Advances in Data Mining: assignment 3

Sjoerd Hermes & Miltiades Violaris

December 12, 2020

1 Problem and data

This report consists of two different sets of problems: a classification and regression problem and a dimensionality reduction and data visualization problem. To this end, various algorithms were employed: the RandomForest and XGBoost algorithms were used during the classification and regression tasks, whereas the Principal component analysis (PCA), Locally-Linear Embedding (LLE) and t-distributed stochastic neighbor embedding (t-SNE) were used during the dimensionality reduction and data visualization tasks. These algorithms were applied on the following data sets:

- The Iris data, obtained from: `https://www.kaggle.com/uciml/iris`.
- The Digits data, which is a data set from the Scikit-learn package in Python and can be obtained by using the `sklearn.datasets.load_digits(n_class=10)` command.
- The Hotel booking demands data, obtained from: `https://www.kaggle.com/jessemostipak/hotel-booking-demand`

The Iris data set is perhaps one of the most well known multivariate data sets, consisting of 4 independent variables: `SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm` and one dependent variable: `Species`. This means that the Iris data is 150×5 dimensional.

The Digits data is a reduced version of the MNIST data, which is a larger (compared to the Iris data set) data set consisting of hand-written digits $\in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. As each of the 1797 samples in the Digits data is an 8×8 image, the Digits data is 1797×65 dimensional (digit label included).

The third and final data set, the Hotel booking demands data, needed a lot of preprocessing before it was usable on the tasks at hand. This prepossessing is covered in section 3.1. After preprocessing, two data sets were derived: one for classification containing the following variables: `hotel, lead_time, arrival_date_year, arrival_date_month, arrival_date_week_number, arrival_date_day_of_month, stays_in_weekend_nights, stays_in_week_nights, adults, children, babies, meal, country, market_segment, distribution_channel, is_repeated_guest, previous_cancellations, previous_bookings_not_canceled, reserved_room_type, assigned_room_type, booking_changes, deposit_type, days_in_waiting_list, customer_type, adr, required_car_parking_spaces, total_of_special_requests`, and one for regression, which is virtually identical to the data set used for the classification task, except for the `is_canceled` and `adr` variables. The `is_canceled` variable is the dependent variable in the classification task (is canceled, yes or no), which has been included in the regression data set as a feature, whereas the `adr` variable is not part of the regression data, but is instead the dependent variable which has to be predicted through regression. The dimensionality of both these classification and regression Hotel bookings data sets is 84679×28.

The Iris and Digits data sets were used by the algorithms for the dimension reduction and visualization task, whereas the Iris, Digits and Hotel bookings data were used for the classification and regression task. This choice was made, because the Iris data is a well known data set to demonstrate dimension reduction and visualization techniques, but it is quite low dimensional: 150×4, excluding the `Species` variable. For this reason, an additional data set was added to demonstrate the usefulness of the LLE and t-SNE algorithms, as the differences between the three methods (including PCA) should be negligible on the Iris data set, but should prove to be quite significant on the Digits data set, as the LLE and t-SNE methods tend to do better when the dimensionality grows than PCA. The Hotel bookings data was not used for the dimension reduction and visualization tasks, as all three algorithms showed bad preformance for this data, which would therefore not add much value to this report. On the other hand, this data was included to be used for the classification and regression tasks as it is a very different data set from the Iris and Digits data and suitable for classification and regression tasks. The definitions of each of the variables in the three data sets can be found in the appendix, see A.

2 Models

Before moving on to the data, a brief explanation of each of the models will be given in this section.

2.1 Ensemble Methods

Ensemble methods are techniques that combine several base models in order to produce one optimal predictive model. They usually produce solutions that are more accurate than a single model would be able to produce. For this report, the techniques RandomForest and XGBoost were used with both of them using as base model Decision Trees, in order to improve the performance. Before explaining the RandomForest and XGBoost algorithms, an explanation of a Decision Tree and its properties will follow.

Decision Trees

Decision trees are a type of model used for both classification and regression. They have a tree-representation with one root node splitting into two nodes and under certain conditions each node continues splitting up until no more splits satisfy the conditions. The nodes that do not split are called leafs or terminal nodes. Each splitting divides the samples falling into a node based on a condition of a predictor variable, for example, $X > 5$. Thereby, the data are split into many leafs, or sometimes called rectangles due to the 2D representation in figure 1.

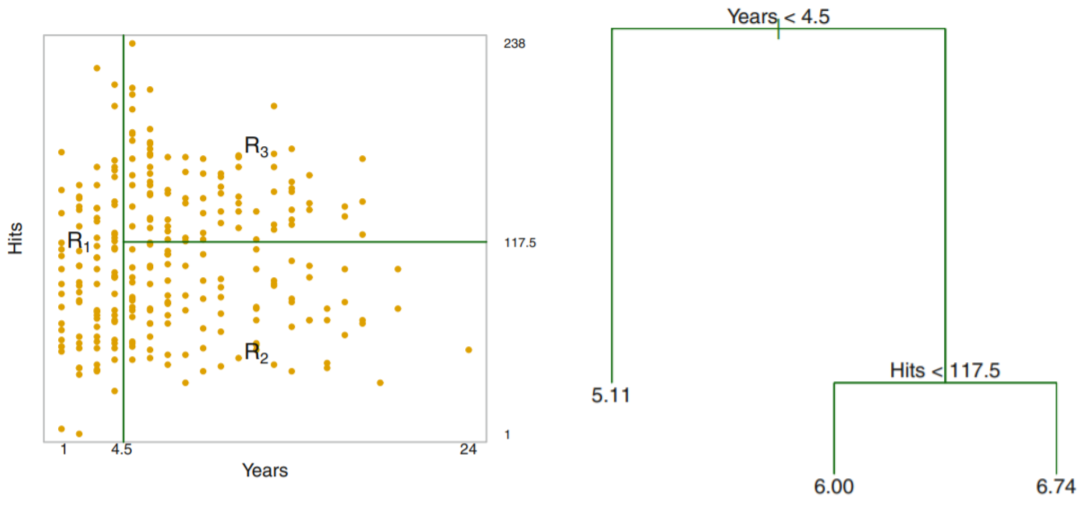


Figure 1: Partitioning of the 2-dimensional space and the corresponding tree respectively (James et al. 2013).

In general, the prediction of a Decision tree for an observation that falls into the region (R_i) is simply the average of the response values for the training observations (R_i). The objective is to find the regions $R_1, ..R_J$ which minimize the RSS given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_i})^2 \tag{1}$$

Since it is computationally infeasible to consider every possible partition of the feature space into J boxes, a greedy approach known as recursive binary splitting is used. Starting form the top of the tree, for every predictor $X_1..X_P$ the cut-point s_p that minimizes the RSS is found and the predictor that minimizes the RSS the most is selected with the respective cut-point. This process is repeated in order to split the data and minimize the RSS as much as possible. In the classification case, instead of the average of every region, the most commonly occurring class in the region is selected. The RSS is not to be used as the criterion in a classification setting, and therefore an alternative has to be used: the classification error rate, which is given by:

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{2}$$

With p_{mk} the proportion of training observations in the mth region that are from the kth class. However, two other measures are preferable to the classification error, namely *Gini* and $Cross - entropy$ or *deviance* (James et al. 2013) given by:

$$Gini: \quad G = \sum_{k=1}^{K} \hat{p}_m k)(1 - \hat{p}_m k)), \quad Entropy: \quad D = -\sum_{k=1}^{K} \hat{p}_m k) \log \hat{p}_m k) \tag{3}$$

Both of these approaches are more sensitive to node purity than is the classification error rate and also being differentiable are more amenable to numerical optimization.

Tree pruning

A fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data. Tree pruning is an easy way to avoid overfitting the data, by trimming off nodes in order to obtain a subtree. The optimal subtree can be obtained with a technique called Cost Complexity Pruning, a complicated and time consuming technique that was not used in this report. Instead restricting the maximum number of nodes or requiring a minimum number of samples for a split to occur achieves the same result.

Bagging

Another method used to avoid overfitting, as well as being the basis of RandomForest, is called Bagging. A simple Decision Tree suffers from high variance, different training data can lead to completely different splits. The idea of Bagging is to generates B different bootstrapped training data sets and build a separate prediction model using each training set. Averaging the resulting predictions will reduce the variance and increase the prediction accuracy of the model. The constructed trees from Bagging are grown deep and are not pruned, hence each individual tree has high variance, but low bias. For regression, averaging the prediction of each tree works well for predicting a quantitative outcome Y. For a qualitative outcome in classification, a majority vote is used, meaning the most common occurring class in all B trees is the overall prediction. On average 1/3 of the data is not included in a bagged tree. Instead of splitting the data into a train and test set, these unused data of each tree can be used to predict the response Y. The resulting OOB error is a valid estimate of the test error for the bagged model, since the outcome of each observation is predicted using only the trees that were trained without that observation.

RandomForest

A RandomForest is an improvement of Bagging, which is attained by slightly tweaking the trees in order to reduce the correlation between them. The algorithm is the same as Bagging, but now each time a split in a tree is to be executed, instead of selecting the best predictor out of the p predictors, a random sample of m predictors is chosen and the best predictor out of the m predictors is selected (James et al. 2013). A fresh sample of m predictors is taken at each split. Typically m is chosen as $m \approx \sqrt{p}$. This can lead to a further reduction in the variance, since the trees are now less correlated, whereas during Bagging trees seem more similar.

Boosting or Gradient Boosting

Boosting works in a similar way to Bagging, but now the trees are grown sequentially, meaning each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling. Instead, each tree is fit on a modified version of the original data set. The boosting approach learns slowly by adding every new tree to the fitted function. With the gradient boosting method the first tree is a usual decision tree, preferably a small tree, and every additional tree afterwards is fitted on the residuals of the current model instead of the outcome Y, as the response. For a basic boosting model, the model is trained by putting more weight on instances with misclassifications and high errors. Each tree is preferably small with only a few terminal nodes. By adding small trees fitted to the residuals, the model is improving where it does not perform well. The gradient boosting minimizes a loss function: it basically finds the partial derivative of the loss function and is used to find the direction in which to change the model parameters in order to reduce the error as much as possible in the next round of training. Each new tree that is added is multiplied by the shrinkage parameter λ to control the rate at which boosting learns. Boosting tends to suffer from overfitting the data as the number of trees added B is increasing.

XGBoost

XGBoost stands for Extreme Gradient Boosting, it improves the Gradient Boosting method by using more accurate approximations to find the best tree model. Specifically, it finds the partial derivative up to the second order term of the Taylor expansion of the loss function. The first order term is the residual and is what is used for the gradient boosting method. Additionally, it implements regularization terms ($L1$ and $L2$), which improves model generalization and helps to prevent overfitting.

2.2 Dimension reduction and data visualization techniques

PCA

The first technique used for dimension reduction and visualization is principal components analysis (PCA). According to Jolliffe and Cadima (2016), PCA is a dimensionality reduction technique, such that the interpretability is increased whilst minimizing information loss. PCA creates uncorrelated variables that successively maximize variance. In short, PCA first standardizes the data, in order to avoid having the scale of the variable influence the amount of variance that variable explains of the data. Subsequently, the covariance matrix of this scaled data is computed. From this covariance matrix, eigenvectors with corresponding eigenvalues are computed and ordered from large to small, to obtain the components that explain most of the variance in the data. Finally, the data is cast along the number of principal components the user wishes to keep, resulting in a lower dimensional solution.

LLE

Locally-Linear Embedding (LLE) is a nonlinear dimensionality reduction technique that computes low-dimensional, neighborhood preserving embeddings of high-dimensional data, and attempts to discover a nonlinear structure in high-dimensional data. In their 2000 paper, Saul and Roweis describe LLE in a very simple way: LLE first assigns K neighbours to to each data point X_i. Then, X_i is reconstructed using linear weights. Using these weights, the high dimensional X_i are mapped to low dimensional Y_i.

t-SNE

t-distributed stochastic neighbor embedding (t-SNE) is the final algorithm that should be discussed before moving on to the next section. The youngest of the 3 dimensionality reduction techniques, (van der Maaten, 2008), t-SNE derives it's name from the Student t-distribution. It first uses a Gaussian distribution to create a probability distribution for the relationships between the points in high-dimensional space. Then, using the Student's t-distribution (hence the name), it tries to recreate the probability distribution in low-dimensional space. The low dimensional embeddings are then optimized using gradient descent. From this it already seems obvious: t-SNE is inherently stochastic, contrary to the other two dimensionality reduction techniques.

3 Experimental setup

3.1 Data preprocessing

As stated in section 1, only the Hotel booking demands data required preprocessing, as the other data sets contained only numeric features without missing values.

The original Hotel bookings data from `https://www.kaggle.com/jessemostipak/hotel-booking-demand` consists of 32 columns and ~120000 rows. For the classification task the response variable was the column `is_canceled` showing whether or not a booking was cancelled. For the regression task the column `adr` (average daily rate) was used as the response variable. As a first step in the data preprocessing procedure, 4 variables were removed, the `agent` and `company` variables were removed due to a high number of missing or 'NaN' values, the `reservation_status_date` variable was removed because the date already exists, and `reservation_status` was removed due to it's similarity to the response variable `is_canceled`, which caused a predictive accuracy of 100%. Subsequently, around 30000 bookings were duplicates and were therefore dropped as well, such that the final data set contains about 85000 bookings (84679 to be precise) and 28 columns. Normally the categorical variables are nominal and should receive a nominal encoding, but this results in a data set consisting of 255 columns, which causes the algorithms to becomes much slower. Instead, the categorical variables received an ordinal encoding. Comparing the nominal and ordinal encoded data sets on a basic model with default values for parameters, the accuracy was the same, where the only difference was the importance of each feature and of course the run time. The extra time is significant later on during Grid Search cross validation. Additional data preprocessing and Feature Engineering was possible but since this is not the purpose of the report the current data preprosessing suffices.

3.2 Exploratory data analysis

Proposed by Andrews (1972), the Andrews plots allow for easy visualization of high-dimensional data. Grinshpun (2016)) notes that such plots "preserve the information about mean values, distance and dispersion and produce a large number of one-dimensional projections onto the vectors $(2-1/2, \sin t, \cos t, \ldots)(-\pi < t < \pi)$". Furthermore, he notes that "since the distances between Andrews (1972) plots are a linear reflection of the distances between data points, the plots (single lines in the figure) that are closer to each other correspond to the two points that are closer as well". This useful property can be used to visualize the various data sets, in order to get a sense of how easy the different classes are to separate using the three dimensional reduction techniques: PCA, LLE and t-SNE. As the Andrews plot mainly provides additional value for dimension reduction and visualization techniques, the Andrews plots are shown for the Iris and Digits data sets:

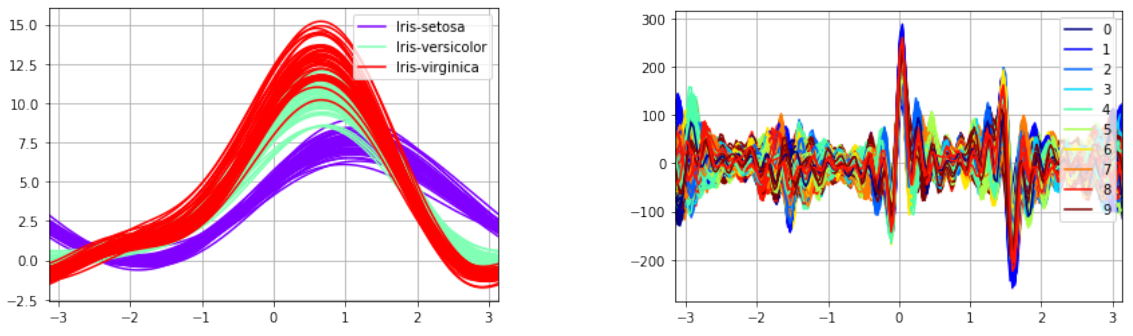


Figure 2: Andrews plots for the Iris data and Digit data respectively.

Above in figure 2 the Andrews plots for the different data sets are shown. The differences are quite striking: within the Iris data, the Setosa species is distinct from the Versicolor and Virginica species. In fact, these two species are also a bit difficult to distinguish from one another. The plot of the right in figure 2 represents the Andrews plots for the Digits data where the different digits seem to be overlapping, implying that the classes are very difficult to distinguish from one another.

For the classification and regression task, a different exploratory data analysis tool can be used: visualizing the distribution of the outcome variables of the data sets through a histogram. If the distribution of the classes (in the classification setting) is very skewed, such that one class contains most observations, a very high accuracy can be attained by simply using the most frequent class as a guess for any new observation on the test set. This will be discussed in section 3.3. Below, the histograms are shown:

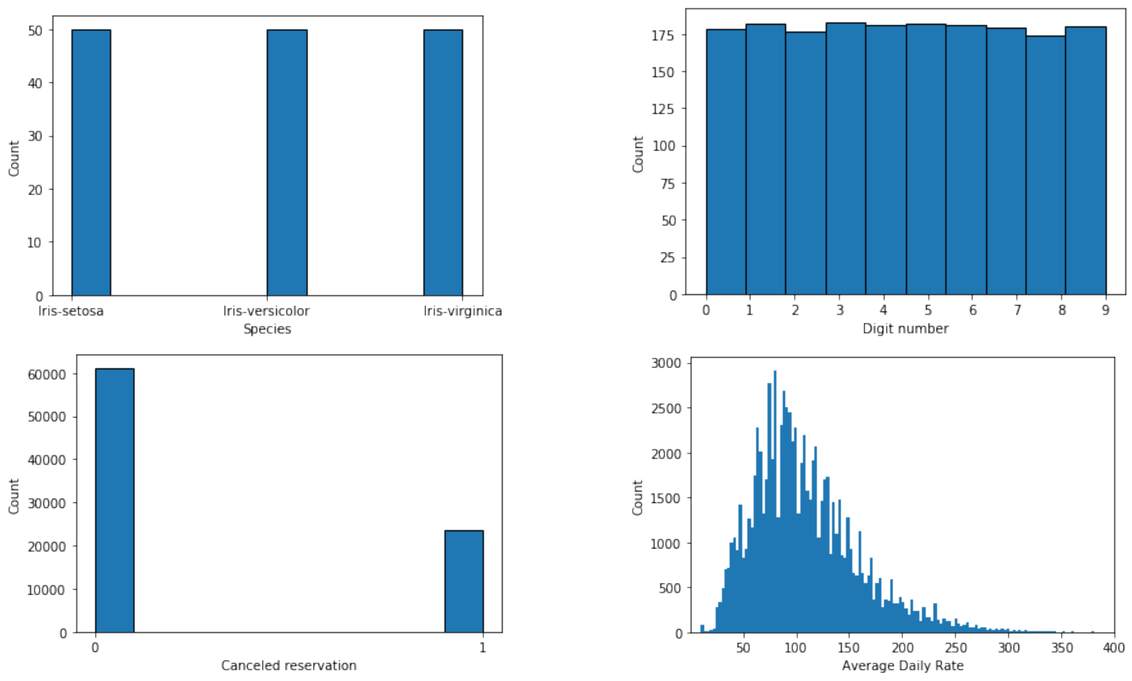


Figure 3: Histograms showing the distributions of the species of the Iris data, the digits of the Digits data and the cancellations and average daily rate for the Hotel bookings data respectively.

The classes for both the Iris and Digits data seem very evenly distributed, with about 50 observations per class for the Iris data and about 175 observations per class for the Digits data. On the other hand, as seen in the bottom left plot of figure 3, the classes for the Hotel bookings data are a lot less evenly distributed: the data contains about 60000 observations with non-canceled reservations and about 25000 observations where the reservation was canceled. Finally, the average daily rates has a shape similar to a Gamma distribution. There is a clear peak at around 100 bookings per day, which is not surprising as the median is approximately 99.

The information from the histograms of figure 3 can be used to construct baseline models, which can subsequently be compared with the complex models.

3.3 Base-line accuracy

Before moving on to the various algorithms, and trying to find the best parameters in order to achieve the best possible result on each of the tasks, it is useful to get a sense of the accuracy that can be achieved using very simple models: so-called baseline models. Various baseline models can be applied, depending on the task at hand. As baseline models have no real application in dimensionality reduction, nor in data visualization tasks, due to the different measure of "accuracy", this section focuses only on the classification and regression tasks. For each task, one baseline model is applied on the different data sets. Subsequently, the obtained results are then set as a "baseline", such that the more complex methods: RandomForest and XGBoost need to at the least preform better than this baseline value in order to be seen as a success.

Before moving on the the two baseline models, it should be noted that each data set was split into a train and test set, where the train set consisted of 75% of the cases and the test set 25% of the cases. To ensure replicability, a random seed of 2020 was used during the splitting of the data. This was done in order to be consistent with the train and test split used on the data for the classification and regression tasks.

For classification, one of the baseline models that is commonly applied is the zero rule classifier. This model simply takes the most common class from the train data and uses that class as the prediction for each object in the test data. From these predictions, the correct classification rate was then computed as:

$$\frac{1}{n}\sum_{i}^{n} I(\hat{y}_i = y_i), \tag{4}$$

where I is the indicator function. This model was applied on all 3 data sets giving the following correct classification rates:

- Iris data: 0.29
- Digits data: 0.10
- Hotel bookings data: 0.72

In the case of the Hotel books data, the bar is set quite high, as the correct classification rate for the zero rule of 0.72 is a pretty good baseline accuracy. This was to be expected, as the histogram in figure 3 shows a skewed distribution for the two classes. Therefore, taking the most frequently occurring class should give accurate results. On the other end of the spectrum: the zero rule achieved only a correct classification rate of 0.29 and 0.10 on the Iris and Digits data respectively, which was to be expected from their seemingly uniform distributions in figure 3. These baseline accuracies should not be too difficult to improve upon for the RandomForest and XGBoost methods.

The regression baseline model was chosen to be the median, which was computed over the train data and then used as a prediction for each of the test samples. Using these predicted values, the root mean squared error (RMSE) over the test set could be computed as:

$$\sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i = y)^2}{n}} \tag{5}$$

which resulted in a RMSE of 51.01 for the Hotel bookings data.

With these baseline accuracies in place, a good starting point for the rest of the analysis has been laid, where the more complex models should do better than the baseline models in order to be assessed as "useful".

3.4 Parameter tuning

This section will cover which parameters were tuned, and how they were tuned. Using a combination of literature-based recommended parameter values and optimal parameter values resulting from various tests, the final sets of parameters could then be used on the models in section 4.

3.4.1 Parameter tuning for classification and regression

Note that for the classification and regression tasks, all three data sets were used. However, the Iris and Digits data sets were only used for classification, whereas the Hotel bookings data set was used for both classification and regression. This choice was made because of the dependent variables in the data sets.

RandomForest parameters

In RandomForest, multiple parameters are available for tuning. In general, these parameters can be split into three categories:

- Tree-Specific Parameters, affecting each individual tree in the model.
- Ensemble Model Parameters, affecting the bagging operation in the model
- Miscellaneous Parameters, other parameters for overall functioning.

From the many available parameters only a select few were tuned, since certain parameters achieve the exact same results, such as pruning. This was done to keep the execution time of the algorithm reasonable. First, with a trial and error process the ranges of these parameters were selected and then with a grid search method the final optimal parameters were found.

The most important tree-specific parameter relates to which loss function to use. In the case of classification the selected loss functions are *Entropy* and *Gini*. Even though these loss functions are similar, both of them were selected for the grid search process. Concerning regression, MSE (mean squared error) and MAE (mean absolute error) are available, but because MAE is very time consuming and not properly optimised only MSE was used. Next, for pruning the max depth of the tree was not restricted, since in a RandomForest model, the potential overfitting of one tree is mitigated through the averaging process. Instead, for pruning the minimum samples split parameter was used for the minimum number of data points required in a node before the node is split, with a selected range of [2,8].

For ensemble model parameters, the maximum number of features m to randomly choose for each split seems to be the most important and the values of $m = [\sqrt{p}, p/3, p]$ were chosen. For regression tasks it is recommended to use $p/3$ random predictors (*The Elements of Statistical Learning p592*) The other important parameter is the number of trees to average over, and a value of 200 good enough for reducing the variance and increasing the accuracy. Therefore, more trees are unnecessary.

For other miscellaneous parameters a random state for bootstrapping and replicability of the results was used with a seed of 2020 and the number of cores of the CPU was selected for the number of jobs to run in parallel.

The grid search with cross validation on the training data for 5 folds was used with the parameters shown in table 1 below:

Table 1: Parameter values examined with the optimal values for each data set and response variable, p

Parameter	Range	Cl Hotel	Reg Hotel	Cl Iris	Cl Digits
Number of trees	100,200	200	200	100	200
Criterion	Gini,Entropy,MSE	Gini	MSE	Gini	Entropy
max. features	p,p\3,$\sqrt{p}$	$\sqrt{p}$	p/3	p	$\sqrt{p}$
min. samples split	2,3,5,8	3	2	2	2

XGBoost parameters

Similar to RandomForest, there are multiple parameters to tune and only a few were selected, also the same process for cross validation grid search was used.

For tree-specific parameters, only the maximum depth of a tree was chosen for having relatively simple trees added at each step and also acts as pruning. Increasing this value makes the model more complex and more likely to overfit. Again, values in [2,8] were selected.

For ensemble model parameters, similar to RandomForest, the maximum number of features m to randomly choose for each split can be used. In this case, a fraction of the p predictors is chosen with values in [0.5,1]. Additionally, the regularization parameters λ for $L2$ and α for $L1$

were chosen only as on and off: $\{0,1\}$. More values in between could be possible, but because of long execution times, only the two were selected. Similar to RandomForest, in order to reduce overfitting, only a sample of the training data was used at each boosting step and a range of [0.8,1] was used. For the learning rate η, the default value of 0.1 works well, but an optimum range of [0.05,0.3] was selected for further examination. Finally, the number of boosting trees, arguably the most important parameter, increases the accuracy of the model as it the number of trees increases. It will eventually converge or overfit the data, thus resulting in decrease of accuracy. Through trial and error, for up until 200 trees the model performance increases. In some occasions it continued to increase until 1000 trees. For time efficiency 200 trees were selected.

For other miscellaneous parameters the number of cores of the CPU was selected for the number of jobs to run in parallel.

The grid search with cross validation on the training data for 5 folds was used with the parameters shown in table 2. The number of trees and learning rate are related, so for efficiency and avoiding many combination of parameters a second grid search was used only tuning learning rate and the number of trees with the values for other parameters being as previously selected. The second grid was used only on Hotel bookings data set since the Iris and Digits data sets are much smaller and therefore require much less time. The final parameters are shown in table 2 for both regression and classification. After finding all the optimal parameters, the model is trained with a larger number of trees to capture any possible change in the performance. This was not possible before as it would need an incredible amount of time.

Table 2: Parameter values examined with the optimal values for each data set and response variable

Parameter	**Range**	**Cl Hotel**	**Reg Hotel**	**Cl Iris**	**Cl Digits**
Number of trees	200,500	500	500	200	200
$\boldsymbol{\eta}$	0.05,0.1,0.2,0.3	0.05	0.1	0.05	0.3
$\boldsymbol{\lambda}$	0,1	1	1	1	0
$\boldsymbol{\alpha}$	0,1	1	0	0	0
colsample bytree	0.7,1	1	0.7	0.7	1
subsample	0.8,1	0.8	1	0.8	0.8
max. depth	2,5,8	8	8	2	2

3.4.2 Parameter tuning for dimension reduction and visualization

Before moving on to the tuning of the parameters corresponding to the three dimension reduction techniques, it should be noted that the data was not splitted in a train and test set, unlike what was done for the classification and regression tasks. This was done because the t-SNE algorithm is not generalizable to holdout data, as t-SNE learns a non-parametric mapping. Therefore, the algorithm does not learn an explicit function that maps data from the input space to the map. In order to be consistent, PCA and LLE where therefore also implemented on the entire Iris and Digits data sets. As the aim is not to make predictions, but merely to reduce the dimensionality of the data and visualize the reduced data in a low-dimensional space, this is a valid approach.

Additionally, note that all of the proceeding figures were obtained using only the features of the Iris and Digits data sets, not the labels. The same principle holds for the figures shown in section 4. The labels were only used to color the samples in the 2-dimensional configurations, but played no role in the separation of the separation of the data points, as all three dimension reduction techniques are unsupervised.

Finally, all results are shown in a 2-dimensional configuration. The task is to do both dimension reduction and data visualization, and using two dimensions, two birds are hit with one stone. This is because data visualizing in more than 2 (or 3) dimensions is not appealing, as one would need to create many 2-dimensional plots.

PCA parameters

The PCA algorithm has no actual parameters that need tuning. If one considers the number of components to be used, then this can be seen as a ”tuning parameter”, but in the case of visualization, the number of components is either 2 or 3. Therefore, rather than tuning the parameters of the algorithm, this section on PCA parameters will consider the optimal amount of components to be used in the solution. Both dimensionality reduction and data visualization benefit from a small number of components (2 or 3 in the case of dimensionality reduction) that retains as much of the information in the data as possible when using all of the original variables in the data. To this end, a scree plot can be used, where either the eigenvalues or the variance accounted for per principal component (and thus per dimension) are plotted. Then, one can look for a so-called "elbow" in the scree plot, indicating that using an additional component in the solution will not explain significantly more of the variance in the data. To this end, for both data sets, scree plots were made. After applying the PCA procedure, the variance accounted for for dimension i is calculated as:

$$\text{VAF}_i = \frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}, \tag{6}$$

as $\sum_{i=1}^{n} \lambda_i$ is equal to the total amount of variance in the data. Also note that λ_i is the eigenvalue corresponding to the ith component. The resulting variance accounted for per component can be found in the scree plots below:

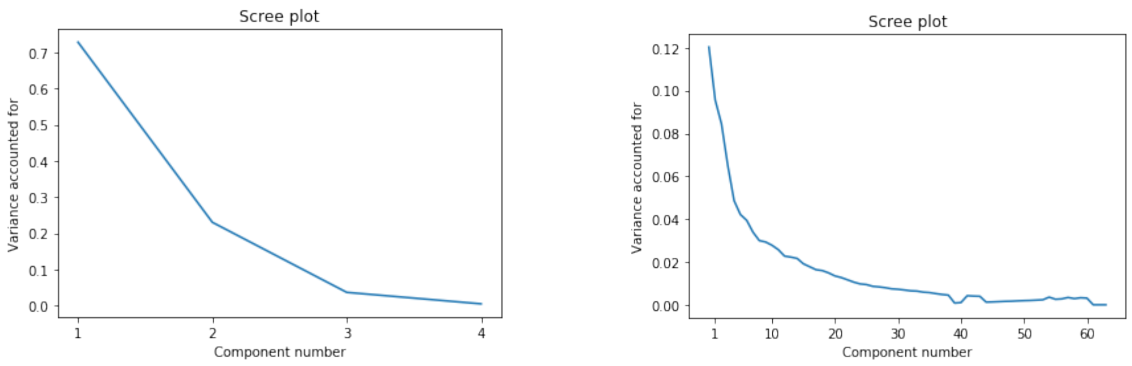


Figure 4: Scree plots for the Iris and Digits data respectively.

From figure 4 it can be observed that PCA is quite successful in reducing the dimensionality of the Iris data: using only the first component, we can account for over 70% of the variance in the data. Adding the second component as well therefore results in a very accurate overview of the structure of the data. PCA does not do as well on the Digits data: the first component only explains 12% of the variance in the data. As each subsequent component explains less variance, over 30 components are needed to explain at least 90% of the variance in the data.

LLE parameters

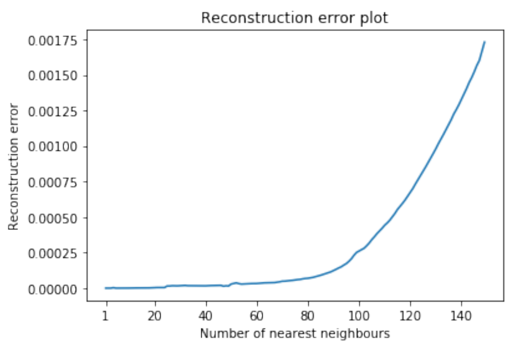
The LLE algorithm only has one parameter that really needs tuning: the number of neighbours K. Kouropteva et al., (2002) mention that "a large number of nearest neighbors causes smoothing or eliminating of small-scale structures in the manifold. In contrast, too small neighborhoods can falsely divide the continuous manifold into disjoint sub-manifolds". Therefore, like usual, a compromise needs to be made. Through experimenting, it was observed that using a small number of neighbours, whilst drastically reducing the execution time of the algorithm, resulted in somewhat "linear" structures in the clusters, which is not uncommonly seen in LLE solutions. Various papers have been written proposing a method to choose the optimal value for K, including Kouropteva et al (2002), which depend on the reconstruction error. Before moving on to the tuning of this parameter, one other parameter needs to be discussed, as this influences K.

Whilst not a "real" LLE parameter, the weight regularization parameter λ could play an important role in the algorithm. While weight regularization was not used (or mentioned) in the original paper of Saul and Roweis (2000), it is one of the possible tuning parameters of Scikit-learn's `LocallyLinearEmbedding()` function. The weight regularization parameter multiplies the trace of the local covariance matrix of the distances and is mentioned by Schuon et al. (2008), as well as by Saul and Roweis (2001) in one of their later articles. Before moving on to the use of this parameter, some additional context should be provided: Saul and Roweis (2003) note that the number of neighbours K should be less than the dimensionality d of the data in order to find the topology-preserving embedding. In fact, there seems to be some margin between K and d, but the size of this margin is not exactly known. However, they also note that "in the unusual case where $K > d$ (indicating that the original data is itself low dimensional)", each data point can be reconstructed perfectly from its neighbors, and the local reconstruction weights are no longer uniquely defined. In this case, some further regularization must be added to break the degeneracy". Furthermore, Schuon et al. (2008), note something similar, adding to the notion of Saul and Roweis (2003) that if $K > d$, the singularity problem arises. If one wants to mimic the original paper of Saul and Roweis (2000), the regularization parameter λ can be set to 0, as this implied no regularization is applied. The default value for λ used by the `LocallyLinearEmbedding()` algorithm is $\lambda = 0.001$. Karbauskaitė et al. (2010) give a broad overview on how λ exactly influences the LLE solution

With this knowledge, the choice for K should be discussed: as neither data set was high-dimensional, singularity problems arose when using $\lambda = 0$. Therefore, the decision was made to deviate from the original LLE approach and use the for the `LocallyLinearEmbedding()` function default value of $\lambda = 0.001$ to search for the optimal value of K using the approach proposed Kouropteva et al (2002). Furthermore, experiments with different values for λ have been executed as well, which will also be discussed here. The LLE algorithm was iterated over the data X using K nearest neighbours for $K = 1, \dots K_{max}$, where K_{max} is the maximum value of K permitted by the `LocallyLinearEmbedding()` function, e.g. $K_{max} < N$. Over all the iterations, the reconstruction error

$$\epsilon = \sum_{i=1}^{N} ||X_i - \sum_{j=1}^{N} W_{ij} X_{ij}||^2, \tag{7}$$

is computed, with W being the weight matrix. Those values of K that correspond to the minimum reconstruction error obtained are then saved, as multiple minima may exist. Before moving on to the rest of the algorithm, it is wise to look at the reconstruction errors obtained over the two data sets:



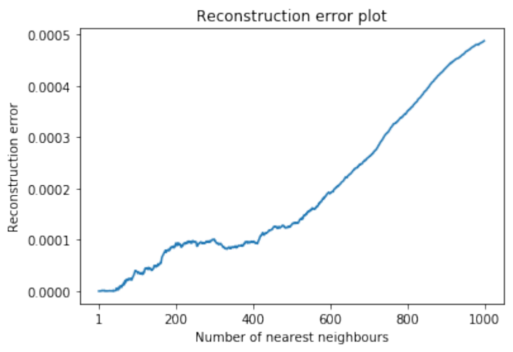


Figure 5: Reconstruction errors plotted against the number of nearest neighbours.

Note that the iterations for the Digits data were run up until $K = 1000$, as the value for ϵ kept increasing and as the number of neighbours goes up, the computation time does too. In all cases, the value of ϵ is smallest when $K = 1$. This might seem nice, as we immediately would have the optimal value of $K : K_{opt}$. However, observe what happens when $K = 1$ is used in the LLE algorithm on the two data sets:

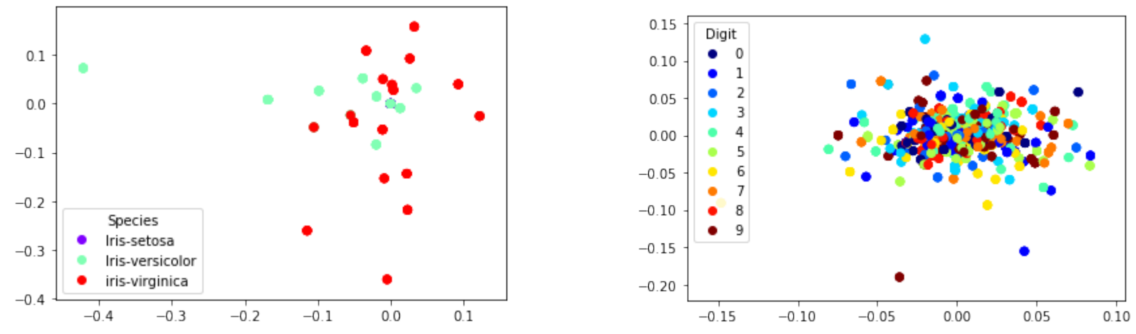


Figure 6: LLE results using $K = 1$.

Both these results are not desirable, as they do not to separate the clusters. Even on the very low dimensional Iris data, using LLE with $K = 1$ leads to bad results. As the `LocallyLinearEmbedding()` function only outputs the reconstruction error (7) as it's loss function, since this is the loss function for the LLE algorithm proposed by Saul and Roweis (2000), the optimization of the choice for K is not one that can be done numerically. Instead, a more subjective approach had to be taken where the value of K was chosen by eyeballing the plotted 2-dimensional LLE solution and looking which value of K separates the classes the best.

For the Iris data set, for $K \in \{1, \dots, 49\}$ the LLE solution depicts the typically linear shapes. And for $K \in \{50, \dots, 149\}$ the LLE solution loses the linear clusters and shows good class separation, with no real observable differences in class separation, as $\forall K \in \{50, \dots, 149\}$, the Versicolor and Virginica species are clustered close to each other, whereas the Setosa species is far removed from the other two.

The Digits data set represents quite different results when trying to tune K. For $K \in \{1, \dots, 55\}$ again linear structures are seen without good separation, except for the 0 digit. As K increases, the class separation improves, but even for values of K up until around 1000, only the digits 0,6,1 and 3 cluster together. Therefore, empirically, LLE seems to require a very high number of neighbours in order to show some separation between the digit classes. In fact, it was observed that as $K \to n - 1$, the 2-dimensional LLE plot shows the best digit class separation, $\forall K$.

Increasing the value of λ drastically changes the resulting solutions. As values of $\lambda \in (-\infty, 0]$ give the singular matrix problem, this parameter could only be increased. Through experimenting on the Digits data (as the LLE algorithm on the Iris data already achieved satisfactory results using $K \in (50, 149)$ and $\lambda = 0.001$) it turned out that the exact value of λ had an interesting effect on how K affected the solution: as λ increases, the useful value of K decreases. In this case, a "useful" value of K is one that renders a desirable solution, as stated before. The problem that happens if λ is increased, whilst having a value of K that is too high, is that the 2-dimensional LLE solution is a straight vertical line upon which all samples are projected. In the end, the solution obtained from the LLE algorithm with parameter combination $K = 350, \lambda = 1$ renders a solution that is of comparable quality to that of the LLE algorithm with parameter combination $K = 1796, \lambda = 0.001$. Both these results are shown in section 4.

All of the above experimentation was done using the "standard" LLE, as described by Saul and Roweis (2000). However, Scikit-learn's `LocallyLinearEmbedding()` function allows for other types of LLE, namely Hessian LLE and Modified LLE.

In the case of Hessian LLE, an additional parameter can be tuned: the Hessian tolerance. However, while experimenting, this tolerance does not have a large impact on the solution, and therefore it was kept at the default value of 0.0001. Furthermore, the impact of λ is also not as significant on Hessian LLE as it is on standard LLE. For values of K up until 25, the 2-dimensional solution is one where only a few random points are presented. Then, for $K \in [26, 100)$ the Hessian LLE algorithm is able to separate the digits 0 and 6 from the other digits, and for $K \in [100, 200)$ the LLE algorithm gradually separates the other digits classes. However, for $K \in [200, 1796]$, there appears to be diminishing returns in increasing K, as it appears to settle on some separation for the digit classes. Keeping computation time in mind, as a higher value of K increases the computation time, the choice of K for the Hessian LLE was 800, as this resulted in relatively good digit class separation, where increasing K did not improve the solution (by visually inspecting it). It should be noted that for some values of $K \in [200, 1796]$, the solution resulted in a straight horizontal line, on which all digit classes are projected, for example for $K = 600$.

Finally, the modified LLE algorithm should be discussed. The results obtained by the Modified LLE were actually very similar to those of both the standard- and Hessian LLE algorithms, in terms of digit class separation. Furthermore, just like for the Hessian LLE, both λ and the Modified tolerance parameter (the Modified LLE's counterpart to Hessian LLE's tolerance) had very little influence on the solution. Furthermore, the way in which K influenced the 2-dimensional solution was also very similar to the Hessian case above: very low values of K resulted in no class separation. By gradually increasing K, digit classes like 0, 6 and 4 get separated from the others, until there comes a point where increasing K further does nothing for the digit class separation but increases the computation time. For the same reasons as in the Hessian case, the value of K was set to 800 for the modified LLE algorithm.

In all cases, the `dense` method was chosen as the method to derive the eigenvalues. Even though the `arpack` method was also able to produce good results, it was found that the random state had too much impact on the solution. As the `dense` method requires no random state for replicability (as it is not a random process), and produced equally good results, this was the chosen method to obtain eigenvalues.

t-SNE parameters

t-SNE minimizes the Kullback-Leibler divergence between the joint probabilities p_{ij} in the high dimensional space and the joint probabilities q_{ij} in the low-dimensional space (Van der Maaten and Hinton, 2008), as stated in section 1. Kullback-Leibler divergence indicates how much information is lost when an approximation is chosen. However, when choosing the parameters for the model, comparing Kullback-Leibler divergence values amongst models with different parameter values and choosing the set of parameters corresponding to the lowest Kullback-Leibler divergence is not a valid approach. This is because when, for example, increasing the perplexity parameter $Perp$, smaller distances in absolute terms are obtained. which results subsequent in smaller Kullback-Leibler divergence values. This then implies that one can keep increasing the perplexity, in order to get a lower Kullback-Leibler divergence. As there unfortunately is no other error value or goodness of fit statistic available, a more subjective criteria was chosen to tune the parameters: visual inspection of the clusters derived by the t-SNE algorithm in 2-dimensional space, similar to the parameter tuning method used for the LLE parameters.

As stated before, t-SNE is inherently stochastic. Therefore, a random seed of 2020 was used throughout all t-SNE experiments to ensure replicability.

There are 4 main parameters that need tuning:

- the perplexity: $Perp$
- the number of iterations: T
- the early exaggeration factor: α
- the learning rate: η

Van der Maaten and Hinton (2008) note that "perplexity $Perp$ can be interpreted as a smooth measure of the effective number of neighbors". Furthermore, they note that a common value for the perplexity parameter is between 5 and 50, although the performance of t-SNE is fairly robust to changes in this value. By increasing the perplexity, more nearest neighbors are used, and as such the algorithm is less sensitive to small structures in the data. t-SNE emphasizes large dissimilarities in the data, and projects the large dissimilarities as even larger distances, this is the reason for using Student's t-distribution as it has wider tails than the Gaussian distribution.

Even though the paper of van der Maaten and Hinton (2008) gives no recommendation for the number of iterations to be used, one of their examples uses a maximum of 1000 iterations. This also happens to be the default option of sklearn's `TSNE()` function. However, generally, increasing the amounts of iterations used in the optimization procedure will not lead to a worse solution. The downside by choosing a very high value for T is that it becomes computationally infeasible at some point. Furthermore, more iterations need not necessarily improve the solution: if a minimum is attained, no matter how many iterations there are left in the procedure, the solution will not improve further. Therefore, the amount of iterations was gradually increased during the experimenting phase, and fixed once the 2-dimensional solution did not improve further by increasing the number of iterations. Note that there are problems associated with having a too small amount of iterations: the optimization procedure might not reach a minimum by the time the final iteration is reached. This is one of the reasons why Scikit-learn's `TSNE()` function requires at least 250 iterations.

The third tuning parameter is the early exaggeration parameter α. This is simply a factor that is used to multiply all the p_{ij}'s during the optimization phase. α controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. The Scikit-learn documentation notes that the exact value of $alpha$ is not "very critical".

Finally, the last parameter that needs tuning is the learning rate η. According to the Scikit-learn documentation, the learning rate requires some balancing, as a too high value of η can result in a 2-dimensional solution where the data may look like "a ball with any point approximately equidistant from its nearest neighbours", whereas for a too low learning rate, most points may look compressed in a dense cloud with few outliers. Furthermore, the documentation states that commonly, for t-SNE we have $\eta \in [10, 1000]$. The learning rate has a direct influence on the convergence of the gradient descent optimization method that t-SNE uses. Kobak and Berens (2019) mention that the default value of the learning rate is 200 for many t-SNE implementations (including Scikit-learn's `TSNE()` function) which is not enough for large data sets and can lead to poor convergence and/or convergence to a sub-optimal local minimum. Experimenting will show whether this value is sufficient on the Iris and Digits data or not.

Just like in the Scikit-learn's `LocallyLinearEmbedding()` function, Scikit-learn's `TSNE()` function also allows for deviation of the basic method. The 'basic' method, denoted by `exact` in the algorithm, is the original t-SNE algorithm described by der Maaten and Hinton (2008). However, a few years later, van der Maaten, (2013) proposed a different method called 'Barnes-hut'. The Barnes-hut algorithm is shown to run in $\mathcal{O}(N \log N)$ rather than $\mathcal{O}(N^2)$ time. Furthermore, van der Maaten, (2013) shows that the the 1-nearest neighbor errors do not differ between the two methods. The time difference is maybe not such a big factor when using SNE on the Iris data set, but it becomes very noticeable on "bigger" data sets like the Digit data set.

One last thing that should be discussed before moving on to the parameter tuning is the initialization of the t-SNE algorithm. Scikit-learn's `TSNE()` function allows for two different initialization options: random and PCA. Random initialization is the default, and is used by van der Maaten and Hinton (2008) in their work, and depends on a Gaussian distribution for the position of clusters. The PCA initialization allows for easy replicability without reliance on random seeds, as seen earlier in this report.

The t-SNE algorithm was first implemented on the Iris data. As the Iris data is relatively easy to separate compared to the Digits data (as judged from the Andrew plots in figure 1), the t-SNE algorithm should provide good class separation without too much parameter tuning. The first experiments were done using the original `exact` t-SNE method with random initialization. What immediately became clear is that using the default $\eta = 200$ did not lead to good results, not matter the value of $Perp$, as the points in 2-dimensional space seemed more or less equidistant with no clusters appearing. Fixing $Perp$ to the upper limit of the recommended values by van der Maaten and Hinton (2008): 50, and fixed all other parameters to their defaults ($T = 1000, \alpha = 12$), whilst varying η, resulted in very good class separation, where decreasing η led to less ball shaped clusters, which are therefore more in line with the clusters obtained by PCA and LLE. In other words, lower values of η allowed the t-SNE algorithm to better capture the local structures of the data. Note that too low values of η: 0.01 and lower resulted in strange linear structures, which do not represent the local structure well. Furthermore, increasing T did not improve the solution, indicating that a minimum was attained. PCA initialization was also tried on the Iris data, but this also resulted in a bad representation of the local structure, where the clusters were too circular. The Barnes-hut implementation was not experimented with on the Iris data, as the exact method was fast enough, due to the small size of the Iris data. Since the draw of the Barnes-hut approach is the fact that it is computationally less expensive than the regular approach, testing this on the Iris data was redundant. As the default settings (except for $Perp$ and η) resulted in a good class separation, similar to what the other algorithms attained, no more experimenting was done on the Iris data, having as the final set of parameter values: $\{Perp = 50, T = 1000, \alpha = 12, \eta = 5\}$.

Just like in the case of the Iris data, the parameter tuning on the Digits data used the exact method at first, together with random initialization and default parameter values, except for $Perp$, which was again set to 50. The solution that resulted from this gave very good digit class separation, moreso than the PCA and LLE approaches on the same data. However, the clusters became quite compact (not much space within a cluster), but with a lot of space inbetween certain clusters. The equidistant points problem observed on the Iris data for $\eta = 200$ did not occur on the Digits data. This could be due to the dimensionality of the Digits data: 1797×64 (since this is an unsupervised method, the labels are left out), versus 150×4 of the Iris data. As stated by Kobak and Berens (2019): large data can handle (or even requires) larger values for η. It quickly became apparent why the Barnes-hut implementation gained traction, enough so to become the default option in Scikit-learn's `TSNE()` function: the t-SNE algorithm is slow on large data sets. For this reason, some experimentation was done using the Barnes-hut approach with exactly the same parameter values used by the exact approach: $\{Perp = 50, T = 1000, \alpha = 12, \eta = 200\}$. This resulted in almost identical 2-dimensional configurations. As good as the digit class separation was for both the exact and Barnes-hut approaches, some digit classes were subdivided into multiple clusters: often 1 large cluster, and one or more very small clusters of that particular digit class. Therefore, more experimenting was done in order to try to limit the amount of clusters to 1 per digit class, whilst retaining separation between the digit class clusters. All else equal, increasing the $Perp$ value did make the clusters less compact, thereby disposing of the characteristic ball shapes that occurred for lower perplexity values. However, there were still too many data points outside of their clusters, which is what the additional experimentation tried to solve. One the other hand, by gradually reducing the value of $Perp$, the within cluster distances becomes smaller, until at single digit values of $Perp$, the within cluster distances increase again. It should be noted that by increasing T, whilst choosing a low value for $Perp$, the configurations will look very similar to those with higher values for $Perp$, but with lower values of T (e.g. the default of $T = 1000$). This shows that the t-SNE algorithm tries to further move the clusters apart as T grows. This makes it seem like the local structure disappears, but in reality the between cluster spaces grow, thereby giving the illusion of within cluster shrinkage. PCA initialization was attempted as well, but to no avail. Changing either α or η did not help either in terms of putting the "loose" observations into their corresponding digit class clusters. In the end, the following set of parameter values was used: $\{Perp = 10, T = 1000, \alpha = 12, \eta = 20\}$. The exact values of α and η did not matter too much, especially in the case of α (as shown by Kobak and Berens (2019)), but $Perp$ was set low (still within the range of $[5, 50]$, as recommended by van der Maaten and Hinton (2008)) to show local structure in the 2-dimensional configuration.

4 Results

4.1 Classification and regression

Iris

Initially, on the Iris data, a basic RandomForest model with the default parameters was able to achieve an accuracy of 86.8% on the test data. After hyper-parameter tuning with grid search cross validation, a new model (the tuned model) with new parameters was was able to achieve an accuracy of 89.5% on the test data. There was an increase of $\sim 3\%$, which is not really significant, due to the very small data set: the difference is 1 observation not getting misclassified. For the XGBoost model, the same procedure was followed where the basic model accuracy was 89.5% and tuned model accuracy was 86.8%. Again, the difference is due to one misclassified observation. Figure 7 shows the results of both models with parameter tuning for an increasing number of trees.

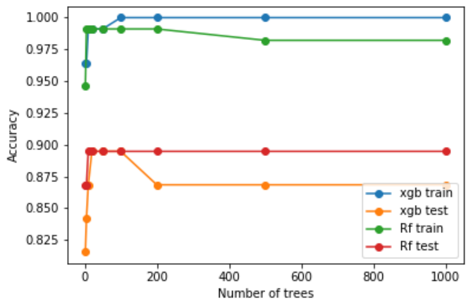


Figure 7: Accuracy of the RandomForest and XGBoost model on the train and test data of the Iris data.

It is worth mentioning that due to the small data set, the random effect used to split the data into training set and test set was important. With some seeds the accuracy could reach 100% for both models, regardless of the parameter choices made.

From figure 7, it is clear that for both models, the train accuracy is much higher than the test accuracy. This is most probably due to the small data set. Furthermore, the accuracy of RandomForest for the test data is much more stable than for the XGBoost algorithm, as expected. It could be argued that the decrease of accuracy for the XGBoost model after 100 trees is due to overfitting, although such a conclusion is not very clear due to the small data set. Additionally, upon investigating the misclassified observations, it appears that both models have misclassified exactly the same observations. The importance of each feature is presented in figure 8 with the exact values represented in table 3 below:

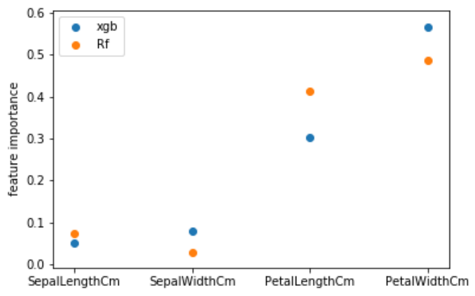


Figure 8: Feature importance comparison between the two models for the Iris data.

Table 3: Feature importance comparison between the two models for the Iris data

Feature	RandomForest	XGBoost
PetalWidthCm	0.485589	0.564728
PetalLengthCm	0.412650	0.303336
SepalWidthCm	0.073444	0.051603
SepalLengthCm	0.028316	0.080333

The importance of the features is similar for both models, showing that both models value the same features for the classification. This explains why both models have misclassified the same observations.

Digits

Similar to the procedure for the Iris data set, a basic RandomForest model with default parameters was implemented and managed to achieve an accuracy 97.3% on the test data for the Digits datatset. After hyper-parameter tuning with grid search cross validation, a new model (the tuned model) with new parameters was achieved an accuracy of 98% on the test data. The increase of $\sim 0.7\%$ is not very significant. In this case, the difference is 3 observations not getting misclassified. For the XGBoost model, the same procedure was followed with a basic model accuracy of 96.2% and a tuned model accuracy of 97.3%. The increase in accuracy is now due to 5 misclassified observations in the basic model being correctly classified by the tuned model. Figure 9 below shows the results of both models with parameter tuning for an increasing number of trees:

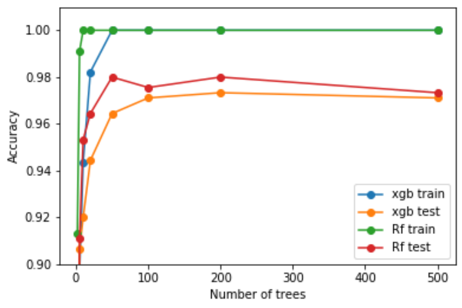


Figure 9: Accuracy of the RandomForest and XGBoost models on the train and test data of the Digits data.

Form figure 9, it is clear that for both models the training accuracy reaches 100%, and is much higher than the test accuracy. This is most probably due to the small data set. Furthermore, the accuracy of the RandomForest algorithm on the test data reached its maximum accuracy very fast: ($< 50 trees$), and the small fluctuations are due to 1-2 misclassifications related to random effects. In comparison, the XGBoost algorithm reached its maximum value more gradual at $n\_tree = 200$. Again, it could be argued that the decrease of the accuracy for the XGBoost model after further increasing the amount of trees beyond 200 trees is due to overfitting. However, such a conclusion is not very clear due to the "small" data set. Additionally, upon investigation the misclassified observations, it appears that both models have misclassified 12 and 13 observations respectively with 7 of those observations being the same. The importance of each feature is presented in figure 10 and the exact values of the top 5 most important features are found in table 4 below:

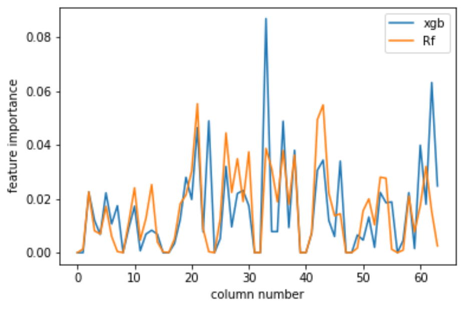


Figure 10: Feature importance comparison between the two models for the Digits data.

Table 4: Top 5 feature importance comparison between the two models for the Digits data

Digit	**RandomForest**	**Digit**	**XGBoost**
21	0.055304	**33**	0.086944
43	0.054875	**62**	0.063220
42	0.049448	**23**	0.048929
26	0.044460	**36**	0.048836
33	0.038731	**21**	0.046425

Looking at table 4, the top 5 most prominent features are different for the two models, where digits 21 and 33 appear for both models. From figure 10 the overall distribution of the feature importance is similar and it is noticeable that the digits between 20-40 are the most important. This likely due to the position in an image as these digits would occupy the center of an image.

Hotel bookings

Classification

The basic RandomForest model with default parameters achieved an accuracy of 84.3% on the test data. After hyper-parameter tuning, with grid search cross validation, a new model with new parameters (the tuned model) achieved an accuracy of 84.5% on the test data. Again, a small increase was observed: $\sim 0.2\%$. The difference is 30-40 observation not getting misclassified by the tuned model. For the XGBoost model, the same procedure was followed where the basic model achieved an accuracy of 84.2% and the tuned model achieved an accuracy of 85.1%. This is a rather significant increase since 5% of the misclassified observations are now correctly classified. This is close to 100-150 additional correctly classified observations. Figure 11 shows the results of both models with parameter tuning for an increasing number of trees.

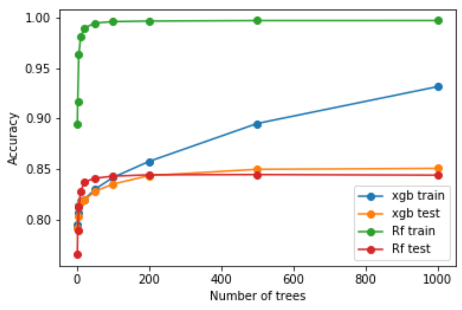


Figure 11: Accuracy of the RandomForest and XGBoost models on the train and test data of the Hotel bookings data in the classification setting.

Form figure 11 above, it is clear that for the RandomForest algorithm, the training accuracy reached close to 100% and is much higher than the test accuracy. In this case, there is no sign of overfitting. Furthermore, the accuracy of the RandomForest algorithm on the test data reached its maximum value relatively quickly: ($< 50 trees$), and then stabilized as is to be expected from Bagging. In comparison, the XGBoost model started with a lower accuracy than the RandomForest model. However, its accuracy gradually increased and even passed the accuracy of the RandomForest model around $n\_tree = 200$. The accuracy of the XGBoost model kept increasing up until 1000 trees. It is possible that the optimal value for the number of trees is above 1000 and that it would keep increasing before overfitting appears. Additionally, upon investigation the misclassified observations, it appeared that the RandomForest and XGBoost models have misclassified 3296 and 3154 observations respectively, with the majority of the misclassifications; 2531 observations, being the same. Below, the importance of each feature is presented in figure 12 with the exact values of the top 5 most important features given in table 5:

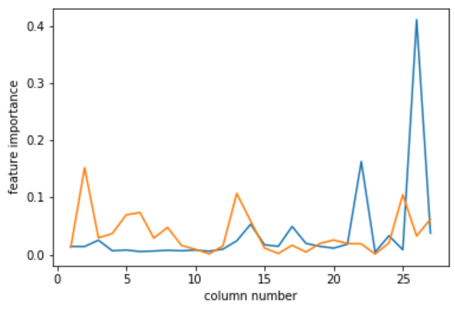


Figure 12: Feature importance comparison between the two models for the Hotel bookings data in the classification setting.

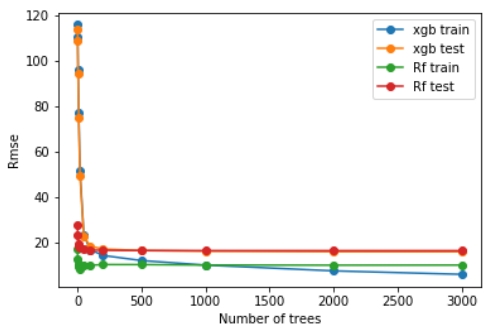
Table 5: Top 5 feature importance comparison between the two models for the Hotel bookings data in the classification setting

Feature	**RandomForest**	**Feature**	**XGBoost**
lead time	0.152	**parking spaces**	0.411
country	0.107	**deposit type**	0.163
adr	0.105	**market segment**	0.053
day of month	0.074	**previous cancellations**	0.049
week number	0.070	**special requests**	0.038

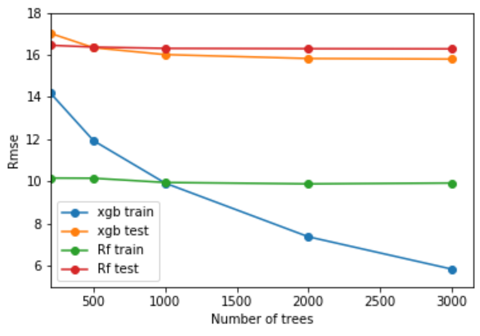
From figure 12 and table 5 above, it appears that the important features between the RandomForest and XGBoost models are unrelated, with each model having different features in the top 5 most important features. The XGBoost model depends strongly on whether or not there is a parking space for cancelling a reservation. When using a nominal encoding for the data, the important features turned out to be more similar but the overall accuracy was the same.

Regression

The basic RandomForest model with the default parameters achieved a RMSE of 16.99 on the test data. After hyper-parameter tuning with grid search cross validation, the tuned model was obtained with new parameters, and achieved a RMSE of 16.45 on the test data. There is an important decrease of the RMSE between the two models of 0.45. For the XGBoost model, the same procedure was followed where the basic model achieved a RMSE of 19.12 and the tuned model achieved a RMSE of 16.34. This is a decrease of the RMSE that is mostly due to the high number of trees: 500. The figure 13a shows the results of both models with parameter tuning for an increasing number of trees up until 3000, since XGBoost performance was still increasing and figure 13b focuses more closely on the accuracies.



(a)



(b)

Figure 13: RMSE of the RandomForest and XGBoost models for the train and test data of the Hotel bookings data in the regression setting. The right plot is focused on a higher number of trees

Form figure 13, it is clear that for the RandomForest model on the train and test data, a low RMSE is attained using only a small number of decision trees. This is because of the high complexity of each tree with no pruning. The test RMSE of the RandomForest converges to its lowest value very quickly at $\sim$ 100 trees, and then stabilizes, as is expected from Bagging. As for the XGBoost model, both on the train and test data, the RMSE of a small number of trees is a lot higher, with a rapid decrease in the RMSE for the first 200 trees, but then the decrease slows down up until 3000 trees. The RMSE on the test data continues decreasing until it reaches a value of 15.80. No overfitting was observed regardless of using 3000 trees. Below, the importance of each feature is presented in figure 14 with the the exact values of the top 5 most important features presented in table 6:

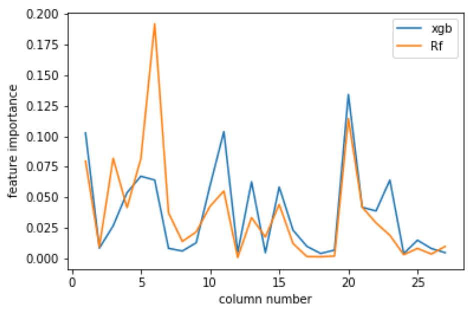


Figure 14: Feature importance comparison between the two models for Hotel bookings data in the regression setting.

Table 6: Top 5 feature importance comparison between the two models for the Hotel bookings data in the regression setting

Feature	**RandomForest**	**Feature**	**XGBoost**
week number	0.192	**room type**	0.134
room type	0.115	**children**	0.104
lead time	0.0820	**hotel**	0.103
month	0.082	**month**	0.067
hotel	0.080	**week number**	0.064

From figure 14, the overall distribution of the feature importance is similar between the RandomForest and XGBoost models, where it can be seen that the distributions have very similar peaks. Looking at table 6, the top 5 most important features of the two models are almost the same, with 'week number', 'room type', 'month' and 'hotel features' appearing in the top 5 for both models. Regardless of the fact that the two models place a high importance on almost the exact same features, the XGBoost algorithm seems to optimise the feature importance better and thereby achieves a slightly lower RMSE.

To summarize, it turns out that both the RandomForest and the XGBoost preform better than the baseline models on all data sets. Therefore, the use of the two models on the different data sets can be seen as a success. Furthermore, it turns out that on the Iris test data, the RandomForest algorithm is more accurate. On the Digits test data, the RandomForest algorithm is again more accurate. In the final classification setting, on the Hotel bookings test data, the XGBoost algorithm is the most accurate of the two models. In the only regression setting, on the Hotel bookings test data, the XGBoost algorithm achieves a lower RMSE than the RandomForest algorithm, where a lower RMSE means that the model preforms better.

4.2 Dimension reduction and visualization

As the goal of this task is both to reduce dimensionality and visualize it, all configuration are shown in 2D. This is because 2-dimensional (or 3-dimensional) configurations are easy to visualize. Furthermore, it seems convention in the literature to present the results for dimension reduction in 2D, rather than 3D. However, if the reader is interested, 3-dimensional configurations were also created, which can be found in the appendix B. Note that these configurations were only produced for the standard and exact methods in case of the LLE and t-SNE respectively.

PCA

Even though the PCA algorithm required no parameter tuning as seen in section 3.4.2, a different amount of components needed to be used for each data set in order to explain a substantial amount of variance in the data. In the case of the Iris data, only 2 components were needed. However, on the Digits data, the "elbow" of the scree plot was located around 5 components, see figure 2. Unfortunately, visualizing 5 dimensions is not an easy feat. Therefore, as visualization is part of the task, only the first 2 components will be used in order to adhere to this task, even if not enough of the variance is explained by this number of components.

Running PCA, using 2 components, resulted in the following figure below, whereby the labels were only used to color the points:

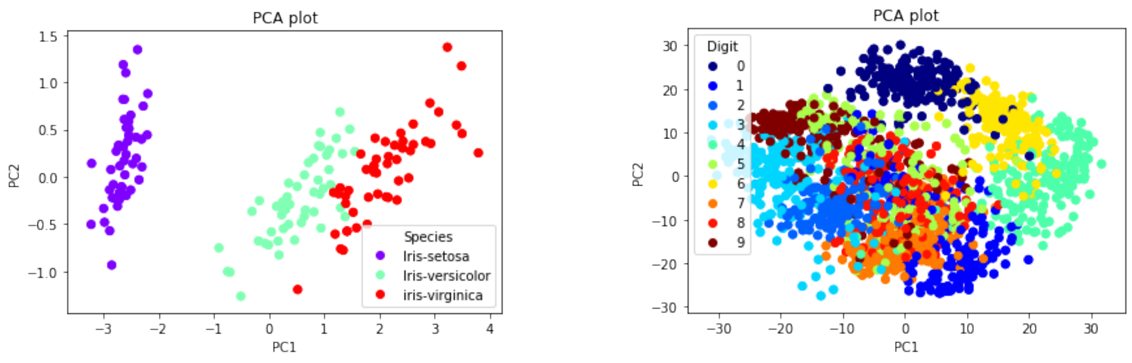
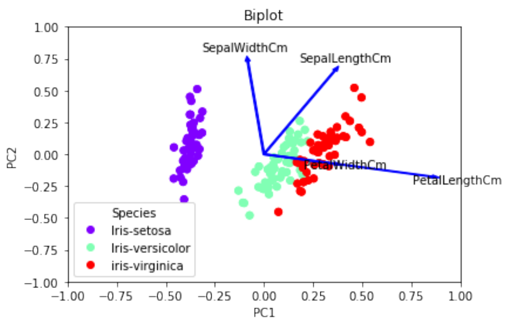


Figure 15: PCA plots for the Iris and Digits data respectively using the first 2 principal components.

As mentioned before, the PCA gives a good solution on the Iris data, no matter whether 2 or 3 components are used, as the third component does not explain much additional variance, see figure 4. On the other hand, using PCA on the Digits data with 2 components results in figures where the different classes are not as separated as in the Iris case, see figure 15. Only the classes for digits 0, 4 and 6 are more or less separated from the others. This was to be expected, as using 2 components only accounts for about 22% of the variance in the data. Even though the data is not high-dimensional, $n < p$, PCA is not as successful on the Digit data.

One advantage of using PCA is that it can be combined with a biplot. A PCA biplot takes the first two principal components as it's axes and allows for the data points to be plotted depending on the corresponding values to either of the components, similar to 15. The difference is that the biplot also shows the original variables from the data X which are shown as vectors in the biplot, see Gower et al., (2011). The use of this is that it becomes easy to give some interpretation to the principal components, as well as to see which classes have high/low values on which variables. The biplots are shown below:



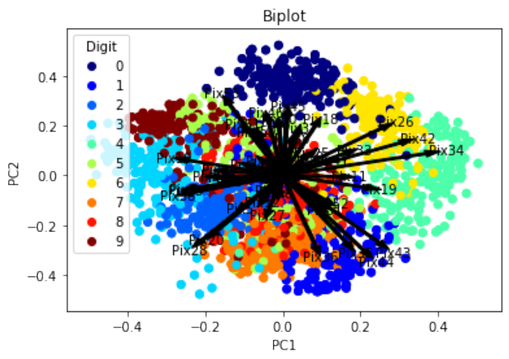


Figure 16: PCA biplots for the Iris and Digit data respectively.

The biplot in figure 16 for the Iris data provides some useful insights: the Setosa species have relatively short and narrow petals, whereas the other two species have longer petals. The Versicolor and Virginica species also tend to have a longer sepal, but not a wider sepal, as this vector points almost straight up, indicating that none of the species have a distinctly long sepal. Finally, the petal length and width variables correlate heavily, as the two vectors point in the same direction.

Concerning the Digits data, the biplot does not give as much useful information. As there are 64 vectors present in the biplot, it is pretty hard to read. Furthermore, each vector represents one pixel. So, while, for example, the digits 5 and 6 have a high value on the 34th and 42nd pixels, the information does not tell most people a lot, unless they know where those pixels are located in the 8×8 image.

LLE

In section 3.4.2, various options in terms of both which LLE method (standard, Hessian or Modified) and which parameter values (for K and λ) were discussed. This section will show the results obtained by the parameter and method combinations that were deemed best by the method described in section 3.4.2. The first result is that of the the LLE algorithm applied on the Iris data:

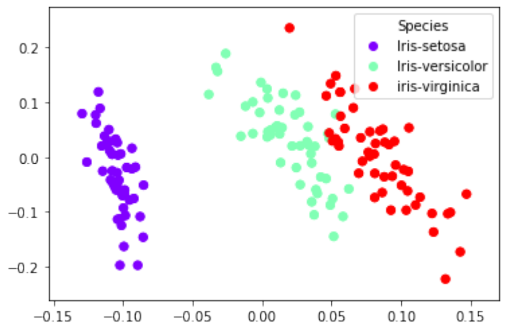


Figure 17: LLE plot for the Iris data.

The figure above almost seems to be a reflected version of figure 4, which represents the PCA solution on the Iris data. Using the standard LLE approach, with $K = 80$ and $\lambda = 0.001$ good digit class separation was achieved, where, just like in the PCA setting the Setosa species is easily separable from the other two, but the Versicolor and Virginica species are more difficult to separate. These two species are probably so similar, due to the fact that they have very similar values for the petal length and width, as derived in the PCA results.

Using LLE on the Digits data set resulted in a lot more figures, as seen below:

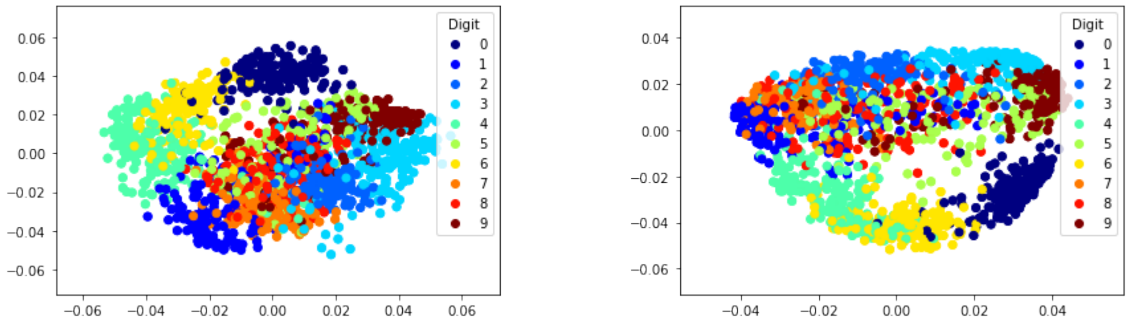


Figure 18: LLE plots for the Digits data using $\{K = 1796, \lambda = 0.001\}$ and $\{K = 350, \lambda = 1\}$ respectively.

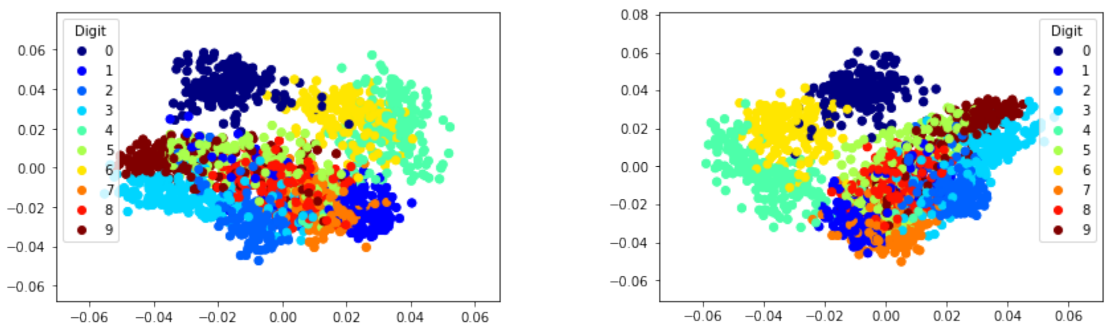


Figure 19: Hessian- and Modified LLE plots for the Digits data respectively.

The best LLE results were obtained using values parameter combinations $\{K = 1796, \lambda = 0.001\}$ and $\{K = 350, \lambda = 1\}$ on the standard method, see figure 18 and parameter combinations $\{K = 800, \lambda = 0.001\}$ and $\{K = 800, \lambda = 0.001\}$ using the Hessian and Modified methods in figure 19.

Even though the four configurations are visually a bit different, they all tell the same story: digit classes 0, 4 and 6 are easily seperable from the other digits, just like in the PCA results. In fact, LLE does not seem to do a much better job in separating the other classes than PCA does, as there is still more or less one big cluster with the other digit classes. The fact that these three classes keep reoccurring as separable from the others, even though it was not directly observed in the Andrews plot in figure 2, is that these three digits look very distinct, which leads to very different pixel values compared to the other 7 digits. Elaborating on this point, representing the configuration in a 2-dimensional space allows for visualising which digits are very distinct from other digits. Clusters that are opposite of each other are most dissimilar in their pixel values, whereas digit clusters that overlap are very similar and were not distinguished by the LLE algorithm. Therefore, for example, digits 3 and 4 are not at all similar, due to them lying on opposite ends on the configurations in all plots in figures 18 and 19, whilst digits 5 and 8 are very similar due to their overlap in all plots in figures 18 and 19.

t-SNE

The final dimensionality reduction and visualization method is the t-SNE method. Unlike the LLE approach, the results obtained from the t-SNE algorithm using only two combinations of parameters will be shown here (for the Digits data), as these two combinations resulted in the best 2-dimensional configurations.

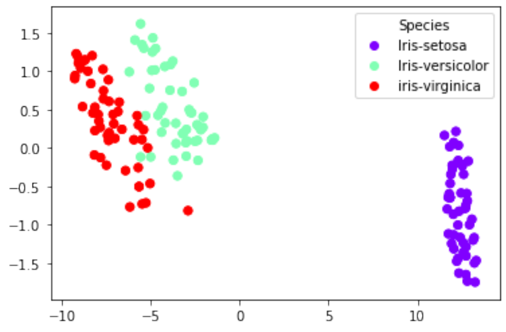


Figure 20: t-SNE plot for the Digits data.

The 2-dimensional t-SNE results for the Iris data are very similar to those of the PCA and LLE algorithms, see figures 15 and 17. What is different is that the 2-dimensional configuration is reflected, such that the Setosa species is found on the right side of figure 20 instead of on the left side like in figures 15 and 17. It should be noted that this does not change anything about the solution, as it can be freely reflected.

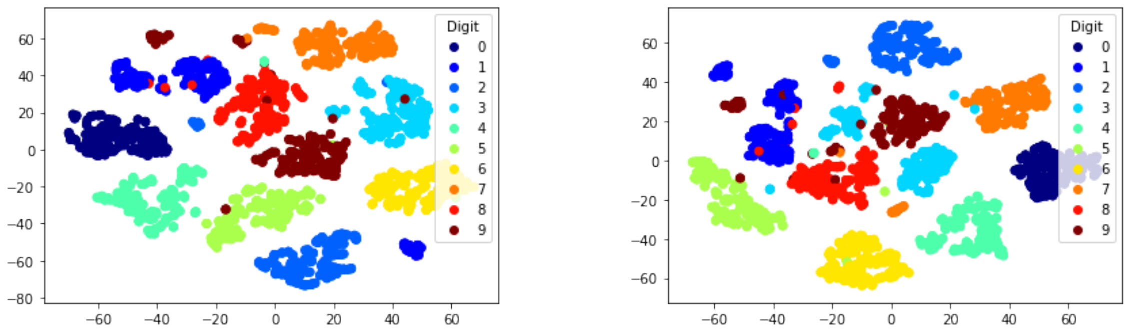


Figure 21: t-SNE plots for the Digits data using the exact and Barnes-hut methods respectively.

What is interesting about figure 21 above is that both the exact method and the Barnes-hut method show very good separation of the 10 Digit classes, better than both PCA and LLE. However, note that in both the left and right plot of figure 21 the small sub-clusters are present that were discussed in section 3.4.2. In fact, there can be quite some distance between the clusters that belong to the same digit: see the clusters belonging to digit 9 in the left plot and also in the right plot. In both plots, the small cluster is far removed from the big cluster. In fact, for this particular digit, the same thing is observed in the PCA and LLE plots for the digit data, see figures 15, 18 and 19, except that these ”loose” points are represented by single samples instead of by small clusters. Maybe the person who wrote down some of these 9's has very sloppy handwriting, such that the computer cannot recognize the badly drawn samples of digit 9 as belonging to the other 9's, due to the fact that the pixel values are too different. One final remarkable thing about figure 21 is the difference between the two plots: the digit class clusters are not at the same distance from one another. Observe that for the exact method, the cluster of 7's is located next to the clusters with 8's and 3's. However, for the Barnes hut method, the cluster of 7's is not located next to the 8's. As such, when comparing both plots in figure 21 to the other figures of the Digits data, it appears that by judging the relative positions of the clusters, the left plot of figure 21, based on the exact approach, is more similar to the other figures than the right plot of figure 21 based on the Barnes-hut approach.

5 Conclusion

This report aimed to tackle two different sets of tasks, where one set of tasks was classification and regression and the other was dimensionality reduction and data visualization. To solve these tasks, 5 different algorithms: RandomForest, XGBoost, PCA, LLE and t-SNE were implemented on 3 different data sets: the Iris data, the Digits data and the Hotel bookings data. Before this was done, The data was preprocessed and inspected, from which baseline models were constructed. These models provided a baseline accuracy which was then set as a threshold for which the 5 complex algorithms needed to improve upon in order to be considered "useful". Before running the complex models on the data and comparing the results with the baseline accuracies, the parameters of the various models needed to be tuned. In the classification and regression methods, this was done through a grid search, whereas for the dimension reduction and data visualization tasks, this was done by visually inspecting the 2-dimensional configurations produced by the algorithms. With the tuned parameters, the models were run once again and the achieved results were evaluated, by themselves, against other models and against the baseline accuracies. Both the RandomForest and XGBoost algorithms produced better results than achieved using the baseline models, on all data sets. Furthermore, no clear "winner" could be chosen between the RandomForest and XGBoost models, as each model showed superior performance in at least one case. On the low dimensional Iris data set, all three dimension reduction algorithms were able to separate the three Iris species. The added value of t-SNE became apparent when using the Digits data, as it was much better capable of separating the digit classes than either PCA or LLE.

References

D. F. Andrews. Plots of high dimensional data. *Biometrics*, 28(1):125–136, 1972.

J. Gower, S. Gardner-Lubbe, and N. le Roux. Understanding biplots. 2011.

V. Grinshpun. Application of andrew's plots to visualization of multidimensional data. *International Journal of Environmental Science Education*, 11(17):10539–10551, 2016.

G. James, D. Witten, T. Hastie, and R. Tibshirani. An introduction to statistical learning with applications in r. 2013.

I. T. Jolliffe and J. Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences*, 374(2065), 2016.

R. Karbauskaitė, G. Dzemyda, and V. Marcinkevičius. Dependence of locally linear embedding on the regularization parameter. *TOP 18*, page 354–376, 2010.

D. Kobak and P. Berens. The art of using t-sne for single-cell transcriptomics. *Nature*, 10(5416), 2019.

O. Kouropteva, O. Okun, and M. Pietikainen. Selection of the optimal parameter value for the locally linear embedding algorithm. *Proc. of 2002 Int. Conf. on Fuzzy Systems and Knowledge Discovery*, page 359–363, 2002.

L. K. Saul and S. T. Roweis. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

L. K. Saul and S. T. Roweis. An introduction to locally linear embedding. *Journal of Machine Learning Research*, 7, 2001.

L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

S. Schuon, M. urković, K. Diepold, and S. M. Jürgen Scheuerle. Truly incremental locally linear embedding. 2008.

L. van der Maaten. Barnes-hut-sne. 2013. URL `https://arxiv.org/abs/1301.3342`. Accessed on 02.12.2020.

L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Appendix

A Tables

Table 7: Variable definitions Iris data.

Variable	**Definition**
`SepalLengthCm`	Length of the sepal (in cm)
`SepalWidthCm`	Width of the sepal (in cm)
`PetalLengthCm`	Length of the petal (in cm)
`PetalWidthCm`	Width of the petal (in cm)
`Species`	Species name

Table 8: Variable definitions Digits data.

Variable	**Definition**
`1,...,64`	Pixel number
`Digit`	Digit number

Table 9: Variable definitions Hotel bookings data.

Variable	**Definition**
`hotel`	Hotel (Resort Hotel or City Hotel)
`lead_time`	Number of days that elapsed between the entering date of the booking into the PMS and the arrival date
`arrival_date_year`	Year of arrival date
`arrival_date_month`	Month of arrival date
`arrival_date_week_number`	Week number of year for arrival date
`arrival_date_day_of_month`	Day of arrival date
`stays_in_weekend_nights`	Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
`stays_in_week_nights`	Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
`adults`	Number of adults
`children`	Number of children
`babies`	Number of babies
`meal`	Type of meal booked
`country`	Country of origin
`market_segment`	Market segment designation
`distribution_channel`	Booking distribution channel
`is_repeated_guest`	Value indicating if the booking name was from a repeated guest or not
`previous_cancellations`	Number of previous bookings that were cancelled by the customer prior to the current booking
`previous_bookings_not_canceled`	Number of previous bookings not cancelled by the customer prior to the current booking
`reserved_room_type`	Code of room type reserved
`assigned_room_type`	Code for the type of room assigned to the booking
`booking_changes`	Number of changes/amendments made to the booking from the moment the booking was entered on the PMS
`deposit_type`	Indication on if the customer made a deposit to guarantee the booking
`days_in_waiting_list`	Number of days the booking was in the waiting list before it was confirmed to the customer
`customer_type`	Type of booking
`required_car_parking_spaces`	Number of car parking spaces required by the customer
`total_of_special_requests`	Number of special requests made by the customer (e.g. twin bed or high floor)
`is_canceled`	Value indicating if the booking was canceled or not
`adr`	Average Daily Rate as defined by dividing the sum of all lodging transactions by the total number of staying nights

B 3D figures

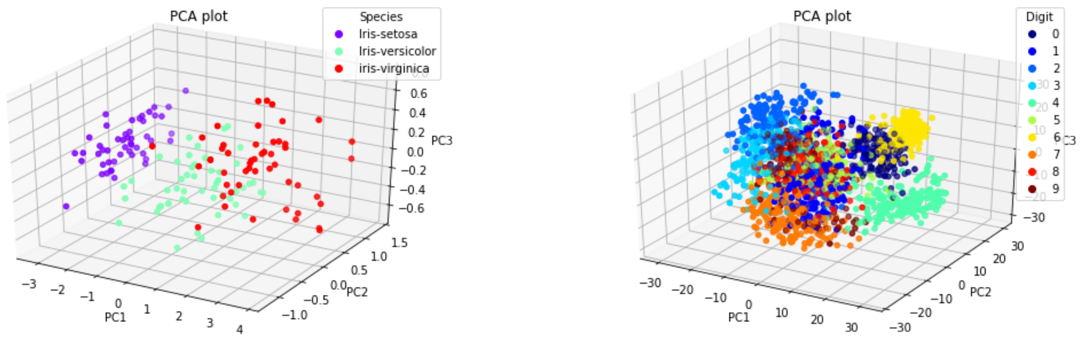


Figure 22: 3D PCA plots for the Iris and Digits data respectively.

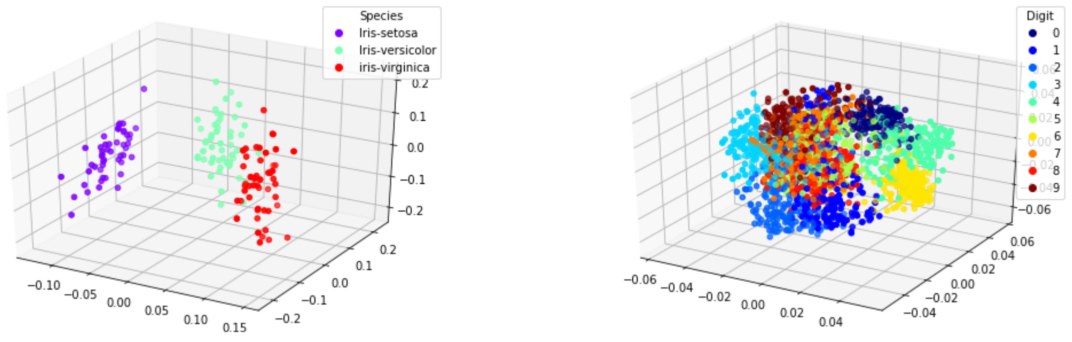


Figure 23: 3D LLE plots for the Iris and Digits data respectively.

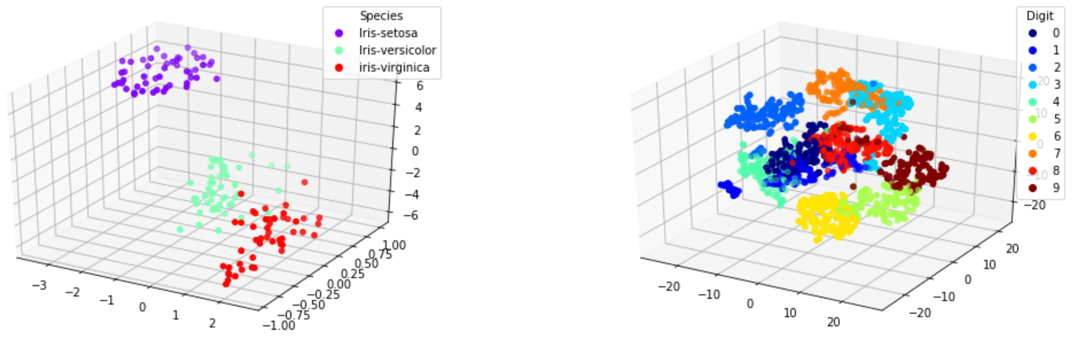


Figure 24: 3D t-SNE plots for the Iris and Digits data respectively.