Spam E-Mail Detector

Submitted in partial fulfillment of the requirements of

Mini Project (CSP605)

for

Third Year of Computer Engineering

By

Ritika Lath	19101B0003
Mrudula Rothe	19101A0019
Dishant	19102A0003
Parth Yadav	19102A0022

Under the Guidance of

Prof. Sanjeev Dwivedi

Department of Computer Engineering

Accredited A+ by NAAC

Vidyalankar Institute of Technology
Wadala(E), Mumbai-400437

University of Mumbai

2019-20

CERTIFICATE OF APPROVAL

This is to certify that the project entitled

“Spam E-Mail Detector”

is a bonafide work of

Ritika Lath	**19101B0003**
Mrudula Rothe	**19101A0019**
Dishant	**19102A0003**
Parth Yadav	**19102A0022**

submitted to the University of Mumbai in partial fulfillment of

Mini Project (CSP605)

for

Third Year of Computer Engineering

Guide
Prof. Sanjeev Dwivedi

Head of Department

Principal

Mini Project Report Approval

This project report entitled ***Spam E-mail Detector*** by

1. **Ritika Lath** **19101B0003**
2. **Mrudula Rothe** **19101A0019**
3. **Dishant** **19102A0003**
4. **Parth Yadav** **19102A0022**

is approved for Mini Project (CSP605) for Third Year of Computer Engineering.

Internal Examiner External Examiner

Date:

Place:

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of student	Roll No.	Signature
1. Ritika Lath	19101B0003	
2. Mrudula Rothe	19101A0019	
3. Dishant	19102A0003	
4. Parth Yadav	19102A0022	

Date:

Place:

Acknowledgements

This mini project which was assigned to us had a great impact on all of us. It not only helped expand our knowledge but also gave us a glimpse so as to what it is like to work in a team, it taught us many new things for which we are very grateful. The completion of this project brought us immense joy and pleasure. We would like to thank **Prof. Sanjeev Dwivedi** our mentor cum project guide for this project, for giving us his valuable time, for guiding us, giving us his useful inputs via various consultations and helping us in the hour of the need.
We would also like to thank the entire **Computer Engineering Department** of **Vidyalankar Institute of Technology** for their cooperation and encouragement which motivated us to finish our project on time.

Abstract

In today's world, spam filtering is a must to protect your business. Spam is not going away. It is estimated that 70 percent of all email sent globally is spam, and the volume of spam continues to grow because spam remains a lucrative business.

Spammers get ever more sophisticated and creative in their tactics to get their messages into your inboxes and wreak their havoc. Spam filtering solutions must continually be updated to address this evolving threat.

In this project in the Machine Learning Domain we are given with emails in an inbox, we are going to identify those email messages that are spam and those that are not.

Mixed E-mails with Hams, Hard Hams, as well as Spams will be the dataset for our Machine Learning Model to be deployed in Python.

The given dataset will be processed and cleaned by removing punctuation marks, emoticons, and special symbols from it.

The cleaned dataset of e-mails will be then tokenized based on the nature of words used in it using Natural Language Processing.

Finally, the dataset will be divided into training and test sets. Since this is a basic Classification problem, the machine learning model will use the algorithms of Logistic Regression on the training dataset to classify the emails.

After the training is over, the model will be deployed with the Test Data Set of E-mails and will predict them as a Spam or a Non-Spam e-mail.

As a final step, we will plug-in this model with our Gmail Inboxes, using the Google Gmail API provided by Gmail itself, and create a new label named "My Spam Detector", which will store all the mails in the inbox identified as a spam by our own Spam Detector model.

Table of Contents

Introduction

In this project, in the Machine Learning Domain, we designed a Spam E-Mail Detector using the concepts of Machine Learning. It is an example of Supervised Learning.

Supervised learning provides you with a powerful tool to classify and process data using machine language. With supervised learning you use labeled data, which is a data set that has been classified, to infer a learning algorithm.

Spam email is unsolicited and unwanted junk email sent out in bulk to an indiscriminate recipient list. Typically, spam is sent for commercial purposes. It can be sent in massive volume by botnets, networks of infected computers.

Often, spam email is sent for commercial purposes. While some people view it as unethical, many businesses still use spam. The cost per email is incredibly low, and businesses can send out mass quantities consistently. Spam email can also be a malicious attempt to gain access to your computer. Spam email can be difficult to stop, as it can be sent from botnets. Botnets are a network of previously infected computers. As a result, the original spammer can be difficult to trace and stop.

If you receive a message that appears to be spam--for example, if you don’t recognize the sender--mark the message as spam in your email application. Don't click any links or attached files, including opt-out or unsubscribe links. Spammers sometimes include these links to confirm that your email address is legitimate, or the links may trigger malicious webpages or downloads.

Spam email can be dangerous. It can include malicious links that can infect your computer with malware. Do not click links in spam. Dangerous spam emails often sound urgent, so you feel the need to act. Keep reading to learn about some of the basic spam types.

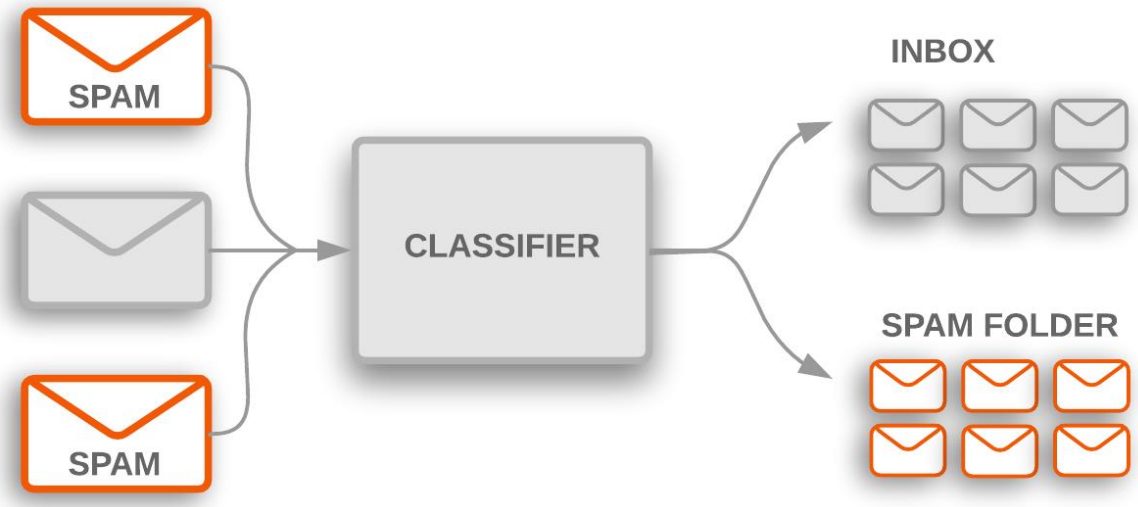
Some common types of Spams are:

1. **Commercial advertisements:** When businesses capture your email address, they often subscribe you to their newsletter by default, as a low-cost way to sell their products. Whenever you fill out an online form, look for a checkbox to opt into or out of marketing email.

2. **Antivirus warnings :** Ironically, antivirus warnings are a common spam tactic. These emails warn you about a computer virus infection and offer a solution--often an antivirus scan--to fix the alleged cyber threat. But taking the bait and clicking the link can grant the hacker access to your system or may download a malicious file.
3. **Email Spoofing**
4. **Sweepstakes Winners :** Spammers often send emails claiming that you have won a sweepstakes or a prize. They urge you to respond quickly to collect your prize, and may ask you to click a link or submit some personal information. If you don't recognize the competition, or if the email address seems dubious, don't click any links or reply with any personal details.
5. **Money Scams**

Problem Statement

Our problem statement is to use the techniques of **Machine Learning**, such as **Logistic Regressing** and the concepts of **basic sentiment analysis**, to predict the emails in our mail inbox as **Spam** or **Ham.**



Along with this, we have to use our trained ML model and **plug-in** it with our **Gmail** inboxes so that our created model can predict the mails as Spam and change their label from ‘Inbox’ to ‘My Spam Model’, a new label to be created using the **Gmail API** provided by Gmail itself.

The dataset required for the above task is already available on the internet, as mentioned in the referred resources.

Literature Survey

Concepts used

1. **Logistic Regression:**
   Logistic Regression is used when the dependent variable(target) is categorical.
   For example,
   i. To predict whether an email is spam (1) or (0)

   ii. Whether the tumour is malignant (1) or not (0)

   Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time.

   From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.

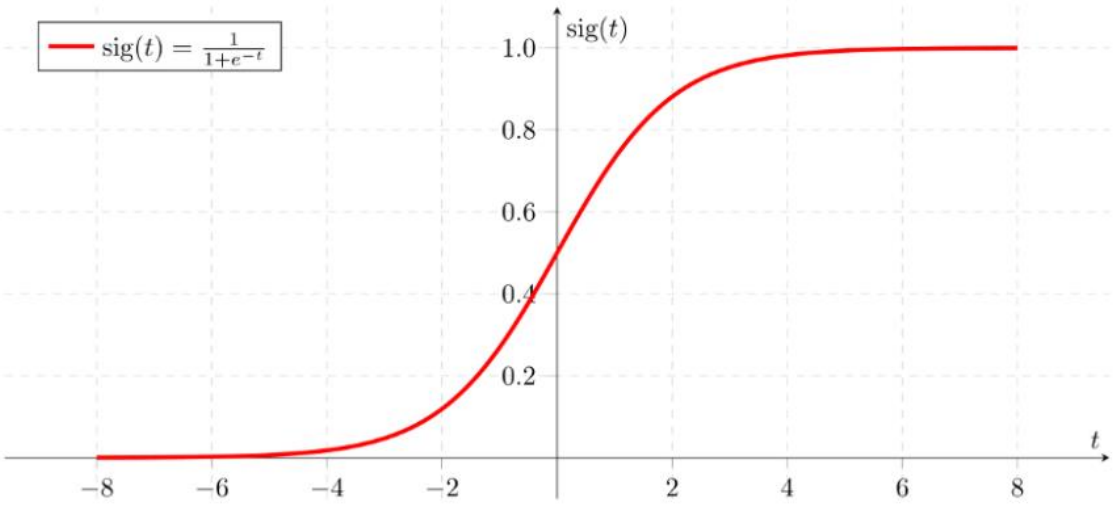
   Model

   Output = 0 or 1

   Hypothesis => Z = WX + B

   hΘ(x) = sigmoid (Z)

Sigmoid Function:



If ‘Z’ goes to infinity, Y(predicted) will become 1 and if ‘Z’ goes to negative infinity, Y(predicted) will become 0. The output from the hypothesis is the estimated probability. This is used to infer how confident can predicted value be actual value when given an input X.

Mathematically this can be written as,

$h_\Theta(x) = P(Y=1 \mid X; \text{theta})$

Probability that Y=1 given X which is parameterized by ‘theta’.

$P(Y=1 \mid X; \text{theta}) + P(Y=0 \mid X; \text{theta}) = 1$

$P(Y=0 \mid X; \text{theta}) = 1 - P(Y=1 \mid X; \text{theta})$

Decision Boundary

To predict which class a data belongs, a threshold can be set. Based upon this threshold, the obtained estimated probability is classified into classes.

Say, if predicted_value ≥ 0.5, then classify email as spam else as not spam.

Decision boundary can be linear or non-linear. Polynomial order can be increased to get complex decision boundary.

Simplified Cost Function

$\text{Cost}(h_\Theta(x), y) = -y \log(h_\Theta(x)) - (1-y) \log(1- h_\Theta(x))$

If y = 1, (1-y) term will become zero, therefore $-\log(h_\Theta(x))$ alone will be present

If y = 0, (y) term will become zero, therefore $-\log(1- h_\Theta(x))$ alone will be present

Gradient Descent

To minimize our cost, we use Gradient Descent

One of the neat properties of the sigmoid function is its derivative is easy to calculate. If you're curious, there is a good walk-through derivation on stack overflow. Michael Neilson also covers the topic in chapter 3 of his book.

$s'(z)=s(z)(1-s(z))s'(z)=s(z)(1-s(z))$

Which leads to an equally beautiful and convenient cost function derivative:

$C'=x(s(z)-y)$

We have used the **Turi Create** package and it's respective libraries.

Turi Create is a highly scalable machine learning library for Python, which also includes the SFrame, a highly-scalable library for data manipulation. A huge advantage of SFrame over Pandas is that with SFrame, you are not limited to datasets that fit in memory, which allows you to deal with large datasets, even on your laptop. And it has awesome data visualization functions too.

Turi Create offers a broad set of packaged application based toolkits as well as algorithms for model creation. Turi Create offer default parameters, building blocks and baseline models that help you get started quickly with their dataset without sacrificing the ability to go back and customize models later. Each incorporates automatic feature engineering and model selection.

We use Jupyter Notebook with Python 3 for further analysis.

Implementation:

The dataset contains a total of 8729 rows and 2 columns named 'text' and 'spam' which store the email text and classification value respectively, 1 for spam and 0 for ham.

Loading:

The first task is to load the dataset from the .csv file to a dataframe known as SFrame in Turi Create and display the SFrame created.

```
In [1]: import turicreate as tc
        sf=tc.load_sframe('final_data')
```

```
In [2]: sf
```

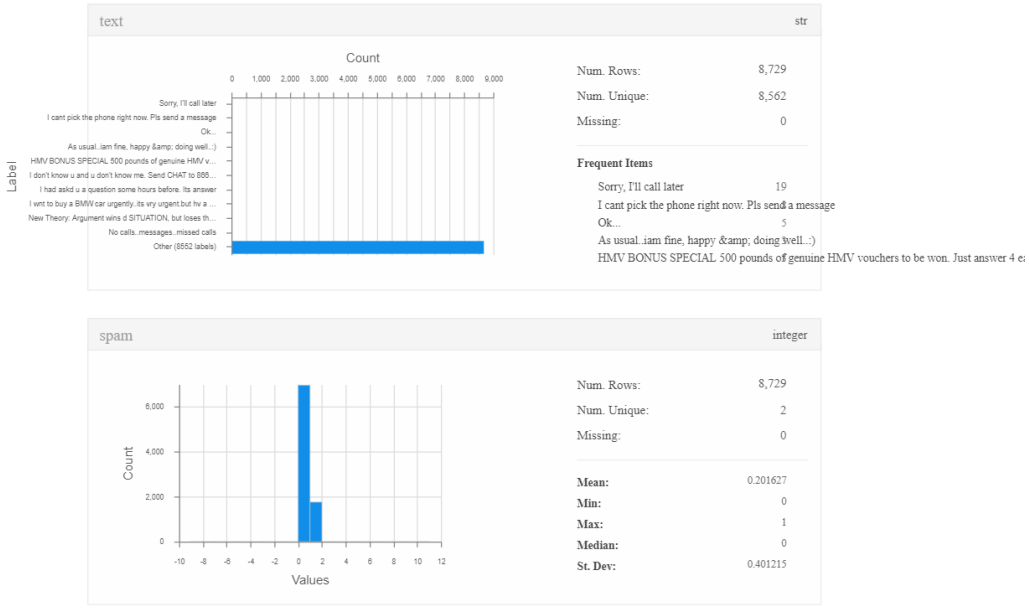
Out[2]:

text	spam
Subject: naturally irresistible your ...	1
Subject: the stock trading gunslinger f ...	1
Subject: unbelievable new homes made easy im ...	1
Subject: 4 color printing special request ...	1
Subject: do not have money , get software cds ...	1
Subject: great nnews hello , welcome to ...	1
Subject: here ' s a hot play in motion homeland ...	1
Subject: save your money buy getting this thing ...	1
Subject: undeliverable : home based business for ...	1
Subject: save your money buy getting this thing ...	1

[8729 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

```
In [3]: sf.show()
```

```
Materializing SFrame
```



We save this loaded SFrame as **'final_data'** named folder.

Cleaning:

The next task is to load this saved SFrame and clean the data and make it free from unwanted values. This simply can be done by checking the ‘spam’ column for any values other than 0 or 1.

Then we will use a pre-defined method in Turi Create to count the occurrence of each word in the given mail, or to Vectorize each mail, in a new column. This method will return a dictionary of words and count as Key-Value pairs.

```
import turicreate as tc
```

```
mail=tc.SFrame('./final_data')
```

```
mail.head()
```

text	spam
Subject: naturally irresistible your ...	1
Subject: the stock trading gunslinger f ...	1
Subject: unbelievable new homes made easy im ...	1
Subject: 4 color printing special request ...	1
Subject: do not have money , get software cds ...	1
Subject: great nnews hello , welcome to ...	1
Subject: here ' s a hot play in motion homeland ...	1
Subject: save your money buy getting this thing ...	1
Subject: undeliverable : home based business for ...	1
Subject: save your money buy getting this thing ...	1

[10 rows x 2 columns]

```
print(mail[(mail['spam']!=1)&(mail['spam']!=0)])
```

```
+------+------+------------+
| text | spam | word_count |
+------+------+------------+
+------+------+------------+
[? rows x 3 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.
```

Here, we check the occurrence of any other value than 0 or 1, but since number of rows with any other value is 0, we see that our data is already clean.

We move forward towards vectorizing the mail. This is a simple process and we do not have to follow a complicated procedure as we have to while using Pandas.

```
mail['word_count']=tc.text_analytics.count_words(mail['text'])
```

```
mail
```

text	spam	word_count
Subject: naturally irresistible your ...	1	{'interested': 1.0, 'have': 1.0, 'this': ...
Subject: the stock trading gunslinger f ...	1	{'albeit': 1.0, 'diffusion': 1.0, ...
Subject: unbelievable new homes made easy im ...	1	{'pittman': 1.0, 'foward': 1.0, 'form': ...
Subject: 4 color printing special request ...	1	{'and': 1.0, 'color': 1.0, '338': 2.0, ...
Subject: do not have money , get software cds ...	1	{'death': 1.0, 'by': 2.0, 'd': 1.0, 'finish': 1.0, ...
Subject: great nnews hello , welcome to ...	1	{'day': 1.0, 'devitalize': 1.0, ...
Subject: here ' s a hot play in motion homeland ...	1	{'guarantee': 1.0, 'sources': 1.0, 'was': ...
Subject: save your money buy getting this thing ...	1	{'get': 1.0, 'aicohol': 1.0, 'with': 1.0, 'mix': ...

Then, we finally save this latest SFrame, with an extra column **'word_count'** by the name of **'cleaned_dataset'** to proceed with the remaining tasks.

```
mail.save('./cleaned_dataset')
```

Training:

We load the previously saved **'cleaned_dataset'** to divide it into 2 SFrames, namely **'train_set'** and **'test_set'**, by using a **random_split()** method in Turi Create and dividing it in the 9:1 ratio.

```
import turicreate as tc
```

```
data=tc.SFrame('./cleaned_dataset')
```

```
train_set,test_set=data.random_split(0.9,seed=0)
```

Now, it is the time to train our **'spam_model'** using in-built methods of logistic regression classification in Turi Create. We will provide **'words_count'** as a feature to the classifier to perform gradient descent to minimize the cost function, and calculate the weight assigned to each word in a particular mail. After this, we also evaluate our predicted 'train-set' to get the accuracy and confusion matrix.

```
spam_model=tc.logistic_classifier.create(train_set,target='spam',features=['word_count'])
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
          You can set ``validation_set=None`` to disable validation tracking.


Logistic regression:

--------------------------------------------------------

Number of examples          : 7454

Number of classes           : 2

Number of feature columns   : 1

Number of unpacked features : 37373

Number of coefficients      : 37374

Starting L-BFGS

--------------------------------------------------------

+-----------+----------+-----------+--------------+-------------------+---------------------+
| Iteration | Passes   | Step size | Elapsed Time | Training Accuracy | Validation Accuracy |
+-----------+----------+-----------+--------------+-------------------+---------------------+
| 0         | 3        | 0.500000  | 1.531702     | 0.904213          | 0.857506            |
| 1         | 5        | 0.500000  | 1.707448     | 0.964046          | 0.890585            |
| 2         | 6        | 0.625000  | 1.779274     | 0.989938          | 0.926209            |
| 3         | 7        | 0.781250  | 1.876616     | 0.994902          | 0.938931            |
| 4         | 8        | 0.976563  | 1.992979     | 0.997317          | 0.954198            |
| 9         | 14       | 1.000000  | 2.461433     | 1.000000          | 0.956743            |
+-----------+----------+-----------+--------------+-------------------+---------------------+
```

```
spam_model.coefficients
```

name	index	class	value	stderr
(intercept)	None	1	-2.1570920972235976	None
word_count	interested	1	-0.01617892508503141	None
word_count	have	1	-0.04094998969196886	None
word_count	this	1	0.011830950093076989	None
word_count	love	1	-0.5458099432614135	None
word_count	surethat	1	0.4127587061091441	None
word_count	no	1	-0.07230349965616438	None
word_count	with	1	-0.020396387902366055	None
word_count	provide	1	0.026410417567550443	None
word_count	extra	1	0.32340118080231656	None

[37374 rows x 5 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

```
spam_model.evaluate(train_set)
```

```
{'accuracy': 0.9978335669682681,
 'auc': 0.9993282858482584,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

 Rows: 4

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |      0       |        0        |  6257 |
 |      1       |        0        |   16  |
 |      1       |        1        |  1573 |
 |      0       |        1        |   1   |
 +--------------+-----------------+-------+
 [4 rows x 3 columns],
 'f1_score': 0.9946253556749921,
 'log_loss': 0.010483588963812433,
 'precision': 0.9993646759847522,
 'recall': 0.9899307740717432,
 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int
```

We have achieved a pretty good accuracy of 99.78% on the training set. Now, we save this trained model as **'final_trained_mode'** for further use in prediction of values in test_set.

```
spam_model.save('final_prediction_model')
```

Predicting:

We now load the saved 'test_set' SFrame as well as the trained classifier model 'final_prediction_mode'.

```
d_set = tc.SFrame('./test_set')
```

```
spam_model=tc.load_model('final_prediction_model')
```

Again, we will use the in-built **predict()** method in Turi Create to store the predicted results in a new column **'predicted_results'** in the same SFrame 'test_set'. Output-type will be set to probability as this will give the probability of each mail being closer to a spam or a ham mail. If **probability>=0.5**, it is a **Spam**, and vice versa.

```
d_set['predicted_results']=spam_model.predict(d_set,output_type='probability')
```

```
d_set
```

text	spam	word_count	predicted_results
Subject: brighten those teeth get your teeth ...	1	{'then': 1.0, 'via': 1.0, 'promotions': 1.0, ...	0.9917837483731741
Subject: search engine position be the very ...	1	{'com': 1.0, 'speedy': 1.0, 'line': 1.0, ...	0.9999788227344301
Subject: want to accept credit cards ? 126432211 ...	1	{'it': 1.0, 'cecks': 1.0, 'do': 1.0, 'subject': ...	0.08450006536858909
Subject: localized software , all languages ...	1	{'reqards': 1.0, 'best': 1.0, 'ianguaqe': 1.0, ...	0.999995646761379
Subject: select small - cap for astute investors ...	1	{'reliable': 1.0, 'including': 1.0, ...	1.0
Subject: localized software , all languages ...	1	{'reqards': 1.0, 'best': 1.0, 'lanquage': 1.0, ...	0.9999825878367608
Subject: i know your company ! It is really ...	1	{'interested': 1.0, 'have': 1.0, 'this': ...	0.999970237696562
Subject: [ilug] re : popular . biz and . com ...	1	{'listmaster': 1.0, 'maintainer': 1.0, ...	0.999999988253215
Subject: enhance your anatomy the longz sy ...	1	{'all': 1.0, 'yes': 1.0, 'much': 1.0, 'remarked': ...	0.9999998818593003
Subject: learn to play texas hold ' em and o ...	1	{'jybwgyay': 1.0, 'here': 1.0, 'and': 1.0, 'oth ...	0.9866798365489347

[579 rows x 4 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

```
spam_model.evaluate(d_set)
```

```
{'accuracy': 0.9620034542314335,
 'auc': 0.9836053628679685,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

 Rows: 4

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |      0       |        0        |  437  |
 |      1       |        0        |   21  |
 |      1       |        1        |  120  |
 |      0       |        1        |   1   |
 +--------------+-----------------+-------+
 [4 rows x 3 columns],
 'f1_score': 0.916030534351145,
 'log_loss': 0.18196168037077015,
 'precision': 0.9917355371900827,
 'recall': 0.851063829787234,
 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int
```

We also evaluated the predicted values found in test_set and achieved an **accuracy of 96.2%** and the **Confusion Matrix**, as shown above.

The Final Combined Code is as showed in the following screenshots.

Mini Project Group M26

Email Spam Detection

Load Email Data from given .csv file

In [1]:
```
import turicreate as tc
data=tc.load_sframe('final_data')
```

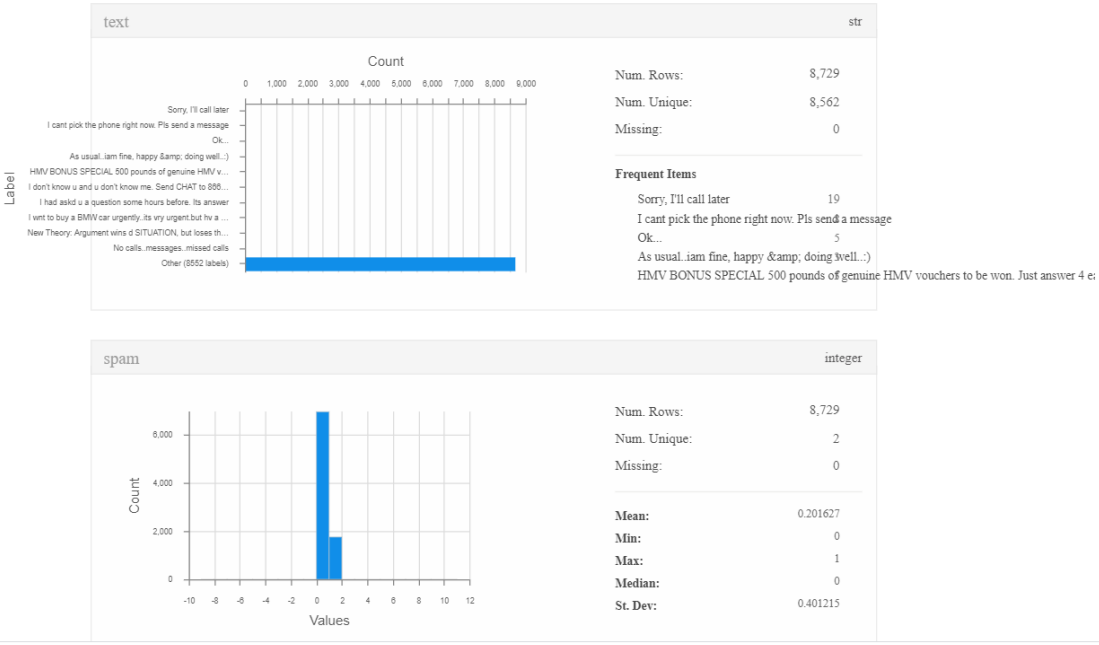
In [2]:
```
data
```

Out[2]:

text	spam
Subject: naturally irresistible your ...	1
Subject: the stock trading gunslinger f ...	1
Subject: unbelievable new homes made easy im ...	1
Subject: 4 color printing special request ...	1
Subject: do not have money , get software cds ...	1
Subject: great nnews hello , welcome to ...	1
Subject: here ' s a hot play in motion homeland ...	1
Subject: save your money buy getting this thing ...	1
Subject: undeliverable : home based business for ...	1
Subject: save your money buy getting this thing ...	1

In [3]:
```
data.show()
```

```
Materializing SFrame
```



In [4]:
```
print(data.num_rows,data.num_columns)
```

```
<bound method SFrame.num_rows of Columns:
        text    str
        spam    int

Rows: 8729

Data:
+-------------------------------+------+
|              text             | spam |
+-------------------------------+------+
| Subject: naturally irresis... |  1   |
| Subject: the stock trading... |  1   |
| Subject: unbelievable new ... |  1   |
| Subject: 4 color printing ... |  1   |
| Subject: do not have money... |  1   |
| Subject: great nnews  hell... |  1   |
| Subject: here ' s a hot pl... |  1   |
| Subject: save your money b... |  1   |
| Subject: undeliverable : h... |  1   |
| Subject: save your money b... |  1   |
+-------------------------------+------+
[8729 rows x 2 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.> <bound method SFrame.num_columns of Columns:
        text    str
        spam    int

Rows: 8729

Data:
+-------------------------------+------+
|              text             | spam |
+-------------------------------+------+
| Subject: naturally irresis... |  1   |
| Subject: the stock trading... |  1   |
| Subject: unbelievable new ... |  1   |
| Subject: 4 color printing ... |  1   |
| Subject: do not have money... |  1   |
| Subject: great nnews  hell... |  1   |
```

Check for Entries other than 0 and 1 in 'spam' column

In [5]:
```
print(data[(data['spam']!=1)&(data['spam']!=0)])
```

```
+------+------+
| text | spam |
+------+------+
+------+------+
[? rows x 2 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.
```

Perform word counting on 'text' column to get dictionary with count of each individual word

In [6]:
```
data['word_count']=tc.text_analytics.count_words(data['text'])
```

In [7]:
```
data
```

Out[7]:

text	spam	word_count
Subject: naturally irresistible your ...	1	{'interested': 1.0, 'have': 1.0, 'this': ...
Subject: the stock trading gunslinger f ...	1	{'albeit': 1.0, 'diffusion': 1.0, ...
Subject: unbelievable new homes made easy im ...	1	{'pittman': 1.0, 'foward': 1.0, 'form': ...
Subject: 4 color printing special request ...	1	{'and': 1.0, 'color': 1.0, '338': 2.0, ...
Subject: do not have money , get software cds ...	1	{'death': 1.0, 'by': 2.0, 'd': 1.0, 'finish': 1.0, ...
Subject: great nnews hello , welcome to ...	1	{'day': 1.0, 'devitalize': 1.0, ...
Subject: here ' s a hot play in motion homeland ...	1	{'guarantee': 1.0, 'sources': 1.0, 'was': ...
Subject: save your money	1	{'get': 1.0, 'aicohol':

In [8]:
```
data.tail()
```

Out[8]:

text	spam	word_count
I am back. Good journey! Let me know if you need ...	0	{'pendent': 1.0, 'shall': 1.0, 'receipts': 1.0, ...
So that takes away some money worries ...	0	{'worries': 1.0, 'money': 1.0, 'some': 1.0, 'aw ...
aight we can pick some up, you open before ...	0	{'tonight': 1.0, 'open': 1.0, 'before': 1.0, ...
Latest News! Police station toilet stolen, ...	1	{'on': 1.0, 'go': 1.0, 'news': 1.0, 'latest': ...
Sac needs to carry on:)	0	{'carry': 1.0, 'to': 1.0, 'on': 1.0, 'needs': 1.0, ...
Just sing HU. I think its also important to find ...	0	{'girls': 1.0, 'ask': 1.0, 'doubt': 1.0, ...
What???? Hello wats talks email address? ...	0	{'address': 1.0, 'hello': 1.0, 'email': 1.0, ...
Except theres a chick with huge boobs. ...	0	{'boobs': 1.0, 'huge': 1.0, 'with': 1.0, ...
Im just wondering what your doing right now? ...	0	{'now': 1.0, 'right': 1.0, 'your': 1.0, 'wh ...
Wishing you a beautiful day. Each moment ...	0	{'it': 1.0, 'enjoy': 1.0, 'smiling': 1.0, 'keep': ...

[10 rows x 3 columns]

Divide dataframe into train and test sets

In [9]:
```
train_set,test_set=data.random_split(0.9,seed=0)
```

Train Binary Classification Model

In [12]:
```
spam_model=tc.logistic_classifier.create(train_set,target='spam',features=['word_count'],validation_set=test_set)
```

```
Logistic regression:
```

Train Binary Classification Model

In [12]:
```
spam_model=tc.logistic_classifier.create(train_set,target='spam',features=['word_count'],validation_set=test_set)
```

```
Logistic regression:

--------------------------------------------------------

Number of examples          : 7847

Number of classes           : 2

Number of feature columns   : 1

Number of unpacked features : 38205

Number of coefficients      : 38206

Starting L-BFGS

--------------------------------------------------------

+-----------+----------+-----------+--------------+-------------------+---------------------+

| Iteration | Passes   | Step size | Elapsed Time | Training Accuracy | Validation Accuracy |

+-----------+----------+-----------+--------------+-------------------+---------------------+

| 0         | 3        | 0.500000  | 0.165448     | 0.901873          | 0.880952            |

| 1         | 5        | 0.500000  | 0.319494     | 0.962406          | 0.908163            |

| 2         | 6        | 0.625000  | 0.412599     | 0.990060          | 0.930839            |

| 3         | 7        | 0.781250  | 0.491008     | 0.995030          | 0.943311            |

| 4         | 8        | 0.976563  | 0.572843     | 0.997451          | 0.955782            |

| 9         | 14       | 1.000000  | 0.875811     | 1.000000          | 0.964853            |

+-----------+----------+-----------+--------------+-------------------+---------------------+
```

Analyse and Evaluate the trained model

In [13]:
```
spam_model.coefficients
```

Out[13]:

name	index	class	value	stderr
(intercept)	None	1	-2.1551537221794774	None
word_count	interested	1	-0.03977551022407681	None
word_count	have	1	-0.04439832885817555	None
word_count	this	1	0.011698978159265388	None
word_count	love	1	-0.590812722642387	None
word_count	surethat	1	0.401799523962274	None
word_count	no	1	-0.06322616653695905	None
word_count	with	1	-0.020093544584771326	None
word_count	provide	1	0.024721268929507313	None
word_count	extra	1	0.34067959740219245	None

[38206 rows x 5 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

In [14]:
```
spam_model.evaluate(train_set)
```

Out[14]:
```
{'accuracy': 1.0,
 'auc': 1.0000000000000058,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

 Rows: 2

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |      0       |        0        |  6258 |
```

```
Data:
+--------------+-----------------+-------+
| target_label | predicted_label | count |
+--------------+-----------------+-------+
|      0       |        0        |  6258 |
|      1       |        1        |  1589 |
+--------------+-----------------+-------+
[2 rows x 3 columns],
'f1_score': 1.0,
'log_loss': 0.0027570261631529058,
'precision': 1.0,
'recall': 1.0,
'roc_curve': Columns:
       threshold       float
       fpr     float
       tpr     float
       p       int
       n       int

Rows: 100001

Data:
+-----------+---------------------+-----+------+------+
| threshold |         fpr         | tpr |  p   |  n   |
+-----------+---------------------+-----+------+------+
|    0.0    |         1.0         | 1.0 | 1589 | 6258 |
|   1e-05   |  0.5583253435602429 | 1.0 | 1589 | 6258 |
|   2e-05   |  0.5314797059763503 | 1.0 | 1589 | 6258 |
|   3e-05   |  0.5150207734100352 | 1.0 | 1589 | 6258 |
|   4e-05   |  0.5031959092361777 | 1.0 | 1589 | 6258 |
|   5e-05   | 0.49312879514221797 | 1.0 | 1589 | 6258 |
|   6e-05   | 0.48386065835730263 | 1.0 | 1589 | 6258 |
|   7e-05   | 0.47571108980504956 | 1.0 | 1589 | 6258 |
|   8e-05   | 0.46836049856184087 | 1.0 | 1589 | 6258 |
|   9e-05   | 0.46308724832214765 | 1.0 | 1589 | 6258 |
+-----------+---------------------+-----+------+------+
[100001 rows x 5 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.}
```

Predict emails in test set as spam(>0.5) or ham(<=0.5)

In [15]:
```
test_set['predicted_results']=spam_model.predict(test_set,output_type='probability')
```

In [16]:
```
test_set
```

Out[16]:

text	spam	word_count	predicted_results
Subject: brighten those teeth get your teeth ...	1	{'then': 1.0, 'via': 1.0, 'promotions': 1.0, ...	0.9917837483731741
Subject: search engine position be the very ...	1	{'com': 1.0, 'speedy': 1.0, 'line': 1.0, ...	0.9999788227344301
Subject: want to accept credit cards ? 126432211 ...	1	{'it': 1.0, 'cecks': 1.0, 'do': 1.0, 'subject': ...	0.08450006536858909
Subject: localized software , all languages ...	1	{'reqards': 1.0, 'best': 1.0, 'ianguaqe': 1.0, ...	0.999995646761379
Subject: select small - cap for astute investors ...	1	{'reliable': 1.0, 'including': 1.0, ...	1.0
Subject: localized software , all languages ...	1	{'reqards': 1.0, 'best': 1.0, 'lanquage': 1.0, ...	0.9999825878367608
Subject: i know your company ! It is really ...	1	{'interested': 1.0, 'have': 1.0, 'this': ...	0.999970237696562
Subject: [ilug] re : popular . biz and . com ...	1	{'listmaster': 1.0, 'maintainer': 1.0, ...	0.999999988253215
Subject: enhance your anatomy the longz sy ...	1	{'all': 1.0, 'yes': 1.0, 'much': 1.0, 'remarked': ...	0.9999998818593003
Subject: learn to play texas hold ' em and o ...	1	{'jybwgyay': 1.0, 'here': 1.0, 'and': 1.0, 'oth ...	0.9866798365489347

[882 rows x 4 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

Evaluate Predictions for Accuracy and Confusion Matrix

In [17]:
```
spam_model.evaluate(test_set)
```

Out[17]:
```
{'accuracy': 0.9648526077097506,
 'auc': 0.9743792204374032,
 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int

Rows: 4

Data:
+--------------+-----------------+-------+
| target_label | predicted_label | count |
+--------------+-----------------+-------+
|      0       |        0        |  709  |
|      1       |        0        |   29  |
|      1       |        1        |  142  |
|      0       |        1        |   2   |
+--------------+-----------------+-------+
[4 rows x 3 columns],
 'f1_score': 0.9015873015873015,
 'log_loss': 0.14873920041723596,
 'precision': 0.9861111111111112,
 'recall': 0.8304093567251462,
 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int

Rows: 100001

Data:
+-----------+--------------------+--------------------+-----+-----+
| threshold |        fpr         |        tpr         |  p  |  n  |
+-----------+--------------------+--------------------+-----+-----+
|    0.0    |        1.0         |        1.0         | 171 | 711 |
```

Again use trained model for predictions on train_set

In [18]:
```
train_set['predicted_results']=spam_model.predict(train_set,output_type='probability')
```

In [19]:
```
train_set
```

Out[19]:

text	spam	word_count	predicted_results
Subject: naturally irresistible your ...	1	{'interested': 1.0, 'have': 1.0, 'this': ...	0.9999963138220579
Subject: the stock trading gunslinger f ...	1	{'albeit': 1.0, 'diffusion': 1.0, ...	0.9999250270162908
Subject: unbelievable new homes made easy im ...	1	{'pittman': 1.0, 'foward': 1.0, 'form': ...	0.9995107355997614
Subject: 4 color printing special request ...	1	{'and': 1.0, 'color': 1.0, '338': 2.0, ...	0.9999248590816272
Subject: do not have money , get software cds ...	1	{'death': 1.0, 'by': 2.0, 'd': 1.0, 'finish': 1.0, ...	0.9961222832690997
Subject: great nnews hello , welcome to ...	1	{'day': 1.0, 'devitalize': 1.0, ...	0.9999856254871661
Subject: here ' s a hot play in motion homeland ...	1	{'guarantee': 1.0, 'sources': 1.0, 'was': ...	1.0
Subject: save your money buy getting this thing ...	1	{'get': 1.0, 'aicohol': 1.0, 'with': 1.0, 'mix': ...	0.9992442567520703
Subject: undeliverable : home based business for ...	1	{'unknown': 1.0, '6': 1.0, 'co': 1.0, '7059': ...	0.9997760324174951
Subject: save your money buy getting this thing ...	1	{'get': 1.0, 'with': 1.0, 'mix': 1.0, 'minutes': ...	0.9994950996417747

[7847 rows x 4 columns]

Note: Only the head of the SFrame is printed.

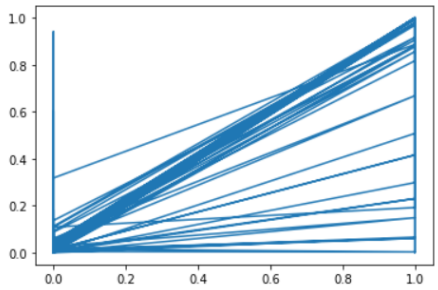
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

Graphically analyse relation b/w given and predicted results

In [20]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

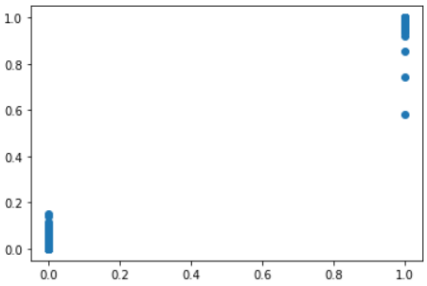
In [21]:
```
plt.plot(test_set['spam'],test_set['predicted_results'],'-')
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x7f4a2a39d0b8>]
```



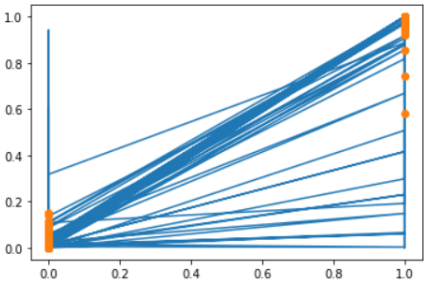
In [22]:
```
plt.plot(train_set['spam'],train_set['predicted_results'],'o')
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x7f4a2a3b7e48>]
```



In [23]:
```
plt.plot(test_set['spam'],test_set['predicted_results'],'-',train_set['spam'],train_set['predicted_results'],'o')
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x7f4a2a29d198>,
 <matplotlib.lines.Line2D at 0x7f4a2a29d208>]
```



The last 3 graphs show the relationship between the provided Spam value and the predicted probability of test_set, train_set, and both taken together in single graph respectively.

We see that in the graph of test_set, the graph forms various straight lines as the probability covers the entire range from 0-1.

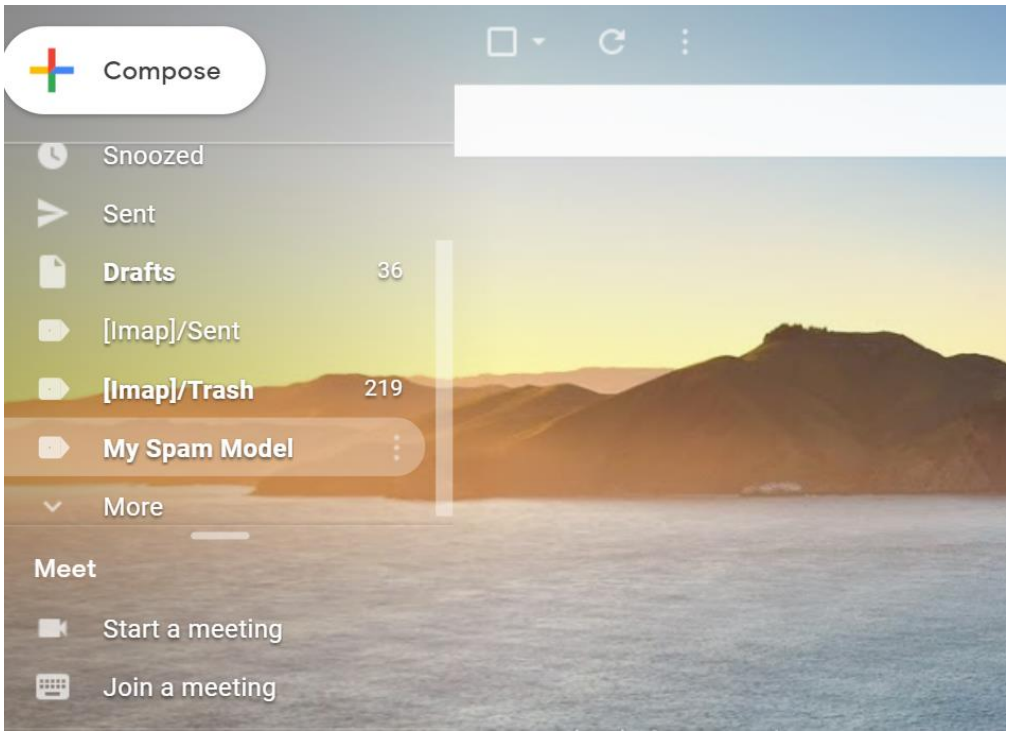
But in train_set's graph, we see only discrete values since the accuracy obtained for this was 1, i.e., all the predictions matched the given values with a probability of 1.

Now, our model is ready to be plugged-in with Gmail, but before that we will create a new label in our Gmail UI, referring the given Gmail API documentation by Gmail.

```python
from __future__ import print_function
import pickle
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request


SCOPES = ['https://www.googleapis.com/auth/gmail.readonly','https://www.googleapis.com/auth/gmail.labels','https://www.googleapis.com/auth/gmail.modify']

def new_label(name,mlv="show",llv="labelShow"):
    label={
        "name":name,
        "messageListVisibility":mlv,
        "labelListVisibility":llv
        }
    return label

def main():

    creds = None

    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)

    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)

    service = build('gmail', 'v1', credentials=creds)
    created_label=new_label(name='My Spam Model')
```

```python
    try:
        create_new=service.users().labels().create(userId='me',body=created_label).execute()
        print("Label Created")
    except Exception as e:
        print("Error occured : {e}")



if __name__ == '__main__':
    main()
```

This will create a new label named ‘My Spam Model’ in our Gmail UI, as shown ahead.



The following screenshots show the code to plug-in the model with Gmail account.

Plug-in ML Model with Gmail

In [1]:
```
import turicreate as tc
```

In [2]:
```
from __future__ import print_function
import pickle
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
```

In [3]:
```
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly','https://www.googleapis.com/auth/gmail.modify']
```

Load Saved Prediction Model

In [4]:
```
spam_model=tc.load_model('final_prediction_model')
```

In [5]:
```
def spam(ids,addLabel='Label_3',removeLabel='INBOX'):
    batch_modify={
        "ids":[ids],
        "addLabelIds":[addLabel],
        "removeLabelIds":[removeLabel]
    }
    return batch_modify
```

In [6]:
```
def predict(text):
    data=tc.load_sframe([text])
    data['word_count']=tc.text_analytics.count_words(data['X1'])
    p=spam_model.predict(data,output_type='probability')
    del(data)
    return p[0]
```

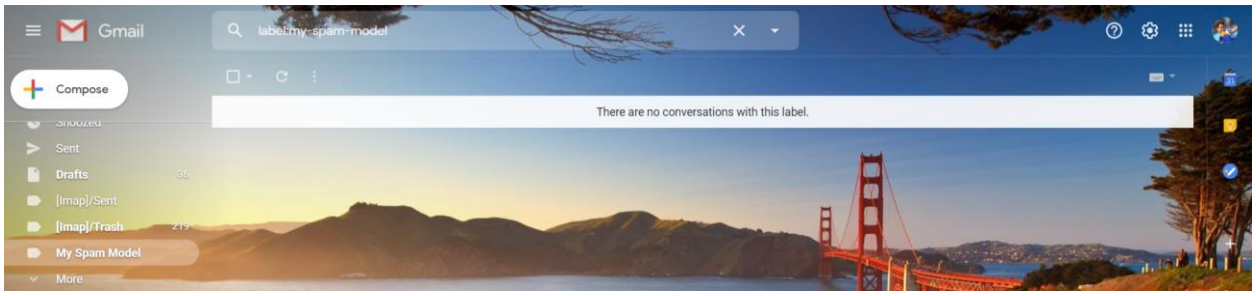
Reading Mails from Inbox and Labeling them as Spam after Prediction

In [7]:
```
def main():

    creds = None
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)

    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)

    service = build('gmail', 'v1', credentials=creds)

    try:
        results=service.users().messages().list(userId='me',maxResults=50,labelIds=['INBOX']).execute()
        messages=results.get('messages',[])
        for message in messages:
            msg=service.users().messages().get(userId='me',id=message['id']).execute()
            text=msg['snippet']
            if predict(text)>0.5:
                new_label=spam(message['id'])
                new_batch=service.users().messages().batchModify(userId='me',body=new_label).execute()
    except Exception as e:
        print("Error occured : {e}")
```

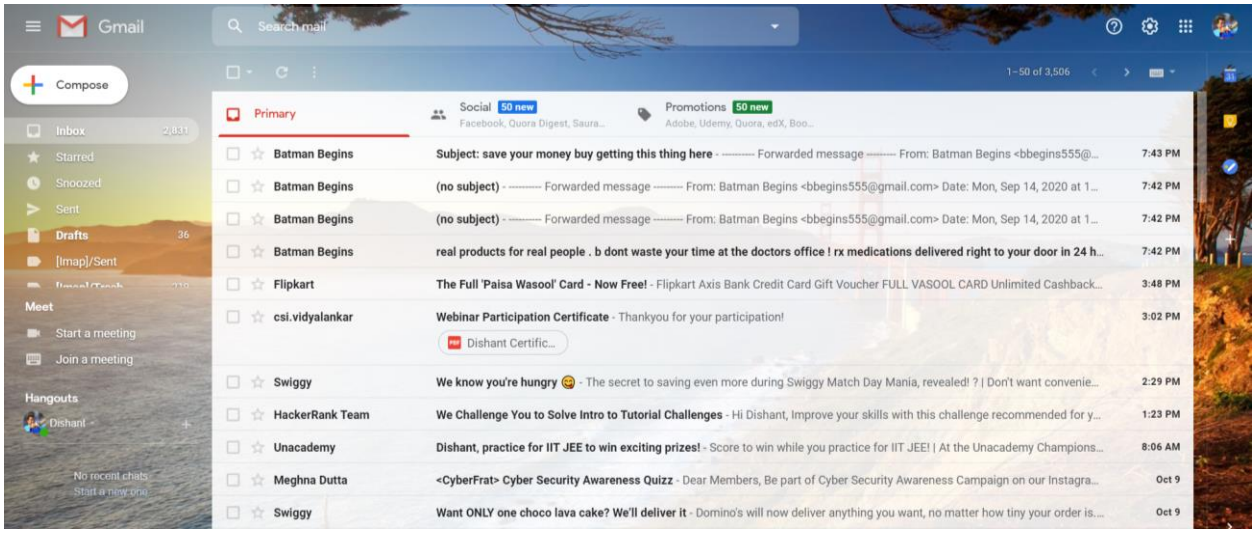
In [8]:
```
main()
print('Detection complete')
```

```
Detection complete
```

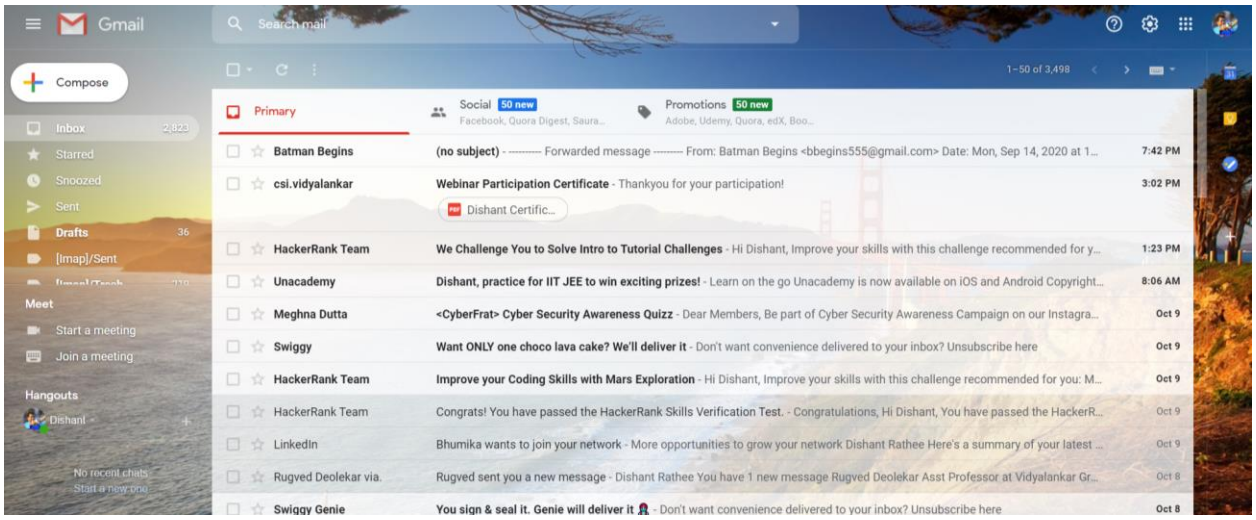
Currently, there are no spam mails in our own label ‘My Spam Model’.



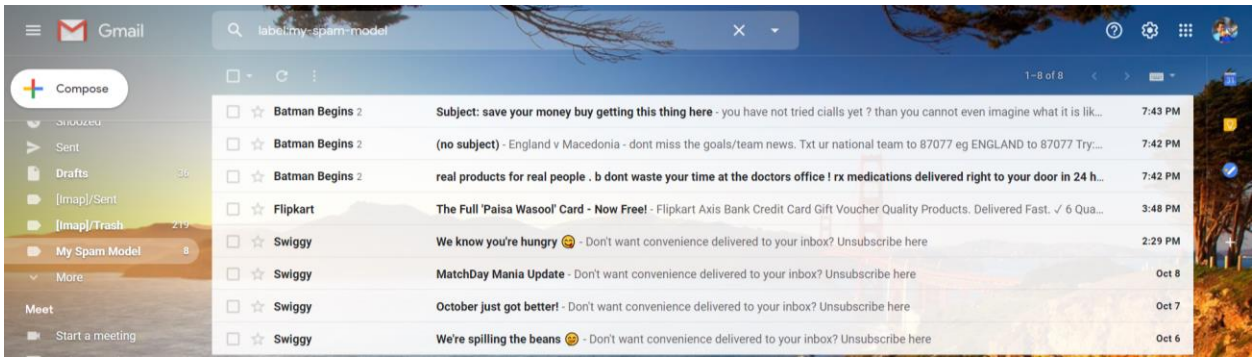
And we have sent some random emails consisting of both ham and spam mails from a random email id, which are visible in our inbox, on the next page.



Now, we executed our plug-in model and got the following results.



Our plug-in model checked on 50 recent emails, and labelled them as spam or ham. It is clearly visible that some spam mails sent by ‘Batman Begins’ are now not there in the Inbox, as they have been transferred to ‘My Spam Model’ with some other mails too, as shown on the next page.



Hence, our Machine Learning Model from Spam Email Detection is working and showing results with it’s plug-in with Gmail.

Conclusion

Our project titled ‘Spam Email Detector’ has successfully been designed and tested on Gmail Inbox mails. We were able to learn and apply various machine learning concepts of Logistic Regression and basic sentiment analysis through this project and predict various emails as Spam or Ham depending on their content.

We also built and were able to inculcate various team-skills in ourselves by taking this project towards success. We also thank our guide Prof. Sanjeev Dwivedi, to trust us and guide us through this interesting project.

Certifications

1. Ritika Lath 19101B0003
   https://www.coursera.org/account/accomplishments/verify/RKUBNTGSNFPH?utm_source=link&utm_medium=certificate&utm_content=cert_image&utm_campaign=pdf_header_button&utm_product=course
2. Mrudula Rothe 19101A0019

   https://www.coursera.org/account/accomplishments/records/4ZAJHQBMEY8H?utm_source=android&utm_medium=certificate&utm_content=cert_image&utm_campaign=sharing_cta&utm_product=course

3. Parth Yadav 19102A0022
   https://www.coursera.org/account/accomplishments/verify/N6DPYMSXHCAA?utm_source=ios&utm_medium=certificate&utm_content=cert_image&utm_campaign=sharing_cta&utm_product=course
4. Dishant 19102A0003
   https://coursera.org/share/6138d5c9d3fe399eb70375a362ce634f

References

1. https://www.cisco.com/c/en/us/products/security/email-security/what-is-spam.html
2. https://www.kaggle.com/veleon/ham-and-spam-dataset?
3. https://www.google.co.in/url?sa=i&url=https%3A%2F%2Fmedium.com%2F%40naveeen.kumar.k%2Fnaive-bayes-spam-detection-7d087cc96d9d&psig=AOvVaw0UR3EzQg6tJcOEK16jSZtw&ust=1602403613296000&source=images&cd=vfe&ved=0CAMQjB1qFwoTCIiN6vzIqewCFQAAAAAdAAAAABAD
4. https://www.sciencedirect.com/topics/computer-science/supervised-learning
5. https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc
6. https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
7. https://developers.google.com/gmail/api