Tower of Babel: Newspaper Classification

Allen Dufort (adufort1), Michelle Liu (mliu107), Jitpuwapat Mokkamakkul (jmokkama), Zuhal Saljooki (zsaljook)

Devpost: https://devpost.com/software/tower-of-babel
Poster: https://tinyurl.com/4r2dd3k6
Github: https://github.com/zuhal-saljooki/TowerofBabel

Introduction

Newspapers from many sources are often classified into categories – business, sports, politics, and so on. We were inspired by online news article recommendations and classifications and wanted to better understand how such a widely used feature could be modeled. In particular, we wanted to automate this classification process by implementing a program that classifies newspapers based solely on the text and article title of newspapers. There are many different classification models that can be applied to such a problem, so rather than choosing one implementation, we compared the performance of many models in order to find the best. As a bonus, we thought it would be helpful post-training to use this model to classify any long text into news article categories as well.

Related Work and Background

We drew our work mainly from one source which aligned closely to our goals while introducing several new classification models not covered in class [1]. This article is relevant to our project because it had a public implementation for classifying news articles into one of 5 categories (business, tech, politics, sport, entertainment), along with preprocessing. The goal of this article was to find the most accurate classification model.

Our other sources allowed us to better understand models such as Multilayer Perceptron [2], Natural Language Processing with Naive Bayes [3], K-means clustering [4], and general classification models [5] [6].

Methodology

Data

We utilized a dataset, BBC News Train.csv [7]. It includes news articles from the BBC. The dataset was obtained from Kaggle and contains 3 features:

1. ArticleId - An ID uniquely assigned to each record
2. Text - The article header and text contained within the article
3. Category - The category of the article (tech, business, sport, entertainment, politics)

The dataset is medium-sized (3.19 MB), containing 1490 different articles. However, each of the article texts are fairly long, as they contain the entire article.

ArticleId	Text	Category
1833	worldcom ex-boss launches defence…	business

Figure 1: Sample training example from the BBC News Train dataset

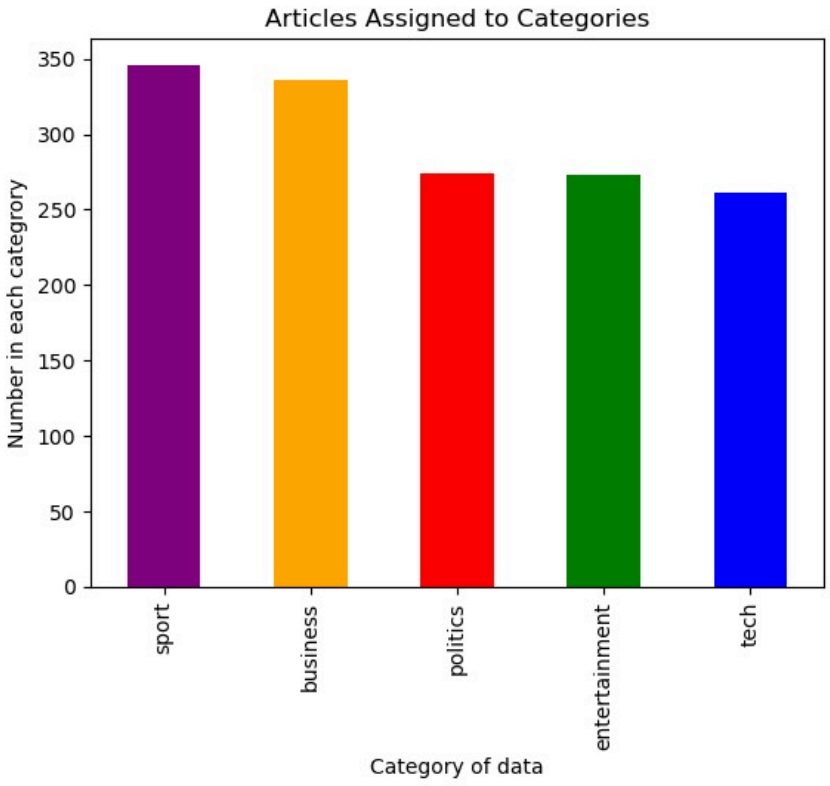

Figure 2: Visualization of data distribution

Preprocessing

Our data required significant preprocessing, as each entry was an unedited article of text containing unique capitalization, punctuation, and so on. Our preprocessing method involved multiple steps:

1. Factorizing by category
2. Removing special characters and stop words
3. Lowercase
4. Lemmatization
5. Count vectorizer
6. Train-test split

The first step in preprocessing was to factorize the dataset by the category, which associates category names (tech, business, sport, entertainment, politics) with a numerical value (0, 1, 2, 3, 4). Our next step in preprocessing was to remove special characters like (e.g. !, %, @, accents). We also removed stop words, which are commonly used words like “the.” To ensure uniformity in capitalization, we converted all words to lowercase. Finally, we lemmatized the words, which just groups words with similar meanings to one word (e.g. ‘rocks’ and ‘rock’ as one word).

To convert the words into vectors, we apply a count vectorizer from the scikit-learn library. CountVectorizer obtains a matrix of token counts from the collection of text documents.

To prepare the preprocessed data for training, we shuffled the data and applied a 70-30 train-test split, that is, 30% of a dataset was allocated for testing.

Model Architectures

We implemented many different classification models:

1. Logistic Regression
    a. Description: Logistic regression is a regression algorithm that tries to fit a logistic curve to the data.
    b. Our model: scikit-learn’s LogisticRegression
    c. Model parameters:
        i. None (utilized the default scikit-learn model)
2. Decision Tree Classifier
    a. Description: Decision tree generates a set of rules from the model, splitting the data at each branch. Each leaf represents a label.
    b. Our model: scikit-learn’s DecisionTreeClassifier

    c. Model parameters:
        i. None (utilized the default scikit-learn model)
3. Random Forest
    a. Description: Random forest is an ensemble-learning decision tree model that consists of generating multiple decision trees and finding the most common prediction from all the generated trees.
    b. Our model: scikit-learn's RandomForestClassifier
    c. Model parameters:
        i. Number of estimators = 100
        ii. Criterion = Entropy
        iii. Random state = 0
4. Support Vector Classifier
    a. Description: Support vector machines find hyperplanes that classify data points.
    b. Our model: scikit-learn's SVC
    c. Model parameters:
        i. None (utilized the default scikit-learn model)
5. K Nearest Neighbors (k=3 and k=7)
    a. Description: K nearest neighbors finds boundaries to classify the articles based on which labels the majority of the "neighbors" (nearby points) are classified as.
    b. Our model: scikit-learn's KNeighborsClassifier
    c. Model parameters:
        i. Number of neighbors (k) = 3, 7 (used both 3 and 7)
        ii. Metric = Minkowski
        iii. Power parameter (for Minkowski metric) = 4
6. Gaussian Naive Bayes
    a. Description: Gaussian Naive Bayes applies Bayes' theorem to the data to predict the label by assuming independence. There is an additional assumption that the likelihood of features is Gaussian.
    b. Our model: scikit-learn's GaussianNB,
    c. Model parameters:
        i. None (utilized the default scikit-learn model)
7. Multinomial Naive Bayes
    a. Description: Multinomial naive bayes predicts the tag of a text, such as an email or a newspaper story, using Bayes' theorem. Then, it calculates the likelihood

of each classification for a given sample and outputs the classification with the greatest likelihood.
    b. Our model: scikit-learn's MultinomialNB
    c. Model parameters:
        i. Alpha = 1.0
        ii. Fit prior = True
8. Multilayer Perceptron Classifier (scikit-learn)
    a. Description: MLP is a fully connected artificial neural network (ANN) that is essentially a deep neural network with one hidden layer.
    b. Our model: scikit-learn's MLPClassifier
    c. Model parameters:
        i. Solver = Adam
        ii. Alpha = 1e-5
        iii. Hidden layer sizes= (6,)
        iv. Random state = 1
9. Sequential Neural Network (TensorFlow)
    a. Description: TensorFlow and Keras allow for the construction of multilayer perceptrons with more details than scikit-learn is able to provide. Each layer can be specified with many parameters.
    b. Our model: We implemented a neural network consisting of several dense layers, based on models we have encountered in class.
        i. Layers
            1. Dense(12, activation= 'relu')
            2. Dense(6, activation= 'relu')
            3. Dense(1, activation= 'sigmoid')
    c. Model parameters:
        i. Loss = Binary Cross Entropy
        ii. Optimizer = Adam
        iii. Batch size = 1000
        iv. Epochs = 25

Metrics

We defined success as having a classification accuracy of over 60% for each model, following a similar threshold in Homework 3. To actually compare models, we utilized 4 metrics: accuracy, precision, recall, and F1 score, focusing primarily on accuracy and F1. Accuracy is defined as the percentage of correctly predicted observations to the total observations.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations, while recall is the ratio of correctly predicted positive observations to all of the actual positive observations. By combining precision and recall, we can get a commonly used accuracy metric known as F1 Score. The equations are listed below, where TP = true positive, TN = true negative, FP = false positive, and TN = false negative.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 Score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

Results

All models surpassed our target accuracy of 60%, and the precision, recall, and F1 scores for all the models were very similar – which isn't abnormal. In fact, all our models performed incredibly well, with our worst model still achieving 76.06% test accuracy.

Our best model was Random Forest, which achieved 97.99% test accuracy and a 0.98 F1 score, followed by Multilayer Perceptron, achieving 97.76% test accuracy and a 0.98 F1 score. The Sequential Deep Neural Network we implemented from TensorFlow also performed really well, at 97.54% test accuracy and a 0.98 F1 score. Our worst model was Gaussian Naive Bayes, at 76.06% test accuracy and a 0.76 F1 score.

As an additional bonus, once trained, our model allows us to classify any text data into the newspaper categories. However, this didn't perform as well as expected, possibly because our models overfit or trained on too small of a dataset – BBC News specifically, rather than all news. For instance, we inputted an article about tesla coils and only some models corrected classified this as technology.

	Accuracy Metrics			
Model	Test Accuracy	Precision	Recall	F1
Logistic	97.09%	0.97	0.97	0.97

Regression				
Decision Tree Classifier	82.10%	0.82	0.82	0.82
Random Forest	97.99%	0.98	0.98	0.98
Support Vector Classifier	96.64%	0.97	0.97	0.97
K Nearest Neighbors (k=3)	79.64%	0.80	0.80	0.80
K Nearest Neighbors (k=7)	77.63%	0.78	0.78	0.78
Gaussian Naive Bayes	76.06%	0.76	0.76	0.76
Multinomial Naive Bayes	97.09%	0.97	0.97	0.97
Multilayer Perceptron (scikit-learn)	97.76%	0.98	0.98	0.98
Sequential Neural Network (TensorFlow)	97.54%	0.98	0.98	0.98

Figure 3: Accuracy metrics on different models

Challenges

Our greatest challenge was deciding on a topic. At the start, we had so many ideas about what we wanted to implement. For instance, we discussed everything from creating a Discord chat bot, to doing some sort of machine translation, to newspaper recommendations. It took a long time to finally decide on a topic where we had concrete action steps.

Later on, when we decided to focus on news articles, we struggled to find and choose algorithms to accomplish our task, since there were a lot of different models that we were unfamiliar with. Our research led us to everything from Naive Bayes to Decision Trees. Since our project's stretch goal revolved around comparing the accuracies of different models, we

needed a way to evaluate which models can work with our task of classifying news articles, which led us to F1 scores.

Furthermore, we ran into some issues with Github that resulted in lost code. Our TensorFlow implementation of a simple neural network was deleted due to pull and push issues.

Reflection

How do you feel your project ultimately turned out? How did you do relative to your base/target/stretch goals?

- Despite the challenges we ran into, this was a really insightful project that taught us a lot about text classification and the variety of models that existed. We actually exceeded our base goals and reached our stretch goal, implementing far more than one model! We are really happy with how our project turned out.

Did your model work out the way you expected it to?

- Overall, our models performed better than expected. We were surprised that Random Forest performed so well, beating even our neural network, which we expected to perform best. That being said, they both had really good and similar accuracies. However, Random Forest is an ensemble decision tree model, so we expect it to be more powerful than Decision Tree. Perhaps decision trees work very well on a dataset containing lots of words. We were also surprised that K Nearest Neighbors performed so poorly since it worked relatively well for CIFAR and MNIST in class. Notably, KNN also took very long to train, more so than all the other models. It seems KNN and Gaussian Naive Bayes models do not work well for text classification. It's especially interesting how much worse Gaussian Naive Bayes did compared with Multinomial Naive Bayes.
- Most of our models did not generalize as well for news articles from different news groups. Neither did they perform that well for randomly fed text, such as "I like to play kickball," which was not classified as sports news for a lot of models. It's possible our model is overfit or trained on too specific of a dataset – after all, not all news groups utilize only 5 categories and BBC News may have specific writing styles.

How did your approach change over time? What kind of pivots did you make, if any? Would you have done differently if you could do your project over again?

- Initially, we planned on implementing just one model for text classification and reporting on it. However, we realized that would not give us much insight into text classification itself. This led us to pivot our stretch goals to include multiple models, to create a much more comprehensive project that also pushed us to consider different models.
- If we could do our project over again, we would have decided on a specific project with action steps a lot sooner rather than juggle many different ideas. It would have allowed us to delve further into this implementation.

What do you think you can further improve on if you had more time?

- If we had more time, we would have loved to explore some other models from class, such as RNNs. It would be a very fun challenge to try and apply RNNs to our dataset, and we're very curious how it would have performed against our other models.

What are your biggest takeaways from this project/what did you learn

- Overall, we learned a lot from this project! Coming in, we did not know anything about Naive Bayes, Decision Trees, or many of the other models utilized. We also learned about the scikit-learn library and how to use its many tools (e.g. train test split with shuffle). This project not only taught us about these models but also taught us how to apply machine learning and deep learning to general datasets!

References

[1] Text Classification of News Articles

[2] MLP model

[3] NLP with Naive Bayes

[4] K-means clustering

[5] Classification Model

[6] Intro to Text Classification Model

[7] BBC News Train.csv