DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Mapping of open-answers using machine learning

VIKING BJÖRK FRISTRÖM

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Mapping of open-answers using machine learning

VIKING BJÖRK FRISTRÖM

Degree Projects in Mathematical Statistics (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2018
Supervisor at Nepa AB: Helena Ryman
Supervisor at KTH: Timo Koski
Examiner at KTH: Timo Koski

TRITA-SCI-GRU 2018:173
MAT-E 2018:30

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

This thesis investigates if a model can be created to map misspelled answers from open-ended questions to a finite set of brands. The data used for the paper comes from the company *Nepa* that uses open-questions to measure brand-awareness and consists of misspelled answers and brands to be mapped to. A data structure called match candidate was created and consists of a misspelled answer and brand that it potentially be mapped to. Features for the match candidates were engineered and based on the edited distances, posterior probability and common misspellings among other. Multiple machine learning models were tested for classifying the match candidates as positive if the mapping was correct and negative otherwise. The model was tested in two scenarios, one when the answers in the training and testing data came from the same questions and secondly when they came from different ones. Among the classifiers tested, the random forest model performed best in terms of PPV as well as sensitivity. The resulting mapping identified on average 92% of the misspelled answers and map then with 98% accuracy in the first scenario. While in the second scenario 70% of the answers were identified with 95% confidence in the mapping on average.

Detta examensarbete undersöker huruvida en model kan skapas för att kartlägga felstavade svar till öppna frågor till ett finit set av företagsnamn. Datan till denna uppsats kommer ifrån företaget *Nepa* som annvänder öppna frågor för att mäta märkesmedvetenhet. Denna data består av öppna svar samt företagsnamn som dessa kan matchas till. En datastruktur skapades som kallas för match candidate och består av ett felstavat svar samt ett företagsnamn som svaret kan matchas med. Attribut skapades till match candidate och bygger bland annat på sträng likhet, aposteriorisannolikhet samt vanliga fel stavningar med mera. Ett flertal maskininlrningsmodeller testades för att klassifiera match candidates som korrekt om och endast om svaret och företagsnamnet matchade och inkorrekt annars. Modellen testades i två olika scenarior. I det första kom datan som modellen tränade och testade på ifrån samma frågor. I det andra scenariot var det olika frågor som tränings och test data byggdes på. Av de makininlärnigs modeller som testades så presterade radom forest modellen bäst i avsende på PPV och sensitivity. Den resulterande kartläggning lyckades i genomsnitt identifiera 92% av alla felstavade svar och matchades i 98% till korrekt företagsnamn i det första scenariot. I det andra scenariot identifiera 70% av alla felstavade svar och matchades i 95% till korrekt företagsnamn i genomsnitt.

Acknowledgements

I would like to thank Prof. Timo Koski who supervised this project and provided feedback on the first draft.

I would also like to thank *Nepa* and the data science team for the support and for giving me the opportunity to conduct my thesis work at the company. I would especially like to acknowledge my supervisor at *Nepa* Helena Rydman for her continuous support and feedback throughout this project.

Stockholm, Maj 2018

Viking Björk Friström

Contents

Nomenclature

ANN Artificial Neural Network

Error Sum of Squares SSE

FN False Negative

FP False Positive

KNN K-Nearest Neighbours

ML Machine Learning

PPV Positive Predictive Value

ReLU Rectified Linear Unit

ROC Receiver Operating Characteristic

SVM Support Vector Machine

TN True Negative

TP True Positive

1 Introduction

Companies spend substantial amounts of capital each year to cut through the market noise and make their brands well known to the public or a specific target audience. The consumers' ability to recall a brand from memory has a positive correlation with brand market performance. This ability is referred to as brand awareness. Different metrics can be used to measure brand awareness, where brand recall is one of the more powerful ones. That is consumers ability to recall a brand when given a product category. When measuring brand recall it is vital that the consumers are not lead to any brand. Therefore, questionnaires have to be designed using questions without answering alternatives in order to measure brand recall.

Consequently, along with correctly spelled brands, these questionnaires will yield misspelled and nonsense answers. In order to make any meaningful analysis of the data the misspelled answers have to be mapped to a correctly spelled brand. The company *Nepa* gathers millions of answers like these each year in order to make inference about consumer behavior. Today the mapping of misspelled answers is done manually at *Nepas* India office with the help of in-house tools providing suggestions based on the Damerau-Levenshtein distance between the misspelled answer and brands from *Nepas* database. Automating this process would free up a lot of man-hours, which would reduce cost and eliminate the human error from manual correction. The problem presented consists of predicting a qualitative response variable, where Nepa posses a great amount of training in the form of previous manually mapped answers. A natural approach to solving the task is then too apply a machine learning algorithms to map these answers.

Previously *Nepa* has had limited success with applying traditional machine learning methods for text analysis as these are in general designed not for the purpose of classifying single words but rather use embeddings based on whole sentences. Another problem is the relatively large solution space, as every brand in *Nepas* database represents a class. However, a more novel approach using a binary classifier, that can tell if two words belong to the same class has shown much better potential. This classifier uses features based on how similar two different words, such as a QWERTY weighted Levenshtein distance measure.

1.1 Aim and scope

The goal of this master thesis is to create and investigate a model using a binary classifier to map misspelled answers. The task amounts to

1. Create a data structure for the answers that can be binary classified.
2. Investigate different machine learning models for classifying the data.

3. Map the answers based on the classification.

Classifiers to be considered are:

- Support Vector Machines
- K-nearest neighbours
- Neural Networks
- Classification Tree
- Random Forest
- Logistic Regression

2 Theory

In this section the theory for which this thesis is based upon is presented.

2.1 Brand awareness

Brand awareness is associated with several positive traits for a brand such as whether a brand is considered when a consumer is buying a product [Hoyer and Brown, 1990], and even positive market performance [Kim et al., 2003]. Brand awareness is defined as to what degree consumers are able to recall or recognize a brand. Brand awareness can further be divided into brand recognition and brand recall depending on the level of awareness. Brand recognition is simply if a consumer is able to recognize a brand when shown for example a logo or product. Brand recall, on the other hand, is the costumers' ability to recall a brand without any priming. This indicates a higher level of brand awareness as this implies that the customer has heard of the brand priory since the chance of a brand being chosen randomly without prior knowledge is very low [Huang and Sarigöllü, 2014].

2.2 String similarity measures

A string refers to a data type which consists of a sequence of characters. Finding similarities between strings is an important subproblem of text similarity which is used in a large range of applications such as text information retrieval, document clustering, machine translation, essay scoring, topic detection etc. [Vijaymeena and Kavitha, 2016]. There are plentiful metrics for string similarity, the ones used in this paper are described below.

2.2.1 Damerau-Levenshtein distance

Damerau-Levenshtein distance is a metric for measuring the edited distance between two strings. The Damerau-Levenshtein distance between two strings, a and b is defined as the number of operations needed to transform a into b. Viable operations for changing a string are:

- Insertion: ran → rain
- Deletion: rain → ran
- Substitution: cat → hat
- Permutation: watre → water

Levenshtein showed in his paper that most misspellings are due to one of the three first operations. [Levenshtein, 1966]. Damerau later added the permutation. For each operation the Damerau-Levenshtein distance is increased by one.[Dang and Phan, 2010]

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) \quad \text{if} \quad min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \quad \text{otherwise} \\ lev_{a,b}(i-1,j-1) + \mathbb{1}_{a_i \neq b_j} \end{cases} \end{cases} \tag{1}$$

The Damerau-Levenshtein distance is an extension of the Levenshtein distance which excludes permutation from the legal operations. This distance in its original form is based upon hand written misspellings. In recent years adaptations of the Damerau-Levenshtein distance based on how the strings were created has been proposed. In the case of computer written text, one can scale the weights depending the physical distance between keys on the keyboard [Pirinen and Lindén, 2010]

2.2.2 Szymkiewicz-Simpson coefficient

Szymkiewicz-Simpson coefficient or Overlap coefficient is a measure of the overlap between two sets and defined in equation (2) [Vijaymeena and Kavitha, 2016].

$$Overlap(X,Y) = \frac{|X \cap Y|}{min(|X|,|Y|)} \tag{2}$$

If X is a subset of Y or vice-versa the overlap coefficient will be equal to 1. On the other hand, if X and Y do not share any elements the overlap coefficient will be 0. The overlap coefficient captures some information that is lost in the Damerau-Levenshtein distance, as the later has a lower limit bound by the difference in string size.

2.3 QWERTY keyboard layout

The keyboard layout commonly referred to as QWERTY, named after first six letters on the top row, already recognized in 1971 by the *International Standards Organization* (ISO) as the standard keyboard layout [Noyes, 1983]. Local variations of the QWERTY layout exists to accommodate regional special characters, such as the Swedish versions that includes the letters å, ä and ö. See fig 1.

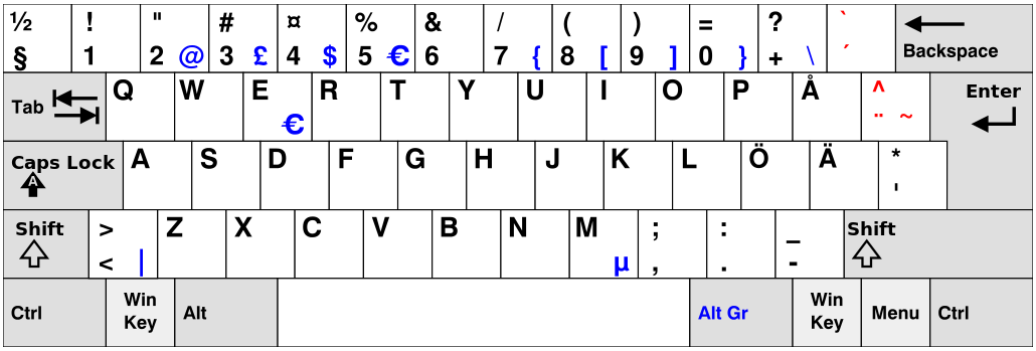

Figure 1: The Swedish QWERTY keyboard layout

The keyboard layout correlates to potential misspellings, as the probability for substitution will increase the shorter the physical distance between the keys are [Dickinson et al., 2012, section 2.2]. For example, on a QWERTY keyboard the probability of substituting d for s is greater then substituting k for s, since s and d are located next to each other on a QWERTY keyboard while s and k are not.

2.4 Classifiers

Machine learning is an area of study within computer science that regard algorithms that improve automatically through experience [Mitchell, 1997, Preface]. In general, these algorithms work by training on a set of data so that later, using a set of input variables or predictors, they can predict an output variable or dependent variable. Machine learning algorithms are categorized depending on whether the output variable is qualitative or quantitative and whether the training data is labeled or not. That is if the training data has a depended variable or not. The problem stated in this paper deals with labeled training data and qualitative dependent variable. Thus making it classifications problem of supervised learning.

The error of a machine learning model comes from three sources. They are bias, variance, and the irreducible error. The irreducible error comes from natural variability in the system and cannot be removed by the model. Thus, most machine learning problems come down to simultaneously trying to reduce the bias and variance. Bias comes from simplifications and assumptions made in the model while variance comes from overfitting the model to a specific type of data. It easy to create a model which have a low value of either variance or bias while having a high value for the other. Thus minimizing the error becomes a question of finding a trade-off between the to such that the total error is minimized [James et al., 2014]

2.4.1 Support Vector Machines

Support Vector Machines (SVMs) constitute a group of classifiers that predict data on the basis of a separating hyperplane in the data space. A hyperplane in p dimensions is given by equation (3) [Hastie et al., 2001, p. 417]. SVMs where the first statistical learning method with a theoretical justification for text classification [Joachims, 2001, p. 74]. Thus making it a highly relevant learning method for this paper.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0 \tag{3}$$

SVMs are binary classifiers that make predictions about observations based on the observations positions relative to the computed hyperplane in accordance with equation (4)

$$sign(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p) = \begin{cases} +1, & if \quad \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p > 0 \\ -1, & else \end{cases} \tag{4}$$

If a dataset can be separated by a hyperplane without any error, in general, exists numerous hyperplanes that fulfill this criterion. In order to maximize the certainty of the classifications, the SVM will pick the hyperplane yielding the largest margin M, where M is defined as the euclidean distance between the hyperplane and the observations closest to it. Maximizing M will minimize the least certain classification since the confidence of a classification increases with its distance from the hyperplane [Joachims, 2001, p. 37]. However, in most cases such hyperplane does not exist. To solve this a soft margin can instead be used. A soft margin classifier allows for a small subset of observations to exist on the wrong side of the margin, and even on the wrong side of the separating hyperplane. This can be achieved by assigning slack variables $\epsilon_1, ..., \epsilon_n$ to each observation that allows individual observations to be on the wrong side of the margin or the hyperplane. C is a non-negative tuning parameter that decides the tolerance for the soft margin classifier. The hyperplane can then be obtained by solving equation (5) through (9)

$$\max_{\beta_0,\beta_1,...,\beta_p,\epsilon_1,...,\epsilon_n} M \tag{5}$$

$$subject\ to\ \sum_{j=1}^{p} \beta_j^2 = 1, \tag{6}$$

$$y_i(\beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + ... + \beta_p x_{i,p}) \geq M(1 - \epsilon_i), \tag{7}$$

$$\epsilon_i \geq 0, \tag{8}$$

$$\sum_{i=1}^{n} \epsilon_i \leq C. \tag{9}$$

The hyperplane created by equation (3) will by linear and thus the classes need to be separable in a linear space in order for the SVM to perform well. In scenarios when this is not the case, the SVM may transform the features into a different feature space where the classes are linearly separable. Let $\phi(\mathbf{X})$ be the function that transforms the inputs $\mathbf{X}$, the hyperplane may then be described as:

$$\beta_0 + \phi(\mathbf{X})^T \boldsymbol{\beta} = 0 \tag{10}$$

It can be shown that $\phi(\mathbf{X})$ is only used is only used though inner products, so the exact transformation ϕ is not needed. All the is required to solve equation (10) is the kernel function given by equation (11). The kernel function calculates the inner product in the new feature space.

$$K(x, x') = \langle \phi(x), \phi(x)' \rangle \tag{11}$$

The optimal form of the kernel will depend on the given data set and how it needs to be transformed. Other popular choices of kernel functions for SVM beyond a linear kernel are for example a radial kernel function given by:

$$K(x, x') = exp(-\gamma ||x - x'||^2) \tag{12}$$

Or having a polynomial kernel of degree d described by equation (13).

$$K(x, x') = (1 + \langle x, x' \rangle)^d \tag{13}$$

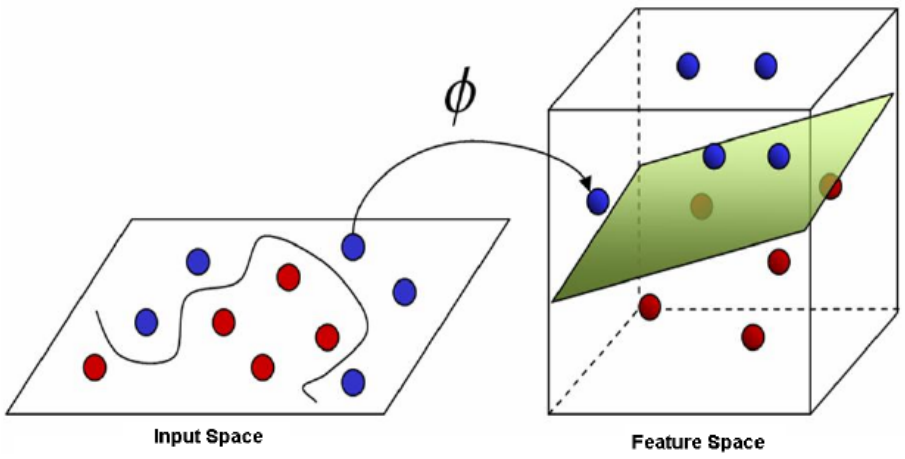

Figure 2: Visualization transformation of feature space using ϕ

The characteristics of an SVM heavily depends on the observations that are located on the hyperplanes margin or on the wrong side of it. If these observations were somehow moved to other locations, the basis upon which observations are classified will radically change since the hyperplane itself will radically change. Hence, these observations are called support vectors [Hastie et al., 2001, p. 132,417,418,421,423].

The classic SVM treats both types of errors in binary classification equally. Occasionally one type of error is less desirable than the other. To accommodate this request an alteration of the SVM can be used, a weighted SVM. A weighted SVM assigns different weights to each type of error, thus making it possible to reduce one type of error at the cost of increasing the other. To achieve this equation (14) is simply changed to.

$$C^+ \sum_{i:y_i=+1} \epsilon_i + C^- \sum_{i:y_i=-1} \epsilon_i \leq C \tag{14}$$

Where C_+ is the cost for misclassifying an observation as positive. Likewise, C_- is the cost for misclassifying an observation as negative [Osuna et al., 1997].

2.4.2 Logistic Regression

Logistic regression can be used to solve binary classification problems with predictors X and response variable $Y \in \{0, 1\}$. In these cases $p(X)$ defines as the posterior probability $p(X) = P(Y = 1|X)$. Classification of some data X_t using this model then simply becomes:

$$Y_t = \begin{cases} 1, & \text{if} \quad p(X_t) > 0.5 \\ 0, & \text{else} \end{cases} \tag{15}$$

The posterior probabilities can additionally be expressed in a linear form, just like the right-hand side of equation 3.

$$\beta_0 + \beta^T X \tag{16}$$

However, in order to create a general model that returns probabilities in the range of $[0, 1]$ for all values of X, β_0 and β needs to be transformed. Logistic regression, just like its name suggests, uses the logistic function to achieve this. Transforming the posterior probabilities to:

$$P(Y = 1|X) = p(X) = \frac{exp(\beta_0 + \beta^T x)}{1 + exp(\beta_0 + \beta^T X)} \tag{17}$$

$$P(Y = 0|X) = 1 - p(X) = \frac{1}{1 + exp(\beta_0 + \beta^T X)} \tag{18}$$

Using some basic algebra equation 17 and 18 may be combined and expressed as

$$log\frac{P(Y = 1|X)}{P(Y = 0|X)} = log\frac{p(X)}{1 - p(X)} = \beta_0 + \beta^T X \tag{19}$$

The right-hand side of equation (19) is now expressed on the linear form and may be referred to as the log-odds, a common way of expressing the probabilities in logistic regression [Bolstad, 2012]. Training a logistic regression model is all about fitting the parameters β_0 and β. This can be achieved by maximizing the maximum likelihood function which can be found iteratively using Newton-Raphson. Maybe expand this part.

Logistic regression does not only work for binary classification but may also be used for multi-class problems. However since this paper only regards binary classification, multi-class classification falls out of the scope for this paper and will not be presented here.

2.4.3 K-nearest neighbours

K-Nearest Neighbors (KNN) is a so-called non-parametric method for classifying data. The term non-parametric refers to the fact that no assumption about the form of relationship between the predictor values, *X*, and the response value, *Y = f(X)*, is made beforehand.

The advantage of this is that the method can fit models to a wider range of possible shapes for f.

KNN works by simply assigning each observation to the same class as the majority k nearest observation from the training set. If one denotes the subset of k nearest observations to an observation by N_0 then the probability of assigning observation x_0 to class j is given by equation (20)

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} \mathbb{1}(y_i = j). \tag{20}$$

The observation x_0 is classified as the class with the largest probability. The value of k can be tuned in order to select an appropriate flexibility for the classifier. A small k is equivalent to a more flexible classifier, and a large k is equivalent to a less flexible classifier. Finding the optimal k then relates to the classical bias-variance trade-off and will be different for each data set [Hastie et al., 2001, p. 463,465].

2.4.4 Artificial Neural Networks

Supervised machine learning is more than often applied, not because humans lack the ability of classification, but rather to decrease manual work by automatization or to perform tasks that are otherwise unsuited for manual work. The human brain is in fact quite good at recognition, classification, and adaptive learning. Tasks it performs with relative ease [Wells, 1993]. Artificial neural networks are machine learning models that try to mimic the processes of the human brain. A process is known as biomimicry or the imitation systems in nature.

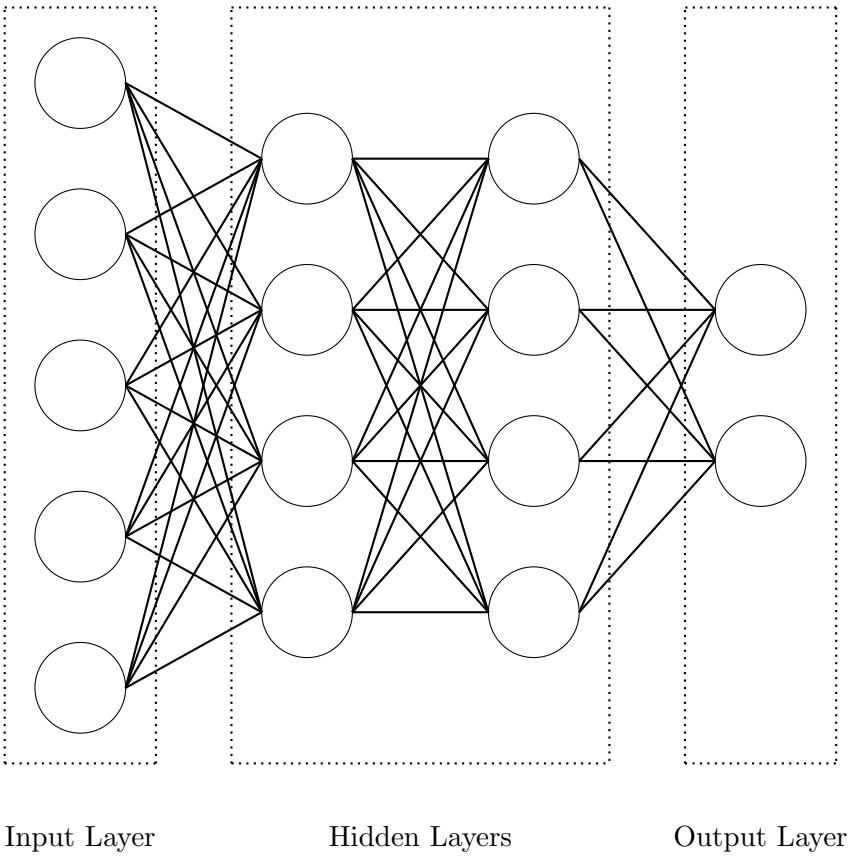

Figure 3: Illustration of a neural network with five input nodes, two hidden layers with four nodes in each and two output nodes.

The building blocks of neural networks are nodes, modeled after the neurons in the human nervous system. These nodes are arranged in input, hidden and output layers. Each node receives information from the nodes in the previous layer and send information to all the nodes in the next one, see figure 3. The nodes in the input layer each represent a feature that is the input or regressors to the model. The output layer represents, as its name suggests, the output of the model. In the case of regression the output layers if often composed out of a single node, while for classification with k classes there are k output nodes. Each output node will, in this case, represent its respective class probability.

Between each neuron, in two adjacent layers, there is a weight attached. Let $w_{i,j}^{(l)}$ denote the weight between node i in layer $l-1$ and j in layer l. For consistency, all superscripts will denote which layer is referred to in the neural network. Every node also has a bias denoted by $b_j^{(l)}$. A visual representation of a single node with inputs and output can be

seen in figure 4

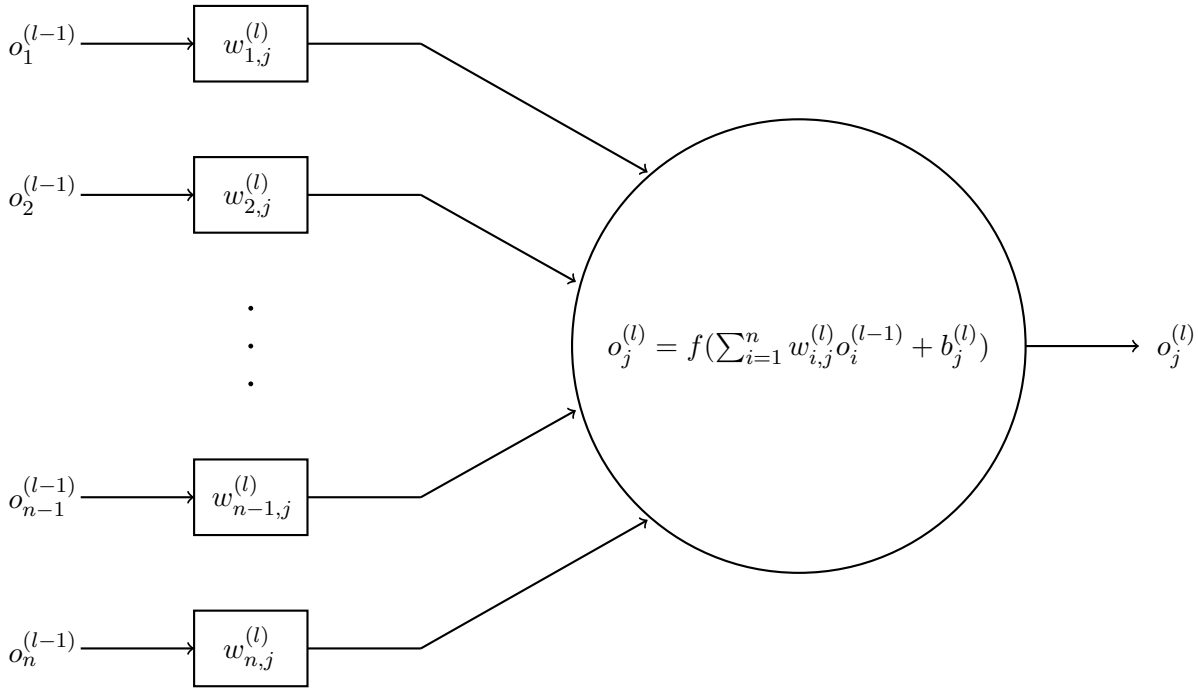

Figure 4: Visual representation of a single node in a neural network.

Where the function f, used to calculate the output of the node $o_j^{(l)}$, is known as the activation function. The most common activation functions are either the sigmoid function given by equation (21). Another popular activation function is the rectified linear unit function, also known as ReLU, seen in equation (22).

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{21}$$

$$f(x) = max(0, x) \tag{22}$$

The performance of a neural network is evaluated by the cost function C. In the case of regression, SSE is used as the cost function. Likewise, if the neural network performs classification C can also be defined as the cross-entropy [Hastie et al., 2001]. Subsequently, the cost function C will be a function dependent on the parameters $\mathbf{w}$ and $\mathbf{b}$ as well as the input and output of the data. The weights and biases are initially chosen randomly but will be tuned during training to minimize the cost C. This is achieved through a process

known as stochastic gradient descent. Stochastic gradient descent is an iterative process in which each iteration a step in the direction of the negative gradient, with a step-size proportional to the negative gradient of the function [Jain et al., 1996].

2.4.5 Classification tree

Tree-based machine learning methods are simple, yet powerful models that can be used for both classification and regression. Since this paper only deals with classification and not regression, only classification trees will be presented in this section.

The basic idea is to create a tree-like structure with leaves and branches by segmenting the feature space using recursive binary spitting. All observations begin in one stem and are thereafter split into two branches each creating a new region. Every split into two new region or branches R_1 and R_2 is done according to the following equations (23) and (24).

$$R_1(s, j) = \{X | X_j < s\} \tag{23}$$
$$R_2(s, j) = \{X | X_j \geq s\} \tag{24}$$

Where X_j belongs to the predictor space X. The parameters j and s are tuned so that the performance metric is optimized. The branching is repeated until all the regions fulfill some predefined criteria. Each terminal region leaf then corresponds to one of the classes in the classification problem. The observations in the terminal node m will be classified to class $k(m)$, which simply is the majority class in that node from the training data. An example of a decision tree can be seen in figure 5, with the terminal regions R_1 to R_6 and using the predictors X_1 and X_2.

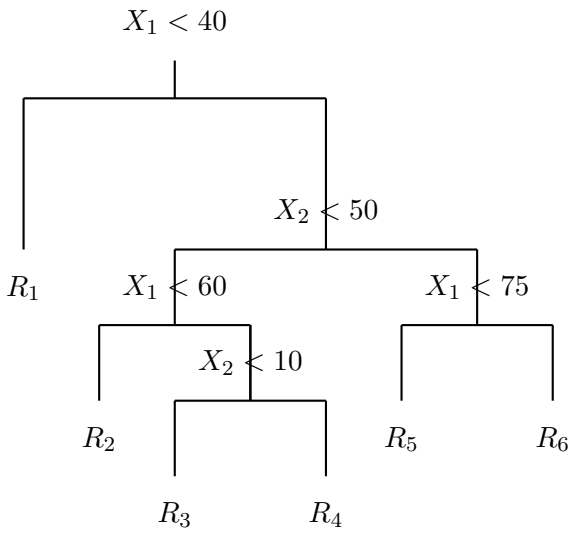

Figure 5: Example of decision tree using two predictors, X_1 and X_2.

The most intuitive way of creating each binary split is to minimize the*classification error rate* in the nodes after the split. However, it turns out that the *classification error rate* does not work well for creating classification trees due to its' low sensitivity. Instead, the to most common ways of measuring node purity is either the *gini index*, which is defined as:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{25}$$

Or *devince* defined in equation (26)

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk} \tag{26}$$

Where $\hat{p}_{mk}$ is the proportion of class k in the terminal node m. The terminal region R_m with N_m observations $\hat{p}_{mk}$ may then be defined as given in equation (27). The binary split are then made such that the node purity in each node is minimized.

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}(y_i \neq k(m)) \tag{27}$$

Decision trees are, as mentioned earlier, simple yet powerful predictive models. Since they in many ways operate like human reasoning are they good models for when a clear classification structure is needed. Unlike other models such as artificial neural networks which are more of a "black box" model. However, classification trees have some drawbacks. For example, when fitting a single tree the structure might become complex which in turn leads to over-fitting and thus high testing errors. This makes trees a low bias, high variance predictive models [Hastie et al., 2001].

2.4.6 Random Forest

One of the most common ways of dealing with over-fitting is using a technique called bootstrap aggregating, also known as bagging. The basic idea behind bagging is to create m bootstrapped samples from the original data. For each bootstrapped sample a model is trained. The output is the average over all the m models for regression and the majority vote for classification. For more information about the bootstrap see section 2.5.2.

Due to their low bias and high variance, trees are excellent candidates for bagging. A modification of bagging for tree models known as random forest is a popular tool for bagging trees. The concept of random forest is as follows. With the original training data $\mathbf{Z}$ new bootstrapped sample $\mathbf{Z}^*$ are drawn. Each bootstrapped set is treated as an I.I.D. sample from their distribution. For each bootstrapped sample a tree is created. However, in each split instead of considering all p predictors only m predictors are considered. This process is repeated until each terminal node fulfills some predefined criteria. This process is repeated B times creating the set of trees $\{T_b\}_1^B$. The prediction from the random forest model then comes from creating B predictions using the set of trees $\{T_b\}_1^B$ and taking the average for regression and the majority vote for classification.

The main difference between random forest and bagging is that in the random forest, not every predictor is considered in the split but only a subsample. The reason for this is to decorrelate the trees in the set $\{T_b\}_1^B$. For example, in a case were the model has one predictor much stronger than the other ones. Bagging would then result in a set of trees which all most likely use this strong predictor in the first split. Thus the set of trees used for prediction will be highly correlated and the variance will not be substantially reduced. However, if random forest is applied instead the probability of the strong predictor to even be considered in the first split is m/p. Thus lowering the correlation between the trees in the model [Hastie et al., 2001].

2.5 Re-sampling methods

Model evaluation may both be done using the training error as well as the testing error. However, the test error gives a better understanding of the model's predictive capability as predicted data from the model was not used when training it. In some situation it is not possible to obtain the testing error. It could be that either there is not enough data to split into training and testing set. Or when tuning parameters when training the model. Re-sampling methods are ways to deal with this and to estimate testing error. The two resampling methods used in this paper, *cross-validation* and *bootstrap* are presented below.

2.5.1 Cross-Validation

A common way to calculate the test error for a model is to train the model on one set of data and then test the model on another set of data with a known response variable. If only one set of data is available, this can still be achieved by splitting the data into a training and testing set. This however introduces some randomness to the result as the test error then depends on how the data set is split. *Cross-validation* is a resampling technique that tries to address this randomness of the test error introduced by the split.

The basic idea behind cross-validation is to divide the samples into multiple sets. One set is omitted, while the other sets are used to train the model. A training error is then calculated using the set left out of training. This process is then repeated once for each set, such that each set is withheld and used as test set once. The error rate is then calculated as the average over each split.

The different types of cross-validation are defined by how the split of the data is made. *Leave-One-Out Cross-validation* or *LOOCV* splits the data by simply putting one observation at a time in the validation set and training the model on the remaining $n-1$ observations. This yields a method which predicts the error with low bias, but the variance can be rather high. Another way of splitting the data is the so-called *k-folded Cross-validation*. Here the data is split into k sets or folds of approximately the same size. Each fold is then used as validation set while the model is trained on the remaining $k-1$ sets. The model is then trained k times, each set used for validation once.

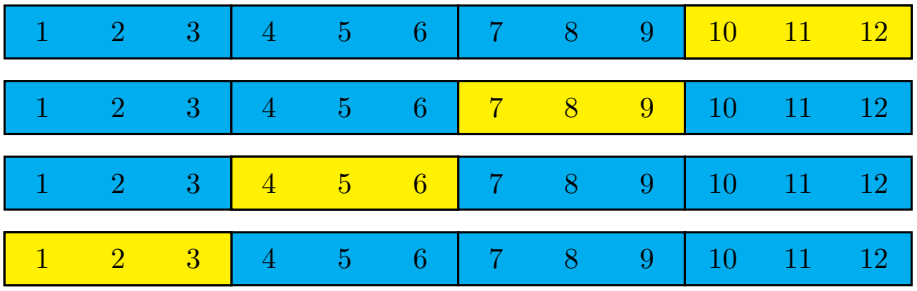

Figure 6: Visual representation of k-folded cross-validation with $k = 4$ on a set of data containing twelve observations, 1 through 12. Adapted from [James et al., 2014, p. 181]

Above in fig 6, a visual representation of k-folded cross-validation is presented where the data is split into four equally sized sets. The data used for training the model is shown in blue, while the testing set is shown in yellow. Here one can see how the model is trained four times and each set is used for testing once and three times for training the model.

The error for k-folded cross-validation is then for classifications calculated as

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} Err_i \tag{28}$$

With

$$Err_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbb{1}(y_j \neq \hat{y}_j) \tag{29}$$

Where k is the number of folds and n_i is the number of observations in the ith fold. Analogous for $LOOCV$ but setting the parameters to $k = n$ and $n_i = 1$. Determining k is a matter of variance bias trade-off. It has been empirically shown setting $k = 5$ or $k = 10$ yields results that neither suffer from high bias or variance [James et al., 2014]

2.5.2 The Bootstrap

When faced with the problem of insufficient data, the most intuitive way of dealing with this would be to collect more data. This would mean replicating the experiment or data collection process that was used to obtain the original data. This however might be time consuming or expensive and in some cases it might not even be possible. The basic idea behind *the bootstrap* is to circumvent this problem by simulating replication of the data

collection process. [Shalizi, 2010] This is achieved the the following way. Given some original dataset Z of size N, containing both predictors and response variables, B new data sets are created Z^{*1} through Z^{*B} each of size N. Each new dataset is created through resampling. That is, by drawing observations from Z with replacements. Meaning that each observation may be drawn more then once every time a new data set is created. Let a statistic from the original data be denoted as $S(Z)$, which belongs to some unknown distribution[Hastie et al., 2001, p. 249]. Using the bootstrapped data sets properties of this statistic can be estimated. For example, the mean can be estimated as

$$\bar{S}^* = \frac{1}{B}\sum_{b=1}^{B} S(Z^{*b}) \tag{30}$$

Likewise, the variance may be estimated as

$$\widehat{Var}[S(Z)] = \frac{1}{B-1}\sum_{b=1}^{B}(S(Z^{*b}) - \bar{S}^*)^2 \tag{31}$$

Bootstrap is a powerful and popular tool for resampling as it is a rather simple and straightforward way of dealing with uncertainty in complected models. Nonetheless, the methods are not without its drawbacks, especially when estimating the prediction error of the model [Shalizi, 2010]. If the model is trained on a bootstrapped sample, most of the observations in the set will also appear in the training data, since the probability of a single observation being drawn when creating a bootstrapped sample is:

$$1 - P(\text{observation not being drawn}) = 1 - (1 - \frac{1}{N})^N \approx 1 - e^{-1} \approx 0.632$$

Meaning that the majority of the observations will have been used when training the model, thereby overfitting the prediction and making the predicted error lower than the true error. To obtain better prediction results more sophisticated forms of the bootstrap have to be used. One way of improving the bootstrap prediction error is to use the leave-one-out bootstrap, inspired by $LOOCV$ presented in section 2.5.1. Let the original prediction error for bootstrap be defined as

$$\widehat{Err}_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, f^{*b}(x_i)) \tag{32}$$

Where $f^{*b}(x_i)$ is the prediction given predictors x_i with the model trained on the dataset Z^{*b}. The function L returns the prediction error for that observation. So in the case of

classification, it simply becomes

$$L(y_i, f^{*b}(x_i)) = \mathbb{1}(y_i \neq f^{*b}(x_i)) \tag{33}$$

The idea of leave-one-out bootstrap is when classifying an observation to omit the bootstrapped data sets that contain it. The estimated prediction error using leave-one-out bootstrap thus becomes:

$$\widehat{Err}_{boot} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, f^{*b}(x_i)) \tag{34}$$

Here C^{-i} are all the bootstrapped sets not containing the ith observation [Hastie et al., 2001, p. 250-252].

2.6 Performance Metrics

In order to properly evaluate a machine learning, model some metrics to determine its performance needs to be established. There are multiple of well-known metrics, where the relevance of the metrics depends on the goal of the objective of the predictive model. Since this paper deal with binary classifiers, only metrics relevant to such models will be presented in this section as other are beyond the scope of this thesis.

Binary classifiers map each observation to the set $\{P, N\}$ defined as positive or negative. This mapping may then be compared to the true class of each observation, resulting in four possible outcomes. The most common way of representing these results is a *confusion matrix*, which can be seen in figure 7.

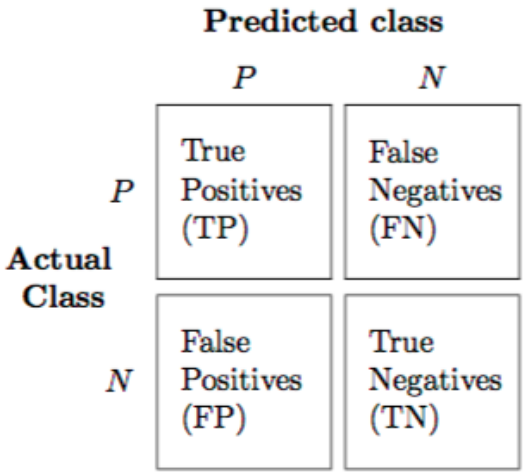

Figure 7: Confusion matrix

The first word indicates if the classification was correct or not. Where *true* represent correct classifications and *false* incorrect ones. The second word shows the prediction of the observation. For example, a *true positive* observation is a positive observation which was also classified as positive. From figure 7 the two types of errors the occur in a binary classifier may be found. These errors are often referred to as type I and type II error. Where a type I error is a false positive observation and type II error a false negative observation. The confusion matrix is the foundation for many of the fundamental metrics for binary classifiers, the ones used in this paper are defined below.

$$Accuracy = \frac{\text{Correct prediction}}{\text{Number of predictions}} = \frac{TN + TP}{TN + FN + TP + FP}$$

$$PPV = \frac{\text{True Positives}}{\text{Positive predictions}} = \frac{TP}{FP + TP}$$

$$Sensitivity = \frac{\text{True positives}}{\text{Actual positives}} = \frac{TP}{TP + FN}$$

$$Specificity = \frac{\text{True Negatives}}{\text{Actual Negatives}} = \frac{TN}{TN + FP}$$

Accuracy simple tells the fraction of correctly classified observations. *Positive predictive*

value or *PPV* is a measure of the confidence in the positive predictions. *Sensitivity* shows what fraction of the positive observations were identified. Lastly *specificity* gives the fraction of negative observations identified

The most intuitive of these is probably *accuracy*. It is an easily interpret metric that gives a good indication of the classifiers performance [Fawcett, 2006]. However, it has its drawbacks. For example, consider a classifier that predicts all observations as positive. Presented with data that has 98% positive observations this classifier would have a 98% accuracy. Even though all negative observations were misclassified. Moreover, the main goal of the model could be to find or not omit observations from a certain class. Consider a medical test for detecting a disease in patients. A positive the test result would then mean that patience is predicted to have the disease. Here detecting all patients with the disease could be the main goal of the model. Even at the cost of miscasting some healthy patience's. Thus *Sensitivity* would poetically be a better metric for evaluating the model.

2.7 Class Imbalance

Class imbalance occurs in any data set where there exists a class substantially smaller than the other classes. This class, referred to as the minority class has fewer observations compared to the other. It is quite simple to illustrate why this will cause problems in the classification. Consider a data set with two classes where 99% of the observations belong to the majority class while the remaining 1% belong to the minority class. A binary classifier that simply assigns all observations to the majority class will have 99% accuracy. This while still miss classifying every instance of the majority class. Even with a smaller imbalance this will be the case as the error rates get minimized at the cost of the accuracy in the classification of the minority class.

The most intuitive way of dealing with class imbalance is to collect more data on the minority class. However, this is not always feasible. In that case there are two common approaches, oversampling and sub-sampling, both involving resampling. In short, oversampling is done by randomly duplicating instances of the minority class in the data set. Likewise, sub-sampling is achieved by randomly removing observations of the majority class. Both approaches lead to a more balanced data.

Another way of dealing with class imbalance that do not involve any resembling is simple to chose a different metric for evaluating the model. Such as sensitivity, specificity, or PPV. The choice of performance metric depends on the goal of the model [Albisua et al., 2013].

3 Method

As previously stated, the aim of this paper is to evaluate different machine learning methods for classification of opened-ended answers to specific brands. The methodology will thus firstly consists of processing the input data such that it can be classified using any statistical learning methods described in the background. Secondly, answers will be classified using the different classifiers.

One approach would be to use a multi-class classifier to map the answers to brands. However, having a large number of classes requires a large feature space in order to obtain accurate results [Abramovich and Pensky, 2017]. Additionally, the relevant information provided from each answer will be a text string and a question for which answer is for. *Nepa's* database of companies contains 139302 brands, each one representing a class. However, it will be shown the number of classes can be reduced to the range of 50-200 for each set of answers. Engineering enough significant features from only a string to obtain an accurate multi-class classifier might not be possible.

However, there are other approaches for classification of multiple classes. One is to use a database search algorithm in order to find matches between the observations and classes. Thereafter a binary classifier determines whether each match is correct or incorrect. This method has been successfully applied in peptide identification using an algorithm called *Percolator* [Käll et al., 2009]. A similar approach will be used in this project where matches will be created between the answers and brands in *Nepa's* database and thereafter a binary classifier determines if the match is correct or not.

The open source statistical programming language *R* was used to create all the scripts used in this project.

3.1 Data

The data provided for this thesis comes from *Nepa's* database. Two different datasets were used and will be referred to as **answers** and **brands**, where answers contain answers to open questions and brands contain brand names. Each entry in the brand's data simply consists of an integer representing the brand id and a string with the brand name, see table 1. Each row in the answers data set represents an answer to an open question in a questioner. Each answer has a unique id, a question id showing what question the answer is for. A brand id which shows to what brand the answer has been manually mapped to and a text string with the actual answer provided, see table 2.

Name	`Brand Id`	`Brand String`
Type	Integer	String
Example	164	Toyota

Table 1: Data structure brands

Name	`Answer Id`	`Question Id`	`Answer Mapping Id`	`Answer String`
Type	Integer	Integer	Integer	String
Example	797244765	145684	164	Toyta

Table 2: Data structure **answers**

For example, the observation in table 2. The `Internal Question Id` maps to the question *"What car brands are you aware of?"*. The `Internal Answer Id`, 164 maps the observation to the brand "*Toyota*", see table 1. The logic behind this manual mapping is quite obvious due to the similarity between `Brand String` for the observation "*Toyta*" and for the brand with `Id` 164, *"Toyota"*. This mapping was done manually at *Nepas* India office.

The rows in the answers data can be divided into subgroups based on context. Context is defined as the country where the question was asked and the industry the question is regarding. In order to limit and further structure the amount of data used in this project three contexts were selected as the base for the analysis. This since the answers data contains millions of observations, a more manageable amount of data was needed in order to perform data cleaning etc and analysis reasonable time. To accommodate the assumptions keyboard layout made in section 3.3.1 the only context where the country is Sweden will be evaluated. Thereby the context will be determined by the industry. The following industries were chosen for evaluation in this paper.

- Flour
- Cars
- E-commerce sites

From the answer data three types of answers can be deducted. Firstly there are the manually mapped answers for which the spelling matches the spelling of the brand it was mapped. These answers will be referred to as *correct*. Then there are the answers which have been mapped to a brand but the answers string does not match the brand string, which will be referred to as *misspelled*. Lastly are the answers which have been mapped to answers which do not represent brands. These answers are mapped to answers such

as *"Don't know"*, *"Unknown"* and *"Other"* and will be referred to as *"nonsense"*. The distribution of these answer can be seen in table 3

Context	Flour	Cars	E-commerce
Correct	7872 (70.2%)	9808 (79.6%)	9923 (69.4%)
Misspelled (Mapped)	1258 (11.2%)	442 (4%)	760 (5.2%)
Misspelled (Nonsense)	2082 (18.6%)	1759 (15.4%)	3814 (26.3%)

Table 3: Distribution of different types of answers in the data, absolute numbers

It is not always necessary to look at every single answer, as the majority of them are duplicates. Both for correctly spelled answers but also common misspellings that occur frequently in the dataset. The number of unique misspellings and nonsense answers in each data set can be seen in figure 4.

Context	Flour	Cars	E-commerce
Unique correct answers	58 (7.8%)	51 (9.7%)	62 (9.3%)
Unique misspellings (Mapped)	325 (43,6%)	211 (40.0%)	224 (34.6%)
Unique misspellings (Nonsense)	362 (48.6%)	265 (50.3%)	381 (56.1%)
Unique misspellings (Total)	745	527	667

Table 4: Unique misspellings and nonsense answers for each context

3.2 Match Candidates

If ML classifiers were to be applied straight to the **answers** data sets, a multi-class classifier would be needed. In order to use a binary classifier, an abstract data type needs to be created. One that can be used as the observations and can be classified as positive or negative. For this purpose, the data structure **match candidate** was created. A match candidate is a match between an answer string and a brand string with features describing the characteristics of the match.

The first step in creating match candidates is to identify all misspelled answers, as these are answers for which a mapping is needed. All the correctly spelled answers are identified by searching if the answer string matches any of the strings in the brands data set. The brands for which a matching string in the answers data was found are put in a separate data set named **context brands**. If the context already exists in the database all the brands previously mapped in that context are also added to the context brands. The rest

of the answers are put in the **misspelled answers** data set. It is from these answers that the match candidates are created.

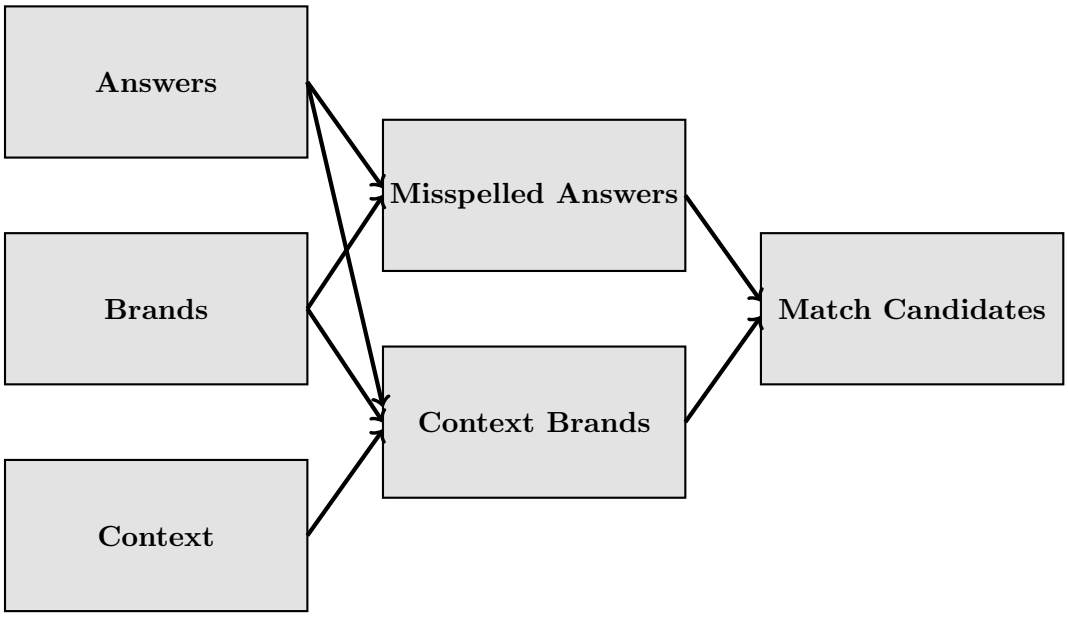

Figure 8: Visualization of the creation of match candidates

For every unique misspelled answer and brand there is a potential match candidate. The dataset brands contains 139,296 brand names. From table 4 one sees that the number of unique misspelled answers in the three context are in the range of 500-750. This sets the magnitude of potential match candidates to around 10^8. Not only would this make the feature calculations described in section 3.3 extremely heavy, but also create an extreme imbalance between the two classes. In order to deal with this, the assumption is made that all of the potential matches for the misspellings are within the subset of brands that are context brands. This way the number of brands for can be reduced to a magnitude of 100.

Name	`Brand Id`	`Brand String`	`Answer String`	`Response`
Type	Integer	String	String	Boolean
Example	164	Toyota	Toyta	Positive

Table 5: Data structure of match candidates before any features have been engineered

The match candidates will form the observations for which the machine learning models

are built upon. A match candidate from the training data will be defined as positive if and only if the `answer id` matches the `brand id`. An example of how the data structure of a match candidate looks like between the examples in table 1 and 2 can be seen in table 5. This is before any features have been added.

3.3 Feature Engineering

The match candidate introduced in section 3.2 in its current form, seen in table 5, does not provide any meaningful features that model can use to classify the match candidates as correct or incorrect. In order to do so, features have to be created, a process known as feature engineering. A detailed explanation of the features engineered for the match candidate is presented in the section below. A shorter summary can be found in table 6

3.3.1 QWERTY weighted distance

Recall the Damerau-Levenshtein distance presented in section 2.2.1. A natural extension for strings written on a QWERTY keyboard is to weight the cost of substitution depending on the distance between the characters to be substituted. The feature `QWERTY weighted Damerau-Levenshtein distance` the Damerau-Levenshtein distance between the answers string and the string representing the brand name, weighted such that substitution cost for neighboring keys on the QWERTY-keyboard is 0.5 instead of the usual 1. In this paper a neighbor key is defined as an adjacent key as seen in figure 1 that produces a character. For example the key *"K"* has six neighbours which are { *"J","I","O","L",",","M"* }

Lets consider the three strings: *'cat'*, *'hat'* and *'fat'*. Then the QWERTY weighted Damerau-Levenshtein distance between them will be the following

$$QWERT\_lev(cat, hat) = 1 \tag{35}$$
$$QWERT\_lev(hat, fat) = 1 \tag{36}$$
$$QWERT\_lev(fat, cat) = 0.5 \tag{37}$$

All strings can be transformed into any of the other two by substitution of the first letter, thus they all have a Levenshtein distance of 1 between them. However, since the 'c' and 'f' keys are neighbors on the substitution cost is reduced from 1 to 0.5.

3.3.2 Overlapping distance

A major drawback of `QWERTY weighted Damerau-Levenshtein distance` is that when used on strings of dissimilar size, the difference in size will set a lower bound for the `QWERTY weighted Damerau-Levenshtein distance`. This may cause problems if an answers string is a substring or close to a substring of the brand string or vice-versa. For example, consider the string *'kungsörnen'* and *'kungsö'* which will have a `QWERTY weighted Damerau-Levenshtein distance` of 4, even though there Szymkiewicz-Simpson coefficient is 1.

In order to capture these subsets, a feature called `overlapping distance` was created. The `overlapping distance` is an adaptation of the intersection between the two strings. The `overlapping distance` is then the `QWERTY weighted Damerau-Levenshtein distance` between the intersection between the two strings.

3.3.3 String size and frequency

Two features were engineered with regards to string size. `Answer_size` which is the number of characters in the answer string and `Brand_size` which likewise is the number of characters in the brand string. Additionally, a feature a for describing how frequent a particular misspelling is in a set of answers was added and called `frequency`. The `frequency` of an answer is calculated as simply the number of answers with that particular misspelled string divided by the total number of misspelled answers.

3.3.4 Brand distance

`Brand distance` is a measure of how similar the string for a brand is compared to all the other brand strings in the context. It gives the shortest Levenshtein distance to all possible brands in the same context. Due to the assumption that the brand-strings in the database are correctly spelled, this distance will not be QWERTY-weighted. The reasoning behind this feature is to indicate how close each brand is to each other and does some indication how close one has to be to it in order to predict a correct match.

3.3.5 Brand Probability

Recall the set of correctly spelled brands presented in section 3.1. This set contains information about the characteristics of the context, which could prove useful for in the classification of the match candidates in that particular context. For example, it provides the frequency of each brand among the correctly spelled answers in a context. With this

information the posterior probability for an arbitrary answer can be calculated. Let A be an arbitrary misspelled answers, $f(A)$ be a function that maps A to the correct brand. The set of correctly spelled answers in the context is denoted by C and B is an arbitrary answer in the context. Then the posterior probability of a mapping of A to B is given by equation 38

$$P(f(A) = B|C) = \frac{|B \cap C|}{|C|} \tag{38}$$

This posterior probability may be calculated for a match candidate and used as a feature in the machine learning model. This feature will be referred to as `brand probability`

3.3.6 Google Suggestion

If one is unsure about a particular spelling most people would now days not turn to a dictionary but rather use the web. More specifically use *Google*. Not only is using the web faster, it also contains information about names on brands, products etc. that are not usually included in a classic dictionary. When a common misspelling is typed into the Google search bar one of two things happens. Either the correctly spelled term will show up as a suggestion, or the search will directly be redirected to the correct spelling. Google does not give exact details on how these suggestions are generated. However, a baseline of how the algorithm works is made public and works the following way:

A user enters a misspelled string into the Google search bar and is presented with links related to the misspelled word. The user does not click on any link as they are not related to the word they are after, realizes that the search string is misspelled and enters the correct one in the search bar. Now the user is presented with relevant links to the word they are after and clicks on one of them. These actions are repeated millions of times by Google's users and will provide correlations between misspelled search strings and the correct ones [Merrill, 2007].

These suggestions could provide useful information for the match candidate as they could tell if the `answer string` is a common misspelling for the `brand string`. A web scraping script was created in `R` to create the binary predictor `google suggestion`. The script enters the `answer string` and returns 1 if the `brand string` is suggested or directly redirected to. Otherwise, the return value is 0.

3.4 Machine learning models

The features engineered in section 3.3 form the features to be used by the machine model. A summary of the features may be found in table 6.

Feature Name	Description	Range
`QWERTY weighted distance`	Similarity measure between answer and brand weighted based on a QWERTY keyboard	$\mathbb{N}/2$
`Overlapping distance`	Distance between overlapping parts of brand and answer	$\mathbb{N}/2$
`Answer size`	Number of characters in answer	$\mathbb{N}$
`Brand size`	Number of characters in brand	$\mathbb{N}$
`Brand distance`	Distance between brand and most similar brand in context	$\mathbb{N}$
`Frequency`	The frequency of the misspelled answers among all misspelled answers	[0,1]
`Brand probability`	Posterior probability of match to brand in the context	[0,1]
`Google suggestion`	Tells if the brand suggested by Google if the answer is searched for	{0,1}

Table 6: Summary of the features with a short description

The machine learning models in section 2.4 and some of their derivatives were used to classify the match candidates. Before the models can be trained their respective parameters need to be set. These are parameters like the k in KNN and the number of nodes in an ANN. In order for the models to perform optimally, their parameters need to be tuned.

The tuning of the parameters was done using the `R` package `caret` in the following way. For each parameter in each model, a set of values are provided to be investigated. Thereafter, provided with training data the testing error is approximated using 10-folded cross-validation. This is repeated for every possible set of variables provided to the model. By default, the model with the lowest testing error is chosen [Kuhn, 2008]. It is however possible to define a different metric of choice to be optimized.

Since the aim of the model is to produce mapping with high confidence, in other words, to reduce the type I error the natural metric to consider would be specificity. However, due to the class imbalance in the data, the high sensitivity will not directly translate into high confidence in positive predictions. Instead, PPV will be used as the main metric for evaluating the performance of the classifiers. If two or more models produce similar results in terms of PPV the secondary metric to evaluate is sensitivity, as this shows what proportion of correct matches identified. High sensitivity leads to a smaller proportion of

the non-nonsense answers being sent to manual mapping. If both PPV and sensitivity are high enough manually mapping may not be needed at all.

3.5 Final Model

All the building blocks for assembling the final model are now presented. The main steps in the model are the following.

1. Create match candidates
2. Tuning parameter and train a classifier for match candidates
3. Classify match candidates
4. Map misspelled answers

The input consists of three elements. Brands, answers, and context. The model works the following way.

3.5.1 Create match candidates

Firstly the model needs to create brand matches out of the brands and answers. In section 3.2 it is presented how for every context brands there is a potential match candidate for each misspelled answer. However, if a match candidate is created for every potential match a severe imbalance between the two classes in the match candidate data would be created. As mentioned earlier, each set of context brand contains roughly 100 brands. In best case scenario, meaning the misspelled answers do not contain any nonsense answers, this would generate about 99 incorrect match candidates for every correct match candidate. But as one can tell from table 3 between 70% to 85% of the misspelled answers are nonsense answers, meaning that they will not yield any positive match candidates.

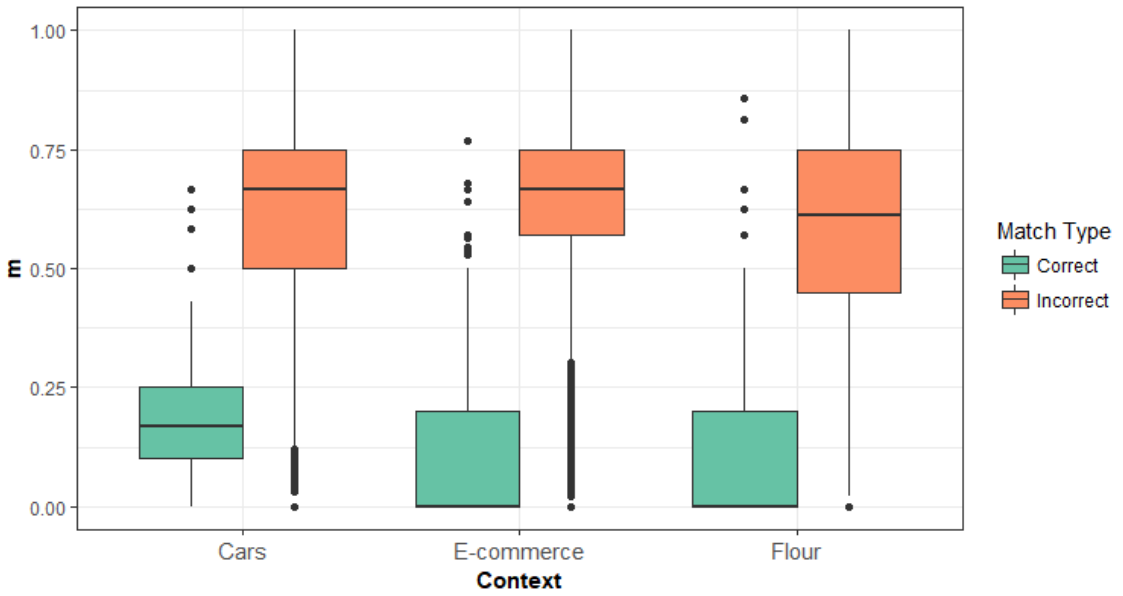

Figure 9: Cut-off

A metric is needed to determine when a match candidate between a brand and answer should be created. To accommodate request, a new variable *match score*, m is created. The match score is based on the szymkiewicz-Simpson coefficient introduced in section 2.2.2, but where the union of the two variables is replaced by the `overlapping distance`. The match score is defined in equation (39).

$$m = \frac{\texttt{overlapping distance}}{min(\texttt{answer size}, \texttt{brand size})} \tag{39}$$

Since the `overlapping distance` has a upper bound by $min(\texttt{answer size}, \texttt{brand size})$ this will yield a value in the range $[0, 1]$. In figure 9 box plots show the distribution of m for both correct and incorrect match candidates in each context. From this plot one can tell that, despite there some overlapping, there is a clear separation where the distributions are centered between the correct and incorrect match candidates for all the contexts. Setting a threshold T, such that match candidates with a m value above T will be discarded, would allow removing the majority of incorrect matches. At the same time, only a smaller fraction of the correct match candidates would be removed.

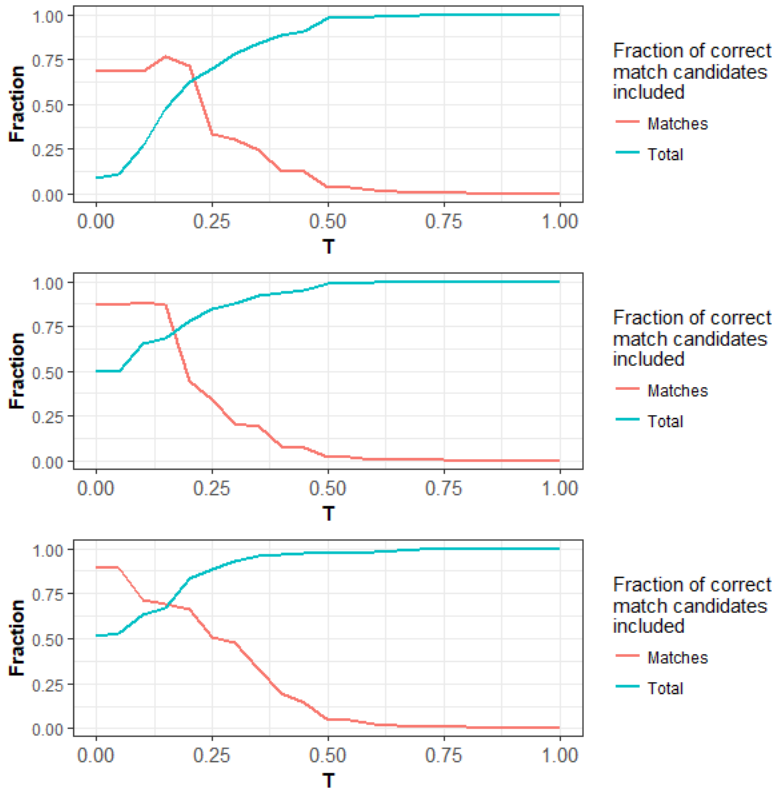

Figure 10: Ratio correct match candidates and fraction correct match candidates remaining with threshold T for each context. Top = cars, middle = e-commerce and bottom = flour

In figure 10 the fraction of correct match candidates for all match candidates created (seen in blue). And the fraction of correct match candidates remaining after applying a threshold T can be seen for different values of T. This shows the trade-off in choosing the variable T between having an even class balance and losing a large number of correct match candidates. There are also certain characteristics in each context with different optimal T

3.5.2 Classify match candidates

Once the match candidates have been created, a binary classifier will classify each observation as correct or incorrect. Beforehand the classifier will have to be trained and parameters tuned according to section 3.4. The training data may both come from the same context or different ones.

3.5.3 Mapping of misspelled answers

With the match candidates classified, the model has all the information it needs to map the misspelled answers. For each unique misspelled answer there are four distinct outcomes after the classification. These four outcomes will be referred to as different output groups named OUT_1, OUT_2, OUT_3 and OUT_4. The answer strings will be sorted into these groups according to the following:

- OUT_1: One and only one match candidate with the answer string being classified as correct.
- OUT_2: Two or more match candidates with the answers string being classified as correct.
- OUT_3: No match candidates with the answer string being classified as correct.
- OUT_4: No match candidates were found for the answer string

The answers in OUT_1 will be mapped to the brand in the corresponding match candidate. As the aim of the model is to create a mapping with high confidence the answers in the rest of the output groups will not be mapped. Treatment of OUT_2, OUT_3 and OUT_4 is out of the scope of this paper but will briefly be discussed in the *future works* section 5.4.

4 Results

Before evaluating the different machine learning models the parameter T needs to be chosen. In table 7 and 8 the results from different values of T can be seen, same as in figure 10. In order to create a general model that will work on different contexts, the value for T was set somewhat conservatively at 0.33 in order not to discard too many correct match classifications in some contexts.

Context	Flour	Cars	E-commerce
T = 1	0.6%	0.5%	0.2%
T = 0.5	14.1%	13.8%	7.8%
T = 0.33	33.5%	21.8%	19.0%
T = 0.2	77.1%	54.2%	47.4%

Table 7: Percentage correct match candidates in observations created for different values of T

T	Flour	Cars	E-commerce
1	100%	100%	100%
0.5	97.6%	98.9%	98.7%
0.33	95.0%	84.3%	91.9%
0.2	83.4%	70.0%	84.5%

Table 8: Percentage correct match candidates remaining for different values of T

By applying this threshold the percentage correct match candidates can be increased from less than 1% to around 20% for all contexts. While at the same time retaining more than 84% of all correct match candidates remain for all contexts.

4.1 Match candidate classification

In this section, the results from the classifications of the match candidates will be presented. The classifiers will be evaluated in two different setups. Firstly, with training data from the same context. Secondly, with training data from the two other contexts. This is done in order to simulate two different cases when *Nepa* maps answers. Either on a new or already existing one. To use common statistical terminology a correct match will also be referred

to as a positive observation. Likewise, an incorrect match candidate may be referred to as a negative observation.

The ML models tested are the ones presented in section 2.4. SVMs with linear, polynomial and radial kernels are tested. However, only the best performing SVM, polynomial, is shown in the section below. For the results for the other kernels along exact number for the figures seen in this section , see appendix A and B.

4.1.1 Scenario 1: trained on the same context

In scenario 1 the classifiers are trained on data from the same context as the match candidates to be classified. In this scenario, the misspelled answers are split into a training set and a testing set, with 25% of the misspelled answers in the testing set and the remaining 75% in the training set. The ML models are then trained on match candidates created using the training data. Thereafter, the match candidates created from the testing data are classified. In figure 11 through 13 the resulting metrics for each model in every context is shown. Along with the metrics of interest, PPV and sensitivity, the metrics accuracy and specificity are shown in order to show the characteristics of the classifier.

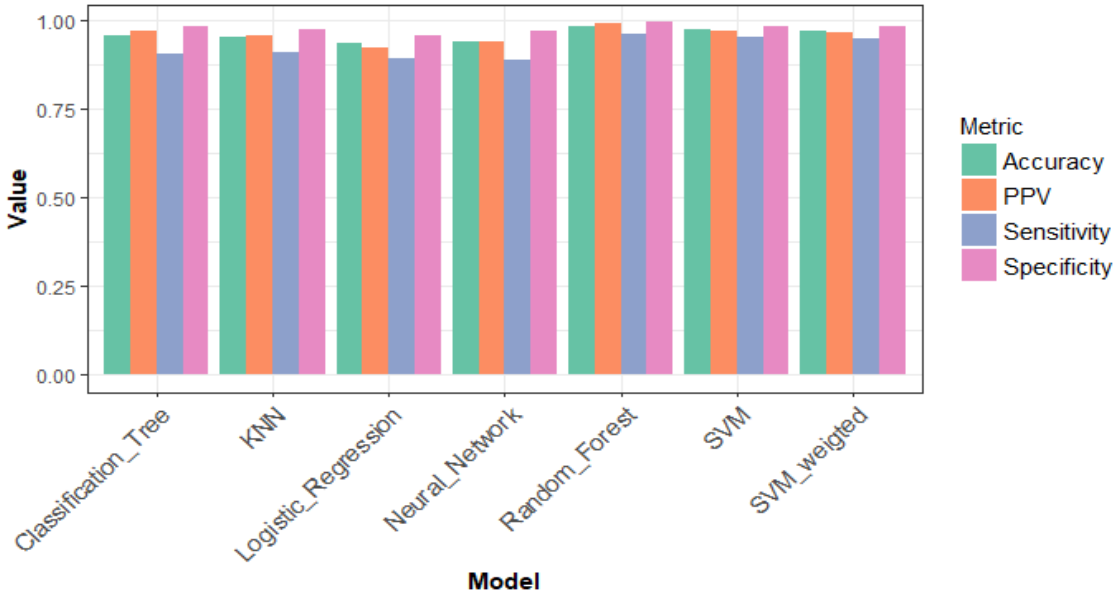

Figure 11: Metrics for match candidates from e-commerce data, trained on the same contexts

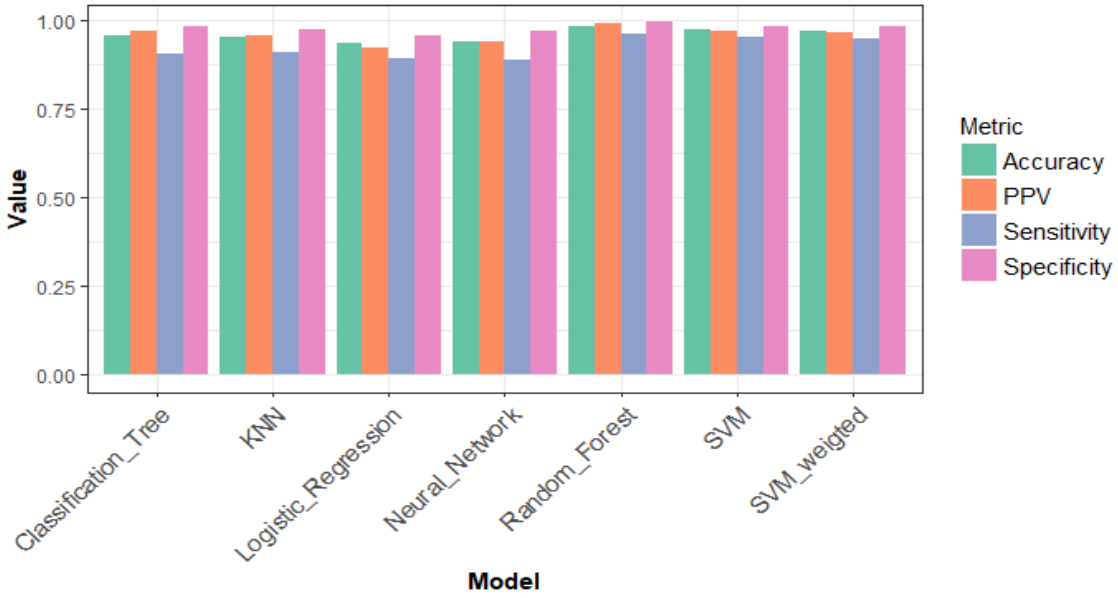

Figure 12: Metrics for match candidates from flour data, trained on the same contexts

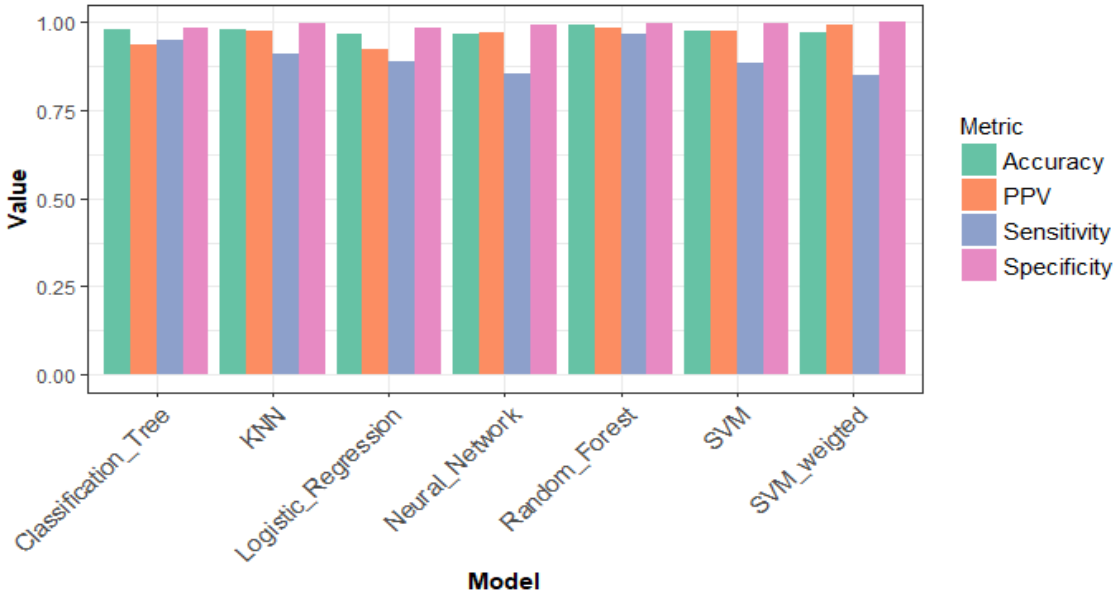

Figure 13: Metrics for match candidates from car data, trained on the same contexts

All classifiers perform exceptionally well in terms of PPV. Especially the random forest

model, both SVMs and the classification tree seems to perform very well. This is verified by figure 14 were the resulting average metric for the three contexts are shown. It shows that in average number the random forest model outperform the other model both in term of PPV and sensitivity.

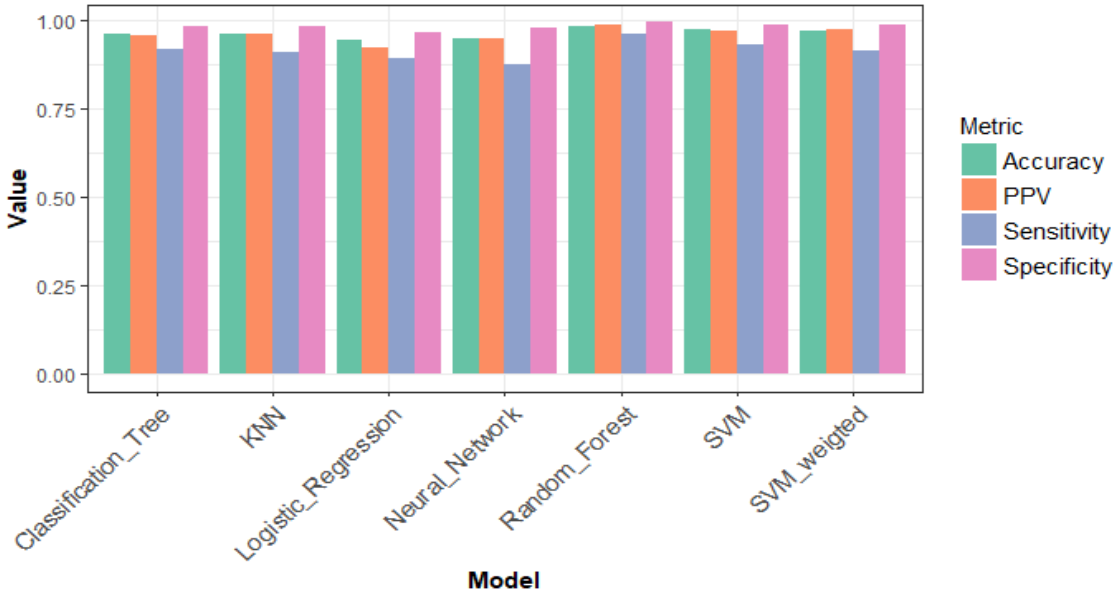

Figure 14: Average metrics for all contexts, trained on same context

4.1.2 Scenario 2: Trained on different contexts

In order for the model to operate on new context it needs to be able to train on context different from the one that is being classified. Thus, each classifier was be trained on the two context not being classified. This process was repeated for each context. The resulting metrics for each context can be seen in figure 15 ti 17. Notable is how much better the model works on the car data in this scenario compared to the other contexts.

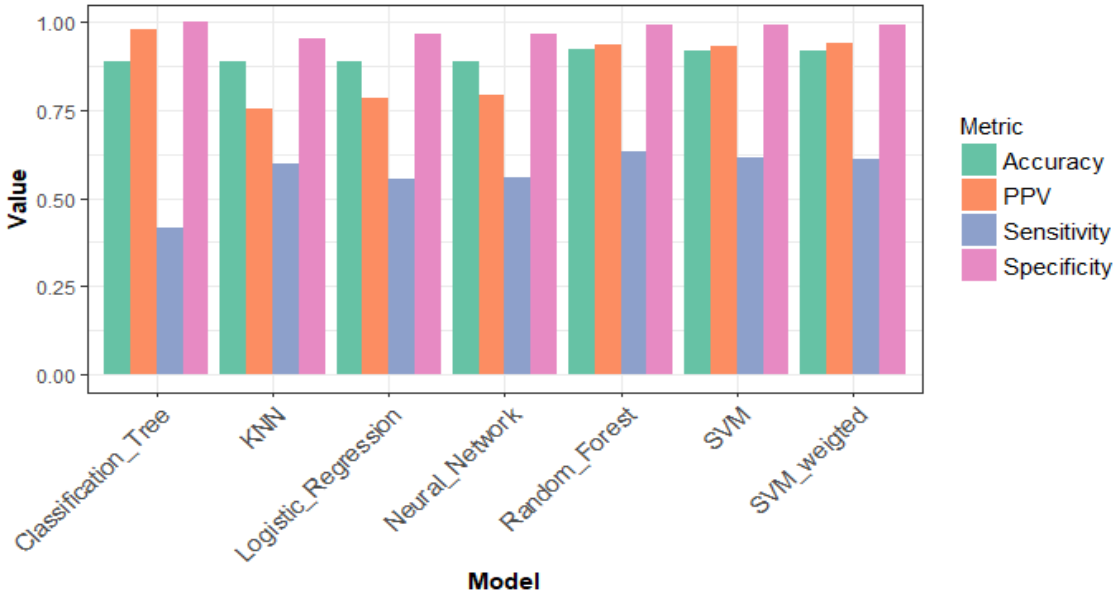

Figure 15: Metrics for match candidates from e-commerce data, trained on other contexts

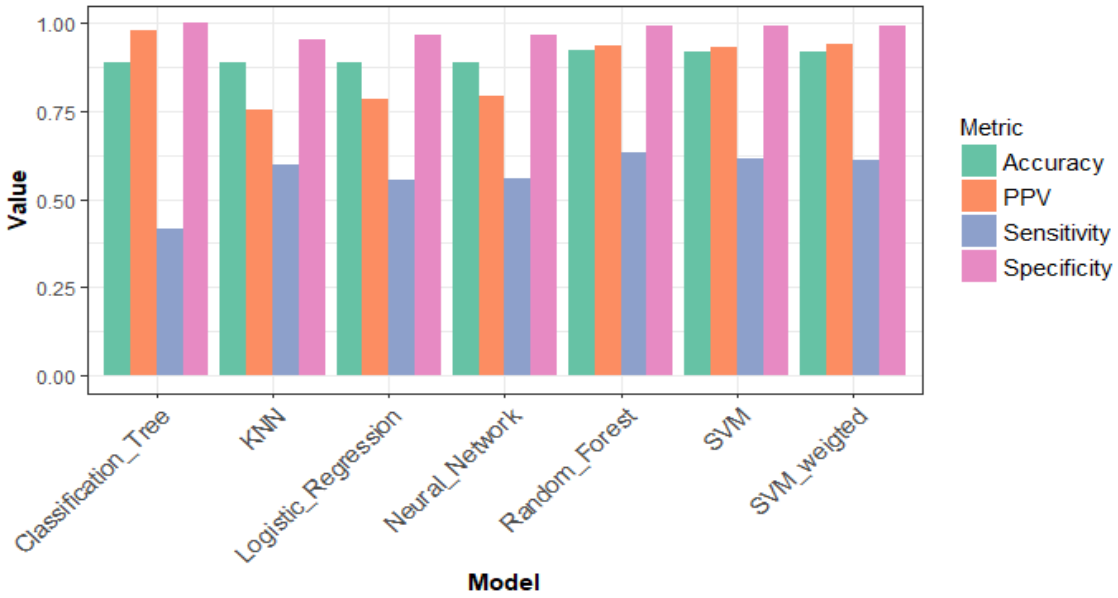

Figure 16: Metrics for match candidates from flour data, trained on other contexts

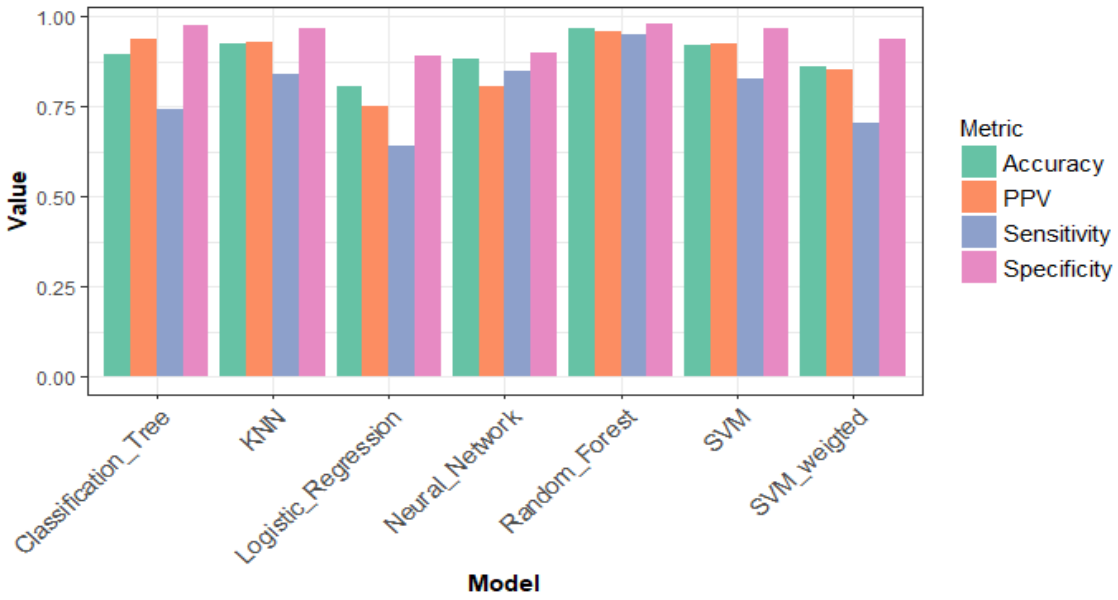

Figure 17: Metrics for match candidates from car data, trained on other contexts

The mean metrics values can be seen in figure 14. Just like in the previous case the random forest model, the SVMs, and the classification tree outperforms the other classifiers in terms of PPV. In this scenario, the classification tree has the average highest PPV. However, it also has the lowest average sensitivity. The random forest classifier has a PPV similar to the classification tree but also has the highest average sensitivity.

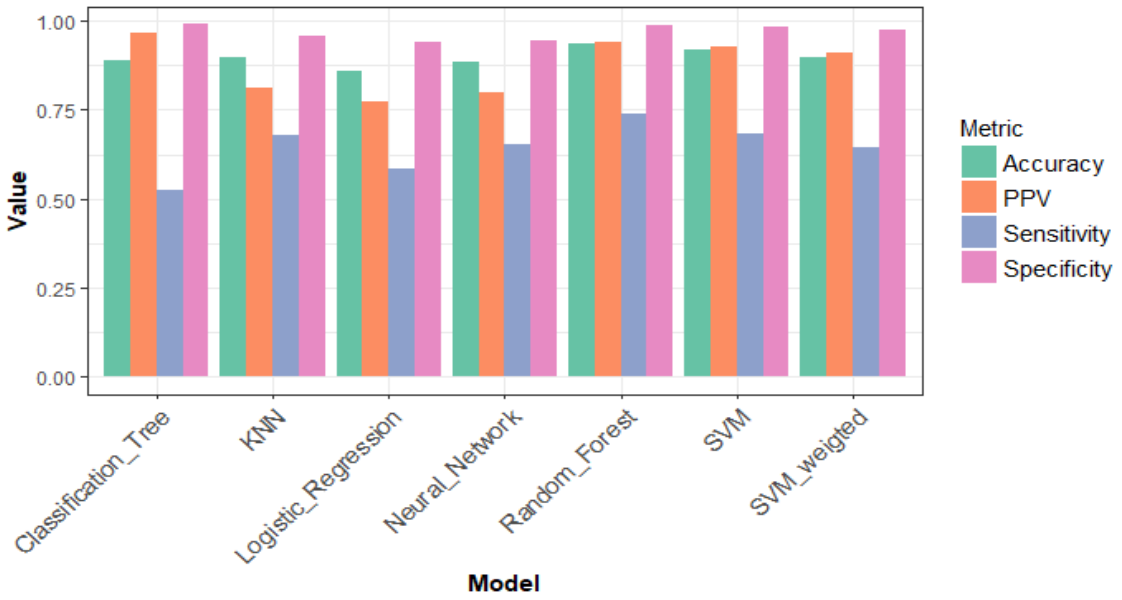

Figure 18: Average metrics for all contexts, trained on other context

4.2 Mapping of misspelled answers

Using the results in the previous section the random forest model was selected for classifying the match candidates. Using this classifier the misspelled answers were mapped as described in section 3.5.3. In table 9 and 10 the results from the mapping for each context, trained on data from the same context is shown.

Context	Flour	Cars	E-commerce
Correctly mapped answers	80	34	47
Incorrectly mapped answers	4	1	3
Omitted mappings	4	3	3

Table 9: Results after mapping looking at unique answer strings. Trained on same context

Context	Flour	Cars	E-commerce
Correctly mapped answers	139	64	90
Incorrectly mapped answers	4	1	3
Omitted mappings	10	6	7

Table 10: Results after mapping looking at total number of answers. Trained on same context

Table 9 shows the results in unique answer strings while 10 shows the same results in form of absolute number of answers. Correctly mapped answers referrers to answer strings which were mapped to the correct brand. Incorrect mapped answers are answers that are either mapped to the incorrect brand, or that should not have been mapped at all. Lastly, omitted mappings are answer strings which should have been mapped to a brand but was not. From table 10 one can deduct that on average the model identifies 92% of the mapped answers in the data. Among the mapped answers, on average 98% are correctly mapped.

Likewise, 11 and 12 shows the same data but for the second case when training was done on data from other contexts

Context	Flour	Cars	E-commerce
Correctly mapped answers	234	153	154
Incorrectly mapped answers	17	3	22
Omitted mappings	91	58	70

Table 11: Results after mapping looking at unique answer strings. Trained on other context

Context	Flour	Cars	E-commerce
Correctly mapped answers	861	379	438
Incorrectly mapped answers	35	7	22
Omitted mappings	397	63	332

Table 12: Results after mapping looking at total number of answers. Trained on other context

In this scenario, the model were able to, on average, identify 70% of the mapped answers in the data. Moreover, among the answers mapped by the model 95% were mapped

correctly.

5 Discussion

5.1 Remarks on the classifier

A review of figure 11 through 18 reveals that most models tested in this thesis provide decent results. However, in terms of PPV and sensitivity, random forest is the superior classifier in both scenarios tested. For some context in the first scenario, the random forest classifier performs nearly perfectly with almost 100% accuracy. From figure 14 and 18 one can see that there is only a small difference in PPV between scenario 1 and 2. The sharpest contrast in terms of performance between the two scenarios is the sensitivity which is noticeably lower in scenario 2. What this tells us is that the classifier's ability to identify positive observations will be substantially lowered when trained on data from a different context. However, the confidence in positive prediction is only lowered by a few percentage points. This shows that each context possesses certain characteristics that the classifier can not learn from other contexts. This information could prove vital if this model were to be applied, meaning that if possible, data from the same context should always be used for training the classifier.

Notable is that adding weights to the SVM did not improve its performance considerably. In scenario 1 it performed just slightly better than the normal SVM while in scenario 2 it performed worse. The reasoning behind this is most likely that the weighted SVM overfits the model, which it gets punished for in scenario 2. All in all, the model seems to favor low bias, high variance models, such as the classification tree. The reason behind this is most likely that most misspellings have very similar characteristics. Recall that in section 2.2 it was stated that most misspellings are due to one in three operations. Deletion, inserting or substitution. Moreover, in scenario 1 some of the observations in the testing data will most likely have duplicates in the training data, due to common misspellings being in both sets of data. Thus, the model can get away with overfitting to the data.

5.2 Remarks on mapping of misspelled answers

The resulting mapping of misspelled answers are in line with the results from section 4.1. The average confidence in mapped answers is at 98% and 95% for scenario 1 and 2 respectively. Just like expected the mapping follows the trend of the classifier and the model omits substantially more mappings in scenario 2 compared to scenario 1. However, the fraction of correctly mapped answers that is identified by the model is lower than the sensitivity for the classifier, 92% and 70% compare to the sensitivity of 97% and 83% respectively. This was expected since the amount of omitted mappings will be the sum of two parts. False negative observations from classifier and correct match candidates omitted in the data due to the match score threshold T.

5.3 Remarks on the validity of the data

As mentioned previously, the validation data is built upon the manual mapping which was done at *Nepa's* India office. Since this mapping is not 100% accurate, the data of mapping accuracy may be affected. Anecdotal evidence of this may be found in the resulting mapping of car data in scenario 2. Table 11 shows that in this scenario three unique answers were mapped incorrectly by the model. Theses answers labeled 1,2 and 3 are shown in table 13 together with the mapping done by the model and the "correct" manual mapping.

#	**Answer String**	**Model Mapping**	**Manual Mapping**
1	astonmartin	aston martin	Unknown (Nonsense answer)
2	hundai	hyundai	honda
3	asda	mazda	Other (Nonsense answer)

Table 13: model vs manual mapping

It is quite obvious that the model mapping of answer 1 is most likely the correct one, as the only difference between the answer and brand string is a spacing in the middle. For answers 2 and 3 one could also argue that the model mapping is superior to the manual. If these mappings were verified to be correct, the PPV of this case could increase to 100%. Of course, there could be questionable manual mappings that the model also mapped the same way that lowers the PPV, but no such was found for this example. Moreover, several instances of answers that are manually classified as nonsense answers that should have been mapped to a brand are found, for which the model also did not map to any brand. This shows that in this case, the true sensitivity of the model is lower then the one calculated. This is only anecdotal evidence but shows that due to the error in the validation data, investigation of the mapping of the model is needed to get its true performance. However, these questionable manual mappings seem to be only a small fraction of the total manual mappings. Thus the calculated results of the model should be in the proximity of the true values.

5.4 Future work

This thesis presents a model which forms a solid foundation for classification of open-ended answers to a finite set of brands. Other approaches for mapping the misspelled answers could be investigated in order to compare with the model presented in this paper. This could include but is not limited to, testing other data structures for the match candidates for classification, creating and using other features etc. The model itself also has some areas for further investigation

5.4.1 More contexts

This thesis only investigates the models' performance of three different contexts. If additional contexts were to be tested more in-depth conclusions about the model could be drawn. Furthermore, as discussed in section 5.2, there is a clear correlation between training the model on the same context and performance. If more contexts were to be tested, it could be investigated if this correlation holds for all context. Or if there are other correlations between different training and testing contexts. Such that whether or not some contexts are better for training than others. Another interesting aspect to examine would be to find out which of the variable in the context, country or branch, relates more to testing results. This way one could possibly create some function for predicting the optimal training context, even if the testing context is new to the model.

5.4.2 Determining the match score threshold

The match score threshold T is in the results somewhat arbitrarily set, such that works for all the three context tested in this thesis. However, as is shown in this thesis, the match score threshold greatly affects the class balance and the number of omitted mappings by the model, both having a large impact on the overall performance of the model. Recalling figure 10 it was shown that the optimal value for T varies from context to context. With further investigation into T effect on the model, a more mathematical way of setting this parameter could be provided in order to optimize the model. One possible way of finding the optimal T would be to calculate the hellinger distance between the distributing of correct and incorrect match distribution with respect to the match score m for each context.

5.4.3 Treatment of output groups

Recall the output groups outlined in section 3.5.3. The model in its current state groups output group OUT_2 through OUT_4 to the same groups. As these answers are not mapped or further treated. If the model is to be realized these groups should be further investigated. Especially interesting is group OUT_2 as it contains answers which have been mapped by the model, just to several brands. There one could most likely find answers which are to be mapped and not seen as nonsense answers. Furthermore, if all possible data are to be extracted from the answers, the answers not mapped by the model would most likely be sent to a manual mapping. The different output groups together with classification score from the classifier could provide useful information to assist this manual mapping.

References

[Abramovich and Pensky, 2017] Abramovich, F. and Pensky, M. (2017). Classification with many classes: challenges and pluses.

[Albisua et al., 2013] Albisua, I., Arbelaitz, O., Gurrutxaga, I., Lasarguren, A., Muguerza, J., and Pérez, J. M. (2013). The quest for the optimal class distribution: an approach for enhancing the effectiveness of learning via resampling methods for imbalanced data sets. *Progress in Artificial Intelligence*, 2(1):45–63.

[Bolstad, 2012] Bolstad, W. M. (2012). Logistic regression. In *Wiley Series in Computational Statistics*, pages 179–202. John Wiley Sons, Inc., Hoboken, NJ, USA.

[Dang and Phan, 2010] Dang, Q. T. and Phan, T. H. (2010). Determining restricted damerau-levenshtein edit-distance of two languages by extended automata. pages 1–6. IEEE Publishing.

[Dickinson et al., 2012] Dickinson, M., Brew, C., and Meurers, D. (2012). *Language and Computers*. Wiley.

[Fawcett, 2006] Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874. ROC Analysis in Pattern Recognition.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[Hoyer and Brown, 1990] Hoyer, W. D. and Brown, S. P. (1990). Effects of brand awareness on choice for a common, repeat-purchase product. *Journal of consumer research*, 17(2):141–148.

[Huang and Sarigöllü, 2014] Huang, R. and Sarigöllü, E. (2014). How brand awareness relates to market outcome, brand equity, and the marketing mix. In *Fashion Branding and Consumer Behaviors*, pages 113–132. Springer.

[Jain et al., 1996] Jain, A. K., Mao, J., and Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, 29(3):31–44.

[James et al., 2014] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2014). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

[Joachims, 2001] Joachims, T. (2001). *Learning to Classify Text Using Support Vector Machines*. Kluwer.

[Käll et al., 2009] Käll, L., Spivak, M., Weston, J., Bottou, L., and Noble, W. S. (2009). Improvements to the percolator algorithm for peptide identification from shotgun proteomics data sets. *Journal of Proteome Research*, 8(7):3737–3745. PMID: 19385687.

[Kim et al., 2003] Kim, H.-b., Gon Kim, W., and An, J. A. (2003). The effect of consumer-based brand equity on firms financial performance. *Journal of consumer marketing*, 20(4):335–351.

[Kuhn, 2008] Kuhn, M. (2008). Building predictive models in r using the caret package. *Journal of Statistical Software, Articles*, 28(5):1–26.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

[Merrill, 2007] Merrill, D. (2007). Search 101. `https://www.youtube.com/watch?v=syKY8CrHkck#t=22m03s`.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Noyes, 1983] Noyes, J. (1983). The qwerty keyboard: a review. *International Journal of Man-Machine Studies*, 18(3):265 – 281.

[Osuna et al., 1997] Osuna, E., Freund, R., and Girosi, F. (1997). Support vector machines: Training and applications.

[Pirinen and Lindén, 2010] Pirinen, T. A. and Lindén, K. (2010). Finite-state spell-checking with weighted language and error modelsbuilding and evaluating spell-checkers with wikipedia as corpus. In *7th SaLTMiL Workshop on Creation and use of basic lexical resources for less-resourced languages LREC 2010, Valetta, Malta, 23 May 2010 Workshop programme*, page 13. Citeseer.

[Shalizi, 2010] Shalizi, C. (2010). Computing science: The bootstrap. *American Scientist*, 98(3):186–190.

[Vijaymeena and Kavitha, 2016] Vijaymeena, M. and Kavitha, K. (2016). A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2):19–28.

[Wells, 1993] Wells, G. (1993). An introduction to neural networks. In *Application of Artificial Intelligence in Process Control*, pages 164–200.

Appendix

A Classifier results scenario 1

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.93	0.92	0.89	0.96
SVM Polynomial	0.97	0.97	0.95	0.98
SVM Radial	0.98	0.99	0.95	0.99
SVM Weighted	0.97	0.97	0.95	0.98
KNN	0.95	0.96	0.91	0.98
Neural_Network	0.94	0.94	0.89	0.97
Classification Tree	0.96	0.97	0.91	0.98
Random Forest	0.98	0.99	0.96	0.99
Logistic Regression	0.94	0.92	0.89	0.96

Table 14: Context: cars

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.93	0.92	0.89	0.96
SVM Polynomial	0.97	0.97	0.95	0.98
SVM Radial	0.98	0.99	0.95	0.99
SVM Weighted	0.97	0.97	0.95	0.98
KNN	0.95	0.96	0.91	0.98
Neural_Network	0.94	0.94	0.89	0.97
Classification Tree	0.96	0.97	0.91	0.98
Random Forest	0.98	0.99	0.96	0.99
Logistic Regression	0.94	0.92	0.89	0.96

Table 15: Context: e-commerce

B Classifier results scenario 2

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.96	0.93	0.88	0.98
SVM Polynomial	0.97	0.97	0.88	0.99
SVM Radial	0.98	0.97	0.95	0.99
SVM Weighted	0.97	0.99	0.85	1.00
KNN	0.98	0.98	0.91	0.99
Neural_Network	0.97	0.97	0.85	0.99
Classification Tree	0.98	0.93	0.95	0.98
Random Forest	0.99	0.98	0.97	1.00
Logistic Regression	0.96	0.92	0.89	0.98

Table 16: Context: flour

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.94	0.92	0.89	0.97
SVM Polynomial	0.97	0.97	0.93	0.99
SVM Radial	0.98	0.98	0.95	0.99
SVM Weighted	0.97	0.98	0.91	0.99
KNN	0.96	0.96	0.91	0.98
Neural_Network	0.95	0.95	0.88	0.98
Classification Tree	0.96	0.96	0.92	0.98
Random Forest	0.99	0.99	0.96	1.00
Logistic Regression	0.94	0.92	0.89	0.97

Table 17: Average results

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.89	0.82	0.55	0.97
SVM Polynomial	0.92	0.93	0.61	0.99
SVM Radial	0.84	0.69	0.29	0.97
SVM Weighted	0.92	0.94	0.61	0.99
KNN	0.89	0.75	0.60	0.95
Neural_Network	0.89	0.79	0.56	0.97
Classification Tree	0.89	0.98	0.42	1.00
Random Forest	0.92	0.93	0.63	0.99
Logistic Regression	0.89	0.78	0.55	0.96

Table 18: Context: cars

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.89	0.82	0.55	0.97
SVM Polynomial	0.92	0.93	0.61	0.99
SVM Radial	0.84	0.69	0.29	0.97
SVM Weighted	0.92	0.94	0.61	0.99
KNN	0.89	0.75	0.60	0.95
Neural_Network	0.89	0.79	0.56	0.97
Classification Tree	0.89	0.98	0.42	1.00
Random Forest	0.92	0.93	0.63	0.99
Logistic Regression	0.89	0.78	0.55	0.96

Table 19: Context: e-commerce

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.84	0.78	0.73	0.89
SVM Polynomial	0.92	0.92	0.83	0.97
SVM Radial	0.79	0.81	0.47	0.94
SVM Weighted	0.86	0.85	0.71	0.94
KNN	0.92	0.93	0.84	0.97
Neural_Network	0.88	0.81	0.85	0.90
Classification Tree	0.90	0.94	0.74	0.97
Random Forest	0.97	0.96	0.95	0.98
Logistic Regression	0.81	0.75	0.64	0.89

Table 20: Context: flour

	Accuracy	PPV	Sensitivity	Specificity
SVM linear	0.87	0.80	0.61	0.95
SVM Polynomial	0.92	0.93	0.68	0.98
SVM Radial	0.82	0.73	0.35	0.96
SVM Weighted	0.90	0.91	0.64	0.97
KNN	0.90	0.81	0.68	0.96
Neural_Network	0.89	0.80	0.65	0.94
Classification Tree	0.89	0.97	0.52	0.99
Random Forest	0.94	0.94	0.74	0.99
Logistic Regression	0.86	0.77	0.58	0.94

Table 21: Average results

TRITA -SCI-GRU 2018:173