FAKE NEWS & REAL NEWS

STA 9665, Fall 2020

Team 8
Danny Huang, Nicole Ly, Nishtha Ram, Qinyi Li, Tanay Mukherjee, Trang Le, Xinyi Li

TABLE OF CONTENTS

Introduction & Background

For our final project, we wanted to analyze a dataset that focuses on a topic that is relevant, applicable and relatable. Having said that, 'Fake News' is very prevalent throughout our generation and is a coined term used very often in our society. The amount of untruthful information, rumors and unreliable news that is out there is growing more and more everyday. It is also becoming extremely difficult to distinguish fake news from real news, as imposters get smarter and technology advances, making it very easy to create websites to mimic legitimate news sources.

Fake news articles also have a greater influence and reach, more than ever now, as news can be shared across the world in seconds, exponentially increasing the reach to consumers, regardless of the legitimacy of the news. This means that the impact this has on our society is now more significant than ever as a majority of the time these articles are created with a malicious intent and targeted to a specific audience in order to influence someone's belief, decision making or stir up negative feelings. We saw the fake news and real news dataset as a good opportunity to help us explore and dive deeper into this topic.

Motivation

As misinformation, rumors or fake news are becoming abundant, we are motivated by the clash between real news and fake news. How can we minimize the damage and public insecurity caused by its explosive growth?

Our project is structured around the 2 main research questions:

1. Can we automatically classify a news article as real or fake news?
2. Can we quickly identify the topic of news through the topwords?

Text Dataset Description

We obtained our dataset from Kaggle[1]. There are two separate CSV files, one named fake news and one named real news. Each file has four columns, labeled as follows: article title, article text, article subject, and publication date.

In order to have a better baseline understanding of the data set, we used different methods to locate general patterns that would later help us with our analysis. With named entity recognition, we found the most frequent entities (such as people and places) mentioned in both real and fake news. This included items such as "Trump", "Mueller", and "War." With N-gram analysis, we found unigrams, bigrams and trigrams to help build models for our LDA topic modeling analysis. The top salient results were "Trump", "Donald Trump", and "President Donald Trump."

[1] https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset

Text Preprocessing Description

Text preprocessing is the first and an important step for NLP tasks. It transforms text into a more digestible form so that machine learning algorithms can perform better. In this project, we will use 3 methods to pre-process our raw datasets. The first one is regular expression. Regular expression can give us a more powerful and flexible method for describing the character patterns we are interested in. Second is tokenization. Tokenization is the task of cutting a string into identifiable linguistic units that constitute a piece of language data. We will also use normalization. Normalization is a process that converts a list of words to a more uniform sequence. By transforming the words to a standard format, other operations are able to work with the data and will not have to deal with issues that might compromise the process.

Before we could begin our analysis, we needed to properly format our dataset. First we joined the “real” and “fake” news datasets. Fake news received the tag “0” and real news was tagged with “1”. We then converted the dates of the publications to a date time format. Next, the data was parsed to exclude the publisher data, i.e. we only wanted the text pertaining to the body of the articles. The data was cleaned to set text to lowercase, remove line breaks and to remove links. And then we finally removed stop words.

Methods & Model Evaluation

Part 1: Topic Modeling

In machine learning and natural language processing, a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. Topic modeling is an unsupervised learning method of representing a corpus as a set of topics (a distribution over a set of topics) and it is a frequently used text-mining tool for discovery of hidden semantic structures in a text body.

The "topics" produced by topic modeling techniques are clusters of similar words. A topic model captures this intuition in a mathematical framework, which allows examining a set of documents and discovering, based on the statistics of the words in each document, what the topics might be and what each document's balance of topics is.

For our project, the reasons we chose topic modeling method are as follows:

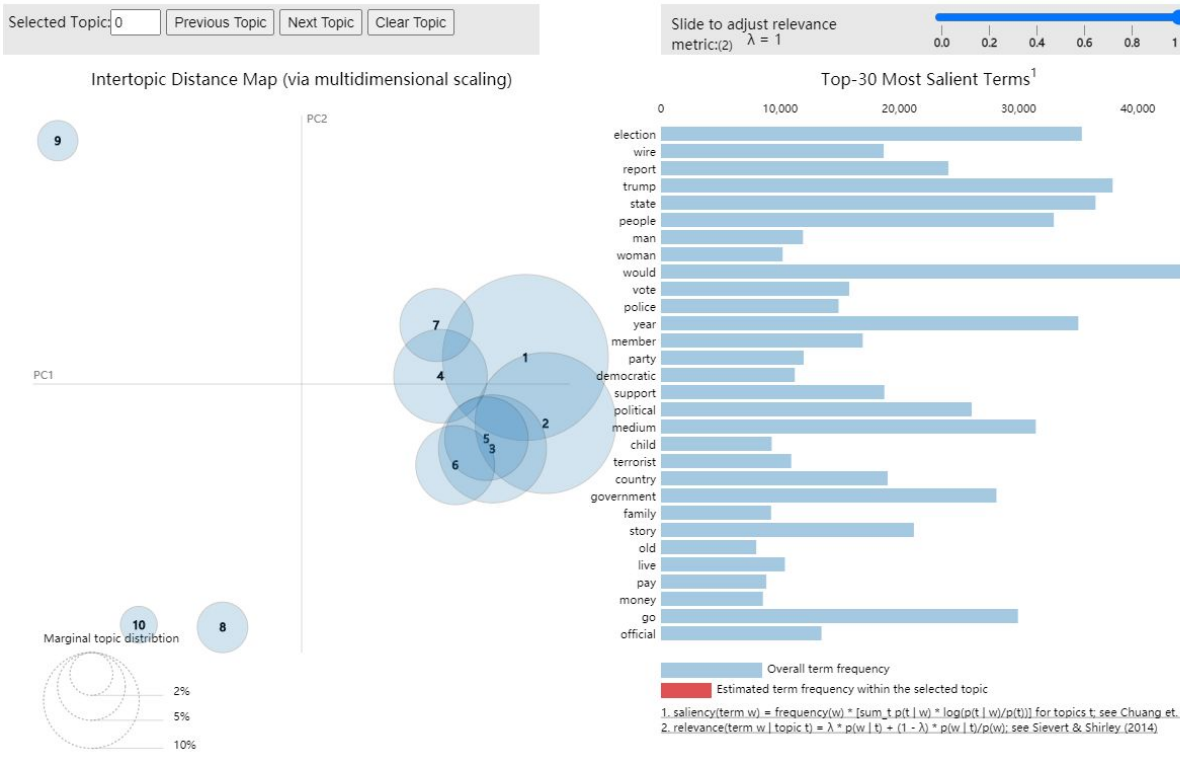
1. To have a general understanding of the news dataset, i.e. how many topics are covered in the data set, what each topic is, and the distribution of topics.
2. Given a piece of news, after classifying it, we can roughly judge what topic it is without reading it word by word.

We utilized the Latent Dirichlet Allocation (LDA) (Blei, Ng and Jordan 2003) model to identify latent topics for fake and real news and assign topic probabilities to these news. The basic idea behind LDA is that documents are represented as random mixtures over latent topics where each topic is characterized by a distribution over words (Blei, Ng and Jordan 2003).

Before applying LDA, we first imported the packages that we need for topic modeling. The core packages are ***gensim***, ***spacy*** and ***pyLDAvis***. Then we cleaned our documents to get a tokenized, stopwords-removed, and lemmatized list of words from a single document. Then we used ***gensim***'s Phrases model to build and implement the n-grams and make all these terms to the dictionary. To generate a LDA model, we would also need to understand how frequently each term occurred within each document by constructing a document-term matrix, which is the corpus for the LDA model.

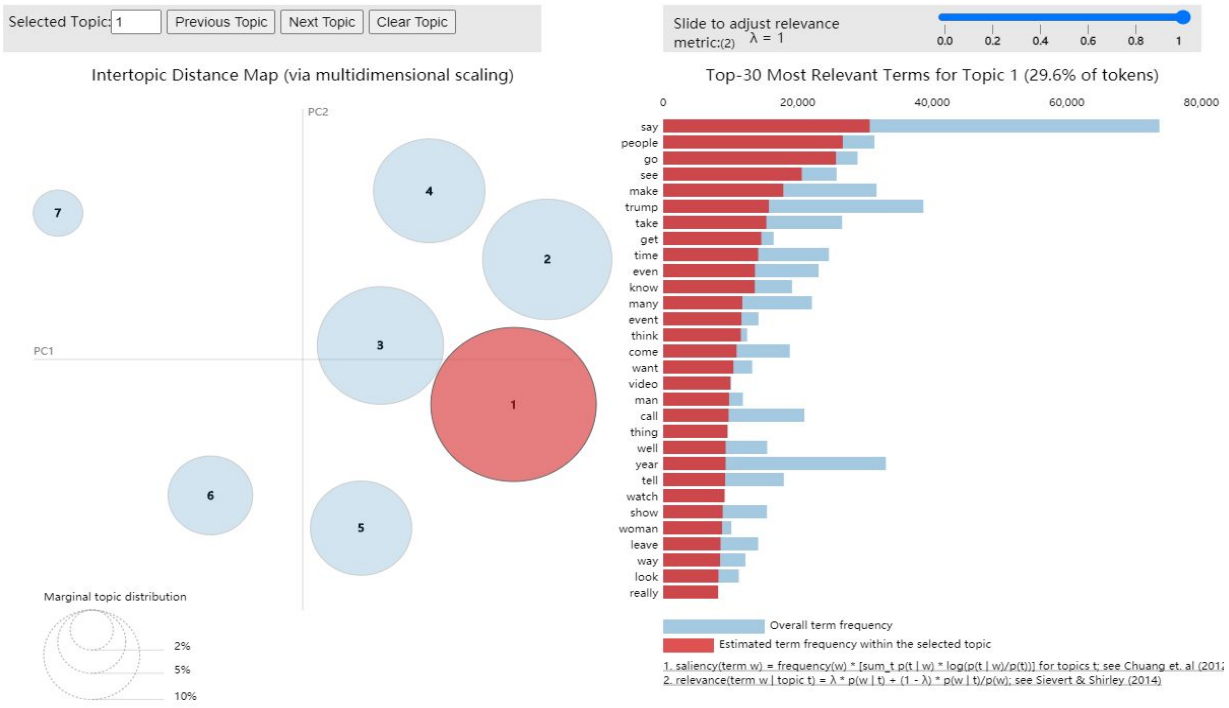
After we had everything required to train the LDA model, we first trained the model with 10 topics, The results can be seen in **Figure 3.1**.

Figure 3.1 *Results of 10 topics LDA model*



We could see seven topics overlapping with each other so we reduced the number of topics and tried 5, 6, 7, 8 topics respectively. After analyzing the data, we selected 7 topics which struck a balance between "having too many topics to be interpretable" and "too few to allow meaningful variation."

Figure 3.2 *Results of 7 topics LDA model*



We can see from the interactive chart above that each bubble on the left side represented a topic. The bubbles were non-overlapping and scattered throughout the chart. The larger the bubble, the more prevalent the topic. When we clicked on one of the bubbles, we would see the words bar on the right side displaying the top 20 words that formed the selected topic. We can adjust the lambda value on the top right side. Values of lambda that are very close to zero will show terms that are more specific for a chosen topic. Meaning that you will see terms that are "important" for that specific topic but not necessarily "important" for the whole documentation. When values of lambda that are very close to one will show those terms that have the highest ratio between frequency of the terms for that specific topic and the overall frequency of the terms from the whole documentation. When lambda = 1, the red bars represent the counts of the terms in a given topic, and the gray bars represent the counts of the same terms overall in the documentation. For example, in this case, the words listed here are the terms that have the highest ratio between frequency of the terms for topic 1 and the overall frequency of the terms from the documentation.

From the chart above, we knew that each topic in the LDA model was a combination of keywords and each keyword contributed a certain weight to the topic.

Figure 3.3 *Topic Composition for LDA model with 7 topics*

```
[(0,
  '0.034*"election" + 0.023*"political" + 0.021*"trump" + 0.015*"campaign" + '
  '0.015*"vote" + 0.014*"presidential" + 0.014*"candidate" + 0.013*"state" + '
  '0.013*"support" + 0.012*"party"'),
 (1,
  '0.034*"bill" + 0.024*"passage" + 0.024*"would" + 0.022*"program" + '
  '0.018*"pass" + 0.018*"parent" + 0.018*"writer" + 0.015*"legislation" + '
  '0.013*"health" + 0.013*"medical"'),
 (2,
  '0.017*"company" + 0.016*"year" + 0.014*"money" + 0.013*"pay" + '
  '0.011*"government" + 0.009*"business" + 0.009*"dollar" + 0.008*"financial" '
  '+ 0.008*"work" + 0.008*"fund"'),
 (3,
  '0.016*"police" + 0.011*"state" + 0.009*"year" + 0.009*"order" + '
  '0.009*"officer" + 0.009*"law" + 0.009*"case" + 0.008*"student" + '
  '0.008*"group" + 0.008*"federal"'),
 (4,
  '0.015*"say" + 0.013*"people" + 0.013*"go" + 0.010*"see" + 0.009*"make" + '
  '0.008*"trump" + 0.007*"take" + 0.007*"get" + 0.007*"time" + 0.007*"even"'),
 (5,
  '0.013*"government" + 0.013*"say" + 0.012*"country" + 0.012*"military" + '
  '0.010*"war" + 0.010*"would" + 0.009*"terrorist" + 0.008*"policy" + '
  '0.008*"world" + 0.007*"syrian"'),
 (6,
  '0.016*"wire" + 0.015*"medium" + 0.015*"report" + 0.014*"say" + 0.013*"news" '
  '+ 0.012*"story" + 0.008*"also" + 0.007*"claim" + 0.007*"russian" + '
  '0.007*"source"')]
```

Topic 0 was a combination of (0.034*election) + (0.023*political) + (0.021*trump) + (0.015*campaign) + (0.015*vote) + (0.014*presidential) + (0.014*candidate) + (0.013*state) + (0.013*support) + (0.012*party). From the top 10 words, we could indicate that topic 0 was related to the presidential election. Going through the remaining topics' keywords, we inferred the content of remaining topics as follows:

Table 3.4 *7 Inferred Topics*

Topic 0	Presidential Election
Topic 1	Bill and Legislation
Topic 2	Economy
Topic 3	Police
Topic 4	People Action
Topic 5	War and Military
Topic 6	News Media

Perplexity and topic coherence provided us the methods to evaluate the topic model. The higher the coherence score, the better the model. The coherence score for the 10 topics model was around 0.39

but the coherence score for the 7 topics model was about 0.45, which also demonstrated that the 7 topics model performed better.

After building the topic model, we could determine the topic of a given news document. We chose the topic number with the highest percentage contribution. The table below was the dominant topic for some documents. For example, the most likely topic of the news with number 0 was Bill and Legislation.

Table 3.5 *Dominant Topic and Contribution Percentage*

	News_No	Dominant_Topic	Perc_Contrib	Topwords	Text
0	0	1.0	0.2692	bill, passage, would, program, pass, parent, w...	WASHINGTON (Reuters) - The head of a conservat...
1	1	3.0	0.5700	police, state, year, order, officer, law, case...	WASHINGTON (Reuters) - Transgender people will...
2	2	6.0	0.5280	wire, medium, report, say, news, story, also, ...	WASHINGTON (Reuters) - The special counsel inv...
3	3	6.0	0.6229	wire, medium, report, say, news, story, also, ...	WASHINGTON (Reuters) - Trump campaign adviser ...
4	4	2.0	0.4782	company, year, money, pay, government, busines...	SEATTLE/WASHINGTON (Reuters) - President Donal...
5	5	1.0	0.2504	bill, passage, would, program, pass, parent, w...	WEST PALM BEACH, Fla./WASHINGTON (Reuters) - T...
6	6	4.0	0.3514	say, people, go, see, make, trump, take, get, ...	WEST PALM BEACH, Fla (Reuters) - President Don...
7	7	4.0	0.3713	say, people, go, see, make, trump, take, get, ...	The following statements were posted to the ve...
8	8	4.0	0.3419	say, people, go, see, make, trump, take, get, ...	The following statements were posted to the ve...
9	9	6.0	0.3454	wire, medium, report, say, news, story, also, ...	WASHINGTON (Reuters) - Alabama Secretary of St...

Part 2: Text Classification

Before we proceed with classification, we need to clarify what features are considered in the prediction of classes, either fake or real news. Since the news articles are text based documents, the most common features can be extracted from text documents are words used and their individual frequency or occurrence. Instead of getting a list of words found on all documents and converting frequency distribution of words in each document with our own python implementation, we use ***CountVectorizer*** under ***feature_extraction.text*** module from ***sklearn*** library to convert each document into an array containing count numbers for each word. Below is a visualization of the count-vectorization method.

Figure 4.1 *Visualization of the count-vectorization method*

Example: doc=["One Cent, Two Cents, Old Cent, New Cent: All About Money"]

	about	all	cent	cents	money	new	old	one	two
doc	1	1	3	1	1	1	1	1	1

In theory (a)

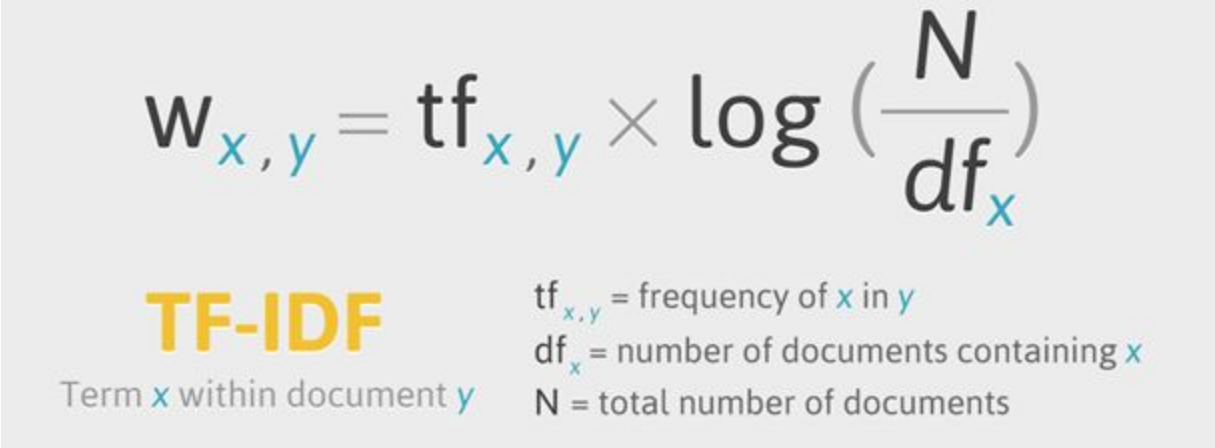
Index	0	1	2	3	4	5	6	7	8
doc	1	1	3	1	1	1	1	1	1

In practice (b)

However, not all words are meaningful and not all words are equally important in classification. We can either remove the stock words from the document before the implementation of count-vectorization or come up with a method to lower the importance of words that appear everywhere. By lower the importance, we mean to reduce the magnitude of the values for features such as stop words. Thus, we need to calibrate the results of a count-vectorization.

One common method used in weighing down the frequent words appear in all documents is Term-Frequency-Inverse-Document-Frequency (TF-IDF) transformation, which considers the frequency of a word in each documents and the frequency of the word in all documents in a way such that, if the word appears in almost all documents, its feature value will be small. In this way, we focus on the medium frequency words in all documents and get rid of the stop words and punctuation as well without manually filtering out them. The implementation is the name of ***TfidfTransformer*** under ***feature_extraction.text*** module in ***sklearn*** library.

Figure 4.2 *TF-IDF Transformation*[2]



With the implementation of feature extractions, we obtained 121,981 features. This is more than the number of observations in the training set which contains 44,888 documents.

Once the features were ready, having done the data pre-processing and the required steps before feeding in the data for modeling, we decided to conduct the classification exercise using various models. Following are the models we decided to experiment with and based on the model which gives us the best accuracy we will hyper-parameterize it.

1. **Logistic Regression:** This is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.
   In a binary logistic regression model, the dependent variable has two levels (categorical).
2. **Decision Tree Classifier:** It is a simple representation for classifying examples. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter. For

[2] The figure is taken from
http://filotechnologia.blogspot.com/2014/01/a-simple-java-class-for-tfidf-scoring.html

classification, a tree is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches.

3. **Random Forest Classifier:** Decision trees we learnt above is the base for this statistical model. It consists of a large number of individual decision trees that operate as an ensemble. The principle behind this model is - "*a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models*".
4. **Naive Bayes Classifier:** A Naive Bayes classifier is a probabilistic machine learning model that's used for classification tasks. The crux of the classifier is based on the Bayes theorem.

Figure 5.1

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption here is that the predictors/features are mutually independent, i.e., presence of one particular feature does not affect the other.

5. **KNN:** K-nearest neighbors is a supervised machine learning algorithm that can be used to solve both classification and regression tasks. In this statistical model the value of a data point is determined by the data points around it. KNN classifier determines the class of a data point by majority voting principle. If k is set to 3, the classes of 3 closest points are checked. Prediction is done according to the majority class. Similarly, KNN regression takes the mean value of 3 closest points.

In our classification metrics, accuracy refers to the percentage of correct predictions made by a model. Recall is the percentage of actual fake news that are classified as fake news. Precision is the percentage of the classified faked news that are actually fake news. F-measure is a combination of recall and precision .

The table below shows the performance scores for each of the 5 models. The numbers in BLUE are the best scores of all 5 models (excluding the regularized logistics). As we can see, logistic regression performs the best in terms of accuracy, recall, and F-measure. While random forest is best for precision. KNN performs the worst as expected since we pick 3 nearest neighbors as our model and it is very flexible and it is overfitting our training data.

Table 5.2 *Comparison of Model Performance*

Metrics	Logistic regression	Decision Tree Classifier	Random Forest Classifier	Naive Bayes	KNN (k = 3)	Regularized Logistic Regression
Accuracy	98.66%	95.79%	98.60%	93.92%	69.12%	99.29%
Recall	98.90%	96.58%	98.65%	95.31%	98.22%	99.45%
Precision	98.57%	95.49%	98.69%	93.3%	63.36%	99.20%
F-measure	98.73%	96.03%	98.67%	94.29%	77.03%	99.32%

Though logistic regression performs pretty well in the test set, it does not mean it is a good model to use in this case. Since we have about 121,981 features and only 44,888 observations, the weights of our logistic regression is not unique and not stable, thus leading to overfitting.

For example, if we have an equation such as y = w1x1 + w2x2 with known values for y, x1, and x2. We can have many solutions (in fact infinite number of solutions) for w1 and w2. The same thing happens to our logistic regression model when we have a larger number of features than number of observations.

Therefore, we need to **regularize our logistic regression model**. But how do we do this? Let's take a look at the expression below, which is the objection function for Lasso Regression. The part on the left side of it is the objective function for basic linear regression model, and the right side is the total penalty for the Lasso Regression.

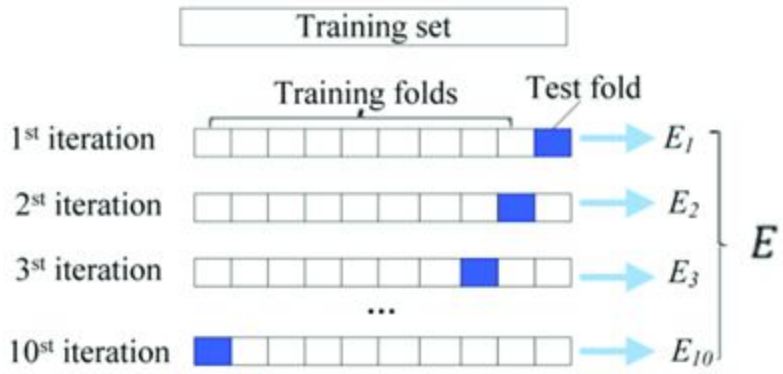
Figure 5.3

$$\sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} |w_j|$$

In general, we will add a penalty term (lambda), λ, to all the weights (the w_i's) of features in our objective function and pick the set of weights such that the objective function is minimized.

Since the penalty value is rather arbitrary, we would like to use Cross Validation (CV), which is a way to estimate the test error.

Figure 5.4



Looking at ***Figure 5.4***, in CV, we first split the training set into 10 folds. And we pick a sequence of 50 lambdas values, generally from 0.1 to 100. For each lambda, in each fold, we fit a model with 90% of the fold data, test our model on the remaining 10%, and calculate the Error E_i. We repeat the same calculation for the remaining 9 folds. And finally calculate the CV error by the average of E_i. We have to do this for all 50 lambdas and pick the lambda with the minimum CV error. By doing so, we can proceed to fit a regularized logistic regression model with that optimal lambda.

As a result of regularization, we have an improvement on all performance metrics (all metrics are above 99%) shown in the last column of the ***Table 5.2*** and thus, our model is less likely to overfit.

Conclusion and Practical Implications

Logistic regression is one of the most basic and simplest methods of machine learning. By tuning the hyperparameters for logistic regression, we can improve the model in terms of reducing the problem of overfitting and increase the accuracy score to 99.26%. Our results suggest that text feature in our data set are effective for distinguishing between fake and real news, and that robust classifiers can be constructed that enable the discovery of fake news articles.

As information shapes how we view the world and influences our decision making, no one can deny the abundance of misinformation, rumors, or fake news. By building an effective classification model, we can filter out news articles that are not credible and as a result, relieve the negative impact of fake news. It is important that such solutions are identified as they will prove to be useful to both readers and tech companies involved in the issue.

References

1. How to build confusion matrix and use countvectorizer and tf idf in pipeline

   Retrieved from https://scikit-learn.org/stable/index.html

2. Understanding Countvectorizer

   Retrieved from https://kavita-ganesan.com/how-to-use-countvectorizer/

3. Topic modeling with Gensim (Python)

   Retrieved from https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/

4. API documentation - pyLDAvis 2.1.2 documentation

   Retrieved from https://pyldavis.readthedocs.io/en/latest/modules/API.html

5. Latent Dirichlet Allocation (David M. Blei, Andrew Y. Ng, Michael I. Jordan; 3(Jan):993-1022, 2003.)