|  |  |
| --- | --- |
| Capstone Project Machine Learning Engineer Nanodegree | Ramkumar SinghAugust 16, 2017 |

Definition

## Project Overview

## Spam filtering is a binary classification task familiar to any user of email services. We will use machine learning classifiers to implement a similar spam filter.

## The task is to distinguish between two types of emails, “spam” and “non-spam” often called “ham”. The machine learning classifier will detect that an email is spam if it is characterised by certain features. The textual content of the email – words like “Viagra” or “lottery” or phrases like “You've won a 100,000,000 dollars! Click here!”, “Join now!” – is crucial in spam detection and offers some of the strongest cues.

## To train the classifier, we need a representative dataset with both spam and ham emails. In this project we will use [Apache SpamAssasin](http://spamassassin.apache.org/old/publiccorpus/) public corpus, which contains about 6047 message with approximate 31% spam ratio.

## Problem Statement

Implement a Spam Filter with the help of a ML classifier which would classify a given mail as spam or ham

## Metric

In real world scenario volume of ham males are generally higher that the spam mails. For such problem f1score is a better metric to analyse performance of a classifier. In the given dataset also the spam ratio is 31% only, so f1score is applicable for the given dataset as well. The solution will evaluate f1score of ML model against the benchmark f1score

Analysis

**Data Exploration**

Apache SpamAssasin public corpus has spam and ham mails data. Below is the overview of data

URL:

* <http://spamassassin.apache.org/old/publiccorpus/>

Content Folder:

* spam: 500 spam messages, all received from non-spam-trap sources.
* easy\_ham: 2500 non-spam messages. These are typically quite easy to differentiate from spam, since they frequently do not contain any spammish signatures (like HTML etc).
* hard\_ham: 250 non-spam messages which are closer in many respects to typical spam: use of HTML, unusual HTML markup, coloured text, "spammish-sounding" phrases etc.
* easy\_ham\_2: 1400 non-spam messages. A more recent addition to the set.
* spam\_2: 1397 spam messages. Again, more recent.

Total count: **6047** messages, with about a **31%** spam ratio.

**Exploratory Visualization**

Emails are pre-processed to normalize, tokenize and lemmatize. After that, for each word, its frequency of appearance is calculated. This word and frequency data is used to create two type of plot for visualization.

1. Word cloud (tag cloud, or weighted list in visual design) is a visual representation of text data. Tags are usually single words, and the importance of each tag is shown with font size or color, which gives greater prominence to words that appear more frequently. Two separate word cloud map is created for spam and ham words. This visualization makes it easier to see relative frequency of words in spam and ham mails.
2. Frequency Histogram : 100 most frequent word and its frequency is plotted for spam email words and ham email words. Y axis has frequency count of top 100 high frequency words and X-axis has the corresponding word. This plot helps in visualizing the actual frequency of words in spam and ham mails.

**Algorithms and Techniques**

To create solution of the problem below are the high level of steps:

* Pre-processing the mail: - In this step we will process the mail content with following changes:
  + Remove html tags
  + Normalize numbers, Urls, email address and Dollar sign ( Replace 0-9 with ‘number, hyperlinks with 'httpaddr', any email address with 'emailaddr' and $ with 'dollar' )
  + Tokenize the words
  + Take only alphanumeric words
  + Lemmatize and change case to lower to reduce the words to their stemmed form.
  + Filter stop words (words which do not have significance like : to, the, a etc)
* Train test Split: - Create separate set for training and testing purpose..
* Extracting the features: - Use the training set data calculate frequency of each word and select high frequency word as feature to be used for learning
* Training different classifier :- Train the classifiers like Logistic Regression, K-Neighbour Classifier, SVM , Naïve Bayes etc
* Evaluating the classifiers: - The above classifiers will be evaluated against the dummy classifier, by comparing their prediction time and f1score.
* Fine Tune the best classifier :- Use GridSearch technique to fine tune the best classifier using different parameters like C, gamma and other applicable parameters

**Benchmark**

In this project we will use the sklearn dummy classifier as a simple baseline to compare with. DummyClassifier is a classifier that makes predictions using simple rules, so the goal is to perform better that this.

Methodology

### Data Preprocessing

Reading Data :

While reading the email from corpus file, opening the file with utf8 encoding. If UnicodeDecodeError occurs due to not supported character, we catch the exception and continue reading next file.

Pre-processing:

In data pre-processing step we normalize, tokenize, lemmatize and filter out stop words and non-alphanumeric words. Below are the details of pre-processing.

* + Clean the mail text by removing html tags
  + Normalize numbers, Urls, email address and Dollar sign ( Replace numeric values with ‘number’ , hyperlinks with 'httpaddr', any email address with 'emailaddr' and $ with 'dollar' )
  + Tokenize the words and take only alphanumeric words
  + Filter stop words (words which do not have significance like : to, the, a etc)
  + Lemmatize and change case to lower to reduce the words to their stemmed form.

Feature Extraction:

We split the data in training and testing sets. To generate NLP features, we use ONLY the training data. While working on GridSearchCV and cross validation, to easily and correctly implement the feature extraction, we are using Sklearn pipelines.

Some of the classifier need dense matrix but the Pipeline produces sparse matrix, so sklearn.preprocessing FunctionTransformer is used to generate dense matrix. This option has an effect on memory footprint.

### Implementation

Pre-processing helper methods:

* ***preprocess\_normalize*** : This method is implemented to normalize the email data using regex rules, which removes html tags and replace numeric values with ‘number’ , hyperlinks with 'httpaddr', any email address with 'emailaddr' and $ with 'dollar'
* ***preprocess\_lemmatize\_tokenize***: This method is using ***nltk* *word\_tokenize*** to tokenize and ***nltk* *WordNetLemmatizer*** to lemmatize the email words. It also filters stopwords using ***nltk.corpus stopwords***. This method also converts email words to lower case and takes only alpha-numeric words to return at the end.
* ***LemmatizerTokenizer***: A custom tokenizer class is implemented which Lemmetize, Tokenize and perform other optimizations using the ***preprocess\_lemmatize\_tokenize*** method mentioned above

Feature Extraction:

* TF-IDF features

For feature extraction, word occurrence count has an issue; longer documents will have higher average count values than shorter documents, even though they might talk about the same topics.

To avoid these potential discrepancies it suffices to divide the number of occurrences of each word in a document by the total number of words in the document: these features are called tf (Term Frequencies).

Another refinement on top of tf is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus. This downscaling is called tf–idf (Term Frequency times Inverse Document Frequency).

To convert a collection of raw documents to a matrix of TF-IDF features, I am using *sklearn.feature\_extraction.text* *TfidfVectorizer*.

* Custom Tokenizer :

***LemmatizerTokenizer***, the custom tokenizer class is passed to *TfidfVectorizer* to be used during feature extraction.

* Pipeline :

We need to generate the NLP features on training data only. This is important because if we generate NLP feature on all of the data and then split the data in train and test set, we have contaminated the test.

Since the NLP features will have already TF-IDF/stats of testing data as well.

This separation of training data and testing data for feature generation and testing becomes complicated with cross validation methods like GridSearchCV, since each new fold need a new set of features. To achieve this separation we are using ***sklearn.pipeline***

* Function Transformer:

Sklearn Pipeline produces sparse matrix, but some of the classifier need dense matrix due to which the pipeline.fit method throws error “*A sparse matrix was passed, but dense data is required*” . To overcome this issue ***sklearn.preprocessing FunctionTransformer*** is passed as an argument to ***sklearn.pipeline make\_pipeline***. This Function Transformer turns a Python function into a Pipeline-compatible transformer object and hence used to convert sparse matrix to dense

Sklearn Classifiiers

Following [*scikit-learn*](http://scikit-learn.org/stable/supervised_learning.html)supervised learning models are used to train the classifier.

* Gaussian Naive Bayes (GaussianNB)
* K-Nearest Neighbors (KNeighbors)
* Logistic Regression
* Support Vector Machines (SVM)

### Logistic Regression

Why: Logistic Regression classification model is a probabilistic model which maximize the posterior class probability. It is also suitable for binomial classification problem like this. Also as it is a high bias/low variance model, it Works well when we have large number of features in comparison to the number of data/sample. So it is a good choice for the problem at hand where the number of data is low but the feature space is relatively large

Advantage: If there's a lot of noise, logistic regression can handle it better. Logistic regression is intrinsically simple, it has low variance and so is less prone to over-fitting

Disadvantage: Unlike SVM, it considers all the points in the data set. This may be not be a preferred approach for certain problems

Source: <https://en.wikipedia.org/wiki/Logistic_regression>

## K-Nearest Neighbours (KNeighbors)

Why: Because the training set size is small so implementing instance based learning such as KNN  
classification can be a good candidate for such scenario. Also we do not have to worry about  
the linear seaparability of the data and implanting the model is very easy as well.

Advantage: Makes no assumption about the data distribution. Learning time is very low. Immediately adapts to the new data not affected by linear separability of the data. An easy algorithm to explain and implement

Disadvantage: Slow during prediction. So difficult to use this for prediction in real time. Storage space can be a challenge if working on more data. Performance impact is high if more features are added (Curse of Dimensionality)

Source: <http://www.dummies.com/programming/big-data/data-science/solving-real-world-problems-with-nearest-neighbor-algorithms/>

## SVM

Why: SVM performs well for most of the cases. It can be used with various Kernel to fit the non  
linear decision boundary as well. Various parameters can be tuned to get a high level of  
accuracy

Advantage High accuracy. By implementing Kernel Trick, we can inject the domain knowledge in the classifier. With appropriate kernel, it can work well even if the data is not linearly separable

Disadvantage. Tuning the model for optimal parameters can be difficult. Memory-intensive. Hard to interpret

Source: <https://en.wikipedia.org/wiki/Support_vector_machine>

Training

A helper function ***train\_predict*** is implemented for training and testing the four supervised learning models we have chosen above.

The function takes as input a classifier, and the training and testing data. It creates a sklearn pipeline using the vectorizer created in data processing step. Using this pipeline, the model is trained.

This function will report the training time , prediction time and f1score for both the training and testing data separately.

We import the dummy classifier and four other supervised learning models mentioned earlier, and run the ***train\_predict*** function for each one. We will perform following action:

* From the X\_ train data set create four different training set of sizes: 25%, 50%, 75% and 100% of data.
* Import the supervised learning models discussed in the previous section.
* Initialize the three models and store them in clf\_A, clf\_B, clf\_C and clf\_D
  + Use a random\_state for each model.
  + Use the default settings for each model — we will tune one specific model in a later section.
* Fit each model with each training set size and make predictions on the test set.

### Refinement

Initially I trained the classifier on sklearn.svm SVC which is not giving a meaningful result. F1\_score is quite low in comparison to other classifiers. Below is the performance matrix for the same

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

A regular SVC with default values uses a radial basis function as the kernel. This nonlinear basis function has higher variance. We can add regularization to the nonlinear model and we will probably see much better results. (This is the C parameter in scikit learn SVMs), however I decided to use a LinearSVC instead since we are training on default parameters and linear kernel has lower variance by default. LinearSVC seems a better choice because we have large feature space in comparison to the training dataset.

The linearSVC with default parameter gives below result, which seems outperforming not only SVC but all other learning models.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Model Tuning :

In this step we fine tune the chosen model. We have used grid search (GridSearchCV) with parameters tuned with at least 3 different values. We have to use the entire training set for this.

Following are the steps implemented for Model Tuning

* Import [sklearn.grid\_search.GridSearchCV](http://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html) and [sklearn.metrics.make\_scorer](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
* Initialize the classifier chosen for fine tuning
* Create the parameters list to tune
* Make an f1 scoring function using 'make\_scorer’ and store it in f1\_scorer
* Initialize Pipeline using the vectorizer, FunctionTransformer and classifier
* Perform grid search on the classifier using the pipeline, parameters and f1\_scorer as the scoring method and store it in grid\_obj
* Fit the grid\_object to the training data and find the optimal parameters,
* Get the best estimator from the grid\_object
* Report the final F1 score for training and testing using the best estimator

Result

### Model Evaluation and Validation

Below are the 16 different outputs— 4 for each model using the varying training set sizes.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Based on the above observation, I have picked LinearSVC model as the best classifier, because the model is giving the highest testing F1\_score among the other models. Training and prediction time is also comparable to other models in the experiment.

Tuning the Model

The best classifier of the model has following parameters and F1 Scores :

* Classifier Best Parameters : {'linearsvc\_\_C': 10, 'tfidfvectorizer\_\_ngram\_range': (1, 1)}
* Tuned model has a training F1 score of 1.0000.
* Tuned model has a testing F1 score of 0.9960.

Confusion Matrix:

Snesitivity Analysis

<http://scikit-learn-general.narkive.com/D4Ui9LzV/sensitivity-analysis>

vectorizer.get\_feature\_names()

Below is the performance of the model while working on different set of data. This shows that the final classifier is robust and performance is not affected much by the changes in input data.

### Justification

The dummy classifier offers performance is as below.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

The final selected SVM model has below performance matrix over the same set of data.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

The result shows that the SVM model has learnt significantly from the training data set and able to predict lot better on the test data set.

Conclusion

### Free-Form Visualization

Feature Heat Chart/histogram with tf-idf value

<https://buhrmann.github.io/tfidf-analysis.html>

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

* *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
* *Is the visualization thoroughly analyzed and discussed?*
* *If a plot is provided, are the axes, title, and datum clearly defined?*

### Reflection

Cleaning the data and preprocessing step to normalize, tokenize and lemmatize the input data is quite interesting. There is lot of preprocessing needed to clean and transform the data, to make it usable to work with feature extraction and model training.

Extracting the feature using vectorizatoin and implanting pipeline for gridsearchCV was not intuitive earlier. Got to know about these during proposal review and hence learned about the limitation of earlier approach.

Also earlier I was relying on word frequency count for feature selection and relying on an empirical value. While learning about vectorization, came across the TF-IDF (Term Frequency times Inverse Document Frequency), which made more sense to me and hence I have used the same in this project.

### Improvement

While researching about Email Spam classifiers, I tried to find out how this is implemented in the email providers like Gmail or Hotmail. I came to know that in 2015, Google added Deep Learning capabilities to the Gmail spam filtering system and it has improved their accuracy in a great deal.

Deep Learning achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts. It also automatically finds out the features which are important for classification, But deep learning algorithms need a large amount of data to understand it perfectly.

With large amount of data and implementing a Deep Learning classifier we may achieve a better result.

Also, the classifiers in this project just analyse the textual content of the email. Any information about sender’s history and reputation can help in achieving better performance.