Capstone Project

Machine Learning Engineer Nanodegree

Ramkumar Singh

August 27, 2017

Contents

I. Definition

A. 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 100,000,000 dollars! Click here!", "Join now!" – is crucial in spam detection and offers some of the strongest clues.

To train the classifier, we need a representative dataset with both spam and ham emails. In this project, we will use Apache SpamAssasin public corpus, which contains about 6047 message with approximate 30% spam ratio.

B. Problem Statement

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

The goal is to create a binary classifier which could mark an email as spam or ham with an acceptable accuracy. The tasks involved are as below:

- Download Spam and Ham mail data from Apache SpamAssasin public corpus.
- Pre-process email to normalize, lemmatize and tokenize the email worlds
- Extract Feature from the spam and ham mail data
- Train various classifiers and compare their performances
- Fine tune the hyper parameters of best classifier model selected in above step

The final classifier is expected to be useful in in classifying email as spam or ham and should perform better than sklearn dummy classifier.

C. Metric

In real world scenario volume of ham males are generally higher that the spam mails. In the given dataset, also the spam ratio is 30% only.

In this case Classification accuracy not enough since, it is the number of correct predictions made divided by the total number of predictions. So, a model that only predicted Ham mail in all case, would achieve an accuracy of approximately 70%. This is a high accuracy, but model is literally doing nothing and blindly classifying all mails as Ham.

For such problem f1score is a better metric to analyse performance of a classifier. It considers both the precision and the recall of the test to compute the score. The solution will evaluate f1score of ML model against the benchmark f1score

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

$$\mathrm{Precision} = \frac{tp}{tp + fp}$$

$$\mathrm{Recall} = \frac{tp}{tp + fn}$$

Where ***tp*** is true positive, ***fp*** is false positive and ***fn*** is false negative

II. Analysis

A. Data Exploration

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

Spam Mail Example

```
'From allcontacts-bounce@briefs.ein.cz  Wed Aug 28 15:29:15 2002\nReturn-Path:
<allcontacts-bounce@briefs.ein.cz>\nDelivered-To:
zzzz@localhost.spamassassin.taint.org\nReceived: from localhost (localhost
[127.0.0.1])\n\tby phobos.labs.spamassassin.taint.org (Postfix) with ESMTP id
BCE1D44155\n\tfor <zzzz@localhost>; Wed, 28 Aug 2002 10:29:14 -0400
(EDT)\nReceived: from phobos [127.0.0.1]\n\tby localhost with IMAP (fetchmail-
5.9.0)\n\tfor zzzz@localhost (single-drop); Wed, 28 Aug 2002 15:29:14 +0100
(IST)\nReceived: from bay.ein.cz (bay.ein.cz [62.24.69.193]) by\n
dogma.slashnull.org (8.11.6/8.11.6) with ESMTP id g7SEMqZ27998 for\n
<webmaster@efi.ie>; Wed, 28 Aug 2002 15:22:52 +0100\nReceived: from oak.ein.cz
(oak.ein.cz [62.24.69.194]) by bay.ein.cz\n    (Postfix) with ESMTP id
2E591135B8; Wed, 28 Aug 2002 15:59:13 +0200 (CEST)\nReceived: by oak.ein.cz
(Postfix, from userid 1002) id F3B511BEA0;\n    Wed, 28 Aug 2002 15:58:49 +0200
(CEST)\nFrom: EIN News <pe@europeaninternet.com>\nTo: EIN Media
<allcontacts@briefs.ein.cz>\nSubject: Iraq Today Daily News - FREE Trial\nX-
Author: pe@einmedia.com\nMessage-Id:
<20020828135844.C27501BE9F@oak.ein.cz>\nDate: Wed, 28 Aug 2002 15:58:44 +0200
(CEST)\nSender: allcontacts-bounce@briefs.ein.cz\nErrors-To: allcontacts-
bounce@briefs.ein.cz\nPrecedence: bulk\n\nDear Sir/Madam,\n\nMy name is Petr
Stanek and I am managing the Iraq Today\nnews service
(www.europeaninternet.com/iraq). Iraq Today\ncontains hourly updated breaking
news headlines, exchange rates,\nmarket news and other important
information.\n\nYou and your associates can have a FREE TRIAL\nSUBSCRIPTION to
Iraq Today.\n\nYou will have access to a collection of 25,000 daily
updated\narticles, a news archive and many other benefits too.\n\nOnce again,
this trial is FREE. To subscribe, just reply to this\ne-mail or sign up
at:\nhttp://www.europeaninternet.com/login/affiliate_register.php3\n\nA partial
list of current EIN subscribers can be found
at:\nhttp://www.europeaninternet.com/mediakit/\n\nIf you have any questions,
comments, or need assistance signing\nup, please contact us personally by
either writing to\nhelpdesk@europeaninternet.com or simply replying to this
email.\n\nPlease feel free to forward this offer to any of your
colleagues.\n\nBest regards,\n\nPetr Stanek\nSubscription Department\nEIN
News\n\n\n\n\n\n\n\n\n\n\n\nTo be removed please reply to:
remove@europeaninternet.com\n\n\n\n\n'
```

Ham Mail Example

```
'From ilug-admin@linux.ie  Thu Aug 22 16:27:21 2002\nReturn-Path: <ilug-
admin@linux.ie>\nDelivered-To: zzzz@localhost.netnoteinc.com\nReceived: from
localhost (localhost [127.0.0.1])\n\tby phobos.labs.netnoteinc.com (Postfix)
with ESMTP id 7A28A43F99\n\tfor <zzzz@localhost>; Thu, 22 Aug 2002 11:27:21 -
0400 (EDT)\nReceived: from phobos [127.0.0.1]\n\tby localhost with IMAP
(fetchmail-5.9.0)\n\tfor zzzz@localhost (single-drop); Thu, 22 Aug 2002
16:27:21 +0100 (IST)\nReceived: from lugh.tuatha.org (root@lugh.tuatha.org
[194.125.145.45]) by\n    dogma.slashnull.org (8.11.6/8.11.6) with ESMTP id
g7MFQmZ12280 for\n    <zzzz-ilug@spamassassin.taint.org>; Thu, 22 Aug 2002
16:26:48 +0100\nReceived: from lugh (root@localhost [127.0.0.1]) by
lugh.tuatha.org\n    (8.9.3/8.9.3) with ESMTP id QAA07188; Thu, 22 Aug 2002
16:25:32 +0100\nReceived: from moe.jinny.ie ([193.120.171.3]) by
lugh.tuatha.org\n    (8.9.3/8.9.3) with ESMTP id QAA07145 for <ilug@linux.ie>;
Thu,\n    22 Aug 2002 16:25:24 +0100\nX-Authentication-Warning:
lugh.tuatha.org: Host [193.120.171.3] claimed to\n    be
moe.jinny.ie\nReceived: from jlooney.jinny.ie (unknown [193.120.171.2]) by
moe.jinny.ie\n    (Postfix) with ESMTP id 938BD7FC46; Thu, 22 Aug 2002 16:25:23
+0100 (IST)\nReceived: by jlooney.jinny.ie (Postfix, from userid 500) id
4F57189D;\n    Thu, 22 Aug 2002 16:25:45 +0100 (IST)\nDate: Thu, 22 Aug 2002
16:25:45 +0100\nFrom: "John P. Looney" <valen@tuatha.org>\nTo: linux-
raid@vger.kernel.org\nCc: ilug@linux.ie\nMessage-Id:
```

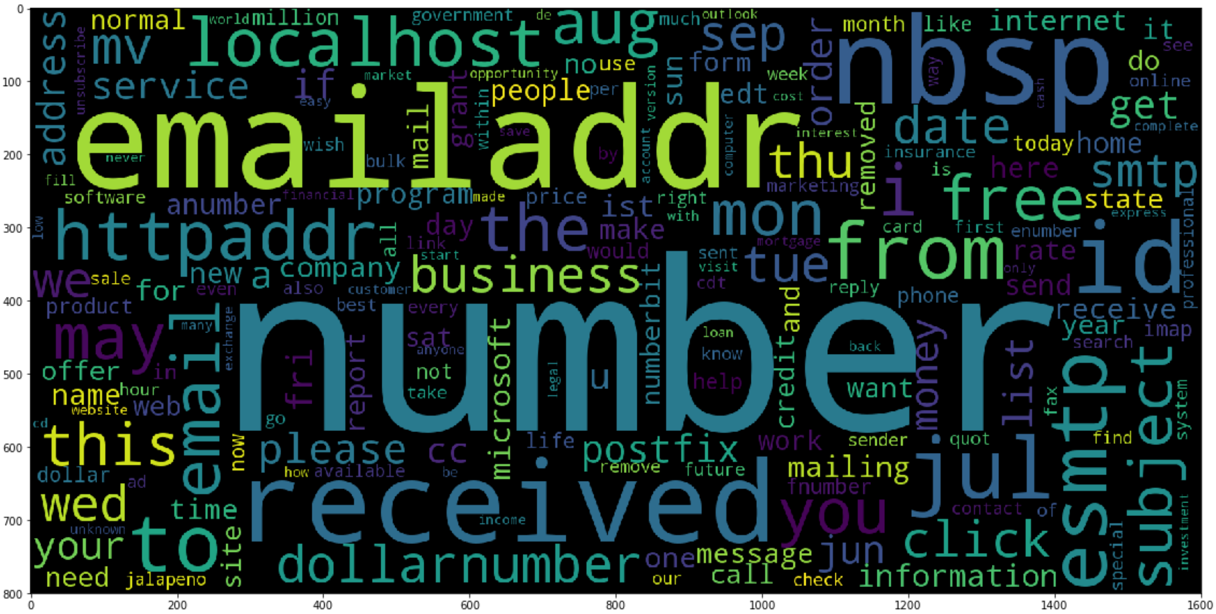
```
<20020822152545.GJ3670@jinny.ie>\nReply-To: valen@tuatha.org\nMail-Followup-To:
linux-raid@vger.kernel.org, ilug@linux.ie\nReferences:
<200208172056.g7HKuHm05754@raq.iceblink.org>\n
<1029624922.14769.119.camel@atherton> <20020819140815.GY26818@jinny.ie>\nMIME-
Version: 1.0\nContent-Type: text/plain; charset=us-ascii\nContent-Disposition:
inline\nIn-Reply-To: <20020819140815.GY26818@jinny.ie>\nUser-Agent:
Mutt/1.4i\nX-Os: Red Hat Linux 7.3/Linux 2.4.18-3\nX-Url:
http://www.redbrick.dcu.ie/~valen\nX-Gnupg-Publickey:
http://www.redbrick.dcu.ie/~valen/public.asc\nSubject: [ILUG] Re: Problems with
RAID1 on cobalt raq3\nSender: ilug-admin@linux.ie\nErrors-To: ilug-
admin@linux.ie\nX-Mailman-Version: 1.1\nPrecedence: bulk\nList-Id: Irish Linux
Users\' Group <ilug.linux.ie>\nX-Beenthere: ilug@linux.ie\n\nOn Mon, Aug 19,
2002 at 03:08:16PM +0100, John P. Looney mentioned:\n>  This is likely because
to get it to boot, like the cobalt, I\'m actually\n> passing root=/dev/hda5 to
the kernel, not /dev/md0. \n\n Just to solve this...the reason I was booting
the box with\nroot=/dev/hda5, not /dev/md0 was because /dev/md0 wasn\'t booting
- it\nwould barf with \'can\'t find init\'.\n\n It turns out that this is
because I was populating md0 with tar. Which\nseems to have \'issues\' with
crosslinked files - for instance, it was\ntrying to make a hard link of
glibc.so to hda - and failing. It was only\nas I did it again with a friend
present, that he spotted the errors, and\nqueried them. We noticed that the
hard linked files just didn\'t exist on\nthe new rootfs.\n\n When we duplicated
the filesystems with dump instead of tar, it worked\nfine, I was able to tell
lilo to use root=/dev/md0 and everything worked.\n\n Woohoo.\n\nKate\n\n\n--
\nIrish Linux Users\' Group:
ilug@linux.ie\nhttp://www.linux.ie/mailman/listinfo/ilug for (un)subscription
information.\nList maintainer: listmaster@linux.ie\n\n'
```


B. Exploratory Visualization

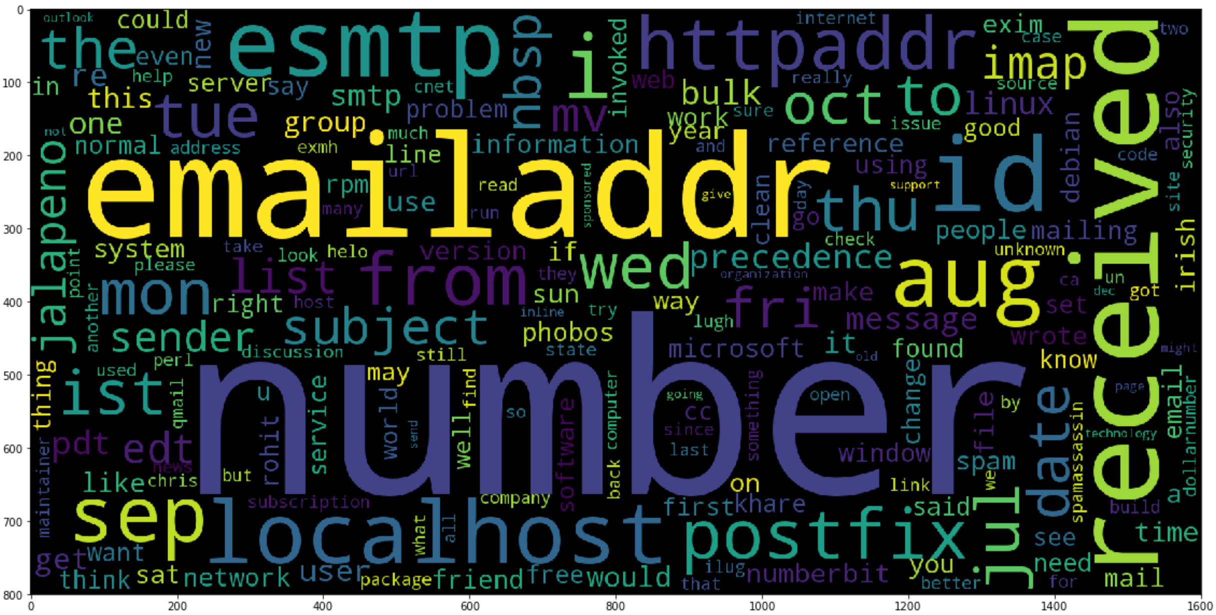
Emails are pre-processed to normalize, tokenize and lemmatize. After that, for each word, its frequency of appearance is calculated. This word and frequency data is used to create two type of plot for visualization.

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.

Spam Mail Word Cloud:

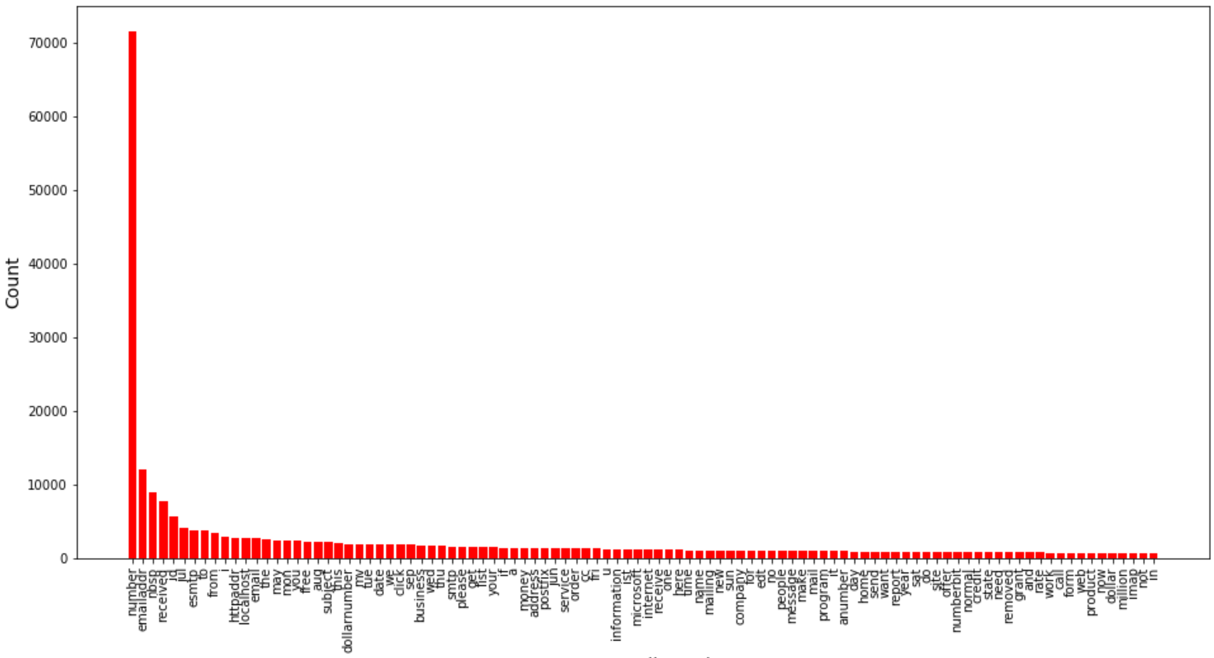


Ham Mail Word Cloud:

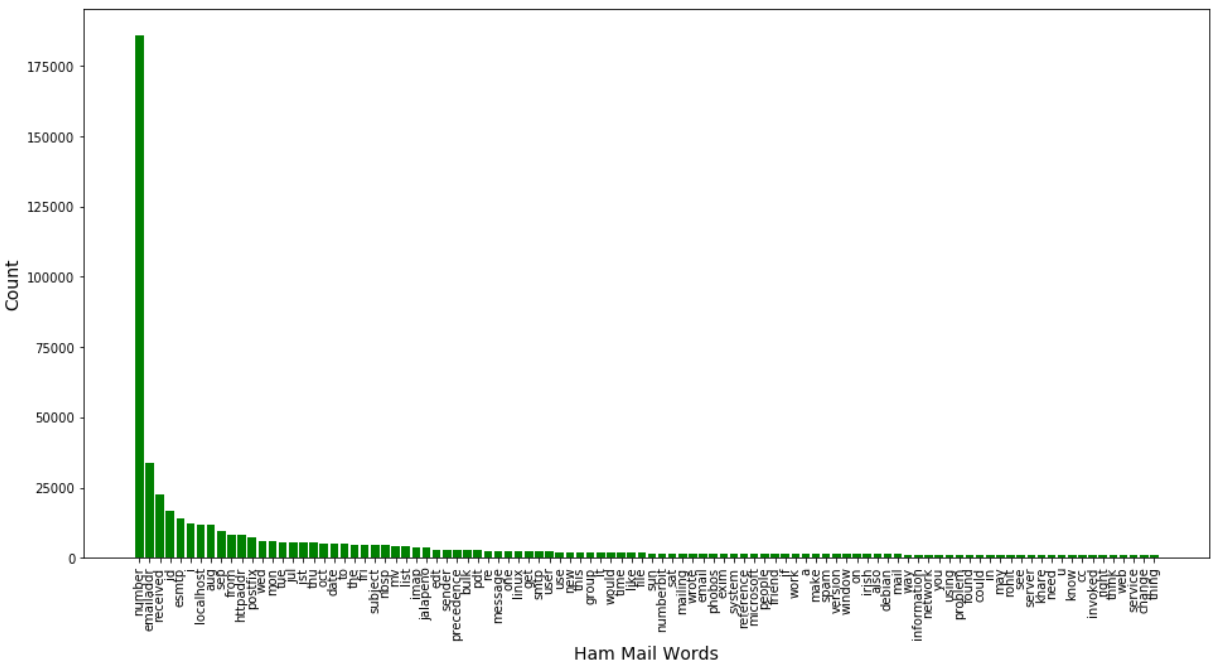


2. Frequency Histogram: 100 most frequent word and its frequency is plotted for spam email words and ham email words. Y axis has frequency count of top 100 high frequency words and X-axis has the corresponding word. This plot helps in visualizing the actual frequency of words in spam and ham mails.

Spam Mail Word Frequency Histogram (Top 100):



Ham Mail Word Frequency Histogram (Top 100):



C. Algorithms and Techniques

Sklearn Classifiers

Since the problem at hand is a typical binary classification problem. A bunch of classifier is evaluated for the classification task. Following *scikit-learn* supervised learning models are selected

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

- **Gaussian Naïve Bayes**

  Naive Bayes classifiers is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It is a simple technique for constructing classifiers that assign class labels to problem instances, represented as vectors of feature values.

  Why: Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. It only requires a small number of training data to estimate the parameters necessary for classification. Also, the Naive Bayes has been used real Spam classification system.

  Advantage: Naive Bayes classifiers are highly scalable, requiring parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers

  Disadvantage: Due the feature independence assumption, it loses the ability to exploit interaction between features, however for classification task this is often is not a problem.

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

- **Logistic Regression**

  Logistic Regression classification model is a probabilistic model which maximize the posterior class probability. It is used to model dichotomous (binary) outcome variables. In this model log odds of the outcome is modeled as a linear combination of the predictor variables

Why: Logistic Regression 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 being a preferred approach for certain problems

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

- **K-Nearest Neighbours (KNeighbors)**
  In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors

  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/
  https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

- **SVM**

  A Support Vector Machine (SVM) is a classifier formally defined by a separating hyperplane. It maps the points in space, so that the separate categories are divided by a clear gap that is as wide as possible (maximizing the margin)

  By implementing Kernel trick, we can perform nonlinear transformation in higher dimensional space. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space.

  Why: SVM performs well for most of the cases. It can be used with various Kernel to fit the nonlinear 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

Feature Extraction:

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).

Avoid overfitting during Cross Validation:

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.

D. 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.

III. Methodology

A. Data Pre-processing

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.

Processed Spam Mail:

```
(['from', 'emailaddr', 'wed', 'aug', 'number', 'number', 'number', 'number',
'number', 'emailaddr', 'received', 'localhost', 'localhost', 'postfix',
'esmtp', 'id', 'bcenumberdnumber', 'wed', 'number', 'aug', 'number', 'number',
'number', 'number', 'edt', 'received', 'phobos', 'localhost', 'imap',
'emailaddr', 'wed', 'number', 'aug', 'number', 'number', 'number', 'number',
'ist', 'received', 'esmtp', 'id', 'gnumbersemqznumber', 'wed', 'number',
'aug', 'number', 'number', 'number', 'number', 'received', 'postfix', 'esmtp',
'id', 'numberenumberbnumber', 'wed', 'number', 'aug', 'number', 'number',
'number', 'number', 'cest', 'received', 'postfix', 'userid', 'number', 'id',
'fnumberbnumberbeanumber', 'wed', 'number', 'aug', 'number', 'number',
'number', 'number', 'cest', 'from', 'ein', 'news', 'to', 'ein', u'medium',
'subject', 'iraq', 'today', 'daily', 'news', 'free', 'trial', 'emailaddr',
'date', 'wed', 'number', 'aug', 'number', 'number', 'number', 'number',
'cest', 'sender', 'emailaddr', 'emailaddr', 'precedence', 'bulk', 'dear',
'my', 'name', 'petr', 'stanek', 'i', 'managing', 'iraq', 'today', 'news',
'service', 'iraq', 'today', 'contains', 'hourly', 'updated', 'breaking',
'news', u'headline', 'exchange', u'rate', 'market', 'news', 'important',
'information', 'you', u'associate', 'free', 'trial', 'subscription', 'iraq',
'today', 'you', 'access', 'collection', 'number', 'number', 'daily',
'updated', u'article', 'news', 'archive', 'many', u'benefit', 'once', 'trial',
'free', 'to', 'subscribe', 'reply', 'sign', 'httpaddr', 'a', 'partial',
'list', 'current', 'ein', u'subscriber', 'found', 'httpaddr', 'if',
u'question', u'comment', 'need', 'assistance', 'signing', 'please', 'contact',
u'u', 'personally', 'either', 'writing', 'emailaddr', 'simply', 'replying',
'email', 'please', 'feel', 'free', 'forward', 'offer', u'colleague', 'best',
u'regard', 'petr', 'stanek', 'subscription', 'department', 'ein', 'news',
'to', 'removed', 'please', 'reply', 'emailaddr'],)
```

Processed Ham Mail:

```
(['from', 'emailaddr', 'thu', 'aug', 'number', 'number', 'number', 'number',
'number', 'emailaddr', 'received', 'localhost', 'localhost', 'postfix',
'esmtp', 'id', 'numberanumberanumberfnumber', 'thu', 'number', 'aug',
'number', 'number', 'number', 'number', 'edt', 'received', 'phobos',
'localhost', 'imap', 'emailaddr', 'thu', 'number', 'aug', 'number', 'number',
'number', 'number', 'ist', 'received', 'emailaddr', 'esmtp', 'id',
'gnumbermfqmznumber', 'thu', 'number', 'aug', 'number', 'number', 'number',
'number', 'received', 'lugh', 'emailaddr', 'esmtp', 'id', 'qaanumber', 'thu',
'number', 'aug', 'number', 'number', 'number', 'number', 'received', 'esmtp',
'id', 'qaanumber', 'thu', 'number', 'aug', 'number', 'number', 'number',
'number', 'host', 'claimed', 'received', 'unknown', 'postfix', 'esmtp', 'id',
'numberbdnumberfcnumber', 'thu', 'number', 'aug', 'number', 'number',
'number', 'number', 'ist', 'received', 'postfix', 'userid', 'number', 'id',
'numberfnumberd', 'thu', 'number', 'aug', 'number', 'number', 'number',
'number', 'ist', 'date', 'thu', 'number', 'aug', 'number', 'number', 'number',
'number', 'from', 'john', 'looney', 'to', 'emailaddr', 'cc', 'emailaddr',
'emailaddr', 'emailaddr', 'emailaddr', u'reference', 'inline', 'red', 'hat',
'linux', 'httpaddr', 'httpaddr', 'subject', 'ilug', 're', u'problem',
'raidnumber', 'cobalt', 'raqnumber', 'sender', 'emailaddr', 'emailaddr',
'precedence', 'bulk', 'irish', 'linux', u'user', 'group', 'emailaddr', 'on',
'mon', 'aug', 'number', 'number', 'number', 'number', 'numberpm', 'john',
'looney', 'mentioned', 'this', 'likely', 'get', 'boot', 'like', 'cobalt', 'i',
'actually', 'passing', 'kernel', 'just', 'solve', 'reason', 'i', 'booting',
'box', 'booting', 'would', 'barf', 'find', 'init', 'it', u'turn', 'i',
'populating', 'mdnumber', 'tar', 'which', 'seems', 'crosslinked', u'file',
'instance', 'trying', 'make', 'hard', 'link', 'hda', 'failing', 'it', 'i',
'friend', 'present', 'spotted', u'error', 'queried', 'we', 'noticed', 'hard',
'linked', u'file', 'exist', 'new', 'rootfs', 'when', 'duplicated',
'filesystems', 'dump', 'instead', 'tar', 'worked', 'fine', 'i', 'able',
'tell', 'lilo', 'use', 'everything', 'worked', 'woohoo', 'kate', 'irish',
```

```
'linux', u'user', 'group', 'emailaddr', 'httpaddr', 'un', 'subscription',
'information', 'list', 'maintainer', 'emailaddr'],)
```

B. Implementation

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

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:**
  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 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 influences memory footprint

- **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 Classifiers

Following *scikit-learn* supervised learning models are used to train the classifier.

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

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.

Challenges and Observations:

I was facing intermittent memory exception on an old laptop having i3 processor and 6 GB RAM. Later I used a laptop with i5 processor and 8 GB RAM. It helped me in running the code faster without memory issue.

In the input data, there are mails with nonstandard keyboard characters. These non-Unicode characters were causing issues in nltk tokenizers These data are filtered while reading by encoding as utf8 character.

Used sklearn.model_selection train_test_split at the place of sklearn.cross_validation train_test_split as later was giving deprecation warning. The sklearn grid_search.GridSearchCV is still giving deprecation warning which is ignored in the code.

C. 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. 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.

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 and sklearn.metrics.make_scorer.
- 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

IV. Result

A. Model Evaluation and Validation

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

Training Classifier: GaussianNB				
Training Set Size	Training Time	Prediction Time (test)	f1 score (training)	f1 score (testing)
1107	12.021	10.775	0.9985	0.7766
2214	19.214	10.936	0.9992	0.8402
3321	29.743	11.136	0.9985	0.8529
4429	40.91	11.493	0.9989	0.8677

Training Classifier: LogisticRegression				
Training Set Size	Training Time	Prediction Time (test)	f1 score (training)	f1 score (testing)
1107	9.893	9.509	0.894	0.8306
2214	18.691	10.193	0.9298	0.904
3321	28.036	10.767	0.9453	0.9219
4429	37.275	9.884	0.9573	0.9329

Training Classifier: KNeighborsClassifier				
Training Set Size	Training Time	Prediction Time (test)	f1 score (training)	f1 score (testing)
1107	9.299	60.775	0.8486	0.7885
2214	20.927	166.08	0.8993	0.8311
3321	33.955	274.445	0.9123	0.8576
4429	48.149	493.512	0.9333	0.8826

Training Classifier: LinearSVC				
Training Set Size	Training Time	Prediction Time (test)	f1 score (training)	f1 score (testing)
1107	11.533	9.919	0.997	0.9601
2214	18.27	9.88	0.9985	0.9721
3321	29.858	9.749	0.998	0.9736
4429	38.874	10.208	0.9985	0.9796

In the above table gives we can see that KNN classification is very slow as per expected and discussed in the Sklearn Classifiers section of Algorithm. Gaussian Naïve Bayes and Logistic Regression Classifier are having similar training time and prediction time, but Logistic Regression Classifiers has higher score. Overall Linear SVC model seems to be 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 Logistic Regression and Gaussian NB models in the experiment.

Tuning the Model

LinearSVC, which is the best classifier as per the experiment is tuned further. This results a final model having following F1 Scores:

- Training F1 score of 0.9993.
- Testing F1 score of 0.9855.

Best Classifier Parameters

Classifier Best Parameters: {'linearsvc__C': 10, 'tfidfvectorizer__ngram_range': (1, 1)}

Confusion Matrix:

	Predicted		
		Ham	Spam
	Ham	3841	4
Actual	Spam	8	1684

Snesitivity Analysis

We are analyzing performance of the model while working on different set of data. The final classifier gives following scores on different set of data:

- F1 score on 25% of training data 0.9985.
- F1 score on 50% of training data 0.9992.
- F1 score on 75% of training data 0.9990.
- F1 score on entire data 0.9964

The above data shows that the final classifier is robust and performance is not affected much by the changes in input data.

B. Justification

The dummy classifier offers performance is as below. The F1 Score for testing and training is quite low. This score represents performance of a model which do not learn anything.

Training Classifier: DummyClassifier , Time in Seconds				
Training Set Size	Training Time	Prediction Time (test)	f1 score (training)	f1 score (testing)
1107	9.316	10.508	0.2888	0.3127
2214	18.956	11.844	0.298	0.3014
3321	28.462	10.744	0.2981	0.2917
4429	39.874	9.797	0.2996	0.315

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

Training Classifier: LinearSVC				
Training Set Size	Training Time	Prediction Time (test)	f1 score (training)	f1 score (testing)
1107	11.533	9.919	0.997	0.9601
2214	18.27	9.88	0.9985	0.9721
3321	29.858	9.749	0.998	0.9736
4429	38.874	10.208	0.9985	0.9796

The result shows that the LinearSVC model has learnt significantly from the training data set and able to predict lot better on the test data set. The fine-tuned Linear SVC model perform marginally better that the LinearSVC with default parameters. The fine-tuned classifier is giving training F1 score of 0.9993 and Testing score of 0.9855.

By looking at the confusion matrix, it is evident that out of 3845 ham mails, model is able to successfully classify 3841 emails and out of 1692 ham mails, the model is able to classify 1684 emails. The false positive count is 4 and false negative is 8. Cases of false positive and false negative are quite low.

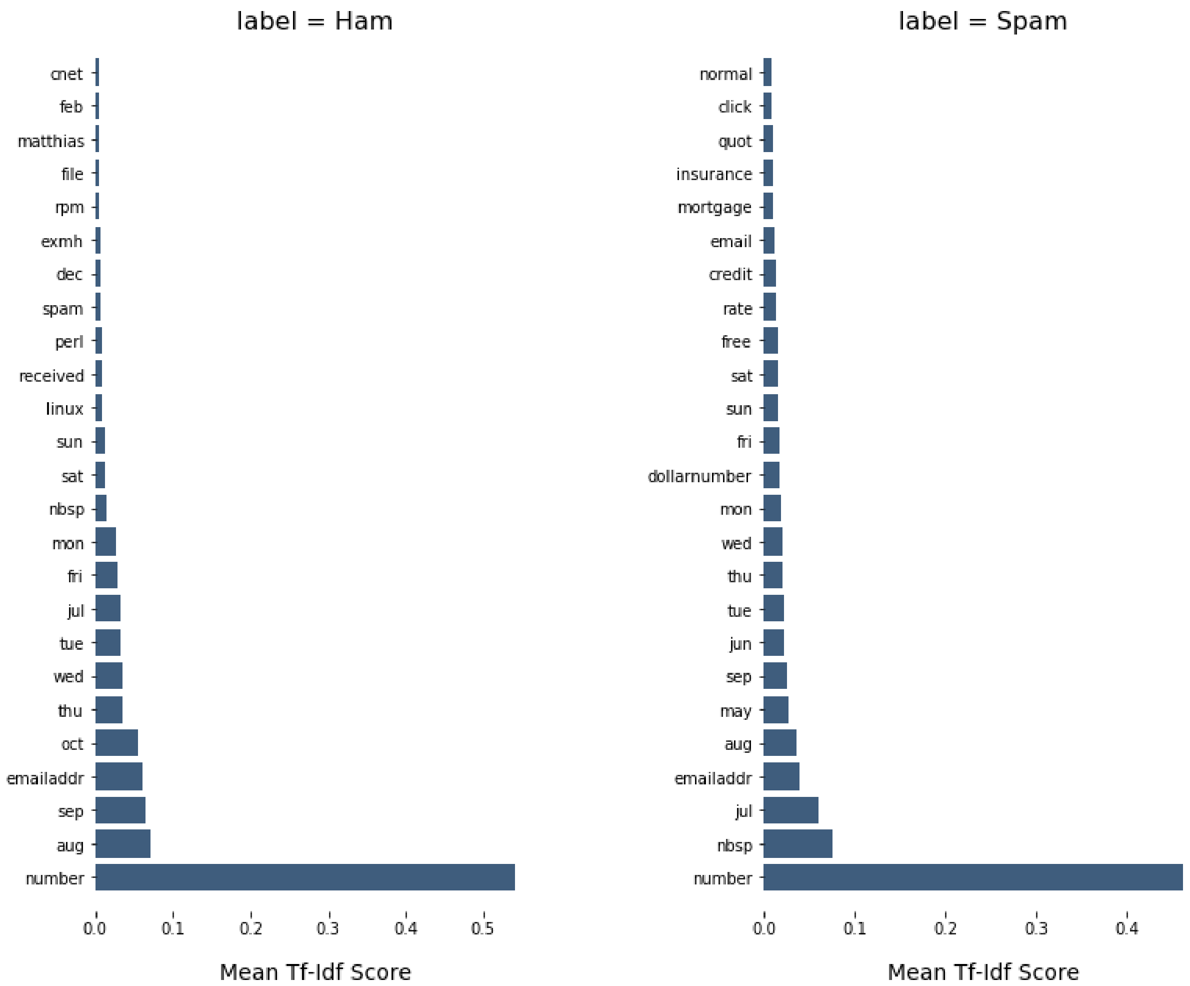
V. Conclusion

A. Free-Form Visualization

TF-IDF Analysis:

TF-IDF vectorization of a corpus of text documents assigns each word in a document a number that is proportional to its frequency in the document and inversely proportional to the number of documents in which it occurs. Very common words, thereby receive heavily discounted tf-idf scores, in contrast to words that are very specific to the document in question. Visualizing such TF-IDF score for Spam and Ham mails below

Source: https://buhrmann.github.io/tfidf-analysis.html



B. 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 to decide the selection cut-off. 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.

C. 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 analyze the textual content of the email. Any information about sender's history and reputation can help in achieving better performance.