Scientax
Jurnal Kajian Ilmiah Perpajakan Indonesia

VOLUME 4 NO 2 | APRIL 2023 Journal homepage: ejurnal.pajak.go.id ISSN 2686-5718

Machine Learning: Classifiying Taxpayer's Supervising Zone Based on The Street Address Using a Natural Language Processing Algorithm

Reno Iqbalsah
Directorate General of Taxes, Jakarta, Indonesia. Email: reno.iqbalsah21@gmail.com

*Corresponding Author: reno.iqbalsah21@gmail.com

ABSTRACT

Assigning taxpayers into their respective Account Representatives is a crucial step to optimize Taxpayers supervision. However, the large number of registered taxpayers and missing data has been a great challenge. A lot of taxpayers only include their street addresses and no additional information such as RT, RW, etc. This will cause additional work to manually search each taxpayer address in the internet and manually assigned, which is not efficient and takes a lot of time. This study will try to solve this problem using Natural Language Processing algorithm. Efficiency and accuracy are the key on creating machine learning model. Choosing the right classifier is crucial to the accuracy. Other than the classifier, managing text data is also challenging, since it cannot be understood directly by computers. Thus, this study will also include how we could transform the text data into arrays of numbers called Bag of Words.

Keywords: machine learning, Natural Language Processing, supervision zones, cosine similarity

1. INTRODUCTION

With regards to Nota Dinas Direktur Ekstensifikasi dan Penilaian No. ND-56/PJ.06/2022 (Directorate General of Taxes, 2022), each KPP Pratama are divided into multiple supervising zones that has to be supervised by Account Representatives from (usually but not always) 5 supervising divisions (Seksi Pengawasan). Every taxpayers have to be assigned to their respective Account Representatives of each zone using the ECTag application (mapping.pajak.go.id). This app assigns the taxpayers using these methods respectively:

1. All Taxpayers with geotagging data will be automatically assigned using point of interest (coordinate) based on the supervising zones that have been created by tax offices.
2. The remaining will be assigned based on the area code, this only apply if the supervising zone is one full village (kelurahan / desa).
3. Other taxpayers with no geotagging data nor area code have to be assigned manually, choosing from a dropdown list of supervising zones, or by uploading a CSV file containing the taxpayer's ID (NPWP) and the number of zones that they are supposed to be assigned.

According to the data from ECTag application, out of more than 56 million taxpayers, only 0.3% (around 202 thousand) are automatically assigned using geotagging, 71% (around 40,3 million) of which are assigned automatically using area code, and the remaining of 16,1 million have to be assigned manually. From this data, we could see that the geotagging data is very limited, and we rely so much on area code to assign taxpayers.

doi: 10.52869/st.v4i2.486
Received: July 31, 2022; Accepted: April 10, 2022; Published: April 27, 2023

How to Cite:
Iqbalsah, R. Machine learning: Classifiying taxpayer's supervising zone based on the street address using Natural Language Processing algorithm. *Scientax: Jurnal Kajian Ilmiah Perpajakan Indonesia*, *4*(2), 233–242. https://doi.org/10.52869/st.v4i2.486

However, area codes are only accurate to village level (kelurahan / desa). In a big city, where there are a lot of tax offices, and one kelurahan could be supervised by more than one Account Representatives, it becomes harder to assign the taxpayers since they cannot be assigned automatically using the area code. For example, in some Regional Tax Offices like Kanwil DJP Jakarta Utara, from 935.000 taxpayers, only 11.000 are assigned automatically by the area code, and 899.000 have to be manually assigned. Another example in Kanwil DJP Jakarta Timur, out of 1,6 million taxpayers, only 36 are automatically assigned using area code, and 1,5 million are still prone to manual assignment.

This study is done to apply a new approach on solving this problem with limited data and resources. Google Maps has a lot of coordinates and address data, we could match the street address data and their coordinates to our predesigned supervising zones. And afterwards, using machine learning to measure the similarity of the addresses, then assign the unassigned address to a zone with the most similar address names.

2. THEORETICAL FRAMEWORK

Machine learning has gained a lot of attention recently all around the world. But what is machine learning and how it works? According to Zhang et al. (2022), machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. In short, we can feed the machine learning algorithm with a set of data containing predictors (X_train) and labels (y_train). Then, the algorithm will learn relations between X and y in the training sets and adjust itself. Resulting in a model that could predict label from an unknown new dataset (X_test).

This study will use the Bag of Words model. Bag of Words turns a set of text or document, into a vector containing numbers representing the count of each word in the document, ignoring all grammatical features and words order (McTear et al., 2016). Street addresses usually don't have any grammatical features anyway, so this is not a big concern on this study. But retaining word order may become an advantage in this case, nevertheless it is outside the scope of this study.

Bag of Words model is one of the most popular approaches in preprocessing text data. Its simplicity makes it easy to use and still gives a high accuracy (McTear et al., 2016). The process of transforming text data into numbers usually called vectorization. To elaborate on how vectorization works, an example is presented in Table 1.

Based on the vectorization illustrated in the Table 1, the vectorized values can be saved into vector A and vector B. With the values of vector A = [1,2,0,1,0,1,0] and B = [1,0,1,1,1,0,1]. We could measure the similarity of those vectors using Cosine Similarity. This formula is based on Euclidean Dot Product formula where the dot product of two non-zero vectors can be calculated using the following equation (Giller, 2012):

$$A \cdot B = \| A \| . \| B \| . cos\ (\theta) \tag{1}$$

or

$$A \cdot B = \sum_{i=1}^{n} A_i \,.\, B_i \tag{2}$$

Then, we could define the cosine similarity as:

$$cosine\ similarity = S_C(A, B) := \cos(\theta) = \frac{A \cdot B}{\|A\| . \|B\|} \tag{3}$$

$$\downarrow\downarrow$$

$$\boldsymbol{\cos(\theta)} = \frac{\sum_{i=1}^{n} \boldsymbol{A_i} \cdot \boldsymbol{B_i}}{\sqrt{\sum_{i=1}^{n} \boldsymbol{A_i}^2} \cdot \sqrt{\sum_{i=1}^{n} \boldsymbol{B_i}^2}} \tag{4}$$

Using these equations, we could calculate the Cosine Similarity of vector A and B is equal to

Table 1 Example of Vectorization

Address		JL	MANGGA	DURIAN	MERAH	MANIS	KP	TANAH
JL MANGGA MERAH, KP MANGGA	→	1	2	0	1	0	1	0
JL DURIAN MANIS, TANAH MERAH	→	1	0	1	1	1	0	1

0,41. This is the basic of the Bag of Words model. Later in this paper, this method will be applied to turn a dataset of street addresses into a sparse matrix containing the count of words of these addresses.

3. RESEARCH METHODOLOGY

The methods used in this research is mainly data processing, data cleansing, and fitting the dataset into a machine learning model. This study uses a dataset of correctly assigned Taxpayers' addresses to build an algorithm based on the addresses and supervising zones. And then, predict the other supervising zones using said model.

3.1 Data Collection

The data used in this paper is a sample data containing street addresses from the first five supervising zone in KPP Pratama Jakarta Koja. The data has been labelled as Zone 1 to 5. The data collection method used is using Python selenium web driver to scrape random addresses from Google Maps. This method is used to minimize the utilization of actual Taxpayer's addresses, to prevent any privacy issues.

According to *Introduction to Statistical Learning (*James et al., 2013) to evaluate errors in the model, the data will be split into training set and test/validation set. The test set will be 20% with a random state of 101 in Scikit-Learn Train/Test Split function. Because the amount of dataset is relatively small, dividing it into 80:20 provides us with the maximum amount of the training set, but still maintaining test data that has never been seen by the model during training. With a bigger dataset, it is preferable to divide the data into three parts: training set, validation set, test set. Reproducing the research with different amount of training set and test set might result in a slightly different performance of the model.

3.2 Tools Used

The research will be done in Python environment using some of built-in functions and additional libraries for data processing and machine learning building.

3.2.1. Hardware

The hardware used is computer with the following specifications:
1. AMD Ryzen 7 3750H 64-bit.
2. 8 GB of DDR4 ram.
3. NVIDIA GeForce GTX 1660Ti, Max-Q Design, with 6 GB display memory.

3.2.2. Software

The operating system used in the research is Windows 10 Home 64-bit. The main data processing tool is Python version 3.9.7 with Anaconda distribution. Python is an interpreted multi-purpose programming language, that supports multiple programming paradigm such as functional programming, procedural programming, and object-oriented programming. Python is largely used in the field of data science, data analytics, dan server-side programming. In addition to the built-in functions in Python, these following libraries would be installed:
1. Pandas (version 1.3.4) a data science and big data analytic library. Used mainly to preprocess, manipulate, and transform the dataset.
2. Selenium Web Driver a web scraping and web automation library. Used to automate GoogleMaps and scrape address and coordinate data.
3. Geopandas (version 0.10.2), Pandas' geographical extension for processing geolocation data. Used to match the coordinates scraped from Google Maps to our predefined supervising zone using spatial join.
4. Scikit-Learn (version 0.24.2), machine learning library for Python, containing predefined model that could be tuned depending on our needs. Used mainly to create machine learning model in this study, and to perform vectorization.

Table 2 Sample Addresses Scraped from Google Maps

Address	Label
Jl. Mahoni II 1-17, RT.12/RW.5, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jl. Panda Lestari II 1-3, RT.8/RW.9, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jakarta Utara, RT.5/RW.5, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jl. Anoa Lestari II 3-8, RT.3/RW.9, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jakarta Utara, RT.1/RW.5, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jl. Panda Lestari I 10-16, RT.8/RW.9, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jakarta Utara, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta	ZP 1
Jl. Anoa Lestari I 18, RT.2/RW.9, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jl. Sukapura No.55, RT.3/RW.5, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
JL. Elang, Sukapura, Cilincing, Komplek Walikota Blok A 3/10, Jakarta, RT.12/RW.5, Sukapura, Cilincing, North Jakarta City, Jakarta 14110	ZP 1
Jl. Tipar Cakung No.55, RT.2/RW.5, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1
Jl. Taman Orchard, RT.8/RW.9, Sukapura, Kec. Cilincing, Kota Jkt Utara, Daerah Khusus Ibukota Jakarta 14140	ZP 1

4. RESULT AND DISCUSSION

After using the web scraping method to collect data, about 573 random street addresses are collected. These street addresses are from Zone 1 to 5 in KPP Pratama Jakarta Koja's supervising zone, and all of them are from the Kelurahan Sukapura. These are some examples of the sample addresses in Table 2.

These are not the full addresses used in the research. For convenience, the full data and source codes will be presented in separate files. After collecting the data, next step is to process the data using Python to create a Machine Learning model based on the bag of words theory.

4.1 Creating Basic Model

The basic model consists of three steps, Tokenization, Vectorization, and model training. After those three steps are done, we will put the model into a data pipeline for an easier reusability and model adjustment.

4.1.1 Tokenization

Tokenization is a step of dividing text or document into each word, this will make the text easier to process in the next step, the vectorization. To do the tokenization, I created a Python function, this function will mostly do the following step:

- Remove all punctuations from the text,
- Split the text using into each word by removing the whitespaces in the text and return it as a list.

And I'm going to call this function as *text_process* function.

In Table 3 there are some examples when the function is being invoked to the sample address text data. This function will not be directly invoked to the data, rather, this function will be passed as analyzer in the vectorization process.

Table 3 Example of Tokenization

Address		Processed Address
Jl. Mahoni II 1-17, RT.12/RW.5, Sukapura	→	["Jl", "Mahoni", "II", "117", "RT12RW5", "Sukapura"]
Jl. Panda Lestari II 1-3, RT.8/RW.9, Sukapura	→	["Jl", "Panda", "Lestari", "II", "13", "RT8RW9", "Sukapura"]
Jl. Anoa Lestari II 3-8, RT.3/RW.9, Sukapura	→	["Jl", "Anoa", "Lestari", "II", "38", "RT3RW9", "Sukapura"]

4.1.2 Vectorization

Vectorization as presented in Table 4 is a step of converting the text into vectors. This is done to make the computer machine easier to process the data, since text data are harder to compute. In this step, the research utilizing Scikit-Learn's built-in text processor called CountVectorizer and use self-defined *text_process* function as the analyzer. The CountVectorizer will then convert my text data into a sparse matrix containing the count of each word. This is what called as the bag of words data.

Table **Error! No text of specified style in document.** CountVectorizer's Sparse Matrix Output

	Address 1	Address 2	...	Address N
Word 1 Count	0	0	...	0
Word 2 Count	2	1	...	1
...	...	...	...	...
Word N Count	1	0	...	0

This sparse matrix will be passed as features in the machine learning training step. Meanwhile the already classified supervising zone will be passed as the target label. I assigned the sparse matrix from the vectorization process as a variable called *bow*, a short for bag of words.

4.1.3 Model Training

In the model training, this study will use one of the most popular classifiers for Natural Language Processing (NLP), the Naïve Bayes classifier. The Naïve Bayes classifier is based on Bayes' Theorem developed by Thomas Bayes to describe the probability of an event based on prior knowledge of the condition that might affect the outcome of an event (Joyce, 2003). Bayes' Theorem stated as mathematical equation as follows:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)} \quad (5)$$

Where y and x are events and P(x) is not 0.

- $P(y \mid x_1, \dots, x_n)$ is the probability of event y, given the set of x occur;
- $P(x_1, \dots, x_n \mid y)$ is the probability of the set of x, given the y occurs; and
- $P(y)$ and $P(x_1, \dots, x_n)$ is the probability of y and x occurs in any circumstances.

Using the naïve assumption that $P(y \mid x_1, \dots, x_n)$ is equal to $P(y \mid x_i)$, the equation is simplified into:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)} \quad (6)$$

Where the $P(y)\prod_{i=1}^{n} P(x_i \mid y)$ is the product of all $P(x_1 \mid y) * P(x_2 \mid y) * \dots * P(x_n \mid y)$. Given the input is saved in the model as the training set, the $P(x_1, \dots, x_n)$ is constant, we can use the following classification rule:

$$\begin{gathered} P(y \mid x_1, \dots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y) \\ \Downarrow \\ \hat{y} = \arg\max_{y} P(y)\prod_{i=1}^{n} P(x_i \mid y) \end{gathered} \quad (7)$$

This study specifically uses the Multinomial Naïve Bayes algorithm from Scikit-Learn library. This algorithm supports multiple input as predictor and is largely used to model NLP algorithm. To train the model, this study creates a machine learning object called *zp_model* and assign the Scikit-Learn's MultinomialNB (Multinomial Naïve

Bayes) as the value. Then, fit the *bow* object as the feature (X), and the supervising zone as the labels (y).

4.1.4 Creating Data Pipeline

The bag of words algorithm contains a lot of process, to make the program more reusable, Scikit-Learn has a built-in function called Pipeline to summarize some machine learning steps and contain it in a single object, transforming the three steps process into just one process. By using pipeline, it's easier for the model to be reused and adjusted, just by modifying the parameters or the steps used. For the rest of the research, I will be using this pipeline to adjust the model and try to find best parameter that suits the data.

I created a pipeline object called model_pipeline and used the fit method to train the model. Then, I use the predict method on the object to return the prediction done by the model. This prediction will be used as a comparation for evaluation purpose.

4.1.5 Model Evaluation

As being said above, the data used in training the model is split into training set and testing set using Scikit-Learn's TrainTestSplit function, with 80% data used as training set and 20% as testing set. Using Scikit-Learn's classification_report function, I printed the model's accuracy by comparing the predicted value of the model and the true value of the labels. With this basic model, I got the model accuracy in Table 5.

Table 5 Basic Model Accuracy

	Precision	Recall	F1-score
ZP 1	90%	97%	93%
ZP 2	76%	68%	72%
ZP 3	70%	78%	74%
ZP 4	78%	81%	79%
ZP 5	82%	78%	80%
accuracy			81%

Precision: Number of true positive compared to number of predicted positive
Recall: Number of true positive compared to actual positive value
F1-Score: Comparation of precision and recall calculated as:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (8)$$

From the Table 5 we can see that the model has the accuracy of 81%. Which is good enough but still has room from improvement for the model to be better.

4.2 Adjusting The Model

To improve the model, this study utilizes some different methods and functions that is relevant to the data used in this research.

4.2.1 Using TF-IDF

Term Frequency – Inversed Document Frequency is a statistical method to weight the text based on the occurrence of the term t. Term Frequency is the number of the word appears on a single text/document compared to the total number of words in the text, denoted as follows:

$$TF(t) = \frac{Number\ of\ a\ specific\ word\ appears\ on\ a\ text/document}{Number\ of\ words\ in\ a\ text/document} \quad (9)$$

Meanwhile Inverse Document Frequency is the logarithm of total number of the document, divided by the number of documents in which a term t appears.

$$IDF(t) = \log_{10} \frac{Total\ Number\ of\ Documents}{Number\ of\ Document\ in\ which\ a\ specific\ word\ appears} \quad (10)$$

By doing so, a word that appears in a text a lot of time will be weighted higher because it is considered important, but if a word appears in a lot of other text/document, it will be weighted less since it is considered as a common word.

Using Scikit-Learn's Pipeline, I created a machine learning object that do the following functions:

- Tokenize and vectorize the text using *text_process* function as analyzer
- Weight the vector using the TF-IDF function from Scikit-Learn
- Train the data using the Multinomial Naïve Bayes from Scikit-Learn

With this model, I got the following performance in Table 6. The model's accuracy has been slightly improved by using the TF-IDF from 81% to 84%.

Table 6 TF-IDF Model Accuracy

	Precision	Recall	F1-score
ZP 1	90%	97%	93%
ZP 2	78%	75%	76%
ZP 3	100%	78%	88%
ZP 4	82%	88%	85%
ZP 5	82%	78%	80%
accuracy			84%

4.2.2 Adjusting the Text Process Function

In the Table 3, we can see something odd on the table. Where the number in the street address stated as number 1-17, but after being tokenized, the number merged into 117. It also happens with the RT and RW in the address. The RT.12/RW.5 is merged into RT12RW5. This happens because we plainly remove any punctuations from the text, so that after a punctuation is removed, any remaining words around the punctuation are merged.

The *text_process* function is adjusted as in Table 7 and created *text_process1* function as replacement. This function does anything the old function does, but given some additional steps by splitting the street number e.g. "1-17" into "1","7" and splitting the "RT.X/RW.Y" into "RTX","RWY". Keep in mind that we don't want to split the RT and RW into "RT","X","RW","Y". Since, it will result in weird behavior where "RT 3 RW 4" will have 50% similarity to "RT 7 RW 8", since both has the word RT and RW. By dividing it into "RTX","RWY" each RT and RW is counted as one part with the number instead of being divided. In addition to that, I also added some step to remove some common words like "Jakarta Utara", "Jalan", "Jl" since all the document has that word in every of them. Using *text_process1* function as replacement of the *text_process*, I got the following performance in Tabel 8. The accuracy has been greatly increased from 84% to 93% using the adjusted *text_process1* function.

Table 7 Adjusted Text Process Function

Address		Processed Address
Jl. Mahoni II 1-17, RT.12/RW.5, Sukapura, Jakarta Utara	→	["Jl", "Mahoni", "II", "1", 17", "RT12", "RW5", "Sukapura"]
Jl. Panda Lestari II 1-3, RT.8/RW.9, Sukapura	→	["Jl","Panda", "Lestari", "II", "1", 3", "RT8", "RW9", "Sukapura"]
Jl. Anoa Lestari II 3-8, RT.3/RW.9, Sukapura, Jakarta Utara	→	["Jl", "Anoa", "Lestari", "II", "3", "8", "RT3", "RW9", "Sukapura"]

Table 8 Model Performance Using Adjusted Text Process Function

	Precision	Recall	F1-score
ZP 1	94%	100%	97%
ZP 2	93%	89%	91%
ZP 3	100%	78%	88%
ZP 4	93%	96%	94%
ZP 5	91%	91%	91%
Accuracy			93%

4.3 Testing Using Different Classifier

In this research, in addition to Multinomial Naïve Bayes classifier, this study also used some other classifier to test which classifier has the highest performance (Sasaki, 2007). The classifiers used in this research are Random Forest classifier and k-Nearest Neighbor classifier. The other parameter will be using our latest model, thus utilizing the new *text_process1* function and TF-IDF.

4.3.1 Using Random Forest Classifier

Random Forest as illustrated in Figure 2 is a classifier based on the Decision Tree classifier. Decision Tree as illustrated in Figure 1 is a set of Boolean value of True or False representing presence of particular feature of a classification.

However, Random Forest uses a set of multiple decision tree to make the model more flexible and minimize the error. Using the Random Forest classifier, the model has accuracy of 88%, slightly lower than our latest model using

Multinomial Naïve Bayes. Illustrated in the classification report in Table 9.

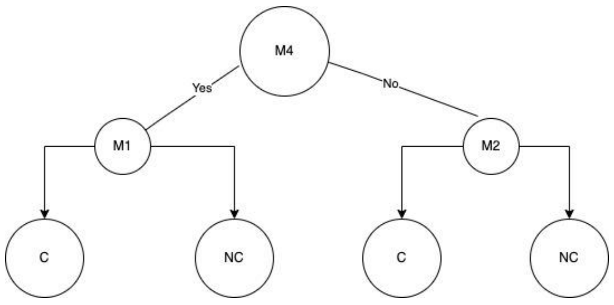

Figure 1 Illustration of Decision Tree

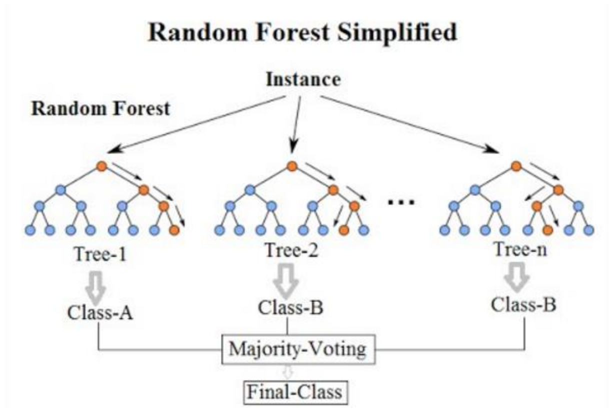

Figure 2 Random Forest Illustration

Table 9 Random Forest Accuracy

	Precision	Recall	F1-score
ZP 1	94%	100%	97%
ZP 2	82%	82%	82%
ZP 3	71%	56%	63%
ZP 4	92%	88%	90%
ZP 5	88%	91%	89%
accuracy			88%

4.3.2 Using *k*-Nearest Neighbor Classifier

K-Nearest Neighbor (KNN) is a classifier used in statistical learning to determine a class of a data based on the "nearest" class in a diagram. The distance is measured using actual distance of each data in the diagram. KNN chooses *k* number of neighbor and determine which class is present more in the dataset. For a better understanding, here is how KNN works in illustrated diagram:

In the Figure 3, if we try to determine the circle's class, we can choose the number of *k*-Nearest Neighbor to classify it. If we use number of *k* = 3, 3 nearest neighbors are 1 square and 2 triangles, making the square is classified as triangle. in the other hand, if we choose the number of *k* = 5, the nearest neighbors are 3 squares and 2 triangles, making it classified as square.

Choosing the number of *k* used in the KNN is the most crucial thing. Defining the number of *k* too low will make the model too flexible and a little change will affect the model in a large way, and vice versa. That's why, I tested the number of *k* between 1-40, collect all list of the error and used the elbow method to determine the right amount of *k* to use.

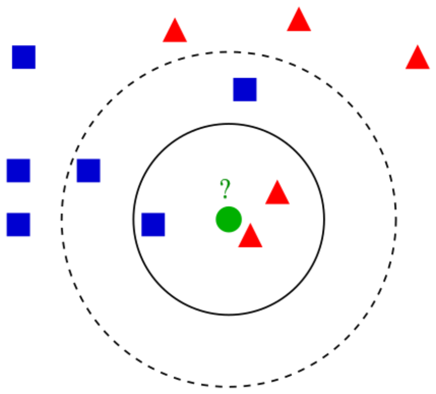

Figure 3 KNN Illustration

In the Figure 4 we can see the error rate drops from high to low and then constantly having a considerably low error rate, making shape of a folded elbow. The elbow method encourages us to pick a number which is closest to the elbow, in this case the number is 10. Picking the number too far to the left will result in the low model accuracy but picking number too far to the right will most probably overfit the model. Thus, this study took the number 10 as *k*-value which results in 91% of

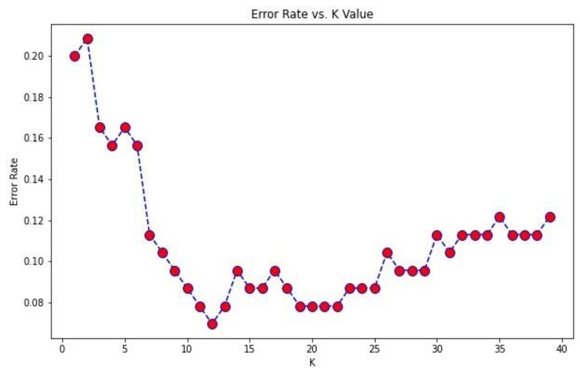

Figure 4 Elbow Method to Determine the Number of K

accuracy as presented in Table 10, very close to the Naïve Bayes model I used earlier.

From these results, we can compare the performance of each model. KNN and Multinomial Naïve Bayes thrives in this research. This implies that the two algorithms perform really well when measuring the similarity of each address. Random forest in the other hand, shows a lower performance, this issue most probably caused by the random nodes chosen by the model, thus creating a bias, resulting in slightly lower accuracy.

However, even though the accuracy of different algorithms may vary, but the differences are not too big. Restructuring the text preprocessor in the other hand, caused a significant change in accuracy. This implies that preprocessing the text "correctly", gives a bigger impact rather than deciding which algorithm to use. With this accuracy, we can confidently implement the model into real world problem in predicting the supervising zones of the taxpayers.

Table 10 KNN Model Performance

	Precision	Recall	F1-score
ZP 1	93%	97%	95%
ZP 2	89%	89%	89%
ZP 3	100%	78%	88%
ZP 4	89%	92%	91%
ZP 5	91%	91%	91%
Accuracy			91%

5. CONCLUSIONS

Based on this research, it is safe to say that that we could create a model to classify Taxpayers' supervising zone using Natural Language Processing with a considerably high accuracy. The most effective model is the one using TF-IDF, adjusted text processing function, and using Naïve Bayes classifier. It's fairly predicted since Multinomial Naïve Bayes is the most popular classifier used in NLP. Also stated in the documentation page of Naïve Bayes on Scikit-Learn's website, Multinomial Naïve Bayes works best when faced with count vectorized data or TF-IDF data.

The most crucial thing in creating the model is processing the text into an acceptable feature to be fitted in the model. This is proven that when the text processing function is adjusted, the performance jumped from 84% to 93% which is very impressive. However, the method of text processing cannot be directly applied to differently structured text data. Thus, a high understanding on the data structure, how changes on the feature affects the model, and a clear vision on what the purpose of the model is a really important key on producing a high-quality machine learning model. Reciting a saying in the machine learning community, "Trash in, Trash out", which means whatever data we put in the model is the most determining thing on the model quality. That's why, most process of machine learning is usually spent on data preprocessing.

With high accuracy in this model, this model can be implemented in the real life when assigning Taxpayers to the respective Account Representatives in KPP Pratama Jakarta Koja. However, this should be furthermore discussed with the higher authority and decision makers such as Office's Executives. Since, reassigning and rearranging the Taxpayers will results in change in the revenue collection policy.

6. IMPLICATIONS AND LIMITATIONS

The model created in this research is limited to be used in KPP Pratama Jakarta Koja only, since the sample data used are addresses around KPP Pratama Jakarta Koja's supervising area. This model is also limited to the data which is divided into each ward. Using this model to predict all possible supervising zone in a Tax Office will result in different (most probably lower) performance and accuracy.

To counter this limitation, users should be dividing the data into each Kelurahan before using the model. Or, if some Kelurahan is unknown, a model to classify street addresses into each Kelurahan first can be created. Using a multi-layered model is way better than using a single model to do all the classification.

REFERENCES

[1] Directorate General of Taxes. (2022). Nota Dinas Direktur Ekstensifikasi dan Penilaian No. ND-56/PJ.06/2022.

[2] Giller, G. L. (2012). The statistical properties of random bitstreams and the sampling distribution of cosine similarity. *Available at SSRN 2167044*.

[3] Gareth, J., Daniela, W., Trevor, H., & Robert, T. (2013). *An introduction to statistical learning: with applications in R*. Spinger.

[4] Joyce, J. (2003). Zalta, Edward N,(Ed.)," Bayes' Theorem.

[5] McTear, M. F., Callejas, Z., & Griol, D. (2016). *The conversational interface* (Vol. 6, No. 94, p. 102). Cham: Springer.

[6] Sasaki, Y. (2007). The truth of the f-measure. 2007. *URL: https://www. cs. odu. edu/mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf [accessed 2021-05-26]*, *49*.

[7] Zhang, W., Zhang, Y., Gu, X., Wu, C., Han, L., Zhang (2022). Machine Learning and Applications. *Application of Soft Computing, Machine Learning, Deep Learning and Optimizations in Geoengineering and Geoscience*, 21-39.