Machine Learning Engineer Nanodegree

Heimdall - Email classifier tool

Jean Jung

February 22st, 2017

I. Definitions

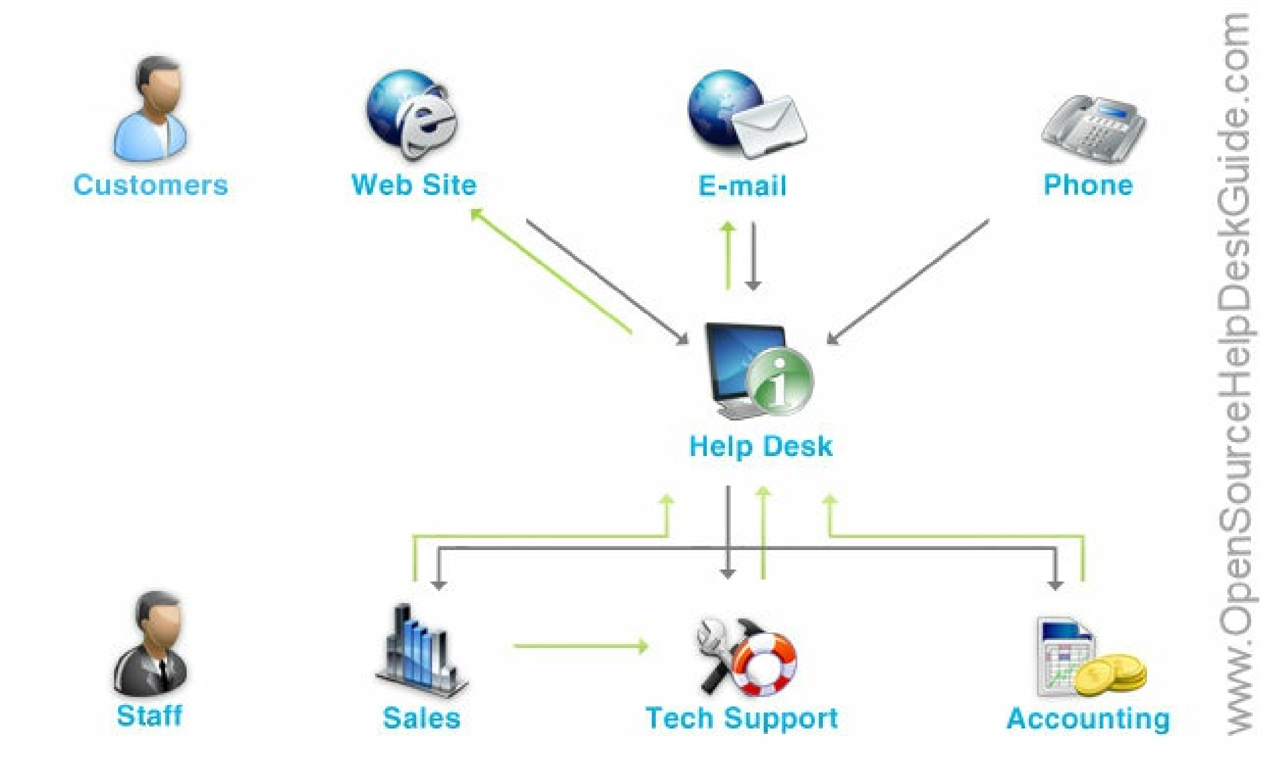
Project Overview

In norse mythology, Heimdall or Heimdallr is the gatekeeper of Bitfröst, the bridge that connects Midgard (mans' earth) to Asgard (Gods' realm). In other words he controls unwanted access to Gods' realm.

This project will try to do both, control unwanted emails received on a mail box and classify wanted emails to be forwarded to someone who is capable of reading, understanding and resolving it as fast as possible.

Problem Statement

All day in help desk systems there are a lot of emails coming from different sources complaining about different problems. Each of those require specific abilities and know-how of some part of the organization's process. A common way to avoid every employee needing to know about the entire organization's processes is to have different groups to check different issues. With this layout there is a triage to check for what group an issue should be sent. This is represented on the image bellow:



Here Help Desk is the centralizer and forward it to the rest of the company. The goal here is not automatize all the Help Desk tasks, but classify incoming issues and create sub-groups of people who can read and understand specific problems, may be avoiding forward a simple problem or doubt to another department.

It takes time to classify every incoming request, the operator needs to open the request, identify the requester, read its content, see if it is a valid request, understand what the requester needs, identify the staff group it belongs to finally classify and forward the request, and this must be done to every request issued.

It would be a great time gain if this could be done automatically as son as any request arrives.

To solve this problem, Heimdall will integrate with help desk systems available on market to get access to email contents, analyze historical data of classified emails training a Supervised Learning Classification model. When ready, it will start to inspect every incoming email and classify them using any type of tagging system available from the chosen platform or forwarding the email to a previously defined list for each issue class.

Metrics

Accuracy

Heimdall will use Accuracy metric to measure performance of models being used/tested to solve the problem.

Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common method used to measure performance on classification models.

The formula can be expressed as:

```
X = t / n * 100
```

Where `X` is the accuracy, `t` is the number of correctly classified samples and `n` is the total number of samples, thus the accuracy will be a number between 0 and 1 representing the percentage of right predictions made.

This can be used to estimate how many emails would be correctly classified per day, for example.

Confusion matrix

During development a confusion matrix can be used to identify the exactly failing point and help to improve the classification model.

A Confusion matrix consists of a table in form `N * N` where `N` is the number of possible values or classes. Every cell has the count of a `predicted vs actual` conflict with the respective values represented by row and column positions.

Example:

```
  A  B  C
A  10 2  3
B  9  20 0
C  1  5  7
```

With the given matrix one can see that `A` was predicted correctly 10 times and there was 2 times it was predicted when the actual value was `B`.

II. Analysis

Data Exploration

One of the best raw text example datasets available on internet currently is the 20 newsgroups dataset, freely distributed on scikit-learn python's library.

This dataset contains around 180000 newsgroups posts on 20 subtopics. The dataset itself is already split on train and test parts to be used on learning models.

Even this dataset is not composed by emails, it can be used to train and measure a email classifier tool, since the tool use only general text classifying tools.

Scikit-learn has functions to fetch and load the data into python arrays containing raw text and labels related. By using the `sklearn.datasets.fetch_20newsgroups` function to retrieve the data a python object with the following two attributes will be returned:

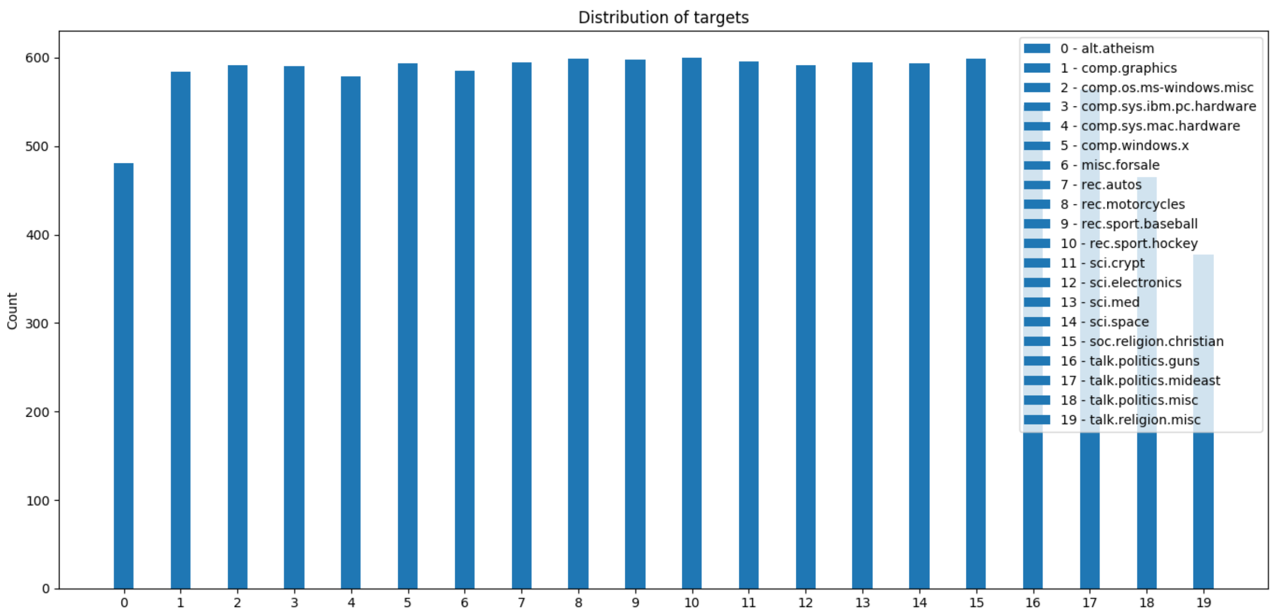
- **data**: A list containing raw texts from news posts.
- **target**: A list where each value is the label of the correspondent entry on the data list.

According to the documentation, the categories are:

```
'alt.atheism',
'comp.graphics',
'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware',
'comp.sys.mac.hardware',
'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc'
```

For Heimdall it will be used only between 4 and 6 categories as this would be the average number of departments a company will have. There is no need of a model that is capable of identifying 20 categories.

Exploring the dataset, the distribution of the categories is described on image bellow:



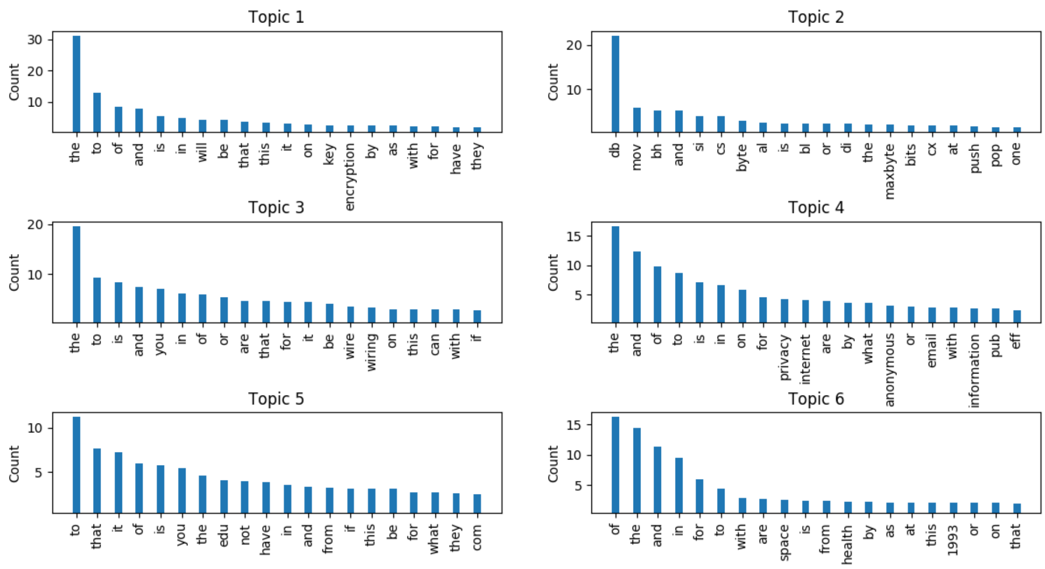
This graph show that the data is well distributed, just some of the categories like `alt.atheism` and `talk.religion.misc` have less examples than others. The `sci` categories group will be chosen to validate Heimdall's model since the distribution is well balanced on that classes and the contents will be very similar, the same as a help desk system.

Filtering dataset to use just chosen categories, data end up with:

```
Size of train data: 2373 samples
Size of test data: 1579 samples
```

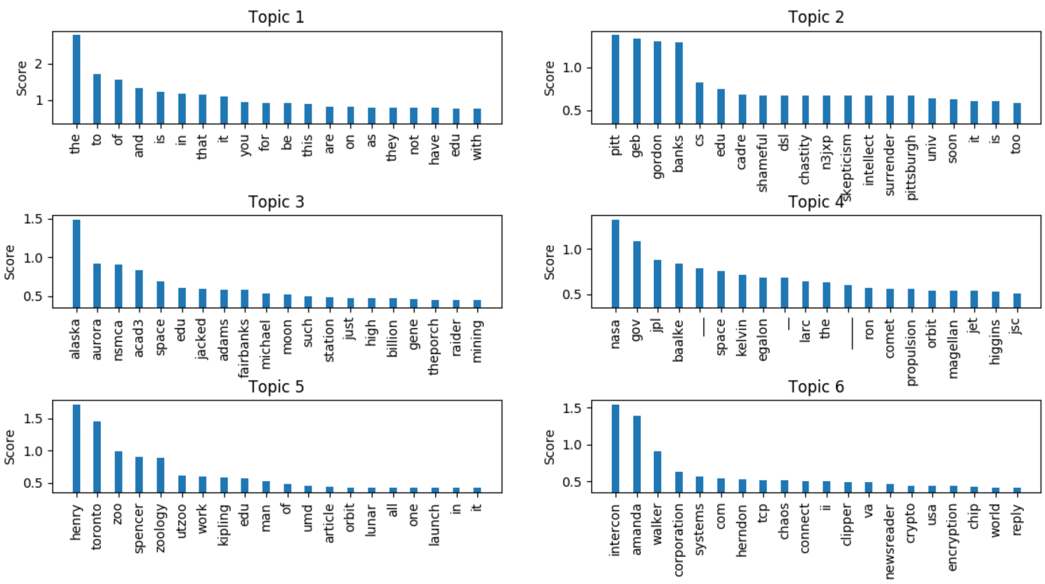
Exploratory Visualization

Below image shows the most repeatable words for six arbitrary documents:



Prepositions and articles like 'the', 'and', 'is', 'in' and etc. appears much more than other words in the great majority of documents. With the exception of Topic 2, all other topics seems to be natural text so this will likely happen with almost any email delivered on the internet today.

It can turn things difficult to classify. To avoid this problem a TF-IDF (Term Frequency–Inverse Document Frequency) can be used to find the words that appear only in few documents. The TF-IDF functionality will be explained later on this paper. Using the TF-IDF algorithm shows us the following graphic:



With TF-IDF all words for each topic have changed turning into more specific words, this make possible to identify a document by analyzing presence of some of these words.

Algorithms and Techniques

In order to achieve his objective Heimdall will use the Bag of Words model, commonly used on natural language processing. The idea behind this model is to create a dictionary containing each word that appears on a document and relating it to a previously chosen metric.

One of the great advantages of using this model is the simplicity and extensibility it supports. There are a lot of implementations based on it.

Feature extraction algorithms

A raw text itself is not a good input source for a classifier algorithm. Classification models expect it's input data to be something with measurable and comparable values, also called features. What do you can extract from "Hi, I have a problem", for us humans it's easy to say that it seems there is someone with problems but what a computer algorithm can extract from this data? The size? Maybe but how to classify texts by size? Large, short? Those are not the target labels here.

This is what feature extraction extends for. Algorithms that transform raw text documents into matrices with relevant features like the frequency a specific word appears for example.

There are a lot of variations and with scikit-learn it's simple to do a benchmark through them, so this will be done. Bellow you can see a brief explanation of each technique that will be measured. Later on Data preprocessing section you will find more details of how this techniques were applied on source datasets.

Count

The Count score is the most simple, it just measures the frequency each word appears on the text, converting raw text into a matrix with rows representing each document and columns each word. The value on the cell represents the number of times a word appears in a document.

TF-IDF

The Term Frequency - Inverse Document Frequency is more complex, it works giving each word the score as a relation between the term frequency over all documents and the inverse frequency of documents it appears. If a word appears a lot on a document, but this is true for a lot of other documents it will have a low score. If it appears more times in a document but does not appear on other documents it will have a great score.

The result of this operation is also a matrix in the same form, rows for documents and columns for words.

Hashing

Almost the same as the Count approach, but this one use a token for indexing instead the word itself to save computational resources.

PCA

Principal Component Analysis is an algorithm used to discover which set of features imply on a higher data variance, ranking, this way, the best features that can be used on a classifier model. It can be used to save computation resources on training and querying phase of one algorithm if needed. The input of this algorithm is a already processed matrix of input features and the result is a matrix with a possibly reduced column count, representing just the top features.

The matrix returning from any of those algorithms can be forwarded to any classification model.

Classifier algorithms

Multinomial Naive Bayes

This algorithm is a implementation of Naive Bayes for multinomially distributed data. This algorithm is known to work well with text documents. Even this is designed for receiving a vectorized word-count data it also works with TF-IDF matrices.

Naive Bayes is based on probabilities evaluation, it makes sense for text processing since calculating the probability that a text being classified as X if it contains the word B makes possible to calculate the probability of a text being X if is true that it contains B. The probability of a text being classified as X can be calculated using the facts for every word known by the model.

The main drawback of this algorithm is that it needs a considerable number of labeled sample data for training.

KNN

The K-Nearest Neighbour classifier works by finding elements that approximate their values, in this scenario it can make sense if two texts have the same words appearing with the same frequency, it's almost certain that they talk about the same thing.

KNN has the advantage of being very fast on training phase but is a bit slow on prediction phase.

SVM

Support Vector Machines classifies data separating their data points with vectors in some dimensions and calculating their distance to that vector. In order to discover all the different categories, the maximum gap between data points will be found, so points can be classified based on which side of the gap they fall.

This algorithm reduce the need for labeled training instances but is a bit slow on both, training and predicting phases.

Model selection algorithms

Cross-Validation

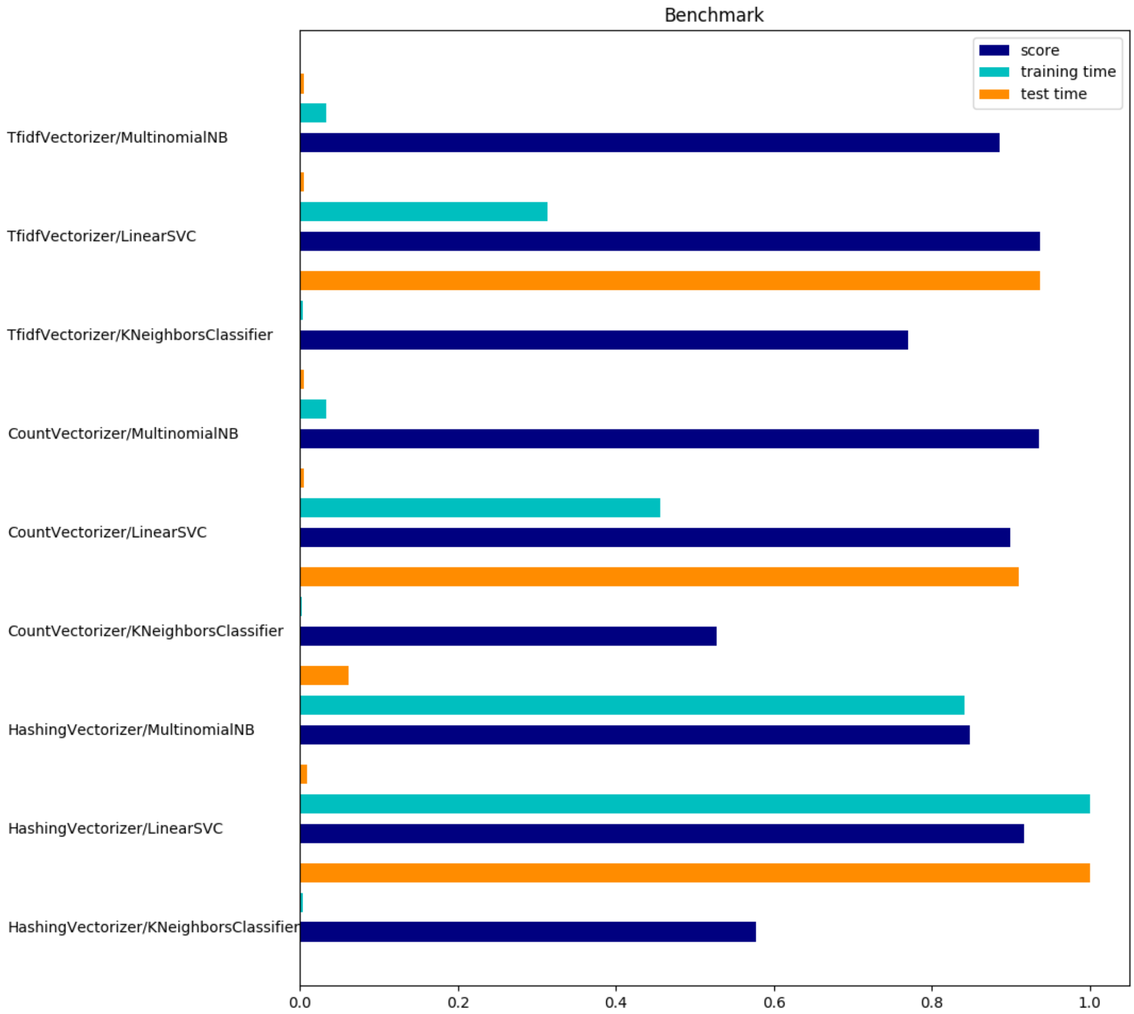
There is a technique called Cross-Validation, it is used to evaluate a specific model with different portions of train data, based on the K-fold system it splits data into K equal sized portions and use them to train and test the model sequentially. The final result will be the average result obtained in each train and test operation.

Scikit-learn has one implementation called `GridSearchCV` where the model is trained using Cross-Validation and different parameter combinations.

Benchmark

In this section a first checkout of how everything mentioned applies on our "simulated real world" is presented.

Using a script, combinations of the algorithms mentioned earlier were executed and a graph was created showing the results:



As predicted the KNN algorithm has the slowest testing time, the surprise is that it has the worst score. The best combination for a KNN model is the TF-IDF vectorizer, it has a good score but a very high test time.

The SVC algorithm has an impressive score, the greatest for Hashing and TF-IDF vectorizers, just staying behind Naive Bayes with the Count value, but in counter part has the slowest training time.

The Multinomial Naive Bayes algorithm performs very well. It has great scores and small training and test times for any combination.

This simple benchmark shows that in combination with any model being used the Hashing Vectorizer was not so good, so it will be discarded from here through the final.

Based on these results a minimum of 95% was established as target for Heimdall's accuracy. Thinking about a help desk department that receives 1000 emails a day, it means that 50 emails would be classified wrong. This is not a big problem it is still faster to re-classify or re-forward 50 emails per day than having to read 1000 emails twice, one for classification and one for solving the problem, without mentioning that a human classifier agent is not error free too.

III. Methodology

Data Preprocessing

As discussed before on Algorithms and Techniques section, for raw text classification there is always the need of some preprocessing step.

The first thing to notice is the time that preprocessing takes, about 4 to 5 seconds on our benchmark. In an attempt to improve the preprocessing time the number of features generated by all vectorizers was reduced to only 150 but that dropped the performance of all models below 50%.

To improve processed data a PCA algorithm was experimented choosing the best features with a proper time. The idea behind that was to let the vectorizers generate any features they can and let a PCA decide what features to use, but that was so slow we could not even measure.

When working with sparse matrices PCA is very slow because for every feature it needs to center all data points around. As matrices returned by a raw text vectorized are known to be very large PCA was taken off.

The NMF - Non-negative Matrix Factorization, an algorithm that can be used for dimensionality reduction, was also tested with slow results.

Another test with Truncated Singular Values - `TruncatedSVD` on scikit-learn - that performs a linear dimensionality reduction using truncated singular value decomposition where, contrary to PCA, there is no need to center data. Even that this algorithm was faster, it can return negative values what is invalid for working with Naive Bayes based algorithms and there was no significant gain with other classifiers, so this algorithm was discarded too.

Thinking better, there is no need to a second algorithm of dimensionality reduction, since the feature extraction algorithms being used can do that! For example, `CountVectorizer` has a parameter called `max_features` that limits the size of the result matrix to a maximum of `n` features. The features chosen will be the `n` top words that appear with more frequency, the same behaviour as expected with PCA.

Implementation

For implementation it was decided to focus only on one model, since availing all models take too many time. Because execution time is important considering that Heimdall will work with a large amount of data, Multinomial Naive Bayes was chosen along with a Count Vectorizer, since they work better together.

A EmailClassifier class was created to wrap all machine learning implementation. This makes easy to add new integrations with other data providers, like email tools as GMail or help desk systems as Zendesk.

Starting with the EmailClassifier class definition, there are two methods: `train` and `classify`.

The `train` method will receive historical data as raw email bodies and their respective labels, using this data to fit a MultinomialNB model that will be used later on incoming email classification.

Here is where GridSearchCV come in. A machine learning is almost unpredictable, it can perform well with a dataset and the same model can perform badly with another dataset. Heimdall is intended to use a lot of different datasets, at least one per user, so each dataset is tested with different configurations and the best one is chosen during the training phase.

The `classify` method will receive new data as raw email bodies and use the already trained model to classify each email.

A `train` operation can be slow, but it only needs to be executed once, after that Heimdall will persist the trained model using scikit-learn's builtin persistence library and for each request load the persisted model without having to train it again. Also, one could want to re-train an already working model in order to improve the performance of the model with items badly classified being fixed.

Two methods, `save_to_file` and `load_from_file` control the persistence feature. The trained vectorized is stored along with the classifier.

Unit tests were added to ensure Heimdall's performance over the 20 news groups dataset.

GMail integration was added so Heimdall is ready to test. If user was using Google's Inbox Application his emails are already classified in categories. Heimdall will use this data to learn. Authentication is done via OAuth 2.0 user's browser will be open on Google's authentication page to authorize Heimdall to open his messages.

Refinement

The first implementation version have used default classifier and vectorizer parameters. The performance was as seen on initial benchmark, a 92.3% accuracy.

By using cross-validation to choose the best classifier we have improved accuracy to over 93.98%. `alpha` parameter is being tested with 25%, 50%, 75% and 100% values. This parameter controls additive smoothness for data. For this example data preferred value seems to be 75% what implies that the smoothing algorithm was cutting off important data when setting it to 100%.

Other parameter used on cross validation was `fit_prior` that controls if the classifier should learn the classes prior from the train phase or not. If not learning from data, it will use a uniform distribution for classes priority. For this example data preferred value is to learn from data. It seems that a uniform distribution is not helpful in this scenario.

All the non-chosen parameters are being maintained on train phase because even these parameters are excellent for the 20 news group dataset it can vary for another data being used on future.

The next step was to refine vectorizer parameters. For Count Vectorizer there are two parameters being passed. The first one is `stop_words` . Stop words is a list of commonly used words on a determined language, this helps vectorizer build his vocabulary by cutting off and not calculating those words. Scikit-learn has an english pre built list bu as Heimdall is looking for a more internationalized fashion a `stop_words` parameter was created on `EmailClassifier` class and will be forwarded to Count Vectorizer.

Searching the internet for a Portuguese list of stop words the Ranks NL Webmaster Tools web page were found. There are a lot of stop words lists for different languages. The Portuguese list was downloaded and included on a model called `stop_words` along with the `models` module. This module was created as a stop words repository including currently three lists, `PORTUGUESE` , `ENGLISH` and `ANY` . `ENGLISH` is an alias for the scikit-learn's builtin list. `PORTUGUESE` is the downloaded list and `ANY` is a combination of both.

This way if input data is english or portuguese there is no need to worry about, we can use `ANY` stop words and the algorithm should perform well.

The second is `max_df`, this parameter controls output vocabulary to have only words with a document frequency lesser than given threshold, in other words it will return only words that appears in some documents. As we have seen before, using TF-IDF only relevant words can be chosen from a document. With Count vectorizer this information was lost, but using this parameter part of the TF-IDF algorithm can be simulated. For this example data the best value was 50%, words that appear in more than a half of documents are ignored.

These adjusts to vectorizer improved accuracy to above 94.44%.

IV. Results

Model Evaluation and Validation

Heimdall was validated using 20 news groups dataset and performed very well. How would it perform against unknown data? To answer this question, Heimdall was used to classify my own GMail messages. These messages are in both portuguese and english language and are already classified by Google in 5 different categories Promotions, Updates, Social, Personal and Forums. Unfortunately this data is private and could not be distributed along side with this program.

Just for the sake of testing it we have used a snippet of each message, GMail APIs already provides a way to get only a short but relevant part for each message instead of the entire message itself.

The `ANY` stop words options was used for those tests.

A 1100 messages sample was used to train and test Heimdall for the first time, his score was just about 50% of accuracy.

A second sample of 2098 messages was used and the score was up to 89%.

This confirms the previously shown advertisement about the number of samples. It seems that Heimdall will need to use a dataset with at least 2500 samples to have a great performance. The final validation result can be inspected bellow:

```
Train data size: 1679 samples
Test data size: 420 samples
Training
Done in 0.354 s at 0.883 MB/s
Testing
Done in 0.017 s at 4.512 MB/s
Accuracy 89.05%
```

Justification

A minimum accuracy of 95% was established for 20 news group dataset and the performance obtained from validation was of approximately 94.5%. The goal was not reached but this was an acceptable result, maybe with a bigger example dataset higher results would be reached.

For validation step the dataset have included portuguese and english messages and even that the model performed almost 90% of accuracy.

Considering this results Heimdall maybe classified as a good email classifier tool.

V. Conclusion

Free-Form Visualization

The model presented here is very simple, with very small training and querying times. With a little effort it can turn into a very good email classifying tool.

One thing that is notable is the adaptation to new data, even it was developed looking for clearly defined and well elaborated texts, availing it over a normal inbox, where the text content is unknown and unaddressed it shows a very good performance for a small sized sample.

Another thing to mention is the time performance. It is very fast to analyze large documents, about 800 KB/s for training and 4.5 MB/s for classifying.

Reflection

Heimdall works. It would be possible to use Heimdall as a triage automation on a Help Desk sector.

One thing that this project show us is that natural text processing is a bit hard. There is a need of a good preprocessing phase and well chosen training datasets for good results.

The Multinomial Naive Bayes was the great key to the victory here, a very good starting point for any project that involves natural text processing.

Improvement

As a improvement, we could try to apply a ensemble learning algorithm combining results from actual solution and a Linear SVC implementation for example. This can lead to a more robust solution with more precise results and maybe the 95% threshold would be elevated.

Aside from the learning part, the rest of the project should be improved, as obtaining the entire message as a training source when using GMail. New email providers or help desk systems integrations in order to put Heimdall on a production environment.

References

1. Scikit-learn documentation: http://scikit-learn.org/stable/index.html

2. Naive Bayes Classifier Wikipedia page: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

3. Stop words Wikipedia page: https://en.wikipedia.org/wiki/Stop_words

4. Additive smoothing Wikipedia page: https://en.wikipedia.org/wiki/Additive_smoothing

5. Ranks NL Webmaster Tools: http://www.ranks.nl/

6. GMail API Documentation: https://developers.google.com/gmail/