Automated Sarcasm Detection

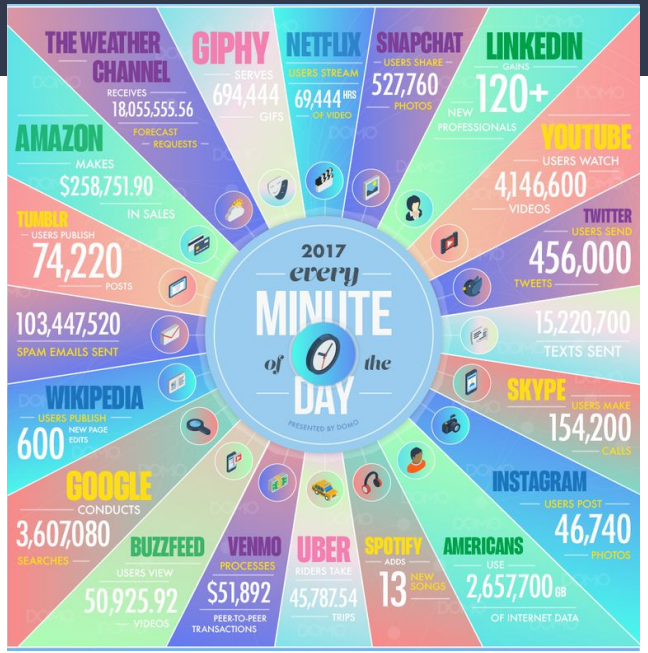
Presentation by Jonathan Waring and
Jonathan Hayne

Sarcasm Detection

- Sarcastic comments have become quite common on the Internet → especially on sites where people are more likely to express opinions (i.e., product reviews or social media)
- Sarcasm can be ambiguous, which makes it difficult for even humans to detect if a comment is sarcastic or not
- Sarcasm is usually detected in face-to-face communication through vocal tone and facial cues that communicate the speaker's intent → detecting it in plain writing is much more difficult

Why bother?

- There's actually been quite a few people who have expressed interest in the area, including the Secret Service, which posted a request for an automated software that detected whether threats on Twitter were serious or not



There are massive amounts of text data being generated every minute, including 456,000 tweets, 74,220 tumblr posts, 600 new Wikipedia edits, 15,220,700 texts sent, etc

Why is this such a hard problem?

- NLP algorithms have been successful in summarizing text, creating chatbots, classifying documents, identifying the emotional sentiment behind text, and automated question answering.
- These algorithms are typically programmed to process text exactly as they see it
- This is problematic when you consider the fact that sarcasm is just saying the opposite of what you mean → sarcasm inherently conveys contradictions
- How do we identify sarcasm from plain text alone?

The Dataset

Self-Annotated Reddit Corpus (SARC)

- The SARC dataset is a large corpus for sarcasm research and for training and evaluating systems for sarcasm detection that was put together by the Computer Science Department at Princeton University[1]
- Each statement is self-annotated — sarcasm is labeled by the author, not an independent annotator — and provided with user, topic, and conversation context
- All the statements come from Reddit, a social media site in which users communicate by commenting on **submissions**, which are titled posts consisting of embedded media, external links, and/or text, that are posted on topic-specific forums known as **subreddits**

Problems with Self-Annotation

- Reddit users have adopted a common method for sarcasm annotation consisting of adding the marker “/s” to the end of sarcastic statements
- As with Twitter hashtags, using these markers as indicators of sarcasm is noisy [2]
  - Many users do not make use of the marker, do not know about it, or only use it where sarcastic intent is not otherwise obvious
- The creators of the corpus estimated a false positive rate of 1.0% and a false negative rate of 2.0%

Data Points

- The SARC dataset contains over 1.3 million quote-response pairs, but the dataset is very imbalanced (approximately 10x more non-sarcasm comments)
- Each datapoint in the dataset also contains a variety of metadata surrounding the actual text including:
    - Author of comment, number of upvotes, number of downvotes, subreddit, date posted, etc.
- Our goal was to create a general automated sarcasm detector from text alone, so we choose to ignore a lot of this metadata (we do acknowledge that this metadata could be potentially useful)
- We also choose to use only a subset of the SARC dataset in order to give us balanced labels
    - We end up with 255,172 training examples and 63,979 testing examples from over 4,000 different subreddits
    - Each quote has at least one sarcastic and one non-sarcastic response

Examples from the SARC Dataset

Sarcastic

- Quote: AMA Request: Anyone involved or has been involved in Organized Crime
  - I think there has already been an AMA for someone working in Goldman Sachs
- Quote: Jeb Bush defends Paul Ryan's lies
  - Wow this is the first Bush to ever lie to the public.

Non-Sarcastic

- Quote: AMA Request: Anyone involved or has been involved in Organized Crime
  - Nice try, newly "made" mob dude seeking out rats to report to your fucking boss.
- Quote: Jeb Bush defends Paul Ryan's lies
  - Jeb Bush - 'nuff said.

Feature Extraction

Preprocessing

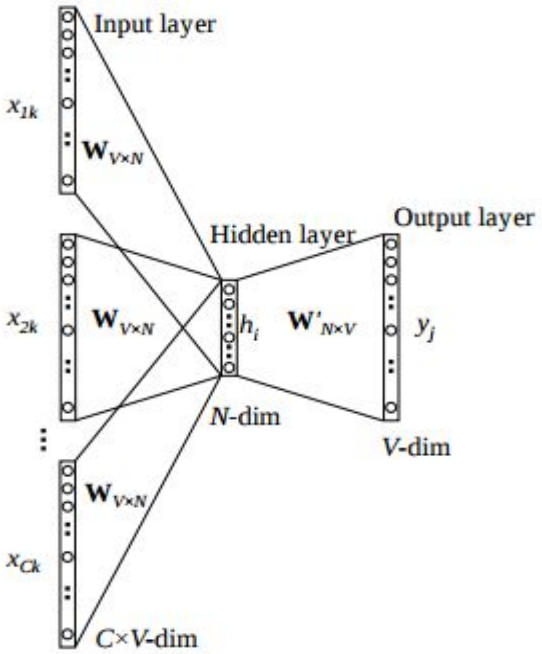
- Stemming
  - “Boiling” a word down to its base meaning
  - Didn’t use stemming
    - how a word is said is almost as important as the content of the word
- Removed Stopwords
  - the, are, that, etc.
- Tokenization
  - Twitter versus Word Tokenizer
  - We used Twitter tokenizer
    - Very flexible

Word2Vec

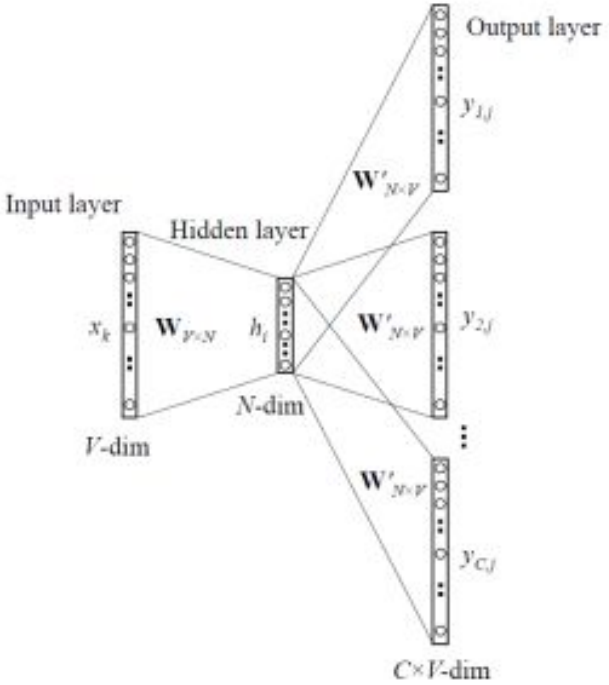
- Turn a corpus into a vector space
- Combines CBOW and Skip Gram Models
- Parameters:
  - Context window (7)
  - Dimension (300)
    - Higher dimension yields better results but slower training time
    - Typical word2vec dimension sizes are 100-300
  - N-gramming

CBOW

- Predicts probability of a word given a context
- Topology
  - Input layer is one hot encoded context window of each word
  - Output layer is vocabulary
  - Hidden layer has same number of nodes as word2vec dimension size
  - Cost function is negative log likelihood of a word given a set of context
  - Linear Activation



Skip-Gram



Input	Output(Context1)	Output(Context2)
Hey	this	<padding>
this	Hey	is
is	this	sample
sample	is	corpus
corpus	sample	corpus
using	corpus	only
only	using	one
one	only	context
context	one	word
word	context	<padding>

- Same as CBOW but reverse architecture
- Predicts probability of a context given a word
- Context window of 1 on left

Afinn

- Database of words and their corresponding valence
  - Measures how positively or negatively charged a word is
- Valence values between -5 (very negative) and +5 (very positive)
  - Ie abandon has score of -2
- Average each individual word’s score to get a sentence’s valence

NRC Word-Emotion Association

- Database with ~14000 words and their associations with 8 emotions
  - Anger
  - Anticipation
  - Disgust
  - Fear
  - Joy
  - Sadness
  - Surprise
  - Trust
- Generated from Tweets hashtagging the above emotions
- Average each individual word’s emotion vector to get emotion vector for a sentence

Standardization of Feature Vectors

- Structure of Feature Vector:
  - [300 dimensional word embedding, emotion vector, valence]
- Why standardize?
  - Necessary for neural nets
    - Feature vectors with larger magnitude have greater influence on estimators
  - Speeds up LR training time
- Don’t need to standardize for NB and XGBoost but we did anyway
- Used Scikit-learn StandardScaler

Machine Learning Models and Results

Naive Bayes

- Even though the independence assumptions of Naive Bayes are often far-fetched, it still performs well in practice
- We use Gaussian Naive Bayes, which assumes that the continuous values (features) associated with each class are distributed according to a Gaussian distribution
- Thus each observation value (comment), v, has its probability distribution given a class C_k, $p(x = v \mid C_k)$ computed by plugging v into an equation for a normal distribution parameterized by μ_k and σ^2_k:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Naive Bayes

- Was by far the fastest to train, but also performed the worst
- Naive Bayes took 0.22 minutes to train

Training Accuracy	54.18%
Test Accuracy	54.21%
Testing Precision	53.31%
Testing Recall	70.58%

Logistic Regression

- Logistic regression uses the concept of odds ratios to calculate the probability of a certain outcome of the dependent variable

$$Odds = \frac{P(y=1|x)}{1-P(y=1|x)} \xrightarrow{\text{Logit transformation}} Logit(P(x)) = \ln\left(\frac{P(y=1|x)}{(1-P(y=1|x)}\right)$$

- In Logistic regression the Logit of the probability is said to be linear with respect to x, so the logit becomes:

$$Logit(P(x)) = a + bx \longrightarrow \frac{P(y=1|x)}{1-P(y=1|x)} = e^{a+bx} \longrightarrow P(y=1|x) = \frac{e^{a+bx}}{1+e^{a+bx}} = \frac{1}{1+e^{-(a+bx)}}$$

- More generally:

$$P(y=1|x) = \frac{1}{1+e^{-(w^T x)}}$$

Logistic Regression

- Not much difference between Logistic Regression and Naive Bayes
- Logistic Regression took 3.35 minutes to train

Training Accuracy	54.73%
Test Accuracy	54.64%
Testing Precision	54.74%
Testing Recall	55.39%

Logistic Regression Optimized

- Performed a Grid Search to optimize for Logistic Regression regularization parameter
- Grid Search took 45 minutes
- Best parameter value was C = 1000 (C is described as the inverse of regularization strength in Scikit Learn's documentation)

Training Accuracy	59.68%
Test Accuracy	59.32%
Testing Precision	59.56%
Testing Recall	58.97%

XGBoost

- XGBoost is short for “Extreme Gradient Boosting”
- XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions
- Boosting is an ensemble technique where new models are added to correct the errors made by existing models.
- Gradient boosting[3] is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction.
- Incredibly powerful/flexible → **MANY** hyperparameters to tune

XGBoost

- Little better than Naive Bayes and Logistic Regression
- XGBoost took 20.40 minutes to train

Training Accuracy	60.71%
Test Accuracy	59.46%
Testing Precision	59.98%
Testing Recall	57.79%

XGBoost Optimized

- Performed a Randomized Search to optimize 6 of XGBoost's hyperparameters
- Randomized Search took almost a day to run
- Almost no improvement

Training Accuracy	71.73%
Test Accuracy	61.24%
Testing Precision	61.89%
Testing Recall	59.22%

Neural Network in TensorFlow

- Is it really a Data Science project if we don't try to use a neural network?
- Built our Neural Network model in TensorFlow using the following:
  - ReLU activation functions
  - Gradient Clipping using global norm [4] to account for vanishing gradients
  - Minibatch gradient descent
  - Adam optimizer [5]
  - Dropout and L2 loss for regularization
  - Convergence threshold of 1.0×10^{-5}
  - Several different architectures

Neural Network - 1 hidden layer

No Regularization

Training Time	11.46 minutes
Training Accuracy	62.77%
Testing Accuracy	59.65%

Dropout Regularization

Training Time	4.32 minutes
Training Accuracy	59.92%
Testing Accuracy	59.24%

L2 Regularization

Training Time	2.44 minutes
Training Accuracy	53.45%
Testing Accuracy	52.30%

Dropout & L2 Regularization

Training Time	3.20 minutes
Training Accuracy	53.05%
Testing Accuracy	52.07%

Neural Network - 2 hidden layers (all same size)

No Regularization

Training Time	53.76 minutes
Training Accuracy	83.40%
Testing Accuracy	56.11%

Dropout Regularization

Training Time	25.82 minutes
Training Accuracy	65.04%
Testing Accuracy	58.95%

L2 Regularization

Training Time	4.44 minutes
Training Accuracy	54.64%
Testing Accuracy	52.30%

Dropout & L2 Regularization

Training Time	7.14 minutes
Training Accuracy	53.79%
Testing Accuracy	52.44%

Neural Network - 2 hidden layers (varying sizes)

No Regularization

Training Time	96.21 minutes
Training Accuracy	92.32%
Testing Accuracy	55.31%

Dropout Regularization

Training Time	76.94 minutes
Training Accuracy	70.66%
Testing Accuracy	57.70%

L2 Regularization

Training Time	28.77 minutes
Training Accuracy	59.37%
Testing Accuracy	52.41%

Dropout & L2 Regularization

Training Time	14.91 minutes
Training Accuracy	54.08%
Testing Accuracy	52.73%

Neural Network – 3 hidden layers (varying sizes, all shrinking)

No Regularization

Training Time	100.34 minutes
Training Accuracy	93.15%
Testing Accuracy	55.12%

Dropout Regularization

Training Time	55.37 minutes
Training Accuracy	67.98%
Testing Accuracy	57.60%

L2 Regularization

Training Time	28.24 minutes
Training Accuracy	59.86%
Testing Accuracy	52.05%

Dropout & L2 Regularization

Training Time	8.69 minutes
Training Accuracy	53.09%
Testing Accuracy	52.32%

Neural Network – 3 hidden layers (varying sizes, expanding then shrinking)

No Regularization

Training Time	349.50 minutes
Training Accuracy	96.64%
Testing Accuracy	54.50%

Dropout Regularization

Training Time	198.92 minutes
Training Accuracy	74.77%
Testing Accuracy	56.55%

L2 Regularization

Training Time	138.23 minutes
Training Accuracy	59.36%
Testing Accuracy	52.18%

Dropout & L2 Regularization

Training Time	26.81 minutes
Training Accuracy	53.16%
Testing Accuracy	52.36%

Neural Network – 3 hidden layers (varying sizes, all shrinking) with varying Dropout

20% Dropout Regularization

Training Time	55.37 minutes
Training Accuracy	67.98%
Testing Accuracy	57.60%

30% Dropout Regularization

Training Time	153.15 minutes
Training Accuracy	69.78%
Testing Accuracy	57.87%

40% Dropout Regularization

Training Time	82.60 minutes
Training Accuracy	65.07%
Testing Accuracy	58.41%

50% Dropout Regularization

Training Time	26.81 minutes
Training Accuracy	61.07%
Testing Accuracy	58.97%

Discussion of Results

- Our best testing accuracy came from XGBoost with a 61.24% testing accuracy
  - 61.89% of sarcastic classified comments were actually sarcastic
  - Correctly classified 59.22% of the sarcastic comments
- Had lots of problems with our neural networks overfitting even with dropout
  - While adding L2 penalties to our loss function helped reduce overfitting, we also got our lowest training/testing accuracies with the added penalties
  - The best testing accuracy from a neural network was from the 1 hidden layer network (59.65%) with no regularization. This is barely better than the optimized Logistic Regression model at 59.32%
  - Got training accuracy as high as 96.64%, but resulted in testing accuracy of only 54.50%

Conclusion

- A similar study that attempted to detect sarcasm on Twitter achieved testing accuracies around 70% using lexical and pragmatic features, along with feature selection techniques [6]
  - Word2Vec, while supposedly better at capturing context, did not help us in our performance. We also only used 22 lexical features, which appear to perhaps be more helpful in this case.
- This same study also showed that human accuracy in detecting sarcasm in plain text on Twitter was only around 62.59% [6]
  - Our best testing accuracy was 61.24%, so we are close to the human baseline
- Potential ways to address this problem in the future:
  - Consider metadata surrounding each sarcastic comment
  - Maybe work on detecting sarcasm within subreddits before trying to build a general sarcasm detector
  - Use more lexical features and maybe employ feature selection techniques

References

[1] Khodak, Mikhail, Nikunj Saunshi, and Kiran Vodrahalli. "A Large Self-Annotated Corpus for Sarcasm." *arXiv preprint arXiv:1704.05579* (2017).

[2] Bamman, David, and Noah A. Smith. "Contextualized Sarcasm Detection on Twitter." *ICWSM*. 2015.

[3] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." *Annals of statistics* (2001): 1189-1232.

[4] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International Conference on Machine Learning*. 2013.

[5] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[6] González-Ibánez, Roberto, Smaranda Muresan, and Nina Wacholder. "Identifying sarcasm in Twitter: a closer look." *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers-Volume 2*. Association for Computational Linguistics, 2011.