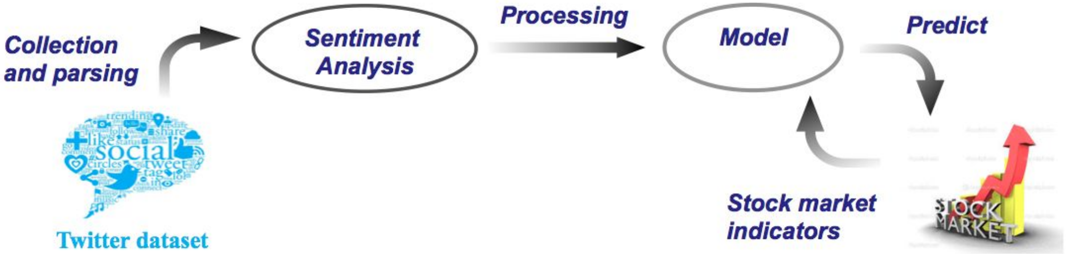
CAB420 Project

Sentiment Analysis of Statements

About the problem - Motivation

- Possible applications of text sentiment analysis:
  - Brands can monitor their public image
  - Inform stock trading strategies
  - Use in economics research
  - Litigation
- Additional Benefits of text
  - Easy to find training and test data
  - Training completes quickly



Prior research - Common techniques and effectiveness

- Models commonly used in sentiment analysis:
  - Naive Bayes - 81.3% accuracy (sentiment140)
  - Max Entropy - 80.5% accuracy (sentiment140)
  - SVM - 82.2% accuracy (sentiment140)
  - Neural Networks - Usually even higher accuracy
- Only lexicon analysis has been used on this dataset

Data set

- Data consists of:
  - 6918 Annotated Statements
  - Annotated with 1 for a positive emotion and 0 for a negative emotion, 3943 positive, 2975 negative
  - 10% retained for test data, another 10% used for validation in LSTM
  - Somewhat simple dataset

Sample extract from data:

```
1       Anyway, thats why I love " Brokeback Mountain.
1       Brokeback mountain was beautiful...
0       da vinci code was a terrible movie.
0       Then again, the Da Vinci code is super shitty movie, and it made like 700 million.
0       The Da Vinci Code comes out tomorrow, which sucks.
```

Data pre-processing

- Using *TextAnalytics Toolbox:*
  - Removing short, long and insignificant words
  - Removed punctuation
  - Removed “Stop Words” (e.g. ‘and’, ‘a’, ‘the’)

Before:
I can only imagine, with the bloated egos and arrogant liberalism at UQ, this effect was magnified...

After:
only image bloated egos arrogant liberalism UQ effect magnified

Problem: Curse of dimensionality

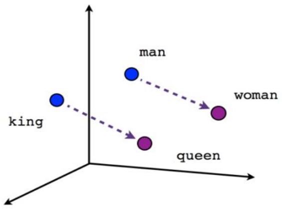
- Even after data processing, way too many words to fit models on
- d = 2000+ when n = 6918
- Results in undefined systems for logistic regression
- Too many parameters to train in neural network
- Inaccurate estimates for priors in naive bayes
- Long training and data prep time

Better solution: Word embedding

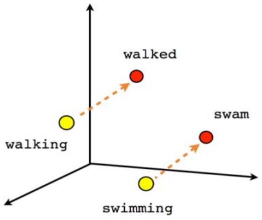
Used Text Analytics Toolbox to utilise *fastTextWordProcessing*.

Also trained own embedding that worked better, reduced model from d = ~2000 to 100

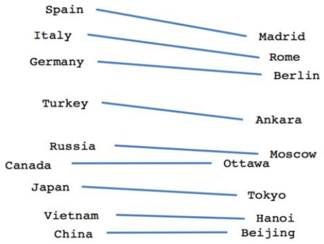
word2vec to easily maps words to numeric vectors using the embedding



Male-Female

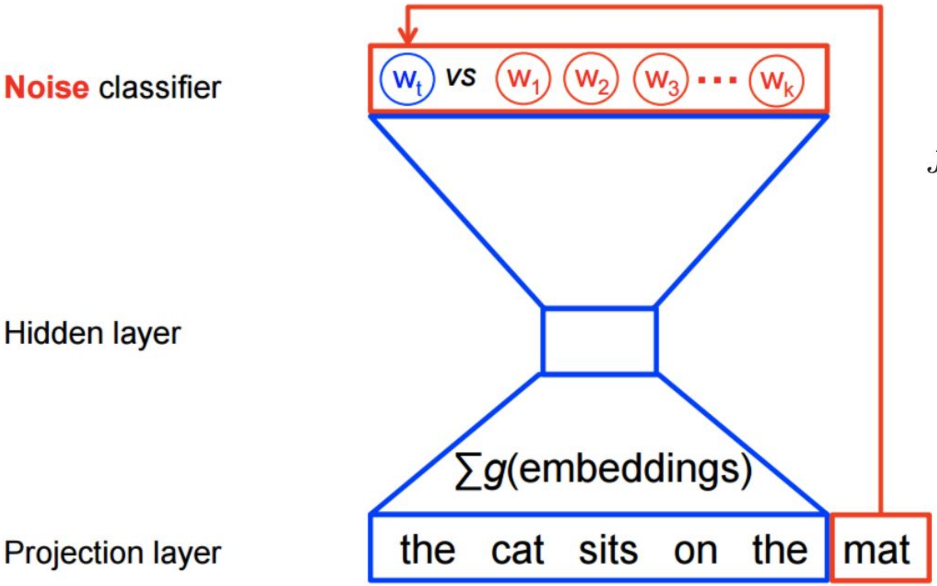


Verb tense



Country-Capital

How word embedding words



$$J_{\text{NEG}} = \log Q_\theta(D=1|w_t, h) + k \underset{\tilde{w} \sim P_{\text{noise}}}{\mathbb{E}} [\log Q_\theta(D=0|\tilde{w}, h)]$$

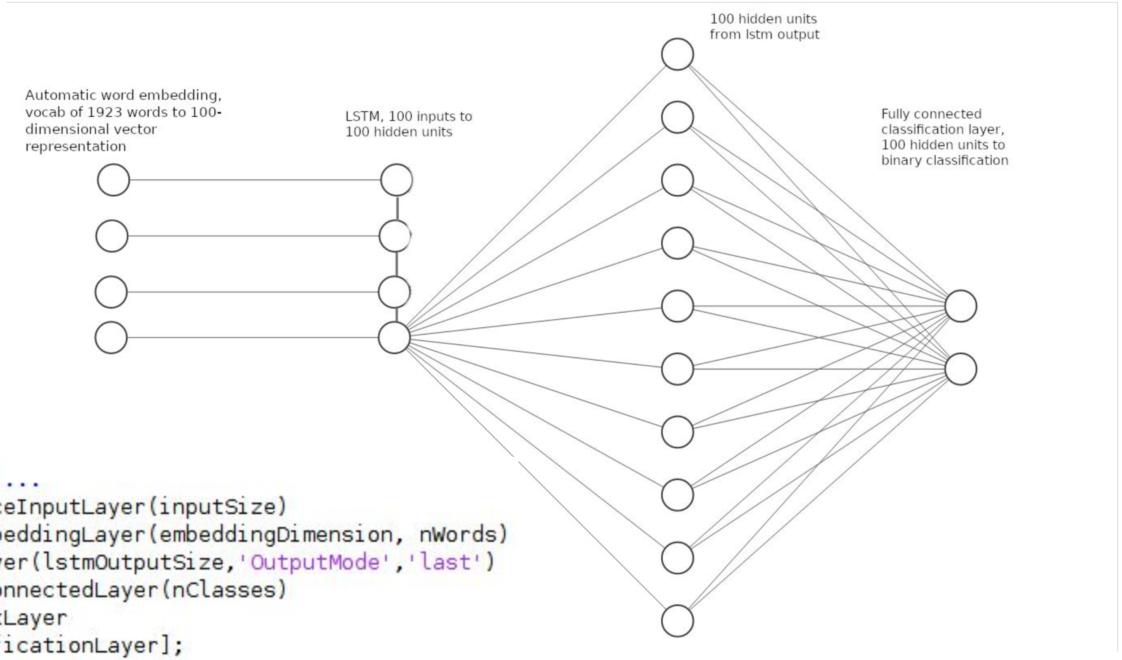
Model 1: Naive Bayes

- Process and Clean the data to reduce the strain on the model
- Run naive bayes using fitcsvm
  - Utilised MATLAB inbuilt classes
- Test on test data

Model 2: SVM

- Process and Clean the data to reduce the strain on the model
- Words were converted into word vectors so that they can be used for SVM
- Data was visualised with word clouds to see if they made sense in the context of the tweet.
- Individual words in tweets were calculated for their polarity to find the total polarity for the tweet.
- Accuracy was tested using sign class with the testing data.

LSTM architecture and hyperparameters




```
layers = [ ...
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension, nWords)
    lstmLayer(lstmOutputSize,'OutputMode','last')
    fullyConnectedLayer(nClasses)
    softmaxLayer
    classificationLayer];
```

- Preprocessing: Normalised data input
- Epochs: 2
- Training rate: 0.01
- Mini Batch size of 20 statements
- All hyperparameters found empirically on validation set

Performance Measurement

Human raters typically only agree about 79% of the time[1]

Therefore measurements of accuracy in the order of 75% could be said to be doing almost as well as a human

However ML methods in regards to sentiment analysis often show naivety in regards to classification of sarcasm and similar literary devices.

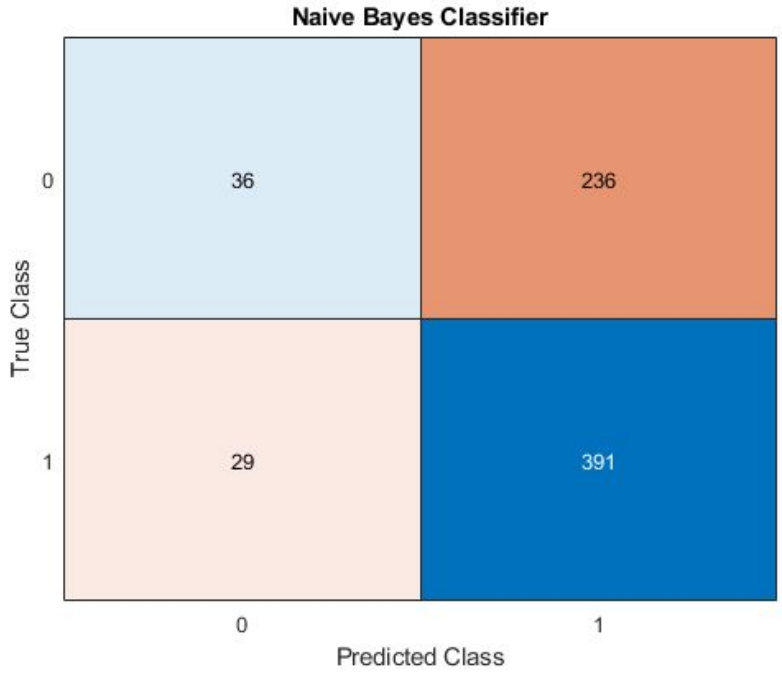
- This can have a significant impact depending on the context that sentiment analysis is used in

[1] https://mashable.com/2010/04/19/sentiment-analysis/

Evaluation of Models

Model 1: Naive Bayes

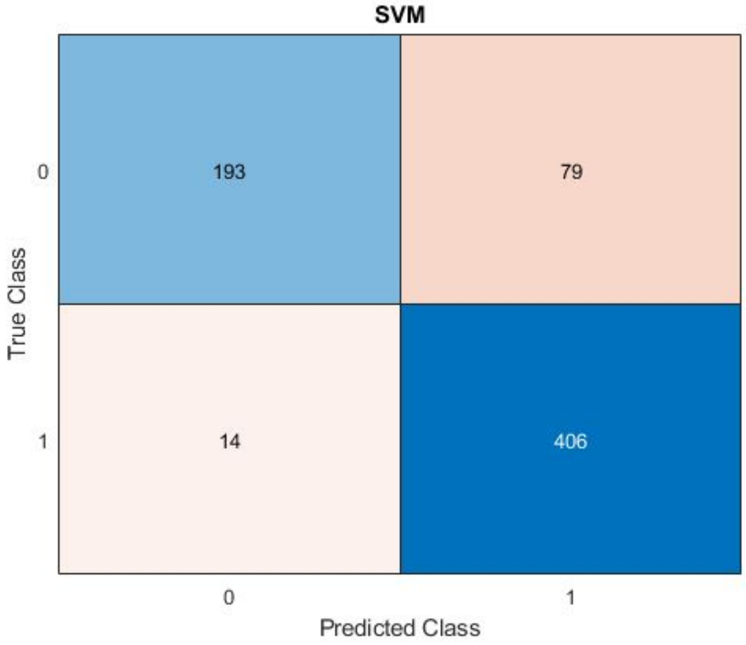
- Achieved 61.71 % accuracy on test data
- Given that in this test set, 0.9061% of the data was classified as positive, this may not be a very good method.
- This only performs 1% better than guessing that all tweet have positive sentiment
- Our implementation is much less effective than the literature



Evaluation of Models

Model 2: SVM

- Achieved 86.56% accuracy
- Good result, in line with literature



Evaluation of Models

Model 3: LSTM

- Achieved 99.57% accuracy on the test data - possible limited vocabulary a problem here, so may have slightly limited utility outside of its vocabulary/dataset

```
>> TestLstm("I hate this")

ans =

  1×2 single row vector

    0.9689    0.0311

>> TestLstm("I love this")

ans =

  1×2 single row vector

    0.0116    0.9884
```

```
>> TestLstm('I hate you')

ans =

  1×2 single row vector

    0.9689    0.0311
```

```
>> TestLstm('I dont hate you')

ans =

  1×2 single row vector

    0.9689    0.0311
```

Improvements and Limitations

- Neutral sentiments
  - Include data that contains neither a positive nor negative sentiment
- CNN-LSTM combination models are popular to find patterns in text
- Possible instances of overfitting to our dataset in algorithms
  - Increase variation in training and test data - more data
  - Change model variables to reduce overfitting
  - More complex dataset
  - Dropout layers