**News Category Classification using NLP**

**Mann Purohit Foram Shah**

**purohit.man@northeastern.edu shah.fo@northeastern.edu**

**Abstract**

Natural Language Processing is intriguing because it is a technology through which we can train machines to learn the “Natural Language of humans”, like text and audio, and generate predictions on the trained model. The traditional approach to generate best predictions, and hence best accuracy, is by improving a machine learning algorithm, while keeping the input data unchanged. But, even a simple algorithm can have great accuracy if it is fed with good, preprocessed data. So, rather than focusing only on developing better algorithms, this report will also focus on how to use data centric approaches to increase the efficiency and accuracy of machine learning models.

**Introduction**

News classification from text has exciting real world applications that can be implemented impactfully to leverage user experience through recommendation. For example, if a user regularly reads financial articles, we can perform news classification using NLP on all the available news and classify them into different categories using a machine learning model. Then, we can recommend news classified as “financial” to this user. Traditional methods involve monitoring the activity of a user manually, or asking a user in advance on the type of news to recommend. Our project aims to automate this process of classifying news. The input for predicting a news category will be textual data in the form of a short description of the news and it’s title. We use and compare different methods like TF-IDF from sklearn and Tokenizer from keras to transform text data into word vectors. We then use 3 machine learning algorithms - Multinomial Naive Bayes, Vanilla (Simple/Plain) Neural Networks, and Recurrent Neural Networks to predict a news category for the given input. We use accuracy as the performance metrics to evaluate the above models.

**Previous Research**

**Inoshika et al. (2013)** [[1](https://ieeexplore.ieee.org/iel7/6547259/6553871/06553926.pdf?casa_token=-6N_3__PNkcAAAAA:U9WzXhqz-ivTAjjyvGoM7AbqL_av1cdSuirDGo0TIaCKQHTwegNj0yWDCK5TtfHAT93vZw)] put forward a machine learning model using the Support Vector Machines (SVM) algorithm which aimed to predict a news category based on the given text data. Each data point contained a short message and among each message, each word was considered an input feature. They used the bag-of-word approach to convert these textual input features to numeric form.They argued that SVMs are good when input data has high dimensions. They used Recall as the performance metrics. Their model had varying recall scores for different categories, where the best score was only 89% whereas the worst score was 39%, which indicates not an overall good fit.

**Lai et al. (2015)** [[2](https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewPaper/9745)] compared traditional machine learning (ML) algorithms (**SVM and Logistic Regression**) with the modern Neural Networks (NNs) with modern and complex NN architectures (**RNNs and CNNs**). They used 4 datasets for comparison and showed that modern NNs outperformed the traditional ML algorithms in NLP by some margin. The datasets contained textual data that aimed at predicting a category. Advanced NNs on average had 7% more accuracy.

**Hughes et al. (2017)** [[3](https://arxiv.org/pdf/1704.06841.pdf)] used different combinations of converting words into vectors with different machine learning algorithms and put forward a comparison between them. They proposed a CNN built on top of Word2Vec algorithm (trained on medical dataset). Accuracy was the performance metrics used to evaluate different approaches, and their proposed model received an accuracy of 68%, while other traditional machine learning approaches had accuracy below 50%, which is quite significant.

On comparing the above techniques, we find that the approach from **Lai et al. (2015),** that uses Hybrid Neural Network formed combining RNN and CNN, performs superiorly better than other algorithms. This is because both, CNNs and RNNs, have the ability to extract meaning and understand patterns from sequential data, and in NLP, the data is always sequential, whether it is text or audio.

**Explaining and Transforming Dataset**

We use a kaggle dataset that contains more than a hundred thousand samples, and has 3 columns in the form of **i) label** - denotes news category, we have 4 news categories where 0 is for “World”, 1 is for “Sports”, 2 is for “Business” and 3 is for “Science/Technology” **ii) title** - title for a news article, and **iii)** short description - an overview and outline of the news article. The last 2 columns contain textual data and serve as input to the machine learning algorithms, while the first column contain the target values. Hence, this is a Supervised Machine Learning problem. For example - let **i) title** = “who Messi is”, **ii) short description** = “Messi is the best football player in the world” and **iii) label** = “Sports”. Given the short description and title for the above example, our aim is to convert this textual data into numeric vectors and feed it to a machine learning algorithm to output prediction as “Sports”.

To achieve this goal, we first start by combining the **“short description”** and **“title”** columns to generate a new column named **“text”**. Weremove the previous 2 columns and only use the “text” column for further operations. Now, we perform data cleaning using **“Regular Expressions”**, in which we first remove all the special characters, digits, emojis, and any other text that does not contain alphabets and convert this text to lowercase. We do so because they are insignificant while predicting the output, and keeping them results in increased dimensionality, which results in expensive computations (it should be avoided). We then remove extra spaces and stopwords, they are words which do not add any important information to a sentence, for example “is” and “the”. After this step, the corpus (corpus is a combination of all the textual data present in the samples) will have words that contain alphabets only.

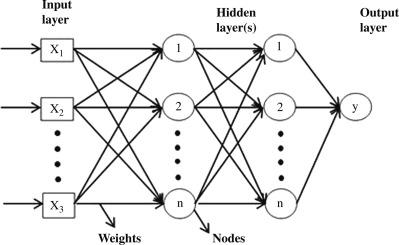
We move on to data preprocessing, where we have 2 techniques to preprocess text data - **i) Stemming** and **ii) Lemmatization**. The aim of both the techniques is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form[[4](https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html)]. Stemming is a process of chopping off the ends of words in order to achieve the above goal and it also often removes derivational affixes. Lemmatization refers to the process of using a vocabulary and morphological analysis to reduce the inflectional forms of words and return a word’s base form, known as *lemma.* Stemming performs the following mapping - {car, cars, cars’, car’s} => {car}, while lemmatization aims to map {am, are, is} => {be}. We use lemmatization over stemming as chopping off words from the end may change a word's interpretation. This completes cleaning and preprocessing the data.

The next step is to convert the textual data into numeric vectors and there are 4 different techniques for doing so. **i)** **One-Hot-Encoding** - this method converts every unique word present in the corpus (corpus refers to the combination of all the text in the data points) into a column. For each word present in the data point, the value of the corresponding column will be 1 and else it will be 0. This generates a very large dataset for us, as we have 120,000 rows and nearly 36,000 unique words, resulting in a sparse matrix of 120,000 X 36,000. This method is highly inefficient and hence we do not include this in our work. **ii)** **Term Frequency - Inverse Document Frequency -** We assign each word a numeric score to signify its importance in the corpus. Let d denote a document (document refers to the text data for a particular data point) and w denote a word. Term Frequency (TF) for a word w in a document d measures the frequency of w in d. **TF(w, d) = (frequency of w in d) / (total words in d)**. Document Frequency, DF(w) measures the count of occurrences of w in an N document set. Inverse Document frequency, IDF is, **IDF(w) = N/DF**. Finally, **TF-IDF = TF(w,d) \* log(N / (DF(w)+1))**. We perform the TF-IDF operation on all the unique words present in our corpus and assign a numeric score to them using the above formula. Then we only keep the top 10,000 words with the highest score, which limits the **vocabulary size to 10,000**, and other words are treated as **0** or **<UNK>** (**unknown**). **iii) Tokenizer -** Tokenizer vectorizes a text corpus by transforming textual data into sequence of integers (where the integer is the index of the token in a dictionary)[[5](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer)]. **iv) Word Embeddings -** Word Embedding is a way of training a model to learn to map a set of words to numeric vectors. They are used to reduce the dimensionality of the input data, which for us is 120,000 X 10,000 using TF-IDF, to only 120,000 X 32, when we use embedding size of 32. Embedding size is the number of clusters/features into which the entire corpus is mapped. Instead of keeping word vectors of size 10,000, we convert each word vector to an embedding vector using the Embedding weights matrix. The **Embedding Weights Matrix (EM)** is used to map words to embedding vectors. We multiply the word vector of size 10,000 with the Embedding Matrix of size (10,000, Embedding size) to generate an embedding vector for each word. For example, a word embedding vector for a word present at index 0 for a tokenizer is calculated using the below formula - We use Word Embedding in Vanilla and Recurrent Neural Networks only.

**Methods used (Machine/Deep Learning models)**

**Multinomial Naive Bayes** is the first algorithm that we introduce. It is a probabilistic machine learning algorithm that takes in prior knowledge as input and computes conditional probability. Let X denote a document from a total of N documents, and y denote one of the 4 news categories. We compute the probability of a document X being in class y as below - where is the conditional probability given y. **Vanilla Neural Network** - A **Neural Network (NN)** is a computing system with an architecture such that every node/element in the architecture is arranged in layers and each of them is connected to its previous and next layers. There are 3 basic types of layers in Neural Networks - **i) Input Layer**, the layer from where we feed the input data, **ii) Hidden layers** - There can be multiple hidden layers, where neurons in each layer are connected to the previous and next layers’ neurons. All the preprocessing happens in this layer, where the hidden layer receives an input that is used to calculate a weighted sum (using Weights W, and Biases B), and this sum is then given to an activation function that performs non-linear operation to help the NN learn complex functions. Lastly we have an **iii) Output Layer**, from where we get the final prediction.

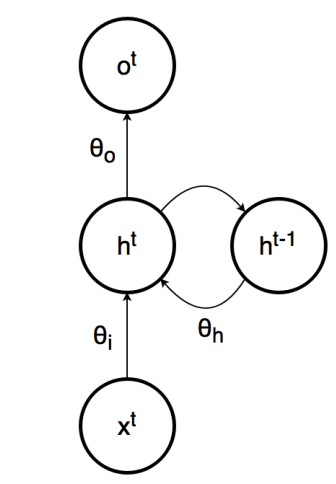
Below image illustrates the architecture of a Neural Network and shows how the output is computed in a process called **Forward Propagation.**

****

Below is the categorical sparse entropy loss function we use to compute the error and gradient.

where y and p are the true and predicted value, respectively, for the class. Cost of the model is, where W, and b are weights and biases of the model and m is the total number of samples.

The NN trains its weights and biases in a process called **Backward Propagation.** is the learning rate of an optimization algorithm which decides how strong should be the step taken to change the weights and biases of the model. We compute Back Propagation as follows, where dW and db are the gradients that the optimizer computes on the loss function that we defined above.

**Recurrent Neural Networks -** Recurrent Neural Networks are advanced architectured NNs that are specifically designed for sequential data like text and audio. Unlike simple Neural Networks, RNNs can memorize the output from the previous nodes, which is then fed to predict current nodes’ output. RNNs have hidden layer connections back to the same hidden layer which allows the states of the hidden layers at one time instant to be used as input to the hidden layers at the next time instant[[6](https://brilliant.org/wiki/recurrent-neural-network/)]. Below is a graphical model for RNN

**(** denote the parameters associated with input, output, and previous hidden layer states respectively).

RNN uses the following equations to perform computations :-

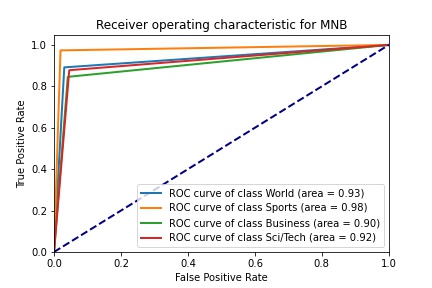
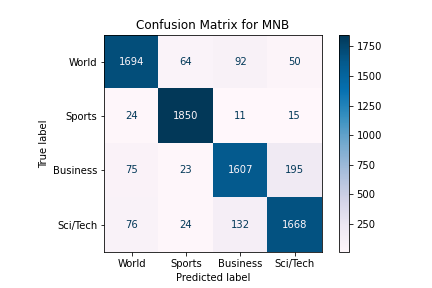
**Experiment/Results/Discussion**

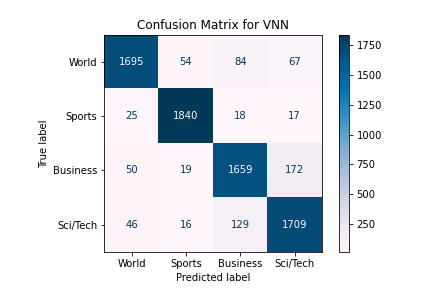
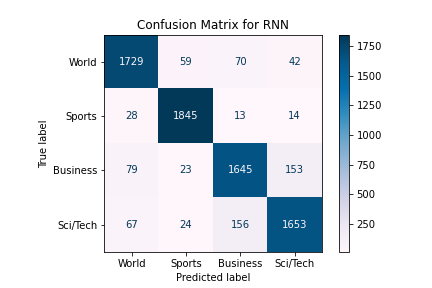
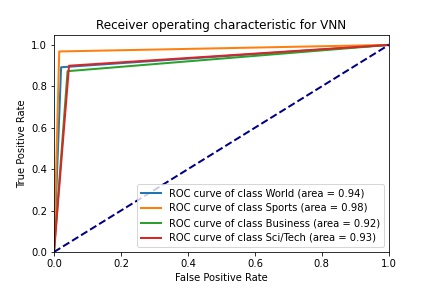
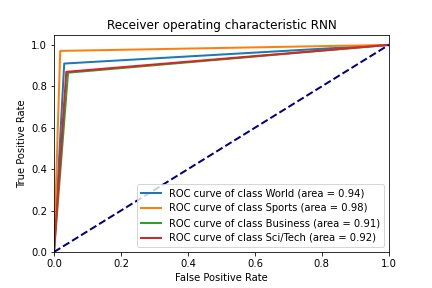
We divide the dataset into a training and validation set where training data has 120,000 samples and test data has 7,600 samples. We use accuracy metrics to compute performance of machine learning algorithms. It is calculated using

We use the above 3 discussed algorithms to predict the news category given input as textual data. In addition, we use multiple techniques to convert the preprocessed text data into numeric vector form, and compare all of them. We construct the models in the following ways - **i) Multinomial Naive Bayes** **(MNB)** - we use hyperparameter grid search with 5 fold cross validation to fine tune the model. The best alpha value for MNB comes out to be 1, where alpha is a smoothing parameter. **ii) Vanilla Neural Network (VNN) -** we use 2 hidden layers (512, and 64 nodes respectively) with Rectified Linear Unit as the activation function in all of them and the output layer (4 nodes) has Softmax activation. In addition, we use L2 regularization at every layer to penalize the weights and biases, which avoids overfitting. ,where is the hyperparameter that controls the regularization effect. To add more regularization, we use a dropout of 20% after each hidden layer. Adam optimizer is used to compute the gradients and update the weights and biases of the NN. Also, we use an **exponential learning rate decay (**decreases the learning rate after a number of steps**)**, with **decay rate 0.9**, **decay steps** as **10000** and a **learning rate** of **0.001**. We train this model for 5 epochs. **iii) Recurrent Neural Network (RNN) -** The first layer of this model is an embedding layer with an embedding size of 128. We then use one layer of simple RNN with 128 nodes followed by GlobalMaxPooling, after which we use a fully connected dense layer with 64 nodes, followed by 20% dropout. The output layers contain 4 nodes with Softmax activation. The optimizer for RNN is the same as the one we used in the VNN and we use a batch size of 256 for them. We train the above models on different training data and obtain the following results -

| **Model** | **Dataset** | **TF-IDF** | **Tokenizer** | **Embedding** |
| --- | --- | --- | --- | --- |
| MNB | Train | 90.72 | 30.12 | NA |
| VNN | Train | 97.49 | 25.80 | 95.68 |
| RNN | Train | NA | NA | 93.52 |
| MNB | Test | 89.72 | 30.23 | NA |
| VNN | Test | **91.04** | 25.67 | 90.28 |
| RNN | Test | NA | NA | 90.48 |

From the above table, we see that the **MNB** and the **VNN** performs worse on the Tokenizer data. This is because the Tokenizer converts words into a length 100 index vector which assigns integers to unique words. This makes it difficult for these algorithms to recognize the pattern, whereas while using **TF-IDF**, we assign score to each word and each word serves as a column, which makes it easier for the algorithms to recognize the independence between the input features and hence they learn better. Among the 3 models, **VNN** obtained the best test and train accuracy, **91.28%** and **97.49%**. We use the AUC - ROC (Area Under Curve for Receiver Operating Characteristic) curve to show the trade off between the True Positive Rate and the Specificity (1 - False Positive Rate). Higher the AUC, better a model does to predict that class. Confusion matrix is a type of visualization that measures the performance of a classifier based on the number of True Negative, True Positive, False Negative and False Positive counts for a model, where the diagonals contain True Positive values. We plot the AUC - ROC Curve and Confusion matrix for the model that fits the best in its category.



From the above plots we say that all the 3 models have similar performance, where the **RNN** performs the best for classifying **World** news (best roc for it), and **VNN** performs best for **Sci/Tech** news. This shows that even a simple model such as Multinomial Naive Bayes that is built upon the concept of conditional probability, can perform similar, or even better at times, than more advanced Neural Network architectures such as **RNNs**. But this does not happen for every dataset, as **MNB** performs the worst when built using the tokenizer data, but when we train a **VNN** on the same data it performs significantly well. Hence, not only the model, but also the data we feed to the model matters a lot when it comes to developing efficient and effective models.

**Conclusion/Future Work**

When a Vanilla Neural Network is used with multiple regularization methods, as we did in our **VNN** model where we added **L2 norm, Batch Normalization, and Dropout**, and is fitted on properly cleaned, preprocessed and transformed data, **TF-IDF** in our case, it can outperform more complex models like the **RNN.** Also, the **MNB** gives decent performance, given that it trains very quickly compared to NNs. We have not included Bidirectional RNNs, they have the ability to predict an output after inputting previous and next words both. The most advanced **RNNs** are built on the concept of **Long-Short-Term-Memory (LSTM).** LSTMs are the state-of-the-art NNs and are used for predicting patterns from sequential data. Its architecture consists of logical gates where it is decided whether to keep a certain part of data or not. The main reason why LSTMs would outperform RNNs is because RNNs suffer from the infamous problem of vanishing gradient, which is quite prevalent in nearly every NN architecture. LSTMs do not suffer from this problem and hence are considered to be the best for sequential data.

**References**

[1] I. Dilrukshi, K. De Zoysa and A. Caldera, "Twitter news classification using SVM," *2013 8th International Conference on Computer Science & Education*, 2013, pp. 287-291, doi: 10.1109/ICCSE.2013.6553926.

[2] Lai, Siwei, et al. "Recurrent convolutional neural networks for text classification." *Twenty-ninth AAAI conference on artificial intelligence*. 2015.

[3] Hughes, Mark, et al. "Medical text classification using convolutional neural networks." *Informatics for Health: Connected Citizen-Led Wellness and Population Health*. IOS Press, 2017. 246-250.

[4] <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>

[5] <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer>

Github repo :- <https://github.com/ProgressingMann/AGnews_classification>