When Was it Written?

Joseph Bakarji, Dimitrios Belivanis, and Sepehr Nezami.

Abstract—We develop a supervised learning algorithm that predicts the publication date of news articles from their content. This study is an exploratory analysis of the relationship between semantic meaning and time. We collect New York Times articles written over 30 years and develop heuristic feature selection measures for choosing the most time-predictive words. We analyze the performance of various supervised learning algorithms, expose the issues faced in NLP data filtering and learning, and proposes solutions. The algorithm correctly classifies around a third of the test set; which is much better than human performance.

Index Terms—Supervised Learning, NLP, Deep learning, Publication date

1 Introduction

Can the language used in an article, blog post or book give us any clues about their date of publication? Google Ngrams [6] reveals that some words are used more often than others during certain periods. This might be due to writing style or important historical events. We would like to develop a supervised learning algorithm that reveals and learns the correlation between the content of an article and its publication date. Applications include assigning a date to blog posts that do not have a date-stamp [3], categorizing articles according to relevant time periods, and more efficient archive navigation and search. After filtering and collecting New York Times articles using their API, we choose words based on heuristic feature selection metrics. Based on those words, we build a bag-of-words matrix with their labels for the training, dev and test sets. Then we use Naive Bayes, Logistic Regression and Neural Networks to predict a date (month and/or year) from the feature words and their frequency. The results are analyzed, and various solutions proposed.

2 Related Work

Our problem can be considered, broadly speaking, a text categorization learning task (which has been done extensively [11]) with dates as categories. The added challenge and new contribution is to incorporate and reveal time-dependent features (or words). More specifically, this study is about machine learning of semantic change. In other words, we want to understand the relationship between the content and time by learning the classification of articles in time. Only recently have researchers started studying semantic change using machine learning methods, especially at Stanford [7], [8]. In [8], Hamilton, Leskovec and Jurafsky attempt to answer the question "What is the role of word frequency in meaning change?" by quantifying semantic change using PPMI, SVD and word2vec. This question motivates our suspicion that publication date is correlated to word frequency. More recent work by [1] analyses text data to track "the semantic evolution of individual words over time". These studies attempt to model semantic change by analyzing word clusters in a semantic space but do not explicitly try to predict the date out of which these semantics come. In relation to this area of research, our study can be seen as a generative model for semantic change where, instead of studying the semantics given the date, we want to find the date given the semantics.

3 Dataset and Features

3.1 Data Collection

Finding a reliable and digitized source of news articles and collecting it is a lengthy and tedious process. First, we got access to the New York Times archive metadata through their API [10]. Having downloaded the metadata for the past 30 years (1987 to 2017), we sampled url's and downloaded the content of a 1000 articles/month. Once we had all the articles, we compressed them in tokenized JSON files (5MB/month) containing a dictionary of articles each with a list of content words (features), the date (labels) and the title. The data went through

• *Emails: {jbakarji, dblivan, nezami}@stanford.edu*

multiple stages of data cleaning as we discovered that many NYT pages only contain a photo or a table and some do not have useful content (like a paid death notice).

Out of the 1000 articles/month, our training set consists of 800 articles/month, the dev set 100 articles/month, and the test set a 100 articles/month as shown in fig. 1.

3.2 Feature Selection

Ideally, we would like to choose the words that have the most predictive power in our model. A naive way would be to add (or remove) one word at a time and see how our algorithm performs. But this is impossible given that our feature space is very high dimensional, i.e. $\sim 10^5$ words. Therefore, we had to come up with heuristic methods for measuring the "time-relevance" of a word. We therefore give higher preference for words with higher relative fluctuation. We define a *score* that measures the *total variation* of word usage, normalized by the average frequency. The *score* of a word w in a given time range $[y_0, y_e]$ is given by

$$\text{score}(w) = C(w)\frac{\sum_{y=y_0}^{y_e-1}|f_w(y+1)-f_w(y)|}{\sum_{y=y_0}^{y_e} f_w(y)} \tag{1}$$

where $f_w(y)$ is the probability that a randomly chosen word in year y is w, and $C(w)$ is a heuristic pre-factor that gives higher weight to the words that appear more. Currently, $C(w) := \log(\sum_{y=y_0}^{y_e} f_w(y))$. Based on this measure, we choose the N most time-informative words as our feature space, out of a total of around $300\,000$ identified words. The top 10 words of the last 10 years are shown in Table. 1. These words correspond to important events and are therefore good indicators of publication time.

In practice, we use a python NLP package (NLTK) [5] to tokenize and remove unwanted words such as stopwords, punctuations, non-english words, and to finally stem the words in order to reduce the number of features and combine words with the same meaning like “dogs” and “dog”, or “finding” and “find”. Word stemming turns out to be computationally expensive, so we generated a stemmed archive from which we build a bag-of-words matrix every time we change the parameters.

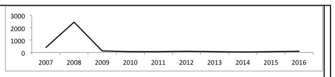
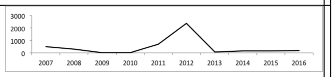
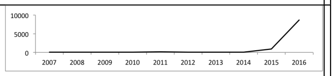
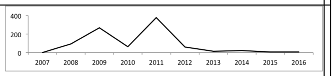
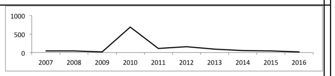
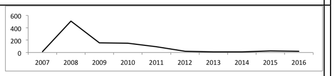
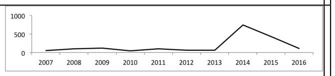
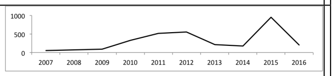
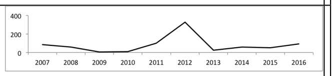
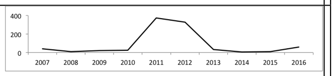
Word	Score	# Occurrence per year plot
mccain	76.0	
romney	72.2	
trump	60.4	
madoff	58.1	
bp	55.6	
palin	46.1	
ukraine	44.1	
greece	44.0	
mitt	42.3	
gingrich	40.5	

Table 1
Top 10 most time informative word for years 2007-2016.

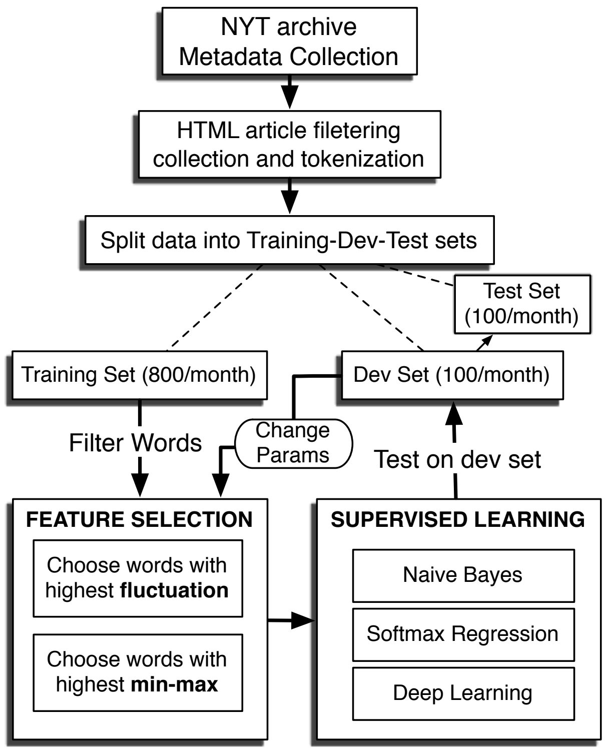

NYT archive Metadata Collection
HTML article filetering collection and tokenization
Split data into Training-Dev-Test sets
Test Set (100/month)
Training Set (800/month)
Dev Set (100/month)
Filter Words
Change Params
Test on dev set
FEATURE SELECTION
Choose words with highest fluctuation
Choose words with highest min-max
SUPERVISED LEARNING
Naive Bayes
Softmax Regression
Deep Learning

Figure 1. General workflow: data collection, feature selection, supervised learning and iterative parameter optimization

4 Methods

Once the articles have been collected, the words stemmed, and the features chosen, we build a Bag-of-words (BOW) matrix X where each entry X_{ij}

corresponds to the number of times word j appears in article i. Each row (article) of X is normalized by the norm. And each article in the training, dev and test sets is labeled with a date (or a one-to-one mapping to it) y_i. But how many labels should we use? 360 classes corresponding to each month-year pair in 30 years, or 30 classes corresponding to the years? Would both lead to the same general result, and how would those results be interpreted?

The added challenge in this NLP learning task is that we are not simply interested in correctly predicting the labels or maximizing the log-likelihood $\log p(y_i|x;\theta)$ with parameters θ. For example, a model that predicts the day would never generalize. Instead, we would like our algorithm to minimize the time between the predicted and the actual date, or minimize a cost function of the form $J(\theta) = 1/2\sum_i(h_\theta(x) - y_i)^2$ where $h_\theta(x)$ is the predicted date and y_i the actual date. However, simple linear regression, as will be seen in section 5.2, does not lead to good results because it assumes the error to be Gaussian around the predictions $h_\theta(x)$ which is clearly not our case. Ideally, we would like to find the actual distribution of the error and build our own GLM model but our preliminary analysis will be restricted to classification algorithms like Softmax, Naive Bayes and Deep Learning.

4.1 Learning Algorithms

In the Softmax algorithm, we assume our dates can take any one of k (360 or 30) values and would like to find the parameter θ that maximizes the conditional distribution of the date y_i given the word vector x where [9]

$$p(y_i|x;\theta) = \frac{\exp \theta_i^T x}{\sum_j^k \exp \theta_j^T x} \tag{2}$$

Or more specifically its total log-likelihood given by $\mathcal{L}(\theta) = \sum_j \log p(y^{(j)}|x^{(j)};\theta)$. The advantage of using Softmax over linear regression is a higher flexibility to fit the nonlinear relationship between word content and publication date.

In Naive Bayes, we assume that the words are conditionally independent given the date and would thus like to maximize the joint likely-hood $\prod_i p(x_i|y)$. Here, we heuristically observe that the frequency of mostly feature words comes from an *exponential* distribution, whose mean is to be learned individually for different months and words.

We further use *Principal Component Analysis* (PCA) on the BOW matrix and perform *Gaussian Discriminant Analysis* (GDA) on the components instead of the feature words. The basic idea is that the different PCA components should have less correlations comparing to the feature words, and therefore, may lead to better results.

Simple linear regression is performed on the BOW matrix to predict the publication date (as a continuous variable) from the feature words. We also consider a *compound model* of Naive Bayes + Linear Regression. We combine these models by multiplying probabilities[1] of the predictions from both models to construct a better model.

In deep learning, two variations of a single layer neural network were implemented. In the first, the final layer is a softmax classifier where each year is a separate category. In the second, the final layer is a linear regression over the outputs of the first layer. The year is predicted in a continuous time frame and the prediction is rounded to fit discrete yearly categories.

Our neural networks were implemented on python with the Tensorflow library [4]. The loss functions were defined as the built in cross entropy function for the softmax and square difference for the linear regression. Different activation function were used for each network as the combination of Relu-softmax and sigmoid-linear regression performed better. Different combination of learning rates were also investigated for gradient descent (with and without momentum) but eventually the Tensorflow Adam algorithm was able to converge in most cases.

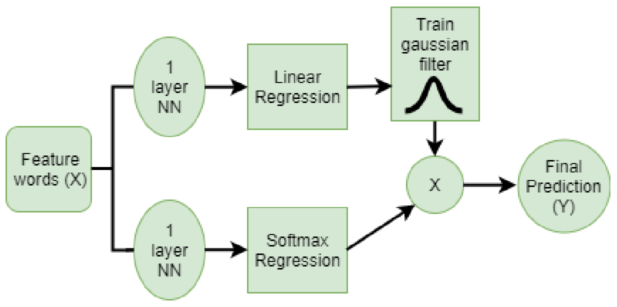


Figure 2. Neural Network hybrid method combining a linear regression and Softmax output

To improve our results we combine the two

1. To assign probability distributions to the outputs of the linear regression model, we take the predicted date of the LinR model, and widen it to a Gaussian probability distribution picked at the predicted date with the variance equal to the variance of the prediction error of the linear regression model, alone. Then multiply these probabilities to the prediction probabilities of NB, where the relative weight of the models is optimized on the Dev set.

models by convoluting the linear regression with a Gaussian PDF and multiplying it with the probability histogram generated by Softmax NN (similarly to what has been used in NB+LinR). This method increases the validity of the model since each individual model can describe different patterns. a visualization of this work flow can be seen in fig. 2

Finally, an iterative method for feature refinement has also been used based on the relative success of words to predict the given year, according to

$$\text{score}_p(\text{w}) = log\frac{P(\text{w}|\text{correct pred.})}{P(\text{w}|\text{false pred.})} \qquad (3)$$

but this approach did not yield better results especially when the number of chosen features is close to the set of all possible features after filtering.

5 Experimental Results and Discussion

5.1 Experiments

The inputs of our algorithm are the learning method, the desired number of features, the feature selection method (e.g. eqns. 1 and 3), the training set size, the start and end dates, monthly or yearly classification, and an optional filtering of articles with less than a certain number of words.

Our "first thing to try" machine learning algorithm was a multinormal Naive Bayes algorithm with a number of categories equal to the number of years or months analyzed. The preliminary results for monthly (360 classes over 30 years) classification has resulted in over-fitting on the training set with 90% accuracy on the training set and 5% accuracy on the dev set. This gave us an indication to run the rest of our experiments using years as classes which we realized generalizes better.

For feature selection, we ran multiple experiments using words that have the highest min-max, standard deviation and total fluctuation. Testing those methods on Naive Bayes showed very similar results. We therefore chose to stick to the total fluctuation (Eqn. 1) metric.

Subsequently, our analysis revolved around comparing different learning algorithms (NB, softmax, DL, and LR) with varying training set size, and various number of features.

5.2 Results

In this study we will report our results in the form of accuracy and average time difference between the predicted and actual dates. More generally we can get the full distribution of the time error as show in Fig. 4. In this context, a confusion matrix will not be as informative given that our classes are a discretization of a continuous metric (date).

The first step in our experiments was to see if we need more training examples and if the number of features we are using generalizes. For this we plot the accuracy of the test and dev sets as a function of the training set size per month, shown in Fig. 3. These experiments were run with 6000 words in the feature space using the average fluctuation as a feature selection method.

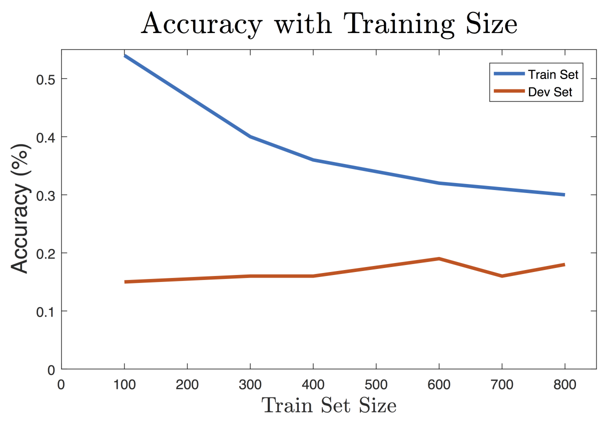


Figure 3. Accuracy of training set and dev set as a function of training set size. The gap suggests that more data is needed and the dev set not generalizing

The accuracy of the training set decreases and that of the dev set increases with the training set size as expected. However, the dev set error increases very slowly leaving a gap between the two sets. The gap suggests that we should collect more data but the desired error is still low. So we are faced with both large variance and large bias. Furthermore, we are unable to deal with more data given our computational resources were limited for this project (on our laptops); already running into many memory problems due to the large size of our training BOW matrix.

We further observed that the performance of NB can be significantly improved by combining it with the linear regression model. LinR itself does not perform well for exact predictions, however when combined with NB, it improves on both exact prediction probabilities and Avg. time error. See fig. 4

The two main disadvantages for the deep learning were that first it was prone to overfitting and second that the optimization of the loss function was highly dependent on hyper-parameters. The first issue was overcome by using fewer neurons

Model	Train set acc.	Dev set acc.	acc. within year	acc. within 2 years.	Avg. time error
NB-sklearn [2]	35%	24%	32%	41 %	5.89 yrs
LogR-sklearn	30%	14%	23%	33 %	6.28 yrs
NB-MAT	31%	12%	28%	40%	8.02 yrs
LinR-MAT	31%	7%	21%	34%	5.77 yrs
LinR + NB	31%	13%	31%	45%	5.71 yrs
GDA+PCA	20%	10%	19%	26%	11.2 yrs
NN+SM	33.2%	12.4%	26.1%	35.7%	6.60 yrs
NN+LinR	6.1%	6.3%	19%	30%	5.44 yrs

in the hidden layer, thus minimizing the degrees of freedom of the model. This change had a big impact on the accuracy of the training set but helped better generalize the model, improving the performance on the dev set.The most accurate results were achieved for a neural networks of 16 Relu neurons with accuracy of 13.9%. The network with a linear regression on the output underperformed in terms of accuracy but could achieve a much better performance on the average time error.

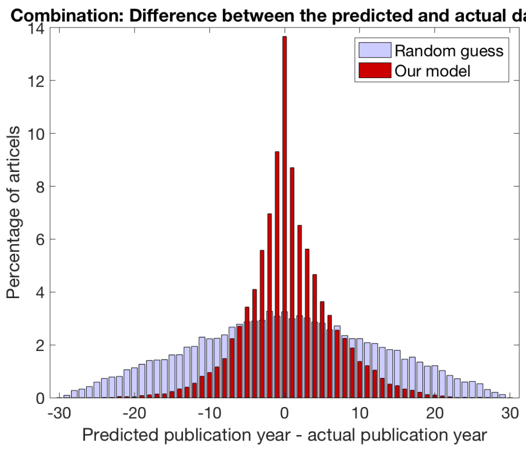


Figure 4. Distribution of time error between the actual and predicted date of publication. The predictability is measured as the volume of the histogram not overlapping with the random guessing histogram.

5.3 Discussion and Future Work

This study is a preliminary analysis of the relationship between word usage and time. The first challenge was finding the appropriate set of features. Our proposed measure given by Eqn. 1 has captured important events as shown in Table. 1 but also included noisy words that would not help our learning algorithm generalize. Some smoothing to the features was applied but this also removed important monthly events. Furthermore, the assumption has been all along that words with fluctuating peaks will inform us about specific events; but in general smoothly increasing word frequencies can also inform us about the time. A better heuristic measure has to be developed in future studies.

The original aim of this study was to analyze articles over a 100 years, but we decided to start with a toy problem. However, we expect our accuracy to get much better when analyzed over a larger time range because words change more over longer time periods. In other words, the noise will become less significant and our model will generalize better over time scales with larger semantic change.

In our study we focused on classification methods but the variable we are trying to predict (time) is continuous. It would make more sense on the long run to find the noise distribution around our mean prediction and derive a GLM that corresponds more closely to our data given a prediction. The nature of the problem points towards a Poisson distribution with words 'arriving' at an article independently of one another. This is worth investigating in the future.

In the future, we would like to explore the dynamics of the semantic change using Word2Vec. Word2Vec has been used extensively for clustering and analyzing semantics as in [8]. The authors are very interested to explore the dynamics of semantic spaces and derive its 'laws of motion' using machine learning. For example, words that tend to have a fixed meaning will tend to have a rigid relationship or structure. One strategy would be to do Word2Vec on each year and for each article find the space in which the words take the least 'semantic volume'.

6 Conclusion

This study has revealed a relationship between word frequency usage in news articles and date of publication. The relationship has been observed but only recently quantified using Machine Learning. Our contribution to the field is a new generative model that predicts the date given the semantics. The results can be improved by considering a larger time range and optimizing the hyper parameters.

7 Contributions

The contribution of each member to the project so far is as follows:

- Joseph Bakarji wrote a code in Python for querying and saving archive metadata for a given range of time. Also filtering, tokenizing, stemming the data and building the BOW for the training/dev/test sets. Also worked on NB and LogR using the sklearn libraries.
- Sepehr Nezami wrote a code in C that tokenizes html content, counts the number of words for various time frames, and devised various measures of feature selection, and worked on the Naive Bayes algorithm. Used PCA with NB and PCA. Came up with a method to combine NB and linear regression.
- Dimitrios Belivanis analyzed word count statistics for feature selection, and focused on developing various methods for softmax and neural networks in Tensorflow. Came up with a method for combining linear regression with neural networks for continuous predictions.

The task separation is not sharp. The team discussed the project at all its stages and each member has been contributing to every task accomplished in this study.

References

[1] Robert Bamler and Stephan Mandt. Dynamic word embeddings. In *International Conference on Machine Learning*, pages 380–389, 2017.

[2] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[3] Jonnathan Crossfield. Should you date-stamp your blog content?, 2014.

[4] Martin Abadi et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] S. Bird et al. Natural language tool kit (nltk), http://www.nltk.org/, 2017.

[6] Google. Google ngrams, books.google.com/ngrams, 2017.

[7] William L Hamilton, Jure Leskovec, and Dan Jurafsky. Cultural shift or linguistic drift? comparing two computational measures of semantic change. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, volume 2016, page 2116. NIH Public Access, 2016.

[8] William L Hamilton, Jure Leskovec, and Dan Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096*, 2016.

[9] Andrew Ng. Supervised learning and discriminative algorithms. *ML CS229 Lecture Notes*, pages 1–30, 2017.

[10] NYTimes. New york times achrive api, https://developer.nytimes.com/, 2017.

[11] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.