continua: data-driven versus model-driven methods as well as inference- versus prediction-focused ones. The two continua primarily relate to the goal of the analysis and consequently it is necessary to establish what exact question one is interested in answering before choosing the appropriate approach and method. The linguistical relationship between approach and method is relevant to emphasize as some methods can be utilized both within ML and EM while others are predominantly or solely used within one approach. In the following, methods are typically referred to as nested within approaches.

[1] There are deviations from these characteristics when it comes to specific algorithms but nonetheless, they provide a lens through which the main ideas in ML can be understood. Relevant deviations will be emphasized throughout.

FIGURE 1.1.: COMPARISON OF EM AND ML



Note: stylized comparison based on ABG's reading of the literature.

In Figure 1.1. ML is placed in the upper-right quadrant and regarded as data-driven and focused on prediction, whereas (traditional) EM is placed in the lower-left quadrant emphasizing being model-driven and focused on inference. The first continuum between inference and prediction relates to the type of generalizations the two approaches prioritize. Inference, specifically causal inference, revolves around estimating the effect of typically one theoretically relevant predictor variable on an outcome controlled for confounding effects of other variables. The actual inference is thus intuitively based on theoretical assumptions about the relationship between predictor and outcome, and empirically based on their association and the statistical modelling of uncertainty. If significant, the effect on the outcome is believed to generalize to the relevant population. On the other hand, prediction in its purest form changes the perspective such that it no longer is the effect of any specific predictor variable that is paramount. Instead predicting outcome values for new, unseen observations is prioritized. The prediction of the outcome values stems from local decision rules the algorithms develop when mapping relationships between the collective input data and outcome values in a so-called training data set, after which these exact local decision rules are applied to unseen observations in a so-called test data set. This latter part examines how well we predict out-of-sample, i.e. generalize what has been learned. To recap, EM is focused on identifying the effect of one predictor on the outcome, whereas ML is engaged with heightening the ability to predict outcome values for yet unseen observations dependent on the full feature space.

The second continuum between data-driven and model-driven methods relates to the amount of flexibility given vis-à-vis assumptions needed to implement different methods. This distinction relates to general differences between parametric and non-parametric models as well, with ML typically being more flexible and less reliant on prespecified assumptions compared to EM. The heightened flexibility is particularly apparent in two ways. First, ML algorithms are relatively often non-parametric and can handle/self-develop more advanced specifications of the input data, meaning that less assumptions about functional form, sparsity, etc., are needed to obtain unbiased, consistent results compared to a, generally speaking, more rigid starting point for EM. If flexibility is prioritized in parametric explanatory modelling, estimation of a greater number of prespecified parameters is required causing that noise rather than signal potentially is modelled – a phenomenon known as overfitting. Second, it is usually possible to adjust ML algorithms locally, i.e. in the specific application, through so-called tuning parameters. The adjustment of these parameters is almost always done in a purely data-driven fashion and if adjusted (i.e. tuned) wisely, they optimize the performance of the algorithm ex by combating overfitting.

Depending on whether the goal of the analysis is prediction, inference, or a combination of the two, different methods may be appropriate. For example, linear models allow for relatively simple and interpretable inference but may not yield as accurate predictions as some other methods. In contrast, some highly non-linear ML approaches that are can potentially provide quite accurate predictions, but this comes at the expense of a less interpretable model for which inference is more challenging. Related, Shmueli (2010) accentuates that predictive methods, and ML algorithms in general, have important analytical roles to fulfill. Large, detailed and rich data sets often contain complex relationships and patterns that are hard to hypothesize a priori. Using predictive modelling can help uncover relevant associations between variables which might enlighten new ways of measuring abstract concepts as well as improve or clarify the scope of existing, well-established explanatory models. As such, ML plays an important role in quantifying the level of predictability of measurable phenomena by creating benchmarks of predictive accuracy. Knowledge of unpredictability and uncertainty is a central component of scientific knowledge, and because predictive models tend to have higher predictive accuracy than explanatory statistical models, they provide an indication of the potential level of predictability. Phrased differently, such models perform a collective litmus test on the quality of input space, e.g. the constructed features and their specification. An explanatory model that is close to the predictive benchmark may suggest that our understanding of that phenomenon can only be increased marginally. On the other hand, an explanatory model that is very far from the predictive benchmark would imply that there are substantial practical and theoretical gains to be had from further model development.

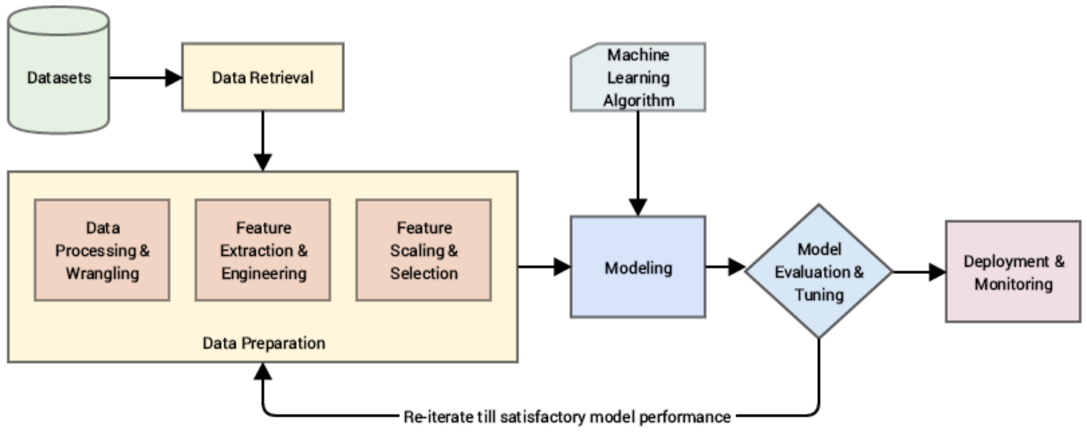
Before blindly applying ML algorithms, a sensible word of caution is appropriate: in any application, it is paramount to acknowledge that badly specified algorithms can produce biased results and quickly end up estimating noise rather than signal (see Weapons of Math Destruction by Kathy O'Neal; The Signal and The Noise by Nate Silver). **In short**: beware, be transparent, and do not go completely off-piste.

To sum up, ML is characterized by flexible methods that can be optimized in a data-driven manner to specific, local contexts and furthermore focuses on predictive performance as well as clarification of patterns in the data. The above comparison with EM is stylized, meaning that some methods and algorithms to a greater or lesser extent bridge the divide between ML and EM – i.e. follow the dotted line depicted in Figure 1.1. – while other by default do not fit in the figure particularly well. These interim methods are also included in coming chapters.

1.1 BASICS OF MACHINE LEARNING

The following presentation focuses on clarifying central elements of ML. Before diving into the substance, figure 1.2. presents a generic pipeline for machine learning projects. It should be read from left to right and, as such, deals with collecting data, preprocessing data, choosing algorithms, modelling, and evaluating performance. Furthermore, the figure emphasize that ML is a re-iterative process. This, and the other aspects of the pipeline, is commented throughout the forthcoming part of the presentation.

FIGURE 1.2.: GENERIC MACHINE LEARNING PIPELINE


Datasets
Data Retrieval
Machine Learning Algorithm
Data Processing & Wrangling
Feature Extraction & Engineering
Feature Scaling & Selection
Data Preparation
Modeling
Model Evaluation & Tuning
Deployment & Monitoring
Re-iterate till satisfactory model performance

Source: Practical Machine Learning with Python (p. 53), Apress

As basis for the presentation, I use the distinction between unsupervised and supervised learning. The two approaches are presented separately with a split focus on relevant concepts/theorical aspects and examples of the most prominent algorithms. Concepts include sample splitting and cross-validation, pre-processing features, the tradeoff between bias and variance focusing on overfitting, the role of tuning parameters, and evaluating the algorithms' performance. After the presentation, I briefly mention other uses for ML, e.g. in relation to imbalanced data sets and imputation of missing data.

1.1.1 UNSUPERVISED LEARNING

Unsupervised machine learning is focused on clarification of ex ante unperceivable patterns in data. It covers a set of tools intended for settings in which focus and/or access is limited to a set of p features, X_1, X_2, …, X_p, measured for n observations (i.e. only the input space, no response variable). Therefore, predictive modelling is not of interest (as it necessitates labeled outputs), and instead the goal is to infer and clarify natural structures present within the input space. Common use-cases for unsupervised learning are exploratory analysis and dimensionality reduction, i.e. discovering unspecified, but interesting, things about the measurements on X_1, X_2, …, X_p – e.g. by answering: Is there an informative way to visualize the data? Can we discover meaningful subgroups among the variables or observations?

Unsupervised learning is useful in exploratory analysis because it automatically identifies structure in the data. This relates to situations where it is either humanly impossible or impractical to propose trends in the data, and as such, exploratory analysis provide initial insights which then can be used to sum up the feature space or test potential hypotheses about structure. Unsupervised learning is furthermore useful for data-driven dimensionality reduction of the feature space, which covers methods used to represent data using less columns (features). Dimensionality reduction consists of learning relationships between individual features, thereby showcasing latent structures that interrelate the initial features. This sparse, latent structure can make further data processing less intensive or eliminate redundant features, and furthermore the latent structures can be used as separate features in proceeding analyses.

The most common algorithms within unsupervised learning are principal component analysis, clustering, sequence analysis, and density estimation. These are elaborated in section 1.1.1.2, i.e. after a brief introduction to the general concepts and theoretical aspects of unsupervised learning. The presentation draws on chapters 1 and 10 from *An Introduction to Statistical Learning* by James et al. (2013) and chapters 1, 8 and 9 from *Machine Learning with* R by Brett Lantz (2013).

1.1.1.1 CONCEPTS AND THEORY

Unsupervised learning is all about extracting patterns, relationships, associations, and clusters from data contingent on inherent structure in the feature space. Typically, focus is either on similarity between observations (across the feature space) or between features (across observations). Both foci necessitate a figurative understanding of the feature space as a token of different degrees of similarity – if for example two individuals share exact values on 90% of the features, they can be viewed as more alike than individuals only sharing 10%. Phrased differently, the distance between two individuals decreases as their similarity increases – and this lesser distance is understood to be intrinsically worthwhile investigating in most unsupervised learning. On a related note, this image of distance highlights that pre-processing features is important as different magnitudes across different features will unevenly affect how they each influence the understanding of distance afterwards. This will be underscored as the individual algorithms are presented below together with relevant tuning parameters. Compared to supervised learning, it is hard to evaluate the obtained results since there is no universally accepted mechanism for validating results on an independent test dataset set in unsupervised learning. If a predictive model is fitted using supervised learning, it is possible to check the performance by seeing how well the model predicts the response out-of-sample, i.e. on observations not used in fitting the model. However, in unsupervised learning, there is no way to check our work because we do not have labelled outputs (we do not know the "true" answer). Evaluation is possible, though, sometimes which I mention specifically.

1.1.1.2 MOST PROMINENT ALGORITMHS

In the following, I present some of the most prominent algorithms related to unsupervised learning. Most algorithms are employed in settings resembling the pipeline presented in figure 1.2 and as such, I will emphasize the importance of e.g. preprocessing features. The algorithms are principal component analysis, clustering, sequence analysis, and density estimation.

1.1.1.2.1 PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) refers to the process by which principal components are computed and used in understanding data. Principal components are used to summarize a large set of correlated numeric variables through a smaller number of representative variables that collectively explain as much of the variability in the original set as possible. PCA is an unsupervised statistical approach, since it involves only a set of features, X_1, X_2, …, X_p, and no response. The main idea is that each of the *n* observations initially lives in a *p*-dimensional space, in which PCA is seeking for a small number of dimensions (components) that are as interesting as possible. The concept of interesting is measured by the degree to which the observations vary along each dimension with higher variability implying that it becomes more interesting. Each of the dimensions found by PCA is a linear combination of the *p* features meaning that each initial feature is ascribed a certain weight with which in influences the observations placing on the principal component. Technically, the number of possible principal components are equal to *n* but typically, depending on the data set in question, only a few components are essential and necessary in order to capture most of the variability in the input space. Determining the number of principal components to use is usually done by looking at a scree-plot which is a line plot of their eigenvalues.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

Apart from producing derived variables, that can be used in supervised learning problems such as principal components regression, where principal components are used as predictors in place of the original set of variables, PCA also serves as a tool for data visualization. Suppose that we want to visualize *n* observations with measurements on a set of p features, X_1, X_2, …, X_p. This can be done by examining two-dimensional scatterplots, each one containing the *n* observations' measurements on two of the features. However, for example, with $p = 10$, this produces 45 plots indicating that, in general, if *p* is large, it becomes almost impossible to interpret at all of them; and moreover, most will not be informative since they each contain just a fraction of the total information. PCA is helpful as it introduces a low-dimensional representation of the data that captures as much of the information as possible. For instance, we can obtain a two-dimensional representation of the *p*-dimensional data which captures most of the information by plotting the observations in this low-dimensional, component-based space.

1.1.1.2.2 CLUSTERING

Clustering is an unsupervised algorithm and refers to a set of techniques for finding subgroups, or clusters, in data. When clustering observations, they are partitioned into distinct groups so that observations within each group are more similar to each other, while observations in different groups are more different.

Clustering is similar to PCA in that both seek to simplify the understanding of the data; they are distinct, though, in that clustering focuses on mapping homogenous subgroups among the observations while PCA focuses on finding a low-dimensional representation of the observations that explain a good fraction of the variance. Focus here will be on the two best-known clustering approaches: *K*-means and hierarchical. In *K*-means clustering, we seek to partition the observations into a pre-specified number of clusters while, in hierarchical clustering, we do not know in advance how many clusters we want; instead, a tree-like visual representation of the observations' indexation, called a dendrogram, allows us to view the clusterings obtained for each possible number of clusters, *n* to 1, and from this, and associated measures of similarity, assess the most meaningful number of clusters to deduce. In general, it is possible to cluster observations based on their features in order to identify subgroups among the observations, or cluster features on the basis of the observations in order to discover subgroups among the features. In what follows, I focus on clustering observations based on the input space, though the converse can be performed by simply transposing the data set.

In ***K*-means clustering** observations are placed into *K* pre-specified number of clusters. Choosing *K* is a crucial step when using this method and can be based in theoretical considerations or cross-validation procedures, where different values for *K* are tried out (see 1.1.2.1 for introduction to cross-valication). A good clustering output means that the within-cluster variation is as small as possible. The within-cluster variation is a measure of the amount by which the observations within one cluster differ from each other in terms of distance between them. The algorithm minimizes this optimization problem (the distance between all observations) and consequently, all included features will have to be numeric and rescaled to a similar scale beforehand. In order to obtain the distances, the observations are placed in a *p*-dimensional space with *p* being the number of features, you want to cluster across. In this *p*-dimensional space, the distance between observations can be calculated using different distance metrics such as Euclidean, Manhattan, etc. Usually, Euclidean distance is used. Visually it can be understood as the "ordinary" straight-line distance (either squared or in absolute terms to counter negative distances) between two data points in the *p* dimensions. The overarching idea is that closer data points are closer because they are more equal across the *p* features, and this uniformity is understood to be expressive of latent structure.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

The algorithm works in a couple of steps: first, it randomly assigns a number, from 1 to *K*, to each of the observations with *K* being the prespecified number of final clusters. The numbers serve as initial cluster assignments for the observations. Secondly, it iterates the following two steps until the cluster assignments stop changing: (a) For each of the *K* clusters, compute the cluster centroid, i.e. a point in the *p*-dimensional space. The *k*th cluster centroid is the vector of the *p* feature means for the observations in the *k*th cluster. (b) Assign each observation to the cluster whose centroid is closest (as calculated by Euclidean distance). When the result no longer changes, a local optimum has been reached. Because the *K*-means algorithm finds a local rather than a global optimum, the results obtained will depend on the initial (random) cluster assignment of each observation. For this reason, it is important to run the algorithm multiple times from different random initial configurations, and then select the best solution, i.e. the one for which the within-cluster distance is smallest. In other words, the within-cluster distance can be construed as a performance metric. There is no industry standard as to how large of a distance, you should accept. This relies on the data at hand.

Hierarchical clustering is an alternative clustering algorithm which overcomes the potential disadvantage in *K*-means of having to prespecify *K*. It has the added advantage that it results in a visually interpretable tree-based representation of the observations, called a dendrogram. Hierarchical clustering exists in two quite similar versions: agglomerative (bottom-up) and divisive (top-down). They differ in terms of which way, quite literally, the algorithms work. This becomes clearer in this section, where I primarily focus on the agglomerative edition as it is the most common type.

USE THIS IF…

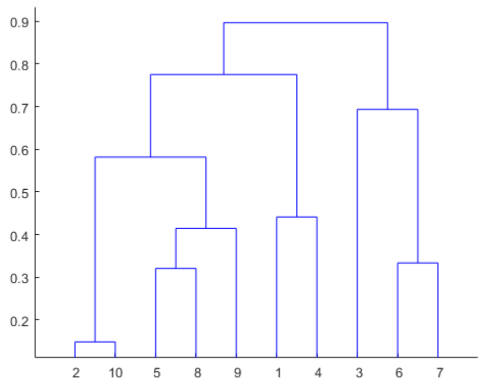
- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

In hierarchical clustering, the algorithms work in a couple of steps: Firstly, begin with *n* observations and a distance metric (such as Euclidean distance) of all the pairwise dissimilarities (i.e. calculate a dissimilarity matrix, $D_{n,n}$). Treat each observation as its own cluster (therefore the name bottom-up). Secondly, for $i = n$, n-1, …, 2 clusters, a) examine all pairwise inter-cluster dissimilarities among the *i* clusters in $D_{n,n}$ and identify the pair that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two

A DENDROGRAM



clusters (ex "2" and "10" in the picture above – they are fused as the first clusters) is indicated by the height in the dendrogram (the y-axis) at which the merged cluster is placed (0,1). b) Compute new pair-wise inter-cluster dissimilarities among the *i*-1 remaining clusters (see section on linkage criteria below for how to compute these). This is visually depicted in the dendrogram above containing 10 observations. It should be read from the bottom up.

Besides choosing distance metric at the beginning, choosing linkage criterium for the continuous fusing of clusters is also of importance. The main function of linkage criteria is the way in which they understand dissimilarities between clusters (i.e. not the single observations). There are several different options – Ward's, complete, average, single and centroid to name a few – and it depends on the context whether one is favorable. Ward's linkage is a viable option if you have a good reason for prioritizing maximizing cluster homogeneity *as much as possible*. At each iteration, it decides which clusters to fuse by minimizing the total within-cluster sum of squared errors – with error being the distance from any single observation in one cluster to their geometrical centroid. The two clusters, by which the total squared error is height-ened the least, are fused. Complete, average and single are simpler and focus, respectively, on the distance between only the two furthest observations in two clusters, the average distance between all observations in two clusters, and the distance between the two closest observations in two clusters. Minimizing the total distance is prioritized in alle om them. Centroid, finally, focuses on the distance between two clusters centroids. In general, average, complete and single – in the mentioned order – are probably most com-mon. When choosing linkage criteria, consider how much every *single* observation (e.g. also outliers) vis-à-vis the aggregate manifestation of observations in each cluster should weigh.

Choosing the optimal place to "cut" the dendrogram is the last important step when using hierarchical clustering. Cutting the dendrogram is equal to choosing the number of clusters, from 1 to *n*, to deduce. At first glance, this seems difficult as the absolute lowest error term is obtained if each observation "stays" as a single cluster. This defeats the purpose of clustering. Instead the dendrogram, or what is usually called an "elbow diagram" highlighting how much the error changes from each iteration, can be helpful. In general, cut the dendrogram when the error-related "win" of dividing two clusters further is not "worth it" due to theoretical or empirical considerations. Beware: clustering algorithms will always produce some result, i.e. place observations into groups – that is not to say these necessarily are meaningful.

1.1.1.2.3 SEQUENCE ANALYSIS

Sequence analysis constitutes another approach used in exploratory analysis and dimensionality reduction. In itself, sequence analysis is only capable of producing visualizations but combined with a clustering algorithm, sequencing can also be used analytically. I will elaborate on this point after an introduction to the main elements from sequence analysis. This presentation builds on *Social Sequence Analysis* (2015) by Benjamin Cornwall.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

Sequence analysis is used for analyzing sequences. Sequences comprise time-ordered vectors, $i_{1,c}$, of categorical states, individuals experience. The notions of "states" and "time-ordered" are used in an all-encompassing fashion meaning that almost anything can be "a state" and almost anything can be used as the time-ordering element (for example yearly educational status from 10 to 14 years old). The analysis-dimension of "sequence analysis" is focused on illustrating homogeneity and representativity among the sequences. Using TraMineR (Gabadinho et al. 2011) it is possible to visualize most frequent sequences, state-distribution across time, and more (see Gabadinho et al., 2011). For example, given data on 9th graders educational and work experiences (states) for four years, on a yearly basis, after completing primary school, it is possible to visualize how many are following all possible four-state trajectories as well as how many who are attending ex high-school in the first year after 9th grade. These graphs can be split on covariates.

As mentioned, it is also possible to use sequence analysis in a more analytical fashion besides graphically. Examining homogeneity is again at center stage, this times in quantitative sense. Imagine a data set with rows constituting observations and columns (variables) constituting time-ordered states, one per column. Each row now becomes one of the aforementioned individual vectors, $i_{1,c}$, and from this, it is possible to calculate how (dis)similar every row (vector) is from each other. The dissimilarities between all individual sequences can be used as content in the dissimilarity matrix, $D_{n,n}$, and hence feed into a clustering algorithm. As such, the sequences will be grouped into clusters with similar sequences indicating that these individuals have experienced the same trajectories ("time-ordered states"). The most common approach to calculating the dissimilarities is called Optimal Matching (a "distance measure") in which each pairwise dissimilarity (each cell in $D_{n,n}$) constitutes the lowest total cost possible necessary in order to transform one sequence into the other. "Total cost" is the total amount of operations the transformation necessitate (with operations comprising insertions/deletions (called indels; inserting or deleting a single state in the sequence) or substitutions (changing one state to another)), multiplied individually by their prices (think of them as weights). The price for one indel and one substitution can be decided on in a data-driven fashion (ex it may make sense that a substitution between two generally frequent states should cost relatively more/less than a substitution between two generally infrequent states) or in lieu of theoretical considerations. Besides choosing distance measure (Optimal Matching is the most typical and flexible; alternatives are Hamming, Levenshtein, Longest Common Subsequence and Longest Common Prefix) and setting operation costs, deciding on whether to normalize the distances is paramount for clustering purposes. Distance normalization is especially worth considering if the sequences are not of exact same length, i.e. if some vectors are missing states. Normalization can be performed in different ways, ex by diving by the length of the longest sequence or pairwise by the geometric average. Graphically and analytically, it is possible to investigate the clusters in more depth. See Gabadinho et al. (2011) and Studer (2013) for measures of quality etc. It is important to question the outcome thoroughly.

1.1.1.2.4 PROBABILITY DENSITY ESTIMATION

Probability density estimation is a general concept within machine learning and constitutes a central element in the application of many different algorithms. In general, it is focused on knowing the probability distribution for a random variable which can calculate its moments and as such, it is useful for general considerations like determining whether an observation is unlikely or very unlikely and might be an outlier or anomaly. I focus primarily on the detection of outliers and anomalies in the presentation. Detection is important when specifying machine learning algorithms as they might distort the training or testing and thus the model's generalizability.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

The starting point in probability density estimation is that the relationship between the outcomes of a random variable, *x*, and its probability distribution, *p*(*x*), is referred to as the probability density. If a random variable is continuous, the probability can be calculated via probability density function. The shape of the probability density function across the domain for a random variable is referred to as the probability distribution and common probability distributions have names, such as uniform, normal, and exponential. Given a random variable, we are interested in the density of its probabilities, e.g. the shape of the probability distribution, the most likely value, and the spread of values. The problem is that we rarely know the absolute distribution for a random variable up-front because we do not have access to all its possible outcomes; they must be estimated. This problem is referred to as probability density estimation as we are using the observations in a random sample to estimate the general density of probabilities beyond just the sample of data. In the following paragraphs, I focus on univariate data, e.g. one random variable, for simplicity. Although it is applicable for multivariate data, it becomes more challenging as the number of variables increases.

There are a few steps in the process of density estimation for a random variable. The first step is to review the density of observations in the random sample with a simple histogram. From the histogram, it might be possible to identify a common, well-understood probability distribution that can be used, such as a normal distribution. If not, you must estimate the distribution which can be done in either a parametric or nonparametric manner. In parametric estimation, the shape of the histogram should match a well-known probability distribution. Once identified, you can estimate the density. For example, the normal distribution has two parameters: the mean and the standard deviation. Given these two parameters, we know the probability distribution function. This process is referred to as parametric density estimation. The reason is that predefined functions are used to summarize the relationship between observations and their probability that can be controlled or configured with parameters.

In some cases, a sample of data may not resemble a common probability distribution or be easily manipulated to fit the distribution. In these cases, the data might have two peaks (bimodal distribution) or many (multimodal distribution). As such, parametric density estimation is not feasible and instead, an algorithm is used to approximate the probability distribution of the data without a pre-defined distribution, referred to as a nonparametric method. The most common nonparametric approach for estimating the probability density function of a continuous random variable is called kernel smoothing or kernel density estimation used to estimate probabilities for new data points. A kernel is a mathematical function that returns a probability for a given value of a random variable. The kernel effectively smooths or interpolates the probabilities across the range of outcomes for a random variable such that the sum of probabilities equals one (1), a requirement of well-behaved probabilities. The kernel function weights the contribution of observations from a data sample based on their relationship or distance to a given query sample for which the probability is requested. A tuning parameter, called the smoothing parameter (bandwidth), controls the scope, or window of observations, from the data sample that contributes to estimating the probability for any new sample/observation. The value for this parameter is usually determined using cross-validation procedures (see 1.1.2.1 for introduction to cross-valication). A large window may result in a coarse density with little details, whereas a small window may have too much detail and not be smooth or general enough to correctly cover new or unseen examples. The contribution of samples within the window can be shaped using different kernel functions, sometimes referred to as basis functions, e.g. uniform, normal, etc., with different effects on the smoothness. As such, it may be useful to experiment with different window sizes and types of kernels and evaluate the results against histograms. The definition of what constitutes an outlier is furthermore necessary to determine beforehand. There is no general rule as to how much deviation there should be permitted – it should be decided as a function of the amount and type of available data.

Transition section: how unsupervised can feed into supervised.

1.1.2 SUPERVISED LEARNING

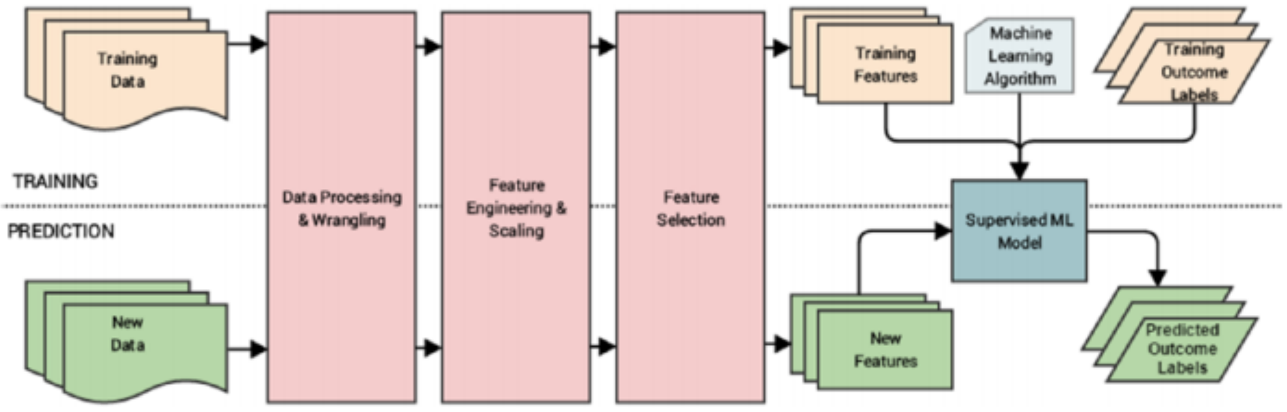
Supervised machine learning is an umbrella term for methods and algorithms primarily focused on prediction. The earlier comparison with explanatory statistical modelling (section 1.0) provides a baseline understanding of its focus. To recap: by mapping relationships between the feature space and response values for observations in a training data set, supervised methods develop local decision rules (typically by minimizing some loss function) which afterwards are applied to yet unseen observations in a so-called test data set in order to evaluate out-of-sample generalizability. Practically speaking, the test observations' values on their input features determine (by way of the learned decision rule) which response value, they are predicted to have. The predicted values are then compared to the observations' actual response values to assess the algorithm's predictive success. Phrased differently, supervised learning revolves around learning a function that, given a sample of data and outputs, best approximates the relationship between observable input and output – with "best" being evaluated out-of-sample.

Supervised learning is typically done in the context of either *classification*, when mapping input features/predictors to a categorical output, or *regression*, when mapping input to a continuous output. In both regression and classification, the goal is to find specific relationships or structures in the input data that allow us to effectively and correctly predict the output. Note that "correctly" is determined entirely from the training data, so while we do have something resembling "a baseline truth" (that our model assumes is true), it is not to say that predictions are always correct in real-world situations. Noisy or incorrectly labelled data will reduce the effectiveness of the model, and furthermore when conducting supervised learning you need to consider model complexity and the bias-variance tradeoff to obtain reliable results. Common algorithms in supervised learning include linear and logistic regression, K nearest neighbors, support vector machines, neural networks, tree-based algorithms such as random forests, and ensembles. These are presented in section 1.1.2.2, i.e. after an introduction to the general concepts and theoretical aspects of supervised learning. The presentation draws on *An Introduction to Statistical Learning* by James et al. (2013), *Machine Learning with* R by Brett Lantz (2013), *Practical Machine Learning with Python* by Sarkar et al. (2017) and other sources mentioned throughout.

1.1.2.1 CONCEPTS AND THEORY

When applying supervised machine learning there is a well-established pipeline building on similar principles as figure 1.2. The pipeline for supervised learning is depicted in figure 1.3. below covering the most important aspects worth considering before jumping headfirst into action. As such, it highlights the different dimensions I will delve further into in this section. I touch briefly on sample splitting, pre-processing features, choosing and tuning algorithms, and evaluating performance. If relevant, these and other aspects are also discussed specifically in the presentation of each algorithm.

FIGURE 1.3.: SUPERVISED MACHINE LEARNING PIPELINE



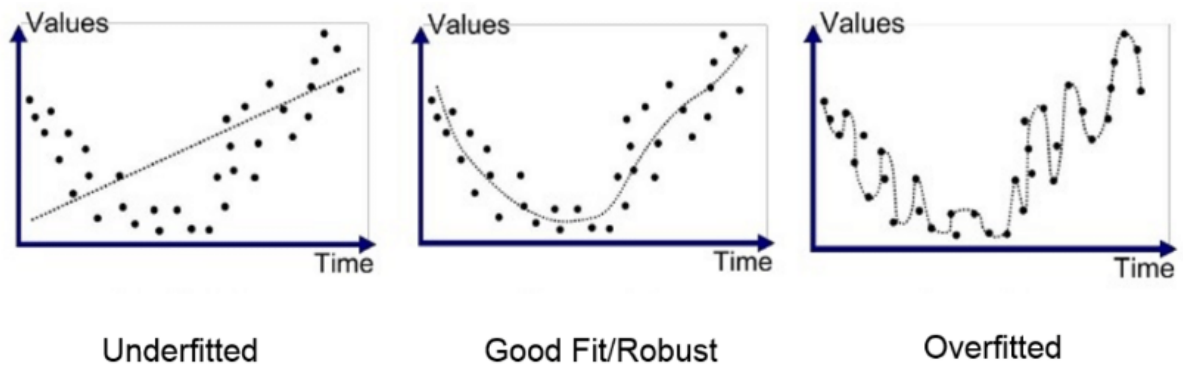
Source: Practical Machine Learning with Python (p. 54), Apress

To understand the pipeline, begin at the left-hand side and move rightwards. The first step is to retrieve and split data into training data and new data (usually called test or hold-out data). This process is called sample splitting which is necessary in order to evaluate the algorithm's predictive performance out-of-sample after it has been trained. After splitting, each data set is preprocessed separately but in a similar manner. It is important **not** to preprocess the data before sample splitting for two reasons: firstly, if the test data is included when preprocessing training data you are introducing future (test) data into the preprocessing process of past (training) data (for example when calculating mean and standard deviation in a standardization procedure). As a general rule, this is complete no-go: we are trying to learn from the past and predict the future, not the other way around. Secondly, e.g. pre-processing trough standardizing before sample splitting would mean that you do not necessarily obtain the desired distribution ($\mu=0$, $\sigma=1$) across all features in both training and test data.

Specifically, preprocessing data includes cleaning, wrangling, and manipulating it. This cover extracting important features from the raw data and creating/engineering new features from existing ones. Furthermore, the features often need to be scaled to prevent algorithms from becoming biased. This is essential when using algorithms that are distance-contingent in their training due to similar reasons as in unsupervised learning: it is necessary to counteract certain features from dominating the others, solely because of their scale or magnitude. Scaling can be implemented in a couple of ways, i.e. through normalization or standardization. Usually, standardization – for each feature, subtract the mean and divide by the standard deviation – is employed (with Andrew Gelman, one of the big shots, arguing for dividing by two times the standard deviation instead to mimic the distribution of dummy variables; see Gelman, 2007: 3). In addition to scaling issues, the presence of categorical features, missing data and outliers are important to handle. Most machine learning algorithms are not fond of categorical features and therefore it is common practice to recode each value on the categorical feature as a single dummy feature and then leave one out (the control group) when training and testing the model to avoid multicollinearity. This is called one-hot encoding. Missing data is another issue, which most algorithms are not able to handle. There are two ways to address missing data: drop observations with missing values or imputation. In section 1.1.3, I present ML approaches to imputation. Lastly, outliers are, as per usual, relevant to detect and possibly handle as they will distort the training and/or testing of the model. This is once again mostly an issue when working with distance-contingent algorithms.

Following the initial preprocessing process, it is often appropriate to select a subset of the available features based on their importance and quality to ensure the most optimal tradeoff between variance and bias. In this regard, variance refers to the amount by which model estimation would change if we estimated it using a different training data set. Since training data are used to fit the algorithm, different training data sets will result in different estimations of models. Ideally, the estimates should not vary too much between training sets. However, if a method has high variance then small changes in the training data results in large changes in model performance. In general, more flexible statistical methods have higher variance. Consider the right-hand section in Figure 1.4.: the flexible curve is following the observations very closely. It has high variance because changing any of the data points may cause the estimate of the model to change considerably. This phenomenon is called overfitting indicating that the model is overly reliant on the *exact* training data. In contrast, the curve on the left-hand side is relatively inflexible and has low variance, because moving any single observation will likely cause only a small shift in the position of the line. As such the model is underfitted.

FIGURE 1.4.: BIAS, VARIANCE, UNDER- AND OVERFITTING

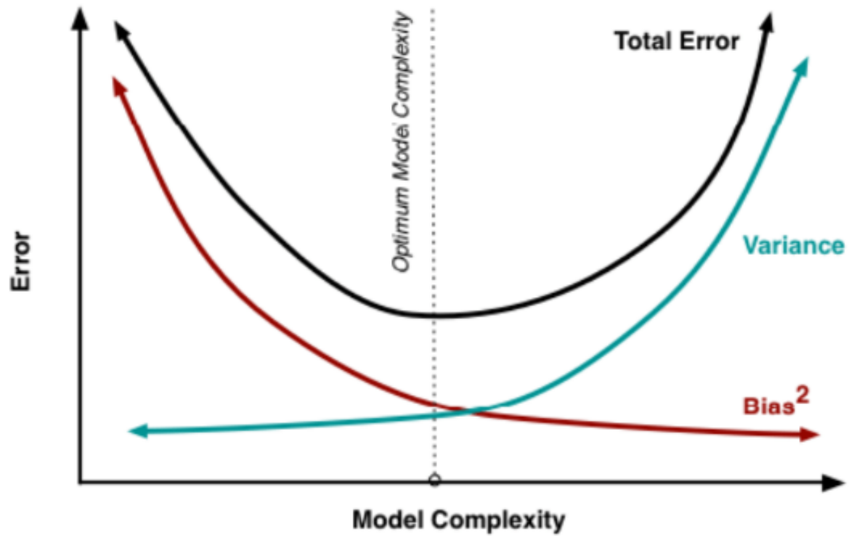


Note: XXX

On the other hand, bias refers to the estimation error that is introduced by approximating a real-life problem, which may be extremely complicated, by a (much simpler) model. Phrased differently, bias constitutes the degree to which the model correctly approximates the true relationship in the training data. For example, linear regression assumes that there is a linear relationship between Y and X_1, X_2, …, X_p. It is unlikely that any real-life problem truly has such a simple linear relationship, so performing linear regression will undoubtedly result in some bias in the estimation. In Figure 1.4. above, the true curve is substantially non-linear, so no matter how many training observations we are given, it will not be possible to produce an accurate estimate using linear regression. Generally, more flexible methods result in less bias while the variance increases. The relative rate of change between these two quantities determines whether the test mean square error (MSE), i.e. the total error, increases or decreases (see figure 1.5. below). As flexibility/complexity increases, bias tends to initially decrease faster than variance increases. Consequently, the expected test MSE declines. However, at some point increasing flexibility has little impact on the bias but starts to significantly increase the variance. When this happens the test MSE increases.

This relationship, between bias, variance, and test MSE displayed in Figure 1.5. below, is referred to as the bias-variance tradeoff and optimizing the total error-function for its minimum is typically one of the most important tasks when using any predictive algorithms. Luckily, the optimization process is often built into the modelling process.

FIGURE 1.5.: BIAS-VARIANCE TRADEOFF



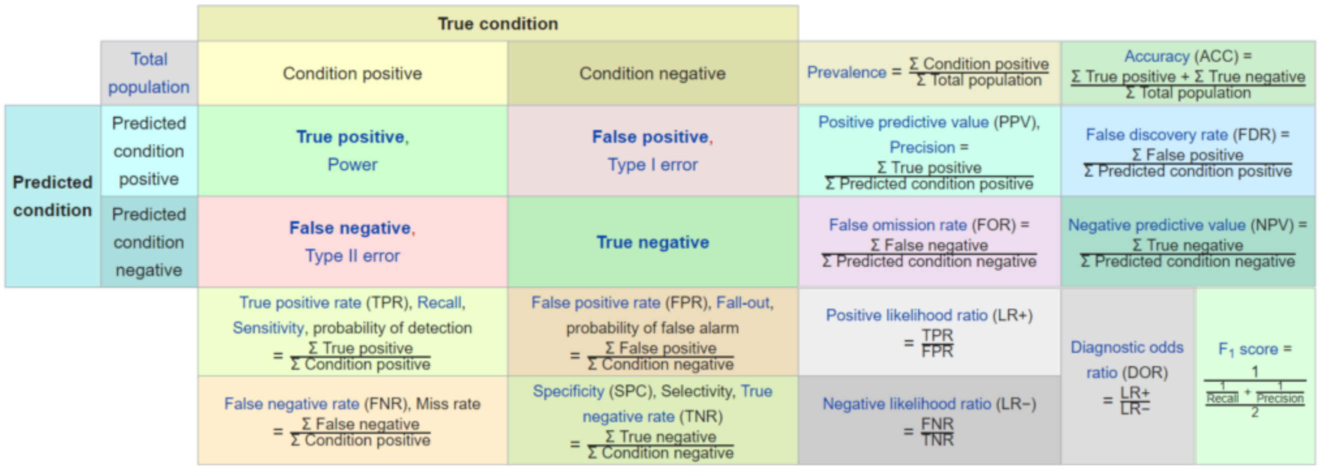
Note: XXX

The bias-variance tradeoff is important because it influences model generalization. In order to estimate models that generalize well, the inherent variance of the model should preferably scale with the size and complexity of the data – small, simple data sets should typically be learned with low-variance models, while large, complex data sets often require higher-variance models to fully learn its structure. When choosing the model, the specific problem and nature of data should inform the decision on where to fall on the bias-variance spectrum. Generally, increasing bias (and decreasing variance) results in models with relatively guaranteed baseline levels of performance, which may be important in certain tasks.

With the bias-variance tradeoff in mind, the next step is choosing an algorithm and tuning it using the training data. Choosing algorithm is naturally dependent on a lot of contextual considerations and the forthcoming presentation of algorithms will hopefully clarify when each makes sense to use. Guiding principles for choosing, will be stated in the beginning of section 1.1.2.2. After choosing an algorithm, tuning becomes central. Tuning encompass the process of initially setting different parameters as well as the optimization strategy, i.e., tuning can be thought of as optimizing the parameters that impact the model in order to enable the best performance. It is oftentimes not possible to know in advance which values for the tuning parameters that ensures the best model. Instead, the normal approach is to conduct a so-called 'grid search', which means that different compositions are tested in order to obtain the most accurate model. There exists a lot of different tuning parameters and therefore, I present them in relation to the specific algorithms.

Once tuned and trained, the algorithm is evaluated on the test data in order to establish generalizability. The evaluation builds on performance metrics. There exists a lot of different metrics, cf. figure 1.6. below in relation to classification, with accuracy, specificity and sensitivity being the most prominent. In regression tasks, mean squared error or mean absolute error are typically used. Sometimes one or more specific performance metrics are of special importance in a given analysis – if/when employing an algorithm for predicting whether a patient is suffering from a deadly illness, it could be argued that falsely predicting ill patients as non-ill should be avoided at a higher rate than non-ill patients predicted as being ill (even though this, of course, is not optimal either). In such cases, sensitivity is more important than specificity. Such considerations are also relevant in the tuning of the algorithm as it oftentimes is possible to adjust the training so that certain performance metrics are prioritized.

FIGURE 1.6.: PERFORMANCE METRICS



		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F$_1$ score = $\frac{1}{\frac{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}{2}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

Note: https://en.wikipedia.org/wiki/Confusion_matrix

A word on ROC and confusion matrices.

As a last caveat to the supervised machine learning process, cross-validation is worth mentioning as many practical implementations of machine learning are set in a cross-validation schemes. Through cross-validation the stability and effectiveness of the model's performance is checked and in addition, cross-validation is useful in determining the optimal values for the beforementioned tuning parameters as it effectively lowers the probability of overfitting or underfitting the model. Cross-validation, specifically k-fold cross-validation, involves randomly dividing the data set of observations into k groups, or folds, of approximately equal size. In the first run, the first fold is treated as test data while the algorithm is trained on the remaining k-1 folds. This process is repeated k times, each time with a new $1/k$ held out for testing. Choosing value for k is important, but there is no formal rule as to how. Typically, cross-validation is performed using $k = 5$ or $k = 10$, as these values have been shown empirically to result in test error rates that suffer neither from excessively high bias or variance. Leave-one-out cross-validation is another possibility in which the value for k is fixed to the size of the dataset, n, to give each test sample an opportunity to be used as test data. This approach is only used in smaller data sets.

1.1.2.2 MOST PROMINENT ALGORITHMS

In the following, I present some of the most prominent algorithms related to supervised machine learning. There exists an ever-expanding corpus of algorithms and consequently, I will not be able to cover the whole lot in this presentation. The inclusion criteria are first and foremost that the algorithms are widely used and relevant for the work done in RFI, and, secondly, that they differ from each other in terms of their main purpose and learning processes. Regarding main purpose, algorithms are typically focused on either prediction or more data mining-like questions with prediction being equal to the earlier depiction (i.e. heightening the predictive accuracy of Y from the unified input space, X) while data mining-like questions are focused on understanding which of the features in the input space that are most important, albeit also in a predictive context. This use of the term "data mining" is not widespread and should primarily be understood heuristically. In relation to learning processes, algorithms can be error-based (regression, regularization and support vector machines), information-based (classification and regression trees), similarity-based (nearest neighbor algorithms) or ensemble-based (random forests). I will walk though algorithms based on the inherent complexity of their learning process and mention if and how they are useable in both pure prediction and data mining.

1.1.2.2.1 K-NEAREST NEIGHBORS

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

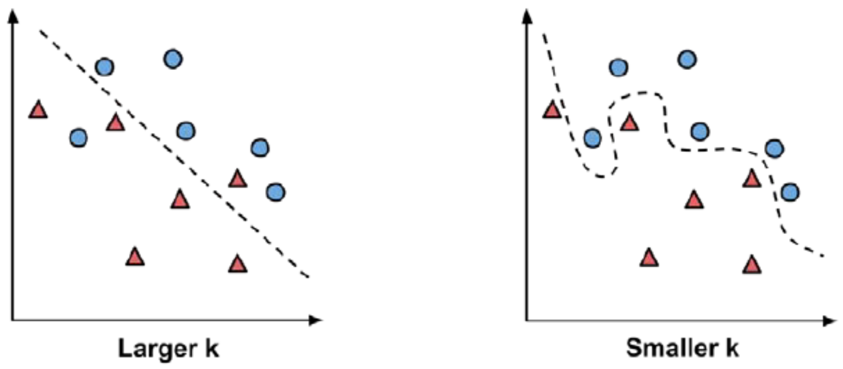
- B
- B
- B

K-nearest neighbors (*k*-NN) is a classification algorithm defined by classifying unlabeled examples by assigning them the class of similar labeled examples – i.e. their "nearest" neighbors. Despite its immediate simplicity, nearest neighbor methods are usually powerful. The algorithm can be understood on basis of an idiom: "If it smells like a duck and tastes like a duck, then you are probably eating duck", i.e. things that are alike are likely to have properties that are alike. In general, nearest neighbor classifiers are well-suited for classification, when relationships among the features and the target classes are numerous, complicated, or difficult to understand, yet the items of similar class type tend to be homogeneous. Another way of putting it would be that if a concept is difficult to define, but you know it when you see it, then nearest neighbors might be appropriate. On the other hand, if the data is noisy and thus no clear distinction exists among the groups, the nearest neighbor algorithms may struggle to identify class boundaries.

The *k*-NN algorithm gets its name from the fact that it uses information about an example's *k* nearest neighbors to classify unlabeled examples. *k* is a tuning parameter implying that any number of nearest neighbors could be used. After choosing *k*, the algorithm requires training data made up of examples that have been classified into several outcome categories, as labeled by a nominal variable. Then, for each unlabeled record in the test dataset, *k*-NN identifies *k* records in the training data that are the "nearest" in similarity. The unlabeled test instance is assigned the class of most of the *k* nearest neighbors. Similarity is usually based on the smallest Euclidean distance between examples in a *p*-dimensional space with *p* being the number of features used in the classification. Using Euclidean distance (or any other distance measure for that sake) necessitates one-hot dummy coding of nominal features and rescaling the input space e.g. by standardization such that each feature is numeric and on the same scale.

The decision of how many neighbors to use for *k*-NN determines how well the model will generalize to future data. The balance between overfitting and underfitting is relevant as choosing a large k reduces the impact or variance caused by noisy data but may bias the learner so that it runs the risk of ignoring small, but important patterns. Suppose we *k* equaling the total number of observations in the training data. Then, with every training instance represented in the final vote, the most common class always has most of the voters. The model would consequently always predict the majority class, regardless of the nearest neighbors. On the opposite extreme, using a single nearest neighbor allows the noisy data or outliers to influence the classification. For example, suppose some of the training examples were accidentally mislabeled. Any unlabeled example that happens to be nearest to the incorrectly labeled neighbor will be predicted to have the incorrect class, even if nine other nearest neighbors would have voted differently. Obviously, the best *k* value is somewhere between these two extremes. The following figure illustrates, more generally, how the decision boundary is affected by larger or smaller *k* values. Smaller values allow more complex decision boundaries that more carefully fit the training data.

FIGURE 1.7: IMPLICATION OF K IN K-NN



Note: Brett (2013: 71).

In practice, choosing *k* depends on the difficulty of the concept to be learned, and the number of records in the training data. One common practice is to begin with *k* equal to the square root of the number of training examples. However, such rules may not always result in the single best *k*. An alternative approach is to test several *k* values on a variety of test datasets and choose the one that delivers the best performance. That said, unless the data is very noisy, a large training dataset can make the choice of *k* less important. This is because even subtle concepts will have a sufficiently large pool of examples to vote as nearest neighbors. Another solution to the problem is to choose a large *k* but apply a weighted voting process in which closer neighbors is considered more authoritative than those far away. Most *k*-NN implementations offer this option. The strengths of *k*-NN is its simplicity and effectiveness and that it makes no assumptions about the underlying distribution of data. The weaknesses are that it does not produce a model per se, limiting our ability to understand how the features influence class-membership and that it requires selection of an appropriate *k* which can lead to highly variable results.

1.1.2.2.2 TREE-BASED ALGORITHMS

USE THIS IF…

- B
- B
- B

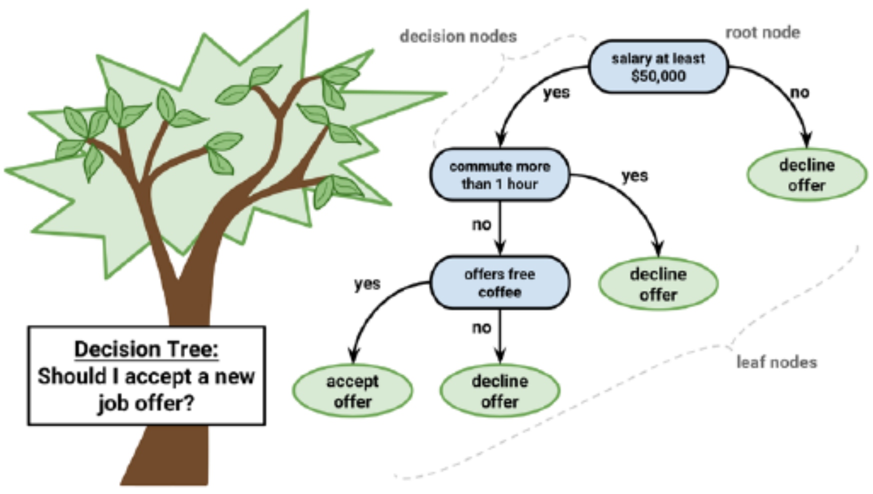
REMEMBER TO CONSIDER…

- B
- B
- B

Tree-based algorithms cover a multitude of different algorithms all based on what visually resembles a tree. I will touch on decision trees and regression/model trees as well as random forests, boosting and bagging which are examples of ensembles algorithms. They are usually are applied in a tree-setting but are, technically speaking, usable across any type of method. In general, these tree-based algorithms are worthwhile applying due to their interpretability and flexibility. They fit most problems, can handle numeric and nominal data, and exclude unimportant features by automatically.

A **decision tree** is a classification algorithm. It is based on a heuristic: when facing a complex decision, it is easier and more meaningful (providing more consistent, thorough answers) to consider a series of simpler questions and from the answers to those questions, and their sequence, deduce an aggregate answer (i.e. a prediction). Decision tree learners are powerful classifiers, which utilize a tree structure to model the relationships among several features and the potential outcomes on a nominal response. As illustrated in figure 1.8., this structure earned its name since it mirrors how a tree begins at a wide trunk, which if followed upward, splits into narrower and narrower branches.

FIGURE 1.8.: STRUCTURE OF TREE-BASED ALGORITHMS



Note: Brett (2013: 126).

In much the same way, a decision tree classifier uses a structure of branching decisions, which channel examples into a final predicted class value. To better understand how this works, consider the tree above, which predicts whether a job offer should be accepted. Considering a job offer begins at the root node (salary) and is then passed through decision nodes that require choices to be made based on the attributes of the job (importance of "commute" and "coffee"). These choices split the data across branches that indicate outcomes of a decision, depicted here as "yes" or "no" (there may be more than two possibilities). The tree is terminated by leaf nodes also known as terminal nodes. In the case of a predictive model, the terminal nodes provide the expected result given the series of events.

A benefit of decision tree algorithms is the flowchart-like tree. After the model is created, the algorithm outputs the structure in a visual, interpretable format. This provides insight into how and why the model works in a given context. This also makes decision trees appropriate for applications in which the classification mechanism needs to be transparent. Decision trees are perhaps the single most widely used machine learning technique and can be applied to model almost any type of data. This said, despite their wide applicability, it is worth noting where trees may not be an ideal fit. One such case might be a task where the data has many nominal features with many levels, or it has a large number of numeric features. These cases may result in a very large number of decisions and an overly complex tree. They may also contribute to the tendency of decision trees to overfit data.

Decision trees are built using recursive partitioning. This approach is also known as "divide and conquer" because it splits the data into subsets, which are then split repeatedly into even smaller sets until the process stops when the algorithm determines that the subsets are sufficiently homogenous, or another stopping criterion has been met. To see how splitting a dataset can create a decision tree, imagine a bare root node that will grow into a mature tree. At first, the root node represents the entire dataset, since no splitting has occurred. Next, the algorithm must choose a feature to split upon; ideally, it chooses the feature most predictive of the target class. The examples are then partitioned into groups according to the distinct values of this feature, and the first set of tree branches are formed. Working down each branch, the algorithm continues to divide the data, choosing the best candidate feature each time to create another decision node, until a stopping criterion is reached. It might stop at a node if: 1) all (or nearly all) of the examples at the node have the same class, 2) there are no remaining features to distinguish among the examples, 3) the tree has grown to a predefined size limit.

The first challenge that a decision tree faces is to identify which feature to split upon: the main rule is to split the data such that the resulting partitions contains examples primarily of a single class. The degree to which a one-branch-subset of examples contains only a single class is known as purity, and any subset composed of only a single class is called pure. There are various measurements of purity that can be used to identify the best decision tree splitting candidate. Usually entropy is used. Entropy is a concept borrowed from information theory that quantifies the randomness, or disorder, within a set of class values. Sets with high entropy are very diverse and provide little information, as there is no apparent commonality. The decision tree looks for splits that reduce entropy, thereby iteratively increasing homogeneity within the groups. If there are only two possible response classes, entropy values can range from 0 to 1. For *n* classes, entropy ranges from 0 to $log_2(n)$. In each case, "0" indicates that the subsets are completely homogenous, while the maximum value indicates that the data are as diverse as possible. In determining the optimal feature to split upon, the algorithm calculates the change in homogeneity (change in entropy) that would result from a split on each possible feature, which produces a measure known as information gain. The information gain for a feature is calculated as the difference between the entropy in the segment before the split and the partitions resulting from the split. One complication is that after a split, the data is divided into more than one partition (iteratively adding one partition more). Therefore, you need to consider the total entropy (the sum) across all partitions at each iteration. It does this by weighing each partition's entropy by the proportion of records falling into the partition. Decision trees use information gain for splitting on numeric features as well. To do so, a common practice is to test various feature splits that divide the observations into groups being greater than or less than a numeric threshold on the feature. The numeric cut point yielding the largest information gain is chosen for the split.

A decision tree can continue to grow indefinitely, i.e. divide the data into smaller and smaller partitions until each example is perfectly classified or the algorithm runs out of features to split on. However, if the tree grows too large, many of the decisions it makes will be overly specific and the model will overfit. The process of pruning a decision tree involves reducing its size such that it generalizes better to unseen data. One solution is to stop the tree from growing once it reaches a certain number of decisions or when the decision nodes contain only a small number of examples. This is called early stopping or pre-pruning. One downside to this approach is that there is no way to know whether the tree will miss subtle, but important patterns that it would have learned had it grown to a larger size. An alternative, called post-pruning, involves growing a tree that is intentionally too large and pruning leaf nodes to reduce the size of the tree to a more appropriate level. This is often a more effective approach than pre-pruning, because it is quite difficult to determine the optimal depth of a tree without growing it first. Pruning later allows the algorithm to be certain that all the important data structures are discovered.

One of the benefits of the decision tree algorithm is that it is opinionated about pruning – it takes care of many decisions automatically using fairly reasonable defaults. Its overall strategy is to post-prune. It first grows a large tree that overfits the training data. Later, the nodes and branches that have little effect on the classification errors are removed. In some cases, entire branches are moved further up the tree or replaced by simpler decisions. These processes of grafting branches are known as subtree raising and subtree replacement, respectively. Balancing overfitting and underfitting a decision tree are a bit of an art, but if model accuracy is vital, it may be worth investing some time with various pruning options to see if it improves the performance on test data.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

Regression trees and **Model trees** are similar to decision trees in their structure and visual expression but instead of being used for classification, they are used for numeric prediction. Despite the name, **regression trees** do not use regression methods. Rather they make predictions based on the average value on the response variables across examples that reach a given leaf, i.e. following the splits in the tree toward the terminal nodes, the averages obtained for each subset of observations should be more and more distinct. As such, trees for numeric prediction are built in much the same way as they are for classification. Beginning at the root node, the data is partitioned using a divide-and-conquer strategy according to the feature that will result in the greatest increase in homogeneity in the outcome after a split is performed. In classification trees, homogeneity is measured by entropy, which is undefined for numeric data. Instead, homogeneity is measured by statistics such as variance, standard deviation, or absolute deviation from the mean. One common splitting criterion (similar to Information Gain in classification) is called the Standard Deviation Reduction (SDR). It measures the reduction in standard deviation by comparing the weighted standard deviation pre-split to the weighted standard deviation post-split. **Model trees** are grown in much the same way as regression trees, but at each leaf, a multiple linear regression model is built from the examples reaching that node. As such, model trees are more complex than regression trees which only uses one feature per split. Depending on the number of leaf nodes, a model tree may build tens or even hundreds of such linear models. This make model trees more difficult to understand with the benefit that they may result in a more accurate model.

Though traditional regression methods are typically first choice for numeric prediction, in some cases, regression or model trees offer distinct advantages. For instance, trees may be better suited for tasks with many features or many complex, non-linear relationships among features and outcome – in a traditional regression setup, these would have to be prespecified which enhances the probability of overfitting. Regression modeling also makes assumptions about how numeric data is distributed that are often violated in real-world data. This is not the case for trees.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

Bagging, **random forests** and **boosting** are three somewhat similar refinements of the tree algorithms. They are usable in both classification and regression settings as well as for data mining purposes. The algorithms are so-called ensembles which, in general, describes settings where multiple "weak learners" are combined to create one "strong learner". The "weak learners" join forces and create a boyband, so to speak, under the assumption that the sum (or average) of one Niall, one Louis, one Zayn, one Harry, and one Liam is performing better than either one alone. Averaging over the multiple directions for model estimation offers one viable direction to pursue. Generally, a learner is weak if small changes in the training data cause relatively large changes in the estimation out-of-sample – and specifically, the variance of a single decision tree is rather high. However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved due to a more balanced bias-variance trade-off than a single tree.

Bagging (short for bootstrap aggregation) builds on a powerful idea called the bootstrap. The bootstrap, or bootstrapping, is a resampling method (random sampling with replacement) and it is used in many situations in which it is hard or even impossible to directly compute the standard deviation of a quantity of interest. Regarding bagging, bootstrapping is used differently though. The setting is, as mentioned, that decision trees suffer for high variance and thus low generalizability. In general, averaging across observations reduces the variance and hence a natural way to reduce the variance and increase the prediction accuracy is to create several training sets, build a separate prediction model using each training set, and average the resulting predictions. Of course, this is not practical because we generally do not have access to multiple training sets. Instead, bootstrap by taking repeated samples from the (single) training data set. While bagging can improve predictions for many regression methods, it is particularly useful for decision trees. To apply bagging to regression trees, we simply construct *B* regression trees using *B* bootstrapped training sets and average the resulting predictions. These trees are grown deep and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these *B* trees reduces the variance. Bagging has been demonstrated to give impressive improvements in accuracy by combining hundreds or even thousands of trees into a single procedure. Thus far, the bagging procedure has been presented in a regression context, i.e. predicting a quantitative outcome. How can bagging be extended to a classification problem? There are a few possible approaches, but the simplest is as follows. For a given test observation, record the class predicted by each of the B trees and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

Estimating test error is next. There is a very straightforward way to estimate the error of a bagged model, without the need to perform cross-validation or the validation set approach. Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations. Thus, it is possible to predict the response for the i^{th} observation using each of the trees in which that observation was OOB. This yield around $B/3$ predictions for the i^{th} observation. In order to obtain a single prediction for the i^{th} observation, average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the observation. The accompanying OOB error is a valid estimate of the test error, since the response for each observation is predicted using only the trees that were not fit using that observation. Next, assessing the importance of variables becomes relevant. As presented, bagging results in improved accuracy compared to a single tree but unfortunately, it becomes difficult to interpret the resulting model as visually mapping and interpreting B decision trees is impossible. As such, it is no longer clear which variables are most important to the procedure, and this generally means that bagging improves prediction accuracy at the expense of interpretability. Although the collection of bagged trees is more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using the RSS (for regression trees) or the Gini index (for classification trees). In bagging regression trees, record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor. Similarly, in the context of bagging classification trees, add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees. These measures of importance can be graphically presented meaning that besides the bettering of prediction accuracy, due to averaging, bagging also produces a less biased overview of the features' importance.

Random forests provide improvements over bagging by way of a small tweak that decorrelates the individual trees. As in bagging, several decision trees are built on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of m

predictors is chosen as split candidates from the full set of *p* predictors. The split is allowed to use only one of those *m* predictors. A new sample of *m* predictors is taken at each split – as standard, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors, but this can be altered at one's discretion. This means, in building a random forest, at each split in the tree, the algorithm is not allowed to consider most of the available predictors. It may sound strange, but there is a clever rationale. Suppose that there is one very strong predictor in the data, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all the trees will use the strong predictor in the top split. Consequently, all the bagged trees will look quite similar to each other and the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated models does not lead to as large of a reduction in variance as averaging many uncorrelated. Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(p\text{-}m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. Think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and more reliable.

Boosting is the last approach I will present for improving the predictions resulting for decision trees. Like bagging, boosting is also a general approach that can be applied to different algorithms for regression or classification. Boosting works in similar ways to bagging, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling – instead each new tree is fit on a modified version of the original data. In a regression setting, boosting does not fit large decision trees to the data, which amounts to fitting the data hard and potentially overfitting. Instead the boosting approach learns slowly. Given the current model, it fits a decision tree to the residuals from the model. That is, it fits a tree using the current residuals as the response, rather than the outcome variable. It then adds this new decision tree into the fitted function in order to update the residuals. Each tree can be rather small, with just a few terminal nodes, determined by the tuning parameter *d*. By fitting small trees to the residuals, it slowly improves model estimation in areas where it does not perform well, that is, it iteratively learns a larger portion of the difficult-to-classify examples by paying more attention to frequently misclassified examples. As additional rounds of weak learners are added, they are trained on data with successively more difficult examples. The process continues until the desired overall error rate is reached or performance no longer improves. At that point, each classifier's vote is weighted according to its accuracy on the training data on which it was built. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to "attack" the residuals. In general, algorithms that learn slowly tend to perform well. Boosting has three tuning parameters: 1) The number of trees, *B*. Unlike bagging and random forests, boosting can overfit if *B* is too large, although this overfitting tends to occur slowly (if at all). Use cross-validation to select B. 2) The shrinkage parameter, λ (a small positive number). This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice depend on the problem. Very small λ can require using a very large value of *B* in order to achieve good performance. 3) The number, *d*, of splits in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted ensemble fits an additive model, since each term involves only a single variable. Generally, *d* is the interaction depth, and controls the interaction order of the boosted model, since *d* splits can involve at most *d* variables. In evaluating boosted trees, rather than giving each learner an equal vote, each learner's vote is given a weight based on its performance. Models that perform better have greater influence over the ensemble's final prediction.

1.1.2.2.3 FEATURE SELECTION AND REGULARIZATION ALGORITHMS

Feature selection and regularization algorithms constitute cures to the omnipresent curse of dimensionality which plagues a lot of analytical settings. In general, the cure consists of imposing structure on the learning process in the hope of balancing bias and variance better than normal linear models, based on ex least squares, can. Imposing structure is achievable in different ways and generally means that less, but (hopefully) well-chosen, parameters are estimated than otherwise. This is especially relevant in high-dimensional settings which I will elaborate on below. Afterwards, I present subset selection and shrinkage methods. Subset selection includes best subset and stepwise approached, while shrinkage methods consist of LASSO, Ridge and Elastic Net. I presume substantial familiarity with linear and logistic regression as these techniques pose as basis for the presented algorithms.

In general, and especially in high-dimensional settings, we might want to use another fitting procedure than least squares as alternative fitting procedures can yield better prediction accuracy and model interpretability. With regard to prediction accuracy, imagine that the true relationship between the response and the predictors is approximately linear, then least squares estimates will have low bias. If $n >> p$ – that is, if n is much larger than p – then least squares estimates tend to also have low variance, and hence will perform well out-of-sample. However, if n is not much larger than p, there is a lot of variability in the least squares fit, resulting in overfitting and consequently poor predictions. And if p > or >> n, then there is no longer a unique least squares coefficient estimate: the variance is infinite so the method cannot be used at all. By constraining or shrinking the estimated coefficients, the variance can shrink considerably at the cost of a negligible increase in bias which leads to substantial improvements in the accuracy og prediction. Furthermore, the model's interpretability is increased. It is often the case that some or many of the variables used in a multiple regression model are in fact not associated with the response. Including such irrelevant variables leads to unnecessary complexity in the resulting model. By removing them – that is, by setting the corresponding coefficient estimates to zero – we can obtain a model that is more easily interpreted. Ordinary least squares are extremely unlikely to yield any coefficient estimates that are exactly zero. The approaches presented can be used for automatically performing feature selection in lieu of a multiple regression model.

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

Subset selection involves identifying a subset of the p predictors that are related to the response and then fit a model using least squares on the reduced set of variables. To perform **best subset selection**, a separate least squares regression for each possible combination of the p predictors is fitted. This produces 2^p different models and choosing the best among these potentially many models is typically broken up into two stages. For $k = 1, 2, \ldots, p$, fir all models containing exactly k features and pick the best among these with best implying smallest residual sum of squares (i.e. largest R^2). Secondly, select the single best model from the p models using cross-validated prediction error or an information criterion (C_p, AIC, BIC, adjusted R^2). Best subset selection is intuitively easy to grasp, but computationally extreme (with $p >= 40$ even most modern computers will succumb). Best subset selection may also suffer from statistical problems when p is large. The larger the search space, the higher the chance of finding models that look good on the training data, even though they might not have any predictive power on future data. Thus, an enormous search space can lead to overfitting and high variance of the coefficient estimates. A marketable alternative is **stepwise selection** which explores a far more restricted set of models. Stepwise selection exists in both a forward and backward edition. Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all the predictors are in the model. At each step the variable that gives the greatest additional improvement to the fit is added to the model. Though forward stepwise tends to do well in practice, it is not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors. For instance, suppose that in a given data set with $p = 3$ predictors, the best possible one-variable model contains X_1, and the best possible two-variable model instead contains X_2 and X_3. Then forward stepwise selection will fail to select the best possible two-variable model, because X_1 is kept in the first iteration. Like forward stepwise selection, **backward stepwise selection** provides an efficient alternative to best subset selection. However, unlike forward stepwise selection, it begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.

Shrinkage methods involves fitting a model involving all p predictors. However, the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as regularization) has the effect of reducing variance as it taps into the uneven fluctuations inherently found in the trade-off between bias and variance. Depending on what type of shrinkage is performed, some of the coefficients may be estimated to be exactly zero. Hence, shrinkage methods can also perform variable selection. Beginning with **Ridge regression**, the starting point is the acknowledgement that least squares minimizes the sum of squared errors (RSS) from data points to the estimated linear association. Ridge (and LASSO for that matter) minimizes a slightly different quantity. It minimizes RSS + $\lambda\sum\beta^2$ instead, where λ is a tuning parameter that is determined separately through cross-validation, and $\sum\beta^2$ equals the sum of the p squared β-coefficients. Combined $\lambda\sum\beta^2$ is called the shrinkage penalty. Its impact becomes smaller when $\beta_1, \ldots, \beta_p$ are close to zero, and practically, it has this exact effect of shrinking the estimates of each β towards zero. λ serves to control the relative impact of RSS vis-à-vis the shrinkage penalty – if $\lambda = 0$, a normal linear regression model is estimated, but as λ grows, the penalty for higher β-coefficients becomes larger and, as such, the β-coefficients approach zero. **LASSO** (short for Least Absolute Shrinkage and Selection Operator) is a relatively new addition to the family of shrinkage methods. It has quickly gained traction and works in a manner much similar to Ridge regression. It minimizes another quantity, though, which entails that LASSO is able to completely nullify some (in general the most superfluous) features. LASSO minimizes RSS + $\lambda\sum|\beta|$, that is the sum of absolute values of the p β-coefficients. In the literature, this difference between Ridge and LASSO is referred to minimizing the L2 and L1 norm, respectively. The main difference in practical terms is, as said, that LASSO is able to nullify some β-coefficients completely which prompt the corresponding feature(s) to "drop out" of the estimation (if $\beta_j = 0$, then $\beta_j X_j = 0$). The intuitive reason that the algorithm succeeds in separating significant from insignificant features, is that the significant features initial β-coefficients are larger and therefore the regularization process (reducing towards zero) will reach its optimum before they reach zero (only the insignificant features' coefficients achieve this unwanted feat). The sparsity a LASSO-models yields, when choosing only a subset of features, is one way to impose structure to the process of model estimation and, as such, its stability and interpretability will generally rise. With regard to predictive performance solely, there is no a priori reason to prefer LASSO over Ridge. Instead applying both in a cross-validation setting and comparing their outputs will usually be a suitable approach. **Elastic Net** is a third shrinkage algorithm and possibly an alternative to trying out both Ridge and LASSO. Quite literally, it comprises the best of these worlds with worlds (worlds being the different approaches). The best of both worlds means that the algorithm is able to decide for itself whether each feature (or a cluster of features) are to be treated by Ridge or LASSO. This intuitively copes with the no-nullification issue in Ridge and the hard-nullification issue in LASSO.

1.1.2.2.4 SUPPORT VECTOR MACHINES

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

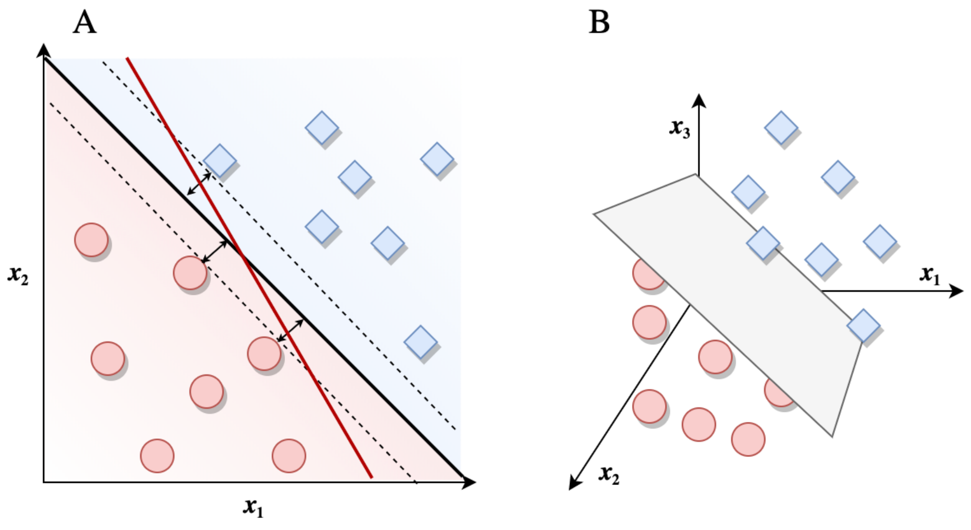
Support Vector Machines (SVM) (as well as Artificial Neural Networks (ANN) which I will not cover) is one of the most powerful families of algorithms within machine learning. They are sometimes referred to as "black box" methods, which is a general concept used to metaphorically describe algorithms where it, intuitively, is not possible to digest how exactly the estimation process functions – only inputs and outputs are seen, not the internal workings (many rebut this "black box" analogy – "everything is just math"). SVM (and ANN) algorithms oftentimes outperform their alternatives. Therefore, when choosing methods, there is a tradeoff between interpretability and explainable versus potential heightened performance. The context of application should inform this choice.

Support Vector Machines is used for classification (Support Vector Regression is the regression alternative). SVM can be used in both binary (the outcome has two categories) and multiclass (the outcome has more than two categories) settings – in relation to the latter, SVM is proven to be among the best performs due to their flexibility. The algorithm represents a generalization of a simple, intuitive classification method called maximum margin classifier, which revolves around the placing of hyperplanes that linearly separate groups of observations with different classes (values) on the outcome variable. The point of departure for SVM is that the observations in the training dataset are modeled in a *p*-dimensional space on the basis of a linear combination of the features, which optimally takes place so that the observations are placed in groups with similar output values, while the distance to the other groups is maximized.

In this modelling process, SVM places hyperplanes, defined as a flat *p*-1-dimensional subset of the *p*-dimensional space, dividing the space into two parts. The general rule when placing hyperplanes is to maximize the perpendicular margin to the closest observations from each class. A larger margin means that the confidence in the prediction increases. The optimal hyperplane functions as the decision rule: a new observation's location relative to the hyperplane determines which outcome class it is expected to belong to. After learning the decision rule in the training dataset, the algorithm uses the decision rule to predict the test dataset outcome based on these observations' linear combination of features.

Using SVM for multi-class classification is based on a "one-vs-one" approach (predict A if on one side of the hyperplane, B if on the other) similar to the one described above. The approach works by modeling one classification model per value-pair on the outcome variable. Thus, in a context of three possible outcome classes – A, B and C – three models are modeled: A vs B, A vs C, B vs C. In all models, the optimal hyperplane is learned, and each observation is predicted to belong to whichever of the two classes that are most likely as a result of the observation's features. Finally, all "one-vs-one" models vote on which class the individual observation is predicted to belong to, and it is attributed to the class with the most votes. If observation *i* is attributed to class A in model A vs B, A in A vs C and B in B vs C, it is predicted to belong to A. The following describes how the algorithm works, and then the tuning parameters are elaborated. The starting point is a low-dimensional context, after which higher dimensions and non-linear separable classes are commented on.

FIGURE 1.9.: MAXIMAL MARGIN CLASSIFIER



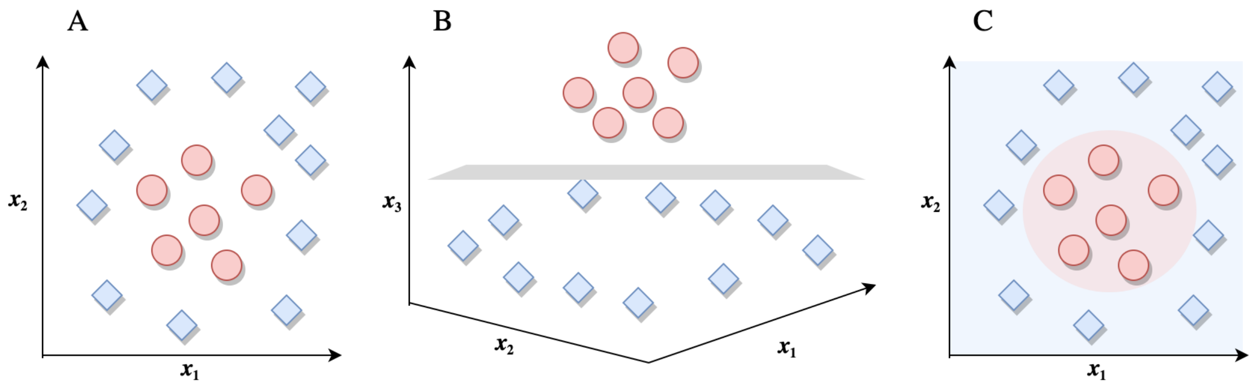
Note: ABG creation, inspired by James et al. (2015)

In a two-dimensional space consisting of variables X_1 and X_2 (figure 1.9, panel A), the black fully-drawn line represents the optimal hyperplane, which linearly separates the two outcome classes – blue squares and red circles – from each other optimally. The red solid line is depicted as an example of a non-optimal hyperplane that also separates the classes from one another but does not maximize the margin to the closest observations. Thus, it is through training that the algorithm learns that the black line is better as hyperplane than the red. In three-dimensional space (panel B), the hyperplane is represented as a two-dimensional surface– and, according to the definition, this evolves into an uninterpretable geometric size of p-1 dimensions in a p-dimensional space. However, this emphasizes that hyperplanes can also be used to inform decision rules in complex modeling of high-dimensional input data.

The location of the optimal hyperplane in panel A highlights the three observations on the dashed lines. These are ‘support vectors’ and capture the maximum margin for each of the group’s closest observations. They are called ‘vectors’ as they constitute a linear combination of p, i.e. a vector, in this p-dimensional space – and ‘support’ because they support the optimal hyperplane: if the three vectors move, the optimal hyperplane moves. This responsiveness makes the model susceptible to observations that do not follow usual distributions across the feature space for a given outcome class. For example, if a blue square is placed around origin in panel A, this would make it impossible to completely separate the groups linearly, while the introduction of a blue square at the bottom of the dotted line with the two red ‘support vectors’ would result in the maximum margin – and thus the decision rule’s certainty – decreasing as the optimal hyperplane (which would then resemble the red line) is forced to encapsulate this square. Such situations weaken the model’s predictability, as there is a greater risk of misclassifying test observations. However, it is emphasized that SVM is relatively flexible which especially becomes visible in situations where the algorithm is challenged. The flexibility is heightened through optimization.

SVM optimization is done by transforming data through kernel functions and adjusting tuning parameters. Overall, the optimization relates to the trade-off between bias and variance with the aim of balancing complete modeling of the training dataset against the ability to predict the test dataset and generalize more accurately. In the following, the intuition around kernel functions and tuning parameters are presented focusing on how they enable classification optimization. The presentation of the tuning parameters is based on James et al. 2015 (Ch. 9), Lantz 2013 (Ch. 7), and Meyer 2019, which can be referenced for a review.

FIGURE 1.10.: KERNEL TRANSFORMATION



Note: inspired by James et al. (2015)

Kernel functions are used when it is not immediately possible to linearly separate the groups in a *p*-dimensional space. Kernels cover ways to transform the input data into a high-dimensional space where it becomes possible to separate different clusters (see James et al. 2013: 350-352 for elaboration on kernels). There exist many different types of kernels – you can even build your own. The intuition of transformation is shown visually in Figure 1.10. In the two-dimensional panel A, it is not possible to separate the two classes linearly, which is possible in panel B, where the feature space is transformed into a three-dimensional space. When the data is subsequently transformed back to the two-dimensional starting point (panel C), the optimal hyperplane functions as a normal decision rule, even if it does not appear visually linear. The purpose of utilizing kernels is to meet the challenges of bias and variance, which is because the maximum margin for the closest observations from each class increases if the data is transformed. The reason for the reduction in bias is that the algorithm becomes better at separating the outcome clusters from each other in the training dataset - sometimes, as in panel A above, it will be practically impossible to separate the classes without transformation. The transformation also reduces the variance to some extent as the decision rule becomes more stable as the margin increases. Therefore, the algorithm is expected to generalize better.

In addition to kernel functions, SVM can be optimized through two tuning parameters. The first parameter, λ (lambda), describes how much misclassification is allowed, while the second, γ (gamma), specifies the impact of training observations on the specification of the decision rule. In relation to λ, the main idea is that the algorithm can be allowed to must misclassify to a certain extent. This is consistent with the intuition behind SVM, which focuses on maximizing the distance from the optimal hyperplane to the closest 'support vectors' from each class. Thus, it is not always possible (or optimal) to avoid misclassification, as misclassifying a few observations in the training dataset may result in the decision rule – the optimal hyperplane – becoming more stable and thus better suited for generalization. In other words, bias deliberately increases slightly but it is countered by greater decrease in variance. Specifically, λ covers the size of a budget by which the algorithm can buy a given number of misclassifications unless it pays to change the optimal hyperplane to capture them. The longer a misclassified observation is from the correct side of the hyperplane, the more it costs. The total cost associated with the misclassifications is the sum of the error links, which can sum to a maximum of λ. The lower the λ, the harder the algorithm is forced to work to avoid misclassification. The second tuning parameter, γ, deals with the impact of training observations on where the decision rule is placed. The influence is expressed in the size of γ, where higher levels cause the decision rule to be influenced only by the observations in their immediate vicinity, while low values means that the training observations far from the decision rule also influence how precisely it is placed. Panel A in Figure 1.9. is an example of a relatively high γ, since only the three closest observations have influence as 'support vectors' while a lower γ mean that all the observations in the figure influence the location. γ is generally used to regulate the trade-off between bias and variance. At a low γ, where most of the observations influence the decision rule, there is a greater risk of overfitting as the rule is made dependent on more information from the training data. Conversely, a lower γ can cause significant, but complex, relationships between the features to be captured, which may cause the algorithm to generalize more precisely. Weighting the observations' influence therefore influences how the optimal hyperplane and thus the decision rule is finally placed, which of course has implications for the following classification. It is not possible to know in advance which values for the tuning parameters or which kernel that ensures the best model – they are found using a 'grid search', typically built into a *k*-fold cross-validation setting.

Not mentioned: Naïve Bayes, Partial Least Squares, Splines, Cubist, Discriminant Analysis, other ensembles – mostly nonlinear models. Some of them should be incorporated, I'd say.

1.1.3 OTHER USES: IMBALANCED OR MISSING DATA

- Dealing with typical issues, we encounter:
    - Imbalanced data sets
        - Cost-sensitive learning, sampling procedures (under, over, SMOTE)

- Missing data
    - Sensitivity analysis:
    - MI or MICE:
    - Joint modelling: JM involves specifying a multivariate distribution for the missing data and drawing imputation from their conditional distributions by Markov chain Monte Carlo (MCMC) techniques. This methodology is attractive if the multivariate distribution is a reasonable description of the data.

In the next chapters: using machine learning at different steps in the RFI-process.

2.0 MACHINE LEARNING IN SOCIAL INNOVATION

Social innovation constitutes the process of developing practical interventions which, from a data-driven point of view, consists of two subprocesses: (i) measuring an outcome (i.e. establishing a target group), and (ii) estimating what heightens/weakens the chance of becoming a part of the target group (i.e. the risk/protective factors). Machine learning techniques are useable in both subprocesses. In the following, I present some of the latest trends in developing interventions assisted by machines. The trends encapsulate useful consideration which mostly have been discussed in relation to developing countries. They are primarily found in blog post etc. which are referenced throughout. Afterwards, I reflect on how the presented algorithms might enable social innovation.

2.1 TRENDS IN DEVELOPING INTERVENTIONS

Machine learning (ML) has been around for decades but is increasingly being recognized as a tool to development for three reasons. First, ML has advanced in recent years; better algorithms and open-source software have made ML tools widely available and accessible. Second, the infrastructure to manage, share, and analyze data (including high-speed computational power) has become affordable at scale. And third, there's been an explosion in the amount of available development data, thanks to both deliberate data collection efforts like surveys, program monitoring, and evaluation studies, and new data sources from e.g. satellites and mobile phones.

In "Stanford Social Innovation Review: Can Machine Learning Double Your Social Impact" a list of four practical requirements for using machine learning is presented which underlines some of the talking points presented earlier. They firstly highlight the need for good predictors. It is important to have predictor data for every person or place you want to make a prediction about. Some data sources systematically undercount vulnerable groups and excluding them from the prediction can lead to the denial of much-needed programs and services. Secondly, high-quality outcomes are necessary. This is multidimensional and includes the need to (i) include enough of the right data to uncover true patterns, (ii) match the granularity of decision-making (if you are interested in system level change, have outcomes at system level), (iii) link predictor and outcome data, and (iv) represent the target population as closely as possible. Matching the granularity of decision-making is relevant to emphasize, as it reminds us to consider whether ex the target group (and response variable) is sufficiently context bound. Related, governments and nonprofits are often accustomed to one-size-fits-all programs and shifting to targeted approaches means it must be legally and politically feasible to prioritize action for some people or communities over others, based on need or estimated risk. Thirdly, the capacity to act on predictors is highlighted, because otherwise actual social impact is impossible. Lastly, there need to be an ability to maintain and employ the machine. Contrary to popular belief, most machine-learning algorithms do not get smarter over time without human help. Instead, they require a human to be able to obtain and load training data, re-run the algorithm, adjust algorithm parameters based on the results, as well as deploy new models.

It is argued that machine learning is particularly ripe for use in addressing two kinds of problems in intervention-development processes. The first is prioritization problems. If an organization focused on conflict resolution can predict where violent conflict is likely to breakout, for example, it can double-down on peacebuilding interventions. If a health NGO can predict where disease is most likely to spread, it can prioritize distribution of public health aid. The second is data-void problems. The data used to target intervention is rarely granular, recent, or accurate enough to pinpoint the specific regions or communities that would benefit most and collecting more-comprehensive data is often expensive. As a result, many of the people who need a program the most might not receive it, and vice versa. If, however, an NGO fighting hunger in a rural state in Ethiopia knows which villages that have the highest malnutrition rates, it can focus its outreach efforts in those communities, instead of oversaturating a different region that has fewer needs. As a concrete example, consider Educate Girls, a nonprofit in India tackling gender and learning gaps in primary education. IDinsight, a global advisory, data analytics, and research organization, previously ran a randomized evaluation of Educate Girl's program as a part of a development impact bond, and found that the program has large, positive effects on both school enrollment and learning outcomes. Yet despite very ambitious plans to reach millions more children over the next five years, Educate Girls could not immediately expand its program to every one of India's 650,000 villages. The question became how to prioritize villages to reach as many out-of-school girls as possible. Educate Girls' records showed that more than half of all out-of-school girls in its current program areas were concentrated in just 10 percent of villages. But there were no up-to-date, comprehensive, reliable data sources that indicated where the most out-of-school girls were concentrated in other areas and knocking on tens of millions of doors would be prohibitively costly. Hence, Educate Girls had a data-void problem that machine learning could help it overcome by way of predicting differences in probability of being out-of-school by geographic area on the basis of surveys. They were able to focus the intervention very specifically, i.e. to separate villages within the same administrative district.

In another blog post, "How can machine learning and artificial intelligence be used in development interventions and impact evaluations?" published by the World Bank, nuancing points are made especially related to measuring outcomes and help targeting interventions. As such, machine learning comes in handy at a micro level, enabling measurement of outcomes we might otherwise struggle to measure. One example uses textual analysis of transcripts of India's village assemblies to identify what topics are discussed, and how the flow of conversation varies with gender and status of the speaker. The vast volume of data would make this very hard to do systematically using traditional measurement methods. They used this to find, for example, that female citizens are less likely to speak, less likely to drive the topic of conversation, and get fewer responses from state officials – but when the village has been randomly chosen to have a female president, female citizens are more likely to receive a response than with a male president. Further, machine learning may help target interventions. This can include both when to intervene as well as where/for whom. Poverty mapping is one example. Other examples (see blog post for references to each) include using remote sensing to detect where deforestation might be starting to take place, to quickly intervene; using machine learning on VAT tax data in India to better target firms for audits; predicting travel demand patterns after hurricane; or during big events such as the Olympics to help figure out where transport interventions are needed; predicting where food insecurity will occur to help target aid interventions; using mobile call records to identify a pool of small businesses that credit can be extended to; and figuring out where there are lots of girls out of school in order for an NGO to figure out which region of India to next expand its program to. These examples are in the proof of concept stage right now, showing that such methods could, in principle, be used for targeting interventions, but few of them are being used currently to target programs.

Besides focusing on precisely targeting interventions, it is highlighted that one should beware of the hype regarding ML. Some think ML is a silver bullet to solve all development problems, but it is not the answer to every question. Organizations should recognize it as a valuable tool, one of many complementary approaches to solving a problem. The key is to start with the problem – then find the right tool. While many researchers are appropriately cautious, there is also a high ratio of pretty pictures demonstrating social impact. We need to be better about also making clear when these methods do not offer improvements (or when they do worse) than current methods or status quo. It is also emphasized that dynamic relationships might be understudied (or underappreciated): a first concern is whether many of the predicted relationships are in fact stable. That is, if you conduct an expensive training set survey to predict the relationship between satellite images and crop yields today, will the same relationship still hold in a year's, or 5 years', time? A second concern is that of behavioral responses, e.g. if people learn their phone calling behavior is being used to determine eligibility for interventions, they may change their behavior.

There is an area called "adversarial machine learning" which is a frontier topic concerned with designing methods more robust to this. Lastly, ethics, privacy and fairness are highlighted. Is it fair to be denied a program because the people you talk to on your cellphone have variable calling patterns? What rights do people have to privacy in an environment where satellites are photographing their house every day, phones are tracking their every move, and their moods are being analyzed on social media, etc.?

From another Stanford Social Innovation Review blog post, Demystifying Machine Learning for Global Development, other highlights are presented. First, they point to the fact that people are not the same. To achieve impact and a shift in behavior, organizations need to move away from a one-size-fits-all approach and toward interventions targeted to different sub-groups of the population. In that regard, cluster analysis is a powerful technique. To recap, the basic principle is to group people in such a way that group members are as similar as possible to each other and as different as possible from those in the other groups. Development programs may base their clustering efforts on demographic factors such as age, gender, education, or urban-rural residence. But the beliefs, motivations, biases, and norms that underlie people's decisions, as well as potential structural barriers (such as the availability of a service), are equally important to targeting efforts. The clustering process could make it easier to afterwards map how the intuitively unobservable characteristics can become slightly less unobservable by examining the relevant individuals from cluster that group "interesting" observables.

Secondly, being able to predict what will happen, where, and when can help organizations focus their limited resources on the right interventions, people, place, and time. Predictive machine learning helps achieve exactly this. Classification methods can, for example, identify poor households in a community using minimal amounts of national survey data, whereas using traditional analytical methods would be expensive and time-consuming. Regression models have a variety of applications too. They can predict levels of wealth based on satellite data identifying homes and property, for example, or the amount of corruption based on administrative indicators such as government data on financial transactions.

Thirdly, in development, we are obsessed with causation. Our default approach to this has been using randomized control trials, but they can be costly, take years, and are often only able to test the effect of one intervention at a time. Causal machine learning lets us identify the network of factors that influence a development outcome. This moves beyond making predictions and helps us understand the underlying causal relationship between variables. For example, to truly understand infant mortality in a certain setting we need to know more than which factors might correlate with the death of a baby. Instead, the question

ML can help answer is precisely which characteristics, behaviors, and health parameters of mothers, and which behaviors by health care providers, lead directly or indirectly to infant deaths? ML algorithms can help us map how all these factors interact with each other, and once we define this underlying, explanatory structure, we can do “what if?” experiments with the data. For example, what if we had front-line workers contact pregnant woman five times during their pregnancy instead of three times? How much change in infant mortality would we expect to see? The Surgo Foundation are currently testing this approach in India. They are using numerous vertical datasets to try to understand the complete set of factors that explain the outcomes we see in that location. They collected variables both from the mothers, such as how many antenatal checkups she attended, and from the health facilities, such as nurse practices. Putting these variables into a regression model might reveal that a clean umbilical cord or an educated mother is associated with infant survival, but all we would know is that the two are correlated. It wouldn’t necessarily be clear whether these were causal factors, or whether other factors played a role. By contrast, causal machine learning helps illuminate how all the factors are linked in a network, see which ones impact the outcome, and identify critical touch points in that system to improve survival rates. Meanwhile, there are issues to consider before deciding whether to use ML such as quality and quantity of data. While amount and types of data generally are expanding rapidly, we tend only to collect what we think is important, and we may therefore miss significant predictors or causal factors of behavior.

2.2 USING ALGORITHMS IN DEVELOPING INTERVENTIONS

- Unsupervised data mining
- Supervised leaning
    - Tree-based, LASSO (feature selection in general), Ensembles

3.0 MACHINE LEARNING AS INTERVENTIONS

Intro and structure:

- Latest trends
- Using ML as/in interventions
    - Duflo and Matching?
    - RiskSLIM (…)
    - …

3.1 TRENDS IN MACHINE LEARNING AS INTERVENTIONS

See above and below.

- https://blogs.worldbank.org/impactevaluations/how-can-machine-learning-and-artificial-intelligence-be-used-development-interventions-and-impact

There currently seem to be fewer cases where artificial intelligence and machine learning are being used for the interventions themselves, but the promise lies in using them for individualized and dynamic treatments. Jake Kendall outlined a vision for this, noting that his organization have been giving small grants to develop artificial intelligence chatbots that act as digital guides and advocates to help the poor navigate through bureaucracies. Examples included chatbots that could provide immigration help in the Dominican Republic and help navigate people in the Philippines through a social welfare program. Another example comes from agriculture, where Ofir Reich explained how they were trying to provide customized agricultural advice to farmers through mobile phones, with rapid testing and feedback being used to provide actionable customized information that farmers could use.

- Something about depression: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6473205/
- Ideas42: https://www.ideas42.org/machine-learning/
- IDinsight: https://www.idinsight.org/projects + https://www.idinsight.org/publications
- Nature (infection forecast): https://www.nature.com/articles/d41586-018-02473-5
- Bias and pain relief: existing instruments can be biased, e.g. "Where does the checklist by which he scores/grades the x-ray come from? In this case, from two researchers who studied coal miners in Lancashire, England in the 1950s. Those original papers that underlie the objective grading

systems used to diagnose knee arthritis today did not mention the race of the subjects – perhaps because there was no heterogeneity along this dimension in the study population. Be that as it may, that's what doctors use today: a grading scheme based on a 1950s, likely mostly white, English population. Could the doctors be under-diagnosing knee problems among minorities and disadvantaged segments of the population? Sure…" → use ML instead, e.g. "Well, with the algorithm trained on patient reports of pain to use x-rays to predict pain (and, by implication, severity of disease), what happens? First, that pesky dummy variable for race showing a large gap, which would not budge when adjusting for grades assigned by humans, declines by approximately half. What's really interesting is that if you limit the x-rays to white knees only during training, the performance of the algorithm gets worse: so, a lot of the action is in the diversity/representativeness of the sample that the algorithm (or, even the human) is trained on. Furthermore, the algorithm seems to be finding real signals, not re-learning and re-adjusting what radiologists do.".

3.2 USING ALGORITHMS AS INTERVENTIONS

- RiskSLIM
    - https://www.researchgate.net/publication/318918444_Optimized_Risk_Scores
    - https://users.cs.duke.edu/~cynthia/docs/WagnerPrizeJournal.pdf (HT)

- Duflo and Matching:

4.0 MACHINE LEARNING IN EVALUATION

Intro and structure:

- Introduction
    - Introduce idea behind chapter name.
    - Semi-detailed walk-through of selected algorithms relevant in RFI with an applied focus
    - Randomized controlled trials especially
- State-of-the-art and breakthroughs
    - Machine learning for ATE
    - Post double selection LASSO
    - Causal and heterogenous effects

4.1 TRENDS IN EVALUATION OF INTERVENTIONS

See above.

- RCT's as unethical: https://towardsdatascience.com/in-the-age-of-machine-learning-randomized-controlled-trials-are-unethical-74acc05724af
- Machine learning to measure treatment heterogeneity. Susan Athey gave an excellent keynote talk that rapidly overviewed how machine learning can be used in economics, and her AEA lectures have more. She noted two different approaches in using machine learning to identify heterogeneity in treatment effects. The first builds on the way we typically do heterogeneity analysis, where we examine heterogeneity by some X variable. The idea here is to use machine learning to figure out what the right groups are for doing so - using causal trees, targeted machine learning, X-learners, or other methods – and then once people are assigned to groups, you can get standard errors on that heterogeneity and it is similar to our standard case. One caveat she noted is in interpreting the groups – e.g. just because the causal tree splits on education and not gender, it does not mean that gender is not important for heterogeneity (the two could be correlated for a start). A second approach is to take a non-parametric approach and try to get an expected treatment effect for each individual unit. This is what causal forests do. This is a rapidly advancing area, with relatively few practical applications to point to so far. Robert On gave one example – they worked with the One Acre Fund in Rwanda to digitally market lime fertilizer to a massive

sample of farmers, and then use this large sample to employ both causal tree and causal forest approaches to examine heterogeneity in treatment impacts.

- Taking care of the Confounders (D'X). In her talk, Susan noted that while machine learning won't solve your identification problem, it can at least help you become more systematic about model selection for the predictive part of your model. This is particularly important in non-experimental applications, and she gave references to machine learning tools for work with matching, instrumental variables, and RDD. Cyrus Samii provided one example, for work in Colombia where they wanted to examine different policies the government could use to reduce criminality among ex-combatants. Intuitively, selection on observables seems more plausible when you have lots of observables – but with 114 observables, standard OLS or propensity score matching approaches may not work well. His work used regularized propensity score methods and compared them to these other approaches – yielding estimates of the impacts of employment and socio-emotional support programs.

4.2 USING ALGORITHMS IN EVALUATION OF INTERVENTIONS

Structure for walk-through of specific methods:

- One-sentence homerun
- Introduction and focus
- Output and relevance in RFI

Machine learning in policy evaluation: new tools for causal inference

Estimating ATE of binary treatment

Double Machine Learning for Causal and Treatment Effects

…

Machine Learning Methods for Estimating Heterogeneous Causal Effects

…

Machine-Learning Tests for Effects on Multiple Outcomes

…

4.2.2: HETEROGENOUS TREATMENT EFFECTS IN RANDOMIZED EXPERIMENTS

USE THIS IF…

- B
- B
- B

REMEMBER TO CONSIDER…

- B
- B
- B

This presentation originates from a paper by Chernozhukov et al. (2017). The key feature of this approach is the ability to estimate and make inference on key features of heterogenous effects in randomized experiments, i.e. inferences on whether an experimental treatment affects the outcome differently across background characteristics (i.e. covariates, also referred to as predictors in the following).

Randomized experiments play an important role in the evaluation of social and economic programs. Researchers are often interested in features of the impact of the treatment that go beyond simple average treatment effects. In particular, they want to know whether treatment effect depends on covariates such as gender, age, etc. Empirically, it is essential to assess if the impact of the program would generalize to a different population with different characteristics, and for social innovators, there is also substantial value in better understanding the driving mechanisms behind the effects of a particular program. To assess heterogeneity in general, an issue arises regarding reporting treatment effects for subgroups: how might we sample split such that the consistency and validity of our models are heightened? Choosing splits *ex post* heightens the risk of overfitting and p-hacking while *ex ante* registration of testing e.g. specific interaction terms amounts to throwing away a large amount of potentially valuable information, especially with large baseline data sets where many potential predictors might have an effect on the designated outcome. Discovering any relevant heterogeneity in treatment effects by co-variates *ex post* in a disciplined fashion suggests using machine learning methods. These methods are especially sound when there are a lot of baseline predictors and little a priori guidance on what to look for, i.e. estimate.

The approach presented here is generic, meaning that it is able to include different algorithms (penalized methods, neural networks, support vector machines, random forests, boosted trees, ensembles), and agnostic, meaning that it is not building on unrealistic or hard-to-check assumptions – instead data-splitting and choosing medians of betas, p-values and confidence intervals are prioritized. This helps coping with a core difficulty related to machine learning (especially in high-dimensional settings): obtaining uniformly valid inference even when methods perform well in terms of predicting empirically. In fact, in high-dimensional settings, absent of strong assumptions, generic machine learning tools may not even produce consistent estimates of conditional average treatment effects. Such assumptions include structured form of linear and non-linear sparsity and super-smoothness, both of which are untestable – and furthermore, most computing packages do not allow for as fine-grained optimization of tuning parameters as the potentially theoretically-sound approaches ultimately necessitate.

Key features of the approach are its validity in high dimensional settings and that effects are proxied by machine learning methods. It estimates: (i) the best linear predictors (BLP) of the conditional average treatment effect (CATE) such that any detectable heterogeneity in the treatment effect based on observables is highlighted; (ii) average effects sorted by impact groups (GATES) meaning, if there is any, the treatment effect for different bins; (iii) average covariate characteristics of most/least impacted groups of observations (CLAN), i.e. a description of which of the covariates that are correlated with this heterogeneity. I elaborate on the output from the approach in relation to BLP, GATES, and CLAN after briefly touching on how CATE is estimated: split the full data set (containing i.i.d. sampled observations) into two sets – a main (M) and an auxiliary (A) – of equal sizes. From A, machine learning estimators of baseline and treatment effects, the proxy predictors, are obtained. These predictors are possibly biased and noisy and in principle they are not even required to be consistent. They are simply treated as proxies and through post-processing, they are used to estimate and make inference on the features of the CATE. When working with M, remember to condition on A, meaning that these maps are frozen. The approach uses many splits of the data into M and A to produce robust estimates and thereby systematically account for two sources of uncertainty, i.e. estimation (conditional on A) and splitting (induced by random partitioning into M and A). For point estimates, take the median of the estimated key features over different random splits into M and A. For confidence intervals, take the medians of many random conditional confidence sets and adjust their nominal confidence level to reflect splitting uncertainty. P-values are constructed by taking medians of many random conditional p-values and adjust their nominal levels to reflect splitting uncertainty. For each split of that data, tuning parameters are chosen based on mean squared error estimates of repeated 2-fold-cross validation (except for random forests, for which algorithm defaults are used). In all included methods, both outcomes and covariates are rescaled to be ranging from 0 to 1, and in tuning and training the methods, it is only the auxiliary data set that are used.

The output from the algorithm (and specifically, the R-script) is as follows. Short description of output.

Checklist for the above:

- One sentence homerun

- Introduction and focus
- Output and relevance in RFI

Estimating each of the three – and their output:

- BLP (best linear predictor of CATE): …
- GATES (sorted group average treatment effects): …
- CLAN (classification analysis, i.e. average characteristics of the most and least affected units): …

5.0 REFERENCES

Gabadinho, A., G. Ritschard, N.S. Müller and M. Studer (2011). "Analyzing and Visualizing State Sequences in R with TraMineR." Journal of Statistical Software, 40(4), 1–37.

Gelman, Andrew. 2007. "Scaling regression inputs by dividing by two standard deviation". *Columbia University.*

Studer, Matthias. (2013). WeightedCluster Library Manual: A practical guide to creating typologies of trajectories in the social sciences with R. LIVES Working papers. 24.

- Shmueli – explaining and predicting
- Nate Silver – The Signal and the Noise
- O'Neal – Weapons of Math Destruction
- Varian – New Tricks
- Machine Learning in R (Lantz 2013)
- An Introduction to Statistical Learning (James et al. 2013)
- Elements of Statistical Learning (Hastie et al. 2008)