Machine Learning Engineer Nanodegree

Capstone Project

George Nizharadze
December 28th, 2017

I. Definition

Project Overview

This project lies in the domain of content analysis through Natural Language Processing. It's focused on on-line news and opinion. Western societies have become increasingly polarized in the last few years, with major shifts occurring in ideological and political trends. Many media outlets are clearly biased or openly espouse certain ideologies and political agenda.

The objective of the project is to develop a methodology for obtaining, processing and analyzing, through machine learning, on-line news and opinion. The project focusses on several known ideological and political megatrends and demonstrates that it is possible to build a machine learning model that takes in a piece of text (news, analysis, opinion) from an on-line outlet and classifies its source, which in turn is representative of a certain ideology.

The following is the link to a related research, which, however, uses different methodology from my project: http://tlfvincent.github.io/2015/10/23/presidential-speech-topics/ .

Problem Statement

The problem is a multi-class classification one. The data comprises news headlines and summary descriptions from four different sources, which respectively represent left-wing liberal, far-right, moderate conservative ideologies and state-sponsored propaganda. The problem lies in processing the text, extracting features and training a model to predict the text's belonging to one of the four classes. Broadly, I have taken the following steps to tackle the problem:

- Registered with a news service, clarified API call protocol and obtained an API key
- Preprocessed the collected text documents and explored the data
- Extracted features from the texts, using the *bag-of-words* approach
- Split the data into training and testing sets
- Selected several candidate frameworks for classification modelling
- Defined performance metrics
- Trained the models
- Fine-tuned the model which I considered to be best-performing

Metrics

Generally, the measurement of performance of classification models is based on Confusion Matrix, which summarizes true positives, true negatives, false positives and false negatives. Metrics such as Accuracy, Precision, Recall and F-score are then calculated from the elements of the confusion matrix.

I decided to assess my models based on Accuracy and F-score, for the following reasons:

- As shown later in the data exploration section, the classes in the data set are fairly balanced. Therefore, Accuracy serves as a good overall indication of the model's performance (for imbalanced class distributions, Accuracy can be misleading).

  Accuracy is calculated simply as the number of correctly classified instances divided by the total number of instances.

- F1-score is the harmonic mean of Precision and Recall, therefore, it nicely captures the model's performance on both metrics. In certain domains, Precision can be more important that Recall, or vice versa. For example, in medical diagnostics, Recall is clearly the most critical metric, as the objective is to always be able to identify a medical condition which requires treatment. In the context of my project, there is no clear priority between these two metrics, therefore, F1-score is an appropriate composite measure of the two. Note that it is possible to "tilt" F-score towards Precision or Recall by modifying the coefficient of the harmonic mean accordingly. This idea is captured by the metric's broader definition as F-beta score.

  F1-score's formula is 2 * (Precision * Recall) / (Precision + Recall). More generally, F-beta score is calculated as $(1 + beta^2) * (Precision * Recall) / (beta^2 * Precision + Recall)$, where Beta can be selected according to how much we want to "tilt" the score towards Precision or Recall, depending on which metric we believe is more relevant and important in our practical context.

II. Analysis

Data Exploration

The data for this project is text. It consists of news headlines and associated summaries for the period 23 September, 2017 to 23 November, 2017. The text was collected through NewsAPI from the following sources:

- New York Times – a proxy for left-wing liberal ideology
- Breitbart News – a proxy for right-wing ideology
- Fox News – a proxy for relatively moderate right-wing ideology
- Russia Today – an example of Russian state-sponsored propaganda

A total of 3,673 pieces of news were collected, with the following distribution among the four classes:

- The New York Times 32%
- Breitbart News 33%
- Fox News 14%
- Russia Today 21%

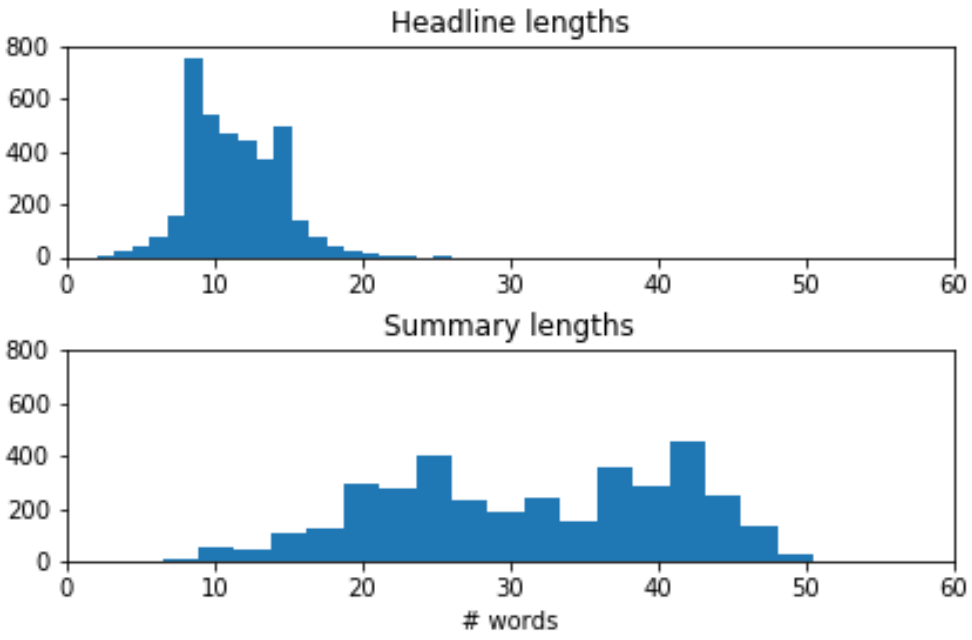
Below are several instances of the data:

Title: *Six Burned in Acid Attack as Groups Clash in East London*
Summary: *A group of men sprayed a caustic substance in the faces of at least six others, the latest in a series of similar crimes in London.*

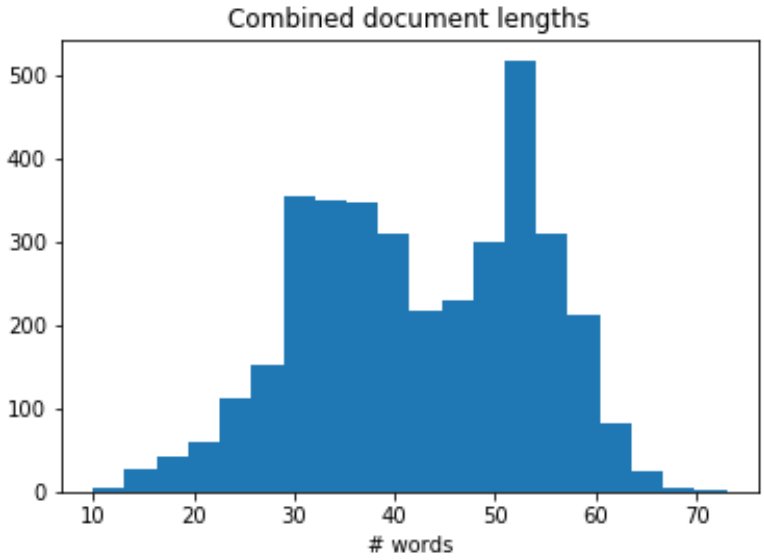
Title: *With Mugabe's Era Ending in Zimbabwe, a Warning Echoes in Africa*
Summary: *President Robert Mugabe of Zimbabwe and his wife, Grace, celebrated Zimbabwe's Independence Day in 2012 in Harare, the capital. On Wednesday, the military placed Mr. Mugabe under house arrest.*

Title: *Lazio football club face disciplinary hearing over anti-Semitic Anne Frank stickers & slogans*
Summary: *The Italian football federation has ordered Serie A club Lazio to attend a disciplinary hearing after its supporters were found guilty of distributing anti-Semitic stickers featuring the image of Holocaust victim Anne Frank*.

The average number of raw, unprocessed words in news headlines is 11.2, min and max being 2.0 and 26.0, respectively. The average number of raw, unprocessed words in news summaries is 31.4, min and max being 4.0 and 53.0, respectively. The histograms below show more detail about the numbers of raw, unprocessed words in news headlines and news summaries:



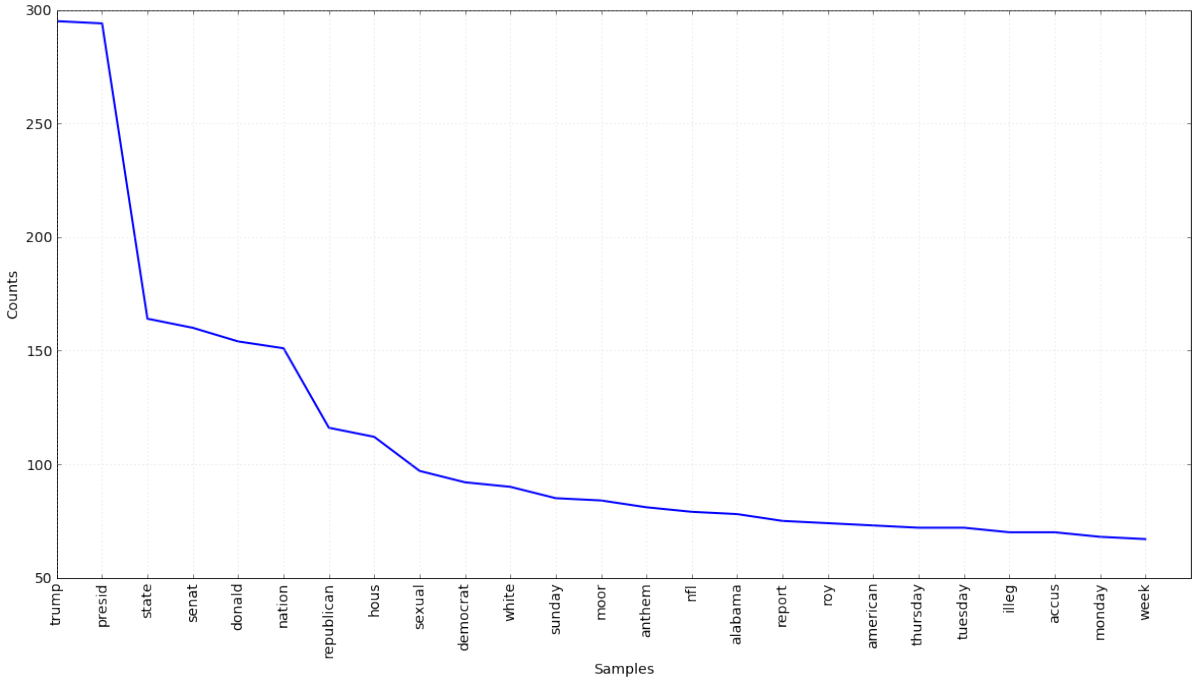
As headlines and summaries are quite brief, for subsequent analysis and model-building, I decided to combine the headline and summary of each piece of news into a single document. This results in a relatively larger text for each instance, which should allow extraction of a richer set of features. On this combined basis, the distribution of the number of raw, unprocessed words in the documents is as follows:



On this combined basis, there are no glaring outliers or anomalies in the distribution of the document lengths. Also, given how this data was collected, there are no missing values.

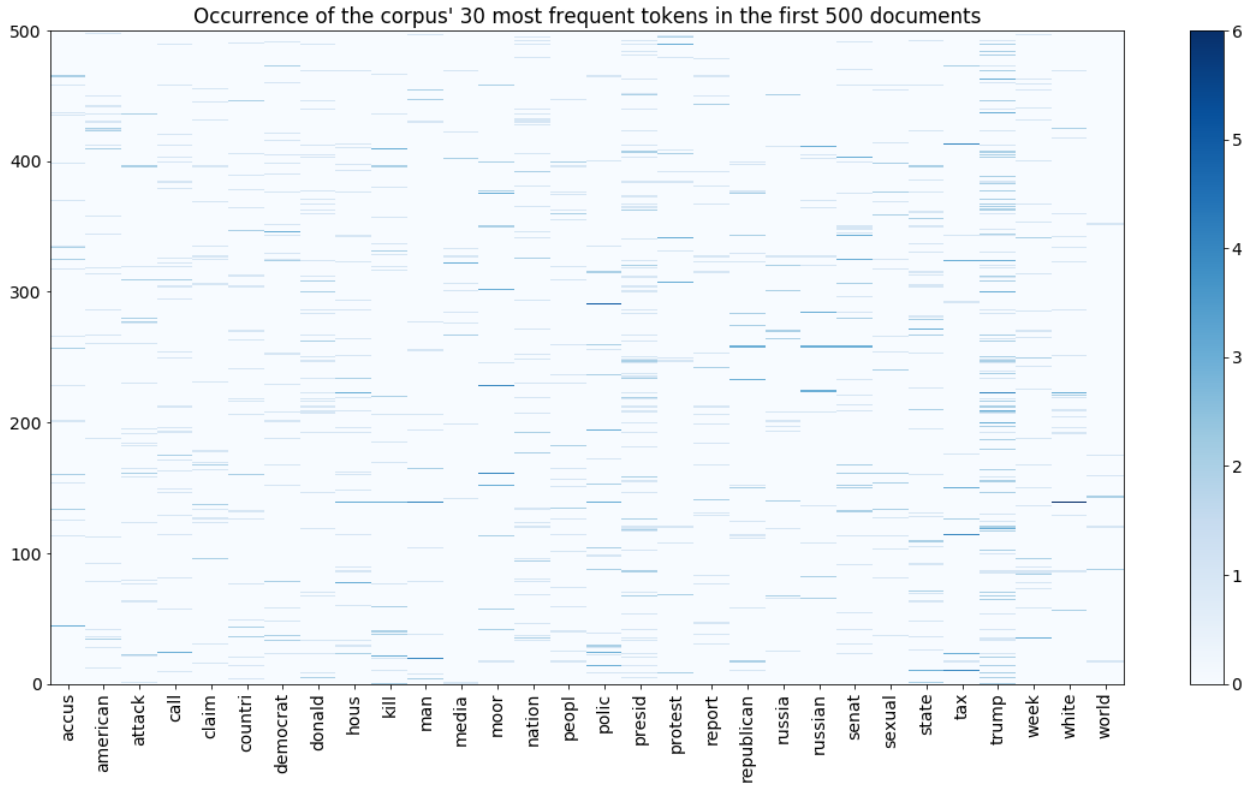
Exploratory Visualization

All the data preprocessing steps are described in detail in a later section, however, I will mention here that feature extraction is based on tokenized and stemmed words from the corpus of documents. To get a feel for the most common tokens, below is a plot of those extracted from Breitbart news corpus[1]:



The following plot visualizes a fragment of the sparse matrix obtained by creating a bag-of-words from the whole corpus vocabulary. The horizontal axis shows the 30 most frequent words in the corpus and the vertical axis has the first 500 documents, the colored horizontal lines showing the number of times the corresponding word occurs in the document. As we can see, the matrix is sparse, with the majority of values being zero. The actual machine learning will be performed on this type of features and associated numerical values. The matrix was created with sklearn's CountVectorizer, which will be discussed later.

[1] A bar plot, rather than line plot, would be more appropriate here, however, this plot is derived with nltk's `freqdist.plot` method, which produces a line plot.



Algorithms and Techniques

After the relevant text processing and creation of Bag-of-Words and TFIDF matrices (discussed in detail in a later section), I plan to try the following algorithms for the classification task. After evaluating the algorithms' initial performance with their default parameters, per metrics defined above, I will select the best one and fine-tune it.

- **Naïve-Bayes** - NB classifiers are used extensively in text classification, such as e-mail spam identification. NB classifiers assume the independence of values across features, which makes computation efficient and fast. Another advantage of NB is that it can be trained incrementally, without having to retrain the model on the entire data set when new data is added. Also, as our dataset is not very large, the high bias / low variance nature of NB is an extra plus. On the flipside, NB can't learn interactions between features, which in some cases may be required for better performance.
  In the training process, NB computes class prior probabilities and feature values' prior probabilities simply by "counting" them. It also computes likelihoods of feature values given a specific class. In the prediction process, feature values are assumed to be independently distributed and the posterior probability of a class given a feature value is calculated by the Naïve Bayes formula, which takes in the above priors and likelihoods.

- **Logistic Regression** – is widely used for classification tasks and has the nice property of yielding the probability of class belonging. This allows easy adjustment of classification thresholds and derivation of confidence intervals.
  In the training process, Logistic Regression relies on Maximum Likelihood Estimation to find the regression coefficients. Under the hood, Logistic Regression outputs continues values. They are turned into probabilities through a sigmoid function and the probabilities, in turn, are translated into class labels based on a probability threshold, the default one being 0.5.
- **Support Vector Classifier (SVC)** – Support Vector Machines, with appropriate kernel specifications, can effectively deal with data which is not linearly separable in the base feature space. On the downside, SVC is computationally complex, however, our data set is not very large, therefore, this may not be a practical concern. Another disadvantage of SVC is that it's interpretability is low.
  The training of Support Vector Machines, in general, is based on the idea of finding the largest margin of a separator between classes. SVM's find linear separators but can also successfully derive complex non-linear separators with the so-called kernel trick – a functional transformation which renders non-linear relationships into linear ones.

Benchmark

As mentioned above, I plan to use Accuracy and F1-score as the key metrics for model performance. Therefore, I will use as the benchmarks the same metrics of a naïve predictor. So, for example, if I have a total of four classes, the benchmark accuracy will thus be 25% for each class. Accuracy of the naïve predictor can be calculated by the rule of naively predicting the most prevalent class, i.e. Breitbart News, all the time. Given the class distribution of our data set, this will yield the accuracy of 33%.

As far as F1-score is concerned, in a multi-class classification setting, the score is the weighted average of the F1 scores of all the classes. In our naïve predictor, the F1-scores of all the classes except Breitbart will be zero, as the respective recalls are zero. Therefore, the F1-score of the naïve benchmark model will be the F1-score of Breitbart class multiplied by its proportion in the dataset's class distribution:

```
F1-score = class_proportion * (2 * (class_precision * class_recall) /
(class_precision + class_recall)) = 0.33 * 2 * (0.33 * 1.0) / (0.33 +
1.0)≈ 0.16
```

Thus, the benchmark Accuracy and F1-score are **33%** and **16%**, respectively.

III. Methodology

Data Preprocessing

As mentioned in the Data Exploration section, I combined the headline and summary of each piece of news into a single document. Thus, the total data set consists of 3,673 labeled text documents, each containing between a minimum of 10 and a maximum of 70 words (see the relevant histogram in the Data Exploration section). Most of the documents contain between 30 and 60 words.

As my analysis utilizes Bag-of-Words method, I plan to try *sklearn*'s `CountVectorizer` and `TfidfVectorizer` for final feature extraction. However, prior to applying these functions, I tokenized and stemmed the text the following way:

- Split out word tokens using a regex
- Converted tokens to lowercase
- Retained only alphabetic character tokens
- Removed stop-words
- Stemmed the tokens using Porter stemmer

After the above steps, I put the processed tokens back together into space-separated strings, as `CountVectorizer` and `TfidfVectorizer` take as input entire text strings[2]. For example, an initial document …

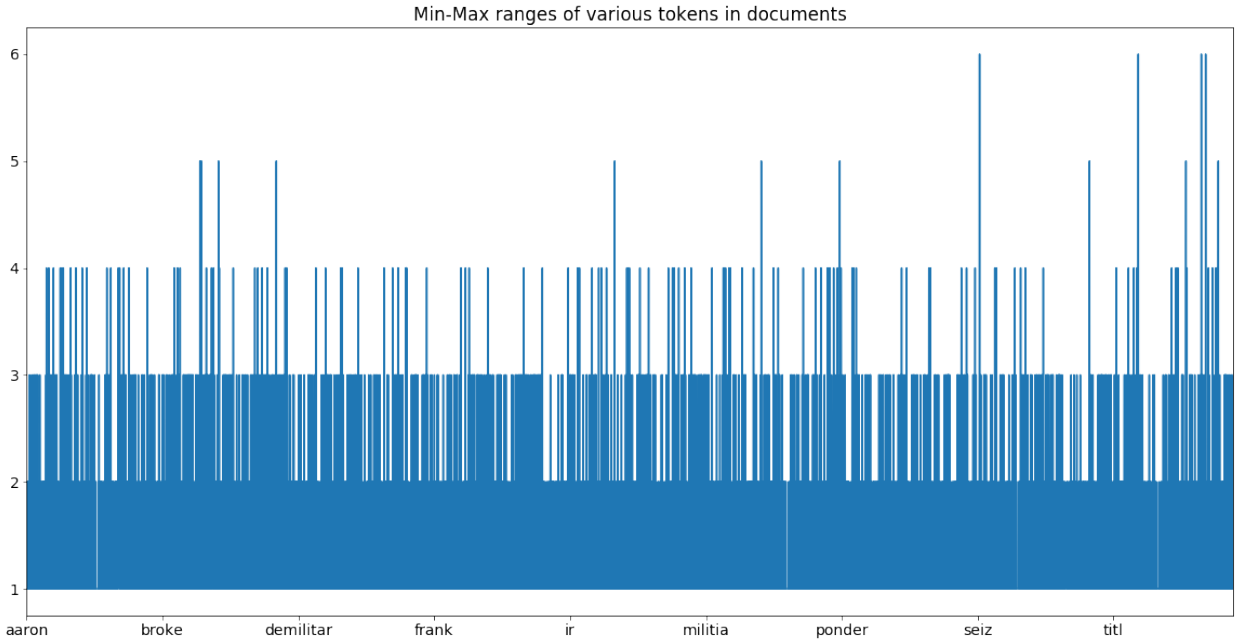
```
'Six Burned in Acid Attack as Groups Clash in East London A group of men s
prayed a caustic substance in the faces of at least six others, the latest
in a series of similar crimes in London.'
```

… looks the following way after the described pre-processing:

```
'burn acid attack group clash east london group men spray caustic substanc
face latest seri similar crime london'
```

I then split these preprocessed documents into training (75%) and test (25%) samples. I fit and transformed the training sample with `CountVectorizer.` A fragment of the output was shown in the Data Exploration section. I then inspected the min-max range of tokens in the training corpus:

[2] Note that sklearn's vectorizers can automatically perform most of the tokenization steps which I performed, however, I decided to do this manually to have more control over stop-words and to enable stemming.



As there is a somewhat big variability in the Min-Max range of tokens in the documents, I applied Min-Max scaling to the training Bag-of-Words matrix. Finally, I applied the corresponding `CountVectorizer` and `MinMaxScaler` transformations to the test data set[3].

In parallel, I applied the same steps to create `TfidfVectorizer` matrices, as I decided to compare model performance with simple Bag-of-Words (`CountVectorizer`) vs Term Frequency – Inverse Document Frequency (`TfidfVectorizer`) approaches. The results and final choice are presented in the next section.

TFIDF stands for Term Frequency – Inverse Document Frequency. TFIDF weights the frequency of a term in a document with this term's frequency of occurrence across all the documents in the corpus. The idea is that if a term occurs in most or all documents in a corpus, it will have low or no predictive power, hence this term should be weighted down or eliminated altogether.

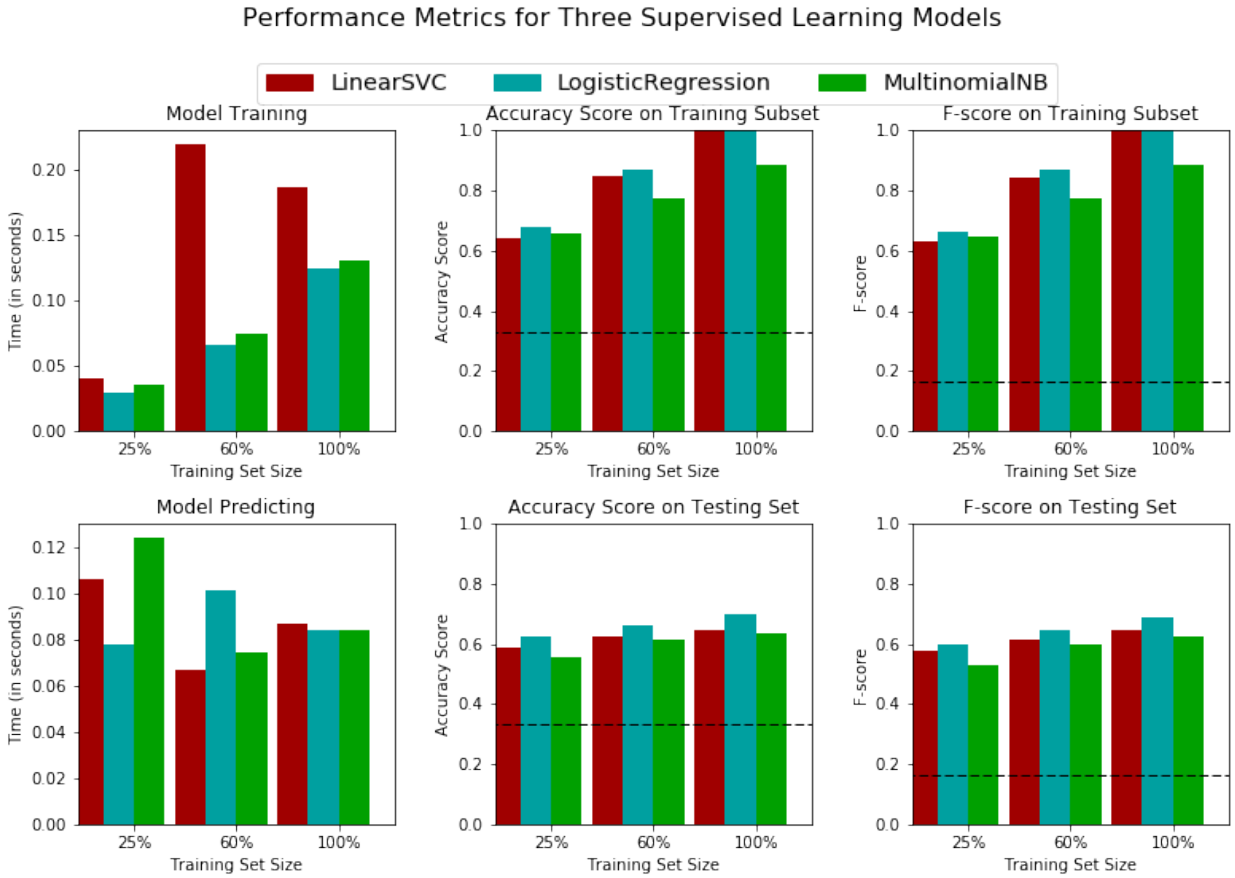
Implementation

Below is a summary of the performance of Naïve-Bayes, Logistic Regression and Support Vector Classifier on the Min-Max scaled simple Bag-of-Words data. The total number of features, i.e. the tokenized and stemmed vocabulary, ended up at 8,884. This is a wide and sparse matrix, as the total number of documents is only 3,673.

I chose 25%, 60% and 100% (no specific reason for the exact choice of 25% and 60% as opposed to, say, 30% and 50%) of the training set to observe the evolution of training and prediction times and get a feel for the learning curve and overfitting effects. I used a subset of 300 (just over 10% of the training set)

[3] I applied both `fit` and `transform` for BoW and min-max scaling to the training data and only `transform` to the test data.

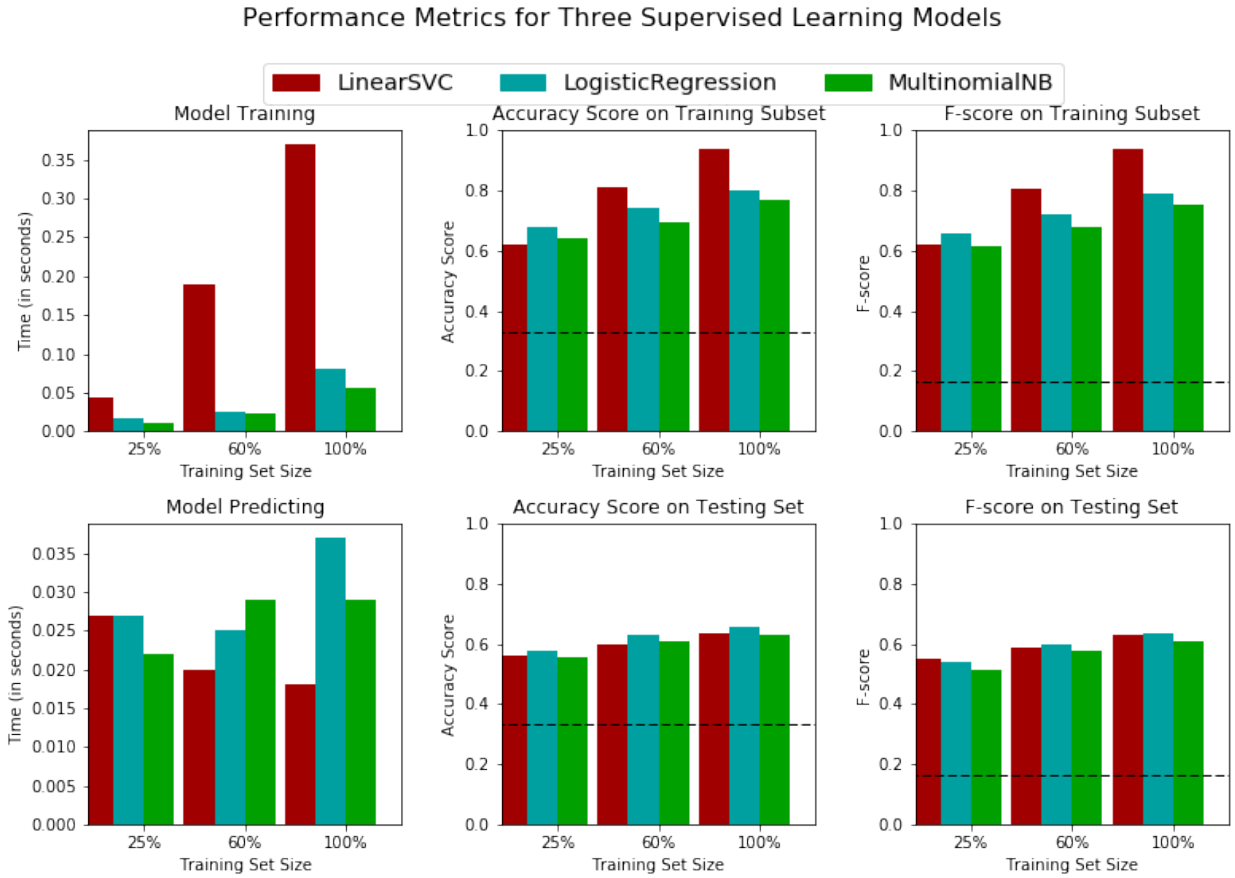
training samples to estimate the evolution of the model's performance on training data. I also experimented with 900 samples and the results were quite similar. I suppose I could have also done predictions on the entire training set. As far as test set is concerned, it was set aside from the beginning and comprises 919 instances. This set is never used for any model training, at any stage, in this project.



The above plots show that all the models significantly outperform the benchmark model baselines, marked by the horizontal dashed lines. The plots also show that Logistic Regression is the clear winner on all metrics – training time, accuracy and F1-score on test data. At the same time, there is a large gap between Logistic Regression's metric values on training and test samples, which indicates overfitting.

The plots below show the performance of the same models on Term Frequency – Inverse Document Frequency (TFIDF) data. To create this data, after a series of trial-and-error, I set the lower and upper bounds on term frequencies at 0.15% and 80%, respectively. This reduced the feature space from 8,884

to 2,297.



We can see from the above plots that, again, all the models outperform the benchmark baselines. We can also see that Logistic Regression still wins on accuracy and F1-score and is the second best in terms of training times. It is also obvious that the use of the reduced feature-size TFIDF, as opposed to raw bag-of-words count, has reduced the overfitting problem – now the gap between the metric values on the training and test samples is much lower.

In summary, Logistic Regression appears to be the best model between the three. The use of simple BoW count, compared to reduced TFIDF, gives 3-4% better accuracy and F1-score on test data but overfitting is severe. I believe it is prudent to accept a few percent lower performance on test data in exchange for the lesser overfitting problem. Therefore, I will finally select the TFIDF approach and try to refine the Logistic Regression on that data in the next section.

Refinement

The default Logistic Regression classifier yields accuracy and F1-score of **0.66** and **0.63**, respectively, on TFIDF data. I fine-tuned the model with `sklearn's GridSearchCV` and managed to increase accuracy and F1-score to **0.66** and **0.64**, respectively, i.e. only one percentage point improvement in F1-score. I believe the fact that the improvement is very minor is due to the relatively small size of the dataset. The whole training set consists of 2,754 observations, and the final test set has 919

observations. When we fine-tune the model with GridSearchCV, a series of cross-validations are performed on the training data, which further reduces the training data to 75% of its initial size, as I applied a 4-fold cross-validation.

The parameter grid which was searched for optimization consisted of C, the regularization parameter and two solver types[4]:

```
param_grid = {
              "C": np.linspace(0.1, 5, 11),
              "solver": ["newton-cg", "lbfgs"]
             }
```

IV. Results

Model Evaluation and Validation

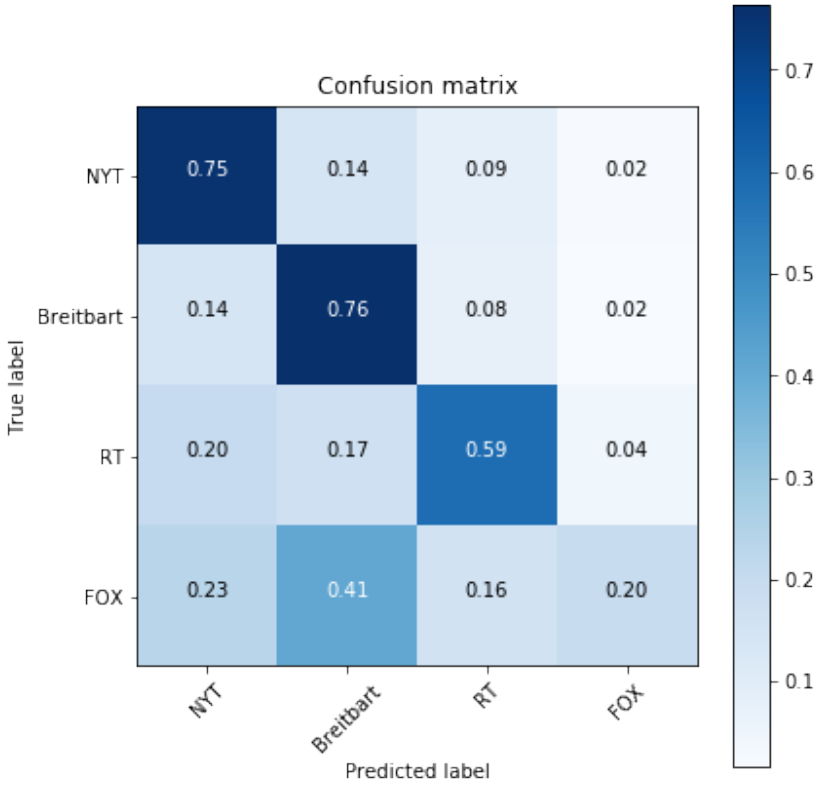
Below is a classification report and confusion matrix for the optimized Logistic Regression model. Please note that these metrics have been generated on the test data, which was not "seen" by the model at any previous stage.

```
Classification report
             precision    recall  f1-score   support

          0       0.69      0.75      0.72       316
          1       0.66      0.76      0.71       312
          2       0.61      0.59      0.60       184
          3       0.53      0.20      0.29       107

Avg / total       0.65      0.66      0.64       919
```

[4] I tried to set the `scoring` parameter of `GridSearchCV` to F1 score, which I created via `make_scorer`. However, I was getting the error "F1 score is ill-defined…", as it appears in some cases predictions were absent for a particular class. Therefore, I resorted to the default scoring parameter of `GridSearchCV`.



Overall, the model's accuracy and F1-score of **66%** and **64%** are significantly better than the benchmark (naïve model) accuracy and F1-score of **33%** and **16%**, respectively.

The model recalls the first two classes (New York Times and Breitbart News) well, the third class (RT) less so and it does not recall the fourth class (Fox News) well at all. This makes intuitive sense – NYT and Breitbart are far-left and far-right institutions, hence they probably cover consistently characteristic topics and use consistently characteristic vocabulary, which the model appears to learn and label effectively. Fox News is a more moderate right-wing source and it appears that some of its content resembles that of Breitbart and some – NYT. Hence the poor classification for Fox.

As far as the model's robustness is concerned, I checked the metrics with different train-test splits and got the following (quite consistent) results:

Random seed	Accuracy	F1-score
42	66%	64%
20	62%	60%
72	65%	63%
56	65%	63%

Justification

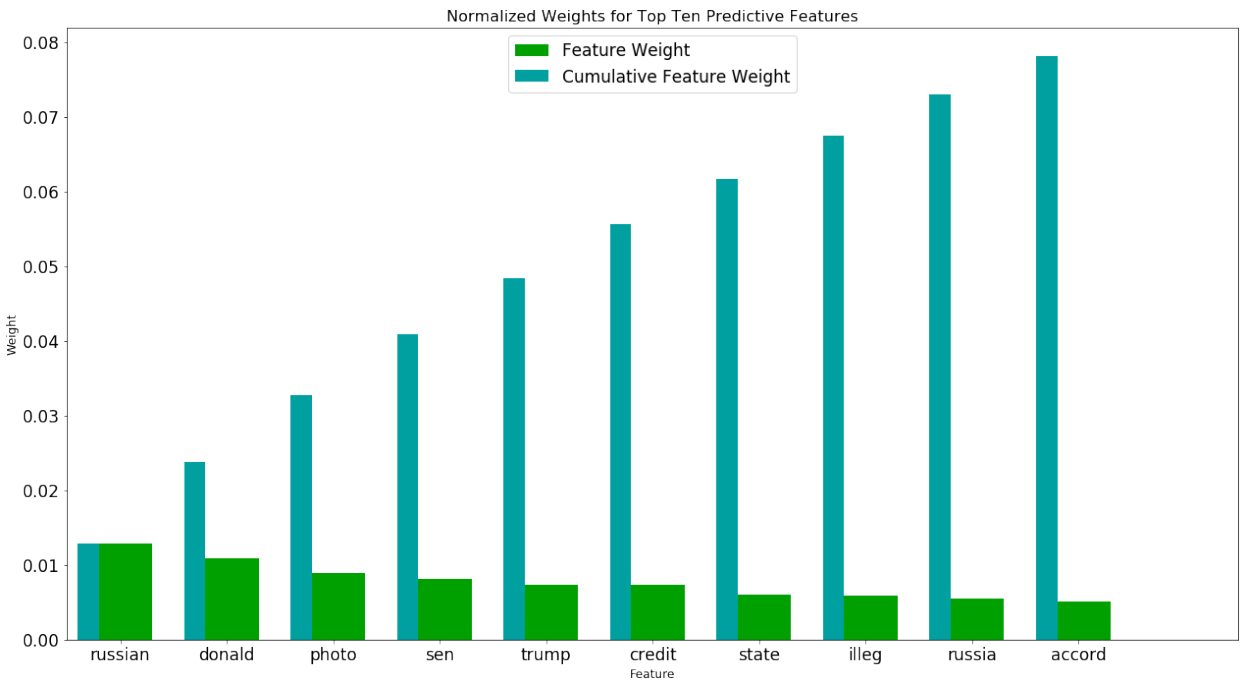
Overall, the model's accuracy and F1-score of **66%** and **64%** are significantly better than the benchmark (naïve model) accuracy and F1-score of **33%** and **16%**, respectively. This is also shown visually in the plots in the Implementation section.

Even though the amount of data used in this project was very modest, I believe the final model serves as a proof that it is possible to create a meaningful prediction capability for various ideological trends from respective text documents such as news and opinion. With bigger and richer data and additional feature extraction, it should be possible to create more powerful models.

V. Conclusion

Free-Form Visualization

It is possible to extract many interesting and intuitively relevant insights from this machine learning exercise. For example, I extracted feature importance information with Random Forest classifier:



Reflection

I undertook the following key steps in this project:

- Obtained the data through a relevant web API
- Consolidated the data in a data frame
- Performed text processing such as tokenization and stemming
- Split the data 75% / 25% for model training and testing
- Created Bag-of-Words and TFIDF matrices
- Defined the relevant performance metrics for model evaluation
- Selected 3 candidate models to evaluate and choose from
- Assessed the performance of the 3 models per metrics chosen previously and selected a model
- Fine-tuned the selected model's parameters using `GridSearchCV`
- Analyzed the final model
- Defined future steps for model improvement

The project was very interesting, as the whole topic of processing text and dealing with NLP was new to me. I researched and became familiar with various ideas and techniques, such as tokenization and stemming, simple bag-of-words and TFIDF feature vectors, Named Entity Recognition, topic models.

I found certain aspects of a multi-class classification problem rather tricky. For example, calculating metrics such as F1-score manually for naïve benchmark model; deriving ROC curves and AUCs for multiple classes. Also, I had a problem with using F1-score as the optimization target in `GridSearchCV`, as described earlier.

Even though the amount of data used in this project was very modest, I believe the final model serves as a proof that it is possible to create a meaningful prediction capability for various ideological trends from respective text documents such as news and opinion. With bigger and richer data and additional feature extraction, it should be possible to create more powerful models.

Improvement

I believe the present text classification problem can be marginally improved by collecting a larger amount of training documents. Also, more sophisticated models such as neural networks, may yield certain improvement if more data were present.

Most importantly, however, I believe this text classification can be improved by new, additional types of feature extraction. Specifically, in the future, I plan to apply Named Entity Recognition (NER) and extract proper names of people, places, etc., as additional features. I also plan to experiment with topic models such as Latent Dirichlet Allocation.