CSC 520 – Artificial Intelligence

Assignment 5 Part 2

Project Report

Mangalnathan Vijayagopal – mvijaya2

Due Date: April 24th 2020

Introduction (Problem Statement)

A study by McAfee shows that over 97% of consumers were unable to correctly identify phishing scams when presented with them. As digital communication has evolved so has the sophistication of hackers and scammers. In addition to phony phone calls and email phishing scams, text scams have become more common with the proliferation of smartphones and the emergence of text messages as a part of everyday communication. However, many people could not correctly figure out whether a text is a spam text or a ham text. According to Wikipedia, spam: “the use of electronic messaging systems to send unsolicited bulk messages, especially advertising, indiscriminately.”, ham: “E-mail that is generally desired and isn't considered spam.”. So identifying when a text message is spam or ham will benefit people in their daily life. In this project, a machine-learning model has been built to classify text into spam or ham.

The steps for classification first begins with splitting the dataset into training and test data. We will use the training data to train the machine learning model. Then we use the test data to verify the actual labels and the predicted labels. Using these information, we can find the accuracy of the model i.e a measure of success of classification.

Dataset and Statistics

We are using the following data set collected from the UCI Machine Learning Repository.

Data: http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection

This dataset is a text file which contains 5574 messages along with labels (spam or ham) tagged by researchers. By analyzing the dataset, we see that there is one attribute, which is the text message and class label with only two possible values. The collection is composed of just one text file, where each line has the correct class followed by the raw message. An example line is as follows:

ham What you doing?how are you?

spam URGENT! Your Mobile No 07808726822 was awarded a L2,000 Bonus Caller Prize on 02/09/03! This is our 2nd attempt to contact YOU! Call 0871-872-9758 BOX95QU

Data Pre-processing and Feature Engineering:

Since the dataset is a text file where each line has correct class followed by the raw message, we must first separate them into text and class labels respectively first.

This is done by splitting every line at every ‘\t’ (tab), and then storing the split values in a pandas dataframe under columns “Text” and “Labels” respectively. The following is a subset of how data is stored in the dataframe:

Label	**Text**
ham	Rofl. Its true to its name
spam	URGENT! Your Mobile No 07808726822 was awarded a L2,000 Bonus Caller Prize on 02/09/03! This is our 2nd attempt to contact YOU! Call 0871-872-9758 BOX95QU

Once we read the data from the text file and store them into the dataframe as shown above, we apply text normalisation to every text in the dataset. Text normalisation is the process of transforming text into a universal form. There are many methods for text normalising. However, after analysing the dataset, case normalization is more than enough for cleaning the data. In this scenario, case normalization is converting all the text to lowercase.

After every text is processed, the dataframe has been split into training data and testing data in the ratio 70:30 respectively. In this project, scikit-learn machine learning package is used to perform splitting of the dataset.

The training data will be used to train models and the test data to test the accuracy of the trained model.

Training the model means to detect patterns between the attributes and the class labels.

Testing the models means to predict the class label for unknown attributes by using all the patterns derived by the model.

Then we check the predicted values and the actual values of the unknown attributes to determine the correctness of the predictions.

After the data has been split into training and testing data, the next step is to convert the Text into a numerical format because patterns in data are easily recognized if they are in numerical format.

To do this, we perform word embedding on the texts(not the labels) alone to convert the texts into a vector of real numbers (Numerical format).

We will be using the Term Frequency – Inverse Document Frequency (TF-IDF) word embedding technique. The TF-IDF Vectorizer will convert a text into a matrix of TF-IDF features. There are a lot of word embedding techniques(such as Latent Dirichlet Analysis, Kulback – Leibler Diverhence, Latent Semantic Analysis etc..) out there, and each one of them have their own advantages. TF – IDF was chosen because it is renowned for it's efficiency and ease in vectorizing text messages.

In TF-IDF method, the Term Frequency and Inverse Document Frequency are the components of calculated scores assigned to each word in the text message.

Term Frequency: How often a word appears in a text message

Inverse Document Frequency: How often the word appears across text messages

The Inverse Document Frequency downscales or reduces importance of words that appears very frequently across text messages.

The TF-IDF score value for a word X in text message (document) Y is calculated as follows:

TF-IDF(X, Y) = TF(X, Y) * IDF(X)

where,

TF (X, Y) = (Number of times word X is in text Y) / (Total number of words in Y)

$$\text{IDF(X)} = \log\left(\frac{Total\ number\ of\ text\ messages}{Number\ of\ text\ messages\ containing\ the\ word\ X}\right)$$

Here log() is natural log with base **e.**

TF-IDF scores highlight words that are interesting i.e frequent within a text message and not across text messages. In this way, we can easily capture unique patterns between each text message and the class labels.

This project uses TFIDVectorizer, which is a part of scikit-learn python package, which performs the entire TF-IDF process of the whole dataset. Therefore, we convert all the text messages (Not the labels) from both training and test data into numerical form using TF-IDF Vectorizer.

We do need to convert the class labels because we are only using them as information for mapping them along with the patterns generated during the training of the machine-learning model.

Models

Support Vector Machines

Support Vector Machines (SVM) are supervised machine learning models used for classification and regression analysis. It is reputed to be a fast and dependable classification algorithm that performs very well even with a limited amount of data.

The basic principles behind support vector machines is to develop an optimal hyperplane or decision boundary which classifies data.

SVM was a clear choice for the given dataset, because it works best and efficiently for two-class classification problems such as ham or spam.

In this case, the SVM model, mathematically derives the patterns from the dataset and generates an optimal hyperplane which can be used for classifying unknown data.

Logistic Regression

Logistic Regression is a statistical model that uses a logistic function to determine the probability of the dependent variable (class label) with respect to the features (text messages). The probabilities calculated determine the class label. Using the training data, we determine the cutoff for the probability values generated to predict class labels.

Logistic Regression is preferred for two-class problems because the probability values calculated by the logistic functions accurately represent the class label for the features.

Naïve Bayes

Naïve Bayes methods are a set of supervised learning algorithms based on applying Bayes theorem with the assumption of conditional independence between the features (attributes) given the class label. It makes use of the Bayes theorem to calculate the probabilistic relationship between the class label and the dependent feature vector (matrix as in our case).

Naïve Bayes classifiers have simplified assumptions, but have worked quite well in many real world scenario such as text classification and spam classification. They require only a small amount of training data to estimate the necessary parameters.

They are considered to be extremely fast compared to more sophisticated methods.

In this project, Multinomial Naïve Bayes classifier, Complement Naïve Bayes and Bernoulli Naïve Bayes have been implemented using the scikit-learn python package.

These models were specifically chosen because they are famously used for efficient text classification. Therefore they are the best solutions to the given problem statement.

Evaluation Methods

Evaluating the performance of the machine learning models developed is a crucial task. The reliability of the model is represented by its performance.

The performance is calculated using evaluation methods.

In this project, the following evaluation methods were used with the help of scikit-learn:

1) Confusion Matrix
2) Accuracy Score

These methods are used to compare the predicted values and the actual values of the class labels.

Confusion matrix is such that each entry C(i,j) is the number of observations known to be in group i and predicted to be in group j

For binary classification, the size of the matrix is 2X2s

For the problem statement the matrix is represented as follows:

$$\begin{bmatrix} \boldsymbol{C(ham, ham)} & \boldsymbol{C(ham, spam)} \\ \boldsymbol{C(spam, ham)} & \boldsymbol{C(spam, spam)} \end{bmatrix}$$

Using this matrix, a variety of scores such as accuracy, recall, precision and f1-score can be generated. For example, accuracy can be calculated as follows:

$$\text{Accuracy} = \frac{[\mathrm{C(ham,ham)} + \mathrm{C(spam,spam)}]}{[C(ha\quad,ham)+C(spam,spam)+C(ha\quad,spam)+C(spam,ham)]}$$

Similar to the accuracy calculation above, accuracy_score counts the number of instances where predicted and actual class labels match and returns the proportion of matching instances over total number of instances in the dataset.

Challenges Faced during Development

The first problem faced was to separate every line into feature and class label.

The main problem faced was to convert text into numerical format so that they can be used to train machine-learning models. This required investigation and exploration in the field of Natural Language Processing to come across word embedding techniques.

Implementing models were seamless because of scikit-learn (sklearn) machine learning library in python. It took some time to figure out how to use these models, what inputs they accept, and how to evaluate these models.

Another important challenge faced was to write clear and concise comments to explain every block of code used in the project.

Along with this, steps had to be taken to devise a smooth method to setup project in a variety of platforms to ensure cross platform portability.

Apart from these challenges, the rest of the process was smooth mainly because of the familiarity of concepts learnt in CSC 522 – Automated Learning and Data Analysis.

Acknowledgements

I would like to thank Dr. Bita Akram and the teaching assistants for their continuous support and insights and also for providing an opportunity to work on this interesting problem.

References

Splitting Data Set:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

TF-IDF Vectorization:

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

Support Vector Machines:

https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Logistic Regression:

https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Naïve Bayes Classifier;

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

https://scikit-learn.org/stable/modules/naive_bayes.html

Evaluation Methods:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html