



UPPSALA
UNIVERSITET

IT 22 028

Examensarbete 30 hp
Maj 2022

AUTOMATED ACCOUNTING USING MACHINE LEARNING

Meenal Pathak

Institutionen för informationsteknologi
Department of Information Technology



UPPSALA
UNIVERSITET

**Teknisk- naturvetenskaplig fakultet
UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

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

Abstract

AUTOMATED ACCOUNTING USING MACHINE LEARNING

Meenal Pathak

The advancements in machine learning and artificial intelligence have touched all the traditional professions. Accountancy is changing and developing as a result of technology and it's giving rise to a new domain called Accounting Engineering. Invoice processing is a part of accounting jobs that involves a human finding and processing the information in the invoices. After reading the required data from the invoices, the accountants classify the invoices into various accounts. In this project, I extracted the data from invoice images and explored a few classifiers that will consume this data and categorize the invoices into the target accounts.

I have experimented with support vector machine, logistic regression, recurrent neural networks and random forest models, along with text encodings like TF-IDF and count vector. With limited availability of data, the maximum accuracy attained by the classifiers was 81%, around 22% improvement over the baseline. With access to more training data, these methods could prove to be a promising platform for further research.

Keywords: Natural language processing, automation of accounting processes, text encoding, word embeddings, Recurrent Neural Networks, LSTM, Random Forest, SVM, regression

Handledare: Max Block
Ämnesgranskare: Antonio Horta Ribeiro
Examinator: Salman Toor
IT 22 028
Tryckt av: Reprocentralen ITC

Acknowledgements

I'd like to thank Dr. Ali Basirat and Dr. Antonio Horta Ribeiro (Uppsala University) and Mr. Max Block (Data Ductus, Uppsala), the project supervisors, for their constant help, guidance and their constructive feedback throughout every stage of this thesis project. The completion of the thesis would not have been possible without their support.

Meenal Pathak

Table of Contents

Abstract	3
Acknowledgements	5
List of Tables	9
List of Figures	10
1 Introduction	11
2 Background	12
2.1 Natural Language Processing	12
2.2 Recurrent Neural Networks	13
2.2.1 RNN Equation and Explanation	14
2.3 LSTM - Long Short Term Memory	15
2.4 Word Representation	16
2.4.1 Frequency Based Embedding - Count Vector	16
2.4.2 Frequency Based Embedding - TF-IDF Vector	17
2.4.3 One Hot Encoding	18
2.5 Ensemble Learning	19
2.5.1 Random Forest	20
2.5.2 Stacking	21
2.6 Logistic Regression	21
2.6.1 One Vs Rest Classification	22
2.6.2 Levenshtein Distance	22
3 Literature Review	24
4 Methodology	27
4.1 Data Description	27
4.2 Data Preparation	27
4.3 Data Cleanup	28
4.4 Baseline Models	31
4.5 Descriptions - 4 architectures	32
4.5.1 Models taking only the invoice text as the input:	32
4.5.2 Models taking Supplier ID and Invoice Text as Inputs:	33

5 Results	36
5.1 Architecture 1	37
5.1.1 Baseline	37
5.1.2 Architecture 1.1 Summary	37
5.2 Architecture 2	42
5.2.1 Baseline	42
5.2.2 Architecture 2.1 Summary	42
5.2.3 Architecture 2.2 Summary	43
5.2.4 Architecture 3 Summary	47
5.2.5 Summary of results:	51
6 Discussion & Analysis	53
7 Conclusion and Future Work	55
References	56
Appendices	59
A [What is the structure of BAS Chart of Accounts]	59

List of Tables

Table 2.1: Count Vector for the Corpus of Documents	17
Table 5.1: Top 10 most frequent cost accounts in the corpus	37
Table 5.2: Confusion Matrix: Architecture 1 : Logistic Regression (TF-IDF) . . .	38
Table 5.3: Confusion Matrix: Architecture 1 : Logistic Regression (Count Vector)	38
Table 5.4: Confusion Matrix: Architecture 1 : SVM (TF-IDF)	39
Table 5.5: Confusion Matrix: Architecture 1 : SVM (Count Vector)	39
Table 5.6: Confusion Matrix: Architecture 1 : Random Forst (TF-IDF)	40
Table 5.7: Confusion Matrix: Architecture 1 : Random Forest (Count Vector) . .	40
Table 5.8: Confusion Matrix: Architecture 1 : Recurrent Neural Network	41
Table 5.9: Architecture 2 : Accuracy of Logistic Regression Models	43
Table 5.10: Confusion Matrix: Architecture 2 : Logistic Regression (TF-IDF) . . .	43
Table 5.11: Confusion Matrix: Architecture 2 : SVM (Encoding of supplier ID + Invoice Text)	44
Table 5.12: Confusion Matrix: Architecture 2 : SVM (Binary conversion of Supplier ID)	44
Table 5.13: Confusion Matrix: Architecture 2 : Random Forest (Encoding of sup- plier ID + Invoice Text)	45
Table 5.14: Confusion Matrix: Architecture 2 : Random Forest (Binary conversion of Supplier ID)	45
Table 5.15: Confusion Matrix: Architecture 2 : Recurrent Neural Network	46
Table 5.16: Confusion Matrix: Architecture 3 : Logistic Regression (TF-IDF) . . .	48
Table 5.17: Confusion Matrix: Architecture 3 : Logistic Regression (Count Vector)	48
Table 5.18: Confusion Matrix: Architecture 3 : SVM (TF-IDF)	49
Table 5.19: Confusion Matrix: Architecture 3 : SVM (Count Vector)	49
Table 5.20: Confusion Matrix: Architecture 3 : Random Forest (TF-IDF)	50
Table 5.21: Confusion Matrix: Architecture 3 : Random Forest (Count Vector) . .	50
Table 5.22: Confusion Matrix: Architecture 3 : Rrecurrent Neural Network	51
Table 5.23: Summary of Results	52

List of Figures

Figure 2.1: Applications of Natural Language Processing	13
Figure 2.2: Loopback in Recurrent Neural Networks	14
Figure 2.3: LSTM Cell	15
Figure 4.1: Top 100 Most Frequent Words in the Corpus	30
Figure 4.2: Vocabulary reduction with data preprocessing	31
Figure 4.3: Architecture 1 : Model with only invoice text as input	32
Figure 4.4: RNN Layers	33
Figure 4.5: Architecture 2 : Model having input as supplier ID concatenated to invoice text	34
Figure 4.6: Architecture 2 : Model having supplier ID and invoice text as Inputs	34
Figure 4.7: Architecture 3 : One Model per Supplier	35
Figure 5.1: Architecture 1: Accuracy of RNN model as number of Epochs increase	41
Figure 5.2: Architecture 1: Time taken for training by each epoch of RNN model as the training proceeds	42
Figure 5.3: Architecture 2: Accuracy of RNN model as number of Epochs increase	46
Figure 5.4: Architecture 2: Time taken for training by each epoch of RNN model as the training proceeds	47
Figure 5.5: Architecture 3: Accumulated Accuracy v/s Number of Suppliers . . .	51

1 Introduction

Invoice processing traditionally involves a human finding out relevant information from the given piece of paper. Thanks to the recent advances in technology, papers can be processed without human intervention. The benefits of using a specialised software and automating such tasks include faster processing, higher accuracy and overall lower cost. [1] [2]

The invoice data used in this project is used in cost accounting of companies. Cost accounting is reporting and analysing a company's cost structure. It involves assigning cost to various cost objects that involve the products, infrastructure and the services that a company uses for its daily operations. There is a cost account mapped to each cost object.

I have evaluated various ML models to classify the invoices into correct cost accounts. Each account corresponds to a class of cost to the company such as salary, travel cost, infrastructure cost etc. The data/invoices required for the experiments comes from real companies. There is a standard set of accounts/chart of accounts or BAS that is used in Sweden. I have used BAS for the classification purposes.

The naive way of doing this task assumes that each supplier has only one associated cost account. I calculated my baseline model based on this assumption. But in practice, one supplier may correspond to multiple cost accounts. The models I developed aspire to classify the invoices in correct cost accounts based on the text on them, in spite of them belonging to the same supplier.

The workflow starts with extracting the text from the images of these invoices in the javascript object notation (JSON) format. This text is preprocessed and encoded in various formats before using it to train the machine learning models.

I have created 3 different architectures based on the kind of input data I have used in them. I have tried these architectures on 4 ML models: Logistic Regression, Support Vector Machine, Random Forest and Recurrent Neural Network. I have compared the results given by these models in combination with different text endings and architectures used.

2 Background

2.1 Natural Language Processing

Natural language processing is a branch of machine learning that lets computers understand and manipulate human languages. [3] History of NLP dates back to seventeenth century. But, due to the availability of Big Data and advanced computing power, the technology has accelerated it rapidly.

With the rise of social media, Internet of Things and automation, a massive amount of data gets generated every day. Analysis of this data without the use of computer is a stupendous task. But using the current NLP technology, this task can be done by the computer tirelessly and accurately.

A natural human language comprises of words while a computer understands the language of ones and zeros at the most basic level. The abstract and ever changing way of the use of a human language and the complex, diverse and flexible nature of the expressions (e.g. sarcasm) make the natural language processing problems difficult for the machines to implement.

Some of the popular applications of Natural Language Processing include machine translations, sentiment analysis, chat-bots, automatic summarising, spam filtering etc.

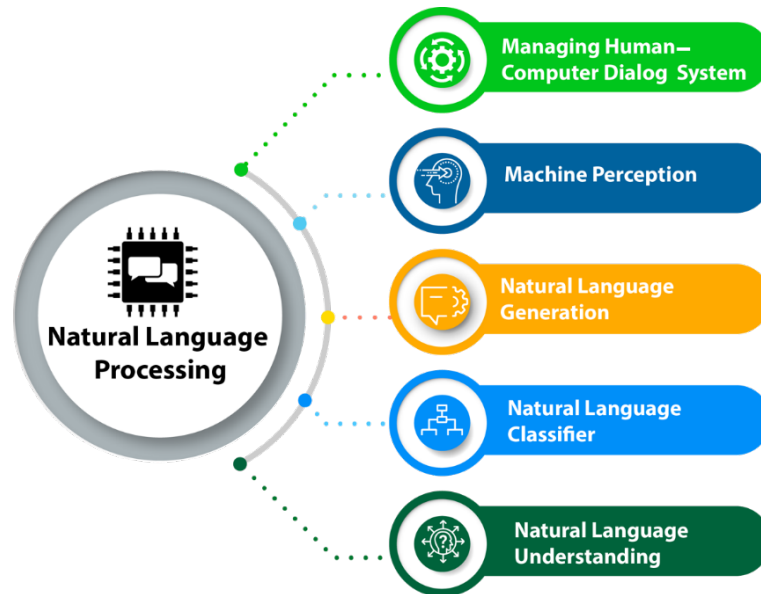


Figure 2.1: Applications of Natural Language Processing

2.2 Recurrent Neural Networks

Recurrent Neural Networks are a type of neural networks where output of the previous step is used as an input in the current step. So, in a way, the recurrent neural networks have 'memory' of all the previous inputs, which is used to calculate the current output.

In traditional feed-forward networks, all the inputs and outputs are assumed to be independent of each other. Such network models have a straightforward association between inputs and outputs. They are largely used in pattern recognition.

On the other hand, in recurrent networks (also called as feedback networks) the output of a step can be looped back allowing flow in backward direction as well. Such networks are very powerful and can get extremely complicated. [4]

One of the powerful features of recurrent neural networks (RNNs) is their ability to process sequences. A vanilla neural network can only work with fixed sized inputs and outputs. Recurrent Neural Networks can work with sequences of any size. This makes them useful in applications in natural language processing like sentiment analysis (sequence classification), machine translation, text generation etc.

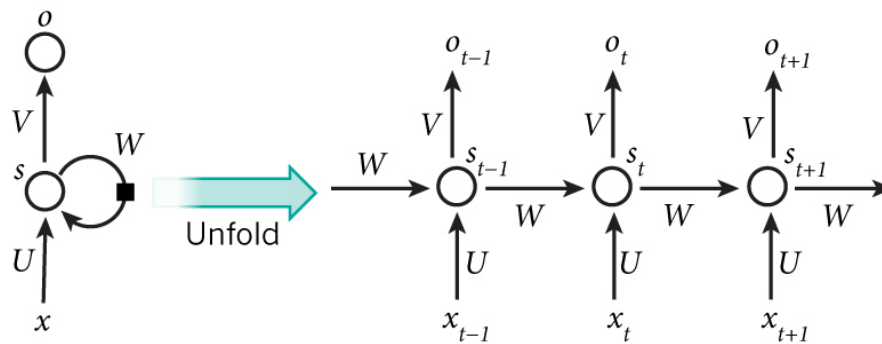


Figure 2.2: Loopback in Recurrent Neural Networks

2.2.1 RNN Equation and Explanation

As shown in the diagram above, the hidden layer in RNN has a loop coming from the past, affecting the current output. Consider an input sequence X . $X(t)$ is the vector at time t . The new state can be calculated using the recurrent formula at every time step.

$$h_t = f_W (h_{t-1}, x_t)$$

where h_t is the state at time t , f_W is a function of weight W , In RNN, unlike in ANN, weight W is shared across time.

RNNs suffer from Vanishing Gradient or Exploding Gradient problem. [5]

Gradient descent algorithm is used to calculate global minima of the cost function used. The error is propagated back to the neurons in a NN to update the weights, i.e. to carry out the learning process. In RNNs, this error updates weights for all neurons (all time steps). So the values $x(t)$ are multiplied by w_{rec} (weight recurring) multiple times in the chain updates. If the w_{rec} value becomes small, multiplying by that value multiple times makes the gradient diminish to almost zero. This is called vanishing gradient problem. On the other hand, if the w_{rec} value becomes large, multiplying by w_{rec} multiple times makes the gradient very large. This is known as the exploding gradient problem.

If the gradient becomes too small, the weights get updated very gradually, making the learning process very slow.

To avoid the exploding gradient problem, we can put a maximum limit on the gradient value or stop back-propagating after a threshold. We can also penalise or reduce the gradient value artificially.

LSTMs or GRUs are used to tackle the problem of vanishing gradients. RNNs also suffer from a problem of short term memory. LSTMs are the solution to both these problems.

2.3 LSTM - Long Short Term Memory

LSTM or Long Short Term Memory "remembers" the previous data and states to give the future states of the RNN. It is specifically designed to handle sequential prediction problems. The LSTM or the memory cells contain gates and weights. There are three types of gates which are used in LSTM: input gate, forget gate and output gate.

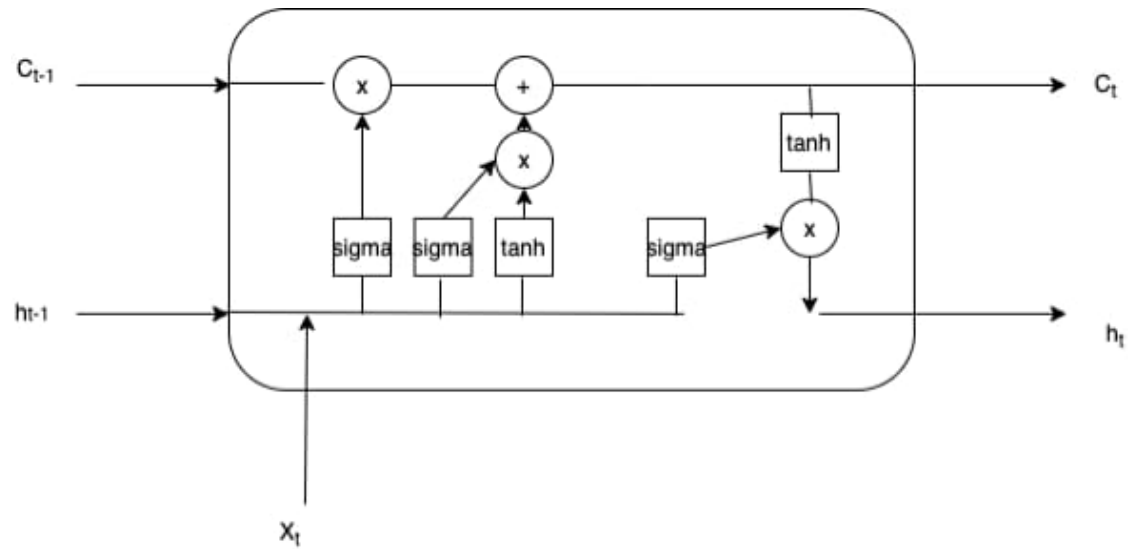


Figure 2.3: LSTM Cell

C_t - Cell state

C_{t-1} - previous cell state

h_t - output

h_{t-1} - previous output

sigma - sigmoid activation function

tanh - hyperbolic tangent activation function

X_t - input data

The forget gate removes unnecessary information before merging it with the cell state. It has 2 inputs: previous cell state and the input data. The sigmoid function removes the unwanted information and merges the rest of the information with the cell state by multiplication. The input gate adds the information to the cell state. It uses tanh activation function to create a vector of the information.

The output gate calculates the output value based on the forget gate and the input gate, multiplies the values and gives the cell output.

Thus, LSTM models, when fed with the sequential data, use the previous data and decisions to make new decisions, by making an educated guess.[6]

2.4 Word Representation

The words are required to be represented as numbers before passing them as inputs to an algorithm, since the string operations are limiting and expensive. Many machine learning algorithms and almost all deep learning architectures cannot process text in its raw form. [7] [8] [9] Thus, word embeddings convert the text into numerical vectors.

The word embeddings can be broadly classified into the following types:

- Frequency Based Embeddings

As the name suggests, this type of embeddings take the frequency of a word in the text corpus into consideration while deciding the numerical representation of the word. These word representation schemes show the importance of a word in the text corpus. But they do not demonstrate semantic relation between the words. These embeddings are designed using deterministic methods. Following frequency based word embeddings are generally used:

- Count Vector
- TF-IDF Vector
- Co-occurrence Vector

- Prediction Based Embeddings

These embeddings assign a vector representation to a word, hence the name 'Prediction Based Embeddings'. Here, semantic similarity between the words is taken into consideration.[10] Thus the vectors assigned to semantically similar words have closer proximity in the geometric space. These embeddings are trained on the text corpus using supervised neural networks. [11] Following are some of the popular prediction based word embeddings:

- Word2Vec
- ELMo
- GloVe
- SGNS

2.4.1 Frequency Based Embedding - Count Vector

This representation of words counts the occurrence of a word in a document and in the corpus. Suppose S is a set of all the words in the corpus, D_1, D_2, \dots, D_n are the documents in it and T_1, T_2, \dots, T_m are the terms (words) in the set S . Let's say $f_{11}, f_{21}, \dots, f_{n1}$ represent the frequency of these terms in document D_1 , $f_{12}, f_{22}, \dots, f_{n2}$ represent the frequency of the terms in document D_2 etc. The count vector for this corpus will be created as follows:

	T_1	T_2	..	T_m
D_1	f_{11}	f_{21}	..	f_{m1}
D_2	f_{12}	f_{22}	..	f_{m2}
D_3	f_{13}	f_{23}	..	f_{m3}
.				
.				
D_n	f_{1n}	f_{2n}	..	f_{mn}

Table 2.1: Count Vector for the Corpus of Documents

[h!] As can be seen from the table, the rows of the table represent documents while the columns represent the terms. The shape of the matrix generated is $m \times n$ i.e. number of documents in the corpus multiplied by the total number of unique terms. The term T_2 as per the matrix above will be given by the vector $[f_{21} f_{22} f_{23} \dots f_{2n}]$. Vector $[f_{13} f_{23} f_{33} \dots f_{m3}]$ represents the document D_3 .

2.4.2 Frequency Based Embedding - TF-IDF Vector

TF-IDF stands for 'Term Frequency - Inverse Document Frequency'. This is another way of text representation that is frequency based. The representation is similar to the Count Vector, but instead of considering the frequency of a word in a single document, it considers the frequency of a word in the entire corpus.

The words which are generally filtered out before doing an NLP task are called as stop words. Some examples of such stop words include prepositions, pronouns, conjunctions etc. The stop words appear in all the documents. Such words do not contribute to the semantics that a model needs to learn. These words have higher frequency than the words that are actually important for the task the model needs to perform. A count vectorizer assigns more importance to these words due to their high frequency than to the words that should carry more weight. TF-IDF representation of the corpus tries to correct this flaw.

The term frequency is given by:

$$TF = (\text{Number of times term } t \text{ appears in a document}) / (\text{Number of terms in the document})$$

And the inverse document frequency is given by:

$$IDF = \log(N/n)$$

where N is the total number of documents in the corpus and n is the number of documents in which the given term is present.

As its apparent from the formula, a word that appears in all the documents will have its inverse document frequency value 0, making the TF-IDF value 0. The IDF values will be smaller for the words that are present in many documents of the corpus, and hence the TF-IDF value lower than those appearing in less number of documents. Thus the stop words are penalised heavily, assigning higher weight to the actual important words. [12]

2.4.3 One Hot Encoding

This is another type of representation that is done on the categorical data. Such variables contain labels and not the continuous numeric values. There is a fixed set of values that such a variable can have. The categories that this data represents may or may not have any natural ordering associated with it. This data type is also called as the nominal data type.

Some algorithms such as decision tree can work with the categorical variables without converting them. But most of the algorithms need numerical data for their efficient implementation. Thus the categorical variables have to be converted into numerical variables.

If such a data column is used as the target of a machine learning algorithm, we also need a mechanism to convert those numerical values back to labels so that the user can make sense out of the predictions made by the model.

Integer encoding and one hot encoding is some of the popular ways of representing the categorical data numerically.

For certain kinds of data, representing a unique category label by an integer suffices. This scheme also lets us maintain the natural order of the variables, if present. This way of representing the data is also called as label encoding. The technique is easily reversible, hence the labels can be obtained from the integer values by simply maintaining a dictionary of label and the corresponding integer value.

Here, the representation is in the binary form. The number of bits in the representation is equal to the total number of unique values the variable can have. Thus, each value is represented by a bit. The bit representing the value of the variable is set to 1 and rest of the bits are set to 0.

For example, let's consider a categorical variable 'Education level' and the set of possible values as 'bachelor's level', 'master's level', 'doctoral level'. As the set has 3 values in total, we need three bits to construct the one hot encoded vector for all possible values of the variable.

The value 'bachelor's level' is represented as

Bachelor's Level	Master's Level	Doctoral Level
1	0	0

Similarly, the value 'master's level' is represented as

Bachelor's Level	Master's Level	Doctoral Level
0	1	0

And, the value 'doctoral level' is given as:

Bachelor's Level	Master's Level	Doctoral Level
0	0	1

Thus, the vectors 100,010,001 representing the education level values are called as one hot encoded vectors.

2.5 Ensemble Learning

Ensemble learning is a special kind of machine learning architecture where ensemble learning models combine the decisions made by more than one machine learning models to give better results. [13]

This can be achieved in various ways. Some of the simple technique for used in ensemble learning are as follows:

- **Max voting** - This technique is mostly used in classification problems. Here, each ML model used makes a prediction. This prediction is treated as a vote for the target class. The class that has maximum votes is considered as the final prediction of the ensemble learning model.
- **Averaging** - This technique can be used in regression as well as in the classification problems. As mentioned above, each model makes a prediction based on the input data. The ensemble learning algorithm combines the predictions by taking an average of all the predictions made by the individual models. For the regression problems, the average can be taken over the output values whereas for the classification problems, the average is taken over the probability values assigned to each class by each individual model.
- **Weighted average** - This technique is very similar to the averaging technique. But before taking the average of the predictions made by the individual models, each of them are assigned a weight based on their importance. Thus, this technique can be thought of as an extension to the averaging technique.

Some of the more advanced techniques of combining the result by the ensemble learning mode are:

- Stacking
- Bagging
- Boosting
- Blending

A number of algorithms have been developed that use the above mentioned techniques. For example, algorithms like random forest and bagging meta estimator use the bagging technique while the algorithms like AdaBoost, GBM, CatBoost use boosting technique.

2.5.1 Random Forest

Random forest is a typical example of bagging or bootstrap aggregation technique used in ensemble learning applied over the decision trees.

Bootstrapping is a method used when sample size is small. This introduces error in calculating descriptive statistical quantities like mean or variance. Bootstrapping method creates many sub-samples of the same dataset with replacement and calculates the statistical quantity over each of the sub-sample. Then taking average of the collected values of the quantity gives better results.

Bootstrap Aggregation [14] or Bagging is a technique of training and fitting machine learning models on each of the sub-samples created by the bootstrapping method and creating an ensemble learning model out of them by aggregating the predictions.

Decision tree is an algorithm that has high variance. If the training data for a decision tree is changed, the tree is built differently and the predictions change. Thus, the decision trees are sensitive to the training data.

Bagging technique reduces the variance of the high variance models like decision trees. The individual decision trees created using the sub-samples have both high variance and low bias. They grow deep and they are not pruned. But the resulting model does not overfit the data.

The bagging decision trees created above have high correlation. They are greedy on choosing the splitting variable. Combining the results of multiple decision trees works the best if the correlation among the individual trees is 0 or very weak. Random forest technique minimizes the correlation among the results produced by the individual decision trees. [15]

2.5.2 Stacking

Stacking or stacked generalization is yet another technique in ensemble learning. Like other ensemble learning methods, stacking involves multiple machine learning models. This technique is appropriate when each of the model being used learns in a different way from the training data. Thus, these models should have low correlation.

In stacking, unlike bagging, different models are involved. These models are called as level 0 or base models. These base models could be very complex and diverse models like neural networks, support vector machines, regression, linear models or an ensemble learning model like random forest.

Apart from two or more base models, stacking involves one more model, called meta-model or level 1 model, that learns to combine the decisions made by the base models. The meta-model is said to be stacked over the base models, hence the name stacking.

Level 0 models train themselves on the training data and make their predictions. The meta-model treats these predictions as features and trains itself to provide the final prediction.

2.6 Logistic Regression

Logistic regression is algorithm used in binary or in multi-class classification problems. The algorithm works by assigning probability to the target classes. Its a predictive analysis algorithm.

Logistic regression uses Sigmoid function, also known as the logistic function. The hypothesis of the algorithm (represented as $h_{\theta}(x)$) limits it between 0 and 1.

$$0 \leq h_{\theta}(x) \leq 1$$

The function is used to map the predicted values to the probabilities. As the Sigmoid function maps any real number to a value between 0 and 1, the use of this function is motivated.

The Sigmoid function, $f(x)$, is given as:

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

For binary classification, the threshold is usually set at 0.5. The values above the threshold belong to one class while those below the threshold belong to another.

2.6.1 One Vs Rest Classification

In this thesis we are trying to classify the invoices in to various cost accounts. Since the number of total unique class accounts is more than 2, the problem is a multi classification problem.

Not all the machine learning classification models support multi-class classification. Models like perceptron, logistic regression and SVM are predictive models for binary classification. These algorithms can be extended to multinomial classification using a variety of strategies.

These techniques are broadly classified into three main categories:

1. Transformation to Binary
2. Extension from Binary
3. Hierarchical Classification

One versus Rest classification strategy comes under the category of 'Transformation to Binary' as it transforms the problem of multi-class classification into multiple binary classification problems, one classifier per class. [16]

These binary classification problems are solved using a binary classifier like Logistic Regression. Each classifier trains itself on training samples, treating the samples belonging to its class as positive samples and the other samples as negative samples. A real valued confidence score is calculate for each of the model. The model with the highest confidence score is used to predict the class.

Since this strategy creates one classifier per model, there could be performance issues associated with this strategy for the problems involving large datasets, very large number of classes or slow binary classifiers.

2.6.2 Levenshtein Distance

Levenshtein distance is a metric used for measuring the distance between two strings. Intuitively Levenshtein distance can be understood as the number of single character edits required to change one of the given strings to the other. The edits could comprise of insertions, deletions and substitutions. [17]

Mathematically, Levenshtein distance between two strings a and b is given by the following recurrence relation.

$$\text{lev}_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1, j) + 1 \\ \text{lev}_{a,b}(i, j-1) + 1 \\ \text{lev}_{a,b}(i-1, j-1) + \mathbf{1}_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Where

i - terminal character position of string a

j - terminal character position of string b

3 Literature Review

A trend of automation of various accounting processes using machine learning techniques has started relatively recently. The repetitive and monotonous tasks in accounting can be done with better accuracy and in less time using these techniques.

As per PEPPOL BIS Billing 3.0, which is the compliant implementation of the European e-Invoicing standard, all purchases in public sector are required to be electronically invoiced, starting 1st April 2019. These invoices must comply with a standard format, hence it's easier to extract data from them. The data extraction process from non-standard invoices as well as non-electronic invoices is much more laborious. Such invoices are available in image forms. Classical approach to extract information from these images includes utilisation of techniques like Optical Character Recognition (OCR). Some deterministic methods, such as position of a word on the document, are used in text extraction. In their paper on 'Extracting structured data from Invoices' [18], Xavier Holt and Andrew Chisholm explore ways of extracting information from unseen document formats using machine learning techniques. [19] explores a Conceptual model that describes the invoice domain as generically as possible. The document model is created by labeling physical rectangles. Once this model is created, its class can be unambiguously figured out from its logo.

Once the data is extracted, the problem of automation of accounting reduces to document classification problem. A lot of exploration has been done in this area over the years. [20]

Shervin Minaee et al [7] give a comprehensive review of 150 deep learning models used for text classification. The paper starts with feed forward neural networks, which are considered as the simplest of the deep learning models that could be used for text classification. Later the paper talks about Recurrent Neural Network models, starting from vanilla RNN and considering its variants, based on the use case. Next, the paper gives an overview of Convolutional Neural Networks, capsule based neural networks, memory augmented networks and ends with hybrid models. The paper also discusses the ways of deciding how to choose a model for your text classification task, by providing quantitative analysis of the performance of selected set of DL models.

In [21], the authors approached a similar problem of document classification. They used stack-overflow data to train the models. They found that many of their models

beat the human-delivered baselines. These include support vector machines, logistic regression, naive Bayes classifier along with Word2Vec encoding.

[12] gives us an overview of use of ML models for text classification. The experiments in this paper focus on various text pre-processing techniques, centered around TF-IDF and how they improve the models. In their work, authors of paper [6] have experimented with recurrent neural network models along with long short term memory for text classification. They experimented with various sizes of input datasets, and multiple optimizers. [15] explores and assesses the use of random forest models for text classification whereas [22] proposes a better random forest algorithm using feature weighting and tree selection method to categorize text documents into dozens of classes. In [23], authors capture both global and local textual semantics and model a higher order label correlation using ensemble application of convolutional and recurrent neural networks for multi-label text categorization problem.

Since I had access to only a limited amount of training data, I looked at the related work done in transfer learning. [24] gives an overview of the transfer learning mechanism and also the taxonomy of different transfer learning approaches. [25] gives an overview of modern transfer learning techniques in NLP, how the models are pre-trained, how to integrate these models in our machine learning tasks. It also presents some case studies where transfer learning is used.

Such pre-trained models are more readily available for text in English language. [26] presents and analyses the multilingual BERT. The authors Pires, Schlinger and Garrette performed a large number of experiments showing that transfer learning is possible using this multilingual BERT even to the languages having a different scripts. This multilingual BERT creates multilingual representations by creating translation pairs. [27] introduces Swedish BERT create at KBLab. The paper highlights difficulties faced by smaller languages like Swedish where there is lack of training data. This BERT is trained using KB's data collection. The test results produced by this model are compared to those of the multilingual BERT created by Google. The paper states that the Swedish BERT outperforms the multilingual BERT.

The alternative to using transfer learning in a non-English training data is to use a translator on the text corpus, to translate it in English. [28] cites models that could be used to convert text from one language to another using a recurrent neural network model. On the other hand [29] describes how the model created using AutoML Translate can produce consistent formality in language translation.

As the literature review suggests, a lot of groundwork has already been done in the domain of natural language processing and a plethora of strategies could be tried to

solve the text classification problem at hand.

We limit the scope of this thesis project to trying out four different ML models along with two different text encodings. We try out three architectures using all these models. These architectures depend upon what input data the models use and how many models are being created.

We will explore following models in this project:

- Logistic Regression
- Support Vector Machine
- Random Forest
- Recurrent neural networks

We will use following text encodings with the above mentioned models:

- Term Frequency-Inverse Document Frequency (TF-IDF)
- Count Vector

We will consider following architectures/paradigms for our explorations using the above mentioned models and text encodings:

- Models taking only invoice text as input
- Models taking invoice text and supplier ID as input - Supplier ID concatenated to the invoice text
- One model created per supplier

4 Methodology

4.1 Data Description

In this project, invoice data is processed. An image of the invoice goes through optical character recognition (OCR) to produce digital text data. This text data is processed by the ML algorithms to recognise the pattern which will help us classify the invoices into different cost accounts. Each account belongs to a class of costs such as salary account, travel costs or office supplies. There is a standard chart of accounts in Sweden called BAS. BAS accounting plan was introduced in Sweden in 1976. Majority of the companies in Sweden, small and large, use the BAS accounting plan.

The Swedish BAS chart of accounts (Basic chart), represents the generally accepted accounting principles (GAAP) and is an open to use chart of accounts for accounting in Sweden. "BAS-kontoplanen" is today the dominant chart of accounts for Swedish enterprises. Approximately 95% of all companies in Sweden use some form of "BAS-kontoplanen". BAS activities cover the maintenance, revision and improvement of the chart of accounts and related publications, as well as other activities that are of importance for the use of "BAS-kontoplanen". [30] The structure of BAS account numbers is illustrated in the appendix.

In order to restrict the scope, all data uses one of the recent BAS account plans, and has only one cost account as the correct answer. The data comes from real companies. The data is composed of 657 suppliers, 171 accounts spread over 14912 invoice documents. The suppliers and accounts have one to many relationship.

4.2 Data Preparation

The images of the invoices cannot directly be used for the classification task. The text from these images is extracted using optical character recognition. For this project, the optical character recognition service provided by Amazon Web Services is used. A Python script is written to extract the text that belongs to the tabular part of the invoices. The output of this service is the file containing plain text.

The text extraction was already done on another batch of invoices. These extracted files have the text in json format.

4.3 Data Cleanup

An invoice usually contains a lot of text. Some of the text is of no use in deciding in which cost account the invoice belongs. Forcing a machine learning algorithm to process all such text takes a lot of time, in most cases, compromising accuracy of the model. Hence some text pre-processing and cleanup is required in order to get better results and to achieve better performance. [31]

The text being used in this project is generated by the optical character recognition method. Some words are misread by the OCR program. After going through the text data, it is found that some words are misspelled. In some cases, 12 different misspelled versions of the same word are found in the corpus. That increased the number of unique tokens in the data considerably.

After exploring the raw text that was extracted from an invoice, it is found that the vowels in some of the words have accents which are not used in the Swedish language. Clearly, this is a type of misspelled words. This increased the vocabulary size considerably as the same word with as many as 5 different accents can be found in the data. To avoid this, all the vowels with accents in the text are replaced with the vowels without accents.

To compensate for the incorrectly spelled words, Levenshtein distance is used. [17] The algorithm works as follows:

1. A list is created from all unique terms in the corpus.
2. The list is sorted alphabetically.
3. Iterate through this list. For the current word, consider the last 10 words occurring before it in the list.
4. Calculate the Levenshtein distance of each of those 10 words from the current word that is being processed in the loop.
5. If the length of the current word and the words being considered from last 10 words in the list is less than 8, and the Levenshtein distance between them is less than 2, the words are considered to be the misspelled version of the same word. Levenshtein distance up to 4 is allowed for the words with length greater than 8 and less than 20, where as the distance up to 7 is allowed for the words having length greater than 20. These numbers are arrived at after going through the unique terms in the corpus.
6. All such words, that are considered to be the misspelled versions of a word, are replaced by one word in the corpus. Thus, for all those words, there will be one token in the corpus after the replacement.

Most of the invoices have words like 'Invoice', 'AB', 'Totalt' or numeric data in the top 20% of the text and words like 'OCR', 'Bankgiro' or 'betalning' pertaining to the payment information in the bottom 20% of the text. These words are present in almost

all the invoices and do not contribute any knowledge regarding the account to which the invoice belongs. Hence, top 20% and bottom 20% text from each of the invoice is removed as a part of data pre-processing.

Document Frequency Thresholding:

Document frequency thresholding is the process where frequency of each unique term from the training corpus is calculated and the term below a certain threshold or above a threshold value are removed. [32] The threshold value can be decided depending on the data and the application.

After all the above data processing, it can be seen that there are words in the data corpus which appear only once or twice in the entire corpus. Such words contribute almost nothing to the model's prediction. Hence, the words that appear once or twice are removed from the corpus.

On the contrary, there are some words which appear in every document. These are generic words and contribute nothing to the model's prediction. After exploring them, top 100 most frequent words were removed from the vocabulary.

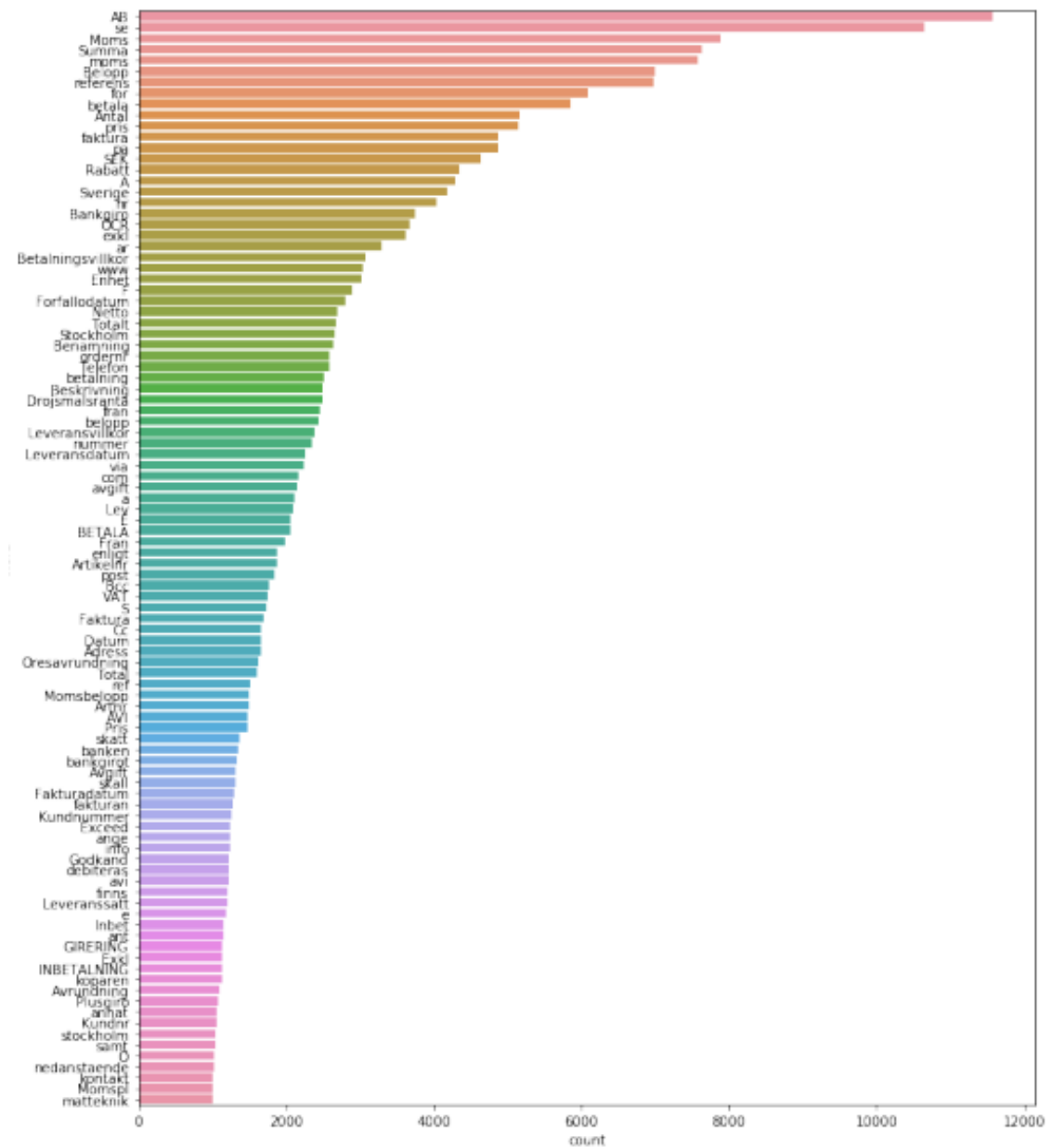


Figure 4.1: Top 100 Most Frequent Words in the Corpus

There were 45919 unique words in the corpus before preprocessing the text. After stemming the words using SnowBallStemmer from the Python nltk library, the number of unique words reduced to 38519. Further reduction in the size of the vocabulary was needed as the square matrix created by the Embedding layer of the recurrent neural network was too big to process. The model did not train itself over the successive epochs. Hence as the next step of text preprocessing, all the characters in the corpus were converted to the lower case and the accents were removed. The number of unique terms reduced to 33728 after this step. Next, after replacing the terms with similar

spellings by a single spelling to compensate for the wrong spellings, the size of the vocabulary reduced to 18901. Finally, after removing the terms that occur only once or twice in the entire corpus, the total number of unique words in the corpus reduced to 13799. Thus, the vocabulary size was successfully reduced from 45919 to 13799.

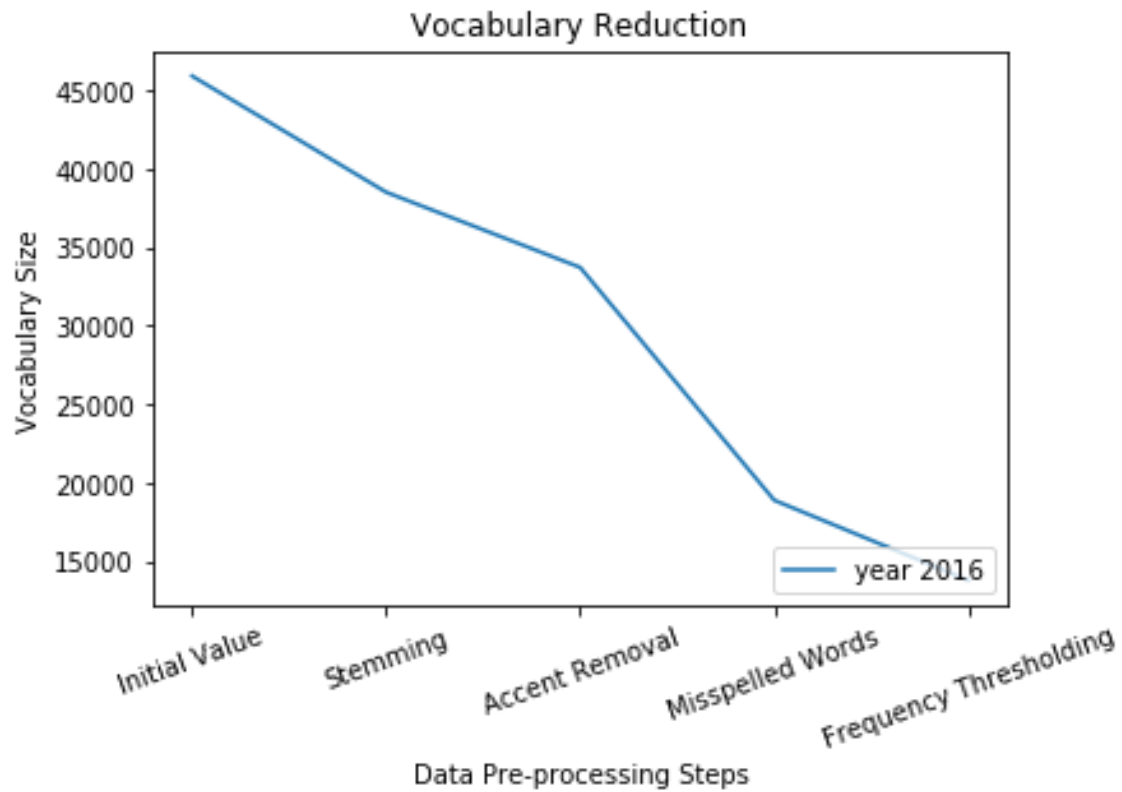


Figure 4.2: Vocabulary reduction with data preprocessing

4.4 Baseline Models

A method that can be used to speed up the accounting process without using any machine learning can be used as a baseline for these experiments.

Two inputs are being considered to make the predictions by the models. First one is the supplier ID and the second one is the text of an invoice. We have developed the models that consider both the inputs and the models that take only invoice text as the input.

For the machine learning models that take only the invoice text as the input, the baseline algorithm is defined as the majority selection. The algorithm finds out the most frequently used account from the training data and uses that account ID as the predicted account ID for each of the data points.

For the machine learning models that take both the invoice text and the associated supplier ID as the inputs, the baseline algorithm is defined as the majority selection per supplier. The algorithm finds out the most frequently used account by a supplier. It predicts this account for each of the data points that belong to the same supplier.

4.5 Descriptions - 4 architectures

In this project I explored 4 different architectures on the following machine learning algorithms along with one versus all classification strategy[21]:

- Logistic Regression
- Support Vector Machine
- Random Forest
- Recurrent Neural Network

As mentioned above, the machine learning model takes either only the invoice text as the input or it takes both the invoice text and the supplier ID associated with that invoice for prediction of the account.

4.5.1 Models taking only the invoice text as the input:

In this strategy only one machine learning model from logistic regression, support vector machine and random forest is created which takes the entire text corpus as the input. The model is trained on two types of encoded text - TF-IDF vector and Count Vector. The target variable - Account ID, is encoded using Label Binarizer, giving one hot encoded vector for the Account IDs in the training data.

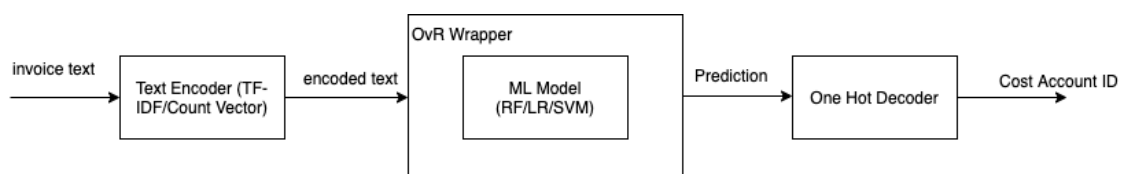


Figure 4.3: Architecture 1 : Model with only invoice text as input

The fourth model belonging to the same category is a recurrent neural network. This model converts the input text into a sequence of integers using the `text_to_sequences` method from Keras library. The target variable, i.e. the Account ID is converted into one hot encoded vector using LabelBinarizer. The recurrent neural network trains on this encoded text to predict the account ID.

The recurrent neural network model has an Embedding layer, a spatial drop out of

20%, a bidirectional LSTM layer of size 100 and three dense layers of neurons of sizes 32, 64 and 168 respectively. The first two dense layers use 'ReLU' activation function while the last layer, which is the output layer uses 'softmax' activation function. On this model, loss is calculated as categorical cross entropy, the optimizer used is the 'adam' optimizer and the metric used is 'accuracy'.

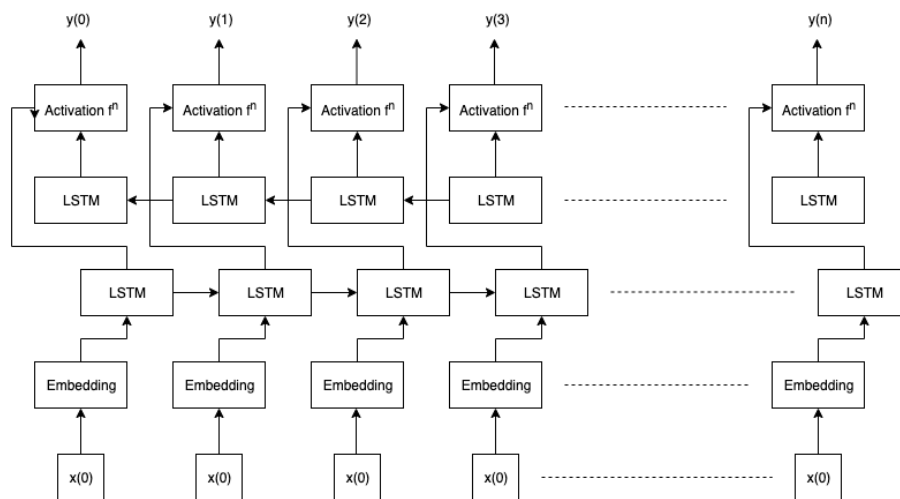


Figure 4.4: RNN Layers

4.5.2 Models taking Supplier ID and Invoice Text as Inputs:

In this category, I explored three different architectures.

Stacking

In this architecture, the stacking ensemble learning approach is explored. There are two level 0 logistic regression models. The first model takes supplier ID as the input while the second one takes the invoice text as the input. The target variable for both the models is the account ID.

The predictions of the level 0 model are used as features by the meta model. The meta model is a stacking classifier that uses logistic regression model. The approach uses one versus rest strategy for multi-class classification as in case of the rest of the models.

This strategy showed accuracy below the baseline, hence it's not explored further using other machine learning models.

Concatenation

In this architecture, the supplier ID is concatenated at the beginning of the invoice text before the model processes it. The classifier used in this approach is a logistic regression model along with the one versus rest classification strategy.

In this approach, four different combinations of the text encoding are tried. At first, the entire concatenated string is encoded using TF-IDF vectorizer. Second encoding involves using CountVectorizer to encode the entire concatenated string.

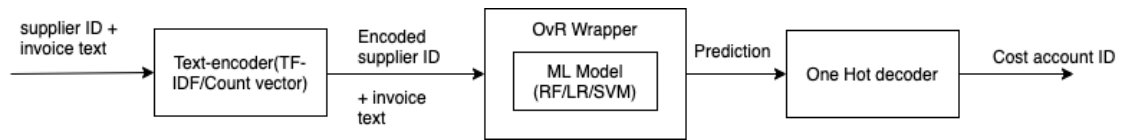


Figure 4.5: Architecture 2 : Model having input as supplier ID concatenated to invoice text

In the third and fourth approaches, the supplier ID string is converted into one hot encoded vector using LabelBinarizer. The invoice text is converted into TF-IDF vector or into Count vector to concatenate with the encoded supplier ID.

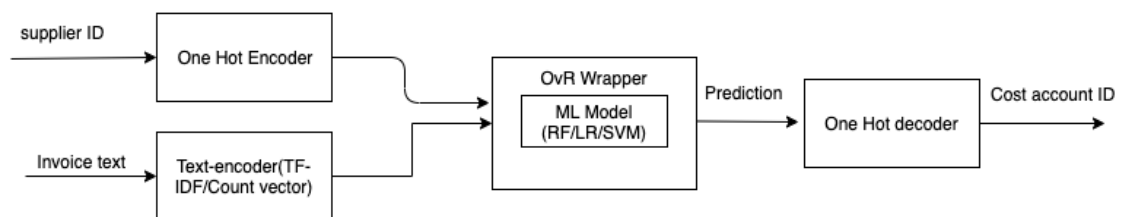


Figure 4.6: Architecture 2 : Model having supplier ID and invoice text as Inputs

One Model Per Supplier

In this architecture, one multi-class classifier is created for each supplier. Thus, each model takes the text that belongs to the invoices associated with the particular supplier only, as input. For logistic regression and support vector machine, one versus rest strategy of classification is used for multiclass classification. Similar to the previous approaches, this architecture is also tested with two types of text encoding - TF-IDF vector and Count Vector. With recurrent neural network, the input text is converted into a sequence of integers using the `text_to_sequences` method from Keras library. The target variable, i.e. the Account ID is converted into one hot encoded vector using LabelBinarizer. Each recurrent neural network trains on this encoded text to predict the account ID.

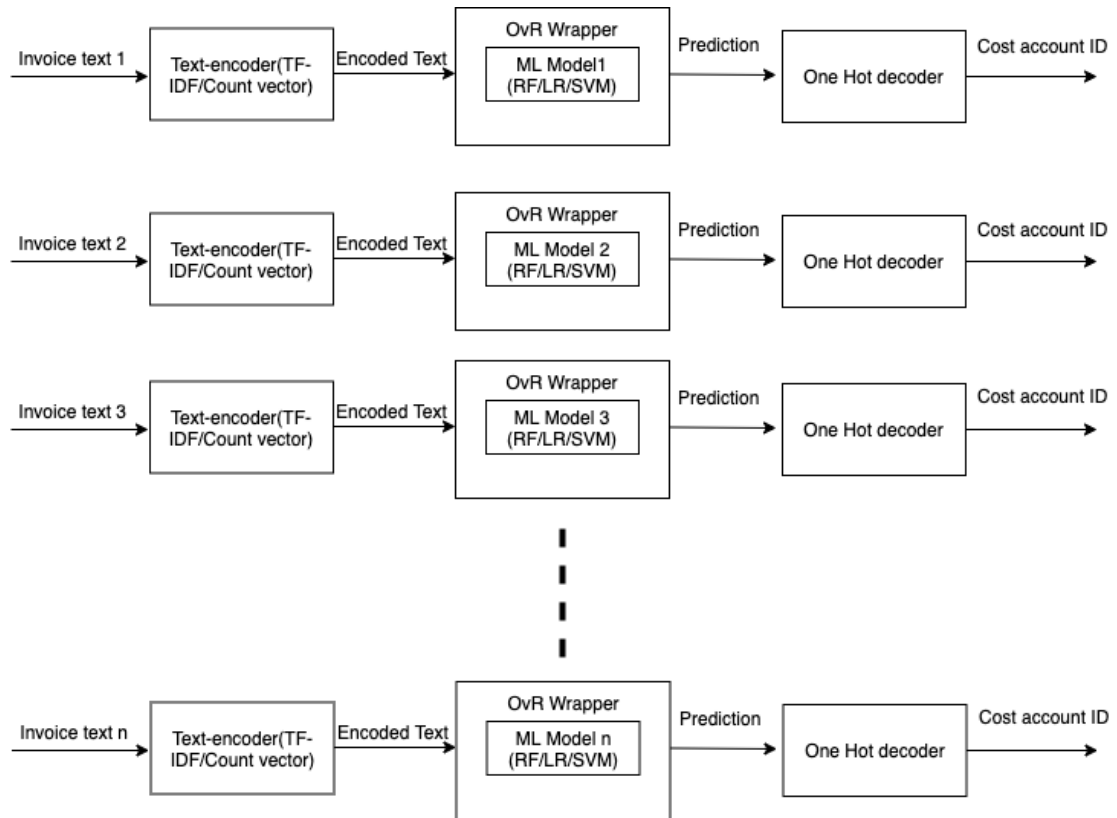


Figure 4.7: Architecture 3 : One Model per Supplier

In this architecture, accuracy of each model is measured individually. The aggregate accuracy of this architecture is calculated as follows:

$$\text{Aggregate Accuracy} = \sum_{i=1}^N A_i * N_i$$

Where

N - Total number of suppliers

A_i - Accuracy of the model for i^{th} supplier

N_i - Number of records used to train the model for the i^{th} supplier

5 Results

The experiments are performed on a laptop with the following configurations:

Processor : Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz 2.90 GHz x64-based processor

RAM : 16 GB

Libraries used:

Library	Version
keras	2.3.1
sklearn	0.22.2.post1
matplotlib	3.1.3
seaborn	0.10.0
nltk	3.4.5
numpy	1.18.1
pandas	1.0.1
alphabet detector	0.0.5

TF-IDF vectorizer uses following parameters. The parameters are adjusted after running the models multiple times to get highest possible accuracy.

max_df - This parameter decides the frequency thresholding. The terms that appear too frequently in the corpus are removed before encoding the text. The parameter is set to 0.8 i.e. the terms appearing in more than 80% documents are removed.

max_features - The parameter, when set to N, takes into account the top N features ordered by term frequency. The frequency is considered across the corpus. The value for this parameter is set to 800 after trial and error to attain maximum possible accuracy.

All the architectures, as mentioned in the Methodology section are experimented with, using following machine learning models:

- Logistic Regression
- Support Vector Machine
- Random Forest
- Recurrent Neural Network

For Logistic Regression, Support Vector Machine and Random Forest, models in the scikit-learn library are used. The Recurrent Neural network model is built as described

in the methodology section.

Two text encoding schemes - TF-IDF and Count vector - are used in the experiments. The dataset consists of 168 labels i.e. 168 cost accounts. To evaluate the models, confusion matrix for top 10 most frequently used cost accounts is used.

Top 10 most frequent accounts with their frequency counts are as follows:

Cost Account	Frequency
4010	3955
1790	1072
4000	621
7690	576
6072	511
5020	415
4041	385
6212	371
5420	349
6250	347

Table 5.1: Top 10 most frequent cost accounts in the corpus

5.1 Architecture 1

In the first architecture, the model has only the invoice text to base its predictions upon.

5.1.1 Baseline

The baseline model is built on the majority selection. This strategy showed the validation accuracy of 26.52%.

5.1.2 Architecture 1.1 Summary

This architecture shows considerable improvement in the accuracy over the baseline.

Logistic Regression Model:

The model shows 45.9% validation accuracy with TF-IDF scheme of text encoding. Confusion Matrix for the top 10 most frequent account in the data set:

670	11	13	0	0	0	0	0	0	0
0	99	0	0	0	0	2	0	0	0
2	0	97	0	0	0	0	0	0	0
2	0	0	36	6	0	0	0	0	0
0	0	0	0	38	0	0	0	1	0
0	0	0	0	0	53	0	0	0	0
0	0	0	0	0	0	15	0	0	0
0	0	0	0	0	0	0	41	0	0
0	0	0	0	2	0	0	0	6	0
0	0	0	0	0	0	0	0	0	45

Table 5.2: Confusion Matrix: Architecture 1 : Logistic Regression (TF-IDF)

The model shows a big improvement in the accuracy when used along with count vector text encoding. The validation accuracy measures up to 67.8%.

Confusion Matrix for the top 10 most frequent account in the data set:

713	8	14	0	0	0	0	0	0	0
1	144	0	0	0	0	0	0	1	0
2	0	110	0	0	0	0	0	0	0
1	0	0	80	2	0	0	0	1	0
0	0	0	4	56	0	0	0	3	0
0	0	0	0	0	72	0	0	0	0
0	0	0	0	0	0	60	0	0	0
0	0	0	0	0	0	0	49	0	0
0	0	0	0	6	0	0	0	26	0
0	0	0	0	2	0	0	0	0	48

Table 5.3: Confusion Matrix: Architecture 1 : Logistic Regression (Count Vector)

Support Vector Machine

The same experiment when ran for support vector machine shows the validation accuracy of 56.86% for TF-IDF text encoding.

The confusion matrix looks as follows:

685	9	13	0	0	0	0	0	0	0
1	108	0	0	0	1	2	0	0	0
1	0	104	0	0	0	0	0	0	0
3	0	0	56	6	0	0	0	0	0
0	0	0	1	43	0	0	0	1	0
0	0	0	0	0	69	0	0	0	0
0	0	0	0	0	1	33	0	0	0
0	0	0	0	0	0	0	50	0	0
0	0	0	0	1	0	0	0	9	0
0	0	0	0	0	0	0	0	0	47

Table 5.4: Confusion Matrix: Architecture 1 : SVM (TF-IDF)

The validation accuracy of the same model increases to 70.5% when used with Count Vector encoding of the text.

Confusion Matrix is as follows:

714	8	16	0	0	0	2	0	0	0
2	153	0	0	0	1	2	0	1	0
4	1	110	0	0	0	0	0	0	0
2	0	0	88	5	0	0	0	2	0
0	0	0	5	62	0	0	0	5	0
0	0	0	0	0	72	0	0	0	0
0	0	0	0	0	1	66	0	0	0
0	0	0	0	0	0	0	48	0	0
0	0	0	2	7	0	0	0	31	0
0	0	0	0	1	0	0	0	0	47

Table 5.5: Confusion Matrix: Architecture 1 : SVM (Count Vector)

Random Forest: The experiment when ran for support vector machine shows the validation accuracy of 64.34% for TF-IDF text encoding.

The confusion matrix looks as follows:

704	9	11	1	0	1	1	0	0	0
1	134	0	0	0	1	0	0	2	0
4	1	105	0	0	0	0	0	0	0
1	0	0	81	5	0	0	0	2	0
0	0	0	5	51	0	0	0	5	0
0	0	0	0	0	71	0	0	0	0
0	0	0	0	0	1	54	0	0	0
0	0	0	0	0	0	0	45	0	0
1	0	0	2	7	0	0	0	19	0
0	0	0	0	1	0	0	0	0	47

Table 5.6: Confusion Matrix: Architecture 1 : Random Forst (TF-IDF)

The validation accuracy of the same model increases to 66.06% when used with Count Vector encoding of the text.

Confusion Matrix is as follows:

701	10	13	0	0	1	1	0	0	0
1	144	0	0	0	0	1	0	0	0
2	0	110	0	0	0	0	0	0	0
3	0	0	84	4	0	0	0	0	0
0	0	0	4	56	0	0	0	1	0
0	0	0	0	0	74	0	0	0	0
0	0	0	0	0	0	64	0	0	0
0	0	0	0	0	0	0	48	0	0
0	0	0	2	8	0	0	0	22	0
0	0	0	0	1	0	0	0	0	47

Table 5.7: Confusion Matrix: Architecture 1 : Random Forest (Count Vector)

Recurrent Neural Network: The shows the validation accuracy of 86.59% on the training set and 67.7% on the validation set.

The confusion matrix looks as follows:

723	9	13	1	2	1	2	0	0	10
1	139	3	0	0	0	3	0	2	0
4	0	108	0	2	0	0	0	0	0
5	0	2	84	9	0	1	0	3	0
0	1	3	9	58	1	0	1	13	0
0	0	0	0	0	65	0	5	0	0
0	0	0	0	3	2	59	0	1	1
0	0	0	0	0	1	0	52	0	2
2	2	0	3	15	0	3	0	25	0
1	0	0	1	0	0	0	0	0	55

Table 5.8: Confusion Matrix: Architecture 1 : Recurrent Neural Network

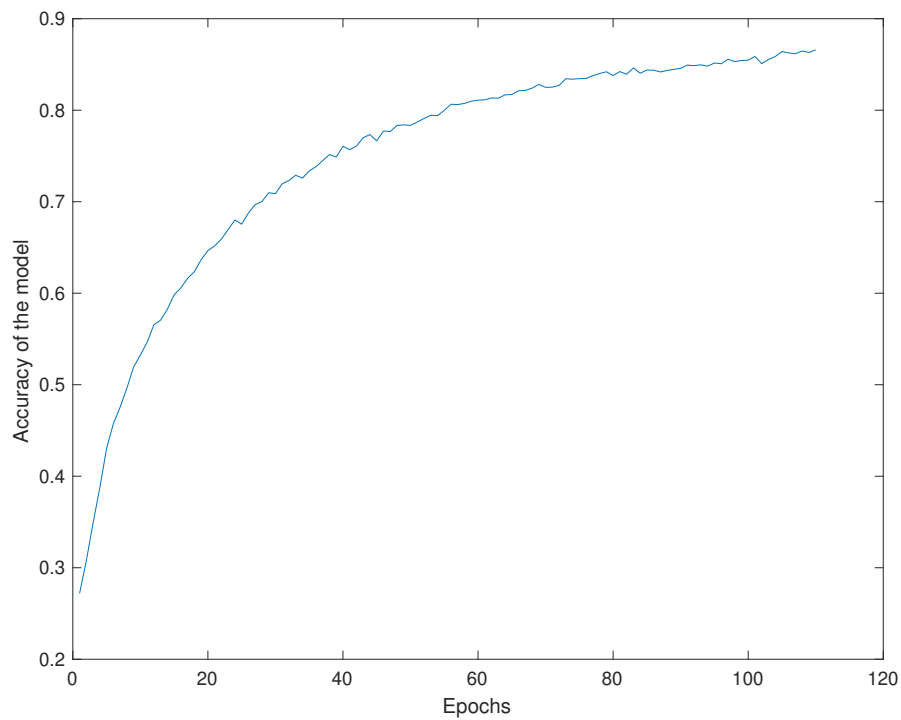


Figure 5.1: Architecture 1: Accuracy of RNN model as number of Epochs increase

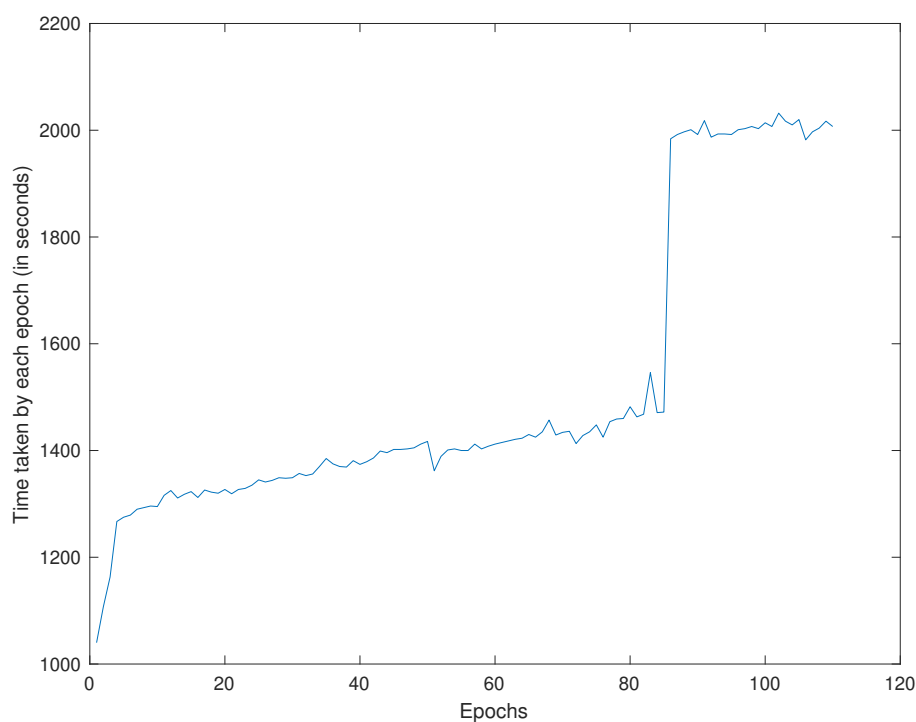


Figure 5.2: Architecture 1: Time taken for training by each epoch of RNN model as the training proceeds

5.2 Architecture 2

In this type of architectures, the invoice text is provided as an input to the model along with the ID of the supplier to whom the invoice belongs.

5.2.1 Baseline

The baseline is grounded on the basis of majority selection per supplier. This scheme showed the validation accuracy of 66%.

Following three architectures take the invoice text and supplier ID as inputs. All of the architectures are based on Logistic Regression model along with the One Versus All classification strategy.

5.2.2 Architecture 2.1 Summary

In this architecture stacking - ensemble learning approach is used. There are level 0 models. The first model takes the supplier ID as input while the second one takes the invoice text as the input to make predictions. The validation accuracy of the meta model is 51.95%, dropping by 14.05% compared to the baseline.

Since this approach shows accuracy that is lower than the baseline, it is not used to experiment with other machine learning models.

5.2.3 Architecture 2.2 Summary

Logistic Regression:

In this architecture, the supplier ID is prepended to the invoice text before passing it as an input to the logistic regression model. This strategy is tested with 4 different encoding schemes.

Supplier ID encoding	Invoice Text Encoding	Accuracy
TF-IDF	TF-IDF	70.29
Count Vectorizer	Count Vectorizer	70.29
Binary encoding	TF-IDF	70.29
Binary encoding	Count Vectorizer	70.66%

Table 5.9: Architecture 2 : Accuracy of Logistic Regression Models

Considering the maximum accuracy given by the combination of CountVectorizer and Binary encoding, this strategy improved the validation accuracy up to 70.66%, 4.66% better than the baseline.

Confusion matrix for this approach with TF-IDF encoding is as follows:

728	8	14	0	0	0	1	0	0	0
1	146	0	0	0	0	3	0	1	0
3	0	111	0	0	0	0	0	0	0
1	0	0	91	3	0	0	0	1	0
0	0	0	4	63	0	0	0	5	0
0	0	0	0	0	75	0	0	0	0
0	0	0	0	0	0	62	0	0	0
0	0	0	0	0	0	0	52	0	0
0	0	0	2	7	0	0	0	30	0
0	0	0	0	2	0	0	0	0	55

Table 5.10: Confusion Matrix: Architecture 2 : Logistic Regression (TF-IDF)

As the accuracies for TF-IDF and Count Vector are identical, and Count Vector shows better accuracy in the previous experiments, only Count Vector approach is used for the further experiments, using remaining three machine learning models.

Support Vector Machine:

In this experiment, supplier ID is prepended to the text and the entire string is encoded using Count Vectorization scheme. The validation accuracy of the approach is 73.65%. The confusion matrix looks as shown:

736	9	16	0	0	0	1	0	0	0
2	155	0	0	0	0	1	0	1	0
4	1	110	0	0	0	0	0	0	0
2	0	0	92	3	0	0	0	2	0
0	0	0	5	64	0	0	0	7	0
0	0	0	0	0	72	0	0	0	0
0	0	0	0	0	0	69	0	0	0
0	0	0	0	0	0	0	54	0	0
0	0	0	1	7	0	0	0	37	0
0	0	0	0	1	0	0	0	0	54

Table 5.11: Confusion Matrix: Architecture 2 : SVM (Encoding of supplier ID + Invoice Text)

The next experiment with support vector machine uses binary conversion of the supplier ID prepended to count vector encoded text. The validation accuracy increases to 74.09%. The confusion matrix looks as:

744	9	15	0	0	0	1	0	0	0
2	156	0	0	0	0	1	0	1	0
4	0	111	0	0	0	0	0	0	0
1	0	0	92	3	0	0	0	2	0
0	0	0	5	64	0	0	0	7	0
0	0	0	0	0	72	0	0	0	0
1	0	0	0	0	0	68	0	0	0
0	0	0	0	0	0	0	54	0	0
0	0	0	1	7	0	0	0	38	0
0	0	0	0	1	0	0	0	0	54

Table 5.12: Confusion Matrix: Architecture 2 : SVM (Binary conversion of Supplier ID)

Random Forest:

In this experiment, supplier ID is prepended to the text and the entire string is encoded using Count Vectorization scheme. The validation accuracy of the approach is 68.68%.

The confusion matrix looks as shown:

750	11	14	0	0	0	2	0	0	0
5	160	0	0	0	0	3	0	2	0
3	0	112	0	0	0	0	0	0	0
1	1	0	96	3	0	0	0	2	1
0	0	0	5	61	0	0	0	8	0
0	0	0	0	0	76	0	0	0	0
0	0	1	0	0	0	70	0	0	0
0	0	0	0	0	0	0	58	0	0
0	0	0	0	11	0	0	0	35	0
0	0	0	0	2	0	0	0	0	56

Table 5.13: Confusion Matrix: Architecture 2 : Random Forest (Encoding of supplier ID + Invoice Text)

The next experiment with random forest uses binary conversion of the supplier ID prepended to count vector encoded text. The validation accuracy increases to 69.05%. The confusion matrix looks as:

721	9	12	0	0	0	1	0	0	0
1	147	0	0	0	0	1	0	1	0
2	0	110	0	0	0	0	0	0	0
1	0	0	90	4	0	0	0	0	1
0	0	0	4	57	0	0	0	2	0
0	0	0	0	0	75	0	0	0	0
0	0	0	0	0	0	66	0	0	0
0	0	0	0	0	0	0	55	0	0
0	0	0	0	9	0	0	0	24	0
0	0	0	0	1	0	0	0	0	55

Table 5.14: Confusion Matrix: Architecture 2 : Random Forest (Binary conversion of Supplier ID)

Recurrent neural network: The recurrent neural network is trained on the text that is prepended with the supplier ID and vectorized using Tokenizer class in Keras. The vectorization is based on word count and term frequency as well as inverse document frequency. The model shows 68.3% accuracy. The confusion matrix for the top 10 most frequent accounts is as follows:

709	10	15	4	3	1	0	0	1	10
3	143	0	0	0	0	4	0	0	0
6	1	107	0	0	0	0	0	0	0
4	2	0	96	6	0	1	0	2	0
0	5	0	5	63	0	2	0	10	0
0	3	0	0	0	68	0	2	0	2
1	1	0	1	1	0	54	0	2	2
0	0	0	0	0	1	0	54	0	2
4	5	0	3	12	0	0	0	28	0
0	0	0	0	1	0	0	0	0	55

Table 5.15: Confusion Matrix: Architecture 2 : Recurrent Neural Network

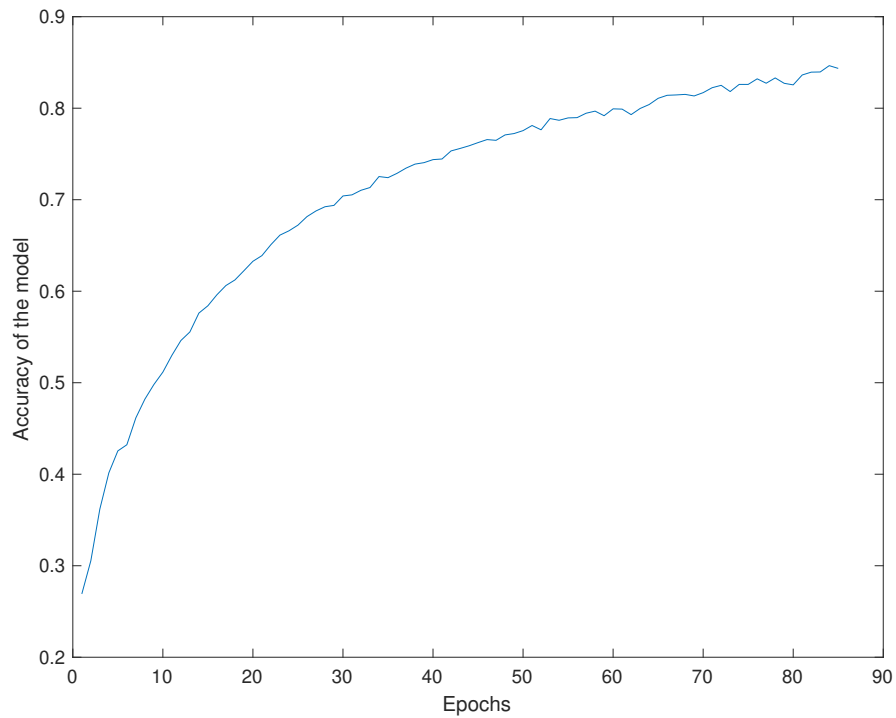


Figure 5.3: Architecture 2: Accuracy of RNN model as number of Epochs increase

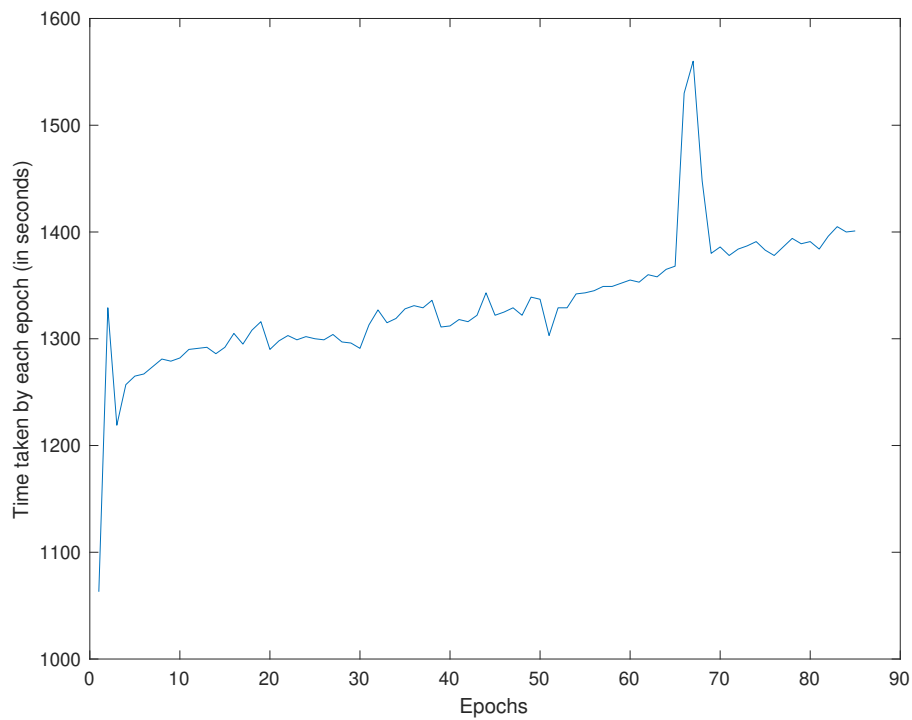


Figure 5.4: Architecture 2: Time taken for training by each epoch of RNN model as the training proceeds

5.2.4 Architecture 3 Summary

In the third architecture, one model is created for each supplier ID. The accuracy is calculated for each of the models individually. The accuracy of the architecture is calculated as the weighted average of the accuracy of all the models.

Logistic Regression:

This accuracy value reaches 74.93% for both TF-IDF encoding. The confusion matrix for the top 10 most frequently used accounts is as follows:

730	17	14	0	0	0	1	0	0	0
4	165	0	0	0	0	3	0	2	0
2	1	112	0	0	0	0	0	0	0
1	1	0	95	1	0	0	0	1	0
0	0	0	4	58	0	0	0	2	0
0	0	0	0	0	76	0	0	0	0
1	3	0	0	0	0	57	0	0	0
1	1	0	0	0	0	0	60	0	0
0	0	0	0	3	0	0	0	27	0
0	0	0	0	0	0	0	0	0	56

Table 5.16: Confusion Matrix: Architecture 3 : Logistic Regression (TF-IDF)

The validation accuracy increases further to 80.91% for count vector encoding. The confusion matrix is as shown:

750	12	14	0	0	0	3	0	0	0
4	166	0	0	0	0	3	0	2	0
4	0	111	0	0	0	0	0	0	0
1	1	0	97	2	0	1	0	2	0
0	0	0	4	65	0	0	0	5	0
0	0	0	0	0	77	0	0	0	0
0	0	1	0	0	0	73	0	0	0
0	0	0	0	0	0	0	58	0	0
0	0	0	0	8	0	0	0	40	0
0	0	0	0	1	0	0	0	0	56

Table 5.17: Confusion Matrix: Architecture 3 : Logistic Regression (Count Vector)

Support Vector Machine:

This model shows the validation accuracy of 79.67% with TF-IDF encoding of the input text. The confusion matrix looks as below:

748	12	14	0	0	0	3	0	0	0
4	170	0	0	0	0	3	0	2	0
1	1	111	0	0	0	0	0	0	0
1	1	0	97	2	0	0	0	1	0
0	0	0	4	62	0	0	0	4	0
0	0	0	0	0	77	0	0	0	0
0	2	1	0	0	0	69	0	0	0
1	0	0	0	0	0	0	64	0	0
0	0	0	0	5	0	0	0	32	0
0	0	0	0	1	0	0	0	0	55

Table 5.18: Confusion Matrix: Architecture 3 : SVM (TF-IDF)

The validation accuracy further increases to 81.317% for count vector encoding.

754	10	14	0	0	0	4	0	0	0
5	166	0	0	0	0	1	0	0	0
3	0	112	0	0	0	0	0	0	0
1	1	0	91	3	0	0	0	3	0
0	0	0	4	67	0	0	0	5	0
0	0	0	0	0	75	0	0	0	0
0	0	1	0	0	0	72	0	0	0
0	1	0	0	0	0	0	54	0	0
0	0	0	1	7	0	0	0	38	0
1	0	0	1	1	0	0	0	0	56

Table 5.19: Confusion Matrix: Architecture 3 : SVM (Count Vector)

Random Forest:

The validation accuracy of the model is 79.60% for TF-IDF text encoding. The confusion matrix for the experiment looks as follows:

750	11	14	0	0	0	2	0	0	0
5	160	0	0	0	0	3	0	2	0
3	0	112	0	0	0	0	0	0	0
1	1	0	96	3	0	0	0	2	1
0	0	0	5	61	0	0	0	8	0
0	0	0	0	0	76	0	0	0	0
0	0	1	0	0	0	70	0	0	0
0	0	0	0	0	0	0	58	0	0
0	0	0	0	11	0	0	0	35	0
0	0	0	0	2	0	0	0	0	56

Table 5.20: Confusion Matrix: Architecture 3 : Random Forest (TF-IDF)

The validation accuracy of the model reaches 80.17% for count vector encoding of the input text. The confusion matrix for the same is as shown below:

753	11	14	0	0	0	3	0	0	0
4	163	0	0	0	0	3	0	2	0
3	0	112	0	0	0	0	0	0	0
1	1	0	99	3	0	0	0	2	1
0	0	0	4	61	0	0	0	7	0
0	0	0	0	0	77	0	0	0	0
0	0	1	0	0	0	71	0	0	0
0	0	0	0	0	0	0	59	0	0
0	0	0	0	11	0	0	0	38	0
1	0	0	0	1	0	0	0	0	56

Table 5.21: Confusion Matrix: Architecture 3 : Random Forest (Count Vector)

Recurrent Neural Network Summary:

In this architecture, one recurrent neural network model is created for each supplier. The overall validation accuracy achieved by the architecture is 74.764%. The confusion matrix can be shown as:

719	20	27	0	0	0	3	1	1	0
9	150	0	0	0	0	4	0	2	0
2	3	110	0	0	0	0	0	0	0
1	1	0	99	11	0	0	0	3	0
0	0	0	5	78	0	0	0	8	0
0	0	0	0	0	77	0	0	0	0
3	0	5	0	0	0	62	0	0	0
1	0	0	0	0	0	0	50	0	0
3	0	0	2	22	0	0	0	38	0
1	0	0	0	2	0	0	0	0	56

Table 5.22: Confusion Matrix: Architecture 3 : Rrecurrent Neural Network

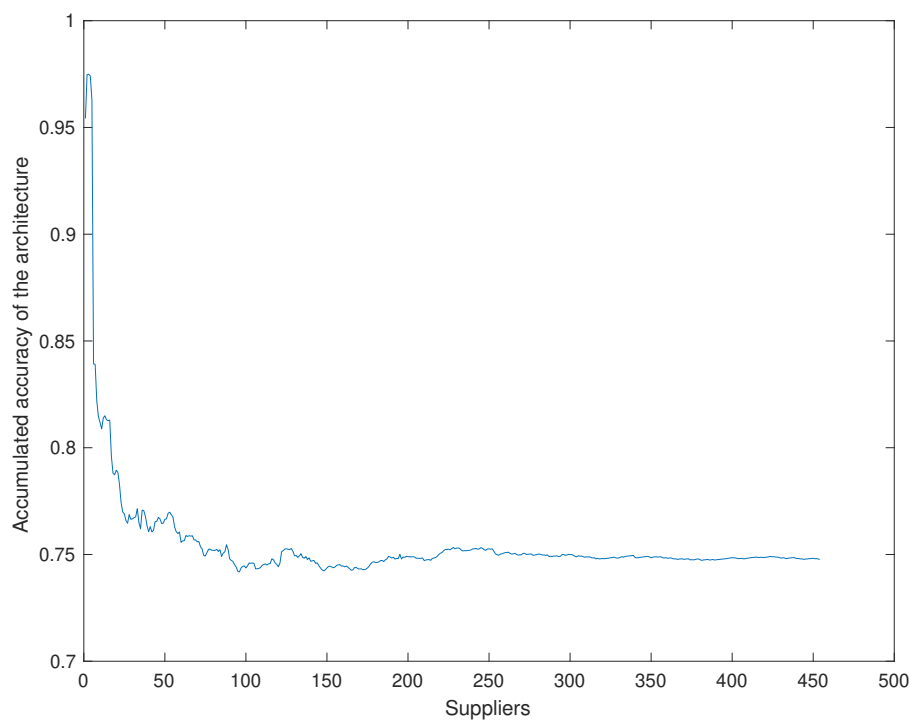


Figure 5.5: Architecture 3: Accumulated Accuracy v/s Number of Suppliers

5.2.5 Summary of results:

Architecture 1: Model with only invoice text as input (baseline 26.52%)

Architecture 2.1: Model with supplier ID concatenated with invoice text as input (baseline 66%)

Architecture 2.2: Model with supplier ID concatenated with invoice text as input, supplier ID encoded using one hot encoding (baseline 66%)

Architecture 3: One model created per supplier

Model Accuracy	Architecture 1	Architecture 2.1	Architecture 2.2	Architecture 3
Logistic Regression + TF-IDF	45.9%	70.29%	70.29%	74.93%
SVM + TF-IDF	56.86%	73.65%	-	79.67%
Random Forest + TF-IDF	64.34%	68.68%	-	79.6%
RNN + TF-IDF	67.6%	68.3%	-	74.764%
Logistic Regression + Count Vector	67.8%	70.29%	70.66%	80.91%
SVM + Count Vector	70.5%	74.09%	-	81.31%
Random Forest + Count Vector	66.06%	69.05%	-	80.17%

Table 5.23: Summary of Results

6 Discussion & Analysis

The support vector machine models consistently give the best results in all the architecture types, whereas the third architecture where one model is created per supplier gives the best accuracy for all ML models.

Considering all the experiments, the best accuracy is achieved for SVM model with Count Vector encoding, when one model is created per supplier.

One interesting observation is that all the models performed better when Count Vector encoding was used compared to the models with TF-IDF encoding. Usually TF-IDF encoding is considered a better form of text encoding as it captures the importance of words whereas count vector encoding only considers number of occurrences of a term. TF-IDF encoding penalises the terms that occur in more documents, thus removing the stop word or less important words.

Since we removed most of the stop words from our text corpus in our data pre-processing stage, TF-IDF encoder penalised important words from the text, which occurred in many documents. The words, which would have helped in classifying the invoices. On the other hand Count Vector encoding correctly assigned more importance to such words, which occurred more frequently in the documents, creating a better invoice classifier.

Another important point to note is that, in Architecture 2.2, where supplier ID is prepended to the text, the accuracy of the classifier improved when the Supplier ID was encoded using One Hot Encoding instead of Count Vectorizer along with the rest of the text. Count Vectorizer treated the supplier ID as rest of the text. But since Supplier IDs are Categorical type of data, One Hot Encoding, as expected, gave better results.

The accuracy of the models created improved as we added more information/input. i.e. the models performed far better when supplier information was provided as an input along with the text. The models performed even better when we created one model per supplier, limiting the possible output of each model to a specified set of cost accounts.

It is also important to note that the accuracy of the models improved after vocabulary reduction as it was expected.

It is interesting to note that the accuracy values for all the models change in a simi-

lar fashion as the architecture and text encoding used changes.

Since the objective of these experiments is to get invoices classified into correct cost accounts, I have measured the accuracy of all the models. But in the scenario when the accuracy values are similar, it might be worth computing other metrics to analyse the results further. It will also help to compare the proportions of first 5 or 10 majority classes.

As mentioned before, the architecture where we created one model per supplier performed the best, giving the highest accuracy. But as the number of supplier increases, the storage overhead of the models increases in this paradigm.

7 Conclusion and Future Work

In this thesis project, I worked on accounting automation, where invoices are classified into corresponding cost accounts based on the text on them, using Natural Language Processing principles.

After retrieval and pre-processing of the text, I tried various combinations of text encoding techniques, ML models along with their architectures. These models performed better than their corresponding baselines.

When only the invoice text was provided to the models as the input, Logistic Regression model gave the best accuracy whereas when supplier ID was given as an input along with the invoice text, one model per supplier architecture gave the best results. In this paradigm, Support Vector Machine model trained on text encoded using Count Vector gave the best accuracy.

To improve the classification accuracy, following are few tasks which could be performed.

1. The classifiers that are already tried could be better trained if more data is available.
2. The hyperparameters could be better tuned using a tool like Sagemaker.
3. Recurrent neural network model can be trained on the supplier information, along with the text.
4. It would be useful to the user if top three predictions of the accounts are provided.
5. Right now, the misspelled words were replaced by the other word available in the vocabulary with minimum Levenshtein distance. But a spell corrector could be used to actually correct the spelling.
6. A transfer learning tool like Stanza could be used to train the model first on the generic Swedish words. This model later could be trained on the text extracted from the invoices.
7. As there are multiple tools for transfer learning and spell correction are available for text in English language, the text could be first translated in English using one of the translator tools available. A classifier could be trained better on translated text.
8. Prediction based word embeddings could be used to capture semantic relationship among words. Such embeddings are more readily available for English language. Hence, working on translated text may let us use those embeddings like Word2Vec or BERT, which in turn may give us a more accurate classifier.

References

- [1] İ. Tekbaş, "The profession of the digital age: Accounting engineering-ifac", *IFAC Proceedings Volumes*, Sep. 2018.
- [2] S. Mohmmad, A. Hamad, H. Borgi Fendri, P. Thu, M. Sial, and A. Alhadidi, "How artificial intelligence changes the future of accounting industry", *International Journal of Economics and Business Administration*, vol. VIII, pp. 478–488, Jul. 2020.
- [3] R. Weischedel, J. Carbonell, B. Grosz, W. Lehnert, M. Marcus, R. Perrault, and R. Wilensky, "White paper on natural language processing", vol. 4, pp. 481–493, November 2003.
- [4] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network", *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, March 2020.
- [5] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem", *CoRR*, vol. abs/1211.5063, 2012. arXiv: 1211.5063.
- [6] W. Sari, D. RINI, R. Malik, and I. AZHAR, "Sequential models for text classification using recurrent neural network", January 2020.
- [7] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning based text classification: A comprehensive review", *CoRR*, vol. abs/2004.03705, 2020. arXiv: 2004.03705.
- [8] Z. Yao, Y. Sun, W. Ding, N. Rao, and H. Xiong, "Dynamic word embeddings for evolving semantic discovery", February 2018, pp. 673–681.
- [9] S. Gupta, T. Kanchinadam, D. Conathan, and G. Fung, "Task-optimized word embeddings for text classification representations", *Frontiers in Applied Mathematics and Statistics*, vol. 5, p. 67, 2020.
- [10] R. A. Stein, P. A. Jaques, and J. F. Valiati, "An analysis of hierarchical text classification using word embeddings", *Information Sciences*, vol. 471, pp. 216–232, 2019.
- [11] X. Rong, "Word2vec parameter learning explained", November 2014.

- [12] H. K. Yadla and D. P. Rao, "Machine learning based text classifier centered on tf-idf vectoriser", *International Journal of Scientific & Technology Research*, vol. 9, pp. 583–586, 2020.
- [13] P. Pintelas and I. E. Livieris, "Special issue on ensemble learning and applications", *Algorithms*, vol. 13, no. 6, 2020.
- [14] T.-H. Lee, A. Ullah, and R. Wang, "Bootstrap aggregating and random forest", in. January 2020, pp. 389–429.
- [15] D. Xue and F. Li, "Research of text categorization model based on random forests", in *2015 IEEE International Conference on Computational Intelligence Communication Technology*, 2015, pp. 173–176.
- [16] R. Rifkin and A. Klautau, "In defense of one-vs-all classification", *Journal of Machine Learning Research*, vol. 5, pp. 101–141, December 2004.
- [17] S. Zhang, Y. Hu, and G. Bian, "Research on string similarity algorithm based on levenshtein distance", *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 2247–2251, 2017.
- [18] X. Holt and A. Chisholm, "Extracting structured data from invoices", in *Proceedings of the Australasian Language Technology Association Workshop 2018*, Dunedin, New Zealand, December 2018, pp. 53–59.
- [19] F. Cesarini, E. Francesconi, M. Gori, S. Marinai, J. Sheng, and G. Soda, "Conceptual modelling for invoice document processing", in *Database and Expert Systems Applications. 8th International Conference, DEXA '97. Proceedings*, 1997, pp. 596–603.
- [20] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, D. Brown, L. Id, and Barnes, "Text classification algorithms: A survey", *Information (Switzerland)*, vol. 10, April 2019.
- [21] W. Arshad, M. Ali, M. Mumtaz Ali, A. Javed, and S. Hussain, "Multi-class text classification: Model comparison and selection", in *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 2021, pp. 1–5.
- [22] B. Xu, X. Guo, Y. Ye, and J. Cheng, "An improved random forest classifier for text categorization", *Journal of Computers*, vol. 7, December 2012.
- [23] G. Chen, D. Ye, Z. Xing, J. Chen, and E. Cambria, "Ensemble application of convolutional and recurrent neural networks for multi-label text categorization", in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2377–2383.

- [24] Z. Alyafeai, M. S. AlShaibani, and I. Ahmad, *A survey on transfer learning in natural language processing*, 2020. arXiv: 2007.04239 [cs.CL].
- [25] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf, "Transfer learning in natural language processing", in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 15–18.
- [26] T. Pires, E. Schlinger, and D. Garrette, "How multilingual is multilingual BERT?", in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 4996–5001.
- [27] M. Malmsten, L. Börjesson, and C. Haffenden, "Playing with words at the national library of sweden – making a swedish bert", Jul. 2020.
- [28] S. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain, "Machine translation using deep learning: An overview", Jul. 2017, pp. 162–167.
- [29] A. Viswanathan, V. H. Wang, and A. Kononova, "Controlling formality and style of machine translation output using automl", in *SIMBig*, 2019.
- [30] <https://www.bas.se/english/>.
- [31] Y. Y. HaCohen-Kerner Y Miller D, "The influence of preprocessing on text classification using a bag-of-words representation", *PLoS ONE* 15(5), 2020.
- [32] X.-Y. Chen, R.-L. Li, and Y.-F. Hu, "Document feature selection based on the minimum term frequency threshold", vol. 19, pp. 531–537, August 2006.
- [33] J. M. Lima and J. Maia, "A topical word embeddings for text classification", October 2018, pp. 25–35.
- [34] I. Ul Haq, I. Gondal, P. Vamplew, and S. Brown, "Categorical features transformation with compact one-hot encoder for fraud detection in distributed environment: 16th australasian conference, ausdm 2018, baururst, nsw, australia, november 28–30, 2018, revised selected papers", in. January 2019, pp. 69–80.
- [35] M. N. Helaskar and S. S. Sonawane, "Text classification using word embeddings", in *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, 2019, pp. 1–4.
- [36] M. A. Hedderich, L. Lange, H. Adel, J. Strotgen, and D. Klakow, "A survey on recent approaches for natural language processing in low-resource scenarios", *ArXiv*, vol. abs/2010.12309, 2021.

Appendices

A [What is the structure of BAS Chart of Accounts]

The BAS chart is structured as each account has four digits. The first number indicates the account class. The following digits are used for more accurate breakdown, where the second digit indicates the account group, the third digit the main account and the fourth digit the sub-account.

1xxx Assets divided as 10xx Intangible fixed assets, 11xx-12xx Tangible fixed assets, 13xx Financial fixed assets (long-term receivables) 14xx Inventories, etc., 15xx-17xx Current receivables, 18xx Short-term investments, 19xx Cash and bank 20xx Equity, 21xx Un-taxed reserves, 22xx Provisions, 23xx Long-term liabilities, 24xx-29xx Current liabilities 3xxx Revenue divided as 30xx-37xx Net sales, 38xx-39xx Other revenue 4xxx Costs for goods, materials and some purchased services 5xxx-6xxx Other external operating expenses 70xx-76xx Personnel costs, 77xx Write-downs, 78xx Depreciation, 79xx Other operating expenses 8xxx Financial items (e.g. interest / currency income / expenses), appropriations, tax and profit for the year 0xxx and 9xxx Internal accounts