Machine Learning

MOD006562

Faculty: Science and Engineering

School: Computing and Information Science

Academic Year: 2023/24

Trimester: 2

SID:2203834

Introduction

In this reflection report I will go through the steps I took to develop three Natural Language Processers (NLP) specific to the data I was provided. I will demonstrate my understanding of the key concepts underpinning Machine learning (ML) such as data structures, feature extraction, creating and training a range of supervised learning model, and analysing these models.

The task provided to me was to develop three end to end machine learning pipelines based on the provided dataset of News categories.

Data pre-processing and preparation

I was provided with a dataset of News Categories. This dataset consisting of over 200,000 rows of data, these contained categories, headlines, authors, links to the news article, date of the news article, and a short description of the news article. This data was categorised into 41 different categories, including but not limited to: ‘CRIME’, ‘PLOITICS’, WORLDPOST’, and RELIGION’.

I first read through the dataset to get insight into what was included in each column and what features of the data could be used to classify the categories most effectively. I was also looking for anomalous data that could harm the training of my chosen supervised models.

During this step I found that many of the columns in the dataset were of no importance to the task and could potentially confuse the model, I proceeded to remove these columns. I removed the ‘authors’ column as I found some others belonged to many different categories and so was directly linked to the category. I removed the ‘links’ column a link could belong to many different categories. I also removed the ‘date’ column for the same reason for removing the ‘links’ and ‘authors’ columns.

A white background with black text

Description automatically generated

Figure 1: image of code removing unnecessary columns

Aswell as finding columns that needed removing, I found that there were two columns that could prove vital in identifying the category. These were the ‘headline’ and ‘shourt\_description’ columns. I identified the usefulness of the ‘short\_description’ column, through reading the short description and finding that I could predict the category by using the context of the words. I also saw that each piece of data had an individual headline, and that by reading the headline I could predict the category.

I also found that some rows of data where empty or duplicates of other rows and so I decided to remove these.

Before moving on to normalization I wanted to make the amount of data for each category as even as possible, I did this by finding the range most categories lay in, which was from 2000 datapoints to 6000 datapoints. I then decided to “down sample” (Meijering, 2002) the number of data points, making the range smaller by 2000, as I wanted to improve the performance of the algorithm and ensure that there was fair representation across variables. This left me with 21 of the 41 original categories.

A graph of numbers and letters

Description automatically generated

Figure 2: graph of dataset, where y Is the categories and x is the number of rows in each category, before down sampling.

A bar graph with text

Description automatically generated

Figure 3:graph of dataset after down sampling

The next step was to normalize the data. The normalization process consisted of removing punctuation, symbols and converting all characters to lowercase. This was necessary as there were many symbols that held no relevance to the short descriptions and headlines, making all characters lower case was important as it ensures that characters that are the same and removes case-dependent biases and creates continuity. Another part of my normalization process was to remove stop words, this consists of removing words that are common and occur frequently, some examples of stop words include "the", "is", "and", "in", "of". “Most text and document data sets contain many unnecessary words such as stop words, misspelling, slang, etc. In many algorithms, especially statistical and probabilistic learning algorithms, noise and unnecessary features can have adverse effects on system performance.” ( Kowsari, 2019). Removing stop words improves the accuracy and performance of the model as stop words tend to dominate the frequency distribution of words and so take away emphasis from words that may be of greater importance to the sentence if these words of greater importance are less frequent.

A computer screen shot of a computer code

Description automatically generated

Figure 4:image of code changing text to lower case, removing punctuation, and removing symbols

A computer screen shot of a computer code

Description automatically generated

Figure 5: image of code removing stop words

A graph with text on it

Description automatically generated

Figure 6: image of data after stop word removal

Another part of my normalization process included lemmatizing and stemming. Lemmatization looks at the word and converts it to its dictionary form based on the words meaning resulting in the new word being a valid word with the same meaning. Stemming is different from lemmatizing as stemming removes the prefixes and or suffixes from a word resulting in a word that may not be a valid word. This difference between the steps influenced me to lemmatize the data before stemming as I could first take the dictionary form of the word, keeping the meaning of the word, and then shorten the word by removing prefixes or suffixes. This helps improve the models performance by changing words that hold the same meaning into the same word, increasing the frequency of the word and putting more emphasis and weight onto the meaning of the word rather than the word itself. This also helps the model as it helps reduce the time it takes to train the model.

A computer screen shot of a computer code

Description automatically generated

Figure 7: image of lemmatization code

A computer screen shot of a computer code

Description automatically generated

Figure 8: image of code for stemming

After pre-processing the data and removing any rows that had missing data and null values, I found I had significantly reduced the amount of categories and data points for each category. For my first model that used the ‘short\_description’ I was left with 12 of the 41 categories and a range of 1750 to 3500 datapoints. For my second and third models that used ‘headline’ I was left with 4 categories with 3500 datapoints each.

A screen shot of a computer code

Description automatically generated

Figure 9: image of code setting all columns to have 3500 datapoints

A blue bar graph with white text

Description automatically generated

Figure 10: graph of data after pre-processing

Preparing data

When preparing the data for fitting to a ML model I used the function “test\_train\_split” from the “sklearn.model\_selection”, this is a function commonly used in machine learning for splitting a dataset into a subset for training the model and a subset for testing the model. I used a split of 80% training data and 20% testing data. For my first model I vectorized the text data in the ‘short\_description’ before splitting the data, and encoded the labels (‘headlines’ column) into numbers rather than text. My reason for this was that popular libraries such as scikit-learn and TensorFlow require numerical inputs and cannot directly handle categorical or textual labels. I used cross-validation to help detect overfitting by assessing the model's accuracy and comparing it to the validation accuracy, and looking at how this changed over multiple epochs.

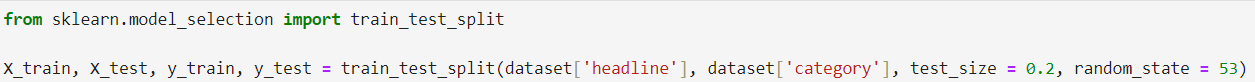


Figure 11: image of test train split

Vectorization of data

Model 1 - Word2Vec

In my first model I decided to use Word2Vec. Word2Vec is a word embedding technique that represents words in a continuous vector space where each word is mapped to a high dimensional vector. It is trained on a large text dataset to learn how to capture the semantic relationships between words based on their context and represents words as dense vectors with words that hold similar meaning closer together. I decided to use this technique for its emphasis on the contextual and semantic between words, making it well-suited to the task of vectorizing a short description. Numerous studies have demonstrated the effectiveness of Word2Vec in various applications relating to the task given, such as sentiment analysis, and document classification (Mikolov, 2013).

A screenshot of a computer program

Description automatically generated

Figure 12: image of code vectorizing text using Word2Vec

Model 2 - HashingVectorizer

In my second model I decided to use hashing. Hashing works by mapping each unique word in a piece of text to a hash value using a hash function that converts the word into a numerical value which is then used as an index in the feature vector. Unlike Word2Vec, which represents data as high-dimensional vectors, hashing produces fixed-size vectors, making it computationally efficient. I decided to use this vectorization technique as I wanted to reduce the training time and resource consumption of the model, as I found word2vec was very resource heavy.

A screenshot of a computer program

Description automatically generated

Figure 14: image of code vectorizing text using hashing

Model 3 - Tf-idf

In my third model I decided to use Term Frequency-Inverse Document Frequency(TF-IDF). TF-IDF works by measuring how often a word appears in a peace of text (TF), and how important the word is to the entire set of data(IDF), then taking the product of TF and IDF to find the importance of the word in the text relative to the dataset. Words with a higher TF-IDF score are more significant to the text. I decided to use TF-IDF as it is a commonly used and proven vectorization technique when crying to classify text data, as it identifies it identifies the important terms and discards terms that are common across the dataset as a whole, reducing the computational power required to train the supervised learning model.

A screenshot of a computer code

Description automatically generated

Figure 13: image of code vectorizing text using TF-IDF

Training models

Model 1: LSTM

My first model was an Long Short Term Model (LSTM) model. LSTM is a type of recurrent neural network, used primarily for sequential data, such as text, timeseries, audio and video data. LSTMs are equipped with memory cells that allow them to retain information over extended time steps. Each memory cell is comprised of a cell state and a hidden state. The cell state carries information through time, while the hidden state serves as a short-term memory. LSTMs incorporate three gates, the forget gate, input gate, and output gate, these control the flow of information. The forget gate determines what information from the previous cell state should be discard, the input gate regulates the incorporation of new information into the cell state, and the output gate generates the next hidden state. These gates enable LSTMs to selectively remember or forget information at each time step. My reasoning for using an LSTM model was that I wanted the model to be trained on the context of the text rather than the individual words.

I started by transforming the vector and categories columns into lists, I did this as it was necessary to move on to the next step. I then tokenized the vector and converted it to a sequence for it to be fed into the training and testing split. I also encoded the categories using one-hot encoding to represent the categorical variables as binary vectors for the same reason.

I went on to creating the model using an embedding layer as I used word2vec as the vectorization technique and wanted map the discrete entities from a high-dimensional vector into a lower-dimensional vector to more easily capture and understand the relationship between entities. I then added a bidirectional LSTM layer with a dropout of 20%. Dropout works by randomly setting a certain proportion of neurons in the network to zero, I decided to use a dropout layer to prevent overfitting. I also added some dense layers to adjust the biases and weights with the aim of minimizing the difference between the predicted output and the true labels.

A screenshot of a computer program

Description automatically generated

Figure 14: image of code creating the model 1

A screenshot of a computer

Description automatically generated

Figure 15: summary of model 1

A screenshot of a computer code

Description automatically generated

Figure 16: image of epochs with training and testing accuracy and loss

As shown in figure 16, as the accuracy of the model when predicting the categories using the training data rose significantly, from 10% to 95%, and the loss lowered at a substantial rate, however the model struggled to accurately predict categories using testing data, with a validation accuracy of 9% in the first epoch and 11% in the last epoch.

A blue and white squares with numbers

Description automatically generated

Figure 17: confusion matrix of model 1

Figure 17 shows the confusion matrix of the true classes and predicted classes, confusion matrixes are a good way to assess the performance of a machine learning model, organising predictions into four categories, true positive, true negative, false positive, and false negative. As seen in figure 17 each class has almost as many false positives and negatives as true positives and negatives. Though the confusion matrix shows most classes are incorrectly predicted it does show which were more correctly predicted than others, for example the true and predicted category two has 110 true positives and 38 true negatives, false positives, and false negatives showing it has a higher accuracy than most other classes.

Model 2: KNN

My second model was a K-Nearest Neighbours (KNN) model. KNN is a simple machine learning algorithm used for classification and regression tasks. The algorithm operates under the assumption that similar data points belong to the same class or have similar output values. To make predictions with KNN, the algorithm computes the distance, typically Euclidean distance, between the new data point, given by the testing data, and all other data points in the training dataset. Once the distances are computed, the k nearest neighbours to the new data point are identified. The class label of the new data point is then determined by the majority class (or average value) among its k neighbours.

A screenshot of a computer code

Description automatically generated

Figure 18: image of code creating the model 2

Figure 18 shows my implementation of the KNN model. I started by importing the “KNeighboursClassifier” class from the neighbours model of scikit-learn library. I then determined the number of neighbours, I ended up making this number 119 through trial and error as I found any fewer or higher would decrease the accuracy, and the higher the difference from the number I had chosen,119, from any other number, the more the accuracy would drop by.

The model was then fitted with training data from the test train split. The train test split was made up of the vectorized text data from the heading column, using the hashing vectorization method, and the labels. Once the headings column went through preprocessing, each peace of data from the heading column belonged to one of four remaining categories.

A number with black lines

Description automatically generated with medium confidence

Figure 19: image of confusion matrix, with x being the true label and y being the predicted label and accuracy of model 2

As seen in figure 19 each class has a large number of true positives and negatives. Though the confusion matrix shows most classes are well predicted it also shows that the model misclassified the first label as every other label over 100 times for each, this could also shows that the model was only lucky when it predicted the first label correctly with the highest accuracy.

Model 3: SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm. Its objective is to find the optimal hyperplane that best separates different classes in a high-dimensional feature space. SVM works by identifying the hyperplane that maximizes the margin between classes in the feature space. The margin is defined as the distance between the hyperplane and the closest data points from each class, known as support vectors. SVM aims to find the hyperplane that not only separates the classes but also generalizes well to unseen data, to prevent over fitting. SVM can handle linearly separable data using linear kernels or non-linearly separable data using kernel tricks like polynomial, radial basis function (RBF), or sigmoid kernels, which map the data into higher-dimensional spaces where linear separation is possible.

A screenshot of a computer program

Description automatically generated

Figure 20: image of code creating model 3

Figure 20 shows my implementation of the SVM model. I started by importing the “svm” class from the sklearn library. I then experimented with the parameter “C” to get the best results out of the model. Parameter “C” represents the regularization parameter, and so by changing “C” you change the trade-off between maximizing the margin and minimizing the classification error. I ended up making “C” equal to four as I found this resulted in the greatest accuracy overall, however making “C” equal to three increased the accuracy in predicting one of the classes whilst reducing the accuracy of the other.

The model was then fitted with training data from the test train split. The train test split was made up of the vectorized text data from the heading column, using the Term Frequency-Inverse Document Frequency vectorization method, and the labels. Once the headings column went through preprocessing, each piece of data from the heading column only belonged to one of four categories.

A number with black text

Description automatically generated with medium confidence



Figure 21: image of confusion matrix and accuracy of model 3

As seen in figure 21 each class has a large number of true positives and negatives. Unlike the first 2 models the confusion matrix shows the SVM model to have no or very little bias in its predictions, classifying correctly and incorrectly one class as much as the next. This is proven by the number of false negatives (FN) and false positives (FP) of each class with FN ranging from 73 to 104 and having a mean of 82.5 and FP ranging from 78 to 112 and having a mean of 97.

Results and Evaluation Analysis

The task given to me has taught me many methods of developing a supervised learning model and methods of preprocessing data. The three models I developed have resulted in differing results. As the project went on I learnt and developed better techniques, leading me to find methods that yielded better results.

My first model was developed at the start of my journey on this course and so was built with my base knowledge of how a text classifier should be developed. I then leart from the results of this model and adapted it to work more efficiently and yield better results. I saw the need to improve my model as when looking at the results I found the accuracy when making predictions given new data was very low, sitting around 11%.

By looking at the confusion matrix in figure 17, it is evident that the supervised model incorrectly predicted much more of the testing set than what was correctly predicted, one reason for this could be due to poor generalization. Generalization refers to how well the model performs on unseen data. I believe this to be one of the causes for the model's low accuracy as my model performed well on the training set but poorly on the test and validating set, indicating poor generalization. Another reason could be that the text data I decided to use was too ambiguous. I believe ambiguity to be the main cause of the misclassifications, as by reading the dataset the column of text data I chose to train the model, ‘short\_description’, I struggled to decipher which category belonged to which description.

My second model was greatly influenced by the results of the first model. I decided to reduce the ambiguity of my input data by using the ‘headings’ column, which contained less words but was much more precise than the ‘short\_descriptions’ column I used in model 1. This resulted in the model being trained on only four categories. The accuracy of my second model was much higher with an accuracy of 72.89%.

By looking at the confusion matrix in figure19, it is evident the model did very well in correctly predicting three of the four classes. The testing set of data belonging to the first class was predicted as every other class almost as many times as it was correctly predicted. There could be many reasons for this, including noise in data, inappropriate choice of k data sparsity or lack of relevant features. I believe the misclassifications are due to overlapping class boundaries. My reasoning for this is that KNNs rely on proximity of data points in a feature space and so the first class may have overlapped with the other classes causing the model to mistakenly classify it for these other classes almost an even number of times.

In my third model I used what I learnt from the first and second model. I used the same text column, ‘headings’, as it removed ambiguousness from the training and testing sets, I also decided to use a different supervised learning technique to avoid the overlapping that occurred in the second model due to the use of KNN. By using an SVM model I was able to achieve an accuracy of 88%.

In conclusion, by evaluating the results of each model I was able to identify the most appropriate algorithm to complete the task of creating a text classification model. I was able to do this by looking at the results of the model and identifying the most likely reason for the results based on my knowledge of how the machine learning models operate.

References

Meijering, E., 2002. *“A chronology of interpolation: From ancient astronomy to modern signal and image processing”.* Proceedings of the IEEE, 90(3), pp.319-342.

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L. and Brown, D., 2019. “*Text Classification Algorithms: A Survey”*. Information, 10(4).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013. “*Distributed representations of words and phrases and their compositionality”*. In: Advances in Neural Information Processing Systems, pp.3111-3119.

Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A.J. and Attenberg, J., 2009. “*Feature Hashing for Large Scale Multitask Learning”*. In: Proceedings of the 26th Annual International Conference on Machine Learning. [online] pp.1113-1120.

Graves, A. & Schmidhuber, J., 2005. “*Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks”*, 18(5–6), pp.602–610.

Cover, T. and Hart, P., 1967. “*Nearest neighbor pattern classification. IEEE Transactions on Information Theory”*, 13(1), pp.21-27.

Cortes, C. & Vapnik, V., 1995. “*Support-vector networks. Machine Learning”*, 20(3), pp.273–297. Available at: <https://doi.org/10.1007/BF00994018>.