# Project Report

**on**

## SPAM FILTERATION

**Bachelor of Technology In Computer Science & Engineering**

under the guidance of

**Dr. Prabhat Verma**
**Assistant Professor, CSE Dept.**

**By:**


| | |
|---|---|
| **PAWAN SHARMA** | **(Roll no. 1504510029)** |
| **ANKIT NEEKHRA** | **(Roll no. 1504510002)** |
| **SACHIN BHAT** | **(Roll no. 1504510037)** |
| **MOHIT GUPTA** | **(Roll no. 1504510026)** |




**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**HARCOURT BUTLER TECHNICAL UNIVERSITY, KANPUR**

# INDEX

# CERTIFICATE

This is to certify that this report entitled **"Spam Filteration"** which is submitted by Pawan Sharma, Mohit Gupta, Ankit Neekhra and Sachin Bhat students of Final year of Bachelor of Technology in Computer Science and Engineering .This is a record of candidates own work carried out by them under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Date: 19/12/2018

**Dr. Prabhat Verma**

**(Assistant Professor C.S.E Dept.)**

# ACKNOWLEDGEMENT

Before we get into thick of things, we would like to add a few words of appreciation for the people who have been a part of this project right from its inception. The writing of this project has been one of the significant academic challenges our team have faced and without the support, patience, and guidance of the people involved, this task would not have been completed. It is to them we owe our deepest gratitude.

It gives us immense pleasure in presenting this report of final year project labeled "Alert detection using social networking sites". It has been my privilege to have a team of project guide who have assisted us from the commencement of this project. The success of this project is a result of sheer hard work, and determination put in by me and my team members with the help of my project guide.

We hereby take this opportunity to add special note of thanks for professor **Dr. Prabhat Verma** who undertook to act as our mentor despite his many other academic and professional commitments. His wisdom, knowledge, and commitment to the high standards inspired and motivated me. Without his insight, support, and energy, this project wouldn't have kick-started and neither would have reached fruitfulness.

Pawan Sharma (1504510029)
Ankit neekhra (3604510002)
Sachin Bhatt (1504510037)
Mohit Gupta (1504510026)

# ABSTRACT

E-mail spam has become an epidemic problem that can negatively affect the usability of electronic mail as a communication means. Besides wasting user's time and effort to scan and delete the massive amount of junk e-mails received, it consumes network bandwidth and storage space, slows down e-mail servers, and provides a medium to distribute harmful or offensive content. Several machine learning approaches have been applied to this problem. In this project, we explore a new approach based on naïve Bayesian classifier that can automatically classify e-mail messages as spam or ham (spam filtration) which is fully content based.

# OBJECTIVE

Machine learning is a branch of artificial intelligence concerned with the creation and study of systems that can learn from data. A machine learning system could be trained to distinguish between spam and non-spam (ham) messages using python. **We aim to analyze current methods in machine learning to identify the best techniques to use in content-based spam filtering using python. The primary goal of our project is to analyze machine learning algorithms and use them to implement a content-based spam filters.**

# INTRODUCTION

## 1.1 Background

Spam emails may lead users to websites with malware or phishing schemes, which can access and disrupt the receiver's computer system. These sites can also gather sensitive information. Additionally, spam costs businesses a huge loss per year due to decreased productivity .

A study in July 1997 reported that spam messages constituted approximately 10% of the incoming messages to a corporate network . More recently, MessageLabs stated in its 2006 Annual Security Report that spam activity has increased significantly in 2006 with levels that reach 86.2% of the e-mail traffic.

## 1.2 Overview

Spam can be very costly to e-mail recipients; it reduces their productivity by wasting their time and causing annoyance to deal with a large amount of spam. According to Ferris Research, if an employee got five e-mails per day and consumes 30 seconds on each, then he/she will waste 15 hours a year on them. Multiplying this by the hourly rate of each employ in a company will give the cost of spam to this company . In addition, spam consumes the network bandwidth and storage space and can slow down email servers. Spam software can also be used to distribute harmful content such as viruses, Trojan horses, worms and other malicious codes. It can be a means for phishing attacks as well .

As a result, spam has become an area of growing concern attracting the attention of many security researchers and practitioners. In addition to regulations and legislations, various anti-spam technical solutions have been proposed and deployed to combat this problem. Front-end filtering was the most common and easier way to reject or quarantine spam messages as early as possible at the receiving server. However most of the early anti-spam tools were static; for example using a blacklist of known spammers, a white list of good sources, or a fixed set of keywords to

identify spam messages. Although these list-based methods can substantially reduce the risk provided that lists are updated periodically, they fail to scale and to adapt to spammers' tactics. They can be defeated easily by changing the sender's address each time, intentionally misspelling words, or forging the content to bypass spam filters.

Recently, various machine-learning methods have been used to address spam filtering including support vector machines , memory-based learning , Bayesian classifiers , sparse binary polynomial hash etc. Among these methods, the naïve Bayesian classifier has been widely applied as one of the most effective methods to counteract spam.

# MACHINE LEARNING ALGORITHMS ANALYSIS & DESIGN

## 1.KNN classifier

The K Means is used to partition the objects in such a way that the intra cluster similarity is high but inter cluster similarity is comparatively low. Simply, kNN classification classifies instances based on their similarity to instances in the training data .The set of n objects are classified into k clusters by accepting the input parameter k. As an alternative of assigning to a test pattern the class label of its closest neighbor, the K Nearest Neighbour classifier finds k nearest neighbors on the basis of Euclidean distance.

$$\sqrt{((x2\text{-}x1)^2 - (y2\text{-}y1)^2)}$$

The value of k is very crucial because the precise value of k will help in better classification.

**KNN selection strategies:**

$y' = argmax_v \quad \Sigma_{(xi,yi)} \in_{Dz} \delta(v, y_i)$

$y' = argmax_v \quad \Sigma_{(xi,yi)} \in_{Dz} w_i \ \delta(v, y_i)$

where $w_i = 1/d(x', x_i)^2$

**KNN classification algorithms:**

K= number of nearest neighbors

Foreach test example $z = (x', y')do$

Compute d(x,x') foreach (x,y) $\in D$

Select Dz $\subseteq D$, the set of k

Closest training examples

$y' = argmax_v \quad \Sigma_{(xi,yi)} \in_{Dz} \delta(v, y_i)$

The notion of distance/similarity measures is important to knn approach .There are numerous distance/similarity measures.

$d(x, x^2) = \sum i = 1^n |x_i\text{-}x_i'|$ (Manhattan distance)

$sim(x, x^2) = \sum_{i=1}^{n} x^1 x'_i / \sqrt{\sum_{1=1}^{e} x_1{}^2} \sqrt{\sum_{i=1}^{e} x'_1{}^2}$ (cosine similarity)

$sim(x, x^2) = 2\sum_{i=1}^{n} x^1 x'_i / \sum_{1=1}^{e} x_1{}^2 + \sum_{i=1}^{e} x'_2{}^2$ (dice's coefficient)

# 2.SVM(support vector machine) CLASSIFIER

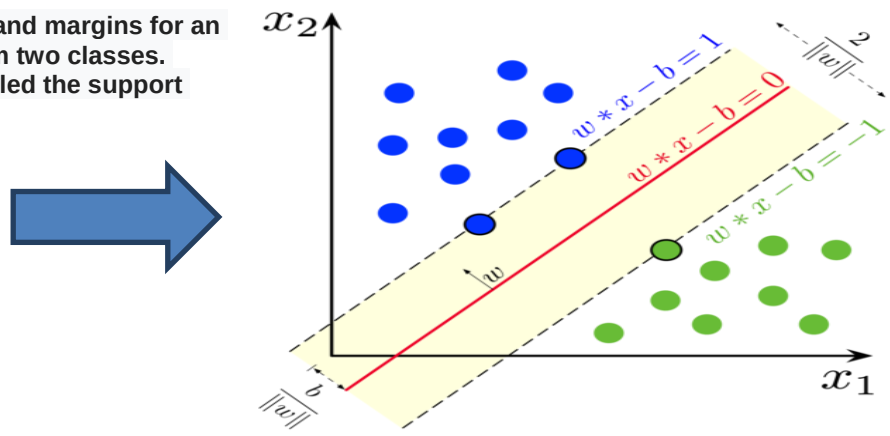
support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.
In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data is unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support vector clustering algorithms, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.
Computing the (soft-margin) SVM classifier amounts to minimizing an expression of the form

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(w \cdot x_i - b)\right)\right] + \lambda\|w\|^2.$$

**Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.**



## 3.Naive Bayes classifier

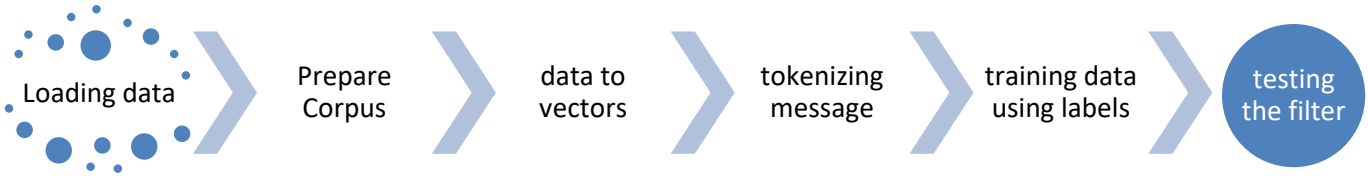
In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It also finds application in automatic medical diagnosis.

Naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector representing some n features (independent variables), it assigns to this instance probabilities
$p(C_k \mid x_1, \ldots, x_n)$
for each of K possible outcomes or classes $C_k$.

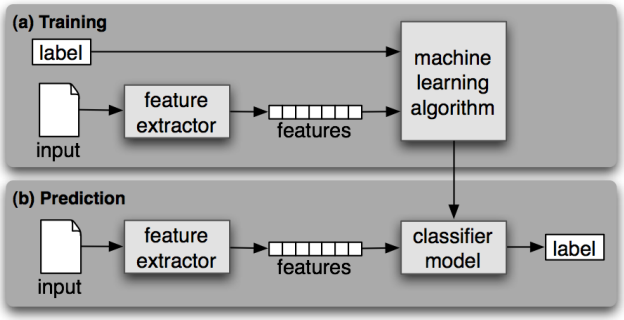
The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

# METHODOLOGY




1. Loading Data- load the spam and ham corpus into the directory.
2. Pre-processing data
    - Making Dictionary- where we put all data(spam or ham) altogether
    - Clean data- remove punctuation ,comma ,numbers and white space.
3. Feature vector- here we select the common words from dictionary with their frequency in order to train our data sets.
4. Tokenizing- This is the process of dividing the message into semantically coherent segments .
5. Training datasets- During the training phase, a model is built based on the characteristics of each label in a pre-classified set of e-mail messages. The training dataset should be selected in such a way that it is varying in content and subject.
6. Testing-using different training sets and feature vectors ,we select and classify these mails as spam or ham using bayesian approach .



# TOOLS USED

## 1.Python

Python is an interpreted, high-level, general-purpose programming language, created by Guido van Rossum and first released in 1991. Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python is commonly used in artificial intelligence projects with the help of libraries like TensorFlow, Keras and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing.

## 2. Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environ-ment for creating Jupyter notebooks documents. A Jupyter Notebook document is a JSON document, fol-lowing a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

## 3. Python Libraries

### 3.1. Scikit-learn

scikit-learn builds on NumPy and SciPy by adding a set of algorithms for common machine learning and data mining tasks, including clustering, regression, and classification. As a library, scikit-learn has a lot going for it. Its tools are well-documented and its contributors include many machine learning experts. Whatâs more, itâs a very curated library, meaning developers wonât have to choose between different ver-sions of the same algorithm. Its power and ease of use make it popular with a lot of data-heavy startups, including Evernote, OKCupid, Spotify, and Birchbox.

### 3.2. Pandas

Pandas adds data structures and tools that are designed for practical data analysis in finance, statis-tics, social sciences, and engineering. Pandas works well with incomplete, messy, and unlabeled data (i.e., the kind of data youâre likely to encounter in the real world), and provides tools for shaping, merging, reshaping, and slicing datasets.)

### 3.3. NumPy

NumPy is the foundational library for scientific computing in Python, and many of the libraries on this list use NumPy arrays as their basic inputs and outputs. In short, NumPy introduces objects for multidi-mensional arrays and matrices, as well as routines that allow developers to perform advanced mathematical and statistical functions on those arrays with as little code as possible.

### 3.4. Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user inter-face toolkits.

### 3.5. Textblob

TextBlob is a Python library for processing textual data. It provides a simple API for diving into com-mon natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, senti-ment analysis, classification, translation, and more.

### 3.6. NLTK

NLTK is a set of libraries designed for Natural Language Processing (NLP). NLTKâs basic functions allow you to tag text, identify named entities, and display parse trees, which are like sentence diagrams that reveal parts of speech and dependencies. From there, you can do more complicated things like sentiment analysis and automatic summarization. It also comes with an entire bookâs worth of material about analyz-ing text with NLTK.)

# IMPLEMENTATION

## Importing Libraries :

```
%matplotlib inline
import matplotlib.pyplot as plt
import csv
from textblob import TextBlob
import pandas
import sklearn
import _pickle as cPickle
import numpy as np
import nltk
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import classification_report, f1_score, accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import StratifiedKFold, cross_val_score, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.learning_curve import learning_curve
```

### Step 1: Analysing the Data

We are using a dataset downloaded from 'https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection'.

This file contains **a collection of more than 5 thousand SMS phone messages (also sometimes called "corpus")** :

In [2]:
```
messages = [line.rstrip() for line in open('SMSSpamCollection')]
print (len(messages))
```

```
5574
```

We are using `pandas` llibrary to list the labeled training set separated by tabs.

Using these ham/spam examples, we'll train a machine learning model to learn to discriminate between ham/spam automatically. Then, with a trained m
we'll be able to classify arbitrary unlabeled messages as ham or spam.

In [3]:
```
messages = pandas.read_csv('SMSSpamCollection', sep='\t', quoting=csv.QUOTE_NONE,
                           names=["label", "message"])
print (messages)
```

```
     label                                            message
0      ham  Go until jurong point, crazy.. Available only ...
1      ham                      Ok lar... Joking wif u oni...
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
5     spam  FreeMsg Hey there darling it's been 3 week's n...
6      ham  Even my brother is not like to speak with me. ...
7      ham  As per your request 'Melle Melle (Oru Minnamin...
```

```
5560   ham                                Sorry, I'll call later
5561   ham  if you aren't here in the next  &lt;#&gt;  hou...
5562   ham                 Anything lor. Juz both of us lor.
5563   ham  Get me out of this dump heap. My mom decided t...
5564   ham  Ok lor... Sony ericsson salesman... I ask shuh...
5565   ham                              Ard 6 like dat lor.
5566   ham  Why don't you wait 'til at least wednesday to ...
5567   ham                                   Huh y lei...
5568  spam  REMINDER FROM O2: To get 2.50 pounds free call...
5569  spam  This is the 2nd time we have tried 2 contact u...
5570   ham             Will ü b going to esplanade fr home?
5571   ham  Pity, * was in mood for that. So...any other s...
5572   ham  The guy did some bitching but I acted like i'd...
5573   ham                        Rofl. Its true to its name

[5574 rows x 2 columns]
```

We can also view aggregate statistics using `pandas` :

In [4]:
```
messages.groupby('label').describe()
```

Out[4]:

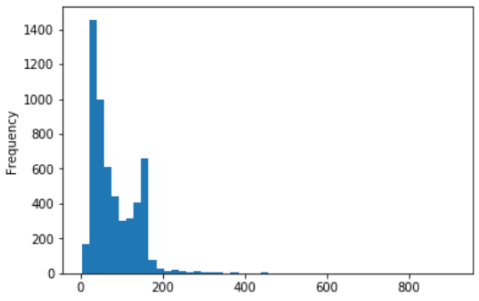
| | message | | | |
|---|---|---|---|---|
| | count | unique | top | freq |
| label | | | | |
| ham | 4827 | 4518 | Sorry, I'll call later | 30 |
| spam | 747 | 653 | Please call our customer service representativ... | 4 |

In [5]:
```
messages['length'] = messages['message'].map(lambda text: len(text))
print (messages.head())
```

```
  label                                            message  length
0   ham  Go until jurong point, crazy.. Available only ...     111
1   ham                      Ok lar... Joking wif u oni...      29
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...     155
3   ham  U dun say so early hor... U c already then say...      49
4   ham  Nah I don't think he goes to usf, he lives aro...      61
```

In [6]:
```
messages.length.plot(bins=50, kind='hist')
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21e3562f8d0>
```



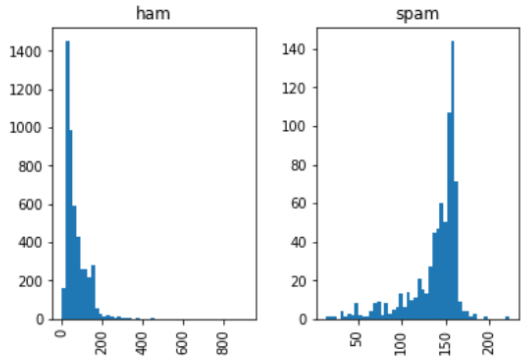

In [7]:
```
messages.length.describe()
```

Out[7]:
```
count    5574.000000
mean       80.478292
std        59.848302
min         2.000000
25%        36.000000
50%        62.000000
75%       122.000000
max       910.000000
Name: length, dtype: float64
```

**Message length difference between spam and ham :**

In [8]:
```
messages.hist(column='length', by='label', bins=50)
```

Out[8]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000021E35507438>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000021E3586E208>],
      dtype=object)
```




## Step 2: Data preprocessing

In this section we'll massage the raw messages (sequence of characters) into vectors (sequences of numbers).

The mapping is not 1-to-1; we'll use the `bag-of-words` approach, where each unique word in a text will be represented by one number.

As a first step, we will write a function that will split a message into its individual words :

In [9]:
```
def split_into_tokens(message):
    message = str(message)  # convert bytes into proper unicode
    return TextBlob(message).words
```

This will divide the message into tokens :

In [10]:
```
messages.message.head().apply(split_into_tokens)
```

Out[10]:
```
0    [Go, until, jurong, point, crazy, Available, o...
1                      [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, in, 2, a, wkly, comp, to, win, F...
3    [U, dun, say, so, early, hor, U, c, already, t...
4    [Nah, I, do, n't, think, he, goes, to, usf, he...
Name: message, dtype: object
```

With textblob, we detect `part-of-speech (POS)` tags :

In [11]:
```
TextBlob("Hello world, how is it going?").tags  # list of (word, POS) pairs
```

Out[11]:
```
[('Hello', 'NNP'),
 ('world', 'NN'),
 ('how', 'WRB'),
 ('is', 'VBZ'),
 ('it', 'PRP'),
 ('going', 'VBG')]
```

and normalize words into their base form (lemmas) with :

In [12]:
```
def split_into_lemmas(message):
    message = str(message).lower()
    words = TextBlob(message).words
    # for each word, take its "base form" = lemma
    return [word.lemma for word in words]

messages.message.head().apply(split_into_lemmas)
```

Out[12]:
```
0    [go, until, jurong, point, crazy, available, o...
1                      [ok, lar, joking, wif, u, oni]
2    [free, entry, in, 2, a, wkly, comp, to, win, f...
3    [u, dun, say, so, early, hor, u, c, already, t...
4    [nah, i, do, n't, think, he, go, to, usf, he, ...
Name: message, dtype: object
```

## Step 3: Data to vectors

Now we'll convert each message, represented as a list of tokens (lemmas) above, into a vector that machine learning models can understand.

Each vector has as many dimensions as there are unique words in the SMS corpus:

In [13]:
```
bow_transformer = CountVectorizer(analyzer=split_into_lemmas).fit(messages['message'])
print (len(bow_transformer.vocabulary_))
```

```
8857
```

Here we used `scikit-learn` (`sklearn`), a powerful Python library for teaching machine learning.

Let's take one text message and get its bag-of-words counts as a vector, putting to use our new `bow_transformer`:

In [14]:
```
message4 = messages['message'][3]
print (message4)
```

```
U dun say so early hor... U c already then say...
```

In [15]:
```
bow4 = bow_transformer.transform([message4])
print (bow4)
print (bow4.shape)
```

```
  (0, 1156)     1
  (0, 1895)     1
  (0, 2891)     1
  (0, 2921)     1
  (0, 4015)     1
  (0, 6724)     2
  (0, 7099)     1
  (0, 7685)     1
  (0, 8000)     2
(1, 8857)
```

So, nine unique words in message nr. 4, two of them appear twice, the rest only once. Sanity check: what are these words the appear twice?

In [16]:
```
print (bow_transformer.get_feature_names()[6736])
print (bow_transformer.get_feature_names()[8013])
```

```
scenario
ugo
```

The bag-of-words counts for the entire SMS corpus are a large, sparse matrix:

In [17]:
```
messages_bow = bow_transformer.transform(messages['message'])
print ('sparse matrix shape:', messages_bow.shape)
print ('number of non-zeros:', messages_bow.nnz)
print ('sparsity: %.2f%%' % (100.0 * messages_bow.nnz / (messages_bow.shape[0] * messages_bow.shape[1])))
```

```
sparse matrix shape: (5574, 8857)
number of non-zeros: 80340
sparsity: 0.16%
```

In [18]:
```
tfidf_transformer = TfidfTransformer().fit(messages_bow)
tfidf4 = tfidf_transformer.transform(bow4)
print (tfidf4)
```

```
  (0, 8000)     0.30500390244920195
  (0, 7685)     0.22530830435091123
  (0, 7099)     0.19139747787841085
  (0, 6724)     0.5233907074133691
  (0, 4015)     0.4563719925781256
  (0, 2921)     0.329688073969027
  (0, 2891)     0.30370462627268274
  (0, 1895)     0.24665241656533501
  (0, 1156)     0.27494440163841516
```

In [19]:
```
messages_tfidf = tfidf_transformer.transform(messages_bow)
print (messages_tfidf.shape)
```

```
(5574, 8857)
```

## Step 4: Training a model, detecting spam

With messages represented as vectors, we can finally train our spam/ham classifier.

We'll be using scikit-learn here, choosing the `Naive Bayes` classifier :

In [30]:
```
%time spam_detector = MultinomialNB().fit(messages_tfidf, messages['label'])
```

```
Wall time: 19 ms
```

Let's try classifying our single random message:

In [31]:
```
print ('predicted:', spam_detector.predict(tfidf4)[0])
print ('expected:', messages.label[3])
```

```
predicted: ham
expected: ham
```

In [22]:
```
all_predictions = spam_detector.predict(messages_tfidf)
print (all_predictions)
```

```
['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']
```

In [23]:
```
print (classification_report(messages['label'], all_predictions))
```

```
              precision    recall  f1-score   support

         ham       0.97      1.00      0.98      4827
        spam       1.00      0.77      0.87       747

 avg / total       0.97      0.97      0.97      5574
```

## Step 5: Running experiments

Now we will split the data into a training/test set, where the model only ever sees the **training data** during its model fitting and parameter tuning. The **test data** is only used for testing.

In [24]:
```
msg_train, msg_test, label_train, label_test = \
    train_test_split(messages['message'], messages['label'], test_size=0.2)

print (len(msg_train), len(msg_test), len(msg_train) + len(msg_test))
```

```
4459 1115 5574
```

The test size is 20% of the entire dataset (1115 messages out of total 5574), and the training is the rest (4459 out of 5574).

In [25]:
```
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=split_into_lemmas)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])
```

In [26]:
```
scores = cross_val_score(pipeline,  # steps to convert raw messages into models
                         msg_train,  # training data
                         label_train,  # training labels
                         cv=10,  # split data randomly into 10 parts: 9 for training, 1 for scoring
                         scoring='accuracy',  # which scoring metric?
                         n_jobs=1,  # 1 = use all cores = faster
                         )
print (scores)
```

```
[0.95749441 0.94170404 0.95067265 0.9529148  0.94394619 0.93946188
 0.96412556 0.9529148  0.9505618  0.9505618 ]
```

The scores for this set:

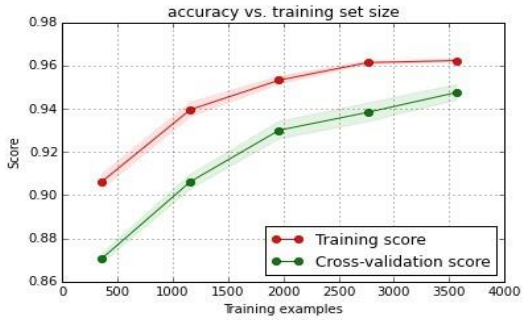
In [27]:
```
print (scores.mean(), scores.std())
```

```
0.9504357912978355 0.0069885463380917135
```

```python
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and traning learning curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : integer, cross-validation generator, optional
        If an integer is passed, it is the number of folds (defaults to 3).
        Specific cross-validation objects can be passed, see
        sklearn.cross_validation module for the list of possible objects

    n_jobs : integer, optional
        Number of jobs to run in parallel (default 1).
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

```
%time plot_learning_curve(pipeline, "accuracy vs. training set size", msg_train, label_train, cv=5)
```

```
CPU times: user 382 ms, sys: 83.1 ms, total: 465 ms
Wall time: 28.5 s

<module 'matplotlib.pyplot' from '/Volumes/work/workspace/vew/sklearn_intro/lib/python2.7/site-packages/matplotlib/py
plot.pyc'>
```



# CONCLUSION

This method considers the content of the message to predict its category rather than relying on the fixed pre –specified set of keywords. Thus , it can adapt to spammer tactics and dynamically build its knowledge base. It was shown that the classifier can achieve very good results provided that we choose enough training samples of legitimate and spam e-mails.

As of now, our system is configured to loading data, prepare corpus, data to vectors, tokenizing ,training the data using label and testing this filteration on new data sets. It will be coded for SMS filteration. Throughout the implementation we learn that naive bayes approach will give 90-95% accuracy if the data sets given are more simplified.

# REFERENCES

[1] L.F. Cranor, and B.A. LaMacchia, “Spam!” Communications of the ACM, vol. 41 no. 8, pp. 74-83, Aug. 1998.

[2] MessageLabs, “MessageLabs Intelligence: 2006 Annual Security Report,” 2006. http://www.messagelabs.com/resources/mlireports

[3] M. Siponen and C. Stucke, “Effective anti-spam strategies in companies: an international study,” In Proc. of the 39th Annual Hawaii Int. Conf. on System Sciences, 2006.

[4] The British Computer Society. Anti-spam blamed for 5M lost hours, 2007. http://www.bcs.org/

[5] X.-L. Wang, and I. Cloete, “Learning to classify e-mail: A survey,” In Proc. of the 4th Int. Conf. on Machine Learning and Cybernetics, Guangzhou, Aug. 2005.

[6] H. Drucker, D. Wu, and V.N. Vapnik, "Support vector machines for spam categorization," IEEE Transactions on Neural Networks, vol. 10, no. 5, pp. 1048 – 1054, Sept. 1999.

[7] G. Sakkis, I. Androutsopoulos, G. Paliouras, “A memory-based approach to anti-spam filtering,” Information Retrieval, vol. 6, pp. 49-73, 2003.

[8] D.C. Trudgian, “Spam Classification Using Nearest Neighbour Techniques,” In Proc. of the Fifth Int. Conf. on Intelligent Data Engineering and Automated Learning (IDEAL04), UK, 2004.

[9] J. Clark, I. Koprinska, and J. Poon, “A neural network based approach to automated e-mail classification,” In Proc. of the IEEE/WIC Int. Conf. on Web Intelligence (WI’03), 2003.

[10] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A Bayesian approach to filtering junk e-mail,” In Proc. of AAAI’98 Workshop on Learning for Text Categorization, Madison, WI, July 1998.

[11] J. Provost, “Naïve-Bayes vs. rule-learning in classification of e-mail,” The University of Texas