MSDS 7331 Data Mining: Project 2

Team: Andrew Abbott, Vivek Bejugama, Patrick McDevitt, Preeti Swaminathan

We are using an online news popularity dataset from the UCI Machine Learning Repository. The dataset is a collection of 61 heterogeneous features for approximately 40,000 articles published by Mashable (www.mashable.com). The features are not the articles, but are extracted from the articles, such as word counts, title word counts, and keyword associations. The data represents a two year period of published articles, ending in January 2015.

We intend to mine this data to understand what parameters can influence an article to be shared on social media more than others. The goal is to predict the number of shares in social networks (popularity).

The business use of this data set / data mining project is ultimately to establish relationships that enable to predict how many social media shares an article published on *www.mashable.com* is likley to generate - with the idea that a more socially shared article has higher business value - increasing traffic to the site, and consequently, potential for higher earnings revenue associated to the increased web traffic. The user of this model is both the publishers for *mashable.com* for article selection, and also for authors, attempting to create content that has higher value on sites like *mashable.com*.

Measures for a successful outcome from a predictive model for this study will be based on overall accuracy metrics (e.g., confusion matrix), as well as AUC type metrics. A baseline (non-predictive, random) scoring will establish a baseline for these metrics, and then the predictive model can be assessed against a random model for measurement of improvement, i.e., value of the model.

The data is located at https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity (https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity)

Citation Request :

K. Fernandes, P. Vinagre and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal.

- The data set has features in these 6 broad categories :
  (ref - see citation reference at beginning of this document)
  - Words
    - Number of words of the title/content
    - Average word length
    - Rate of unique/non-stop words of contents
  - Links
    - Number of links
    - Number of links to other articles in Mashable
  - Digital Media
    - Number of images/videos
  - Time
    - Day of the week/weekend
  - Keywords
    - Number of keywords
    - Worst/best/average keywords (#shares)
    - Article category
  - NLP

Classification Model Development - Overview

The request for this project includes 2 different classification tasks and (at least) the development and interpretation of 3 different classification models for each task. This section is provided as an overview of the structure of the model development that was employed for both tasks, and for each classifier evaluation. Due to the length of this report, this section is provided as an aid to follow the logic employed throughout this report.

The report includes 2 classification tasks :

1.0 - Prediction of article popularity for articles proposed for publication on *mashable.com* web-site. This is a binary prediction model (popular / not-popular). This was the original intention of the development of this dataset.
2.0 - Assignment of each article to the appropriate **data_channel**. This is the additional task developed by this team, in fulfillment of the additional use of the data set to provide an appropriate business use. In this case, the problem is a multi-classifier problem, to identify to which of 7 data channels (e.g., World, Entertainment, Business, Technology, ...) an article is most appropriate.

This report includes 4 classifier models for each of the above 2 classification tasks :

1. • Logistic Regression
2. • Decision Tree
3. • Random Forest
4. • Naïve Bayes

The model development approach in all cases is as described here below :

1. **Holdout** 20% of data set for final sacred test set
2. Split remaining 80% into train / test for grid search
3. Run **Grid Search with Cross-Validation = 3** on each classification type
   a. Search on range of model parameters to identfy best set of parameters within the searched range
4. Identify best accuracy / best parameters from **Grid Search** a. Run each classifier model with the best parameters that were identified from the grid search. This classifier model is then used in the subsequent full **10-fold Cross-Validation** verification
   b. Verify results are consistent with grid search model results
5. Run full **10-fold CV** with the best parameters for each classifier a. Verify results are consistent with prior grid search results
   b. Evaluate the 10-fold CV on model metrics (in our case Accuracy)
6. Identify best overall model from the 4 classifiers after 10-fold CV
7. Run best overall model on **Holdout** 20% sacred data set
   a. Report results from this test set run as expected model capability
8. Identify **feature importance** from each of the 4 classifier models (best model each)
   a. Consider if feature scaling is beneficial for interpretation
9. Identify strengths / weakness of model prediction capability (e.g., some classes well predicted or not)

   <img src = 'model_development_method.png'>

Modeling & Evaluation - Part 1

We intend to mine this data to understand what parameters can influence an article to be shared on social media more than others. The goal in *Task 1* is to predict the number of shares in social networks (popularity).

The business use of this data set / data mining project is ultimately to establish relationships that enable to predict how many social media shares an article published on *www.mashable.com* is likley to generate - with the idea that a more socially shared article has higher business value - increasing traffic to the site, and consequently, potential for higher earnings revenue associated to the increased web traffic. The user of this model is both the publishers for *mashable.com* for article selection, and also for authors, attempting to create content that has higher value on sites like *mashable.com*.

Measures for a successful outcome from a predictive model for this study will be based on overall **accuracy** metrics. A baseline (non-predictive, random) scoring will establish a baseline for these metrics, and then the predictive model can be assessed against a random model for measurement of improvement, i.e., value of the model.

Risk

For this business case, we consider 2 risks :

- 1 - there is an article that has high potential for high popularity and our model classifies this incorrectly as not popular. To identify this risk we will keep a check on False Negatives and **Recall**.
- 2 - there are articles that are not likely to be popular, and our model incorrectly classifies the article as high potential for high popularity
  To identify this risk we will keep a check on False Positives and **Precision**.

  Since we are looking at both Recall and Precision, **F-Score** is also considered for our evaluation metrics. Additionally, we also look at **processing speed**. In case when two or more model has good evaluation metrics, we would decide best model based on speed.

  In the first case, our model is missing opportunity to identify a highly valuable asset, whereas in the second case our model highly values that which has no value. For our business case, the distinction between popular and not_popular was established at the median value for number of shares of each article. In this case, then, there are, by this measure, an equal number of popular and not_popular articles.

Import required packages

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.simplefilter('ignore',DeprecationWarning)
import seaborn as sns
import time

from pylab import rcParams
#import hdbscan

from sklearn.model_selection import ShuffleSplit
from sklearn.preprocessing import StandardScaler

#from sklearn.datasets import make_blobs

from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn import metrics
from sklearn import metrics as mt
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score as acc
from sklearn.metrics import confusion_matrix as conf
from sklearn.metrics import f1_score, precision_score, recall_score, classi
fication_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support as score

from tabulate import tabulate

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Read in dataset from .csv file

In [4]:
```
data_dir = 'data/'
data_file = 'OnlineNewsPopularity.csv'

file_2_read = data_dir + data_file
df = pd.read_csv(file_2_read)
```

In [5]:
```
df.columns = df.columns.str.strip()
col_names = df.columns.values.tolist()
```

Data Preparation Part 1

Task 1 data set definition

- For Task 1 we intend to mine this data to understand what parameters can influence an article to be shared on social media more than others. The goal is to predict the number of shares in social networks (popularity).
- The business use of this data set / data mining project is ultimately to establish relationships that enable to predict how many social media shares an article published on *www.mashable.com* is likley to generate - with the idea that a more socially shared article has higher business value - increasing traffic to the site, and consequently, potential for higher earnings revenue associated to the increased web traffic. The user of this model is both the publishers for *mashable.com* for article selection, and also for authors, attempting to create content that has higher value on sites like *mashable.com*.
- In order to support this classification task, we create a new dependent variable column **popular** which is true if the value of **shares** is greater than 1400.

Remove variables that are not useful

In [6]:
```
popular_binary = np.where(df['shares'] > 1400, True, False)

df.insert(loc = 0, column = 'popular', value = popular_binary)

del df['shares']
```

In [7]:
```
del df['n_non_stop_words']
del df['n_non_stop_unique_tokens']
del df['n_unique_tokens']
del df['url']
```

In [8]:
```
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	
timedelta	39644.0	354.530471	214.163767	8.00000	164.000000	3
n_tokens_title	39644.0	10.398749	2.114037	2.00000	9.000000	1
n_tokens_content	39644.0	546.514731	471.107508	0.00000	246.000000	4
num_hrefs	39644.0	10.883690	11.332017	0.00000	4.000000	8
num_self_hrefs	39644.0	3.293638	3.855141	0.00000	1.000000	3
num_imgs	39644.0	4.544143	8.309434	0.00000	1.000000	1
num_videos	39644.0	1.249874	4.107855	0.00000	0.000000	0
average_token_length	39644.0	4.548239	0.844406	0.00000	4.478404	4
num_keywords	39644.0	7.223767	1.909130	1.00000	6.000000	7
data_channel_is_lifestyle	39644.0	0.052946	0.223929	0.00000	0.000000	0
data_channel_is_entertainment	39644.0	0.178009	0.382525	0.00000	0.000000	0
data_channel_is_bus	39644.0	0.157855	0.364610	0.00000	0.000000	0
data_channel_is_socmed	39644.0	0.058597	0.234871	0.00000	0.000000	0
data_channel_is_tech	39644.0	0.185299	0.388545	0.00000	0.000000	0
data_channel_is_world	39644.0	0.212567	0.409129	0.00000	0.000000	0
kw_min_min	39644.0	26.106801	69.633215	-1.00000	-1.000000	-
kw_max_min	39644.0	1153.951682	3857.990877	0.00000	445.000000	6
kw_avg_min	39644.0	312.366967	620.783887	-1.00000	141.750000	2
kw_min_max	39644.0	13612.354102	57986.029357	0.00000	0.000000	1
kw_max_max	39644.0	752324.066694	214502.129573	0.00000	843300.000000	8
kw_avg_max	39644.0	259281.938083	135102.247285	0.00000	172846.875000	2
kw_min_avg	39644.0	1117.146610	1137.456951	-1.00000	0.000000	1
kw_max_avg	39644.0	5657.211151	6098.871957	0.00000	3562.101631	4
kw_avg_avg	39644.0	3135.858639	1318.150397	0.00000	2382.448566	2
self_reference_min_shares	39644.0	3998.755396	19738.670516	0.00000	639.000000	1
self_reference_max_shares	39644.0	10329.212662	41027.576613	0.00000	1100.000000	2
self_reference_avg_sharess	39644.0	6401.697580	24211.332231	0.00000	981.187500	2
weekday_is_monday	39644.0	0.168020	0.373889	0.00000	0.000000	0
weekday_is_tuesday	39644.0	0.186409	0.389441	0.00000	0.000000	0
weekday_is_wednesday	39644.0	0.187544	0.390353	0.00000	0.000000	0
weekday_is_thursday	39644.0	0.183306	0.386922	0.00000	0.000000	0
weekday_is_friday	39644.0	0.143805	0.350896	0.00000	0.000000	0
weekday_is_saturday	39644.0	0.061876	0.240933	0.00000	0.000000	0
weekday_is_sunday	39644.0	0.069039	0.253524	0.00000	0.000000	0
is_weekend	39644.0	0.130915	0.337312	0.00000	0.000000	0

Assign certain variables to type integer, as appropriate

In [9]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  convert the data type to Integer
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

to_int = ['timedelta','n_tokens_title', 'n_tokens_content',
    'num_hrefs','num_self_hrefs', 'num_imgs', 'num_videos', 'num_keywords',
    'weekday_is_monday',
    'weekday_is_tuesday',
    'weekday_is_wednesday',
    'weekday_is_thursday',
    'weekday_is_friday',
    'weekday_is_saturday',
    'weekday_is_sunday',
    'is_weekend']


df[to_int] = df[to_int ].astype(np.int64)
```

In [10]:
```
df[df.duplicated()]
```

Out[10]:

	popular	timedelta	n_tokens_title	n_tokens_content	num_hrefs	num_self_hrefs	num_imgs	nun

0 rows × 57 columns

Impute kw_avg_max for 0-values and re-scale to standard normal scale

- A small number of rows have 0 value for **kw_avg_max**, which is completely out of range for the remaining rows of this variable.
- We will impute these rows to median value of the column
- The magnitude of this column of data is markedly different than the range of values in the remaining columns in the data set. To bring this back in line, we will re-scale the values in this column to standard normal range

In [11]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  impute to median value for a few rows of kw_avg_max
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

condition = df['kw_avg_max'] == 0
df.loc[condition, 'kw_avg_max'] = df.kw_avg_max.median()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  scale to standard normal scale
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df.kw_avg_max = (df.kw_avg_max - df.kw_avg_max.mean()) / df.kw_avg_max.std(
)
```

Constant offset for variables with min value < 0

- This allows to consider these variables for ln() transform if highly right-skewed and also supports some classification models that only accept independent variables that are > 0
- Method here is to just add -1 * min_value of any column for which min_value < 0

In [12]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  for all columns with negative values, add +1 to all values in the co
lumn
# ...  - the only columns with negative values are polarity / sentiment mea
sures
# ...  - adding a constant to all values does not modify distributions
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df_numeric = df.select_dtypes(['number'])
numeric_col_names = df_numeric.columns.values.tolist()

# ... store min value for each column

df_mins = df.min()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  loop on each column, test for min < 0, add constant as applicable
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

for column in numeric_col_names :
    if df_mins[column] < 0 :
        df[column] = df[column] - df_mins[column]

#        print('--> min_value < 0 adjusted : ', column, df_mins[column])
```

Ln() transform for variables that are right skewed (skewness > 1)

- This facilitiates maintaining more normally distributed residuals for regression models
- Likely, this will not be needed for the classification task, at present, but also does not have negative effects for this current activity

In [13]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  ln() transform right skewed distribution variables (skewness > 1)
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df_numeric = df.select_dtypes(['number'])

numeric_col_names = df_numeric.columns.values.tolist()

# ... store min value for each column

df_mins = df.min()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  loop on each column, test for skewness, create new column if conditi
ons met
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

columns_to_drop = []

for column in numeric_col_names:
    sk = df[column].skew()

    if(sk > 1):
        new_col_name = 'ln_' + column
        print (column, sk, new_col_name)

        if df_mins[column] > 0:
            df[new_col_name] = np.log(df[column])
            columns_to_drop.append(column)

        elif df_mins[column] == 0:
            df_tmp = df[column] + 1
            df[new_col_name] = np.log(df_tmp)
            columns_to_drop.append(column)

        else:
            print('--> Ln() transform not completed -- skew > 1, but min va
lue < 0 :', column, '!!')


# ... delete tmp data

del df_tmp
del df_mins
del df_numeric

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  based on inspection, a few of these are just not valid ranges in ln(
) space
# ...  -- just delete these few back out of the data set
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

print (columns_to_drop)

del df['ln_LDA_00']
del df['ln_LDA_01']
del df['ln_LDA_02']
```

```
n_tokens_content 2.94542193879 ln_n_tokens_content
num_hrefs 4.0134948282 ln_num_hrefs
num_self_hrefs 5.17275110576 ln_num_self_hrefs
num_imgs 3.94659584465 ln_num_imgs
num_videos 7.0195327863 ln_num_videos
data_channel_is_lifestyle 3.99301914336 ln_data_channel_is_lifestyle
data_channel_is_entertainment 1.6835848094 ln_data_channel_is_entertainment
data_channel_is_bus 1.87687018599 ln_data_channel_is_bus
data_channel_is_socmed 3.75887963097 ln_data_channel_is_socmed
data_channel_is_tech 1.61997576469 ln_data_channel_is_tech
data_channel_is_world 1.40516938412 ln_data_channel_is_world
kw_min_min 2.37494728018 ln_kw_min_min
kw_max_min 35.3284337312 ln_kw_max_min
kw_avg_min 31.3061081027 ln_kw_avg_min
kw_min_max 10.3863716348 ln_kw_min_max
kw_max_avg 16.4116695554 ln_kw_max_avg
kw_avg_avg 5.76017729162 ln_kw_avg_avg
self_reference_min_shares 26.2643641603 ln_self_reference_min_shares
self_reference_max_shares 13.8708490494 ln_self_reference_max_shares
self_reference_avg_sharess 17.9140933777 ln_self_reference_avg_sharess
weekday_is_monday 1.77590824423 ln_weekday_is_monday
weekday_is_tuesday 1.61054706191 ln_weekday_is_tuesday
weekday_is_wednesday 1.60097097689 ln_weekday_is_wednesday
weekday_is_thursday 1.6370700483 ln_weekday_is_thursday
weekday_is_friday 2.03030483518 ln_weekday_is_friday
weekday_is_saturday 3.63708575997 ln_weekday_is_saturday
weekday_is_sunday 3.3999273763 ln_weekday_is_sunday
is_weekend 2.18850033431 ln_is_weekend
LDA_00 1.5674632332 ln_LDA_00
LDA_01 2.08672182342 ln_LDA_01
LDA_02 1.31169490203 ln_LDA_02
LDA_03 1.23871598638 ln_LDA_03
LDA_04 1.17312947598 ln_LDA_04
global_rate_negative_words 1.49191730919 ln_global_rate_negative_words
min_positive_polarity 3.04046773746 ln_min_positive_polarity
abs_title_sentiment_polarity 1.70419343991 ln_abs_title_sentiment_polarity
['n_tokens_content', 'num_hrefs', 'num_self_hrefs', 'num_imgs', 'num_videos
', 'data_channel_is_lifestyle', 'data_channel_is_entertainment', 'data_chan
nel_is_bus', 'data_channel_is_socmed', 'data_channel_is_tech', 'data_channe
l_is_world', 'kw_min_min', 'kw_max_min', 'kw_avg_min', 'kw_min_max', 'kw_ma
x_avg', 'kw_avg_avg', 'self_reference_min_shares', 'self_reference_max_shar
es', 'self_reference_avg_sharess', 'weekday_is_monday', 'weekday_is_tuesday
', 'weekday_is_wednesday', 'weekday_is_thursday', 'weekday_is_friday', 'wee
kday_is_saturday', 'weekday_is_sunday', 'is_weekend', 'LDA_00', 'LDA_01', '
LDA_02', 'LDA_03', 'LDA_04', 'global_rate_negative_words', 'min_positive_po
larity', 'abs_title_sentiment_polarity']

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

Number of current columns in dataset : 80
```

Data Preparation Part 2

Data Selection - Task 1 - Popularity classification

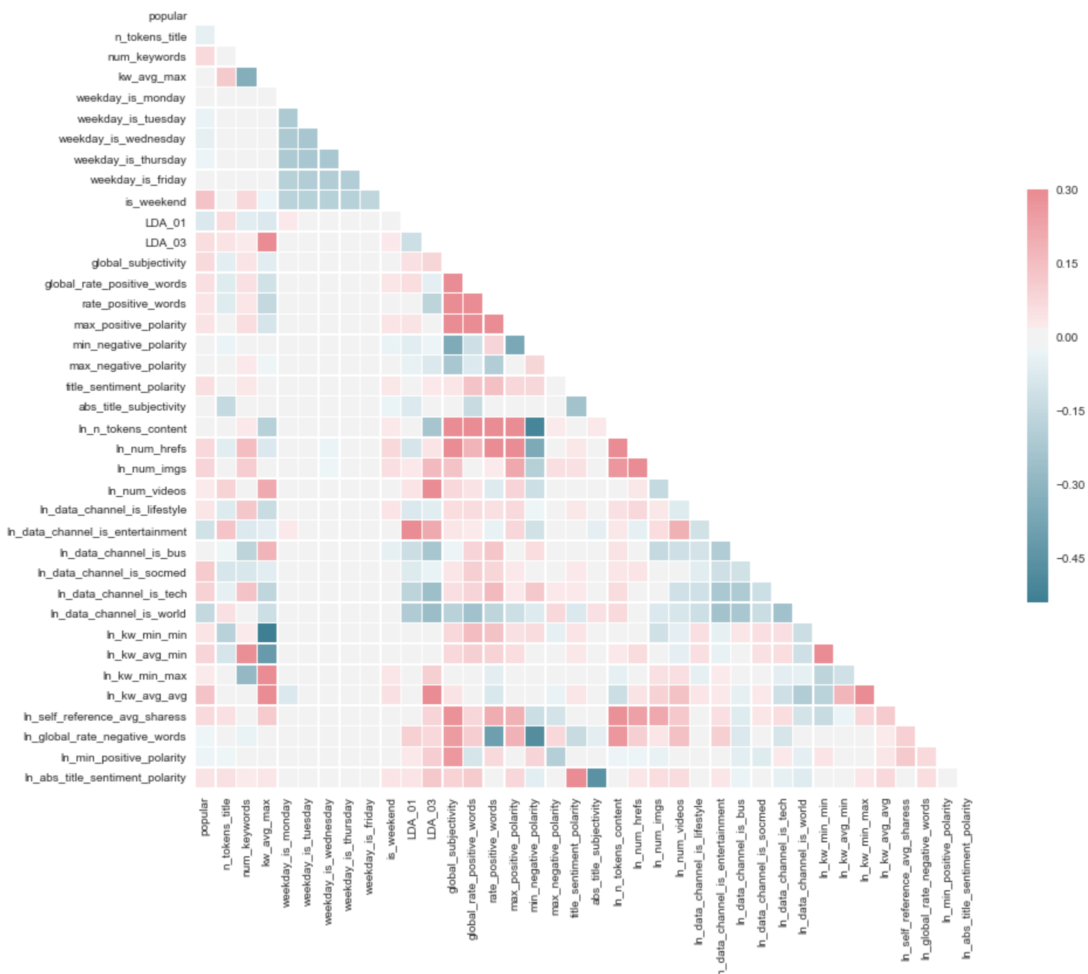
- There are 60 columns in the original data set; we added an additonal column based on the value of shares as explained above.
- From this data set, we did a simple correlation matrix to look for variables that are highly correlated with each other that could be removed with little loss of information.
- With that downselection, we proceeded with additional evaluation of these remaining variables.
- We recognize that there is likely additional opportunity for modeling improvements with the remaining variables; we will look to re-evaluate the data set to further consider that with future work. Those opportunities will become apparent following the outcome of this present evaluation.

In [14]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... display highest correlation pairs from corr() matrix
# ...
# ... https://stackoverflow.com/questions/17778394/list-highest-correlation
-pairs-from-a-large-correlation-matrix-in-pandas
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df_numeric = df.select_dtypes(['number'])

def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n = 5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels = labels_to_drop).sort_values(ascending =
False)
    return au_corr[0:n]

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... list out Top30 correlations
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

n_val = 30

top_30_corr_list = get_top_abs_correlations(df_numeric, n_val)
print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")
print("Top Absolute Correlations\n")
print(top_30_corr_list)


icor = 0
drop_column = list()
while (top_30_corr_list[icor] > 0.65):
    drop_column.append(top_30_corr_list[top_30_corr_list == top_30_corr_lis
t[icor]].index[0][0])
    icor += 1

drop_column = list(set(drop_column))

print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")
print("Columns Recommended for removal based on correlation > 0.65")
print("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n")

print("\n".join(sorted(drop_column)))

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... drop one of the high correlation columns (2nd of the pair)
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df = df.drop(drop_column, axis = 1)
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Top Absolute Correlations

ln_self_reference_max_shares  ln_self_reference_avg_sharess       0.994639
ln_self_reference_min_shares  ln_self_reference_avg_sharess       0.971014
ln_kw_max_min                 ln_kw_avg_min                       0.946087
ln_self_reference_min_shares  ln_self_reference_max_shares        0.945943
kw_min_avg                    ln_kw_min_max                       0.926784
ln_kw_max_avg                 ln_kw_avg_avg                       0.899409
LDA_02                        ln_data_channel_is_world            0.836618
timedelta                     ln_kw_min_min                       0.822783
kw_max_max                    ln_kw_min_min                       0.820625
rate_negative_words           ln_global_rate_negative_words       0.782517
LDA_00                        ln_data_channel_is_bus              0.774651
average_token_length          ln_n_tokens_content                 0.768795
LDA_04                        ln_data_channel_is_tech             0.749737
avg_negative_polarity         min_negative_polarity               0.748896
title_subjectivity            ln_abs_title_sentiment_polarity     0.741229
global_sentiment_polarity     rate_positive_words                 0.727827
avg_positive_polarity         max_positive_polarity               0.703558
weekday_is_sunday             is_weekend                          0.701648
ln_num_self_hrefs             ln_self_reference_max_shares        0.675339
weekday_is_saturday           is_weekend                          0.661707
global_sentiment_polarity     rate_negative_words                 0.650717
max_positive_polarity         ln_n_tokens_content                 0.643190
timedelta                     kw_max_max                          0.637824
ln_num_self_hrefs             ln_self_reference_avg_sharess       0.631820
global_subjectivity           avg_positive_polarity               0.631749
global_rate_positive_words    rate_positive_words                 0.628626
ln_n_tokens_content           ln_num_hrefs                        0.614357
LDA_01                        ln_data_channel_is_entertainment    0.599384
average_token_length          global_subjectivity                 0.597629
avg_negative_polarity         max_negative_polarity               0.580108
dtype: float64


-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Columns Recommended for removal based on correlation > 0.65
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

LDA_00
LDA_02
LDA_04
average_token_length
avg_negative_polarity
avg_positive_polarity
global_sentiment_polarity
kw_max_max
kw_min_avg
ln_kw_max_avg
ln_kw_max_min
ln_num_self_hrefs
ln_self_reference_max_shares
ln_self_reference_min_shares
rate_negative_words
timedelta
title_subjectivity
weekday_is_saturday
weekday_is_sunday


-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

In [15]:
```
sns.set(style="white")

# Compute the correlation matrix
corr = df.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(15, 13))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})


# from example found at https://www.kaggle.com/maheshdadhich/strength-of-vi
sualization-python-visuals-tutorial/notebook
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x117bb9208>



Save cleaned / reduced data set to external .csv file

- This provides opportunity to just read in this .csv file and no need to repeat data cleaning / reduction process for each execution

In [17]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... store in ../data/ directory
# ... write as .csv file for future recall
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

data_dir = 'data/'
data_file = 'mashable_clean_dataset_for_lab_02_task_01.csv'

file_2_write = data_dir + data_file

df.to_csv(file_2_write, index = False)
```

Modeling and Evaluation 2

Holdout, Training and Test split

In [18]:

```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  copy data frame to classification working data frame
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

# ... data set with text categorical target values

df_pop = df.copy()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  separate X and y matrices
# ...
# ...  convert to numpy matrices by calling 'values' on the pandas data fra
mes
# ...  they are now simple matrices for compatibility with scikit-learn
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

if 'popular' in df_pop:
    y = df_pop['popular'].values         # set 'popular as dependent
    del df_pop['popular']                 # remove from dataset
    X = df_pop.values                        # use everything else for inde
pendent EVs

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  setup master train and test , golden traina and test
# ...  master sets - first 80% of original data set which will be base trai
ning for model building
# ...  Golden sets - 20% of original will be used in the final best model f
or prediction
# ...  split into training and test sets
# ....  --> 10 folds
# ...   --> 80% / 20% training / test
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
```

In [19]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... Set-up golden test data set
# ... This data-set will be used to evaluate the predictive capability of t
he final
# ... model on a data set that was not included in any of the prior train/t
est sets
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

num_cv_iterations = 1
cv_object = ShuffleSplit(n_splits = num_cv_iterations,
                         test_size  = 0.2)

print(cv_object)

for train_indices, test_indices in cv_object.split(X, y):
    master_X_train = X[train_indices]
    master_y_train = y[train_indices]
    golden_X_test  = X[test_indices]
    golden_y_test  = y[test_indices]
    print(master_X_train.shape)
```

```
ShuffleSplit(n_splits=1, random_state=None, test_size=0.2, train_size=None)
(31715, 37)
```

In [20]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... Set-up training set to be used on 'best' model from grid search resul
ts
# ... This data-set will be used to verify 10-fold-CV-model has results con
sistent
# ... with the model produced from grid search
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

num_cv_iterations = 1
cv_object = ShuffleSplit(n_splits = num_cv_iterations,
                         test_size  = 0.2)

print(cv_object)

for train_indices, test_indices in cv_object.split(master_X_train, master_y
_train):
    X_train = master_X_train[train_indices]
    y_train = master_y_train[train_indices]
    X_test  = master_X_train[test_indices]
    y_test  = master_y_train[test_indices]
    print(X_train.shape)
```

```
ShuffleSplit(n_splits=1, random_state=None, test_size=0.2, train_size=None)
(25372, 37)
```

In [21]:
```
# set required variables for model comparison

comparison_tbl = pd.DataFrame(columns = [
    'Model Name',
    'Accuracy',
    'Precision',
    'Recall',
    'FScore',
    'Processing Time'])

i_index=[]
i_index = 0

# preparation for cross validation and model comparison, each classifier is
appended once model is fit

models = []
```

Modeling and Evaluation 3

For task 1 we have chosen the following 4 models:
a. Binary logistic regression with parament selection using Grid Search
b. Decision Tree with parament selection using Grid Search
c. Random Forest with parament selection using Grid Search
d. Naive Bayes

Each of these models will be evaluated on Accuracy, Precision, Recall, FScore and Execution time

a. Linear logistic regression

For linear LR we have set standard attributes with: class_weight = balanced
search params:
tolerance parament tol
Regularization parament C

Grid selection for logistic regression

In [22]:
```
from sklearn.grid_search import GridSearchCV

lr_model = LogisticRegression(
    class_weight = 'balanced',
    solver = 'lbfgs',
    C = 10,
    tol = 0.1)

params = {
    'C':[100, 1000],
    'tol': [0.001, 0.0001]
}

# ... --> changed the scoring on Sat 28-Oct
# ...     - from : log_loss
# ...     - to : neg_log_loss
# ...  (this avoids the deprecation warning)

clf = GridSearchCV(
    lr_model,
    params,
    scoring = 'neg_log_loss',
    refit = 'True',
    n_jobs = -1,
    cv = 3)

grid_search = clf.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_C = best_parameters['C']
best_tol = best_parameters['tol']
```

```
/Users/andrewabbott/.virtualenvs/dl4cv/lib/python3.6/site-packages/sklearn/
cross_validation.py:41: DeprecationWarning: This module was deprecated in v
ersion 0.18 in favor of the model_selection module into which all the refac
tored classes and functions are moved. Also note that the interface of the 
new CV iterators are different from that of this module. This module will b
e removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/Users/andrewabbott/.virtualenvs/dl4cv/lib/python3.6/site-packages/sklearn/
grid_search.py:42: DeprecationWarning: This module was deprecated in versio
n 0.18 in favor of the model_selection module into which all the refactored
classes and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

Best parameter values for logistic regression:

In [23]:
```
best_accuracy
best_parameters
```

Out[23]: -0.6445648230563811

Out[23]: {'C': 1000, 'tol': 0.001}

Create main logistic model using best paraments for further analysis and model comparisons

In [24]:
```
tic = time.clock()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... basic Logistic Regression
# ... - normalize features based on mean & stdev of each column
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

lr_model1 = LogisticRegression(
    class_weight = 'balanced',
    solver = 'lbfgs',
    C = best_C,
    tol = best_tol)

lr_model1.fit(X_train, y_train)  # train object

y_hat = lr_model1.predict(X_test) # get test set predictions

toc =  time.clock()

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_hat))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_hat,average=
'weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_hat,average='weigh
ted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_hat,average='weighte
d'))

exetime = '{0:.4f}'.format(toc-tic)

# print statistics

print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", conf(y_test, y_hat))
print('process time',exetime)
print("\n")

# save statistics for model comparison

raw_data = {
    'Model Name' : 'Logistic Regression',
    'Accuracy' : accuracy,
    'Precision' : precision,
    'Recall' : recall,
    'FScore' : f1_score,
    'Processing Time' : exetime
}

df_tbl = pd.DataFrame(raw_data,
    columns = ['Model Name', 'Accuracy', 'Precision', 'Recall', 'FScore', '
Processing Time'],
    index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation
```

```
Out[24]: LogisticRegression(C=1000, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='lbfgs', tol=0.001, verbose=0, warm_start=False)
```

```
accuracy 0.6338
precision 0.6338
recall 0.6338
f1_score 0.6338
confusion matrix
 [[2004 1176]
 [1147 2016]]
process time 0.2633
```

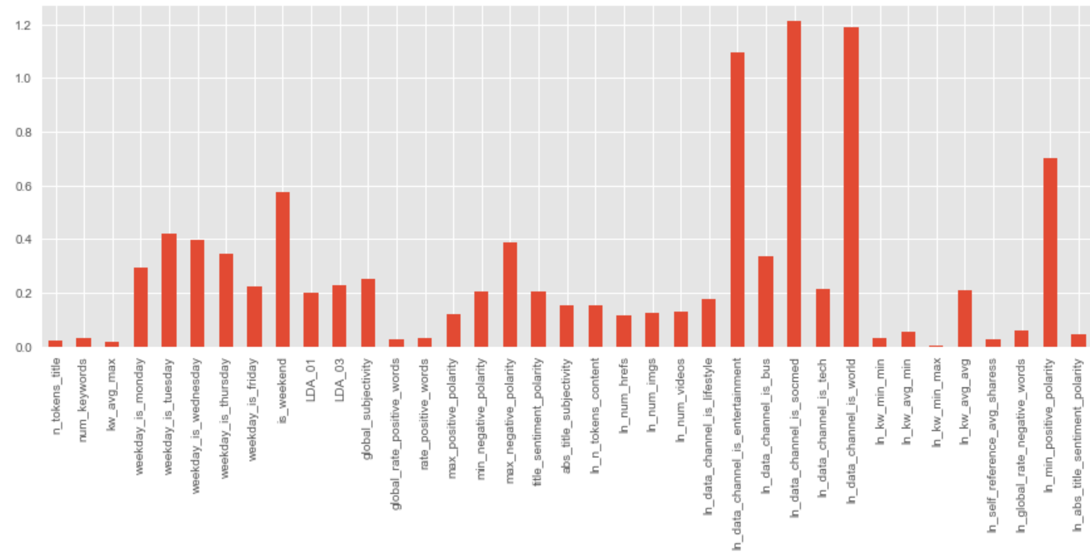
Interpreting Weights

In [25]:
```
# Interpreting weights
zip_vars = zip(sum(abs(lr_model1.coef_)).T,df_pop.columns) # combine attrib
utes
zip_vars = sorted(zip_vars)
for coef, name in zip_vars:
    print('\t%-35s - weight = %9.3f' % (name, coef)) # now print them out
```

```
	ln_kw_min_max                       - weight =     0.002
	kw_avg_max                          - weight =     0.017
	n_tokens_title                      - weight =     0.020
	ln_self_reference_avg_sharess       - weight =     0.026
	global_rate_positive_words          - weight =     0.028
	rate_positive_words                 - weight =     0.028
	num_keywords                        - weight =     0.029
	ln_kw_min_min                       - weight =     0.032
	ln_abs_title_sentiment_polarity     - weight =     0.044
	ln_kw_avg_min                       - weight =     0.056
	ln_global_rate_negative_words       - weight =     0.057
	ln_num_hrefs                        - weight =     0.114
	max_positive_polarity               - weight =     0.120
	ln_num_imgs                         - weight =     0.123
	ln_num_videos                       - weight =     0.129
	abs_title_subjectivity              - weight =     0.152
	ln_n_tokens_content                 - weight =     0.152
	ln_data_channel_is_lifestyle        - weight =     0.176
	LDA_01                              - weight =     0.199
	title_sentiment_polarity            - weight =     0.203
	min_negative_polarity               - weight =     0.204
	ln_kw_avg_avg                       - weight =     0.209
	ln_data_channel_is_tech             - weight =     0.214
	weekday_is_friday                   - weight =     0.224
	LDA_03                              - weight =     0.226
	global_subjectivity                 - weight =     0.253
	weekday_is_monday                   - weight =     0.291
	ln_data_channel_is_bus              - weight =     0.335
	weekday_is_thursday                 - weight =     0.346
	max_negative_polarity               - weight =     0.389
	weekday_is_wednesday                - weight =     0.396
	weekday_is_tuesday                  - weight =     0.419
	is_weekend                          - weight =     0.575
	ln_min_positive_polarity            - weight =     0.704
	ln_data_channel_is_entertainment    - weight =     1.095
	ln_data_channel_is_world            - weight =     1.190
	ln_data_channel_is_socmed           - weight =     1.215
```

In [26]:
```
%matplotlib inline
rcParams['figure.figsize'] = 15, 5
plt.style.use('ggplot')

weights = pd.Series(sum(abs(lr_model1.coef_)), index = df_pop.columns)
weights.plot(kind = 'bar')
plt.show()
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x11e2818d0>



Above is a visual representaion of the magnitudes of the coefficients.

To interpret the weights of each variable, I used the sums of the absolute values of the coefficients of each variable for each class. Because a particular variable might be highly positively predictive of one class and highly negatively predictive of another class, their sums would appear to have little value. I sum the absolute values to measure the total predictive value across all classes.

It does not surprise me to see that Social Media data channel is the most predictive, since social media by nature involves sharing with connections, followed by World data channel and positive polarity. Overall, data channels social media, world, and entertainment are more predictive of the popularity.

b. Decision Tree Classifier using Grid Search

Grid search parameter set-up

In [27]:
```
# Applying Grid Search to find the best model and the best parameters

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

DTclassifier = DecisionTreeClassifier(criterion = 'entropy', random_state =
0)

parameters = [
      {
        'criterion': ['gini'],
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 100, 1000],
        'min_samples_leaf': [1, 10, 100],
        'max_features': [None],
        'max_leaf_nodes': [None]
      },
      {
          'criterion': ['entropy'],
          'max_depth': [None, 5, 10],
          'min_samples_split': [2, 100, 1000],
          'min_samples_leaf': [1, 10, 100],
          'max_leaf_nodes': [None]
      }
     ]

grid_search = GridSearchCV(estimator = DTclassifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 3,
                           n_jobs = -1)

grid_search = grid_search.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_accuracy
best_parameters

best_criterion = best_parameters['criterion']
best_max_depth = best_parameters['max_depth']
best_max_leaf_nodes = best_parameters['max_leaf_nodes']
best_min_samples_leaf = best_parameters['min_samples_leaf']
best_min_samples_split = best_parameters['min_samples_split']
```

Out[27]:
```
0.63947658836512689
```

Out[27]:
```
{'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_samples_leaf': 100,
 'min_samples_split': 1000}
```

Best parameters for Decision Tree

In [28]:
```
best_accuracy
best_parameters
```

Out[28]:
```
0.63947658836512689
```

Out[28]:
```
{'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_samples_leaf': 100,
 'min_samples_split': 1000}
```

use best parameters to create best Decision Tree model for further analysis and model comparison

In [29]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,classification_report

tic = time.clock()

# train and fit

DTclassifier = DecisionTreeClassifier(
    criterion = best_criterion,
    min_samples_leaf = best_min_samples_leaf,
    min_samples_split = best_min_samples_split,
    max_leaf_nodes = best_max_leaf_nodes,
    max_depth = best_max_depth)

DTclassifier.fit(X_train, y_train)
y_predDT = DTclassifier.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predDT))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predDT,avera
ge='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predDT,average='we
ighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predDT,average='weig
hted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predDT))
print('process time',exetime)
print("\n")

# save statistics for model comparison

raw_data = {
    'Model Name':'Decision Tree Classifier',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Decision Tree Classifier', DTclassifier))
```

```
Out[29]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=100, min_samples_split=1000,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')
```

```
accuracy 0.6335
precision 0.6343
recall 0.6335
f1_score 0.6330
confusion matrix
 [[1898 1282]
 [1043 2120]]
process time 0.2592
```

Interpretation of importances.

In [30]:
```
# Interpreting weights

zip_varsDT = zip(DTclassifier.feature_importances_.T,df_pop.columns) # comb
ine attributes

zip_varsDT = sorted(zip_varsDT)

for importance, name in zip_varsDT:
    print('\t%-35s - weight = %9.3f' % ( name, importance)) # now print the
m out
```

```
	abs_title_subjectivity              - weight =     0.000
	ln_abs_title_sentiment_polarity     - weight =     0.000
	ln_data_channel_is_bus              - weight =     0.000
	ln_global_rate_negative_words       - weight =     0.000
	ln_num_videos                       - weight =     0.000
	max_negative_polarity               - weight =     0.000
	max_positive_polarity               - weight =     0.000
	min_negative_polarity               - weight =     0.000
	num_keywords                        - weight =     0.000
	weekday_is_monday                   - weight =     0.000
	weekday_is_thursday                 - weight =     0.000
	weekday_is_tuesday                  - weight =     0.000
	weekday_is_wednesday                - weight =     0.000
	title_sentiment_polarity            - weight =     0.002
	LDA_03                              - weight =     0.002
	n_tokens_title                      - weight =     0.003
	rate_positive_words                 - weight =     0.004
	ln_data_channel_is_lifestyle        - weight =     0.004
	weekday_is_friday                   - weight =     0.005
	global_rate_positive_words          - weight =     0.006
	ln_num_hrefs                        - weight =     0.007
	ln_kw_min_min                       - weight =     0.008
	LDA_01                              - weight =     0.009
	global_subjectivity                 - weight =     0.010
	ln_kw_avg_min                       - weight =     0.015
	ln_data_channel_is_world            - weight =     0.016
	ln_num_imgs                         - weight =     0.019
	ln_n_tokens_content                 - weight =     0.019
	ln_min_positive_polarity            - weight =     0.020
	ln_kw_min_max                       - weight =     0.023
	kw_avg_max                          - weight =     0.042
	ln_data_channel_is_socmed           - weight =     0.070
	ln_data_channel_is_tech             - weight =     0.086
	ln_self_reference_avg_sharess       - weight =     0.090
	ln_data_channel_is_entertainment    - weight =     0.097
	is_weekend                          - weight =     0.126
	ln_kw_avg_avg                       - weight =     0.319
```

In the decision tree model, the importance of the variables are not the same as they were in the logistic regression model examined earlier. The amount of key words is most important, next are self reference average shares. This result is interesting and maybe not intuitive, but variables such as the weekend indicator and variables related to the actual sharing process such as keywords show up here.

c. Random Forest Classifier

Grid Search parameter set-up for Random Forest classifier

In [31]:
```
RFclf = RandomForestClassifier(
    criterion = 'entropy',
    max_features= 'sqrt',
    max_depth = 5,
    n_estimators = 10,
    n_jobs = -1)

#RFclf.fit(master_X_train, master_y_train)

param_grid =[
    {
    'criterion': ['gini'],
    'n_estimators': [100, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 50]
},
 {
    'criterion': ['entropy'],
    'n_estimators': [100, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 50]
}
]

RF_grid_search = GridSearchCV(
    estimator = RFclf,
    param_grid = param_grid,
    cv = 3)

grid_search = RF_grid_search.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_criterion = best_parameters['criterion']
best_max_depth = best_parameters['max_depth']
best_max_features = best_parameters['max_features']
best_n_estimators = best_parameters['n_estimators']
```

best parameters for Random Forest Classifier

In [32]:
```
best_accuracy
best_parameters
```

Out[32]:
```
0.66558410846602556
```

Out[32]:
```
{'criterion': 'entropy',
 'max_depth': 20,
 'max_features': 'sqrt',
 'n_estimators': 500}
```

using best parameters for main model for further analysis and model comparison

In [33]:
```
from sklearn.ensemble import RandomForestClassifier

tic = time.clock()

# train and test

RFclf = RandomForestClassifier(
    criterion = best_criterion,
    max_depth = best_max_depth,
    max_features = best_max_features,
    n_estimators = best_n_estimators,
    n_jobs = -1)

RFclf.fit(X_train, y_train)
y_predRF = RFclf.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predRF))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predRF, aver
age ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predRF, average =
'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predRF, average = 'w
eighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predRF))
print('process time',exetime)
print("\n")


# save statistics for model comparison

raw_data = {
    'Model Name':'Random Forest Classifier',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Random Forest Classifier', RFclf))
```

```
Out[33]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entrop
         y',
                     max_depth=20, max_features='sqrt', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=-1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

         accuracy 0.6636
         precision 0.6636
         recall 0.6636
         f1_score 0.6635
         confusion matrix
          [[2148 1032]
          [1102 2061]]
         process time 53.9133
```

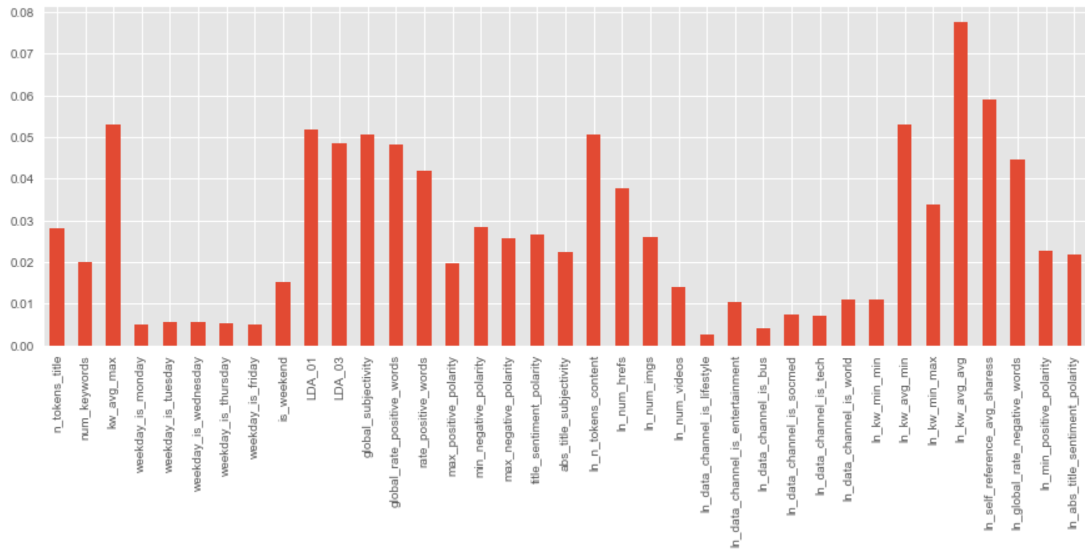
Interpreting weights

In [34]:
```
# Interpreting weights
zip_varsRF = zip(RFclf.feature_importances_.T,df_pop.columns) # combine att
ributes
zip_varsRF = sorted(zip_varsRF)
for importance, name in zip_varsDT:
    print('\t%-35s - weight = %9.3f' % ( name, importance)) # now print the
m out
```

```
	abs_title_subjectivity              - weight =     0.000
	ln_abs_title_sentiment_polarity     - weight =     0.000
	ln_data_channel_is_bus              - weight =     0.000
	ln_global_rate_negative_words       - weight =     0.000
	ln_num_videos                       - weight =     0.000
	max_negative_polarity               - weight =     0.000
	max_positive_polarity               - weight =     0.000
	min_negative_polarity               - weight =     0.000
	num_keywords                        - weight =     0.000
	weekday_is_monday                   - weight =     0.000
	weekday_is_thursday                 - weight =     0.000
	weekday_is_tuesday                  - weight =     0.000
	weekday_is_wednesday                - weight =     0.000
	title_sentiment_polarity            - weight =     0.002
	LDA_03                              - weight =     0.002
	n_tokens_title                      - weight =     0.003
	rate_positive_words                 - weight =     0.004
	ln_data_channel_is_lifestyle        - weight =     0.004
	weekday_is_friday                   - weight =     0.005
	global_rate_positive_words          - weight =     0.006
	ln_num_hrefs                        - weight =     0.007
	ln_kw_min_min                       - weight =     0.008
	LDA_01                              - weight =     0.009
	global_subjectivity                 - weight =     0.010
	ln_kw_avg_min                       - weight =     0.015
	ln_data_channel_is_world            - weight =     0.016
	ln_num_imgs                         - weight =     0.019
	ln_n_tokens_content                 - weight =     0.019
	ln_min_positive_polarity            - weight =     0.020
	ln_kw_min_max                       - weight =     0.023
	kw_avg_max                          - weight =     0.042
	ln_data_channel_is_socmed           - weight =     0.070
	ln_data_channel_is_tech             - weight =     0.086
	ln_self_reference_avg_sharess       - weight =     0.090
	ln_data_channel_is_entertainment    - weight =     0.097
	is_weekend                          - weight =     0.126
	ln_kw_avg_avg                       - weight =     0.319
```

In [35]:
```
%matplotlib inline
rcParams['figure.figsize'] = 15, 5
plt.style.use('ggplot')

weights = pd.Series(abs(RFclf.feature_importances_), index = df_pop.columns
)
weights.plot(kind = 'bar')
plt.show()
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x117e17ac8>



In the random forest model, the importance of the variables are not the same as they were in the logistic regression model examined earlier but are very similar to the decision tree model. The amount of key words is most important, next are self reference average shares. This result is interesting and maybe not intuitive, but variables such as the weekend indicator and variables related to the actual sharing process such as keywords show up here.

d: Naive Bayes

d.1 Multinomial Naive Bayes

In [36]:
```
from sklearn.naive_bayes import MultinomialNB

MNBclf = MultinomialNB(
    alpha = 0.01,
    class_prior = None,
    fit_prior = True)

params = {
    'alpha':[0.1, 0.5, 1.0]
}

MNB_grid_search = GridSearchCV(
    MNBclf,
    params,
    cv = 3)

grid_search = MNB_grid_search.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_accuracy
best_parameters

best_alpha = best_parameters['alpha']
```

Out[36]: 0.61961217089705189

Out[36]: {'alpha': 0.5}

In [37]:
```
tic = time.clock()

# train and test

MNBclf = MultinomialNB(
    alpha = best_alpha,
    class_prior = None,
    fit_prior = True)

MNBclf.fit(X_train, y_train)
y_predMNB = MNBclf.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predMNB))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predMNB, ave
rage ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predMNB, average =
'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predMNB, average = '
weighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predMNB))
print('process time',exetime)
print("\n")


# save statistics for model comparison

raw_data = {
    'Model Name':'Multinomial Naïve Bayes',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Multinomial Naïve Bayes', MNBclf))
```

```
Out[37]: MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True)

accuracy 0.6177
precision 0.6184
recall 0.6177
f1_score 0.6170
confusion matrix
 [[2100 1080]
 [1345 1818]]
process time 0.0176
```

Interpreting weights

```
In [41]: # Interpreting weights
zip_varsMNB = zip(sum(abs(MNBclf.coef_.T)),df_pop.columns) # combine attrib
utes
zip_varsMNB = sorted(zip_varsMNB)
for coef, name in zip_varsDT:
    print('\t%-35s - weight = %9.3f' % ( name, coef)) # now print them out
```

```
        abs_title_subjectivity              - weight =     0.000
        ln_abs_title_sentiment_polarity     - weight =     0.000
        ln_data_channel_is_bus              - weight =     0.000
        ln_global_rate_negative_words       - weight =     0.000
        ln_num_videos                       - weight =     0.000
        max_negative_polarity               - weight =     0.000
        max_positive_polarity               - weight =     0.000
        min_negative_polarity               - weight =     0.000
        num_keywords                        - weight =     0.000
        weekday_is_monday                   - weight =     0.000
        weekday_is_thursday                 - weight =     0.000
        weekday_is_tuesday                  - weight =     0.000
        weekday_is_wednesday                - weight =     0.000
        title_sentiment_polarity            - weight =     0.002
        LDA_03                              - weight =     0.002
        n_tokens_title                      - weight =     0.003
        rate_positive_words                 - weight =     0.004
        ln_data_channel_is_lifestyle        - weight =     0.004
        weekday_is_friday                   - weight =     0.005
        global_rate_positive_words          - weight =     0.006
        ln_num_hrefs                        - weight =     0.007
        ln_kw_min_min                       - weight =     0.008
        LDA_01                              - weight =     0.009
        global_subjectivity                 - weight =     0.010
        ln_kw_avg_min                       - weight =     0.015
        ln_data_channel_is_world            - weight =     0.016
        ln_num_imgs                         - weight =     0.019
        ln_n_tokens_content                 - weight =     0.019
        ln_min_positive_polarity            - weight =     0.020
        ln_kw_min_max                       - weight =     0.023
        kw_avg_max                          - weight =     0.042
        ln_data_channel_is_socmed           - weight =     0.070
        ln_data_channel_is_tech             - weight =     0.086
        ln_self_reference_avg_sharess       - weight =     0.090
        ln_data_channel_is_entertainment    - weight =     0.097
        is_weekend                          - weight =     0.126
        ln_kw_avg_avg                       - weight =     0.319
```

For the multinomial naive bayes classifier, numbers of keywords and if it is posted on a weekend are the most predictive of popularity.

d.2 Gaussian Naive Bayes

In [42]:
```
from sklearn.naive_bayes import GaussianNB

tic = time.clock()

# train and test

GNBclf = GaussianNB()

GNBclf.fit(X_train, y_train)
y_predGNB = GNBclf.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predGNB))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predGNB, ave
rage ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predGNB, average =
'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predGNB, average = '
weighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predGNB))
print('process time',exetime)
print("\n")


# save statistics for model comparison

raw_data = {
    'Model Name':'Gaussian Naïve Bayes',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Gaussian Naïve Bayes', GNBclf))
```

```
Out[42]: GaussianNB(priors=None)

         accuracy 0.6262
         precision 0.6272
         recall 0.6262
         f1_score 0.6254
         confusion matrix
          [[2141 1039]
          [1332 1831]]
         process time 0.0303
```

Interpreting weights

```
In [44]: # Interpreting weights
         zip_varsGNB = zip(sum(abs(GNBclf.theta_.T)),df_pop.columns) # combine attri
         butes
         zip_varsGNB = sorted(zip_varsGNB)
         for theta, name in zip_varsDT:
             print('\t%-35s - weight = %9.3f' % ( name, theta)) # now print them out
```

```
        abs_title_subjectivity              - weight =     0.000
        ln_abs_title_sentiment_polarity     - weight =     0.000
        ln_data_channel_is_bus              - weight =     0.000
        ln_global_rate_negative_words       - weight =     0.000
        ln_num_videos                       - weight =     0.000
        max_negative_polarity               - weight =     0.000
        max_positive_polarity               - weight =     0.000
        min_negative_polarity               - weight =     0.000
        num_keywords                        - weight =     0.000
        weekday_is_monday                   - weight =     0.000
        weekday_is_thursday                 - weight =     0.000
        weekday_is_tuesday                  - weight =     0.000
        weekday_is_wednesday                - weight =     0.000
        title_sentiment_polarity            - weight =     0.002
        LDA_03                              - weight =     0.002
        n_tokens_title                      - weight =     0.003
        rate_positive_words                 - weight =     0.004
        ln_data_channel_is_lifestyle        - weight =     0.004
        weekday_is_friday                   - weight =     0.005
        global_rate_positive_words          - weight =     0.006
        ln_num_hrefs                        - weight =     0.007
        ln_kw_min_min                       - weight =     0.008
        LDA_01                              - weight =     0.009
        global_subjectivity                 - weight =     0.010
        ln_kw_avg_min                       - weight =     0.015
        ln_data_channel_is_world            - weight =     0.016
        ln_num_imgs                         - weight =     0.019
        ln_n_tokens_content                 - weight =     0.019
        ln_min_positive_polarity            - weight =     0.020
        ln_kw_min_max                       - weight =     0.023
        kw_avg_max                          - weight =     0.042
        ln_data_channel_is_socmed           - weight =     0.070
        ln_data_channel_is_tech             - weight =     0.086
        ln_self_reference_avg_sharess       - weight =     0.090
        ln_data_channel_is_entertainment    - weight =     0.097
        is_weekend                          - weight =     0.126
        ln_kw_avg_avg                       - weight =     0.319
```

Modeling and Evaluation 4

Evaluation metrics

In [45]:
```
# converting acc, pre, recall, fscore and time to numeric values for plots

comparison_tbl = comparison_tbl.reset_index(drop=True)
comparison_tbl['Precision'] = pd.to_numeric(comparison_tbl['Precision'])
comparison_tbl['Accuracy'] = pd.to_numeric(comparison_tbl['Accuracy'])
comparison_tbl['FScore']= pd.to_numeric(comparison_tbl['FScore'])
comparison_tbl['Processing Time'] = pd.to_numeric(comparison_tbl['Processin
g Time'])
comparison_tbl['Recall'] = pd.to_numeric(comparison_tbl['Recall'])


comparison_tbl
```

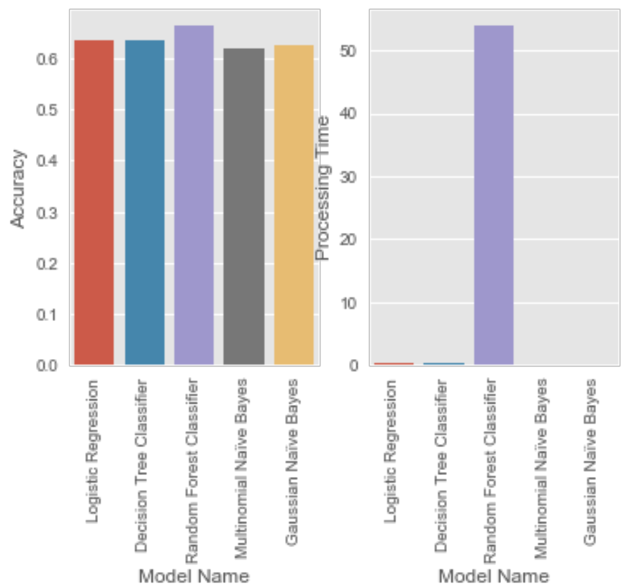
Out[45]:

	Model Name	Accuracy	Precision	Recall	FScore	Processing Time
0	Logistic Regression	0.6338	0.6338	0.6338	0.6338	0.2633
1	Decision Tree Classifier	0.6335	0.6343	0.6335	0.6330	0.2592
2	Random Forest Classifier	0.6636	0.6636	0.6636	0.6635	53.9133
3	Multinomial Naïve Bayes	0.6177	0.6184	0.6177	0.6170	0.0176
4	Gaussian Naïve Bayes	0.6262	0.6272	0.6262	0.6254	0.0303

Visualization of metrics

In [46]:
```
from pylab import rcParams
%matplotlib inline

fig, axs = plt.subplots(ncols = 2)
plt.setp(axs[0].xaxis.get_majorticklabels(), rotation = 90 )
plt.setp(axs[1].xaxis.get_majorticklabels(), rotation = 90 )
sns.barplot(x = 'Model Name', y = 'Accuracy', data = comparison_tbl, ax = axs[0])
sns.barplot(data = comparison_tbl, y = 'Processing Time', x = 'Model Name', ax = axs[1])
```

```
Out[46]: [None, None, None, None, None, None, None, None, None, None, None, None]

Out[46]: [None, None, None, None, None, None, None, None, None, None, None, None]

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x14b477128>

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x11bc84550>
```



10-K Cross-Validation for each classifier

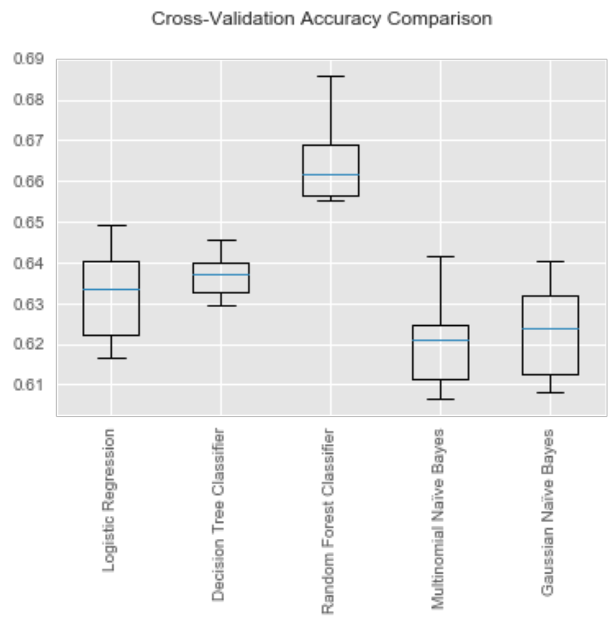
For each classifier we run 10 fold cross validation which will help us narrow down one final model

In [47]:
```
from sklearn import model_selection

# evaluate each model in turn

results = []
names = []
scoring = 'accuracy'

for name, model in models:
    cv_results = model_selection.cross_val_score(
        model,
        master_X_train,
        master_y_train,
        cv = 10,
        scoring = scoring)
    results.append(cv_results)
    names.append(name)
    msg = "\t%-35s mean (std dev): %.3f (%.4f)" % (name, cv_results.mean(),
cv_results.std())
    #print(cv_results)
    print(msg)
```

```
	Logistic Regression                mean (std dev): 0.633 (0.0109)
	Decision Tree Classifier           mean (std dev): 0.638 (0.0083)
	Random Forest Classifier           mean (std dev): 0.664 (0.0092)
	Multinomial Naïve Bayes            mean (std dev): 0.621 (0.0107)
	Gaussian Naïve Bayes               mean (std dev): 0.624 (0.0113)
```

In [48]:
```
# boxplot accuracy comparison

fig = plt.figure()
fig.suptitle('Cross-Validation Accuracy Comparison')
ax = fig.add_subplot(111)

plt.boxplot(results)

ax.set_xticklabels(names, rotation=90)
plt.show()
```

```
Out[48]: <matplotlib.text.Text at 0x11bd437f0>

Out[48]: {'boxes': [<matplotlib.lines.Line2D at 0x11bb273c8>,
           <matplotlib.lines.Line2D at 0x11bb3ffd0>,
           <matplotlib.lines.Line2D at 0x11bb5e470>,
           <matplotlib.lines.Line2D at 0x11bb7c5c0>,
           <matplotlib.lines.Line2D at 0x11bb9e0b8>],
          'caps': [<matplotlib.lines.Line2D at 0x11bb31e48>,
           <matplotlib.lines.Line2D at 0x11bb396d8>,
           <matplotlib.lines.Line2D at 0x11bb4f978>,
           <matplotlib.lines.Line2D at 0x11bb4fb70>,
           <matplotlib.lines.Line2D at 0x11bb66dd8>,
           <matplotlib.lines.Line2D at 0x11bb6fcc0>,
           <matplotlib.lines.Line2D at 0x11bb83f28>,
           <matplotlib.lines.Line2D at 0x11bb8b7b8>,
           <matplotlib.lines.Line2D at 0x14b8914e0>,
           <matplotlib.lines.Line2D at 0x14b4e2390>],
          'fliers': [<matplotlib.lines.Line2D at 0x11bb3f7b8>,
           <matplotlib.lines.Line2D at 0x11bb57c50>,
           <matplotlib.lines.Line2D at 0x11bb76da0>,
           <matplotlib.lines.Line2D at 0x11bb94898>,
           <matplotlib.lines.Line2D at 0x11bb9eeb8>],
          'means': [],
          'medians': [<matplotlib.lines.Line2D at 0x11bb398d0>,
           <matplotlib.lines.Line2D at 0x11bb57400>,
           <matplotlib.lines.Line2D at 0x11bb6feb8>,
           <matplotlib.lines.Line2D at 0x11bb8b9b0>,
           <matplotlib.lines.Line2D at 0x11b185940>],
          'whiskers': [<matplotlib.lines.Line2D at 0x11bb27d68>,
           <matplotlib.lines.Line2D at 0x11bb27f98>,
           <matplotlib.lines.Line2D at 0x11bb48898>,
           <matplotlib.lines.Line2D at 0x11bb48a90>,
           <matplotlib.lines.Line2D at 0x11bb5ecf8>,
           <matplotlib.lines.Line2D at 0x11bb66be0>,
           <matplotlib.lines.Line2D at 0x11bb7ce48>,
           <matplotlib.lines.Line2D at 0x11bb836d8>,
           <matplotlib.lines.Line2D at 0x14b4c9dd8>,
           <matplotlib.lines.Line2D at 0x14b4cf6a0>]}

Out[48]: [<matplotlib.text.Text at 0x14b5011d0>,
          <matplotlib.text.Text at 0x11bd942e8>,
          <matplotlib.text.Text at 0x11bba7e48>,
          <matplotlib.text.Text at 0x11bbab940>,
          <matplotlib.text.Text at 0x11bbb2438>]
```



Holdout test data set prediction with our final model

Our final best model is Random Forest Classifier. We will run prediction on that fit with the test data set we set aside at the beggining of the project. We will calculate statistics for the prediction.

In [54]:
```
y_predFinal = RFclf.predict(golden_X_test)

# calculate statistics

accuracy = metrics.accuracy_score(golden_y_test, y_predFinal)
precision = metrics.precision_score(golden_y_test, y_predFinal, average ='w
eighted')
recall = metrics.recall_score(golden_y_test, y_predFinal, average = 'weight
ed')
f1_score = metrics.f1_score(golden_y_test, y_predFinal, average = 'weighted
')
toc =  time.clock()
exetime = toc-tic

# print statistics
print ("\tSelected Model Validation on Hold-Out Data Set")
print ("\t\tAccuracy ....... = %9.3f" % (accuracy))
print ("\t\tPrecision ...... = %9.3f"% (precision))
print ("\t\tRecall ......... = %9.3f" % (recall ))
print ("\t\tF1_score ....... = %9.3f\n" % (f1_score))
print ("\t\tConfusion matrix = \n", confusion_matrix(y_test, y_predRF))
print ("\t\tProcess time ... = %9.3f" % (exetime))
print ("\n")
```

```
        Selected Model Validation on Hold-Out Data Set
                Accuracy ....... =     0.661
                Precision ...... =     0.661
                Recall ......... =     0.661
                F1_score ....... =     0.661

                Confusion matrix = 
 [[2148 1032]
 [1102 2061]]
                Process time ... =   630.403
```

Model Validation

- For Task 1 - binary classifier for **popularity** of *mashable* articles, we selected the Random Forest classifier as having the best overall metrics from the 10-fold cross-validation.
- We then deploy that recommended model on our 20% Hold-Out data set - the data set that was not previously used in any of the model development.
- The results of the validation on the Hold-Out data set show that similar accuracy, precision, and recall values are achieved on this data set as were achieved from the 10-fold cross-validated model - thus, we have good confidence that the recommended model is sufficiently generalizable for additional data sets (of similar population characteristics) to be a useful model for future predictions of article popularity.

<hr size = "5">

Task 02 - Second Classifier

<hr size = "5">

To the reader :

This next section is data preparation for the Task 2 - classifier for assigning an article to a data_channel. The method is very similar to what was employed for Task 1, but the steps here need to be repeated since the data columns retained are different between the 2 classification tasks.

The next several sections are similar in structure and content to what was shown above, but here the data reduction and cleaning is now done specifically for the 2nd classification task.

To avoid reviewing the repetitive nature of this section, feel free to skip to the section beginning with **Modeling & Evaluation 2** - which begins the work that is dedicated to developing this 2nd classification model.

Data Prep 01

Similar methods and processes as constructed for Task 1, with differences related to different data set content specific to Task 2 classification model.

Data Prep 02

Similar methods and processes as constructed for Task 1, with differences related to different data set content specific to Task 2 classification model.

Modeling and Evaluation 01

Choose and explain the evaluation metrics that we will use. Why are the measures appropriate for analyzing the results of this model ? Give a detailed explanation backing up any assertions.

Read in dataset from .csv file

In [3]:
```
data_dir = 'data/'
data_file = 'OnlineNewsPopularity.csv'

file_2_read = data_dir + data_file
df = pd.read_csv(file_2_read)
```

In [4]:
```
df.columns = df.columns.str.strip()
col_names = df.columns.values.tolist()
```

Data Preparation Part 1

Define and prepare your class variables. Use proper variable representations (int, float, one-hot, etc.). Use pre-processing methods (as needed) for dimensionality reduction, scaling, etc. Remove variables that are not needed/useful for the analysis.

Task 2 data set definition

- For Task 2 classification we will classify the articles according to which data channel they are most likely to belong. The business case for this is to support directing the article to the data_channel most appropriate for the article content.
- In order to support this classification task, we create a new dependent variable column **data_channel** which combines all of the individual binary boolean columns of **data_channel_is_xxx** to **data_channel** column with appropriate value
- We create 2 sets of this dependent variable, **data_channel** and **data_channel_n** . The only difference between these 2 columns is that **data_channel** contains the text values for data channel category while the **data_channel_n** contains an integer classifier (1 --> 7) which we associate to the text description in alphabetic order. We create 2 versions of the column to be able to use the text version, when feasible, and the integer version in the case that that is required for a particular classifier routine.
- There are approx 15% of the articles which contain no identified **data_channel** in the original data set. We create a new category, *Others*, for the articles without assignation to one of the standard data channels.

In [5]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... creating data_channel categorical variable
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df['data_channel'] = 'Others'

condition = df['data_channel_is_lifestyle'] == 1
df.loc[condition, 'data_channel'] = 'Lifestyle'

condition = df['data_channel_is_entertainment'] == 1
df.loc[condition, 'data_channel'] = 'Entertainment'

condition = df['data_channel_is_bus'] == 1
df.loc[condition, 'data_channel'] = 'Business'

condition = df['data_channel_is_socmed'] == 1
df.loc[condition, 'data_channel'] = 'Social Media'

condition = df['data_channel_is_tech'] == 1
df.loc[condition, 'data_channel'] = 'Technology'

condition = df['data_channel_is_world'] == 1
df.loc[condition, 'data_channel'] = 'World'

del df['data_channel_is_lifestyle']
del df['data_channel_is_entertainment']
del df['data_channel_is_bus']
del df['data_channel_is_socmed']
del df['data_channel_is_tech']
del df['data_channel_is_world']
```

In [6]:
```
df.data_channel.value_counts()
```

Out[6]:
```
World            8427
Technology       7346
Entertainment    7057
Business         6258
Others           6134
Social Media     2323
Lifestyle        2099
Name: data_channel, dtype: int64
```

In [7]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... integer value of categorical values for multinomial NB classification
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df['data_channel_n'] = 0

condition = df['data_channel'] == 'Business'
df.loc[condition, 'data_channel_n'] = 1

condition = df['data_channel'] == 'Entertainment'
df.loc[condition, 'data_channel_n'] = 2

condition = df['data_channel'] == 'Lifestyle'
df.loc[condition, 'data_channel_n'] = 3

condition = df['data_channel'] == 'Others'
df.loc[condition, 'data_channel_n'] = 4

condition = df['data_channel'] == 'Social Media'
df.loc[condition, 'data_channel_n'] = 5

condition = df['data_channel'] == 'Technology'
df.loc[condition, 'data_channel_n'] = 6

condition = df['data_channel'] == 'World'
df.loc[condition, 'data_channel_n'] = 7
```

Remove variables that are not useful

Delete shares from the Task 2 data set

shares is the Task 1 dependent variable

we are excluding it from the Task 2 dataset as per the business model this value is not available during data_channel selection

The business model being developed here is that an article is proposed for publication, a set of text processing routines will extract and develop the model features from the raw article. The next step is to deploy the **data_channel** assignment (classification) model, which is then also a necessary ingredient for the final model, Task 1, which is to estimate the **popularity** of the article and thus provide recommendation to publish or not to publish.

Thus, for this Task 2 data set, we exclude the **shares** data value.

In [8]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  shares is task 1 dependent variable
# ...  we are excluding it from this model as per business model this value
is not available
# ...  during data_channel prediction
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

del df['shares']
```

In [9]:
```
del df['n_non_stop_words']
del df['n_non_stop_unique_tokens']
del df['n_unique_tokens']
del df['url']
```

In [10]:
```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	
timedelta	39644.0	354.530471	214.163767	8.00000	164.000000	339.
n_tokens_title	39644.0	10.398749	2.114037	2.00000	9.000000	10.0
n_tokens_content	39644.0	546.514731	471.107508	0.00000	246.000000	409.
num_hrefs	39644.0	10.883690	11.332017	0.00000	4.000000	8.00
num_self_hrefs	39644.0	3.293638	3.855141	0.00000	1.000000	3.00
num_imgs	39644.0	4.544143	8.309434	0.00000	1.000000	1.00
num_videos	39644.0	1.249874	4.107855	0.00000	0.000000	0.00
average_token_length	39644.0	4.548239	0.844406	0.00000	4.478404	4.66
num_keywords	39644.0	7.223767	1.909130	1.00000	6.000000	7.00
kw_min_min	39644.0	26.106801	69.633215	-1.00000	-1.000000	-1.0
kw_max_min	39644.0	1153.951682	3857.990877	0.00000	445.000000	660.
kw_avg_min	39644.0	312.366967	620.783887	-1.00000	141.750000	235.
kw_min_max	39644.0	13612.354102	57986.029357	0.00000	0.000000	140
kw_max_max	39644.0	752324.066694	214502.129573	0.00000	843300.000000	843
kw_avg_max	39644.0	259281.938083	135102.247285	0.00000	172846.875000	244
kw_min_avg	39644.0	1117.146610	1137.456951	-1.00000	0.000000	102
kw_max_avg	39644.0	5657.211151	6098.871957	0.00000	3562.101631	435
kw_avg_avg	39644.0	3135.858639	1318.150397	0.00000	2382.448566	287
self_reference_min_shares	39644.0	3998.755396	19738.670516	0.00000	639.000000	120
self_reference_max_shares	39644.0	10329.212662	41027.576613	0.00000	1100.000000	280
self_reference_avg_sharess	39644.0	6401.697580	24211.332231	0.00000	981.187500	220
weekday_is_monday	39644.0	0.168020	0.373889	0.00000	0.000000	0.00
weekday_is_tuesday	39644.0	0.186409	0.389441	0.00000	0.000000	0.00
weekday_is_wednesday	39644.0	0.187544	0.390353	0.00000	0.000000	0.00
weekday_is_thursday	39644.0	0.183306	0.386922	0.00000	0.000000	0.00
weekday_is_friday	39644.0	0.143805	0.350896	0.00000	0.000000	0.00
weekday_is_saturday	39644.0	0.061876	0.240933	0.00000	0.000000	0.00
weekday_is_sunday	39644.0	0.069039	0.253524	0.00000	0.000000	0.00
is_weekend	39644.0	0.130915	0.337312	0.00000	0.000000	0.00
LDA_00	39644.0	0.184599	0.262975	0.00000	0.025051	0.03
LDA_01	39644.0	0.141256	0.219707	0.00000	0.025012	0.03
LDA_02	39644.0	0.216321	0.282145	0.00000	0.028571	0.04
LDA_03	39644.0	0.223770	0.295191	0.00000	0.028571	0.04
LDA_04	39644.0	0.234029	0.289183	0.00000	0.028574	0.04
global_subjectivity	39644.0	0.443370	0.116685	0.00000	0.396167	0.45

Assign certain variables to type integer, as appropriate

In [11]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  convert the data type to Integer
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

to_int = ['timedelta','n_tokens_title', 'n_tokens_content',
    'num_hrefs','num_self_hrefs', 'num_imgs', 'num_videos', 'num_keywords',
    'weekday_is_monday',
    'weekday_is_tuesday',
    'weekday_is_wednesday',
    'weekday_is_thursday',
    'weekday_is_friday',
    'weekday_is_saturday',
    'weekday_is_sunday',
    'is_weekend',
    'data_channel_n']


df[to_int] = df[to_int ].astype(np.int64)
```

In [12]:
```
df[df.duplicated()]
```

Out[12]:

	timedelta	n_tokens_title	n_tokens_content	num_hrefs	num_self_hrefs	num_imgs	num_videos

0 rows × 52 columns

Impute kw_avg_max for 0-values and re-scale to standard normal scale

- A small number of rows have 0 value for **kw_avg_max**, which is completely out of range for the remaining rows of this variable.
- We will impute these rows to median value of the column
- The magnitude of this column of data is markedly different than the range of values in the remaining columns in the data set. To bring this back in line, we will re-scale the values in this column to standard normal range

In [13]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  impute to median value for a few rows of kw_avg_max
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

condition = df['kw_avg_max'] == 0
df.loc[condition, 'kw_avg_max'] = df.kw_avg_max.median()


# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  scale to standard normal scale
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df.kw_avg_max = (df.kw_avg_max - df.kw_avg_max.mean()) / df.kw_avg_max.std(
)
```

Constant offset for variables with min value < 0

- This allows to consider these variables for ln() transform if highly right-skewed and also supports some classification models that only accept independent variables that are > 0
- Method here is to just add -1 * min_value of any column for which min_value < 0

In [14]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  for all columns with negative values, add +1 to all values in the co
lumn
# ...  - the only columns with negative values are polarity / sentiment mea
sures
# ...  - adding a constant to all values does not modify distributions
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df_numeric = df.select_dtypes(['number'])
numeric_col_names = df_numeric.columns.values.tolist()

# ... store min value for each column

df_mins = df.min()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  loop on each column, test for min < 0, add constant as applicable
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

for column in numeric_col_names :
    if df_mins[column] < 0 :
        df[column] = df[column] - df_mins[column]

#        print('--> min_value < 0 adjusted : ', column, df_mins[column])
```

Ln() transform for variables that are right skewed (skewness > 1)

- This facilitiates maintaining more normally distributed residuals for regression models
- Likely, this will not be needed for the classification task, at present, but also does not have negative effects for this current activity

In [15]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  ln() transform right skewed distribution variables (skewness > 1)
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df_numeric = df.select_dtypes(['number'])

numeric_col_names = df_numeric.columns.values.tolist()

# ... store min value for each column

df_mins = df.min()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  loop on each column, test for skewness, create new column if conditi
ons met
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

columns_to_drop = []

for column in numeric_col_names:
    sk = df[column].skew()

    if(sk > 1):
        new_col_name = 'ln_' + column
        print (column, sk, new_col_name)

        if df_mins[column] > 0:
            df[new_col_name] = np.log(df[column])
            columns_to_drop.append(column)

        elif df_mins[column] == 0:
            df_tmp = df[column] + 1
            df[new_col_name] = np.log(df_tmp)
            columns_to_drop.append(column)

        else:
            print('--> Ln() transform not completed -- skew > 1, but min va
lue < 0 :', column, '!!')


# ... delete tmp data

del df_tmp
del df_mins
del df_numeric

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  based on inspection, a few of these are just not valid ranges in ln(
) space
# ...  -- just delete these few back out of the data set
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

print (columns_to_drop)

del df['ln_LDA_00']
del df['ln_LDA_01']
del df['ln_LDA_02']
```

```
n_tokens_content 2.94542193879 ln_n_tokens_content
num_hrefs 4.0134948282 ln_num_hrefs
num_self_hrefs 5.17275110576 ln_num_self_hrefs
num_imgs 3.94659584465 ln_num_imgs
num_videos 7.0195327863 ln_num_videos
kw_min_min 2.37494728018 ln_kw_min_min
kw_max_min 35.3284337312 ln_kw_max_min
kw_avg_min 31.3061081027 ln_kw_avg_min
kw_min_max 10.3863716348 ln_kw_min_max
kw_max_avg 16.4116695554 ln_kw_max_avg
kw_avg_avg 5.76017729162 ln_kw_avg_avg
self_reference_min_shares 26.2643641603 ln_self_reference_min_shares
self_reference_max_shares 13.8708490494 ln_self_reference_max_shares
self_reference_avg_sharess 17.9140933777 ln_self_reference_avg_sharess
weekday_is_monday 1.77590824423 ln_weekday_is_monday
weekday_is_tuesday 1.61054706191 ln_weekday_is_tuesday
weekday_is_wednesday 1.60097097689 ln_weekday_is_wednesday
weekday_is_thursday 1.6370700483 ln_weekday_is_thursday
weekday_is_friday 2.03030483518 ln_weekday_is_friday
weekday_is_saturday 3.63708575997 ln_weekday_is_saturday
weekday_is_sunday 3.3999273763 ln_weekday_is_sunday
is_weekend 2.18850033431 ln_is_weekend
LDA_00 1.5674632332 ln_LDA_00
LDA_01 2.08672182342 ln_LDA_01
LDA_02 1.31169490203 ln_LDA_02
LDA_03 1.23871598638 ln_LDA_03
LDA_04 1.17312947598 ln_LDA_04
global_rate_negative_words 1.49191730919 ln_global_rate_negative_words
min_positive_polarity 3.04046773746 ln_min_positive_polarity
abs_title_sentiment_polarity 1.70419343991 ln_abs_title_sentiment_polarity
['n_tokens_content', 'num_hrefs', 'num_self_hrefs', 'num_imgs', 'num_videos
', 'kw_min_min', 'kw_max_min', 'kw_avg_min', 'kw_min_max', 'kw_max_avg', 'k
w_avg_avg', 'self_reference_min_shares', 'self_reference_max_shares', 'self
_reference_avg_sharess', 'weekday_is_monday', 'weekday_is_tuesday', 'weekda
y_is_wednesday', 'weekday_is_thursday', 'weekday_is_friday', 'weekday_is_sa
turday', 'weekday_is_sunday', 'is_weekend', 'LDA_00', 'LDA_01', 'LDA_02', '
LDA_03', 'LDA_04', 'global_rate_negative_words', 'min_positive_polarity', '
abs_title_sentiment_polarity']

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

Number of current columns in dataset : 69
```

Data Preparation Part 2

Data Selection - Task 2 - data_channel classification

- There are 60 columns in the original data set; we added a few additonal columns based on observed opportunities (e.g., _publication*date*, ...) as explained above.
- From this data set, we did a simple correlation matrix to look for variables that are highly correlated with each other that could be removed with little loss of information.
- With that downselection, we proceeded with additional evaluation of these remaining variables.
- we recognize that there is likely significant additional opportunity for modeling improvements with many of the remaining variables, and will look to re-expand the data set to further consider that with future work.

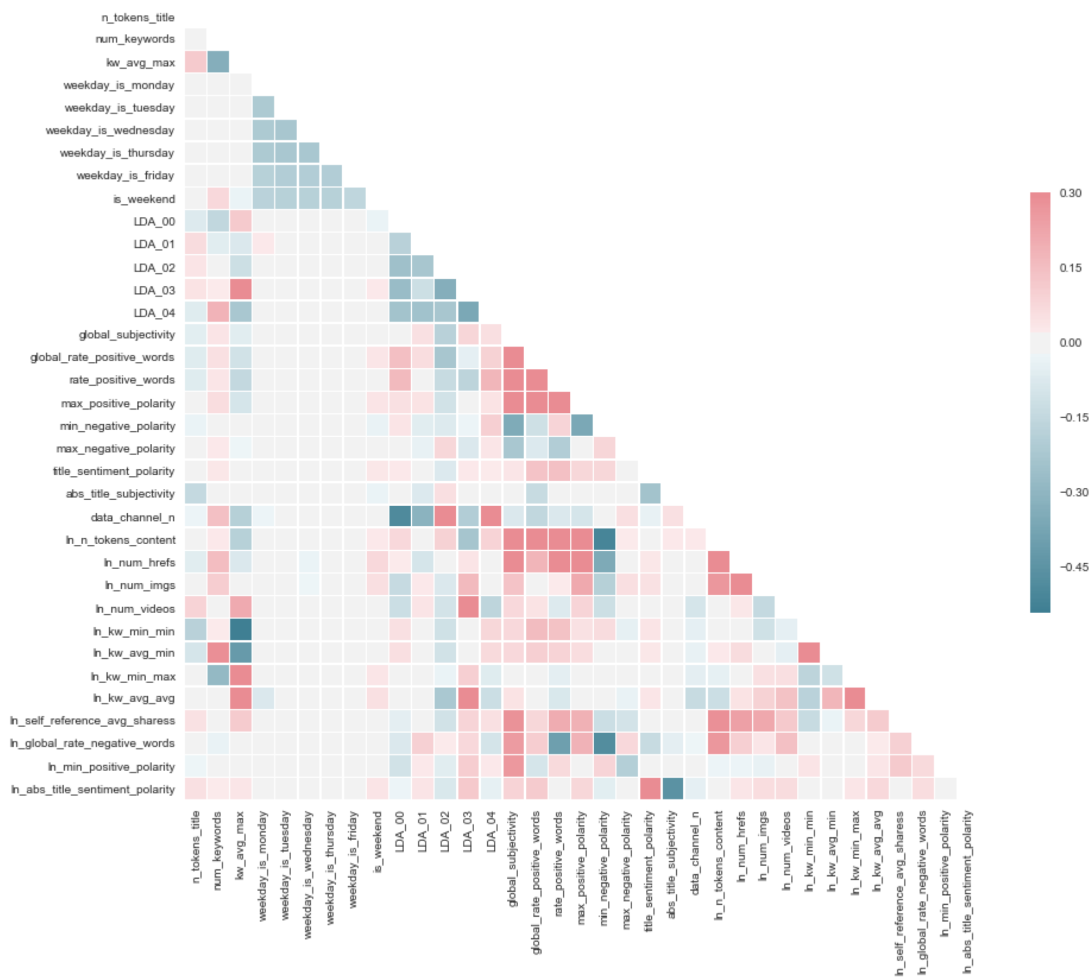
In [16]:

```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... display highest correlation pairs from corr() matrix
# ...
# ... https://stackoverflow.com/questions/17778394/list-highest-correlation
-pairs-from-a-large-correlation-matrix-in-pandas
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df_numeric = df.select_dtypes(['number'])

def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n = 5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels = labels_to_drop).sort_values(ascending =
False)
    return au_corr[0:n]

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... list out Top30 correlations
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

n_val = 30

top_30_corr_list = get_top_abs_correlations(df_numeric, n_val)
print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")
print("Top Absolute Correlations\n")
print(top_30_corr_list)


icor = 0
drop_column = list()
while (top_30_corr_list[icor] > 0.65):
    drop_column.append(top_30_corr_list[top_30_corr_list == top_30_corr_lis
t[icor]].index[0][0])
    icor += 1

drop_column = list(set(drop_column))

print("\n\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-")
print("Columns Recommended for removal based on correlation > 0.65")
print("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n")

print("\n".join(sorted(drop_column)))

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... drop one of the high correlation columns (2nd of the pair)
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

df = df.drop(drop_column, axis = 1)
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Top Absolute Correlations

ln_self_reference_max_shares  ln_self_reference_avg_sharess      0.994639
ln_self_reference_min_shares  ln_self_reference_avg_sharess      0.971014
ln_kw_max_min                 ln_kw_avg_min                      0.946087
ln_self_reference_min_shares  ln_self_reference_max_shares       0.945943
kw_min_avg                    ln_kw_min_max                      0.926784
ln_kw_max_avg                 ln_kw_avg_avg                      0.899409
timedelta                     ln_kw_min_min                      0.822783
kw_max_max                    ln_kw_min_min                      0.820625
rate_negative_words           ln_global_rate_negative_words      0.782517
average_token_length          ln_n_tokens_content                0.768795
avg_negative_polarity         min_negative_polarity              0.748896
title_subjectivity            ln_abs_title_sentiment_polarity    0.741229
global_sentiment_polarity     rate_positive_words                0.727827
avg_positive_polarity         max_positive_polarity              0.703558
weekday_is_sunday             is_weekend                         0.701648
ln_num_self_hrefs             ln_self_reference_max_shares       0.675339
weekday_is_saturday           is_weekend                         0.661707
global_sentiment_polarity     rate_negative_words                0.650717
max_positive_polarity         ln_n_tokens_content                0.643190
timedelta                     kw_max_max                         0.637824
ln_num_self_hrefs             ln_self_reference_avg_sharess      0.631820
global_subjectivity           avg_positive_polarity              0.631749
global_rate_positive_words    rate_positive_words                0.628626
ln_n_tokens_content           ln_num_hrefs                       0.614357
average_token_length          global_subjectivity                0.597629
LDA_02                        data_channel_n                     0.587449
avg_negative_polarity         max_negative_polarity              0.580108
average_token_length          rate_positive_words                0.578894
global_sentiment_polarity     global_rate_positive_words         0.570667
kw_max_max                    kw_avg_max                         0.552729
dtype: float64


-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Columns Recommended for removal based on correlation > 0.65
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

average_token_length
avg_negative_polarity
avg_positive_polarity
global_sentiment_polarity
kw_max_max
kw_min_avg
ln_kw_max_avg
ln_kw_max_min
ln_num_self_hrefs
ln_self_reference_max_shares
ln_self_reference_min_shares
rate_negative_words
timedelta
title_subjectivity
weekday_is_saturday
weekday_is_sunday


-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-


Top Absolute Correlations (2nd Pass)
```

In [17]:
```
sns.set(style="white")

# Compute the correlation matrix
corr = df.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(15, 13))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})


# from example found at https://www.kaggle.com/maheshdadhich/strength-of-vi
sualization-python-visuals-tutorial/notebook
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x108ad4da0>



Save cleaned / reduced data set to external .csv file

- This provides opportunity to just read in this .csv file and no need to repeat data cleaning / reduction process for each execution

In [18]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... store in ../data/ directory
# ... write as .csv file for future recall
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

data_dir = 'data/'
data_file = 'mashable_clean_dataset_for_lab_02_task_02.csv'

file_2_write = data_dir + data_file

df.to_csv(file_2_write, index = False)
```

Modeling and Evaluation 2

Holdout, Training and Test split

In [19]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  copy data frame to classification working data frame
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

# ... data set with text categorical target values

df_data_channel = df.copy()
del df_data_channel['data_channel_n']

# ... data set with integer categorical target values

df_data_channel_n = df.copy()
del df_data_channel_n['data_channel']

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  separate X and y matrices
# ...
# ...  convert to numpy matrices by calling 'values' on the pandas data fra
mes
# ...  they are now simple matrices for compatibility with scikit-learn
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

if 'data_channel' in df_data_channel:
    y = df_data_channel['data_channel'].values         # set 'data_channel'
as dependent
    del df_data_channel['data_channel']                # remove from datase
t
    X = df_data_channel.values                         # use everything els
e for independent EVs

if 'data_channel_n' in df_data_channel_n:
    y_n = df_data_channel_n['data_channel_n'].values    # set 'data_channel
' as dependent
    del df_data_channel_n['data_channel_n']             # remove from datas
et
    X_n = df_data_channel_n.values                      # use everything el
se for independent EVs

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ...  setup master train and test , golden traina and test
# ...  master sets - first 80% of original data set which will be base trai
ning for model building
# ...  Golden sets - 20% of original will be used in the final best model f
or prediction
# ...  split into training and test sets
# ....  --> 10 folds
# ...   --> 80% / 20% training / test
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
```

In [20]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... Set-up golden test data set
# ... This data-set will be used to evaluate the predictive capability of t
he final
# ... model on a data set that was not included in any of the prior train/t
est sets
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

num_cv_iterations = 1
cv_object = ShuffleSplit(n_splits = num_cv_iterations,
                         test_size  = 0.2)

print(cv_object)

for train_indices, test_indices in cv_object.split(X, y):
    master_X_train = X[train_indices]
    master_y_train = y[train_indices]
    golden_X_test  = X[test_indices]
    golden_y_test  = y[test_indices]
    print(master_X_train.shape)

for train_indices_n, test_indices_n in cv_object.split(X_n, y_n):
    master_X_train_n = X_n[train_indices_n]
    master_y_train_n = y_n[train_indices_n]
    golden_X_test_n  = X_n[test_indices_n]
    golden_y_test_n  = y_n[test_indices_n]
```

```
ShuffleSplit(n_splits=1, random_state=None, test_size=0.2, train_size=None)
(31715, 34)
```

In [21]:
```
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... Set-up training set to be used on 'best' model from grid search resul
ts
# ... This data-set will be used to verify 10-fold-CV-model has results con
sistent
# ... with the model produced from grid search
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

num_cv_iterations = 1
cv_object = ShuffleSplit(n_splits = num_cv_iterations,
                         test_size  = 0.2)

print(cv_object)

for train_indices, test_indices in cv_object.split(master_X_train, master_y
_train):
    X_train = master_X_train[train_indices]
    y_train = master_y_train[train_indices]
    X_test  = master_X_train[test_indices]
    y_test  = master_y_train[test_indices]
    print(X_train.shape)

for train_indices_n, test_indices_n in cv_object.split(master_X_train_n, ma
ster_y_train_n):
    X_train_n = master_X_train_n[train_indices_n]
    y_train_n = master_y_train_n[train_indices_n]
    X_test_n  = master_X_train_n[test_indices_n]
    y_test_n  = master_y_train_n[test_indices_n]
```

```
ShuffleSplit(n_splits=1, random_state=None, test_size=0.2, train_size=None)
(25372, 34)
```

In [22]:
```
# set required variables for model comparison

comparison_tbl = pd.DataFrame(columns = [
    'Model Name',
    'Accuracy',
    'Precision',
    'Recall',
    'FScore',
    'Processing Time'])

i_index=[]
i_index = 0

# preparation for cross validation and model comparison, each classifier is
appended once model is fit

models = []
```

Modeling and Evaluation 3

For task 2 we have chosen the following 4 models:
a. Multinomial logistic regression with parament selection using Grid Search
b. Decision Tree with parament selection using Grid Search
c. Random Forest with parament selection using Grid Search
d. Naive Bayes

Each of these models will be evaluated on Accuracy, Precision, Recall, FScore and Execution time

a. Multinomial logistic regression

For multinomial LR we have set standard attributes with: class_weight = balanced
multi_class = multinomial
search params:
tolerance parament tol
Regularization parament C

Grid selection for logistic regression

In [23]:
```
from sklearn.grid_search import GridSearchCV

lr_model = LogisticRegression(
    class_weight = 'balanced',
    multi_class = 'multinomial',
    solver = 'lbfgs',
    C = 10,
    tol = 0.1)

params = {
    'C':[100, 1000],
    'tol': [0.001, 0.0001]
}


clf = GridSearchCV(
    lr_model,
    params,
    scoring = 'neg_log_loss',
    refit = 'True',
    n_jobs = -1,
    cv = 3)

grid_search = clf.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
best_C = best_parameters['C']
best_tol = best_parameters['tol']
```

```
/Users/andrewabbott/.virtualenvs/dl4cv/lib/python3.6/site-packages/sklearn/
cross_validation.py:41: DeprecationWarning: This module was deprecated in v
ersion 0.18 in favor of the model_selection module into which all the refac
tored classes and functions are moved. Also note that the interface of the
new CV iterators are different from that of this module. This module will b
e removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/Users/andrewabbott/.virtualenvs/dl4cv/lib/python3.6/site-packages/sklearn/
grid_search.py:42: DeprecationWarning: This module was deprecated in versio
n 0.18 in favor of the model_selection module into which all the refactored
classes and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

***Best parameter values for logistic regression*:**

In [24]:
```
best_accuracy
best_parameters
```

Out[24]: -0.7604406218350708

Out[24]: {'C': 1000, 'tol': 0.001}

Create main logistic model using best paraments for further analysis and model comparisons

In [25]:
```
tic = time.clock()

# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=
# ... basic multiclass Logistic Regression
# ... - normalize features based on mean & stdev of each column
# ... -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=

lr_model1 = LogisticRegression(
    class_weight = 'balanced',
    multi_class = 'multinomial',
    solver = 'lbfgs',
    C = best_C,
    tol = best_tol)

lr_model1.fit(X_train, y_train)  # train object

y_hat = lr_model1.predict(X_test) # get test set precitions

toc =  time.clock()

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_hat))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_hat,average=
'weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_hat,average='weigh
ted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_hat,average='weighte
d'))

exetime = '{0:.4f}'.format(toc-tic)

# print statistics

print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", conf(y_test, y_hat))
print('process time',exetime)
print("\n")

lr_model1_confusion_matrix = conf(y_test, y_hat)

# save statistics for model comparison

raw_data = {
    'Model Name' : 'Logistic Regression',
    'Accuracy' : accuracy,
    'Precision' : precision,
    'Recall' : recall,
    'FScore' : f1_score,
    'Processing Time' : exetime
}

df_tbl = pd.DataFrame(raw_data,
    columns = ['Model Name', 'Accuracy', 'Precision', 'Recall', 'FScore', '
Processing Time'],
    index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)
```

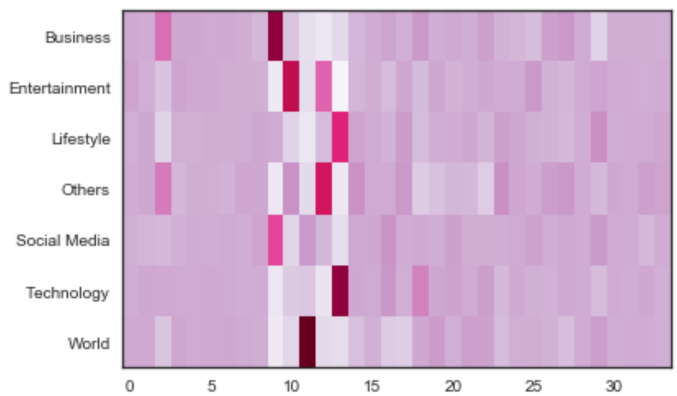
```
Out[25]: LogisticRegression(C=1000, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, max_iter=100,
                   multi_class='multinomial', n_jobs=1, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.001, verbose=0,
                   warm_start=False)

         accuracy 0.7091
         precision 0.7456
         recall 0.7091
         f1_score 0.7229
         confusion matrix
          [[ 752   19   39   16  131   33   17]
          [   8  820   31  200   55    9   40]
          [  13    8  156   15   36   96    6]
          [   4  150   27  752   25    3    6]
          [  64   26   31   16  171   15   40]
          [  24   13  274    3   28  792   62]
          [  13   33   46   33   70   67 1055]]
         process time 1.5800
```

Heatmap of co-efficients from logistic regression viewed by data_channels

In [26]:
```
channels_list = sorted(df['data_channel'].unique())
channels_list.insert(0,0)
features_list = df_data_channel.columns.values.tolist()

fig, ax = plt.subplots()

plt.imshow(lr_model1.coef_, cmap = plt.get_cmap('PuRd'), aspect = 'auto')

ax.set_yticklabels(channels_list)
#ax.set_xticklabels(features_list, rotation = 'vertical')

plt.grid(False)
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x108afe8d0>
```

Out[26]:
```
[<matplotlib.text.Text at 0x108ab6748>,
 <matplotlib.text.Text at 0x108ae4588>,
 <matplotlib.text.Text at 0x108aa81d0>,
 <matplotlib.text.Text at 0x108aa77f0>,
 <matplotlib.text.Text at 0x108aa7780>,
 <matplotlib.text.Text at 0x108aa6fd0>,
 <matplotlib.text.Text at 0x108aa56d8>,
 <matplotlib.text.Text at 0x108aa5240>]
```



As we can see features 9 thru 12 are influencers for data_channel. Feature 9 to 12 are LDA_00 thru LDA_04.

Heatmap of confusion matrix from logistic regression

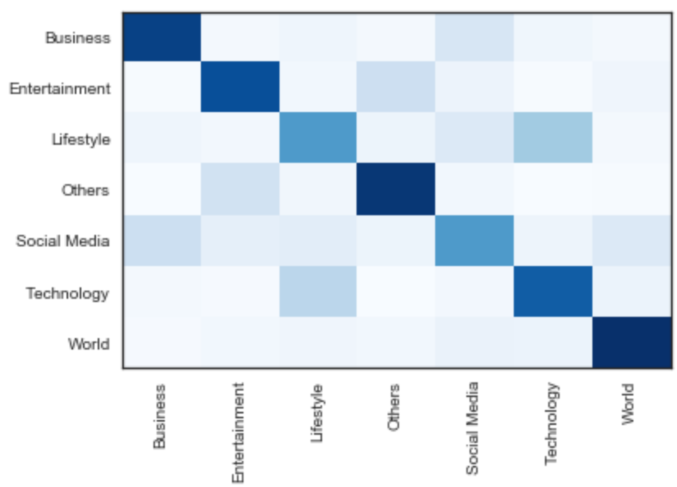
For multiclass models, confusion matrix is better visualizied as heat map

In [27]:
```
lr_model1_confusion_matrix = conf(y_test, y_hat)

cm_normalized = lr_model1_confusion_matrix.astype('float') / lr_model1_conf
usion_matrix.sum(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()

plt.imshow(cm_normalized, cmap = plt.get_cmap('Blues'), aspect = 'auto')

ax.set_yticklabels(channels_list)
ax.set_xticklabels(channels_list, rotation = "vertical")

plt.grid(False)
```

Out[27]: <matplotlib.image.AxesImage at 0x108aefa20>

Out[27]:
```
[<matplotlib.text.Text at 0x108a9ca58>,
 <matplotlib.text.Text at 0x108a9ef28>,
 <matplotlib.text.Text at 0x108a906d8>,
 <matplotlib.text.Text at 0x108a8f9e8>,
 <matplotlib.text.Text at 0x108a8eeb8>,
 <matplotlib.text.Text at 0x108a8e9b0>,
 <matplotlib.text.Text at 0x108a8d080>,
 <matplotlib.text.Text at 0x108a8c668>]
```

Out[27]:
```
[<matplotlib.text.Text at 0x108a9ae10>,
 <matplotlib.text.Text at 0x108a99668>,
 <matplotlib.text.Text at 0x108a93b38>,
 <matplotlib.text.Text at 0x108a8d6a0>,
 <matplotlib.text.Text at 0x108a8c828>,
 <matplotlib.text.Text at 0x108a8b0b8>,
 <matplotlib.text.Text at 0x108a8ac50>,
 <matplotlib.text.Text at 0x108a8a898>]
```



Interpret weights

In [32]:
```
zip_vars_LR = zip(sum(abs(lr_model1.coef_)).T, df_data_channel.columns) # c
ombine attributes

print(zip_vars_LR)
for coef, name in zip_vars_LR:
    print('\t%-35s - weight = %9.3f' % (name, coef)) # now print them out
```

```
<zip object at 0x108c5ed08>
        n_tokens_title                      - weight =     0.591
        num_keywords                        - weight =     0.706
        kw_avg_max                          - weight =     5.398
        weekday_is_monday                   - weight =     0.980
        weekday_is_tuesday                  - weight =     0.429
        weekday_is_wednesday                - weight =     0.399
        weekday_is_thursday                 - weight =     0.433
        weekday_is_friday                   - weight =     0.387
        is_weekend                          - weight =     0.939
        LDA_00                              - weight =    14.090
        LDA_01                              - weight =     9.278
        LDA_02                              - weight =    12.340
        LDA_03                              - weight =    10.438
        LDA_04                              - weight =    15.548
        global_subjectivity                 - weight =     2.057
        global_rate_positive_words          - weight =     0.339
        rate_positive_words                 - weight =     2.888
        max_positive_polarity               - weight =     2.226
        min_negative_polarity               - weight =     3.383
        max_negative_polarity               - weight =     1.366
        title_sentiment_polarity            - weight =     1.191
        abs_title_subjectivity              - weight =     0.991
        ln_n_tokens_content                 - weight =     2.238
        ln_num_hrefs                        - weight =     2.013
        ln_num_imgs                         - weight =     0.784
        ln_num_videos                       - weight =     1.561
        ln_kw_min_min                       - weight =     1.418
        ln_kw_avg_min                       - weight =     2.398
        ln_kw_min_max                       - weight =     0.080
        ln_kw_avg_avg                       - weight =     3.662
        ln_self_reference_avg_sharess       - weight =     0.274
        ln_global_rate_negative_words       - weight =     0.123
        ln_min_positive_polarity            - weight =     0.904
        ln_abs_title_sentiment_polarity     - weight =     0.705
```

For data channel classification it is not surprising that variables that are defined to measure topical contents would be the most important. The LDA variables are much more important than others when categorizing data channel.

b. Decision Tree Classifier using Grid Search

Grid search parameter set-up

In [33]:
```
# Applying Grid Search to find the best model and the best parameters

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

DTclassifier = DecisionTreeClassifier(criterion = 'entropy', random_state =
0)

parameters = [
      {
        'criterion': ['gini'],
        'max_depth': [None],
        'min_samples_split': [2, 100, 1000],
        'min_samples_leaf': [1, 10, 100],
        'max_features': [None], 'max_leaf_nodes': [None]
      },
      {
          'criterion': ['entropy'],
          'max_depth': [None, 5, 10],
          'min_samples_split': [2, 100, 1000],
          'min_samples_leaf': [1, 10, 100],
          'max_leaf_nodes': [None]
      }
     ]

grid_search = GridSearchCV(estimator = DTclassifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 3,
                           n_jobs = -1)

grid_search = grid_search.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_criterion = best_parameters['criterion']
best_max_depth = best_parameters['max_depth']
best_max_leaf_nodes = best_parameters['max_leaf_nodes']
best_min_samples_split = best_parameters['min_samples_split']
best_min_samples_leaf = best_parameters['min_samples_leaf']
best_max_features = best_parameters['max_features']
```

Best parameters for Decision Tree

In [34]:
```
best_accuracy
best_parameters
```

Out[34]:
```
0.75651899731988015
```

Out[34]:
```
{'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_samples_leaf': 10,
 'min_samples_split': 100}
```

use best parameters to create best Decision Tree model for further analysis and model comparison

In [35]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,classification_report

tic = time.clock()

# train and fit

DTclassifier = DecisionTreeClassifier(
    criterion = best_criterion,
    max_depth = best_max_depth,
    min_samples_split = best_min_samples_split,
    min_samples_leaf = best_min_samples_leaf,
    max_features = best_max_features,
    max_leaf_nodes = best_max_leaf_nodes)

DTclassifier.fit(X_train, y_train)
y_predDT = DTclassifier.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predDT))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predDT,avera
ge='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predDT,average='we
ighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predDT,average='weig
hted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predDT))
print('process time',exetime)
print("\n")

# save statistics for model comparison

raw_data = {
    'Model Name':'Decision Tree Classifier',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Decision Tree Classifier', DTclassifier))
```

```
Out[35]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=10, min_samples_split=100,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')
```

```
accuracy 0.7616
precision 0.7505
recall 0.7616
f1_score 0.7528
confusion matrix
 [[ 860   19   21   12   37   41   17]
 [  20  889    9  189    7   20   29]
 [  24   16   91   27   13  149   10]
 [  11  111   20  799    4    6   16]
 [  96   45   36   28   93   25   40]
 [  41   24   78   11    7  975   60]
 [  25   44    7   28   24   65 1124]]
process time 0.6126
```

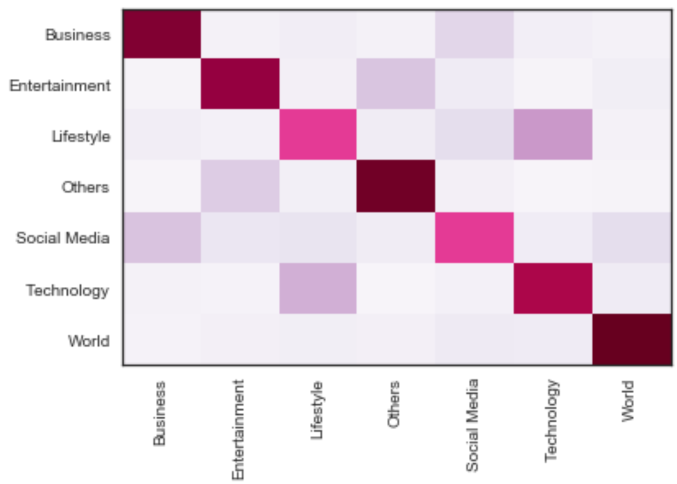
heatmap of confusion matrix for Decision Tree Classifier

In [36]:
```
DT_confusion_matrix = confusion_matrix(y_test, y_hat)

cm_normalized = DT_confusion_matrix.astype('float') / DT_confusion_matrix.s
um(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()

plt.imshow(cm_normalized, cmap = plt.get_cmap('PuRd'), aspect = 'auto')

ax.set_yticklabels(channels_list)
ax.set_xticklabels(channels_list, rotation = "vertical")

plt.grid(False)
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x108aef940>
```

Out[36]:
```
[<matplotlib.text.Text at 0x108a87668>,
 <matplotlib.text.Text at 0x108a8b9e8>,
 <matplotlib.text.Text at 0x108a7aac8>,
 <matplotlib.text.Text at 0x108a79438>,
 <matplotlib.text.Text at 0x108a77748>,
 <matplotlib.text.Text at 0x108a76908>,
 <matplotlib.text.Text at 0x108a76d30>,
 <matplotlib.text.Text at 0x108a757b8>]
```

Out[36]:
```
[<matplotlib.text.Text at 0x108a83208>,
 <matplotlib.text.Text at 0x108a87c50>,
 <matplotlib.text.Text at 0x108a7ad30>,
 <matplotlib.text.Text at 0x108a73668>,
 <matplotlib.text.Text at 0x108a72f28>,
 <matplotlib.text.Text at 0x108a76940>,
 <matplotlib.text.Text at 0x108a791d0>,
 <matplotlib.text.Text at 0x108a84940>]
```

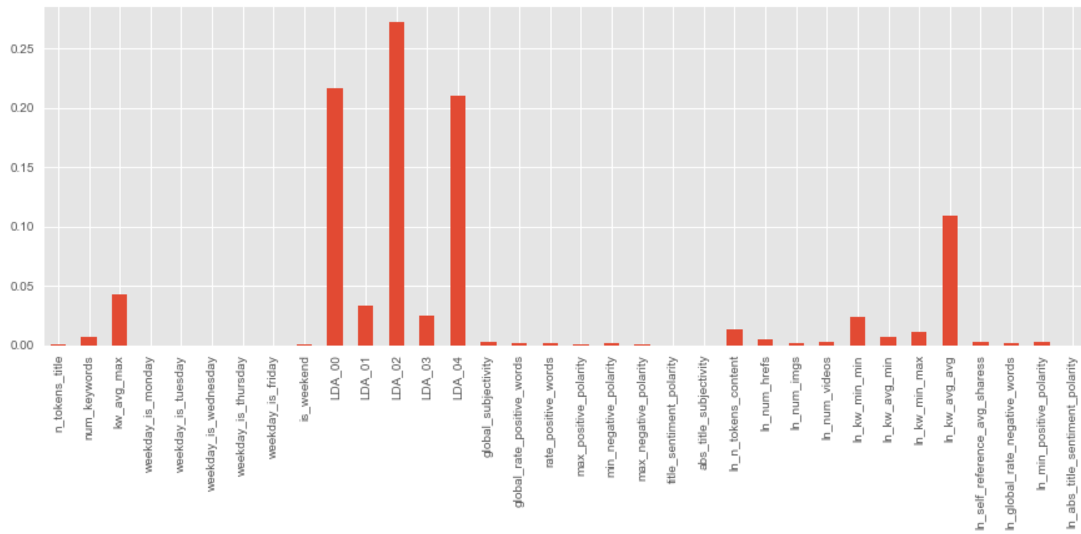


In [37]:
```
# Interpreting weights
zip_varsDT = zip(DTclassifier.feature_importances_.T, df_data_channel.colum
ns) # combine attributes

zip_varsDT = sorted(zip_varsDT)

for importance, name in zip_varsDT:
    print('\t%-35s - importance = %9.3f' % ( name, importance)) # now print
them out
```

```
	weekday_is_friday                   - importance =     0.000
	weekday_is_thursday                 - importance =     0.000
	weekday_is_wednesday                - importance =     0.000
	weekday_is_monday                   - importance =     0.000
	weekday_is_tuesday                  - importance =     0.000
	ln_abs_title_sentiment_polarity     - importance =     0.000
	title_sentiment_polarity            - importance =     0.000
	abs_title_subjectivity              - importance =     0.000
	max_negative_polarity               - importance =     0.000
	is_weekend                          - importance =     0.001
	max_positive_polarity               - importance =     0.001
	n_tokens_title                      - importance =     0.001
	global_rate_positive_words          - importance =     0.001
	min_negative_polarity               - importance =     0.002
	ln_num_imgs                         - importance =     0.002
	ln_global_rate_negative_words       - importance =     0.002
	rate_positive_words                 - importance =     0.002
	ln_self_reference_avg_sharess       - importance =     0.002
	ln_num_videos                       - importance =     0.002
	ln_min_positive_polarity            - importance =     0.003
	global_subjectivity                 - importance =     0.003
	ln_num_hrefs                        - importance =     0.005
	ln_kw_avg_min                       - importance =     0.007
	num_keywords                        - importance =     0.007
	ln_kw_min_max                       - importance =     0.011
	ln_n_tokens_content                 - importance =     0.014
	ln_kw_min_min                       - importance =     0.024
	LDA_03                              - importance =     0.025
	LDA_01                              - importance =     0.034
	kw_avg_max                          - importance =     0.043
	ln_kw_avg_avg                       - importance =     0.109
	LDA_04                              - importance =     0.210
	LDA_00                              - importance =     0.216
	LDA_02                              - importance =     0.272
```

In [38]:
```
%matplotlib inline
rcParams['figure.figsize'] = 15, 5
plt.style.use('ggplot')

weights = pd.Series(abs(DTclassifier.feature_importances_), index = df_data
_channel.columns)
weights.plot(kind = 'bar')
plt.show()
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x108a75198>



Just as with the logistic regression, it is the LDA variables that stand out as predictive of data channel using a decision tree classifier.

c. Random Forest Classifier

Grid Search parameter set-up for Random Forest classifier

In [40]:
```
RFclf = RandomForestClassifier(
    criterion = 'entropy',
    max_features= 'sqrt',
    max_depth = 5,
    n_estimators = 10,
    n_jobs = -1)

#RFclf.fit(master_X_train, master_y_train)

param_grid =[
    {
    'criterion': ['gini'],
    'n_estimators': [100, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 50]
},
 {
    'criterion': ['entropy'],
    'n_estimators': [100, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 50]
}
]

RF_grid_search = GridSearchCV(
    estimator = RFclf,
    param_grid = param_grid,
    cv = 3)

grid_search = RF_grid_search.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_criterion = best_parameters['criterion']
best_max_depth = best_parameters['max_depth']
best_max_features = best_parameters['max_features']
best_n_estimators = best_parameters['n_estimators']
```

best parameters for Random Forest Classifier

In [41]:
```
best_accuracy
best_parameters
```

Out[41]:
```
0.80769352041620679
```

Out[41]:
```
{'criterion': 'entropy',
 'max_depth': 50,
 'max_features': 'sqrt',
 'n_estimators': 500}
```

using best parameters for main model for further analysis and model comparison

In [43]:
```
from sklearn.ensemble import RandomForestClassifier

tic = time.clock()

# train and test

RFclf = RandomForestClassifier(
    criterion = best_criterion,
    max_depth = best_max_depth,
    max_features = best_max_features,
    n_estimators = best_n_estimators,
    n_jobs = -1)

RFclf.fit(X_train, y_train)
y_predRF = RFclf.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predRF))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predRF, aver
age ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predRF, average =
'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predRF, average = 'w
eighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predRF))
print('process time',exetime)
print("\n")


# save statistics for model comparison

raw_data = {
    'Model Name':'Random Forest Classifier',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Random Forest Classifier', RFclf))
```

```
Out[43]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entrop
         y',
                     max_depth=50, max_features='sqrt', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=-1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

         accuracy 0.8066
         precision 0.7987
         recall 0.8066
         f1_score 0.7889
         confusion matrix
          [[ 910   14    2    8    9   41   23]
          [  21  944    1  140    3   26   28]
          [  37   11   47   26    8  190   11]
          [  10   57   10  862    1   10   17]
          [ 100   31   10   25  122   33   42]
          [  32   19   14    3    8 1063   57]
          [  28   31    1   32    4   53 1168]]
         process time 68.7788
```

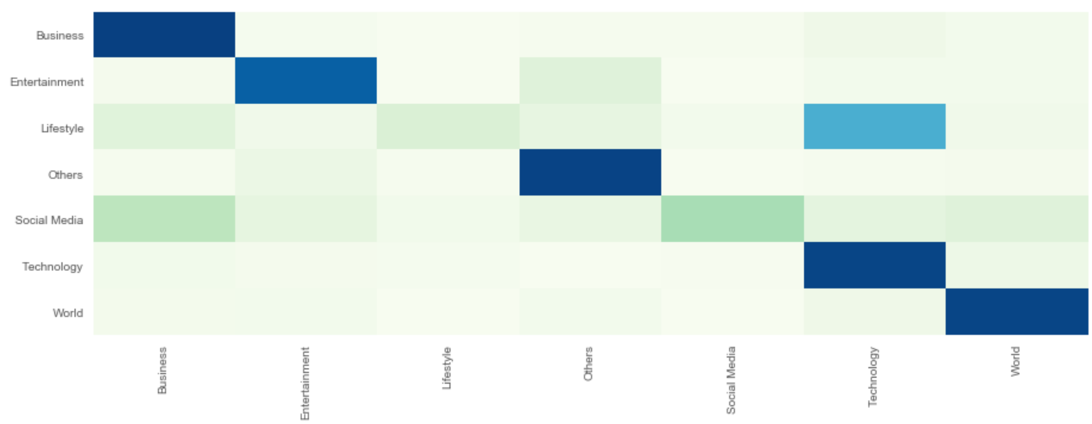
heatmap of confusion matrix for Random Forest Classifier

In [44]:
```
RF_confusion_matrix = confusion_matrix(y_test, y_predRF)

cm_normalized = RF_confusion_matrix.astype('float') / RF_confusion_matrix.s
um(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()

plt.imshow(cm_normalized, cmap = plt.get_cmap('GnBu'), aspect = 'auto')

ax.set_yticklabels(channels_list)
ax.set_xticklabels(channels_list, rotation = "vertical")
plt.grid(False)
```

Out[44]:
```
<matplotlib.image.AxesImage at 0x108a54ba8>
```

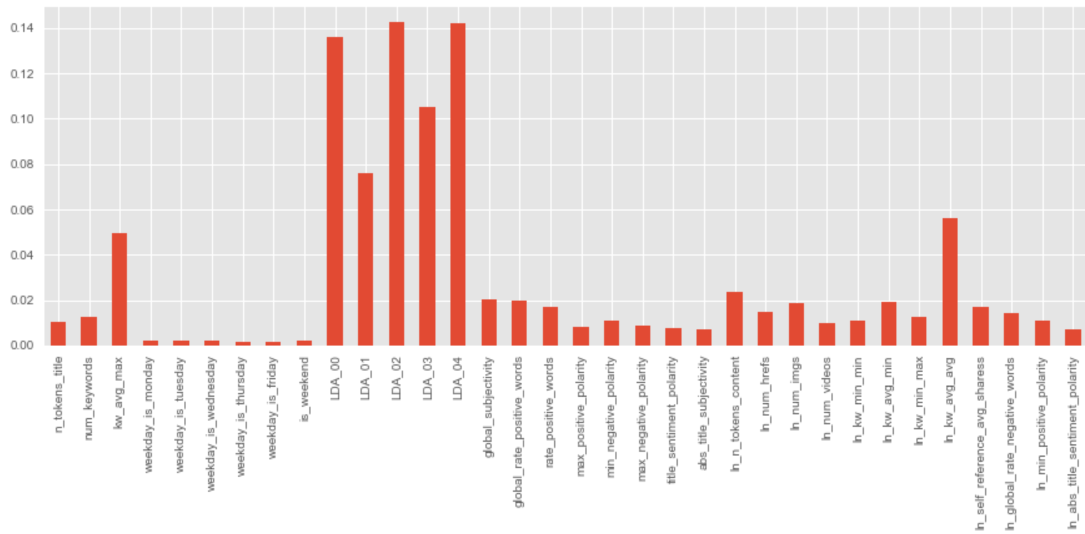
Out[44]:
```
[<matplotlib.text.Text at 0x1069db7f0>,
 <matplotlib.text.Text at 0x1069df0b8>,
 <matplotlib.text.Text at 0x105dc7860>,
 <matplotlib.text.Text at 0x105dc7ac8>,
 <matplotlib.text.Text at 0x105dc6f98>,
 <matplotlib.text.Text at 0x105dc5e48>,
 <matplotlib.text.Text at 0x1069d8a90>,
 <matplotlib.text.Text at 0x1069db470>]
```

Out[44]:
```
[<matplotlib.text.Text at 0x1069d7668>,
 <matplotlib.text.Text at 0x1069df048>,
 <matplotlib.text.Text at 0x1069ce6a0>,
 <matplotlib.text.Text at 0x1069ede80>,
 <matplotlib.text.Text at 0x105dc5588>,
 <matplotlib.text.Text at 0x105dc40b8>,
 <matplotlib.text.Text at 0x105dc6d30>,
 <matplotlib.text.Text at 0x105dc3e80>]
```



In [45]:
```
%matplotlib inline
rcParams['figure.figsize'] = 15, 5
plt.style.use('ggplot')

weights = pd.Series(abs(RFclf.feature_importances_), index = df_data_channe
l.columns)
weights.plot(kind = 'bar')
plt.show()
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x108a692e8>



As expected, the LDA variables are the most predictive of data channel when using a random forest classifier, just as with the logistic regression and decision tree for this task.

d. Naive Bayes

- Evaluate both Mulinomial and Gaussian
- Gaussian has only default parameters, so will run grid search only on Multinomial

d.1 Multinomial Naive Bayes

In [46]:
```
from sklearn.naive_bayes import MultinomialNB

MNBclf = MultinomialNB(
    alpha = 0.01,
    class_prior = None,
    fit_prior = True)

params = {
    'alpha':[0.1, 0.5, 1.0]
}

MNB_grid_search = GridSearchCV(
    MNBclf,
    params,
    cv = 3)

grid_search = MNB_grid_search.fit(master_X_train, master_y_train)

best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

best_accuracy
best_parameters

best_alpha = best_parameters['alpha']
```

Out[46]:
```
0.64139996846917857
```

Out[46]:
```
{'alpha': 0.1}
```

In [47]:
```
tic = time.clock()

# train and test

MNBclf = MultinomialNB(
    alpha = best_alpha,
    class_prior = None,
    fit_prior = True)

MNBclf.fit(X_train, y_train)
y_predMNB = MNBclf.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predMNB))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predMNB, ave
rage ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predMNB, average =
'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predMNB, average = '
weighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predMNB))
print('process time',exetime)
print("\n")


# save statistics for model comparison

raw_data = {
    'Model Name':'Multinomial Naïve Bayes',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Multinomial Naïve Bayes', MNBclf))
```

```
Out[47]: MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)

accuracy 0.6377
precision 0.6139
recall 0.6377
f1_score 0.6190
confusion matrix
 [[ 742   37   36   25   16   82   69]
 [  35  666   56  215   15  103   73]
 [  30   18   20   35    2  181   44]
 [  21  198   48  620    8   46   26]
 [ 112   43   19   29   29   52   79]
 [  63   35   23   20    6  898  151]
 [  24   26   18   27   18  134 1070]]
process time 0.1526
```

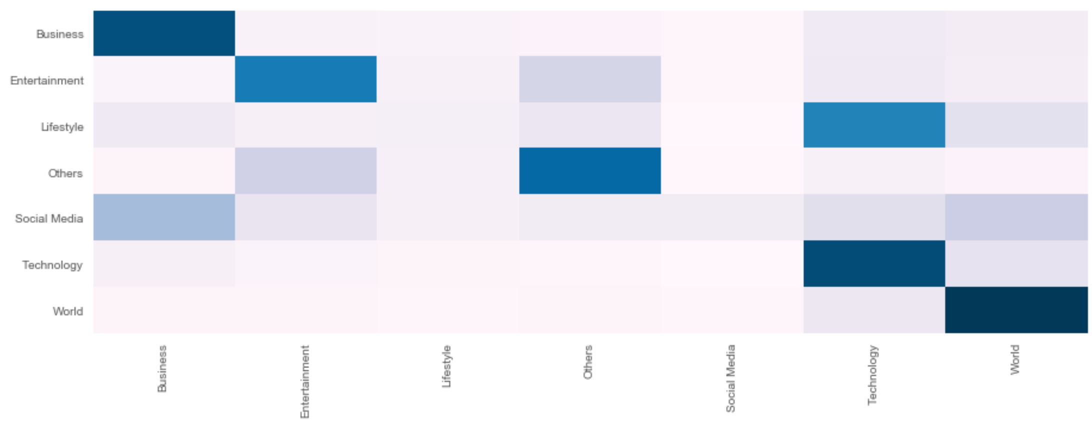
heatmap of confusion matrix for Multinomial Naive Bayes Classifier

In [48]:
```
MNB_confusion_matrix = confusion_matrix(y_test, y_predMNB)

cm_normalized = MNB_confusion_matrix.astype('float') / MNB_confusion_matrix
.sum(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()

plt.imshow(cm_normalized, cmap = plt.get_cmap('PuBu'), aspect = 'auto')

ax.set_yticklabels(channels_list)
ax.set_xticklabels(channels_list, rotation = "vertical")

plt.grid(False)
```

Out[48]:
```
<matplotlib.image.AxesImage at 0x106a015c0>
```

Out[48]:
```
[<matplotlib.text.Text at 0x1058a3208>,
 <matplotlib.text.Text at 0x1058a6278>,
 <matplotlib.text.Text at 0x10589bb70>,
 <matplotlib.text.Text at 0x10589e630>,
 <matplotlib.text.Text at 0x105d9ecc0>,
 <matplotlib.text.Text at 0x105d8b358>,
 <matplotlib.text.Text at 0x105d9d0f0>,
 <matplotlib.text.Text at 0x1058c4550>]
```

Out[48]:
```
[<matplotlib.text.Text at 0x105892278>,
 <matplotlib.text.Text at 0x1058910f0>,
 <matplotlib.text.Text at 0x10589b828>,
 <matplotlib.text.Text at 0x1058c30f0>,
 <matplotlib.text.Text at 0x1058c2cf8>,
 <matplotlib.text.Text at 0x105d93cc0>,
 <matplotlib.text.Text at 0x1058c1898>,
 <matplotlib.text.Text at 0x1058b0fd0>]
```



In [50]:
```
# Interpreting weights
zip_varsMNB = zip(sum(abs(MNBclf.coef_.T)),df_data_channel.columns) # combi
ne attributes
zip_varsMNB = sorted(zip_varsMNB)
for coef, name in zip_varsDT:
    print('\t%-35s - weight = %9.3f' % ( name, coef)) # now print them out
```

```
        weekday_is_friday                   - weight =     0.000
        weekday_is_thursday                 - weight =     0.000
        weekday_is_wednesday                - weight =     0.000
        weekday_is_monday                   - weight =     0.000
        weekday_is_tuesday                  - weight =     0.000
        ln_abs_title_sentiment_polarity     - weight =     0.000
        title_sentiment_polarity            - weight =     0.000
        abs_title_subjectivity              - weight =     0.000
        max_negative_polarity               - weight =     0.000
        is_weekend                          - weight =     0.001
        max_positive_polarity               - weight =     0.001
        n_tokens_title                      - weight =     0.001
        global_rate_positive_words          - weight =     0.001
        min_negative_polarity               - weight =     0.002
        ln_num_imgs                         - weight =     0.002
        ln_global_rate_negative_words       - weight =     0.002
        rate_positive_words                 - weight =     0.002
        ln_self_reference_avg_sharess       - weight =     0.002
        ln_num_videos                       - weight =     0.002
        ln_min_positive_polarity            - weight =     0.003
        global_subjectivity                 - weight =     0.003
        ln_num_hrefs                        - weight =     0.005
        ln_kw_avg_min                       - weight =     0.007
        num_keywords                        - weight =     0.007
        ln_kw_min_max                       - weight =     0.011
        ln_n_tokens_content                 - weight =     0.014
        ln_kw_min_min                       - weight =     0.024
        LDA_03                              - weight =     0.025
        LDA_01                              - weight =     0.034
        kw_avg_max                          - weight =     0.043
        ln_kw_avg_avg                       - weight =     0.109
        LDA_04                              - weight =     0.210
        LDA_00                              - weight =     0.216
        LDA_02                              - weight =     0.272
```

Once again, LDA variable are most predictive of the data channel.

d.2 Gaussian Naive Bayes

In [51]:
```
from sklearn.naive_bayes import GaussianNB

tic = time.clock()

# train and test

GNBclf = GaussianNB()

GNBclf.fit(X_train, y_train)
y_predGNB = GNBclf.predict(X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(y_test, y_predGNB))
precision = '{0:.4f}'.format(metrics.precision_score(y_test, y_predGNB, ave
rage ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(y_test, y_predGNB, average =
'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(y_test, y_predGNB, average = '
weighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy",accuracy )
print("precision",precision )
print("recall",recall )
print("f1_score",f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predGNB))
print('process time',exetime)
print("\n")


# save statistics for model comparison

raw_data = {
    'Model Name':'Gaussian Naïve Bayes',
    'Accuracy':accuracy,
    'Precision':precision,
    'Recall':recall,
    'FScore':f1_score,
    'Processing Time': exetime
}

df_tbl = pd.DataFrame(raw_data,
        columns = ['Model Name','Accuracy','Precision','Recall','FScore','P
rocessing Time'],
        index = [i_index + 1])

comparison_tbl = comparison_tbl.append(df_tbl)

#append model classifier for cross-validation

models.append(('Gaussian Naïve Bayes', GNBclf))
```

```
Out[51]: GaussianNB(priors=None)

         accuracy 0.7110
         precision 0.7021
         recall 0.7110
         f1_score 0.7058
         confusion matrix
          [[ 808   22   21   11   72   50   23]
          [  16  832   20  228   27   14   26]
          [  36   10   68   25    9  175    7]
          [   1  197   36  689   35    3    6]
          [ 141   32   27   21   75   22   45]
          [  28   19  102    5   14  957   71]
          [  27   49   10   24   42   84 1081]]
         process time 0.1218
```

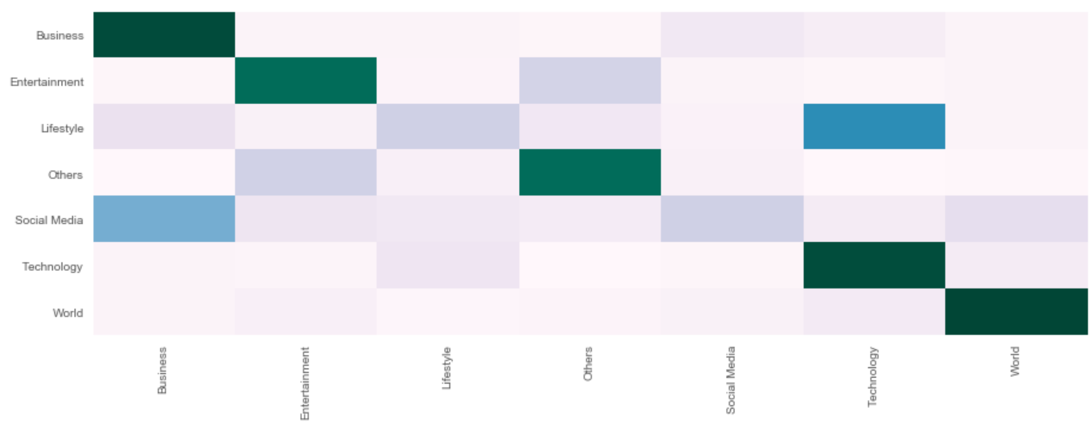
heatmap of confusion matrix for Gaussian Naive Bayes Classifier

In [52]:
```
GNB_confusion_matrix = confusion_matrix(y_test, y_predGNB)

cm_normalized = GNB_confusion_matrix.astype('float') / GNB_confusion_matrix
.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots()

plt.imshow(cm_normalized, cmap = plt.get_cmap('PuBuGn'), aspect = 'auto')

ax.set_yticklabels(channels_list)
ax.set_xticklabels(channels_list, rotation = "vertical")

plt.grid(False)
```

Out[52]:
```
<matplotlib.image.AxesImage at 0x1069dd0f0>
```

Out[52]:
```
[<matplotlib.text.Text at 0x10588c278>,
 <matplotlib.text.Text at 0x105024f60>,
 <matplotlib.text.Text at 0x10589fda0>,
 <matplotlib.text.Text at 0x1058a0b38>,
 <matplotlib.text.Text at 0x1058a1e80>,
 <matplotlib.text.Text at 0x10589a630>,
 <matplotlib.text.Text at 0x10589a2e8>,
 <matplotlib.text.Text at 0x105899f98>]
```

Out[52]:
```
[<matplotlib.text.Text at 0x10588eac8>,
 <matplotlib.text.Text at 0x105023eb8>,
 <matplotlib.text.Text at 0x10589f240>,
 <matplotlib.text.Text at 0x105898048>,
 <matplotlib.text.Text at 0x105897d30>,
 <matplotlib.text.Text at 0x1058992b0>,
 <matplotlib.text.Text at 0x105896fd0>,
 <matplotlib.text.Text at 0x105896240>]
```



Interpreting weights

In [53]:
```
# Interpreting weights
zip_varsGNB = zip(sum(abs(GNBclf.theta_.T)),df_data_channel.columns) # comb
ine attributes
zip_varsGNB = sorted(zip_varsGNB)
for theta, name in zip_varsDT:
    print('\t%-35s - weight = %9.3f' % ( name, theta)) # now print them out
```

```
	weekday_is_friday                   - weight =     0.000
	weekday_is_thursday                 - weight =     0.000
	weekday_is_wednesday                - weight =     0.000
	weekday_is_monday                   - weight =     0.000
	weekday_is_tuesday                  - weight =     0.000
	ln_abs_title_sentiment_polarity     - weight =     0.000
	title_sentiment_polarity            - weight =     0.000
	abs_title_subjectivity              - weight =     0.000
	max_negative_polarity               - weight =     0.000
	is_weekend                          - weight =     0.001
	max_positive_polarity               - weight =     0.001
	n_tokens_title                      - weight =     0.001
	global_rate_positive_words          - weight =     0.001
	min_negative_polarity               - weight =     0.002
	ln_num_imgs                         - weight =     0.002
	ln_global_rate_negative_words       - weight =     0.002
	rate_positive_words                 - weight =     0.002
	ln_self_reference_avg_sharess       - weight =     0.002
	ln_num_videos                       - weight =     0.002
	ln_min_positive_polarity            - weight =     0.003
	global_subjectivity                 - weight =     0.003
	ln_num_hrefs                        - weight =     0.005
	ln_kw_avg_min                       - weight =     0.007
	num_keywords                        - weight =     0.007
	ln_kw_min_max                       - weight =     0.011
	ln_n_tokens_content                 - weight =     0.014
	ln_kw_min_min                       - weight =     0.024
	LDA_03                              - weight =     0.025
	LDA_01                              - weight =     0.034
	kw_avg_max                          - weight =     0.043
	ln_kw_avg_avg                       - weight =     0.109
	LDA_04                              - weight =     0.210
	LDA_00                              - weight =     0.216
	LDA_02                              - weight =     0.272
```

It is no surprise that LDA variables continue to be the most predictive of data channel.

Modeling and Evaluation 4

Evaluation Metrics

In [55]:
```
# converting acc, pre, recall, fscore and time to numeric values for plots

comparison_tbl = comparison_tbl.reset_index(drop=True)
comparison_tbl['Precision'] = pd.to_numeric(comparison_tbl['Precision'])
comparison_tbl['Accuracy'] = pd.to_numeric(comparison_tbl['Accuracy'])
comparison_tbl['FScore']= pd.to_numeric(comparison_tbl['FScore'])
comparison_tbl['Processing Time'] = pd.to_numeric(comparison_tbl['Processin
g Time'])
comparison_tbl['Recall'] = pd.to_numeric(comparison_tbl['Recall'])
comparison_tbl
```

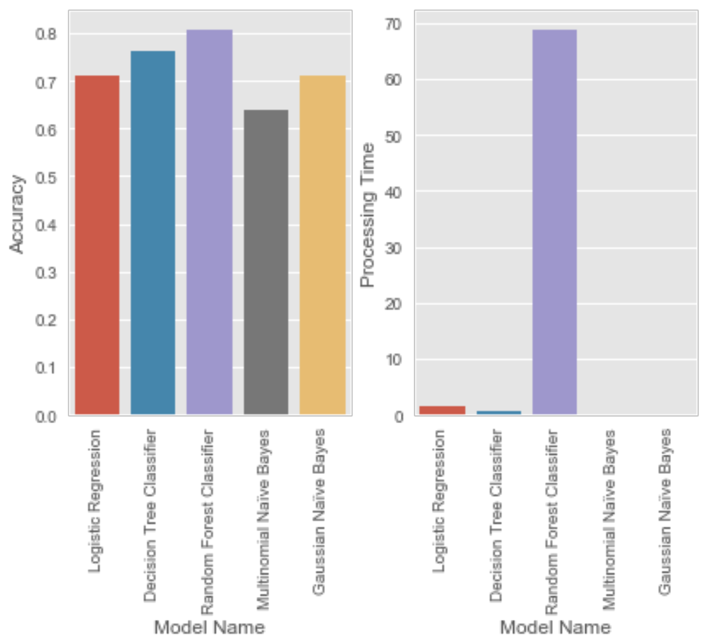
Out[55]:

	Model Name	Accuracy	Precision	Recall	FScore	Processing Time
0	Logistic Regression	0.7091	0.7456	0.7091	0.7229	1.5800
1	Decision Tree Classifier	0.7616	0.7505	0.7616	0.7528	0.6126
2	Random Forest Classifier	0.8066	0.7987	0.8066	0.7889	68.7788
3	Multinomial Naïve Bayes	0.6377	0.6139	0.6377	0.6190	0.1526
4	Gaussian Naïve Bayes	0.7110	0.7021	0.7110	0.7058	0.1218

Visualization of metrics

In [56]:
```
from pylab import rcParams
%matplotlib inline

#comparison_tbl.plot()

fig, axs = plt.subplots(ncols=2)
fig.tight_layout()
plt.setp(axs[0].xaxis.get_majorticklabels(), rotation = 90 )
plt.setp(axs[1].xaxis.get_majorticklabels(), rotation = 90 )

sns.barplot(x = 'Model Name', y = 'Accuracy', data = comparison_tbl, ax = a
xs[0])

sns.barplot(data = comparison_tbl, y = 'Processing Time', x = 'Model Name',
ax = axs[1])
```

```
Out[56]: [None, None, None, None, None, None, None, None, None, None, None, None]
Out[56]: [None, None, None, None, None, None, None, None, None, None, None, None]
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x105d98a58>
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x105020908>
```



10-K Cross-Validation for each classifier

For each classifier we run 10 fold cross validation which will help us narrow down one final model

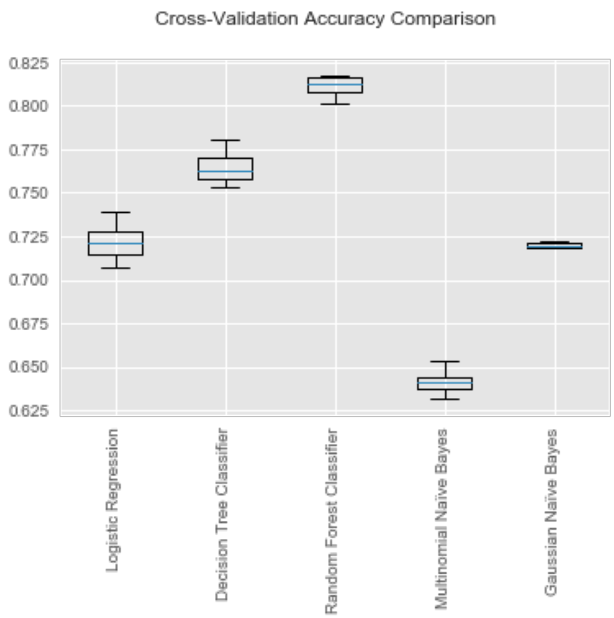
In [57]:
```
from sklearn import model_selection

# evaluate each model in turn

results = []
names = []
scoring = 'accuracy'

for name, model in models:
    cv_results = model_selection.cross_val_score(
        model,
        master_X_train,
        master_y_train,
        cv = 10,
        scoring = scoring)
    results.append(cv_results)
    names.append(name)
    msg = "\n\t%-35s mean (std.dev) : %.3f (%.4f)" % (name, cv_results.mean
(), cv_results.std())
    #print(cv_results)
    print(msg)
```

```
	Logistic Regression                 mean (std.dev) : 0.722 (0.0094)

	Decision Tree Classifier            mean (std.dev) : 0.765 (0.0086)

	Random Forest Classifier            mean (std.dev) : 0.811 (0.0056)

	Multinomial Naïve Bayes             mean (std.dev) : 0.641 (0.0058)

	Gaussian Naïve Bayes                mean (std.dev) : 0.720 (0.0075)
```

Cross-Validation accuracy comparison of all models

In [58]:
```
# boxplot accuracy comparison

fig = plt.figure()
fig.suptitle('Cross-Validation Accuracy Comparison')
ax = fig.add_subplot(111)

plt.boxplot(results)

ax.set_xticklabels(names, rotation = 90)
plt.show()
```

```
Out[58]: <matplotlib.text.Text at 0x105dc0c18>

Out[58]: {'boxes': [<matplotlib.lines.Line2D at 0x104fe66a0>,
           <matplotlib.lines.Line2D at 0x104fe1c50>,
           <matplotlib.lines.Line2D at 0x104fdcac8>,
           <matplotlib.lines.Line2D at 0x104fd8b00>,
           <matplotlib.lines.Line2D at 0x104fd4278>],
          'caps': [<matplotlib.lines.Line2D at 0x104fe4d68>,
           <matplotlib.lines.Line2D at 0x104fe3198>,
           <matplotlib.lines.Line2D at 0x104fdfcf8>,
           <matplotlib.lines.Line2D at 0x104fdecc0>,
           <matplotlib.lines.Line2D at 0x104fdb160>,
           <matplotlib.lines.Line2D at 0x104fdac88>,
           <matplotlib.lines.Line2D at 0x104fd6eb8>,
           <matplotlib.lines.Line2D at 0x104fd63c8>,
           <matplotlib.lines.Line2D at 0x104fd2898>,
           <matplotlib.lines.Line2D at 0x104fd2198>],
          'fliers': [<matplotlib.lines.Line2D at 0x104fe2be0>,
           <matplotlib.lines.Line2D at 0x104fddfd0>,
           <matplotlib.lines.Line2D at 0x104fd9b38>,
           <matplotlib.lines.Line2D at 0x104fd5940>,
           <matplotlib.lines.Line2D at 0x104feeb00>],
          'means': [],
          'medians': [<matplotlib.lines.Line2D at 0x104fe35c0>,
           <matplotlib.lines.Line2D at 0x104fde198>,
           <matplotlib.lines.Line2D at 0x104fd9eb8>,
           <matplotlib.lines.Line2D at 0x104fd5b00>,
           <matplotlib.lines.Line2D at 0x104fd16d8>],
          'whiskers': [<matplotlib.lines.Line2D at 0x104fe5908>,
           <matplotlib.lines.Line2D at 0x104fe5320>,
           <matplotlib.lines.Line2D at 0x104fe1b70>,
           <matplotlib.lines.Line2D at 0x104fdfa90>,
           <matplotlib.lines.Line2D at 0x104fdc0f0>,
           <matplotlib.lines.Line2D at 0x104fdbc88>,
           <matplotlib.lines.Line2D at 0x104fd8128>,
           <matplotlib.lines.Line2D at 0x104fd7908>,
           <matplotlib.lines.Line2D at 0x104fd3b00>,
           <matplotlib.lines.Line2D at 0x104fd3b38>]}

Out[58]: [<matplotlib.text.Text at 0x1058b77f0>,
          <matplotlib.text.Text at 0x104ff1cc0>,
          <matplotlib.text.Text at 0x104ff1630>,
          <matplotlib.text.Text at 0x104fd06d8>,
          <matplotlib.text.Text at 0x104fcf7b8>]
```



Holdout test data set prediction with our final model

Our final best model is Random Forest Classifier. We will run prediction on that fit with the test data set we set aside at the beggining of the project. We will calculate statistics for the prediction.

In [59]:
```
y_predFinal = RFclf.predict(golden_X_test)

# calculate statistics

accuracy = '{0:.4f}'.format(metrics.accuracy_score(golden_y_test, y_predFin
al))
precision = '{0:.4f}'.format(metrics.precision_score(golden_y_test, y_predF
inal, average ='weighted'))
recall = '{0:.4f}'.format(metrics.recall_score(golden_y_test, y_predFinal,
average = 'weighted'))
f1_score = '{0:.4f}'.format(metrics.f1_score(golden_y_test, y_predFinal, av
erage = 'weighted'))
toc =  time.clock()
exetime = '{0:.4f}'.format(toc-tic)

# print statistics
print("accuracy", accuracy )
print("precision", precision )
print("recall", recall )
print("f1_score", f1_score )
print("confusion matrix\n", confusion_matrix(y_test, y_predRF))
print('process time', exetime)
print("\n")
```

```
accuracy 0.8113
precision 0.8084
recall 0.8113
f1_score 0.7910
confusion matrix
 [[ 910   14    2    8    9   41   23]
 [  21  944    1  140    3   26   28]
 [  37   11   47   26    8  190   11]
 [  10   57   10  862    1   10   17]
 [ 100   31   10   25  122   33   42]
 [  32   19   14    3    8 1063   57]
 [  28   31    1   32    4   53 1168]]
process time 804.0741
```

Final Statement regarding Task 2 Classifiers

Deployment

Usefulness

The usefulness of this model is to provide guidance as to which articles are to be published on the mashable.com web-site. One facet of Mashable's business model is to generate revenue by selling advertising and sponsored content on the mashable.com web-site. The value of the site to advertisers, and the revenue stream that can be protected for mashable, can be measured by the number of articles that mashable readers share with their on-line social media network. The value of this classification model is to identify the site content with higher likelihood of being popularly shared within the the target audience.

Measurement of model value

A way to measure the model's value is by monitoring the success of the prediction model in terms of increasing social media shares from mashable published articles. Since there can be many (very many !) factors influencing the popularity (number of shares) of articles that are beyond the applicability scope of the model, it is recommended that the value measurement be assessed by controlled A/B releases of content that is decided by current methods, side-by-side, with content that is recommended with this classification. By using a side-by-side A/B evaluation, the effectiveness of the classification model for improving content recommendations can be credibly assessed.

Deployment method / external data support

The current vision for deployment of the model includes these elements :

- a web-based user interface in which the user (the article author or other proposer) can deposit the article (title, content, embedded images and videos) in similar fashion as DropBox or other similar media-sharing file servers
- a parsing / feature extraction machine deployed at the shared file server site
  - the model requires the extraction of several explanatory variables (feature extraction) from the content of the article in order to evaluate the article's popularity score in the model
    - among the features to be extracted from the article as model inputs are the following :
      - key-word statistics, positive and negative sentiment counts, LDA_00 --> LDA_04 scores, number of images and number of videos, sentiment polarity based on title, related hyperlink references
    - a feature extraction machine will be deployed at the file server site to process the article content and provide the model inputs
    - in addition, another element of the deployment is to assign the article to an appropriate data channel (e.g., Social Media, Technology, Business). this will also be an element deployed on the file server site. the text and title will be processed through a text-mining approach to evaluate the appropriate data channel.
  - once the key features have been developed from the text processing machines, then all of the inputs for the classification model are available and can be executed
  - the concept of the deployment is :
    - the article is uploaded to the submittal site,
    - the feature extraction machine is launched, then
    - the popularity classification model is executed, and
    - the recommendation to publish or not to publish is provided to the mashable content editor.
    - the execution time associated to this activity is anticipated to be real-time, within less than a few seconds.
  - the goal of the application is that the article, with only some minimal contextual information (requested publication date, author contact information) is sufficient to provide a **publish** / **no-publish recommendation** to the content editor;

Recommendation

Task 1 - mashable article popularity prediction

For the popularity classification, we completed an evaluation of 4 different classifier models to identify method to classify an article as likley to be either popular or not-popular. The results of this effort identified that the Random Forest Classifier provided the best overall results. The metrics of this classifier are as follows : Accuracy:0.6636 Precision:0.6636 Recall: 0.6636 FScore: 0.6635

Task 2 - mashable article data_channel prediction

For the data_channel classification, we completed an evaluation of 4 different classifier models to identify method to classify an article as a particular channel.

The results of this effort identified that the Random Forest Classifier provided the best overall results. The metrics of this classifier are as follows : Accuracy:0.8066 Precision:0.7987 Recall: 0.8066 FScore: 0.7889

Exceptional Work

We have implemented Grid search in our parament selection process.