Coordinate Matrix Machine: A Human-level Concept Learning to Classify Very Similar Documents

Amin Sadri and M. Maruf Hossain*
Machine Learning as a Service
Intelligent Automation
Enterprise Data and Intelligent Automation
{sadria, maruf.hossain}@anz.com

Abstract

Human-level concept learning argues that human usually learns new concepts from just a single example unlike the machine learning algorithms that typically require hundreds of samples to learn a single concept. Our brain subconsciously picks up the important features and learns in a richer way.

Contribution: In this paper, we present a coordinate matrix-based approach that augments human intelligence to learn the structure of the document and use this information to classify the documents. Our approach only processes important features that are what a human considers for classification. Although the documents are more structured and contains very similar content to one another, only one sample is used per class similar to what a human needs to do similar task.

Advantage: Our algorithm outperforms every popular vectoriser techniques in combination with traditional machine learning and most advanced deep learning models that rely on a larger number of elements to achieve good accuracy. The advantages of our algorithm are:

1. Better accuracy
2. Faster computation
3. Generic, expandable and extendable
4. Explainable
5. Robust against unbalanced classes
6. Does not require large number of labelled data

Keywords: Human-level, Concept Learning, Augmented Intelligence, Text Mining, Coordinate Matrix, Lazy Learning, Classification

1 Introduction

Human-level concept learning is a relatively newer area of research. The goal is to provide a solution to a problem that is easy for human to do, yet still hard for machines despite of their computational power. People can learn new concepts easily from only one or very few samples, while machine learning methods needs plenty of example to find correlations and understand features. The reason is that people subconsciously pick the important features and then generalise to other distinct areas [1, 2].

In this paper, we deploy a human-level concept learning approach for document classification. Document classification is a common task in machine learning in the form of Natural Language Processing (NLP) where a document is assigned to one or more classes. Current approaches rely heavily on the context of the document in order to classify them [3, 4]. Identifying the

*Corresponding author.

relevant topic is the primary motivator for such document classification [5–7]. These models assume we have access to plenty of labelled data and the context of the documents are enough informative to distinguish between classes. Some other methods use the image information of the documents which makes the labelling process even worse. As an image is a high dimensional data, we need to label so many training samples that is not doable in most of the problems in reality [8].

A practical situation would be, when bank statements from other financial institutions are received along with loan applications, the task falls on to identify the templates for those statements so that existing functions can be applied to extract information from those statements. In reality, when we ask someone to classify these statements, we do not need to show them hundreds of samples for each class. Just one sample is enough for presenting one class. This fact shows us one sample is informative enough for classification if the algorithm is well-designed. This is our motivation for developing our algorithm.

Our aim is to improve the classification performance on high number of classes by incorporating domain knowledge, i.e., augmenting human intelligence, when there is very small training data and the documents are very similar to one another.

1.1 Challenges

When there is a large variation of structured documents that needs to be classified based on the structure of the document, machine learning techniques tends to suffer from poor performance. We have identified the following challenges when dealing with bank statements in particular:

- A large number of documents need to be labelled to train a model with sufficient accuracy, which is both time and resource consuming.
- There are way too many class labels at the template level. For the five banks included in this experiment, there were 53 templates (classes) available. There are not enough representative samples for each class.
- All bank statements look very similar to one another as all of them include many transaction descriptions, dates, and amounts.
- Majority of the words in bank statements are personal and highly contextual, such as account holder's name, address or transaction items, which produces lots of noise when training a classification model.
- The words which are not noise are very similar across a range of bank statements, e.g., 'Name', 'Account', 'Balance', 'Date'.

Due to these challenges, traditional machine learning models and more recent deep learning models do not perform well.

Example 1.1. While classifying statements using Term Frequency Vectoriser with Logistic Regression, the model with the best possible parameters yielded an F-measure of 79% when we attempted to classify the templates.

This lower performance can discourage users to use simplistic models and inclined to use more complex models like Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) [9, 10], which are by default not explainable, and rely on additional techniques [11] to explain, especially to the governing body for compliance purpose.

Example 1.2. After running hours to build CNN or Long Short-Term Memory (LSTM) RNN architecture, we have found that CNN could yield a F-measure of 93%, but LSTM failed to classify 53 templates.

1.2 Our Contribution

In this paper, we apply a human-level concept learning [1] to address this problem. We choose this approach because as a human we do not need hundreds of training samples to learn one class, and we are able to classify the documents even when the content are similar, sometimes even without reading the texts. Only one sample per template is enough and we should consider not only the content but also the location of certain words.

The closest work to our work in terms of the nature of the approach is the research conducted by Lake et al. [1]. They have argued that in most cases people can learn a new concept from just one or a handful of examples, while typical machine learning algorithms require hundreds of examples to perform similarly. They aim towards a different problem and recognise every handwritten characters from image data. While in this paper, we aim to classify structured documents from text data. Although the problems are different, both works use human-level concept learning by looking at the way a human solve the problem. Both approaches only use one sample per class and learn from that sample.

Given documents of similar structure, as human we look at the structure of the document and/or the position of some keywords (e.g., Account Name, Date, Bank Name etc.). Similarly, in our approach, we have proposed a hybrid lazy learning algorithm, Coordinate Matrix Machine (CM^2), to create an input matrix (which contrasts with traditional vector-based input strategy) containing the coordinates for the keywords and use this matrix to classify documents.

The rationale behind the technique are as follows:

1. With structured documents, the position carries more importance than the occurrence or the order of the occurrence for the terms. Furthermore, by augmenting the understanding of the subject matter experts, we can avoid developing huge corpus, thus remove noises and eventually improve classification performance.
2. This technique would allow us to avoid labelling large amount of documents often needed to train machine learning and deep learning models.

1.3 Advantages

The advantages of our algorithm are:

1. It has better accuracy compared to all algorithms tested.
2. It has faster computation.
3. It is generic, expandable and extendable. For newer samples, there is no need for any feature engineering or feature extraction.
4. It is explainable. It is easy to trace the reason behind classification and modify, if necessary.
5. It is not affected by unbalanced classes, low-volume or low-quality training data.
6. It does not require large number of labelled data.

1.4 Organisation

The remainder of the paper is organised as follows. In Section 2, we briefly discuss the related work in document classification. Our algorithm is formally described in Section 3. Section 4 presents the design of the experimental investigation. We present and discuss the results in Section 5 and 6. Finally, in Section 7, we suggest future improvements and conclude the paper.

2 Related Work

As document classification has become an emerging area in the text mining research, a large amount of prior work has focused on it. A typical document classification has several pre-processing steps and researches focused on each step to improve the performance, such as stopwords removal [12, 13], Tokenization [14, 15], Part-of-Speech Tagging [16], Stemming [17].

The second step usually focus on feature extraction and selection. For example, Yang et al. [18] used titles and other tag data to label the text features. Shih and Karger [19] used the geometry of the rendered HTML page to build up tree models based on the incoming link structure. Term Frequency–Inverse Document Frequency (TF-IDF) is a popular way for feature selection [3]. Power et al. [4] focused on web page classification and filtered the output of the TF-IDF algorithm to improve the performance.

Once the feature vector is identified, the third step is to apply a machine learning model for classification. Naïve Bayes is one of the probability-based classifiers [6]. Decision tree-based approaches have also been used for different purposes such as inappropriate web content blocking [20]. Support Vector Machines (SVM) are also widely used for document classification [7,21]. There are multiple approaches under artificial/deep neural network. Hassan and Mahmood [5] applied Convolution Neural Networks (CNN) for document classification. Unlike most of the machine learning models, Recurrent Neural Networks (RNN) take the sequence of the occurrences of the words into consideration, therefore documents with similar words will have different outputs because of the order of the words [9, 10].

3 Methodology

We now describe in more detail, the steps in our algorithm for creating the coordinate matrix and inducing the classifier using the matrix.

3.1 Pre-processing the Documents

Each training data is a PDF file contains either a scanned image or a digital document. We have ensured that each page of the document is set to 300 dpi before we run the pages through an Optical Character Reader (OCR) engine to get the words and their respective coordinates. The output of the OCR is stored as eXtensible Mark-up Language (XML) files that are used as input to induce our model.

3.2 Building the Coordinate Matrix

For each class, only one sample is required for training. We accompany a Comma Separated Value (CSV) file with each XML file, which contains key-value pair for all keywords within the document.

Example 3.1. Statement A contains the term 'Account No.' followed by the account number '061234-12345678', the term 'Account Holder' with the value 'John Doe', and the term 'Account Type' with the value 'Savings Account'. Whereas, in statement B, the relevant term 'Account No.' followed by the number '064321-87654321', and the term 'Account Name' with the value 'Jane Smith' is recorded. So in the CSV file for statement A, we have the entries "`Account No.,061234-12345678`", "`Account Holder,John Doe`" and "`Account Type,Savings Account`"; and for statement B, we have "`Account No.,064321-87654321`" and "`Account Name,Jane Smith`" recorded. We then search for the keywords (e.g., 'Account No.', 'Account Name') in the XML file and construct a matrix with top and left position for each keyword for each document as shown in Tab. 1.

Table 1: Example of the Coordinate Matrix

Document ID	Keyword	Top	Left
Statement A	Account No.	254	1231
Statement A	Account Holder	261	1231
Statement A	Account Type	269	1231
Statement B	Account No.	1123	231
Statement B	Account Name	100	359

The first column comes from the training data, the second column comes from the CSV file accompanied with each XML file, and the last two columns are the search result for the keywords in the XML files. Algorithm 1 lists the steps.

Algorithm 1: Build the Coordinate Matrix

Input : $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$, where t_i is a training sample of i^{th} class and N is the total number of class.
$\mathcal{K} = \{K_1, K_2, \ldots, K_N\}$, where $K_i = \{k_1^i, k_2^i, \ldots k_{M_i}^i\}$ is a set of keywords for i^{th} class, k_j^i is the j^{th} keyword of i^{th} class, and M_i is the total number of the keywords for i^{th} class.

Output: $\mathcal{CM}$: Matrix of coordinates for all keywords for all training samples, i.e., $\langle t_i, k_j^i, x_{ij}, y_{ij} \rangle$ for every $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M_i$

begin
  $M \leftarrow \sum_{i=1}^{N} M_i$; `// total number of keywords`
  $row \leftarrow 1$;
  $\mathcal{CM} \leftarrow M \times 4$ matrix;
  for $i \leftarrow 1$ *to* N **do**
    for $j \leftarrow 1$ *to* M_i **do**
      $\langle x_{ij}, y_{ij} \rangle \leftarrow$ `GetCoordinates`(t_i, k_j^i);
      $\mathcal{CM}_{row} \leftarrow \langle t_i, k_j^i, x_{ij}, y_{ij} \rangle$;
      $row \leftarrow row + 1$;
    end
  end
  return $\mathcal{CM}$;
end

`// Function to get the coordinates of a keyword`
Function `GetCoordinates`($\mathcal{D}$, k)
  Input : $\mathcal{D}$: a document as XML
    k: the keyword to find in the document.
  Output: $\langle x, y \rangle$: the coordinates of the start position of the keyword k in the document $\mathcal{D}$.

  if $k \in \mathcal{D}$ **then**
    return $\langle x, y \rangle$; `// the keyword is found`
  end
  return $\varnothing$;
end

Table 2: Coordinate Matrix for the Test Case

Document ID	Keyword	Top	Left
Test case	Account No.	1120	230
Test case	Account Holder	–	–
Test case	Account Type	–	–
Test case	Account Name	101	360

3.3 Classifying New Documents

When a new document comes in, we run OCR on the documents. On the XML produced by the OCR, we look for all the words in the test document. We do this to ensure the approach stays robust against rotation or shift of the coordinates that can occur during the document scanning process.

We match the extracted words against all the keywords on the training data. Once we extract the matched keywords, we then form a coordinate matrix for the test cases comprising the top and left position.

Example 3.2. Test case contains the term 'Account No.' followed by the account number '061111-11111111', and the term 'Account Name' with the value 'Jane Doe'. So the coordinate matrix for the test case would be as shown in Tab. 2. The second column in the table includes all the keywords available in the training coordinate matrix.

We then calculate the distance for each keyword between the test data vs every training sample. To calculate the distance between the keywords, Manhattan distance has been used. We used Manhattan distance because it gives less value to the horizontal and vertical shifts compared to the diagonal shifts. The horizontal and vertical shifts is more likely in a document due to extra empty line or space.

We have introduced only one parameter in this algorithm.

Definition 3.1. Maximum Penalty is the maximum distance allowed between two keywords. If the distance between the same keywords from two documents is more than this threshold, then the actual distance is substituted by this value. Besides, when a keyword is not found in a document the distance for that keyword is set to this value. In other word, if the distance between the keyword found and its counterpart is more than the **maximum penalty**, we assume that keyword is not found in the document.

We defined this parameter for two reasons:

1. There should be a distance value even when we cannot locate a keyword in a document otherwise we cannot apply the mean function in the next step.
2. We want to ensure that the algorithm is robust by limiting the effect of a single keyword. Otherwise, if a keyword in two documents is very far, either due to the poor extraction quality of the OCR or being a variant of the training data, the calculated distance will be too great.

Finally, we calculate the mean distance for all keywords of each training sample to get a similarity score between the test case and that training sample. The sample that has the minimum distance to the test case identifies the class. Algorithm 2 lists the steps and Example 3.3 demonstrates the calculation.

Example 3.3. Let us assume that the `maximum_penalty` is 200. Table 3 shows the training data and the test case for each keywords.

Algorithm 2: Classifying New Documents

Input : $\mathcal{CM}$: the Coordinate Matrix which contains $\langle t_i, k_j^i, x_{ij}, y_{ij} \rangle$ for each rows
$\mathcal{D}$: the test document as XML
θ: maximum_penalty, the maximum distance allowed between two keywords

Output: $\langle \mathcal{C}, \Delta \rangle$, where $\mathcal{C}$ is the predicted class and
Δ is the minimum distance between the training and the test document

```
1  begin
2      for i ← 1 to N do
3          for j ← 1 to M_i do
               // D_ij represents the distance from k_j^i to its counterpart in D
4              coords ← GetCoordinates(D, k_j^i);
5              if coords = ∅ then
6                  D_ij ← θ ;                          // the keyword is not found
7              else
8                  ⟨tx_ij, ty_ij⟩ ← coords;
9                  D_ij ← |x_ij − tx_ij| + |y_ij − ty_ij|;
10                 if D_ij > θ then
11                     D_ij ← θ;
12                 end
13             end
14         end
15     end
       // Identifying the class
16     C ← ∅;
17     Δ ← θ;
18     for i ← 1 to N do
19         distance ← Σ_{j=0}^{M_i} D_ij × 1/M_i;
20         if distance < Δ then
21             C ← t_i;
22             Δ ← distance;
23         end
24     end
25     return ⟨C, Δ⟩;
26 end
```

The distance between 'Account No.' in Statement A is 200 because the Manhattan distance exceeds 200 pixels. For 'Account Holder' and 'Account type' the distances are 200 because they are not found in the test case. As a result the distance between the test case and Statement A is 200 while this value is the average of 4 and 2 for Statement B. Therefore, the test case belongs to Statement B class with a similarity score of 3.

3.4 Complexity of the Algorithm

Let us consider a training dataset $\mathcal{T}$ of N samples, where each sample comprises of several keywords and M is the total number of keywords in all N samples. The time complexity to classify one test case containing L number of words would be $O(LM)$, because for each of the M available keywords, which can contain several words, we have to search the test document. We can see that the computational complexity of our algorithm is linear to the size of the test

data. Assuming that the number of keywords in each document is same or within a range, M is proportional to N (i.e., $M \sim N$), and thus $O \sim N$. This means that the computational complexity of our algorithm is also linear to the number of training samples.

4 Experiments

For the experimental analysis, we compare a combination of 6 vectorisation and 9 classification techniques. We have used three Bag of Words vectoriser techniques: Term Frequency Vectoriser [22], TF-IDF Vectoriser [3], Hashing Vectoriser [23, 24]; two Word Embedding techniques: Global Vector (GloVe) [25]'s 6B pre-trained tokens, Google's pre-trained Word2Vec [26], and one paragraph embedding technique: Doc2Vec [27]. The classification algorithms used are: Logistic Regression, Decision Tree (with CART and C4.5), SVMs (with linear, polynomial, RBF and Gaussian kernel), Random Forest, Naïve Bayes, k-Nearest Neighbour. We also compared against several deep learning classifiers: Artificial Neural Network, CNN and LSTM-based RNN.

4.1 Data Set

We have randomly selected 475 statements for 5 banks that have been received with home loan applications in 2019. The distribution of the statements for the banks are shown in Fig. 1. These statements falls under 53 templates. 16 of which has only 1-2 samples.

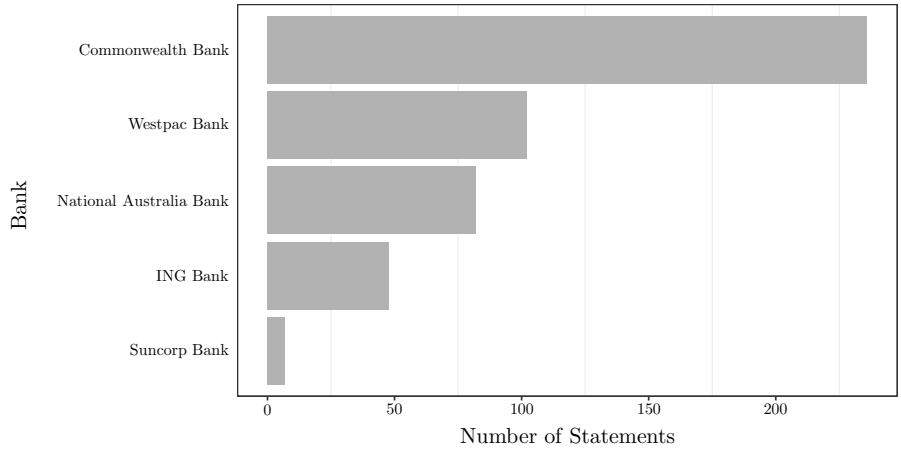

Figure 1: The Distribution of Statements for Each Bank

4.2 Experiment Design

Each of the vectoriser is applied with each classifier, except Naïve Bayes which we could not use with Hashing, Google's Word2Vec and Doc2Vec vectoriser (because it is not compatible with them), making a total of 51 combinations to compare against our approach.

Table 3: The Distance between the Training Data and the Test Case for Each Keywords

Document ID	Keyword	Calculation	Distance
Statement A	Account No.	$\|254 - 1120\| + \|1231 - 230\|$	200
Statement A	Account Holder	Not found	200
Statement A	Account Type	Not found	200
Statement B	Account No.	$\|1123 - 1120\| + \|230 - 231\|$	4
Statement B	Account Name	$\|101 - 100\| + \|360 - 359\|$	2

Table 4: Parameters Used during Grid Search for Each Algorithm from Sci-kit Learn [24] and Keras [28]

Algorithm	Parameters
Frequency Vectoriser, TF-IDF Vectoriser, Hashing Vectoriser	`ngram_range`: [1, 4) `max_features`: [20, 40, 60, ..., 6800]
GloVe, Google's Word2Vec, Doc2Vec	`max_features`: [20, 40, 60, ..., 6800]
Logistic Regression	`penalty`: {none, ℓ_2} `C`: {0.000001, 0.009, 0.001, 0.09, 0.01, 1, 5, 10, 25} `max_iter`: {100, 120, 130, 140, 150}
Decision Tree	`criterion`: {gini, entropy} `min_samples_split`: {2, 4, 5, 6} `max_features`: {auto, sqrt, log2, none}
Support Vector Machine	`C`: {0.001, 0.01, 0.1, 1, 10, 100} `kernel`: {linear, poly, rbf, sigmoid} `degree`: {1, 2, 3} `tol`: {0.0001, 0.001, 0.1} `decision_function_shape`: {ovo, ovr}
Random Forest	`n_estimators`: {10, 11, 13, 15, 100, 115, 120, 125, 150, 200} `criterion`: {gini, entropy} `min_samples_split`: {2, 4, 5, 6} `max_features`: {auto, sqrt, log2, None}
Naïve Bayes	`alpha`: {0., 0.0001, 0.001, 0.01, 0.1, 1, 10} `fit_prior`: {True, False}
k-Nearest Neighbour	`n_neighbors`: {3, 5, 7, 9, 11, 13} `algorithm`: {ball_tree, kd_tree, brute} `leaf_size`: {30, 35, 30, 45, 50, 55}
Artificial Neural Network, CNN, LSTM-based RNN	`activation`: {relu, tanh} `optimizer`: {SGD, Adam, Adamax, Adagrad, Adadelta, Nadam, RMSprop} `epochs`: {10, 50, 100} `learn_rate`: {0.01, 0.1, 0.2}

We have set aside 127 samples as test data while optimising the parameters using grid search. And used the remaining 348 to train models. We have ensured that for the minority classes, at least one sample remains in the training set. We have employed 10-fold cross validation (CV) techniques for the machine learning models and 3-fold CV for the deep learning models to select the best parameters using the training set. Table 4 shows all the parameter values used during the grid search. Once the best parameters are identified, we have induced a final model with the entire training set and evaluate the model on 127 test data.

To test the effect of the size of the training set, we have also used 53 samples to train each model and used the remaining data to test the model performance. We then continue to add 59 samples to the training data and check the model performance on the remaining data. Eventually, we have trained each model with 53, 112, 171, 230, 289 and 348 training data and test the model performance on the remaining.

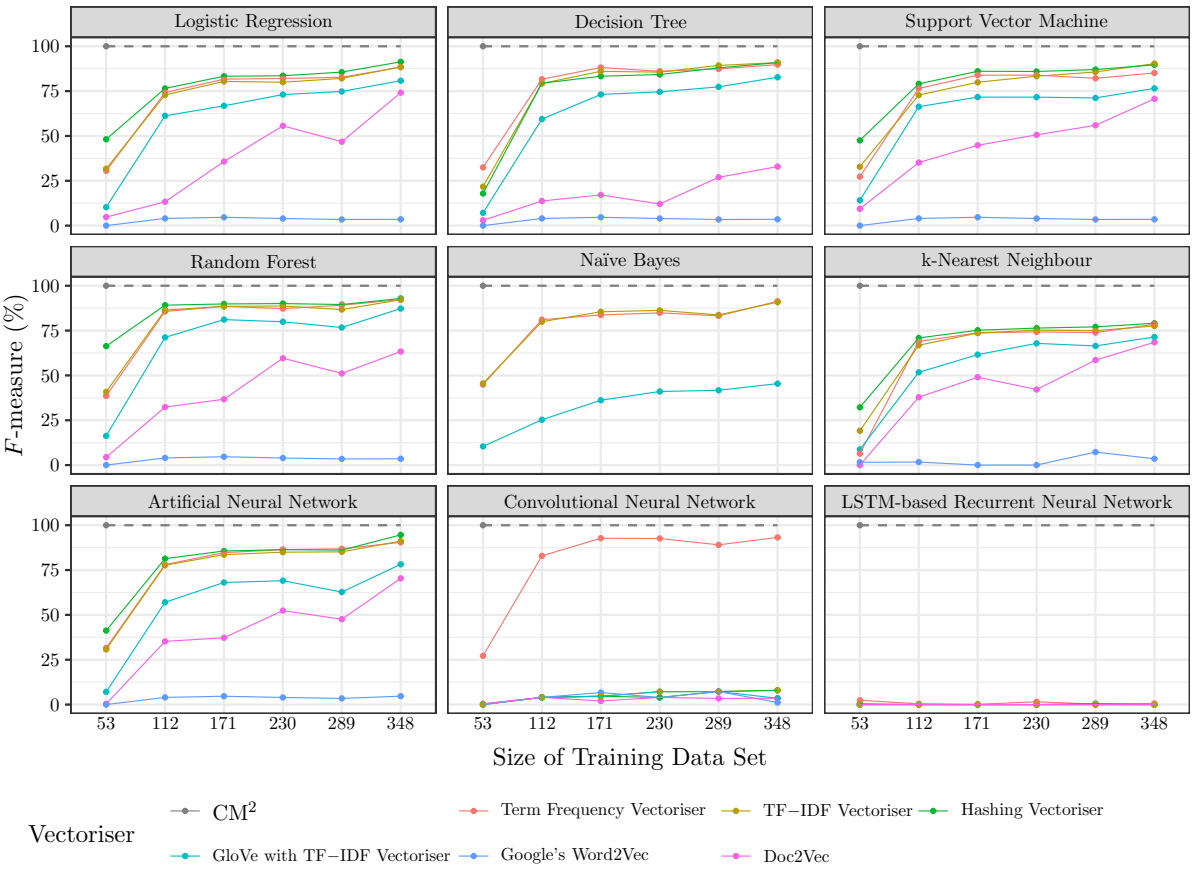

Figure 2: Performance of Different Vectoriser with Every Classifier Compared Against CM^2 for Varying Training Data Set

5 Results

Figure 2 shows the results for different algorithms used with different vectorisation methods while the size of the training data is growing. Each chart belongs to one machine learning approach and the colours denote the vectorisation method. As expected, the line charts show upward trends which indicates that the performance improves with the increased number of the training samples. This upward trend also demonstrates the trade-off between having the labelled data and the performance.

On average, Hashing Vectoriser performs better with most classifiers compared to other vectorisation methods especially for when using a smaller training data. Google's Word2Vec has the worst performance with any selected classifier. This is primarily because of banks sometimes use non-standard abbreviations in the transaction descriptions to fit larger words in a fixed-width text field and Google's pre-trained model is not designed for this type of situation.

The ensemble classifier Random Forest outperforms all other classification techniques, closely followed by a simple ANN. While LSTM performed very poorly due to the fact that it relies on very large training data and word sequence. In our experiment, LSTM had a training accuracy over 98% when 348 documents have been used for training, but the model struggles to correctly identify more than 2 samples during testing. Furthermore, bank statements do not really contain any sentences by which techniques like LSTM could benefit.

For this experiment, we run our algorithm only once as we only use 53 samples, and the result is shown with a grey dashed line in the charts. As we can see, our method outperforms all the

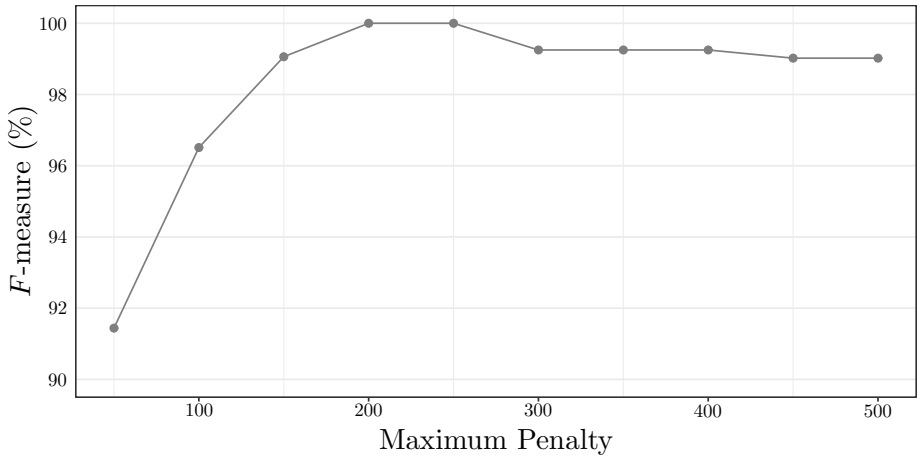

Figure 3: Effect of Different Maximum Penalty on CM^2 Performance

machine learning methods regardless of the type of the classifier and vectoriser and the size of training data. It should be noted that all of the other models run with the best parameters we obtained from the grid searches. Though we have reported the best performances of these models using the best parameters, our algorithm still yields a better performance with a simple setting and the smallest set of training data. This shows considering the right features and designing a right approach is more important than switching or ensembling models.

Figure 3 shows the parameter sensitivity analysis for the maximum penalty, which is the only parameter required by our algorithm. We can see that for very small value the performance is lower because, our approach is quite sensitive to the coordinates of the keywords. In other words, if the keyword is just a little shifted from its expected position, we assume the keyword is not found. In practice, the location of the keywords may change because of scanning or having extra lines or space. At `maximum_penalty` = 200, the accuracy is 100%. Let us consider the resolution of an input image 2480 × 3500. If the keyword is around 10% of the width apart from the original location of the keyword, it is considered as not found.

By increasing the maximum penalty more than 250, we also see a drop in accuracy because the keywords that are relatively far from each other are now considered matched. After 400, we do not see much difference in performance. The reason is when we put a very high threshold, only a few distances fall below the threshold. Theoretically, all the Manhattan distances in a 2480 × 3500 document is less than 2480 + 3500. Therefore, setting a threshold higher than 2480 + 3500 is useless and has no effect.

6 Discussion

We have demonstrated how a human-level concept learning, when designed the right way, is far superior than machine learning methods especially when learning from limited data. In contrast to machine learning methods, which need hundreds of samples for each class to learn the concept, our approach needs only one sample and use the concept in a richer way. Just like human, our approach tries to match the test case to each of the known samples, estimates the similarities, and chooses the most similar class. Taking a deeper look, our approach finds the keywords and matches the keywords between the documents. However, if the matched keywords are too far from each other, it discards the matching and assumes there is no match for that keyword. Similarly, as a human, if we want to match the keywords to detect a document type, we would not match the keywords that are too far.

Another advantage of the human-level concept learning is the way we set the parameter. In a

human-level concept learning, our understanding of the problem helps us to define the values for the algorithm parameters while for machine learning approaches, a parameter optimisation technique (i.e., grid search) needs to be used to find the best value. This can be very time consuming, and depends on how many parameters needs to be tuned, how many values to consider for each parameter or the computational complexity of the algorithm. Furthermore, the algorithm needs to run for each combination of the parameter setting. Still then, there is no guarantee that the best parameter value is amongst the considered parameter values.

When we want to set a value for *maximum penalty*, we should ask ourselves this question "what is the maximum distance that we want allow to match keywords?" or "how far a keyword can shift due to some extra line or extra space in a document?". By looking at some sample documents, we estimate that 200 is an appropriate value for the *maximum penalty*. Definitely, it is better to check the other values but we expect the best value to be around 200. On the other hand, assume we are using a deep learning model and we want to set values for ℓ_1 and ℓ_2. It is very hard to understand what these values actually mean for our problem. Therefore, we have to choose them by trial and error or by using grid search. In most cases, grid search with higher number of parameters results in overfitting [29].

Designing a human-level concept learning starts with thinking about the way human does the job. First we look at the data, and then we should ask ourselves as human how do we solve the challenge. How do we recognise a template for bank statements where the contents are almost similar? The answer to the question identifies what feature to be used and how to design the approach.

7 Conclusion

Considering the way human learns, we developed a novel algorithm for classifying structured documents with similar content. The main strategy for us to design the algorithm was augmenting human intelligence to achieve a higher performance for our data. Unlike other typical machine learning algorithms that act mostly like a black-box, our approach encorporated the way we classify this type of documents in real life.

Human-level concept learning is not limited to only document classification or character recognition. The pain of providing the labelled data is obvious to industry regardless of the types of problems. This motivates us to deploy human-level concept learning approaches in various problem. As this model uses the distance of the keywords to identify similarity, in future, we would extend on this feature to extract entities relevant to the keywords from the structured documents.

Acknowledgement

The authors thank ANZ Document, Knowledge and Content Automation team for providing the statements in PDFs and running through OCR to produce XML to conduct this research.

References

[1] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan

Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–333, 2015.

[3] Karen Spärck Jones. A Statistical Interpretation of Term Specificity and Its Application in Retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[4] Russell Power, Jay Chen, Trishank Karthik, and Lakshminarayanan Subramanian. Document Classification for Focused Topics. pages 67–72, 2010.

[5] Abdalraouf Hassan and Ausif Mahmood. Efficient Deep Learning Model for Text Classification Based on Recurrent and Convolutional Layers. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1108–1113, Cancun, Mexico, 2017. IEEE.

[6] Igor Kotenko, Andrey Chechulin, and Dmitry Komashinsky. Evaluation of Text Classification Techniques for Inappropriate Web Content Blocking. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, pages 412–417, Warsaw, Poland, 2015. IEEE.

[7] X. Lin, Hong Peng, and Bo Liu. Support Vector Machines for Text Categorization in Chinese Question Classification. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 334–337, Hong Kong, China, 2006. IEEE.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25(2), 2012.

[9] Imon Banerjee, Yuan Ling, Chen Matthew C., Sadid A. Hasan, Curtis P. Langlotz, Nathaniel Moradzadeh, Brian Chapman, Timothy Amrhein, David Mong, Daniel L. Rubin, Oladimeji Farri, and Matthew P. Lungren. Comparative Effectiveness of Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) Architectures for Radiology Text Report Classification. *Artificial Intelligence in Medicine*, 97:79–88, 2019.

[10] E Naresh, B. P. Vijaya Kumar, V. S. Pruthvi, K Anusha, and V Akshatha. Survey on Classification and Summarization of Documents. *International Journal of Research in Advent Technology*, 7(6S):28–32, 2019.

[11] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *2016 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 97–101, San Francisco, CA, USA, 2016. ACM.

[12] Sonya Rapinta Manalu. Stop Words in Review Summarization using TextRank. In *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 846–849, Phuket, Thailand, 2017. IEEE.

[13] R. J. Prathibha and M. C. Padma. Design of Rule based Lemmatizer for Kannada Inflectional Words. In *2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, pages 264–269, Mandya, India, 2015. IEEE.

[14] Juhaida Abu Bakar, Khairuddin Omar, Mohammad Faidzul Nasrudin, and Mohd Zamri Murah. Tokenizer for the Malay Language using Pattern Matching. In *2014 14th International Conference on Intelligent Systems Design and Applications*, pages 140–144, Okinawa, Japan, 2014. IEEE.

[15] Said Gadri and Abdelouahab Moussaoui. Information Retrieval: A New Multilingual Stemmer based on A Statistical Approach. In *2015 3rd International Conference on Control, Engineering & Information Technology (CEIT)*, pages 1–6, Tlemcen, Algeria, 2015. IEEE.

[16] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, and Nathan Schneider. Part-of-Speech Tagging for Twitter: Word Clusters and Other Advances. Tech Report CMU-ML-12-107, School of Computer Science, Carnegie Mellon University, 2012.

[17] Haiyi Zhang and Di Li. Naïve Bayes Text Classifier. In *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pages 708–708, Fremont, CA, USA, 2007. IEEE.

[18] Yiming Yang, Seán Slattery, and Rayid Ghani. A Study of Approaches to Hypertext Categorization. *Journal of Intelligent Information Systems*, 18(2):219–241, 2002.

[19] Lawrence Kai Shih and David R. Karger. Using URLs and Table Layout for Web Classification Tasks. In *2004 13th international conference on World Wide Web (WWW2004)*, pages 193–202, New York, New York, USA, 2004. ACM.

[20] Han Liu, Mihaela Cocea, and Weili Ding. Decision Tree Learning based Feature Evaluation and Selection for Image Classification. In *2017 International Conference on Machine Learning and Cybernetics (ICMLC)*, pages 569–574, Ningbo, China, 2017. IEEE.

[21] Mojgan Farhoodi and Alireza Yari. Applying Machine Learning Algorithms for Automatic Persian Text Classification. In *2010 6th International Conference on Advanced Information Management and Service*, pages 318–323, Seoul, South Korea, 2010. IEEE.

[22] Yin Zhang, Rong Jin, and Zhi Hua Zhou. Understanding Bag-of-Words Model: A Statistical Framework. *International Journal of Machine Learning and Cybernetics*, 1:43–52, 2010.

[23] scikit-learn Developers. scikit-learn 6.2. Feature extraction, 2007–2019.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *2014 Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. ACL Anthology.

[26] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Computing Research Repository (CoRR)*, abs/1301.3781, 2013.

[27] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *2014 31st International Conference on Machine Learning (ICML), Journal of Machine Learning Research (JMLR) W&CP*, volume 32, pages II–1188–II–1196, Beijing, China, 2014. ACM.

[28] François Chollet et al. Keras. `https://keras.io`, 2015.

[29] Gavin C. Cawley and Nicola L. C. Talbot. On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.