STAD94 – Statistic Project (Supervised Study)

Fake News Detection: A Statistical Approach via Machine Learning and Data Mining

Nagarjun Ratnesh

1002280195

Contents

Introduction

Purpose: To see if it is possible to classify a given article as fake or real article based on the count of POS (Part of Speech) tagging of the each individual words in each article, by utilizing variety of machine learning techniques and algorithms specifically Naïve Bayes, Logistic Regression and Neural Network Algorithm for classification. In addition, also to compare the results of these algorithm with the already implemented Algorithm in Scikit-learn python library.

Significance: This project is significance because since the last two decades more and more people have been more reliant of digital media to retrieve their information, it is only growing, both the number of people relying on digital media for their daily news and the number of contents available on digital media. Hence, there is a huge need to accurately verify these contents. However, there are so many facts to take into consideration, starting from the source of the media such as the who produced the information, all the way to how the content is written – which is what will be analyzed in this paper.

Limitations:

- One of the key limitation to the sample is that the ratio of fake to real dataset might not represent the real world appropriately.
- The data is from 2016, and it's mainly on political related articles, hence, it maybe not be applicable to other datasets.

Preprocessing Details

The Dataset: The dataset contains a total of 3990 observations. (53.1% - fake news, 46.9 % - real news).

The Explanatory Variable:

Firstly, 35 features have been extracted based on the content of an article. Each article was tokenized by sentence, then by words and each word was identified given a Part-Of-Speech tag, then for each features the count of number of POS tagged words were assigned for each article.

Features	*Description*
CC	CC coordinating conjunction
CD	CD cardinal digit
DT	DT determiner
EX	EX existential there (like: "there is" ... think of it like "there exists")
FW	FW foreign word
IN	IN preposition/subordinating conjunction
JJ	JJ adjective 'big'
JJR	JJR adjective, comparative 'bigger'
JJS	JJS adjective, superlative 'biggest'
MD	MD modal could, will
NN	NN noun, singular 'desk'
NNP	NNP proper noun, singular 'Harrison'
NNPS	NNPS proper noun, plural 'Americans'
NNS	NNS noun plural 'desks'
PDT	PDT predeterminer 'all the kids'
POS	POS possessive ending parent's
PRP	PRP personal pronoun I, he, she
PRP2	PRP$ possessive pronoun my, his, hers
RB	RB adverb very, silently,
RBR	RBR adverb, comparative better
RBS	RBS adverb, superlative best
RP	RP particle give up
SYM	Symbol
TO	TO to go 'to' the store.
UH	UH interjection errrrrrrrm
VB	VB verb, base form take
VBD	VBD verb, past tense took
VBG	VBG verb, gerund/present participle taking
VBN	VBN verb, past participle taken
VBP	VBP verb, sing. present, non-3d take
VBZ	VBZ verb, 3rd person sing. present takes
WDT	WDT wh-determiner which
WP	WP wh-pronoun who, what
WP2	WP$ possessive wh-pronoun whose
WRB	WRB wh-abverb where, when

The Response variable:

The label is either 0 or 1 (binary): 0 implies the article is fake, and 1 implies that it trust worth.

Note*The csv file **01_preprocessed.csv** contains the preprocessed summary of the dataset. The first 35 columns being the predictors and the last on being the response.

Exploratory Data Analysis

One of the first thing to do with the preprocessed data is to standardize it such that different variables have can be compared in the same scale. The formula for standardizing the data is

$$z = (x_i - \mu)/\sigma$$

So for each column, calculate the mean(μ), the standard deviation(σ), then for each value, calculate the z-score to obtain new vectors of standardized variable values.

Even though standardizing the variables puts the data to scale, 35 variables is still quite a bit, specifically considering that there are many 0's in the dataset.

The Density and Sparsity:

For this collected Dataset, the Sparsity is 0.607683 there for the Density is 1-0.607683 = 0.392317. The sparsity is calculated by taking percentage of 0's found in the dataset for each row and then taking the average of the percentage (aka; the number of cells containing 0/total number of cells), and the Density is 1 – Sparsity. Density and Sparsity tells us how spread the data is. When we have more and more dimensions added, we should expect to see the Sparsity increase while Density decrease. This is because when the number of features increases, the volume of space between those data points increase as well. So then one would end up having huge amount of empty space with data points spread out that it is difficult to find a pattern. This leads to a well-known issue known as the curse of dimensionality.

A well know technique used for such task is the Principle Component Analysis, which is the next task of this project. However, since the sparsity is not too high, we might not be able to reduce the number of dimensions too much.

Principle Component Analysis

PCA is a feature Extraction technique. In a nutshell, PCA calculates a matrix that summarizes how our variables all relate to one another, then break this matrix down into two separate components the direction and magnitude, Eigen vectors and Eigen values respectively.

Implementation Details as Follows:

1. **Take only the features (basically take all vectors except the label) and compute the d-dimensional mean vector**
   First sperate the predictor variable and standardize them, we have already done that in the Exploratory section. This is simply taking the mean and standard deviation, then normalizing each vector.
2. **Compute the Variance Covariance Matrix**
   The matrix contains details as follows

$$\mathbf{V} = \begin{bmatrix} \frac{\sum x_1^2}{N} & \frac{\sum x_1^2 x_2^2}{N} & \cdots & \frac{\sum x_1^2 x_f^2}{N} \\ \frac{\sum x_2^2 x_1^2}{N} & \frac{\sum x_2^2}{N} & \cdots & \frac{\sum x_2^2 x_f^2}{N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\sum x_f^2 x_1^2}{N} & \frac{\sum x_f^2 x_2^2}{N} & \cdots & \frac{\sum x_f^2}{N} \end{bmatrix}$$

Where f is the n[th] feature column.

3. **Compute the corresponding eigenvectors and eigenvalues**
   With Eigen value and Eigen vectors, what we are essentially trying to find is a constant λ**,** such that it summaries the contents of Matrix A

$$A\boldsymbol{x} = \lambda\boldsymbol{x}$$

4. **Sorting the eigenvectors by decreasing eigenvalues and choosing k eigenvectors with the largest eigenvalues**
   This part is straight forward as all you will have to do is order the corresponding eigen vectors and value in descending order. See the table on the next page for the analysis of 01_preprocessed.csv dataset.

   There are multiple methods of choose this value for K. The one I decided to choose is “percentage of cumulative variance”. What this means is, each eigenvalue is looked at as a percent out of the whole dataset and try to capture as much variance as possible. From looking at the Cumulative Variance column, the first 30 components are needed to explain 99% of the variance in the dataset

5. **Transform the sample onto the new subspace**

Finally, as the last step, transform the original feature matrix into the new subspace using Eigen vectors such that (35x3990) * (3990x30)

As one can see from the chart, in order to explain 99% of the variance, at least 30 variables are needed and this is because (1) the sparsity of our data is not too enough, (2) There is not much multi collinearity from the Correlation matrix.

Eigen Values	Variance each component represents	Cumulative Variance
12.65310413	0.36142666	0.36142666
1.31471395	0.03755384	0.3989805
1.15616467	0.033025	0.4320055
1.10915475	0.03168219	0.46368769
1.08215645	0.03091101	0.4945987
1.06356127	0.03037985	0.52497855
1.03833454	0.02965927	0.55463782
1.02776947	0.02935748	0.5839953
1.01949333	0.02912108	0.61311638
0.9774324	0.02791964	0.64103602
0.95665605	0.02732618	0.6683622
0.93994584	0.02684886	0.69521106
0.91900183	0.02625061	0.72146167
0.88357826	0.02523877	0.74670044
0.84926868	0.02425874	0.77095918
0.813071	0.02322478	0.79418396
0.78673116	0.0224724	0.81665636
0.77993015	0.02227813	0.83893449
0.73969374	0.02112881	0.8600633
0.65041154	0.01857853	0.87864183
0.60639982	0.01732137	0.8959632
0.56759612	0.01621297	0.91217617
0.53849929	0.01538184	0.92755801
0.51541489	0.01472245	0.94228046
0.50216425	0.01434395	0.95662441
0.3704904	0.01058279	0.9672072
0.24157477	0.00690041	0.97410761
0.21167904	0.00604646	0.98015407
0.1687133	0.00481917	0.98497324
0.13720733	0.00391923	0.98889247
0.11885788	0.00339509	0.99228756
0.10351536	0.00295684	0.9952444
0.07936107	0.00226689	0.99751129
0.06058112	0.00173046	0.99924175
0.0265463	0.00075828	1.00000003

Challenges Faced and Solution Derived:

- One of the main challenges with PCA is having a good understanding on linear algebra.
- Also figuring out how to select the component also posed some challenge as reduction from 35 to 30 feature doesn't necessary help too much with organizing the data.

Algorithms using SciKit learn python library

Notes: Analysis using Naïve Bayes, Logistic Regression and Neural Network Algorithms from the SciKit learn library. For each algorithm, the follow is used

-For Naïve Bayes: *class* sklearn.naive_bayes.MultinomialNB ()
-For Logistic Regression: *class* sklearn.linear_model.LogisticRegression()
-For Neural Network: *class* sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(30, 30))
30, 30 indicates to use 30 hidden layers with 30 perceptron for each of the layer. Hence, the it takes time for the algorithm to learn the model. Table: This is the result before PCA

Test Size	**Naïve Bayes** Confusion Matrix	**Logistic Regression** Confusion Matrix	**Neural Network** Confusion Matrix	**Accuracy (%)** (Neural Network)	**Accuracy (%)** (Naïve Bayes)	**Accuracy (%)** (Logistic Regression)	**Total Articles**
0.1	$\begin{bmatrix}118 & 92\\ 22 & 167\end{bmatrix}$	$\begin{bmatrix}170 & 40\\ 63 & 126\end{bmatrix}$	$\begin{bmatrix}132 & 78\\ 24 & 165\end{bmatrix}$	74.4	76.69	71.42	399
0.25	$\begin{bmatrix}276 & 248\\ 53 & 421\end{bmatrix}$	$\begin{bmatrix}436 & 88\\ 152 & 322\end{bmatrix}$	$\begin{bmatrix}495 & 29\\ 312 & 162\end{bmatrix}$	65.8	75.95	69.84	998
0.5	$\begin{bmatrix}859 & 182\\ 152 & 322\end{bmatrix}$	$\begin{bmatrix}546 & 495\\ 108 & 846\end{bmatrix}$	$\begin{bmatrix}575 & 466\\ 334 & 620\end{bmatrix}$	59.9	69.77	59.2	1995
0.75	$\begin{bmatrix}746 & 845\\ 160 & 1242\end{bmatrix}$	$\begin{bmatrix}1311 & 280\\ 581 & 821\end{bmatrix}$	$\begin{bmatrix}700 & 891\\ 147 & 1255\end{bmatrix}$	65.3	71.23	66.42	2993
0.9	$\begin{bmatrix}1541 & 358\\ 951 & 741\end{bmatrix}$	$\begin{bmatrix}1120 & 779\\ 339 & 1353\end{bmatrix}$	$\begin{bmatrix}1851 & 48\\ 1527 & 165\end{bmatrix}$	56.1	68.86	63.55	3591

There are many other different parameters that can be altered for all 3 of these algorithms in the scikit-learn library.

After feature extraction through PCA, the accuracy for Neural Network and Logistic Regression increased much higher, with the cross-validation average of 73.5 and 81.9 respectively.

For each Algorithm, the following details will be collected similarly to how it is done for the Analysis done using SciKit learn library:

(i) False Positive Rate (FP): The number of misclassified fake news articles
(ii) False Negative Rate (FN): The number of misclassified real news articles
(iii) True Positive (TP): The number of real news articles correctly identified
(iv) True Negative (TN): The number of fake news articles that are correctly classified
(v) Accuracy: Percentage of correctly identified real and fake news articles
(vi) Recall: Percentage real news indicated as real news (*true positives / number of actual positives)*
(vii) Precision: Percentage of correctly labelled for real news articles (*true positives / number of predicted positives.)*

Model Construction

Not all algorithms work the same way, each algorithm have their pros and cons. For this project, I've decided to implement three of the wildly used supervised algorithms meaning that these algorithms will try to classify the data given some training sample data. The order of the implementation of the algorithm will follow as going from the simplest one to the most complicated one. Naïve Bayes -> Logistic Regression -> Neural Network.

The chart below shows the results of my algorithm for Naïve Bayes and Logistic Regression.

Test Size	**Naïve Bayes** Confusion Matrix	**Logistic Regression** Confusion Matrix	**Recall, Precision Logistic Regression**	**Recall, Precision Naïve Bayes**	**Accuracy (%)** (Naïve Bayes)	**Accuracy (%)** (Logistic Regression)	**Total Articles**
0.1	$\begin{bmatrix}191 & 132\\ 28 & 48\end{bmatrix}$	$\begin{bmatrix}141 & 190\\ 106 & 135\end{bmatrix}$	75.0, 63.3	26.66, 63.15	59.89	69.17	399
0.25	$\begin{bmatrix}464 & 346\\ 60 & 128\end{bmatrix}$	$\begin{bmatrix}334 & 106\\ 190 & 368\end{bmatrix}$	77.6, 65.95	27, 68,08	59.32	70.34	998
0.5	$\begin{bmatrix}954 & 662\\ 119 & 260\end{bmatrix}$	$\begin{bmatrix}670 & 403\\ 205 & 717\end{bmatrix}$	77.76, 64	28.19, 68.6	60.85	69.52	1995
0.75	$\begin{bmatrix}1315 & 298\\ 830 & 551\end{bmatrix}$	$\begin{bmatrix}1007 & 604\\ 311 & 1070\end{bmatrix}$	77.48, 63.91	39.89, 65.05	62.36	69.42	2993
0.9	$\begin{bmatrix}1698 & 1200\\ 218 & 475\end{bmatrix}$	$\begin{bmatrix}1248 & 668\\ 429 & 1246\end{bmatrix}$	74.38, 65.01	28.35, 68.54	60.51	69.45	3591

Looking at the results, it makes sense for Naïve bayes to consistently have more article classified as fake for in our sample, the percentage of fake articles are higher. And the proportion of classification in the dataset has a huge image for this algorithm specifically. More details to follow below.

Naïve Bayes

Implementation Details:

The features contain real values hence:

In the case of real-valued features, we can use the Gaussian distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D}\mathcal{N}(x_j|\mu_{jc},\sigma_{jc}^2)$, where μ_{jc} is the mean of feature j in objects of class c, and σ_{jc}^2 is its variance.

Figure 1: Taken from Machine Learning: A probabilistic Perspective

The formula I the diagram above is the core of the algorithm with one exception instead of using a normal distribution, the naïve bayes implemented will a simple uniform distribution.

To calculate the posterior probability:

$$P(real_{news}|features) = \frac{P(f_2|real_{news}) * P(f_2|real_{news}) * \dots * P(f_n|real_{news}) * P(real_{news})}{Evidence}$$

Similarly

$$P(fake_{news}|features) = \frac{P(f_1|fake_{news}) * P(f_2|fake_{news}) * \dots * P(f_n|fake_{news}) * P(fake_{news})}{Evidence}$$

Where **n** is the number of features.

Laplace Correction:

Will need to be used if the feature is to have a probability of 0, hence, I would need to replace that value you by a small number possible eg.,1.

I will be Gaussian Distribution: Which from mentioned above follows

$$P(feature|Y = real) = \frac{P(feature\ and\ real)}{P(Y = real)}$$

Similarly, for fake article,

$$P(feature|Y = fake) = \frac{P(feature\ and\ fake)}{P(Y = fake)}$$

Finally, the probability $P(real_{news}|features)$ and $P(fake_{news}|features)$ is to be compared and the largest of the label of the largest probability will be assigned to the specific article.

Similarly, for fake news class, the probability of each features given fake or real news class can be calculated using the normal distribution, since each feature will be normalized.

Challenges Faced and Solutions Derived:

- If a specific column does not have any values, then this will set the probability as 0, which in turn will set the whole probability to zero, this mainly happens when training set size is small, some features have no values. In order to over come this issue Laplace Correction must be done.
    - Laplace Correction (smoothing) is simply adding the value 1 to the numerator such the probability of the count does not equal to zero.
- Run time – larger the test data, the longer it takes.
- The model is highly reliant on the accurate representation by the observations of the real world. As mentioned in the limitation, there is no real verifiable method to figure out the correct representation of the ratio between real and non real articles.

Logistic Regression

Logistic regression is intended for binary classification problems. There for it should be a better model than Naïve Bayes.

Implementation Details:

Binary Classification Model

The probability of a specific observation x given information of two details can be calculated as follows.

$$p(x) = \frac{p(x|c1)}{p(c1)} + \frac{p(x|c2)}{p(c2)}$$

Re-arranging for c1 or c2, we get

$$p(c1) = \frac{p(x|c1)p(c1)}{p(x|c1)p(c1) + p(x|c2)p(c2)} = \frac{1}{1 + e^{a(x)}} \; where \; a(x) = \ln(\frac{p(x|c1)p(c1)}{p(x|c2)p(c2)})$$

$$g(a) = \boldsymbol{w}^T \boldsymbol{x} + b$$

Sigmoid function = $z(a) = \frac{1}{1+e^a}$

It is used to map predicted values to probabilities. Maps real values to values between 0 and 1.

The decision boundary can be changed on a case by case basis, in this case,

Decision boundary p >= 0.5 -> class = 1

P < 0.5 -> class = 0

Putting it all together, essentially, $P(c_i) = \frac{1}{1+e^{\boldsymbol{w}^T \boldsymbol{x}+b}}$

Information Theory

Cost function: Cross-Entropy aka Log Loss (INFORMATION THEORY: Entropy)

Cost function for each class: We are trying to minimize the cost function using gradient descent algorithm described below. The cross entropy tells us how close the predicted distribution to the true distribution is. It calculated as following, by computing the expectation of log(q) under p.

$$C(h(x), y) = \; -\log\big(h(x)\big) \; if \; y = 1$$

$$C(h(x), y) = \; -\log\big(1 - h(x)\big) \; if \; y = 0$$

Putting the cost functions together, we get

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log\big(h(x_i)\big) + (1 - y_i)\log(1 - h(x_i))]$$

In Vectorized form: $L(w) = [-y^T \log(h) - (1-y)^T \log(1-h)]$, which is used in the code, in the check_cross_entropy method.

Gradient Descent

gradient descent is a way for us to calculate the best set of values for the parameters of concern

In layman's terms, the steps are as follows:

With the given gradient, calculate the rate of change in coefficient with respect to the size of step taken. Once a new coefficient is found, calculate the new gradient for the parameter. The process will be repeated until a convergence is reached.

The Size of the step is Alpha (The learning rate): in order for Gradient Descent to work a reasonable value should be set to α (learning rate). This parameter determines how fast or slow we will move towards the optimal weights.

If the α is very large

- High value -> bigger steps, might miss minimum
- Too Low -> slow descent, too many iterations for it to converge

Therefore, choosing an appropriate alpha value is crucial in the algorithm.

Iteration: Optimal iteration values (aka best number of steps to take before it's pointless)

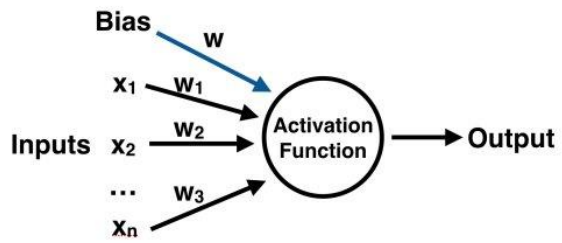
Challenges Faced and Solutions Derived:

- Time to find appropriate matrix methods for calculation. Had to draw out matrix.
- Understand the gradient descent and incorporating it into logistic regression (finding gradient)

Artificial Neural Network (Artificial)

How neural network works is that based on the initial input features, it will create multiple different classification, which are known as hidden layers. Each layer will have different types of classification based on the input values.

Perceptron: Below is a single perceptron which is simple a node that takes in multiple input and produces an output. Neural network is composed of many of this perceptron, each producing various output based on initial inputs.



Initially, the process starts with the **feed-forward** phase where the final value of a specific node is calculated by first multiplying the inputs with the corresponding weights, then the product is sent through an activation function such as the sigmoid function used in the logistic regression and eventually sent as output or as input for the next layer.

In the **back-propagation** phase, the predicted output is verified with the actual output to calculate the error so to see if the cost function can be minimized. This is done through gradient descent similarly to how it is done for logistic regression.

Validation

K-Fold Cross Validation:

K-Fold Cross validation is simply partitioning the data in to K different chunks, then leaving one chunk out as test dataset, and then training on the rest of the data then testing on the chunk that is left out. This process is done for every partition. Then the average of the accuracy is considered to be the accuracy of the model (Mean squared Error). K-Fold assists in reducing bias. However, as you may have noticed when running the Artificial Neural Network, K-Fold cross validation can be very time consuming.

Implementation Details:

Implementation of K-Fold Cross Validation is straight forward. It follows the LOOCV principle and it makes use of K-Fold SciKit learn library.

When running my_cross_validation.py

1. **It will ask for the K, the number of splits that you would like to split the dataset into.**
2. **It will ask for the type of algorithm you would like to use to run the cross validation on.**
    a. **For Naïve Bayes, no additional parameters are required**
    b. **For Logistic Regression:**
        i. **It will ask for alpha value: the learning rate**
        ii. **The number of iterations**

Resources

Machine Learning and Data Mining Lecture Notes (2018, Aaron Hertzmann and David J. Fleet)

Machine Learning a Probabilistic Perspective (2012, Kevin P. Murphy)

Fake News Detection on Social Media: A Data Mining Perspective (2017, Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Li)