Fake News Recommendation System using Natural language processing.

- The dataset is from https://github.com/GeorgeMcIntire/fake_real_news_dataset
- Natural language processing (NLP) is the relationship between computers and human language. More specifically, natural language processing is the computer understanding, analysis, manipulation, and/or generation of natural language.

In [0]:

```
# Fuse wrapper the google drive
!pip install pydrive
!pip install gensim
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:

```
download = drive.CreateFile({'id':'1m50raYcuWAVBtsRzHS7rYcZYVW9-oYEx'})
download.GetContentFile('fake_or_news.csv')
```

In [0]:

```
# loading libraries and data

%matplotlib inline

import sqlite3                          # for sql database
import pandas as pd
import numpy as np
import nltk                             # nltk:- Natural Language Processing Toolkit
import string
import re
import io
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

In [0]:

```
df = pd.read_csv('fake_or_news.csv')
```

In [8]:

```
df.head()
```

Out[8]:

	Unnamed: 0	title	text	label
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE

	Unnamed: 0	title	text	label
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

In [9]:

```
df.describe()
```

Out[9]:

	Unnamed: 0
count	6335.000000
mean	5280.415627
std	3038.503953
min	2.000000
25%	2674.500000
50%	5271.000000
75%	7901.000000
max	10557.000000

In [10]:

```
df.shape
```

Out[10]:

```
(6335, 4)
```

In [11]:

```
df['label'].size
```

Out[11]:

```
6335
```

In [0]:

```
df['label_num'] = df['label'].map(lambda x: 1 if x == "REAL" else 0)
```

In [13]:

```
df.head()
```

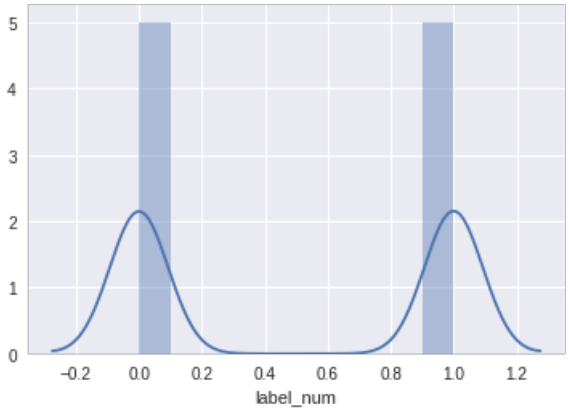
Out[13]:

	Unnamed: 0	title	text	label	label_num
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE	0
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE	0
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL	1

	Unnamed: 0	title	text	label	label_num
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE	0
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL	1

In [14]:

```
sns.distplot(df['label_num'],bins=10)
plt.show()
```



Text Preprocessing: Stemming, stop-word removal and Lemmatization.

Now that we have to finish deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe fake and real news

1) Data Cleaning

(i) Data Deduplication

In [15]:

```
df.duplicated(subset={"title"}).value_counts()
```

Out[15]:

```
False    6256
True       79
dtype: int64
```

In [0]:

```
# Deleting the duplicates
df1 =  df.drop_duplicates(subset={"title"},keep="first")
```

In [17]:

```
df1.duplicated(subset={"title"}).value_counts()
```

Out[17]:

```
False    6256
dtype: int64
```

2) Text Preprocessing

(i) HTML Tag Removal

In [0]:

```
import re
# cleaning html symbols from the sentence
def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
```

(ii) Punctuations Removal

In [0]:

```
# cleaning punctuations from the sentence
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

(iii) Stopwords

- A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

In [20]:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = stopwords.words('english') #All the stopwords in English language
#excluding some useful words from stop words list as we doing sentiment analysis
excluding = ['against','not','don', "don't",'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't",
             'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't",
             'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",'shouldn', "shouldn't", '
wasn',
             "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
stop = [words for words in stop if words not in excluding]
print(stop)
```

```
[nltk_data] Downloading package stopwords to /content/nltk_data...
```

```
[nltk_data]   Package stopwords is already up-to-date!
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
"you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's",
'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', '
is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'd
id', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'between', 'into', 'through', 'during', 'before', 'after', 'ab
ove', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'fu
rther', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'eac
h', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'should', "should've", 'now', 'd', 'll', '
m', 'o', 're', 've', 'y', 'ma', 'shan', "shan't"]
```

(iv) Stemming

- A word stem is part of a word. It is sort of a normalization idea, but linguistic. For example, the stem of the word waiting is wait.

In [0]:

```
from nltk.stem import SnowballStemmer
snow = SnowballStemmer('english') #initialising the snowball stemmer
```

In [22]:

```
print(snow.stem('Secretary')) #same meaning word with different spellings
```

```
secretari
```

PreProcessing on all the news

In [0]:

```
i = 0
string1 = ' '
final_string = []
fake_words = []
real_words = []
s = ''

for sent in df1['text'].values:
    filtered_sentence = []
    sent = cleanhtml(sent)
    sent = cleanpunc(sent)
    for w in sent.split():
        if((w.isalpha()) and (len(w)>2)):
            if(w.lower() not in stop):    # If it is a stopword
                s = (snow.stem(w.lower())).encode('utf8')
                filtered_sentence.append(s)
                if (df1['label'].values)[i] == 'REAL':
                    real_words.append(s)
                if(df1['label'].values)[i] == 'FAKE':
                    fake_words.append(s)
            else:
                continue
        else:
            continue
    string1 = b" ".join(filtered_sentence)
    final_string.append(string1)
    i += 1
```

Fake and Real words in text

In [24]:

In [24]:

```
from collections import Counter
print("Number of Real words: ", len(real_words))
print("Number of Fake words: ", len(fake_words))
```

```
Number of Real words:  1467750
Number of Fake words:  1082029
```

In [25]:

```
df1['CleanedText'] = final_string
df1.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[25]:

	Unnamed: 0	title	text	label	label_num	CleanedText
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE	0	b'daniel greenfield shillman journal fellow fr...
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE	0	b'googl pinterest digg linkedin reddit stumble...
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL	1	b'secretari state john kerri said monday stop ...
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE	0	b'kayde king novemb lesson tonight dem time de...
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL	1	b'primari day new york hillari clinton donald ...

In [0]:

```
# without stem
i = 0
string1 = ' '
final_string_nostem = []
s = ''

for sent in df1['text'].values:
    filtered_sentence=[]
    sent = cleanhtml(sent)
    sent = cleanpunc(sent)
    for w in sent.split():
        if((w.isalpha()) and (len(w)>2)):
            if(w.lower() not in stop):
                s = w.lower().encode('utf8')
                filtered_sentence.append(s)
            else:
                continue
        else:
            continue
    string1 = b" ".join(filtered_sentence)
    final_string_nostem.append(string1)
    i += 1
```

In [27]:

```
df1['CleanedText_NoStem'] = final_string_nostem
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [28]:

```
df1.head(3)
```

Out[28]:

	Unnamed: 0	title	text	label	label_num	CleanedText	CleanedText_NoStem
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE	0	b'daniel greenfield shillman journal fellow fr...	b'daniel greenfield shillman journalism fellow...
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE	0	b'googl pinterest digg linkedin reddit stumble...	b'google pinterest digg linkedin reddit stumbl...
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL	1	b'secretari state john kerri said monday stop ...	b'secretary state john kerry said monday stop ...

Convert words to vector

Using Google's Trained W2Vec on Google News

In [0]:

```
from gensim.models import KeyedVectors
#!wget -c "https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"
w2v_model_google = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin.gz',
binary=True)
```

In [3]:

```
w2v_model_google.wv.most_similar('Secretary')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to
deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion of the se
cond argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be
treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

Out[3]:

```
[('secretary', 0.7287614941596985),
 ('Assistant_Secretary', 0.7153509259223938),
 ('Undersecretary', 0.7121137380599976),
 ('Secre_tary', 0.6972907185554504),
 ('Secretery', 0.6963104605674744),
 ('Secreatry', 0.6765152812004089),
 ('ecretary', 0.671087384223938),
 ('Secretaries', 0.6705101728439331),
 ('Sectretary', 0.6645406484603882),
 ('Sectary', 0.6633391976356506)]
```

In [4]:

```
w2v_model_google.wv["word"].size
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to
deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  """Entry point for launching an IPython kernel.
```

Out[4]:

```
300
```

In [31]:

```
import random
avg_vec_google = []
datapoint = 3000
sample_cols = random.sample(range(1, datapoint), 1001)

for sent in df1['CleanedText_NoStem'].values[sample_cols]:
    cnt = 0
    sent_vec = np.zeros(300)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
            wvec = w2v_model_google.wv[word]
            sent_vec += wvec
            cnt += 1
        except:
            pass
    sent_vec /= cnt
    avg_vec_google.append(sent_vec)
avg_vec_google = np.array(avg_vec_google)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: DeprecationWarning: Call to
deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: invalid value
encountered in true_divide
```

In [33]:

```
np.any(np.isnan(avg_vec_google))
```

Out[33]:

```
True
```

In [34]:

```
np.all(np.isfinite(avg_vec_google))
```

Out[34]:

```
False
```

In [0]:

```
col_mean = np.nanmean(avg_vec_google, axis=0)
```

In [36]:

```
inds = np.where(np.isnan(avg_vec_google))
inds
```

Out[36]:

```
(array([109, 109, 109, ..., 972, 972, 972]),
 array([  0,   1,   2, ..., 297, 298, 299]))
```

In [0]:

```
avg_vec_google[inds] = np.take(col_mean, inds[1])
```

In [38]:

```
avg_vec_google
```

Out[38]:

```
array([[ 0.04753368,  0.01504507, -0.0101147 , ..., -0.02910873,
         0.01647589, -0.01087545],
       [-0.06260173,  0.05096436, -0.08337402, ..., -0.12487284,
        -0.05589294,  0.08426921],
       [ 0.05987947,  0.05090068,  0.02056292, ..., -0.07432445,
        -0.01330204,  0.02458463],
       ...,
       [ 0.02535722,  0.05004649,  0.02385925, ..., -0.0734846 ,
         0.00853295,  0.0134999 ],
       [ 0.02437418,  0.01793511,  0.00687065, ..., -0.03399273,
         0.01450141,  0.01481058],
       [ 0.03974567,  0.03144396,  0.01209868, ..., -0.07372945,
         0.0045726 ,  0.0053955 ]])
```

In [0]:

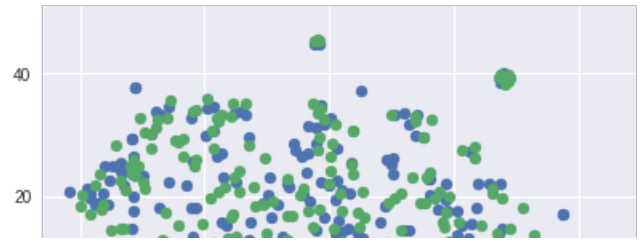
```
from sklearn import preprocessing
avg_vec_google_norm = preprocessing.normalize(avg_vec_google)
```

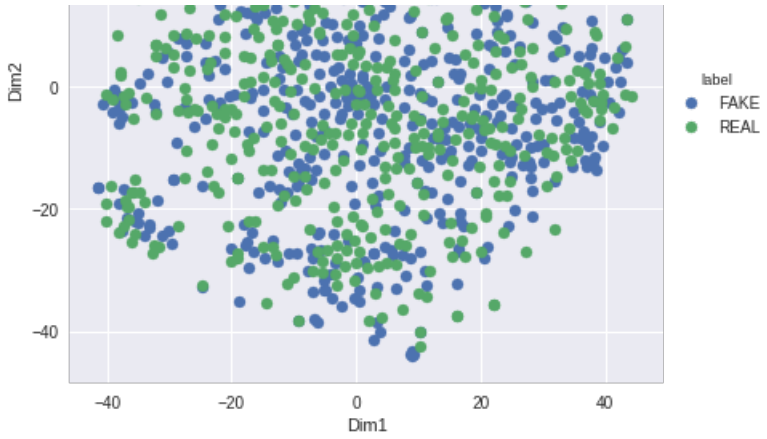
Visualizing using TSNE

In [40]:

```
# Perplexity = 20

from sklearn.manifold import TSNE
import random

n_samples = 1000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]

sample_class = df1['label'][sample_cols]
sample_class = sample_class[:,np.newaxis]
model = TSNE(n_components=2,random_state=0,perplexity=20)

embedded_data = model.fit_transform(sample_features)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
tsne_data = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","label"])

sns.FacetGrid(tsne_data,hue="label",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/series.py:696: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  return self.loc[key]
```

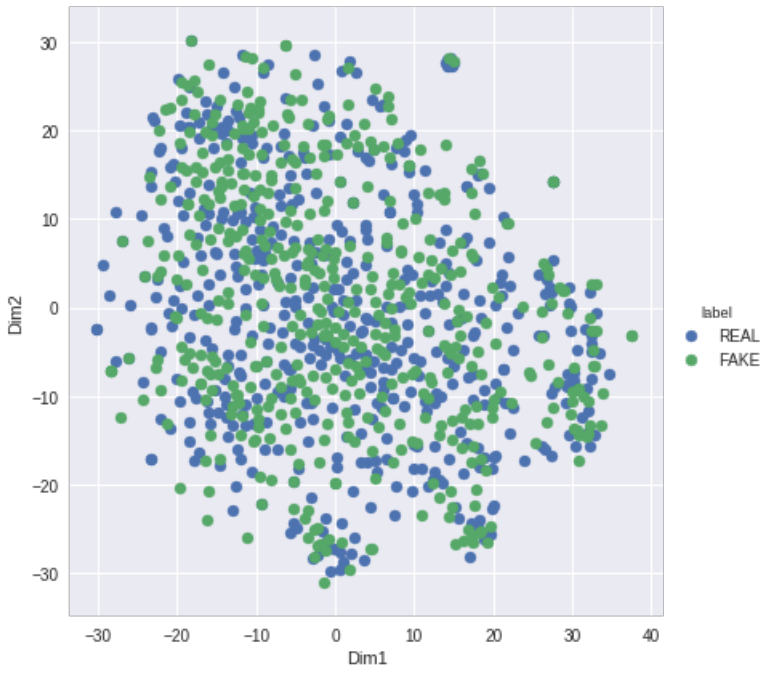




In [41]:

```
# Perplexity = 30

from sklearn.manifold import TSNE
import random

n_samples = 1000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]

sample_class = df1['label'][sample_cols]
sample_class = sample_class[:,np.newaxis]
model = TSNE(n_components=2,random_state=0,perplexity=30)

embedded_data = model.fit_transform(sample_features)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
tsne_data = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","label"])

sns.FacetGrid(tsne_data,hue="label",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/series.py:696: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  return self.loc[key]
```

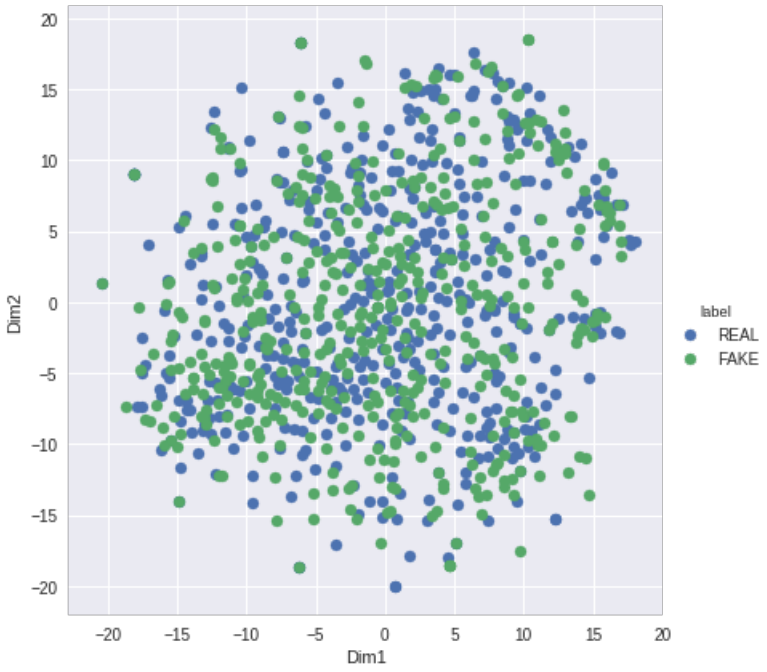


In [42]:

In [42]:

```
# Perplexity = 50

from sklearn.manifold import TSNE
import random

n_samples = 1000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]

sample_class = df1['label'][sample_cols]
sample_class = sample_class[:,np.newaxis]
model = TSNE(n_components=2,random_state=0,perplexity=50)

embedded_data = model.fit_transform(sample_features)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
tsne_data = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","label"])

sns.FacetGrid(tsne_data,hue="label",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/series.py:696: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  return self.loc[key]
```



Observation:-

- We can say that there is less overlapping fake and real news points in the graph.
- Therefore Average Word2Vec will give the best classifying the news as fake or real.

Avg Word2vec

- word2vec is an algorithm for constructing vector representations of words, also known as word embeddings. The vector for each word is a semantic description of how that word is used in context, so two words that are used similarly in text will get similar vector represenations. Once you map words into vector space, you can then use vector math to find words that have similar semantics.

In [0]:

```
from sklearn.base import TransformerMixin, BaseEstimator
from gensim.models import Word2Vec
```

```
class w2v(TransformerMixin, BaseEstimator):

    def __init__(self, size=100, alpha=0.025, window=5, min_count=10, max_vocab_size=None, sample=1
e-3, seed=1,
                 workers=3, min_alpha=0.0001, sg=1, hs=0, negative=5, cbow_mean=1, iter=10, null_wo
d=0,
                 trim_rule=None, sorted_vocab=1, batch_words=10000):
        """
        Sklearn wrapper for Word2Vec model. See gensim.models.Word2Vec for parameter details.
        """
        self.model = None
        self.size = size
        self.alpha = alpha
        self.window = window
        self.min_count = min_count
        self.max_vocab_size = max_vocab_size
        self.sample = sample
        self.seed = seed
        self.workers = workers
        self.min_alpha = min_alpha
        self.sg = sg
        self.hs = hs
        self.negative = negative
        self.cbow_mean = int(cbow_mean)
        self.iter = iter
        self.null_word = null_word
        self.trim_rule = trim_rule
        self.sorted_vocab = sorted_vocab
        self.batch_words = batch_words

    def fit(self, X, y=None):
        """
        Fit the model according to the given training data.
        Calls gensim.models.Word2Vec
        """
        X_tokenized = X.apply(lambda sent: sent.split())

        self.model = Word2Vec(
            sentences=X_tokenized, size=self.size, alpha=self.alpha,
            window=self.window, min_count=self.min_count, max_vocab_size=self.max_vocab_size,
            sample=self.sample, seed=self.seed, workers=self.workers, min_alpha=self.min_alpha,
            sg=self.sg, hs=self.hs, negative=self.negative, cbow_mean=self.cbow_mean,
            iter=self.iter, null_word=self.null_word, trim_rule=self.trim_rule,
            sorted_vocab=self.sorted_vocab, batch_words=self.batch_words
        )
        return self

    def transform(self, X):
        return X.apply(self.avg_word_vector)

    def avg_word_vector(self, sent):
        """
        Returns an average word vector
        from list of words
        if words are not in the wordvector
        vocabulary or in wordlist is
        empty then returns a zero vector
        """
        word_list = sent.split()
        sent_vec = np.zeros(self.size)
        word_cnt = 0
        for word in word_list:
            try:
                vec = self.model.wv[word]
                sent_vec += vec
                word_cnt += 1
            except KeyError:
                pass
        if word_cnt != 0:
            sent_vec /= word_cnt
        return pd.Series(sent_vec)

    def most_similar(self, word):
        return self.model.wv.most_similar(word)

    def vocabulary(self):
        """
```

```
        Returns a Dataframe with
        word as index and rows as vectors
        """
        # build a list of the terms, integer indices,
        # and term counts from the word2vec model vocabulary
        ordered_vocab = [(term, voc.index, voc.count)
                         for term, voc in self.model.wv.vocab.items()]

        # sort by the term counts, so the most common terms appear first
        ordered_vocab = sorted(ordered_vocab, key=lambda k : k[2], reverse=True)

        # unzip the terms, integer indices, and counts into separate lists
        ordered_terms, term_indices, term_counts = zip(*ordered_vocab)

        # create a DataFrame with the word vectors as data,
        # and the terms as row labels
        wordvectors = pd.DataFrame(self.model.wv.vectors_norm[term_indices, :],
                                   index=ordered_terms)

        return wordvectors
```

In [32]:

```
w2v_model = w2v()

w2v_features = w2v_model.fit_transform(df1['text'])
w2v_features.shape
```

Out[32]:

```
(6256, 100)
```

In [0]:

```
from sklearn import preprocessing
avg_w2v_norm = preprocessing.normalize(w2v_features)
```

Applying Classification Techniques

In [0]:

```
# spliting data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(avg_w2v_norm,df1['label'],
                                                    test_size=0.3, shuffle=False)
```

K Nearest Neighbor

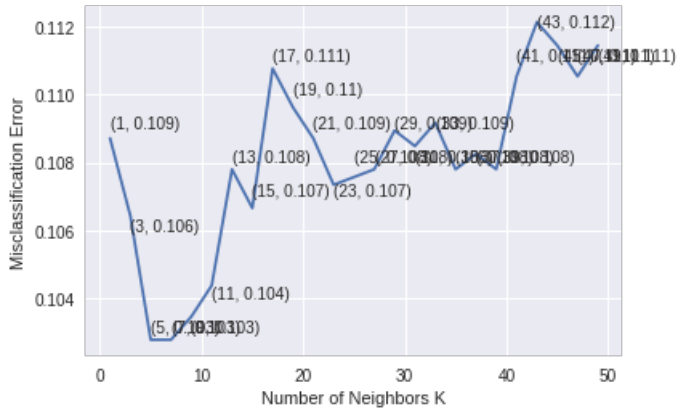
In [40]:

```
# loading libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation

#creating odd list of K for KNN

myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []
```

```
# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
plt.plot(neighbors, MSE)

for xy in zip(neighbors, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(MSE,3))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This mo
dule was deprecated in version 0.18 in favor of the model_selection module into which all the refa
ctored classes and functions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
The optimal number of neighbors is 5.
```




```
the misclassification error for each k value is :  [0.109 0.106 0.103 0.103 0.103 0.104 0.108 0.107
0.111 0.11  0.109 0.107
 0.108 0.108 0.109 0.108 0.109 0.108 0.108 0.108 0.111 0.112 0.111 0.111
 0.111]
```

In [41]:

```
# KNN with k = optimal_k
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train, y_train)

# predict the response
pred = knn_optimal.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 5 is 89.611082%
```

Naive Bayes

In [74]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
clf = MultinomialNB()
clf.fit(X_train, y_train)
```

Out[74]:

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [75]:

```
y_pred = clf.predict(X_test)

print('Accuracy on test set:', accuracy_score(y_test, y_pred))
print('\n                    Classification Report')
print(classification_report(y_test, y_pred))
```

```
Accuracy on test set: 0.7464038359083645

                    Classification Report
              precision    recall  f1-score   support

        FAKE       0.99      0.50      0.66       939
        REAL       0.67      0.99      0.80       938

avg / total       0.83      0.75      0.73      1877
```

Logistic Regression

In [0]:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import Normalizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
```

In [78]:

```
# Train model with default hyperparameter
clf = LogisticRegression()
clf.fit(X_train, y_train)

print('Accuracy on test set:', clf.score(X_test, y_test))
```

```
Accuracy on test set: 0.9062333510921684
```

In [79]:

```
# Column standardization
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

clf = Pipeline([('Scalar', StandardScaler(with_mean=False)),
                ('logreg', LogisticRegression())])
clf.fit(X_train, y_train)
print('Accuracy on test set:', clf.score(X_test, y_test))
```

```
Accuracy on test set: 0.9168886521044219
```

In [0]:

```
from sklearn.model_selection import GridSearchCV

def search_param(clf, param_grid, X, y, test_size=0.3, cv=10, n_jobs=-1,
                 random_search=False, n_iter=10, return_train_score=False):
    """
    Splits the data using time based slicing
    Performs grid search to search for optimal parameter
    using exaustive or random search
    Print accuracy and classification report with
    best parameter on test set
    """

    # time based splitting of dataset
    X_train, X_test, y_train, y_test = train_test_split_by_time(X, y, test_size)

    if random_search:
        grid_search = RandomizedSearchCV(clf, param_grid, n_iter=n_iter,
                                         n_jobs=n_jobs, verbose=1, cv=cv,
                                         return_train_score=return_train_score)
    else:
        grid_search = GridSearchCV(clf, param_grid, n_jobs=n_jobs,
                                   verbose=1, cv=cv,
                                   return_train_score=return_train_score)

    print('Performing grid search...\n')
    print('Parameters:')
    print(param_grid)
    print()

    grid_search.fit(X_train, y_train)
    print('\n')

    print("Best CV score: %f" % grid_search.best_score_)
    print("Best parameters set:")
    best_parameters = grid_search.best_estimator_.get_params()
    for param_name in sorted(param_grid.keys()):
        print("\t%s: %r" % (param_name, best_parameters[param_name]))

    print("\nTest score with best_estimator_: %f" % grid_search.best_estimator_.score(X_test, y_tes
t))
    print("\n")
    print("                 Classification Report Test Data")
    print(classification_report(y_test, grid_search.best_estimator_.predict(X_test)))

    return grid_search
```

In [0]:

```
def train_test_split_by_time(X, y, test_ratio = 0.3):
    """
    y parameter needs to be a pandas series
    with timestamp as its index
    """

    train_ratio = 1 - test_ratio
    train_end = int(train_ratio * len(y))

    sorted_index = y.index.argsort()
    train_index = sorted_index[:train_end]
    test_index = sorted_index[train_end:]

    try:
        return X[train_index], X[test_index], y.iloc[train_index], y.iloc[test_index]
    except KeyError:
        # if X is a pandas data structure
        return X.iloc[train_index], X.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
```

In [0]:

```
class power_uniform():
```

```
    def __init__(self, low=0.0, high=1.0, base=10.0):
        self.low = low
        self.high = high
        self.base = base

    def rvs(self, random_state):
        return self.base ** np.random.uniform(self.low, self.high)
```

In [90]:

```
# optimal C for l2 regularization
param_grid = {
    'log__C': power_uniform(-4, 4)
}

clf = Pipeline([
    ('norm', Normalizer()),
    ('log', LogisticRegression(penalty='l2'))
])

best_log = search_param(clf, param_grid, w2v_features, df1['label'], random_search=True)
```

```
Performing grid search...

Parameters:
{'log__C': <__main__.power_uniform object at 0x7fc3f9d35358>}

Fitting 10 folds for each of 10 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    6.5s finished


Best CV score: 0.929893
Best parameters set:
 log__C: 589.6200891891958

Test score with best_estimator_: 0.923815


                Classification Report Test Data
              precision    recall  f1-score   support

        FAKE       0.93      0.92      0.92       946
        REAL       0.92      0.93      0.92       931

avg / total       0.92      0.92      0.92      1877
```

In [92]:

```
# optimal C for l1 regularization
# Using Random search
param_grid = {
    'log__C': 10.0 ** np.arange(-4, 4)
}

clf = Pipeline([
    ('norm', Normalizer()),
    ('log', LogisticRegression(penalty='l1'))
])

best_log = search_param(clf, param_grid, w2v_features, df1['label'], n_jobs=-1)
```

```
Performing grid search...

Parameters:
{'log__C': array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])}

Fitting 10 folds for each of 8 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:   47.7s finished
```

```
Best CV score: 0.929893
Best parameters set:
 log__C: 100.0

Test score with best_estimator_: 0.922216


                Classification Report Test Data
              precision    recall  f1-score   support

        FAKE       0.93      0.91      0.92       946
        REAL       0.91      0.93      0.92       931

avg / total       0.92      0.92      0.92      1877
```

Support Vector Machine Classifier

In [100]:

```
from sklearn.svm import SVC
X_sample, X_sam, y_sample, y_sam = train_test_split(w2v_features, df1['label'], train_size=0.04)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:2026: FutureWarning: From
version 0.21, test_size will always complement train_size unless both are specified.
  FutureWarning)
```

In [0]:

```
# spliting data into train and test
X_train, X_test, y_train, y_test = train_test_split_by_time(X_sample, y_sample)
```

In [102]:

```
# Train model with default hyperparameter
clf = SVC()
clf.fit(X_train, y_train)

print('Accuracy on test set:', clf.score(X_test, y_test))
```

```
Accuracy on test set: 0.5733333333333334
```

In [103]:

```
# Column standardization
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

clf = Pipeline([('Scalar', StandardScaler(with_mean=False)),
                ('svc', SVC())])
clf.fit(X_train, y_train)
print('Accuracy on test set:', clf.score(X_test, y_test))
```

```
Accuracy on test set: 0.9066666666666666
```

In [104]:

```
# Using Random search
param_grid = {
    'svc__C': power_uniform(-4, 4),
    'svc__gamma': power_uniform(-4, 4)
}

clf = Pipeline([
    ('norm', Normalizer()),
    ('svc', SVC())
])
```

```
best_svm = search_param(clf, param_grid, X_sample, y_sample, random_search=True, n_iter=20)
```

```
Performing grid search...

Parameters:
{'svc__C': <__main__.power_uniform object at 0x7fc3f29286d8>, 'svc__gamma':
<__main__.power_uniform object at 0x7fc3f2928048>}

Fitting 10 folds for each of 20 candidates, totalling 200 fits


Best CV score: 0.891429
Best parameters set:
 svc__C: 488.20483906356435
 svc__gamma: 0.0778711669169114

Test score with best_estimator_: 0.893333


                Classification Report Test Data
             precision    recall  f1-score   support

       FAKE       0.88      0.88      0.88        32
       REAL       0.91      0.91      0.91        43

avg / total       0.89      0.89      0.89        75


[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    0.9s finished
```

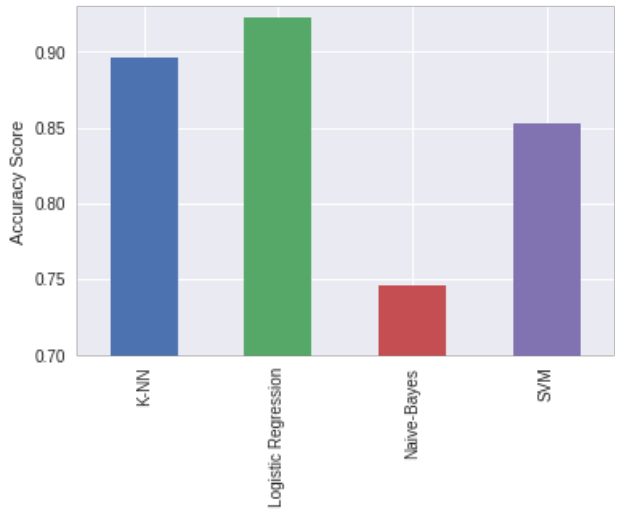
In [105]:

```
# Using Grid search
param_grid = {
    'svc__C': 10.0 ** np.arange(-4, 4),
    'svc__gamma': 10.0 ** np.arange(-4, 4)
}

clf = Pipeline([
    ('norm', Normalizer()),
    ('svc', SVC())
])

best_svm = search_param(clf, param_grid, X_sample, y_sample)
```

```
Performing grid search...

Parameters:
{'svc__C': array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]), 'svc__gamma':
array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])}

Fitting 10 folds for each of 64 candidates, totalling 640 fits


Best CV score: 0.891429
Best parameters set:
 svc__C: 10.0
 svc__gamma: 1.0

Test score with best_estimator_: 0.853333


                Classification Report Test Data
             precision    recall  f1-score   support

       FAKE       0.86      0.78      0.82        32
       REAL       0.85      0.91      0.88        43

avg / total       0.85      0.85      0.85        75


[Parallel(n_jobs=-1)]: Done 640 out of 640 | elapsed:    3.5s finished
```

In [112]:

```
model_accuracy = {
    'K-NN': 0.8961,
    'Naive-Bayes': 0.7464,
    'Logistic Regression': 0.9222,
    'SVM': 0.8533
}

pd.Series(model_accuracy).plot(kind='bar')
plt.ylabel('Accuracy Score')
plt.ylim((0.70, 0.93));
```



Conclusion:-

- From the bar graph we can say that Logistic Regression is giving best accuracy of 92%.