UPPSALA
UNIVERSITET

IT 17 091
Examensarbete 15 hp
December 2017

Predicting house prices using Ensemble Learning with Cluster Aggregations

Johan Oxenstierna

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Predicting house prices using Ensemble Learning with Cluster Aggregations

Johan Oxenstierna

The purpose of this investigation, as prescribed by Valueguard AB, was to evaluate the utility of Machine Learning (ML) models to estimate prices on samples of their housing dataset. Specifically, the aim was to minimize the Median Absolute Percent Error (MDAPE) of the predictions. Valueguard were particularly interested in models where the dataset is clustered by coordinates and/or attributes in various ways to see if this can improve results. Ensemble Learning models with cluster aggregations were built and compared against similar model counterparts which do not partition the data. The weak learners were either lazy kNN learners (k nearest neighbors), or eager ANN learners (artificial neural networks) and the test set objects were either classified to single weak learners or tuned to multiple weak learners. The best results were achieved by the cluster aggregation model where test objects were tuned to multiple weak learners and it also showed the most potential for improvement.

Handledare: Lars Erik Eriksson
Ämnesgranskare: Kristiaan Pelckmans
Examinator: Olle Gällmo
IT 17 091
Tryckt av: Reprocentralen ITC

Contents

1 Introduction 1
2 Related Research 5
2.1 Regression using Artificial Neural Networks (ANN's) 5
2.2 Regression using kNN 7
2.3 Bootstrap Aggregating and Boosting 10
2.4 Cluster Aggregations 12
3 Method 19
3.1 Preprocessing 19
3.2 Division into training/validation/test set 20
3.3 Model Building 21
3.3.1 Test objects tuned to a single weak learner 21
3.3.2 Test objects tuned to multiple weak learners 24
4 Results 28
5 Discussion 30
6 Future Work 34
7 References 35

1 Introduction

This investigation assessed Ensemble Learning models with Cluster Aggregations on subsamples of Valueguard AB's housing dataset. The aim was to minimize the median and mean errors of house final prices and evaluate the computational time required for this. The housing dataset contains dozens of categorical, binary and continuous features following various distributions and there are also several incomplete or missing features. The clustering part of the problem was to partition the dataset into as meaningful subsets as possible and predictions in each cluster or 'weak learner' were then made using one of two regression models: A lazy learning model in the form of kNN (k Nearest Neighbor); an eager learning model in the form of ANN (Artificial Neural Network), after which results were aggregated for final evaluation, hence 'cluster aggregations'.

The reasoning behind carrying out the data-partitions was to study whether computational time could be decreased while maintaining or even improving predictive accuracy. If an ensemble can be built where weak learners are not interdependent this can have clear benefits in terms of reducing computational time since each weak learner can be optimized on a separate processing unit (Cartmell, 2010). Concerning predictive accuracy this could also see improvements if the dataset contains significant heterogeneous sub-patterns (Trivedi, 2012; Ari & Guvenir, 2002; Judea, 2015; Abbott, 2001).

The two regression models are here briefly introduced. kNN finds the most similar neighbors to a certain object and makes a prediction based on the target values in this 'neighborhood' (Duda & Hart, 1973). This neighborhood can be searched for by transforming the test set to information such as where the neighbors of training set cluster centroids are located (Zhang & Sun, 2010, Hastie et. al, 1997 and Ougiaroglou1 et. al, 2007). It is a lazy algorithm because there is no model building phase where training data is looped through many times.

ANN's are modeled on the cerebral cortex of mammals and work by running inputs through layers of weights and then adjusting these to fit target values, in this case for the purpose of approximating a function (McCullohs; Pitts, 1943; Werbos, 1974 et.al.). ANN's usually require more time to train compared to kNN but less time to test since they use eager learning instead of lazy learning. Eager learning is a type of learning where a model is built based on a search for meaningful patterns in the training set.

Both kNN and ANN's are non-parametric and capable of predicting noisy and incomplete data but the ways in which they do this differ significantly. Where kNN based methods are often considered simpler, white box and lower level, ANN's are often considered more complicated operating on the higher abstraction level. It is often easier to tune kNN methods in terms of hyperparameters but harder to tune them in terms of feature selection and engineering, which is to a larger extent controlled within the autonomous training process of the ANN's.

An extension to the cluster aggregation kNN model was developed where each test object was transformed based on all cluster centroids assembled by the training data. This was inspired by model stacking and specific cluster evaluation techniques (Gionis, 2007; Wolpert, 1992; Ari & Guvenir, 2002; Trivedi, 2017). This was not attempted on ANN's since it is harder to evaluate ANN parameters learnt in the training phase and apply them in the testing phase (see 'delimitations' below).

The two main research questions in this investigation can be summarized as follows: What is the performance of the cluster aggregation models in terms of:

1. Minimizing predictive error?
2. Minimizing computational time?

The performance was evaluated against whole-dataset models which use kNN, ANN or linear regression. These whole dataset versions were constructed to be as similar as possible to the cluster aggregation models only lacking the clustering part. Concerning the research questions there were no exact directives from Valueguard nor any deduced concepts as to how important 'minimizing predictive error' can be considered against 'minimizing computational time'. It was part of the investigation to search for insights and discuss where the line between the two can reasonably be drawn.

Valueguard AB

Valueguard are mostly working with providing real estate statistics to the Stockholm stock exchange housing index (HOX) but they are also making predictions and providing these as a service to e.g. banks and real estate agents. They have assembled a database with most real estate in Sweden and several features including coordinates, final price (since 2009), size, year built, quality, number of rooms, square meters etc. (more than 30 features in total).

Abbreviations/terms:

ANN: Artificial Neural Network. Sometimes 'ANN's' is mentioned generally and this refers to one of the models used in this study, based on MATLAB's 'fitnet' or 'function fitting' neural network.

Cross validation: A set of regression validation techniques in which the model is 'trained' on one part or several parts of the data and then tested on another part or parts of the data.

Eager Learning: A model is built in the training phase which is input independent and in which post-training queries have no effect on the model itself. Also known as offline learning.

Ensemble Learning: The method in which data is split up into heterogeneous subparts that are optimized separately and then recombined.

Features: Here treated as synonymous to ‘attributes’ or ‘factors’. For a housing dataset these are e.g. distance to water, taxation value or indoor area.

Feature Engineering: Finding good combinations of existing features or creating new ones out of the existing ones.

Hyperparameters: Parameters that are not related to the dataset used but rather to the functionality of a model. In this investigation the hyperparameters for the ANN model are: Learning rate (when applicable), training function, stopping condition, layers and neurons, features used and feauture weights. For the CkNN model: Distance metric, k, epsilon, features used and feature weights.

kNN: K-Nearest Neighbor: A method to search for the objects which resemble an object to the largest extent. k stands for the number of objects in the neighborhood.

Lazy learning: An algorithm that does not require a ‘training’ phase but where the model is built ‘online’ when a query from a test object is made for classification/prediction.

MAPE/MDAPE: Mean Absolute Percent Error/Median Absolute Percent Error.

ML: Machine Learning.

Model iteration: One attempt at getting predictions from one or both of the models.

NN: Nearest neighborhood.

Noise: ‘Noise’ is here used as a general term to denote random and systematic errors in datasets due to spurious readings, measurement errors and background data (Philips, 2017).

Overfitting: When a model is too complex for the quality of a dataset it can ‘overfit’, i.e. finding patterns in it which are not generalizable.

PCA: Principle Component Analysis.

Performance: When just the word ‘performance’ is used this refers to how good a model is at obtaining good predictions or good predictions as a function of computational time or data sampling.

PU: Processing Unit. In this study CPU’s were used.

Weak/Strong learner: In Ensemble Learning the data is split into subgroups, each of which is often called a ‘weak learner’, on which a unique model is applied and the results from these are then agglomerated into a ‘strong learner’.

Resources used

Uppsala university Scientific Linux servers ‘Vitsippa’, ‘Gullviva’ and ‘Tussilago’, with the following specifications: AMD Opteron (Bulldozer) 6282SE or 6274, 2.2 – 2.6 GHz, 16 cores, dual socket.

Matlab 9.0 (R2016a) with Neural Network and Statistical toolboxes.

Microsoft Office 2016.

QGIS(Las Palmas).

RapidMiner 7.6.

Delimitations

The main delimitations can be categorized into the following four areas: Model, centroid transformation, geography, PCA.

There are several ML models that could have been applied and which are potentially better than ANN’s or kNN at generating good predictions (e.g. Support Vector Machines (SVM), Random Forest, Bayesian, AdaBoost or XGBoost). In section 0 there is a discussion about why cluster aggregations could perform better than conventional ensemble learning methods but a comparison in performance to these methods was never made. It is also possible that several manually designed conventional approaches which rely on domain knowledge and feature engineering: Certain features, such as the taxation value, could possibly provide strong predictions with just a few other features given certain expert adjustments. Expert knowledge of dealing with outliers is probably also necessary to reach stronger predictions. Another delimitation was that cluster centroid transformation was applied and tested much more on the kNN model, something which could have been done for the ANN model as well but which was too time expensive for the Rapidminer implementation. The ANN’s were delimited in MATLAB as well on a general level since the architecture size was limited to three hidden layers and around 60 neurons in each level, and larger architectures were only to be selected if trends were found that were pointing toward that direction. It could, however, be the case that much larger sizes were needed to deliver meaningful results in the first place. Another delimitation was that the geographic location of the dataset used was delimited to Stockholm and Uppsala municipalities, as against a much larger geographical region. A last delimitation was that PCA was not carried out on the dataset and a comparison PCA/no PCA hence was not made. The reasoning behind this was that the features on their own were found to be too weak to be correlated to other individual features. Mahalanobis distance, which eliminates correlation between features was however applied in various tests on both models.

2 Related Research

2.1 Regression using Artificial Neural Networks (ANN's)

ANN's have been developed to simulate neurological function in the cerebral cortex of mammals. They consist of neurons and synapses which are capable of holding information in synapse weights and they can then simulate behavior by adjusting these weights to better fit certain input and target data. The weights are adjusted or 'learnt' by running through the data many times in so called 'epochs' and evaluating results from each epoch (McCullohs; Pitts, 1943; Werbos, 1974; Pasero, 2010).

A single neuron can process inputs by applying an activation function to a linear set of inputs:

Equation 1:

$$y_i = \varphi_i \left(\sum_{j=1}^{N} w_{ij} x_j + b_i \right)$$

where x represents the inputs and w represents the weights which connect each input j to a certain neuron i. φ is the activation function which serves to allow the ANN to solve non-linear problems, y is the output. b is the bias which serves to make the ANN more flexible and usually improves performance, especially if the data is distributed in a non-convex form. There are many types of ANN architectures which balance computational time, memory use and readability/maintainability. One of the most common architectures is the 'feedforward' ANN (also known as Multi Layer Perceptron), where neurons are organized into several layers and it is the architecture used in this investigation. Below is a visual representation of the neuron (left) and the feedforward net (right).

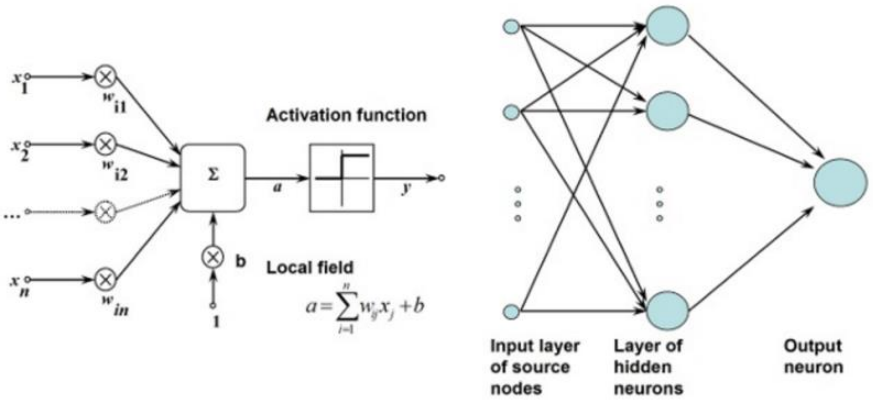

Figure 1: A picture of a feedforward ANN.

The weights of the neurons and bias neurons can be adjusted into an optimal configuration to reduce the difference between the output and the target output. There are several training functions to search for this optimal configuration. Gradient descent backpropagation, in which the weights are adjusted based on the derivatives of the outputs in the 'next' layer (Jacobian matrix optimization) is an effective mechanism that can train the synapse weights of an ANN to reach optimal results (Werbos, 1974). Today there are dozens of developed modifications and alternatives that generally work faster and are better at avoiding local minima i.e. the failure to reach the best global solution, at the cost of increased memory usage. Some of the most used optimization algorithms for this purpose are Levenberg-Marquardt, BFGS Quasi-Newton, resilient backpropagation (RPROP), Polak-Ribiere Conjugate Gradient (e.g. described by Hudson, Hagan & Demuth, 2017). Either these methods are intended to optimize the search for first order derivatives in the Jacobian matrix (e.g. RPROP), or optimizing second order derivatives in the Hessian matrix in some combination with first order derivatives (e.g. Levenberg-Marquardt) (Quezada, 2017). Since it is difficult to cultivate a-priori assumptions as to which training function works best in combination with computational time, one approach is to search for it using cross validation (Singh, 2013). Other hyperparameters such as number of layers and neurons, learning rates (for certain training functions), cost function and stopping conditions can also be searched for with cross validation (Bergstra, 2012). Stopping conditions refer to when the model should stop 'training' and usually this is taken as the best results obtained without causing overfitting. In order to avoid overfitting the dataset is commonly divided into three parts, a training, validation and test set, where the model can run in parallel on the three sets and stop training once the performance on the validation set stops improving (Hudson, Hagan & Demuth, 2017).

Using ANN's on housing data have been found to perform better than hedonic regression techniques in studies in Singapore and New Zealand (Lim, 2016; Visit, 2004). In the New

Zealand study, ANN's are believed to have performed better due to the "non-linear relationship between house features and house price, the lack of some environmental/geographical features, and inadequate number of sample size" (Visit, 2004). If we assume that a dataset is not linearly separable the use of ANN's seems warranted due to their high suppleness (Elizondo, 2006, Stergiou, 1996).

2.2 Regression using kNN

One way to predict the price of an object is to search for its most similar objects in the dataset and then take an average or weighted-with-distance average of the prices of this 'nearest neighborhood' (NN). The simplest form of this is to set the size of the neighborhood k and then make each object find its closest k matches in the dataset, whose target values are then used to make predictions, with or without weighting with the distance (Duda & Hart, 1973).

If the data is multidimensional with variously important and scaled features, such as is the case for housing data, it is necessary to normalize and then scale the dataset according to a set of weights. If we have a number of features F the distance *d* between objects x_i and x_j could be represented as:

$$d(x_i, x_j) = \sum_{k=1}^{F} \alpha_k d_k(x_i, x_j)$$

Equation 2: The distance between objects in kNN.

where k represents the feature in question and the weights α_k to be learned. d is the distance between objects. The above formula can be used to find the optimal configuration of α and *d* by studying kNN neighborhoods and transforming the data in such a way that the distances between target values within the neighborhood are minimized against values appearing outside the neighborhood.

Formally, a dataset $\{(x_i, y_i)\}_{i=1}^{n}$ consisting of n samples and the corresponding output y, are $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$. Then the above intuition is implemented by the solution to:

$$\max_{\delta,\alpha} \delta$$
$$\text{s.t.} \quad \begin{cases} \delta \leq \left(\sum_{k=1}^{F} \alpha_k d_k(\mathbf{x}_i, \mathbf{x}_j)\right) & \forall i < j : |y_i - y_j| \geq \epsilon \\ 0 \leq \alpha_k \leq 1 \end{cases}$$

Equation 3: Objects can only be in the same NN if their corresponding target values are similar enough.

where $\epsilon > 0$ is determined by the user. This problem is an instance of a linear optimization problem where the weights are adjusted so that outputs which differ by a factor at least ϵ, are

prevented from sharing the same neighborhood of size δ. Samples with a similar output value might or might not be in the same neighborhood. Finding the best distance representation by optimizing for the α's and *d*'s is generally called *distance metric learning* and Atkeson et.al. proposed several methods for achieving this, e.g. by evaluating kernel width to control the size of the neighborhood (Atkeson et.al., 1997).

One possible delimitation to the problem is to assume that the dataset could be scaled with weights that are calculated with the same distance metric for all features, i.e. calculating *d* in Equation 2 using just one distance formula. This is plausible if the features are similarly distributed hence enabling the α weights to scale the data in a close to optimal fashion. For this purpose the Euclidean, Seuclidean, Chebychev Minkowski, Cityblock, Mahalanobis, Cosine, Jaccard, Spearman or Hamming metrics could be attempted, and these will now be briefly described (these are documented in SMLT part 18, 2017).

Euclidean distance is the most commonly used distance metric and it also forms the fundament for several other distance metrics. It is calculated by the following formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Equation 4: Euclidean distance.

where *d* is the distance and *x* and *y* denote two points in an *n*-dimensional real vector space. Since the original values of the points are used normalization will be required if the features have different scales, which is the case for housing data. The Seuclidean or 'standard Euclidean' distance scales the features by their standard deviations from the features means, but the means are not transformed to lay at the center which implies, amongst other things, that we assume the features are following a Gaussian distribution. Mahalanobis distance also measures standard deviations from means and it also centers the means and transforms the data so that the distance is measured without the effect of correlation between features. Cosine distance is based on the angle between origo and the point and measures the directional similarity instead of the magnitude. Cityblock distance is similar to Euclidean distance but where Euclidean takes the root of the sum of squared distances Cityblock just sums the distances (exponent = 1). Minkowski distance is a generalization of Euclidean and Cityblock distance where the exponent is set to an arbitrary number, if it is 2 it is the Euclidean distance, if it is 1 it is the Cityblock distance and if it is ∞ it is Chebychev distance, which is the maximum of the projections on the n-dimensional coordinate axes. Jaccard and Hamming measure distances primarily on binary values and Spearman measures distances by a ranking system 'Spearman's rank correlation' (SMLT part 18, 2017).

When one searches for the best neighborhood with one of these metrics a cross validation scheme as proposed by Brownlee can be applied (Brownlee, 2016). According to an evaluation on eight synthetic datasets by Chamboon et. al. the neighborhood ended up looking very similar and leading to good results for the Euclidean, Standardized Euclidean, Mahalanobis, City block, Minkowski, Chebychev, Cosine and Correlation distance metrics, whereas Hamming and Jaccard and to a certain extent Spearman led to substandard results (Chamboon et.al., 2015). If weights are to be optimized on a dataset with mixed features, for this particular dataset this would be continuous and binary, the Gower metric can be applied, which produces two distance metrics, i.e. splitting up Equation 2 into a continuous and a binary part, calculating the distances in each of them and then recombining them (Cox, 2001).

The transform optimization can be improved to fit local information for each object's neighborhood. Hastie and Tribiani developed a method called DANN (Discriminant Adjusted Nearest Neighborhood) to adjust the neighborhood for each object by finding optimal discriminants between classes, and found that the objects received more meaningful neighborhoods this way and better classification (Hastie & Tribiani, 1996).

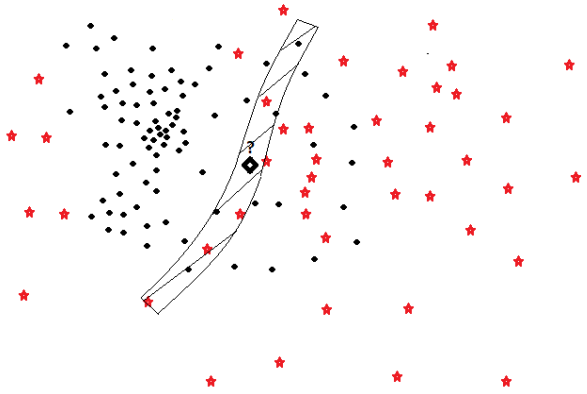

Figure 2: The DANN method constructs a tailor made discriminant for each object. All the objects within the dashed shape are part of the NN. If the correct class is a dot only 50% of the NN objects are correct in this case.

Zhang and Qingjiu (2010) proposed to first compute an ordinary kNN for an object of interest but then instead of classifying the object by all the NN objects, to first find the centroids of all labels in the NN and take the label of the closest centroid. The reasoning they used was that in regions of significant class overlap there will be objects belonging to the correct label but also the wrong label (impostors) in the NN. They found that using class centroids can be used to decrease the impact of these objects (Zhang & Qingjiu, 2010).

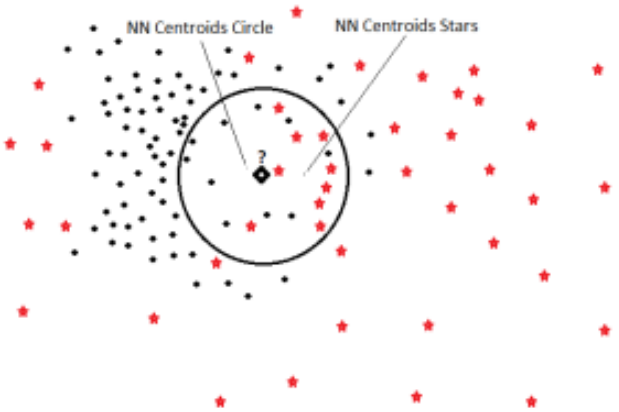

Figure 3: If dot is the correct class of the object in this case the use of cluster centroids may help lead to correct classification.

2.3 Bootstrap Aggregating and Boosting

Another way that the class overlap problem can be solved is by the use of 'Ensemble Learning' or 'Ensemble Modeling', i.e. splitting up the dataset into 'weak learners' that are then modeled separately to then be aggregated back into a 'strong learner' for evaluation (Wortman, 2011; Breiman, 1996). The evaluation is carried out through a proportionate voting or weighting scheme and if a certain weak learner has sampled many objects from a problematic class overlap region, its bad results will not disrupt the whole model since they will be downplayed by the other weak learners. Ensemble learning has been found to be an effective method to reduce predictive errors for datasets with heterogeneous sub-divisions that are hard to find on the global homogenous level (Abbott, 2001; Gashler et.al., 2008). Heavily optimized ensemble methods (particularly XGBoost) are currently winning a large portion of ML competitions for structured data (Zhu, 2016).

There exists several Ensemble methods and one is called 'Bootstrap Aggregation' (Bagging) and a closely related one is 'Boosting' (Breiman, 1996; Machova, 2006). They differ mainly through their default sampling techniques in the training phase. If the samples from the training set are determined randomly with replacement and without regard to the result of other weak learners it is 'Bagging' (Breiman, 1996). 'Boosting' populates a first weak learner randomly from the training set, builds a model from these data points and this model is then tested on the whole training set. Then the next weak learner is populated by a semi-random scheme in which badly predicted samples from the previous weak learner are given more weight. The procedure is repeated until the error of subsequent weak learners does not change much (Machova, 2006).

In both methods the default sampling size is usually set to around 60% of the total dataset. What makes these ensemble techniques so effective is that each weak learner is able to specialize to a

sub-part of the problem by only focusing at e.g. 60% of the data, by at the same time limiting the effect of outliers since results from all weak learners are studied and, through random or semi-random sampling, by preserving the weak learners ability to contribute effectively to the general solution (Sabzevari, 2014).

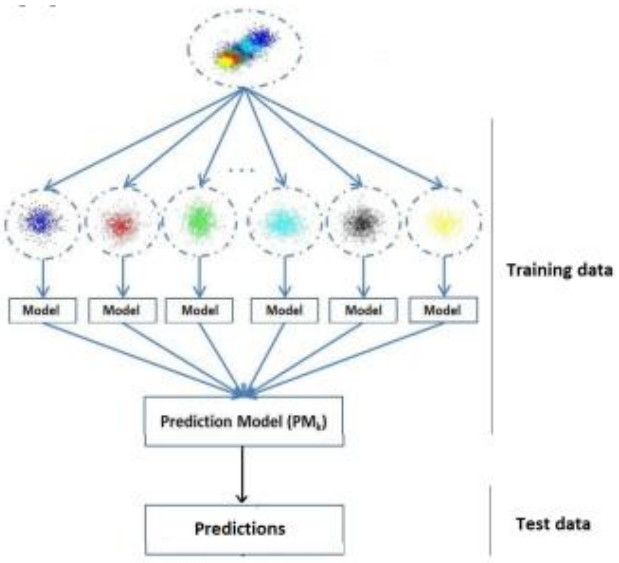

Figure 4:The general layout of an Ensemble Learning model.

Below is a more formal description of the standard bootstrap aggregation/bagging implementation:

If we have a set of coefficients that we call θ (a *statistic*) and want to optimize θ within a multivariate equation with X_1 , ... X_N samples, bagging over the θ statistic is defined as averaging the samples:

$$\theta^B(X_1, \ldots, X_N) \;=\; \mathrm{ave}_{X_1^*,\ldots,X_N^*}\; \theta^B(X_1^*, \ldots, X_N^*)$$

where the observations X_i^* are independent and identically distributed from { X_1, …, X_N}. This can also be written as:

$$\theta^B(X_1, \ldots, X_N) \;=\; \frac{1}{N^N} \sum_{i_1,\ldots,i_N} \theta^B(X_{i_1}, \ldots, X_{i_N})$$

Here we assume N^N samples, each with a probability $1/N^N$ (Buja & Stuetzle, 2002).

After the weak learners are populated a classification/prediction model is then applied to solve for θ. This could be e.g. linear, kNN or ANN regression. If samples are small the first two are probably more suitable, whereas an ANN could work better for larger ones (at least initially considering that ANN's have a big overhead for setting up the model). In previous work ANN's within bagged ensembles has been successfully applied in this manner, whereas there is a lack of research on ANN's in boosted samples since these two in conjunction can be very computationally expensive and according to some research risk overfitting (Mao, 1998; Ruqing & Jinshou, 2007, Schimbinschi, 2015).

There exist variants of Bagging and Boosting that are less common and may be more suitable for specific datasets (Buja & Stuetzle, 2002; Rakhlin, 2006). Sabzevari et. al. showed that for noisy datasets the default sample size should ideally be decreased from 60% to 20% since this decreases the effect of outliers (Sabzevari et.al, 2014).

2.4 Cluster Aggregations

If we have domain knowledge about what the weak learners should look like at the same time as the dataset is disorganized, with several known missing features and with significant heterogeneous subpatterns, random sampling may be unable to reach optimal solutions. A usual housing dataset could display the above issues because we already have a-priori domain knowledge that there exists 'housing neighborhoods' that follow diverse data-patterns (e.g. for rich beach house neighborhoods the features of interest may not be the same as for poor suburban neighborhoods) at the same time that features that have a definite impact on price (e.g. information on the auction bidding) are missing in the data. Together with these facts a housing dataset could exhibit a high level of unpredictable error and quality because it has been accumulated over several years by multiple institutions. Different institutions are likely to have carried out data collection in separate geographical areas, even more strengthening the effect of subpatterns (at least for the coordinate features). It might hence no longer be a matter of reducing the effect of outliers or dealing with noise through adjusting the sampling rate as described above, but a question of whether *random sampling* in the training phase is a viable technique on a more fundamental level (Sobhani, 2014, Breiman 1996).

These problems have been described as 'mix-effects' or 'omitted variable bias' in milder forms, or as 'Simpson's paradox' in more extreme cases (Armstrong & Wattenberg, 2014; Guyon & Elisseff, 2003). *Simpson's paradox* is the

> phenomena whereby the association between a pair of variables (X, Y) reverses sign upon conditioning of a third variable, Z, regardless of the value taken by Z. If we partition the data into subpopulations, each representing a specific value of the third variable, the phenomena appears as a sign reversal between the associations measured in the disaggregated subpopulations relative to the aggregated data, which describes the population as a whole. (Judea, 2015 p. 1).

Simpson's paradox can be shown in a figure:

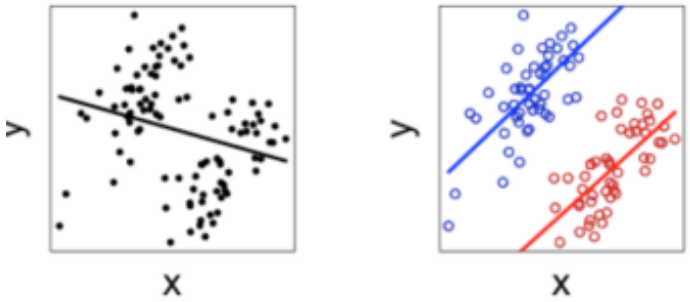

Figure 5: The direction of association between the variables can be improved and even reversed by partitioning the data.

The effects of Simpson's paradox can also be shown by a simple formula. Consider two collections of data samples A and B which we want to classify and we get 'correct classifications' *a* within *A* and correct classifications *b* within *B* and the following relationship:

Equation 5:

$$\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} A_i} \geq \frac{\sum_{i=1}^{n} b_i}{\sum_{i=1}^{n} B_i}$$

for all $i = 1, 2, \ldots, n$ and $0 \leq a_i \leq A_i$, $0 \leq b_i \leq B_i$

But at the same time there exists at least one pair of correct classification data samples that are not adhering to the above relationship and instead exhibit the following one:

Equation 6:

$$\frac{a_i}{A_i} \leq \frac{b_i}{B_i}$$

Building a whole-dataset model on the assumption that the first case in Equation 5 holds is suboptimal for certain subgroups in this case. If we optimize A with regard to the first equation it will be overfitting with regard to the second equation and if we optimize A with regard to the second equation it will overfitting with regard to the first one.

For aggregate bootstrapping there is no strong solution for this problem since the sampling is random with replacement and all weak learners are thus unable to extract substantial subpatterns. For Boosting the problem is alleviated by the larger diversity between the weak learners, but it is questionable whether it can still group together objects that belong to such subpatterns by only using the error term fixated on the target feature. For the housing case we know different neighborhoods can share a similar mean price, e.g. a neighborhood at the outskirts of a big city

can have a similar mean price as a neighborhood in the center of a smaller one, even though the ideal pricing model cannot be assumed to be the same. Boosting may be unable to reduce the error of the weak learners since these have been constructed on the assumption that optimal partitioning can be done based on weighting on predictive accuracy singly, which may not be the case.

It might be more meaningful to try a different approach, to sample the weak learners in a non-random manner and keep them independent from each other. For this purpose existing clusters in the data could be used (e.g. real geographical house neighborhoods) or, in case these are too weak they could be combined with unsupervised or semi supervised built clusters. Since evaluating clusters is subjective it is necessary to pick a technique from a stack of alternatives. The main problem is that this thrusts another layer of complexity upon the model; it is now necessary to solve two problems: 1. How to evaluate to what extent this *non-random* weak learner sampling technique works. 2. How to solve the classification/prediction problem.

There are techniques known as 'Consensus' or 'Aggregate' clustering' where stacks of clusters are evaluated through various convergence algorithms (see e.g. Gionis, 2007; Nguyen & Caruama, 2017; Monti et.al., 2003; Li & Ding, 2008). If these techniques are used within the larger scope of a prediction/classification problem it is a case of 'Model Stacking'. Wolpert (1992) presents a model stacking methodology where an outer learner evaluates results coming out of base learners to get softer and less biased results, he himself refers to this as a type of cross validation of models (Wolpert, 1992). A version of this technique 'Cascading Generalization' uses forward selection to create partitioned weak learners (Gama & Brazdil, 2000). Results were reinforced by Candillier et.al., although they heavily depend on the right selection of a clustering algorithm and parameter tuning (Candillier et.al. 2006).

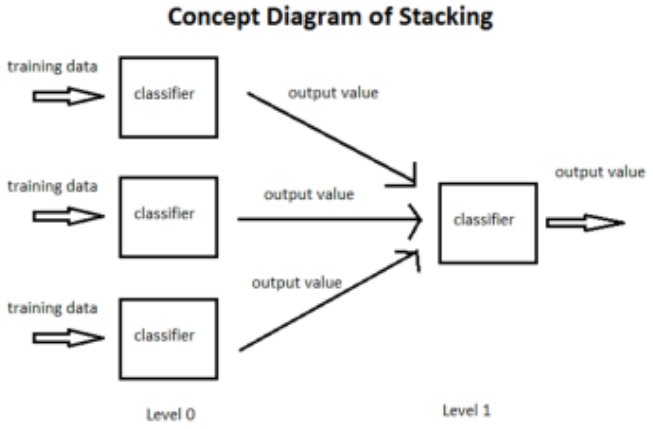

Figure 6: A simple picture of model stacking for a classification problem. Models are arranged in a hierarchical order.

There is a debate among certain researchers and communities on how to know when a dataset is in a state suitable for cluster aggregations (e.g. Statistics SE, 2012). Clustering the data

inevitably, regardless of method, brings an error into the model which was not there in the first place and the benefits of doing it must be bigger than this error. Trivedi et. al lead a discussion on this in "The Utility of clustering in prediction tasks" (2017) where they state that "Clustering … can be seen as a trade-off between the quality of the representing groups in the data and the complexity of the same" (Trivedi, 2017, p. 2). It can be seen as a tug of war between exposing the model to clustering error on the one hand, versus surrendering it to the occurrence of subpatterns in the dataset, on the other. Trivedi et. al. (2012), similarly to Candillier et.al. and Gama & Brazdil, found that there exist several cases where it works but that it requires clusterable data and parameter tuning:

> It must also be noted that clustering a dataset at a single value of k [in this case they refer to the k in k-means clustering], with any predictor … rarely improved prediction accuracy in a statistically significant manner compared to the predictor trained without clustering. But, if done at different scales with a prediction obtained at each scale and then combined by means of a naïve ensemble, the improvement is [sic.] very significant ... It was also observed that in datasets that were not very clusterable, this technique did not improve upon much. (Trivedi et. al., 2012, p. 10).

Even if it is theoretically possible to build a strong aggregate cluster model in the training phase, it might be very hard to find it because it is an implicit problem to find this model together with the corresponding ideal classification/regression solution within the built clusters. Below is a figure showing the advantages and disadvantages of the cluster aggregations method against the other mentioned:

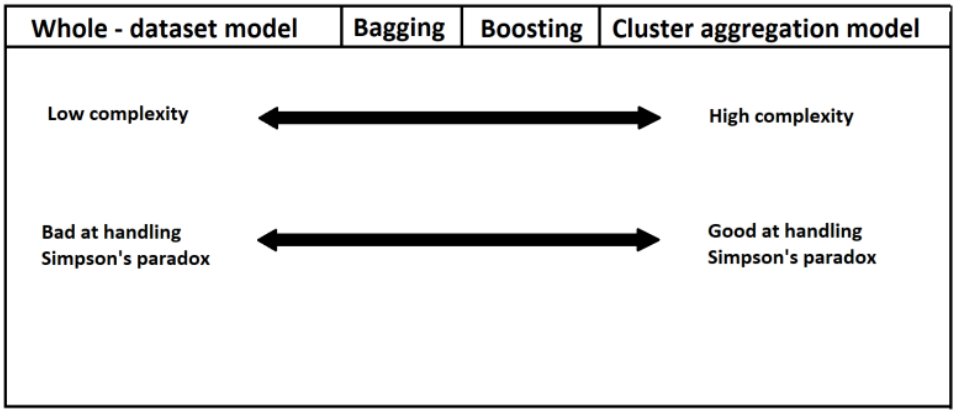

Figure 7: The differences in model complexity and ability to handle Simpson's paradox for the various models.

If such a solution is still sought in the training phase, it can be evaluated by looping through cluster ensembles, predicting prices within each cluster of the ensemble, using e.g. lazy learning in the form of kNN or eager learning in the form of linear regression or ANN's and then recombining them to evaluate how good the ensemble works.

Ari & Guvenir (2002) propose using eager learning with linear regression as opposed to lazy learning arguing that a lazy learning implementation requires too much computational time in the testing phase and that linear regression should be sufficient to solve problems in well clustered subsets. A kNN classifier/regressor requires plenty of computational time in the testing phase since each object needs to be fitted to individual weights that have been found in the training phase.

The main evaluation criterion could be aggregated predictive accuracy, but it could also include other criteria, e.g. cluster compactness or variance. Gionis et.al (2007) and Monti et. al. (2003) suggest evaluating how good a clustering algorithm is by studying how often the same objects appear in the same cluster in several ensembles and pick the algorithm where this happens most often. Below is a summarized and somewhat simplified description of the method by Monti et.al. (2003):

Assume $D = \{e_1, e_2, \dots\ e_n\}$ dataset with e items and a clustering algorithm *Cluster*, a resampling scheme *Resample*, a binary connectivity matrix *M*, a quality measure for a certain clustering attempt *Q*, a number of resampling iterations *H* and a number of cluster attempts to try $K = \{K_1, K_2, \dots\ K_{max}\}$.

$for\ each\ K\ do$
    $M \leftarrow zeros$ // preallocate the connectivity matrix
    $for\ h = \{1, 2, \dots\ H\}\ do$
        $D^{(h)} \leftarrow\ Resample\ (D)$ // shuffle the dataset
        $M^{(h)} \leftarrow\ Cluster(D^{(h)}, K)$ //cluster dataset into *K* partitions
        $M \leftarrow\ M \cup M^{(h)}$ //populate the connectivity matrix with the partitions
    $end\ (for\ h)$
    $Q^{(K)} \leftarrow\ compute\ quality\ (0-1)\ of\ the\ clustering$ from $M = \{M^{(1)}, \dots, M^{(H)}\}$
$end\ (for\ K)$
$\widehat{K} \leftarrow\ pick\ best\ K\ from\ all\ Q^{(K)}$
$P \leftarrow Partition\ D\ into\ \widehat{K}\ clusters\ based\ on\ Q^{(\widehat{K})}$
$return\ P\ and\ Q^{(K)}$

The binary connectivity matrix $M^{(h)}$ is constructed with $N \times N$ objects, where N is the number of items in cluster *K*. When a cell in the matrix is 1 it means that two objects belong to the same cluster and if it is 0 they do not. The algorithm can be made to work with sampling with or without replacement. If sampling with replacement is sought the quality of the clustering should be normalized by an indicator matrix which specifies which objects from the original dataset have been picked. The indicator matrix *I* has cells with value 1 if both of two corresponding objects are present in the randomly sampled dataset and 0 if they do not:

$$Q(i,j) = \frac{\sum_h M^{(h)}(i,j)}{\sum_h I^{(h)}(i,j)} .$$

Equation 7: The quality of a cluster ensemble is high if objects appear in the same cluster in several ensembles.

The quality is similar to a stability measure in that higher values are received if clusters manage to settle on solutions where the same objects appear in the same clusters over several runs. There are several variants and alternatives to the above method, some presented by Nguyen & Ding (2008) and Li & Caruana (2007), e.g. by creating and aggregating clusters based on individual features. All the methods, however, share that there is a loop in the training phase which delivers a stack of aggregate cluster evaluations, from which the best solution is eventually picked or found through some convergence. If this model is adopted for a prediction task it could be carried out in the following way:

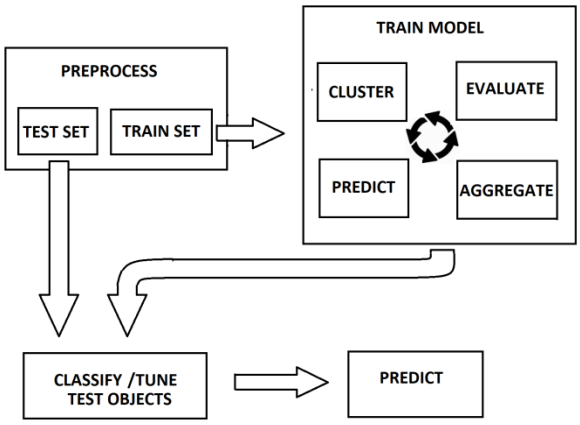

Figure 8: A conceptual drawing of how a cluster aggregation model could be incorporated within a prediction task.

The aggregate cluster model conducts four basic steps in a loop for the training data:

1. Cluster the data with a clustering algorithm (such as K-means or hierarchical clustering).

2. Run a prediction model on the individual clusters (such as kNN or ANN).

3. Aggregate the data and the results from the clusters.

4. Evaluate the results of the ensemble, e.g. by mean or median error. Other metrics can also be used, such as cluster compactness.

When a satisfactory ensemble or an aggregate of several ones have been found each object could in a more simple layout be classified to one of the clusters according to similarity and then be predicted with the training model and training data in that cluster. A more advanced layout would be to transform the test set according to various information gained when building the model in the training stage, such as cluster centroids and optimal parameters found within each

cluster. This layout is much easier to implement for a simple predictor such as kNN since it uses lazy learning and most of the work is done in the test/prediction phase.

Above there was a discussion on the potential weakness of a Boosting model since the partitioning of its weak learners is done semi-randomly with weak learner predictive accuracy as a sole criterion, which may hinder the extraction of meaningful subpatterns. The implementation shown above does not suffer from this problem since its weak learners are independent from one another and the model iterates different ensembles of weak learners until a satisfactory ensemble or aggregation of ensembles have been found. The supposition is that, for a dataset suffering from heterogeneous subpatterns, these can be built with more success by iterating ways of conducting clustering, rather than by random or semi-random sampling.

For a housing dataset Ensemble Learning with Cluster Stacking could be a suitable approach not only because of the reasons described above but also because of particular reasons only relating to housing data. A summary of all reasons is presented below:

- Known heterogeneous subpatterns.
- Known missing features.
- Partitioning of the dataset should preferably be carried out somehow anyway because the dataset is large. It is e.g. likely unnecessary to try and compare objects that are located very far away from one another. Searching for similar objects is much faster if the dataset is partitioned.
- Running computations on clusters is parallelizable similarly to aggregate bootstrapping. Boosting or Deep Learning is harder to parallelize because weight updates depend on iterative results.

3 Method

3.1 Preprocessing

Methods used to deliver clean datasets are briefly described here because the main focus of this investigation was predictive models. There were two datasets with Stockholm and Uppsala municipality objects available for this investigation: One with ~90000 objects with predictor values and more features, and one with ~330000 objects without predictor values and fewer features. Samples from the 90000 object dataset was used to develop the supervised learning models on and the 330000 dataset was only used to improve this dataset. For most of the developing process a sample with Uppsala municipality with around 5000 objects with final price was used to test the models and the final results are concerning this subsample.

Both datasets were first checked for NULL and negative values and in those cases replaced with dataset averages or removed depending on feature importance. Then modifications and recalculations of the features was carried out with the following: For the smaller dataset the x and y coordinates of each object were used to implement a distance to populated places algorithm where each object was assigned points based on how close it was to the center of various populated places. Several categorical parameters were converted into multiple binary ones in accordance to common preprocessing techniques (Pasta, 2009). The feature with the taxation value was detrended by taking the averages of each year, calculating the difference between these averages as a factor and then multiplying all values within a certain year with the corresponding factor. The distance to water was calculated with shape files with polygons of the locations of significant lakes, rivers and coastlines. For the bigger dataset values for ‘distance to 15 or 50’ nearby houses and the corresponding values for quality was calculated and this information was joined into the smaller dataset.

A copy of the dataset was taken where all features were normalized or standardized. This was done in order to keep feature coefficients on a similar scale as this is required by certain kNN distance metrics and it makes features more comparable for the optimization problem. In the cases in which a feature’s distribution passed a test for normal or χ^2 distribution the data was standardized into each point’s number of standard deviations from the mean as a way to preserve data precision (UFLDL-Stanford, 2017). These standardized features and the others were finally normalized using z-transformation, i.e. the number of standard deviations each object’s feature deviated from the mean of all the object’s for that feature.

Feature correlation and importance analysis was carried out throughout the project to see if certain features could be removed or transformed to improve certain stages of the development. The results from two initial such tests are shown below:

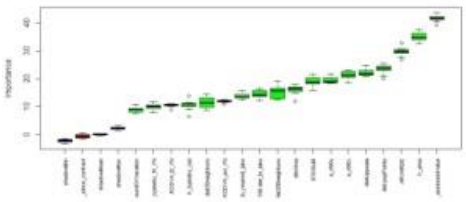

Figure 9: The importance of several features by a built in model (the R Boruta package).

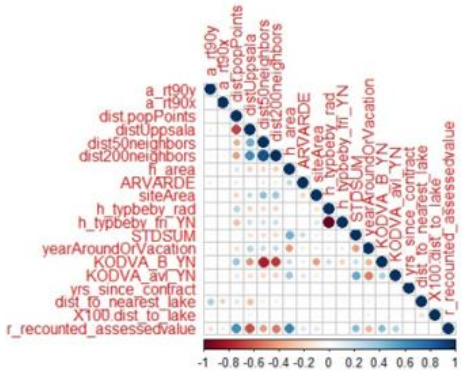

Figure 10: Correlation between several of the features.

The initial results from importance of features and features correlation was that all features were to be considered important and that severe correlation issues only appeared in a couple of obvious cases.

3.2 Division into training/validation/test set

Since Valueguard are testing their model on data one month ahead this study also used the chronologically latest data to test the models on and older data to train on using cross validation. Problems with this were that one month did not constitute a large enough nor representative sample of the whole dataset. Therefore the test-data-range was extended to between 6-12 months (which amounted to about 8-15% of the whole dataset) but this caused a slight other problem – i.e. that the data with the best quality -the newest data- was not available for training. A random data division could perhaps produce a more robust model but then it would be harder to test its performance due to a reduced 'new' test set. Also, it was of interest to study how the feature 'years since contract date' i.e. how old the predictor feature is, would affect performance. For the ANN model a validation set was picked randomly without replacement as $1/6^{th}$ of the training set applying either cross validation or split validation (Hudson, Hagan & Demuth, 2017). The test set was also divided into two parts, one which was used to develop the models and one that was used as a final evaluation.

3.3 Model Building

The training data was sent to the model building stage where it was first clustered using the k-Means or the Ward's Linkage clustering method. Ward's Linkage cluster algorithm clusters objects based on an analysis of feature variance between objects, progressively forming an agglomerative hierarchical tree structure, a dendrogram (Mojena, 2006). At the first step, all objects belong to their own cluster and with subsequent steps objects are grouped into clusters according to the variance explained by a given overarching cluster. The flexibility with the Linkage clustering algorithm is that the number of clusters returned can be predetermined. This is important for the implementation with stacked clusters as the number of objects in each cluster needs to be sufficiently large for both training and testing purposes (in the subsequent cluster group-specific neural network). A limitation to the algorithm is that binary variables have no impact on variance and hence cannot contribute to building the clusters.

In k-Means k number of clusters are determined a-priori and are represented as cluster-centre points in a multidimensional input space (Macqueen, 1967). At the first iteration, the k cluster-points are inserted into the input space as far away from each other as possible. Each data-object is then associated to its nearest cluster-point. In the next iteration, each cluster defined in the previous step generates a new cluster-point, placed at the center of weight of the cluster determined in the previous step. Each object is then updated by associating it with the nearest cluster-point. This process is looped until there is no more movement in the k cluster-points, which now represent the center of weight of k number of clusters. In the current implementation, four different distance calculation methods were implemented with k-Means, sqeuclidean, cosine, cityblock and correlation. The number of clusters specified apriori for the k-Means clustering implementation varied between 2 and 10.

Once clustered, objects were assigned cluster ID's and a model was built on each cluster using cross validation. The procedure was repeated a number of times in order to find the best clustering settings and was different depending on whether eager or lazy learning was used.

3.3.1 Test objects tuned to a single weak learner

A neural network was implemented using the MATLAB 2016b neural network toolbox to predict prices for test objects in each cluster. I.e. the models and data within the models were partitioned and kept separate in their partitions from the clustering stage onwards. The clustering was unsupervised and only the training set was used to construct these (the final result cannot be properly evaluated if the test set is included here). For each cluster there were thus a produced training set, a validation set and a test set. A constraint faced was to ensure that each cluster was assigned a sufficient number of objects (>30 objects) so that the subsequent test, training and validation set sizes would be large enough to ensure a meaningful implementation of the ANN

and kNN models. When clustering a sample of objects in the Uppsala region this limitation meant that the number of clusters that could be generated, ranged between 1 and 10 clusters, depending on the clustering algorithm used. The number of clusters, parameter selection, and clustering algorithm were at first randomized, and then optimized for reducing prediction error using grid search, which is a simple brute force algorithm where all combinations of parameters are tried within constraints (Mao, 1998; Ruqing & Jinshou, 2007).

The ANN model was applied to these clusters through MATLAB's 'fitnet' with semi-randomly selected hyperparameters: The training function was selected from the following list: {Levenberg-Marquardt, Bayesian regularization, Scaled conjugate gradient, Resilient backpropagation, Fletcher-Powell conjugate gradient, Polak-Ribiere conjugate gradient, one step secant, variable learning rate backpropagation}. Number of hidden neurons and layers were selected as between 9 and 30 hidden neurons if 1 layer was used, between 9 and 15 neurons if 2 layers were used and between 9 and 12 neurons if 3 layers were used, with the number of layers being picked from a gamma distribution, promoting a smaller architecture. The assumption was that results would show whether larger or smaller architectures work better (see 'delimitations' above). The stopping condition was picked as 5 – 10 epochs where the validation set did not see an improvement in predictions. For the cost function the default of the corresponding training function was used as they do not share the same ones. Most commonly 'MSE' or mean squared error was used.

Features were selected and weighted semi randomly in each model iteration with selection/weights saved in tabular form. The weights, α, were set in the range $0 \leq \alpha \leq 1$. During the developmental process results were continuously analyzed and if certain features were observed to improve or degrade the performance of the models their weights/selections were adjusted accordingly either for the whole dataset (if clustering was not used) or for each corresponding dataset (if clustering was used).

After the network had been trained on the training set it was tested on the test set as well as the training set. This was done so that the MDAPE's could be compared for the two sets and if they differed by a large degree it could be seen as an indication of an unideal data-division or overfitting. All hyperparameters used as well as computational time as MATLAB's 'tic-toc' and 'cputime' for the model were recorded.

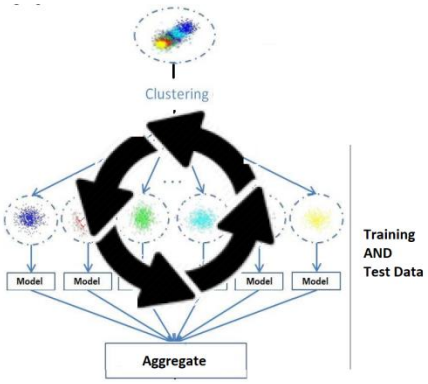

Figure 11:Conceptual drawing of cluster aggregation with test objects classified to single clusters. The test objects were sent together with training objects to clusters (built by training data) and predictions were made inside of these.

The ANN and kNN models were initially tested using the same MATLAB framework i.e. the one described above where the test data was classified to a single weak learner, assuming weak learner independence for optimal results (the multiple weak learner model for the kNN presented below in 3.3.2 was built after this first implementation). The kNN implementation was also carried out in MATLAB for this with its built-in 'knnsearch' function (SMLT, 2017). 'knnsearch' takes in a normalized dataset and a distance metric as parameters and outputs the location and distance to the nearest neighbors in the dataset in sorted order. The distance metric was initially selected randomly from the following list: {Euclidean, Seuclidean, Cityblock, Chebychev, Minkowski, Mahalanobis, 'hamming', 'cosine', 'correlation', 'spearman', 'jaccard'}. Although 'jaccard' and 'hamming' were considered theoretically unsuitable they were still used initially as controls to understand the effect of distance metrics. In initial testing all of the distance metrics were found capable of delivering reasonable results on the pre-processed dataset.

After each object had received a NN based on the 'training' part of the data a prediction was made using a weighted average with closer objects getting more weight. All hyperparameters used as well as computational time as MATLAB's 'tic-toc' and 'cputime' for the model were recorded.

Hence the first step conducted a 'rough' search with certain limitations on the input space and the second step conducted the final tuning to attempt to reach the best solutions. This general technique was used in order to limit the search space and maintain more control over how the clustering was conducted.

3.3.2 Test objects tuned to multiple weak learners

As described in 2.2 it is possible to transform each test object in various ways to provide the best NN's for test objects. This implementation was only attempted on kNN since a lazy learner where most of the model building occurs online in the actual testing phase, was assumed to be less time requiring and more suitable: The construction of NN's in the training phase can be expected to provide more meaningful information compared to ANN's, whose optimized parameters are less certain to work well 'outside' the clusters they have been trained on.

Information from all created clusters was thus used in this implementation with kNN according to the following procedure which was implemented in Rapidminer 7.6:

1. First the training set with all features was normalized and clustered just as in the previous case but here there was no constraint as to how many clusters could be used, so number of clusters varied between zero and the number of objects in the dataset. For k-Means certain constrictions had to be implemented here because it requires a high amount of computational time when the number of clusters grows to hundreds. For Ward's Linkage clustering this was not a problem since the model only needs to be built once (the dendrogram is then cut at various levels). Clusters that ended up with fewer than 14 objects were removed. The centroids of all calculated clusters were saved for later use. For k-Means the centroids are represented as means and for Ward's Linkage the means were calculated as feature means.

2. The data within each cluster was used to optimize weights and other information for the kNN model inside the same cluster. The weight optimization algorithm used was a wrapper forward selection where combinations of weights were tried within constraints in an iterative fashion. First the algorithm runs with each feature individually, then the best features are combined with corresponding weights and more features are progressively added until results stop improving (Guyon & Elisseff, 2003; Kaushik, 2016). The constraints were the following: Feature weights (α): 0 – 1. If certain features were deemed unhelpful to the kNN algorithm they received a 0 value. In Rapidminer 7.6 this optimization function is called 'optimize weights (forward)' and the documentation includes the following description:

> This operator performs the weighting under the naive assumption that the features are independent from each other. Each attribute is weighted with a linear search. This approach may deliver good results after short time if the features indeed are not highly correlated. (RMD, 2017)

This optimization method was assumed to perform well since the correlation analysis in section 3.1 did not show a significant problem of correlation between features and because each weak learner optimizer had the ability to throw out features that were deemed unimportant – including those that were highly correlated. To make sure correlation was not an issue certain features were removed from the model.

The optimal k in kNN was searched for within the same optimization with values ranging between 1 and the population size of the weak learner the object was located in. For data splitting cross validation was used.

3. Each test object's distance to all cluster centroids was calculated using Euclidean distance.

4. Each test object's features were scaled according to multiple clusters optimal weights according to the following formula:

Assume a set of objects $X = \{x_1, x_2, \dots\ x_n\}$, a cluster optimized weight vector $W = \{\alpha_1, \alpha_2, \dots\ \alpha_n\}$, distance to cluster Centroids $D = \{d_1, d_2, \dots\ d_n\}$, and an object specific weights matrix, $A = W\ \times D\ \in\ \mathbb{R} \times \mathbb{R}$. Then, for a test object *i* and a feature *j*, the optimal weight is calculated by the following equation:

$$A_{ij} = \frac{\sum_{k=1}^{n} \alpha_{kj} d_{kj}}{\sum_{k=1}^{n} d_{kj}}$$

Equation 8: How test objects were adjusted based on optimal cluster weights and corresponding distances to the clusters.

where k iterates over the number of objects in a particular cluster. Hence we can see an object's weight for a certain feature is determined by a weighted average of optimal cluster weights with distances to those clusters. Below is a method for how this object specific weight optimization was implemented:

Assume $G = \{g_1, g_2, \dots\ g_n\}$ larger scale model running attempts where each g represents a unique set of hyperparameters for the implementation, a $D = \{e_1, e_2, \dots\ e_n\}$ normalized dataset with e items, a clustering algorithm *Cluster*, a resampling scheme *Resample*, a training set matrix *M*, a number of resampling iterations *H* and a number of cluster attempts to try $K = \{K_1, K_2, \dots\ K_{max}\}$.

$for\ each\ G\ do$ // outer loop for tuning ensemble hyperparameters

$D^{(train)}$; $D^{(test)} \leftarrow\ Resample\ (D)$ // sample a test and training set
$R \leftarrow zeros$ // preallocate test results matrix

// Training phase
$for\ each\ K\ do$ // ensemble learning attempts
    $M \leftarrow zeros$ // preallocate training set matrix
    $C \leftarrow zeros$ // preallocate centroids matrix
    $A \leftarrow zeros$ // preallocate optimal weights matrix
                // the last column of this matrix contains zeros for optimal k's

```
        for h = {1, 2, ... H} do // clustering partitions
            M^(h) ← Cluster(D^(train), K) //cluster dataset into K partitions
            M^(h) ← M^(cluster) − smallClusters( M^(h)) // remove if too small
            C^(h) ← computeCentroids(M^(h)) // using L2 norm
            A^(h) ← optimize(M^(h)) // optimize weights using grid search
                                    // optimal k also optimized for each h
        end (for h)
        M ← M ∪ M^(h) //append results into corresponding matricies
        A ← A ∪ A^(h)
        C ← C ∪ C^(h)
    end (for K)

    // Testing phase
    for each T do // for each test object
        A^(T) ← computeOptimalWeights(C, A) // see Equation 8
        k ← getOptimalK(A^(T))
        T ← T × A^(T) // scale the test object with optimal weights
        D^(T) ← D^(train) × A^(T) // scale training set with optimal weights
        P^(test) ← kNN(k, D^(T))
    end (for T)
end (for G)
```

There is hence one outer loop which tunes hyperparameters for the ensemble learner, i.e. starting values for number of clusters, random seed numbers and clustering algorithm. Within this outer loop there are two loops, one which builds a model out of the training data and one which tests test objects using the built model. The test phase is quite computationally intensive compared to the single cluster models since the training set needs to be scaled with the optimal weights for each object.

The model was implemented in Rapidminer 7.6 using a total of 80 operators divided into around 15 subprocesses and 5 scripts. Below is a screenshot of the implementation in Rapidminer. Observe the similarity to Figure 8.

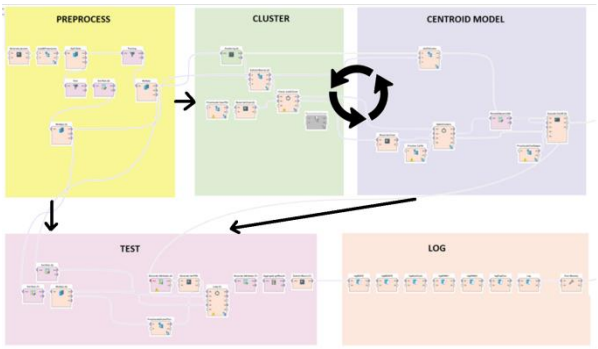

Figure 12: The Rapidminer implementation of cluster aggregations where test objects were tuned to multiple clusters. The operators were placed in a manner resembling that of Figure 8.

Duplicates: If the object has been sold before

Around 20% of the objects appear several times in the dataset, i.e. they have been sold before and have the same unique key and similar features but different sale-dates and final price. Before a regression model was applied to predict an object a search through the dataset was conducted to see if the object had been sold before and if so the final price of this sale was recorded. After the regression model had delivered its prediction a weighted average was taken between the previous sale price and the model prediction, with the weight being declared randomly as a hyperparameter.

4 Results

The results concern the 5000 object strong Uppsala municipality sample. Tests were also run on several other samples to confirm the feasibility of the results. The best predictions were achieved with the cluster aggregation model with the test set tuned to multiple clusters at 8.31% MDAPE when the test set constituted three unseen months worth of data into the future and the average of the seven 'latest' ensembles was used. The difference on a monthly basis was about ± 1.2% MDAPE so the expected final result for an unseen month test set should be expected to lie within a range of 8.3 ± 1.2% MDAPE. The optimization of weights, k and the bounds of the grid-search was limited for the scope of this study and better results are likely to be achieved given more computational resources and certain improvements (see section 0). Below is a figure that shows how the best result was achieved by continuously adding more cluster ensembles to aggregate:

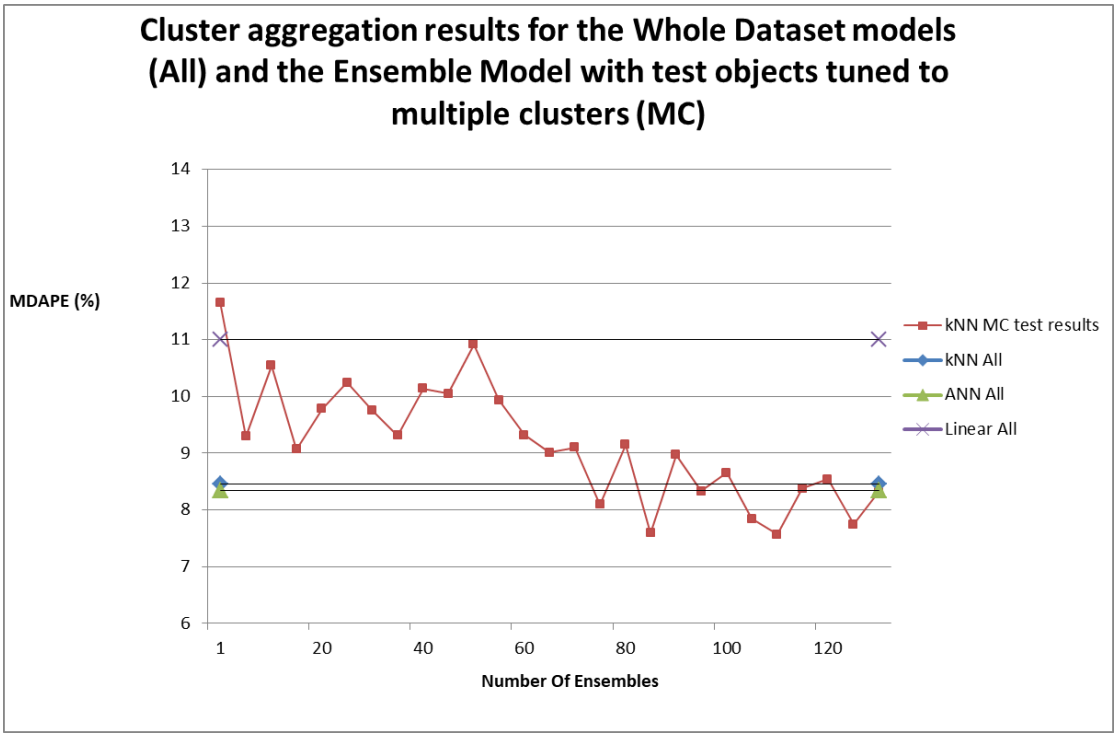

Figure 13: The results of the cluster aggregation model with test objects tuned to multiple clusters.

The figure shows that the results, although improving, did not converge and that test sets have predictive results ranging between approximately 7.8 – 8.8% MDAPE when the number of ensembles evaluated are more than 100. The higher number of ensembles there were the more centroids each test object was adjusted to. The figure shows that the model did not see much improvement after more than 100 ensembles were stacked. The average results and CPU times (4 core processor) from all the models are shown below:

	Linear	kNN All	ANN All	kNN SC	ANN SC	kNN MC
MDAPE (%)	11	9.02	8.35	8.47	8.35	8.31
MAPE (%)	18.5	14.9	13.5	14.2	13.7	13.6
CPU-time training (min)	0.1	17	26	11	18	14
CPU-time predicting (min)	0.1	1.3	0.1	1.5	0.1	7.3

'Linear': Linear regression
'All': No partitioning of the data was carried out
'SC': The test objects were tuned to single clusters
'MC': The test objects were tuned to multiple clusters.

Figure 14: Average results from the models.

The single cluster models were limited by the fact that they were only partitioned into 1-10 clusters, something that may very well may have weakened results. K-Means in smaller tests generally performed better than Ward's Linkage with results about 0.5% MDAPE below hierarchical, but the clustering results displayed above were all generated with the latter method since it was found too time requiring to run the process with K-Means. In section 0 there is a discussion on how this could have compromised results.

The results in Figure 14 were obtained with a test set spanning 3 months of 'latest' data. This was necessary to have a sufficiently large test set to work with. The monthly differences with regard to MDAPE were around ±1.2 %.

The CPU times for all the clustering models are not to be considered definite because it remains uncertain how many clustering iterations are needed for results to converge. The multiple cluster model was run with a total of 45 iterations (i.e. 45 stacked ensembles) in the training phase but the CPU time shown for the model is only for one loop. The results from the 45 loops can be reused for a new unseen test set so it would be inaccurate to combine them all for the CPU time.

5 Discussion

The discussion is structured around the two research questions. What was the performance of the cluster aggregation models in terms of:

1. Minimizing predictive error?

The use of cluster aggregations models for this study proved to be a promising technique but too complex for definite evaluation. It was not expected to necessarily work unless the dataset exhibits significant heterogeneous sub-patterns and results, although inconclusive, were on par with or better than the corresponding whole dataset ones. The technique hence seems warranted and has great potential for strong predictions given finalization of the training procedure. Given more time the results shown in Figure 13 could at the very least be expanded enough to build up prediction confidence intervals.

The best clustering algorithm for the cluster aggregation model was found to be k-Means over Ward's Linkage agglomerative method and this is likely because the sizes of the optimal clusters varied.

Figure 15: One of the dendrograms delivered by the hierarchical clustering model. It is hard to surmise an optimal height to cut it.

As we can see in a built dendrogram for the 5000 dataset with 12 features in Figure 15 clusters appear with varying size. The hierarchical method requires the dendrogram to be cut at an optimal point to work well and it requires cluster sizes to be similar. By studying the dendrogram we can see that cutting the dendrogram at any point will produce some meaningful clusters while it will break others up. Even so this model produced the best results (k-Means was never attempted on this model) and that is mainly why it is believed to be improvable. The implementation of cluster aggregations in this study did not attempt any more advanced techniques to stack the cluster models which could be based on the research by Gionis et.al, Monti et.al., Wolpert, Candillier et.al. and Gama & Brazdil (Gionis et.al., 2007; Monti et. al., 2003; Wolpert, 1992; Candillier et.al., 2006, Gama & Brazdil, 2000).

In order to reach better results feature engineering in the form of PCA and more pre-processing, data collection including finding new features (such as bidding history of property sales) and manual changes based on domain specific knowledge would probably also help.

2. Minimizing computational time?

Concerning CPU times the ones for ANN's in the testing phase were very low compared to the kNN models. This is in line with the analysis by Ari & Guvenir concerning CPU times of eager and lazy learning (2002): ANN's are much faster than kNN in the test/predicting phase because they apply eager learning with the bulk of the computational time spent on model training. For the kNN model in the cluster aggregation model with test objects tuned to multiple weak learners, each test object searches through all training objects to construct the optimal NN and then conducts a weighted average calculation, O($ndm + kn + l$), where n is the cardinality, d is the dimension, m is the distance calculation using some distance metric, k is the neighborhood size and l is the weighted average part. The problem is the n * d * m multiplication which becomes large when the whole dataset is used. The search for the NN is parallelizable as discussed in section 0, which can bring down computational time by a significant amount (Cartmell, 2010). The comparative time complexity for predicting test objects with a ANN is approximately O(NL) where L is number of hidden layers and N is the average number of neurons in each hidden layer.

For the ANN's the training set time complexity could be seen as laying somewhere between O($nd + NL$) and O($n^3 + NL$) but these numbers should not be considered definite because there are a variety of hyperparameters, such as the training function, which affect time to a large extent. This study could also not produce a meaningful quantitative comparison between lazy kNN and eager ANN learning because the kNN regression model was always run within the framework of an optimization model which searched for weights, k and other hyperparameters. As the study progressed it was on several occasions found that the constraints of the various optimizers could be modified to reach better results faster, undermining previous computational time conclusions. The multiple cluster aggregation model is particularly hard to evaluate in terms of exact time complexity since it relies on building a stack of cluster centroids, the ideal amount of which remains unclear as evinced by Figure 13. This model was only attempted with kNN regression since it was deemed unsuitable to apply black box ANN's to it. For a fair comparison the ANN's could have been implemented within a manually constructed deep learning architecture with certain hidden layers specialized on data partitioning and certain hidden layers specialized on feature selection. In summary, research question 2 concerning computational time could not be answered definitely. For all the cluster aggregation model the advantages pertaining to parallelization explained in section 0 can be confirmed by simply observing that there is no dependency between the weak learners (Abbott, 2011).

Concerning a-priori assumptions

As mentioned earlier the underlying intuition for applying cluster aggregations on a housing dataset was the domain knowledge that houses are divided into 'neighborhoods' which may or may not follow the same pricing patterns. In the worst case these different patterns can accumulate to Simpson's paradox behavior, which can be disparaging for a whole dataset model (section 2.4). According to related research cluster aggregation models can be expected to perform as good or better than whole dataset models or bagging and boosting for a dataset that suffers enough from these issues (section 2.4). Cluster aggregations has a higher model complexity though and there is a risk that the training set will not converge, as in this study, with final results only being output as a range. It is hence as it is so often a case of 'no free lunch' where model complexity weighs against potential advantages (see e.g. Figure 7). Although no exact line can be drawn between model complexity and potential for solving Simpson's paradox, in ML communities the rule is to start with a simple model first and then progress towards more complex models if satisfactory results could not yet be attained (e.g. SOE, 2017). This study can ratify this methodology and it should also be clarified that 'satisfactory results' can only be definitely achieved if the result converges. Even if there are strong a-priori assumptions concerning certain subpartitions in a dataset, making a cluster aggregation model highly likely to deliver best predictions, it should still in the ideal case be preceded by models that are known to converge.

kNN regression compared to ANN regression

Clustering allowed for more information on what sizes and type of datasets that suited the two regression models. ANN's received better results than kNN when the data was divided into many smaller clusters, whereas kNN was slightly better when there were fewer larger clusters. One easy explanation for this is that kNN is more protected against the effect of outliers compared to ANN's because it cannot select outliers for the NN which the predictions are based on (especially since robust regression was applied). Even though ANN's did not get the benefit of local modeling for predictions or adjustments it still beat KNN in many of the cases, which attests to its flexibility in dealing with noisy data (Stergiou, 1996; Elizondo, 2006).

There are several interesting observations from the ANN results. For example, there is no strong indication that a larger ANN architecture would reduce predictive errors, even when the whole dataset was used. One possible explanation for this is that there is a very high inherent error in the predictor feature, at least 10 % mean error on average, due to the uncertainty that accompanies a house sale, and that the small improvement that could accompany an increase in ANN architecture size is overarched by this error. The optimal training functions were found to be the Levenberg-Marquardt algorithm and Bayesian regularization and in almost all clusters they performed similarly well, with a slight but insignificant edge for Bayesian regularization for

datasets that were more difficult to predict correctly. The other training functions produced results that were around 2 – 3 % higher with MDAPE and this is too much to make them competitive even if their computational time is lower.

As expected normalization, standardization and weighting did not improve predictions for ANN's as this procedure is already being carried out within the model as weight – adjustments. Any pre-normalization/standardization/weighting can speed up the training but the effect is mostly limited given a large dataset.

6 Future Work

- The ensemble Models developed here should be compared against already existing ensemble models rather than whole-dataset models. Such models could be bootstrap aggregations and boosting, but also random forests.
- Feature engineering, PCA and collecting more features. Data could be gathered from map services, such as distances to road and water.
- An artificial dataset could be constructed in which heterogeneous sub-patterns could be progressively increased. Different models could be compared in terms of how well they can handle the increasing presence of Simpson's paradox.
- Valueguard AB should create an experimental dataset on which they can run their predictive model. Now they are only running predictions 1 month ahead on all of Sweden, which is too much data to run experiments with new models.

7 References

Abbott, Dean: Benefits of Creating Ensembles of Classifiers, The data administration newsletter, 2001, http://tdan.com/benefits-of-creating-ensembles-of-classifiers/4960, 8-July-2017.

Ari, Bertan H.; Guvenir, Altay: Clustererd linear regression, Knowledge-Based Systems 15, 2002.

Armstrong, Zan and Wattenberg, Martin: Visualizing statistical mix effect and Simpson's paradox. Google Research, 2014.

Atkeson, Christopher G. et.al.: Locally Weighted Learning, Artificial Intelligence Review 11, 1997.

Bakker, Bart; Heskes, Tom: Clustering ensembles of neural network models, Neural Networks 16, 2003.

Bergstra, James; Bengio, Yoshua: Random Search for Hyper-Parameter Optimization, Journal of Machine Learning Research 13, 2012.

Blum, Avrim; Langford, John: PAC-MDL bounds, CMU, 2003.

Breiman, Leo: Bagging Predictors, Machine Learning Volume 24, Issue 2, 1996.

Brownlee, Jason: Machine Learning Mastery online, 2016: http://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/, 8-July-2017.

Buja, Andreas; Stuetzle, Werner: Observations on Bagging, AT&T Labs, 2006.

Candillier, Laurent; et.al: Cascade evaluation of clustering algorithms, European Conference on Machine Learning, 2006.

Cartmell, John: Methods to Pre-Process Training Data for K-Nearest Neighbors Algorithm, InterDigital Communications LLC, 2010. Concerning parallelization he refers to https://code.google.com/archive/p/cell-knn/wikis/kNN.wiki, 14-July-2017

Chauarasia, Shalinee and Jain, Anurag: Ensemble neural network and kNN classifiers for intrusion detection. International Journal of Computer Science and Information Technologies, 2014.

Chen, Ruqing and Yu, Jinshou: An improved bagging neural network ensemble algorithm and its application. Third International Conference on Natural Computation, 2007.

Chomboon, Kittipong, et al.: An Empirical Study of Distance Metrics for k-Nearest Neighbor Algorithm, Proceedings of the 3rd International Conference on Industrial Application Engineering, 2015.

Cox, T.F.: Multidimensional Scaling, Chapman and Hall, 2001.

Duda, Richard O.; Hart, Peter E.; Stork, David G.: Pattern Classification 2nd edition, Wiley, 2000. Original version published 1973.

Eberhart, Bert: Computational intelligence, 1st edition. Elsevier, 2007.

Eikhoff, Ralf, et al: Robust local cluster neural networks. European Symposium on Artificial Neural Networks Bruges, 2006.

Elizondo, D: The linear separability problem: Some testing methods in Neural Networks. IEEE Transactions on, vol.17 - 2, 2006.

Ester, Martin; et.al.: A density-based algorithm for discovering clusters in large spatial databases with noise. KDD 96 proceedings, 1996.

Everitt, B. S.; Landau, S.; Leese, M. and Stahl, D: Miscellaneous Clustering Methods, in Cluster Analysis, 5th Edition, John Wiley & Sons, Ltd, Chichester, UK, 2011.

Gajawada, Satish, et. al.: Optimal Clustering Method Based on Genetic Algorithm, Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS), 2011.

Gällmo, Olle: Machine Learning course, https://studentportalen.uu.se/portal/portal/uusp/student/student-course?uusp.portalpage=true&toolMode=studentUse&entityId=139924, 11-November-2017.

Gama, Joao; Brazdil, Pavel.: Cascade Generalization, Machine Learning 41, 2000.

Gashler, Mikel et.al.: Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous , Department of Computer Science Provo, UT, U.S.A., 2008.

Gionis, Aristedes; Manilla, Heiki; Tsaparas, Panayiotis: Clustering Aggregation, ACM Transactions on Knowledge Discovery from Data, Vol. 1, No. 1, 2007.

Guvenir, H. Altay; Aynur Akkus: Weighted k nearest neighbor classification on feature projections, TUBITAK project, Bilkent University, 1996.

Guvenir, H. Altay; I. Uysal: Regression on feature projections, Elsevier, 2000.

Guvenir, H. Altay: Clustered linear regression, Elsevier, 2002.

Guyon, Isabelle; Elisseeff Andre: An Introduction to Variable and Feature Selection, Journal of Machine Learning Research 3, 2003.

Hassan, Rania, et. al.: A comparison of particle swarm optimization and the genetic algorithm, American Institute of Aeronautics and Astronautics, 2004.

Heckman, Judd, et. al: Where should you live? Berkley School of Information, 2015: from https://www.ischool.berkeley.edu/projects/2015/where-should-you-live, , 8-July-2017.

Hudson, Beale Mark; Hagan, Martin T.; Demuth, Howard B.: Neural network toolbox(TM) User's Guide. The MathWorks Inc., 2017

J. B. MacQueen: Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, 1967.

Jeff, M, Philips: Data mining course - spring 2016, https://www.cs.utah.edu/~jeffp/teaching/cs5140/L19-Noise.pdf, 2017.

Jovanovic, Radisa and Sretonovic, Aleksandra A.: Ensemble of Radial Basis Neural Networks With K-means Clustering for Heating Energy Consumption Prediction, FME Transactions, 2017.

Kennedy, J. and Eberhart, R. C: Particle swarm optimization. Proc. IEEE int'l conf. on neural networks Vol. 4, 1995.

Kohavi and G. John. Wrappers for feature selection. Artificial Intelligence, 1997.

Kohonen, T: Self-Organizating Maps, New York - Springer-Verlag, 1997.

Lee, Sangwook, et. al.: Modified binary particle swarm optimization, Progress in Natural Science, 2008.

Li, Tao; Ding,Chris: Weighted Consensus Clustering, SIAM, 2008.

Lim, Teng Wan, et. al.: Housing price prediction using neural networks, Fuzzy Systems and Knowledge Discovery. ICNC-FSKD Conference paper, 2016.

Machova, Kristina; Puzsta, Miroslav, Barcak, Frantisek, Bednar, Peter: A Comparison of the Bagging and the Boosting Methods Using the Decision Trees Classifiers, ComSIS Vol. 3, No. 2, 2006.

Mao, Jianchang. A case study on bagging, boosting and basic ensembles of neural networks for OCR. IEEE World Congress on Computational Intelligence, 1998.

McCullogh, Warren S.; Pitts, Walter H.: A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 1943. http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf, 8-July-2017.

Mojena, Richard: Ward's clustering algorithm. Encyclopedia of Statistical Sciences, 2006.

Monti, Stefano; et.al.: Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data, Machine Learning 52, 2003.

Nam Nguyen, Caruana Rich: Consensus Clusterings, Cornell University, 2007.

Ougiaroglou, Stefanos; et.al.: Adaptive k-Nearest-Neighbor Classification Using a Dynamic Number of Nearest Neighbors, DIEE, 2007.

Pasero, Eros; Mesin, Luca: Artificial Neural Networks to Forecast Air Pollution, Intech, Environmental Engineering, 2010.

Pasta, David J, et.al.: Learning When to Be Discrete: Continuous vs Categorical Predictors. ICON Clinical Research, 2009.

Pearl, Judea: Understanding Simpson's paradox, The American Statistician, 2015.

Pow, Nissan et.al.: Prediction of real estate property prices in Montreal, Thesis Project, McGill University, 1999.

Quesada, Alexandro: Artificial Intelligence Techniques; Neural Designer online teaching, 2017: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network, 8-July-2017.

Rakhlin, Sasha: Bagging and Boosting, MIT lecture series, 2006.

RMD : Rapidminer Documentation on Optimize Weights (Forward), 2017. https://docs.rapidminer.com/studio/operators/modeling/optimization/feature_weighting/optimize_weights_forward.html, 12-September-2017.

Sabzevari,Maryam; et.al.: Improving the Robustness of Bagging with Reduced Sampling Size, ESANN proceedings, 2014.

Saurav, Kaushik: Introduction to Feature Selection methods with an example, Machine Learning Python, Analytics Vidhya, 2016, https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/, 8-September-2017.

Schimbinschi, Florian: Machine Learning Forum Discussion concerning lack of research on boosting, 2015. https://stats.stackexchange.com/questions/185616/boosting-neural-networks, 8-May-2017.

Singh, Amarpal; Saxena, Piyush; Lalwani, Sangeeta: A study on various training algorithms on neural network for triangular based problems, International Journal of Computer Applications, 2013.

Sinta, Dewi: Ensemble k-nearest neighbors method to predict rice price in Indonesia, Applied Mathematical Sciences, 2014.

SMLT: Statistics and Machine Learning Toolbox documentation. The MathWorks Inc., 2017.

Sobhani, Parinaz; et al.: Learning from Imbalanced Data Using Ensemble Methods and Cluster-based Undersampling, University of Ottawa, 2014.

SOE Stack Overflow Stack Exchange Community, https://stackoverflow.com/questions/16686966/machine-learning-algorithms-which-algorithm-for-which-issue, 8-Aug-2017.

SSE Statistics Stack Exchange Community: https://stats.stackexchange.com/questions/31440/clustering-as-a-means-of-splitting-up-data-for-logistic-regression, 8-July-2017.

Stergiou, Christos; Siganos, Dimitrios: Neural Networks Computer Science Online Resource, 1996: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html, 8-July-2017.

Trivedi, Shubdhendu, et.al.: The Utility of Clustering in Prediction Tasks, University of Chicago, 2017.

Weinberger, Kilian Q.; Saul, Lawrence, K.: Distance Metric Learning for Large Margin Nearest Neighbor Classification, Journal of Machine Learning Research 10, 2010.

Werbos, Paul J.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, PhD thesis, Harvard, 1974.

Wolpert, David H.: Stacked Generalization, Neural Networks, 5, 1992.

Wortman, Jennifer Vaughan: Machine Learning Theory, Lecture text, UCLA, 2011. http://www.jennwv.com/courses/F11/lecture13.pdf, 8-July-2017.

Yiyuan She: Sparse regression with exact clustering, Phd Thesis, Stanford, 2008.

Zhang, Quingjiu; Sun Shiliang: A centroid k - Nearest Neighbor method, ECNU, 2010.

Zhu, Nan et.al: XGBoost: Implementing the Winningest Kaggle Algorithm in Spark and Flink, Forum Post, https://www.kdnuggets.com/2016/03/xgboost-implementing-winningest-kaggle-algorithm-spark-flink.html, 8-Aug-2017.