Humor Detection in Yelp Reviews

Information Retrieval and Extraction

Major Project

Team 3
Srishti Aggarwal (201325049)
Vanaja Penumatsa (201505505)
Vishnu Kumar (201301040)

PROBLEM STATEMENT

To utilize the state-of-the-art in deep learning and show that deep learning can capture the higher order structure of humor in Yelp reviews, by comparing with state-of-the-art shallow learning.

Reference: Luke de Oliveira and Alfredo Ĺainez Rodrigo, "Humor detection in Yelp reviews"

APPLICATIONS OF HUMOUR DETECTION

Humor is a necessary part of all verbal communication. With the advancement of computational technologies, increasingly more emphasis is being placed on systems that can handle human natural language effectively. Thus, without humor detection and generation, no natural language system can be considered successful. It is necessary for a full computational understanding of natural language documents and for enabling intelligent conversational agents to handle humor.

Domain specific application: Yelp is a platform which aims to provide accurate, crowd-sourced opinion about a place of business. Humour detection in the reviews present there, can be used to determine the importance, or the genuineness of a particular review.

CHALLENGES

- Any results derived from work on Yelp data may end up being highly domain specific. But, the lack of any openly available dataset for humour leaves no option.
- The human classification of something as "humour" can be very subjective.

DATASET

We are using the Yelp Dataset Challenge dataset, which consists of about 1.6 million reviews by 366,000 users for 61,000 businesses. Each yelp review has user given votes for three categories - "funny", "cool" and "useful". We are using this community provided data to get our dataset. Any review with more than 3 "funny votes" is taken as funny, and the others as not-funny. We extracted a balanced set of 1,00,000 reviews, containing equal number of samples for both classes. This data is be used for training and testing purposes for this project.

MODULES

Two types of learning algorithms have been applied using different word representations. These are explained below.

1. Shallow learning methods:

SVM (Support Vector Machines)

SVM is a supervised learning algorithm which aims to detect and exploit complex patterns in data and use that information for tasks like classification.

SVM Kernels:

A kernel is a similarity function. It is a function that you, as the domain expert, provide to a machine learning algorithm. It takes two inputs and spits out how similar they are. Using kernels allows us to apply SVM to any dataset (even if it is non linearly separable), without having to extract specific similarity features.

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Mathematics:

Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, in two classes, and a vector $y \in \{1, -1\}^n$, SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$
$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, ..., n$$

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$
$$\text{subject to } y^T \alpha = 0$$
$$0 \leq \alpha_i \leq C, i = 1, ..., n$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ Where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is:

$$\operatorname{sgn}(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho)$$

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.

For this project, we have used the scikit-learn python implementation, and performed experiments using both linear and rbf kernel.

Data representation: bag-of-words model. Using tf-idf scores.

Parameters have been set as follows:

- C = 1 (The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors.)
- Gamma = auto (the gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The gamma parameters can be seen as the

inverse of the radius of influence of samples selected by the model as support vectors.)

Experiment 1: SVM, Linear Kernel

a. Single fold:
   i. 1/5th of the dataset taken as testing data, rest as training data
   ii. Accuracy score = 0.83325

b. 5-fold cross-validation:

The following accuracy scores were achieved.

- 1st iteration: 0.8328
- 2nd iteration: 0.83435
- 3rd iteration: 0.8299
- 4th iteration: 0.835
- 5th iteration: 0.83365
- Mean accuracy score: 0.83314

Experiment 2: SVM, Rbf kernel

a. Single fold:
   i. 1/5th of the dataset taken as testing data, rest as training data
   ii. Accuracy score =0.4998

b. 5-fold cross-validation:
   i. The following accuracy scores were achieved.
      - 1st iteration: 0.4996
      - 2nd iteration: 0.49285
      - 3rd iteration: 0.49315
      - 4th iteration: 0.49995
      - 5th iteration: 0.49985
      - Mean accuracy score: 0.49708

Analysis: Linear kernel gives a reasonably good accuracy. Rbf kernel on the other hand gives an accuracy of less than 50% - which is in fact a poorer performance than a random classifier considering that we are only trying to do binary

classification. This may be because the parameters haven't been tuned, and multiple reruns of the rbf classifier with various parameters adjustments need to be done to come up with optimum values of c and gamma.

Experiment 3: Using a secondary dataset

Secondary dataset: a dataset consisting of about 16,000 one-liner jokes (funny), and 2000 quotes (not funny) was taken (collected from LTRC). This data was used for testing the classifiers trained using the Yelp review dataset.

a. Linear kernel:
    i. Accuracy: 0.58
b. Rbf kernel:
    i. Accuracy: 0.69

Analysis: Here in this experiment, linear kernel performs poorly, while rbf performs comparatively well. These results combined with the results of experiments 1, 2 and 3 do not give an conclusive results as such. We can say that linear SVM performs well on the yelp review dataset, but its general utility (out of domain/context) cannot be determined. This may be because of the quality of the datasets, the subjective domain of "humour" classification, or just that this learning technique fails to capture the essence of what the features of a humorous text are. A lot more experiments, using different dataset specifications and parameters need to be done if we want to come up with come more conclusive results.

2. Deep Learning methods:

Data representation: Word vectors

Feed Forward networks

A feedforward neural network is a biologically inspired classification algorithm. It consist of a (possibly large) number of simple neuron-like processing *units*, organized in *layers*. Every unit in a layer is connected with all the units in the previous layer. These connections are not all equal: each connection may have a different strength or *weight*. The weights on these connections encode the knowledge of a network. Often the units in a neural network are also called 'nodes'.

Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, that is when it acts as a classifier, there is no feedback between layers. This is why they are called *feedforward* neural networks.

Feedforward neural networks uses a supervised learning algorithm: besides the input pattern, the neural net also needs to know to what category the pattern belongs. Learning proceeds as follows: a pattern is presented at the inputs. The pattern will be transformed in its passage through the layers of the network until it reaches the output layer. The units in the output layer all belong to a different category. The outputs of the network as they are now are compared with the outputs as they ideally would have been if this pattern were correctly classified: in the latter case the unit with the correct category would have had the largest output value and the output values of the other output units would have been very small. On the basis of this comparison all the connection weights are modified a little bit to guarantee that, the next time this same pattern is presented at the inputs, the value of the output unit that corresponds with the correct category is a little bit higher than it is now and that, at the same time, the output values of all the other incorrect outputs are a little bit lower than they are now. (The differences between the actual outputs and the idealized outputs are propagated back from the top layer to lower layers to be used at these layers to modify connection weights. This is why the term *backpropagation network* is also often used to describe this type of neural network.)

We have built a feed forward network on the top of word vectors generated using gensim library of python. Pybrain library was used to build the feed forward network.

The network was constructed to have 100 input nodes, 20 hidden layers and one output node. Here we have generated a dataset that supports 100 dimensional inputs and one dimensional targets since each input vector is 100 dimensional target is funny or non funny represented in 1 dimension (1 or 0) . Finally the network was trained till convergence

Experiment:

- 1/4th part of the dataset is taken as test data, rest as training data.
- Training Error score : 0.25
- Full score detail: http://pastebin.com/PsB9MUp4
- Detailed Scores:
    - Training accuracy: 0.75
    - Validation accuracy:
        - 1st Epoch : 0.74978

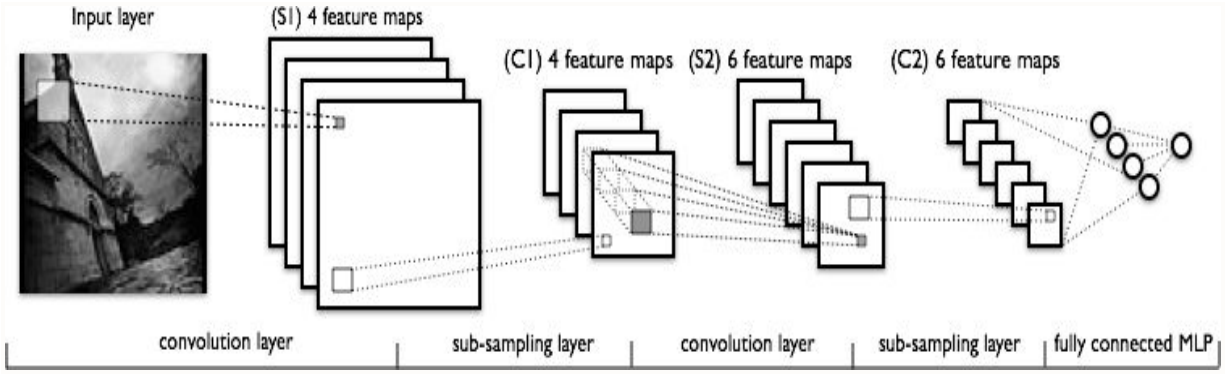
- 2nd Epoch : 0.7500733333333334
- 3rd Epoch : 0.74978

Convolutional Neural Network (CNN or ConvNet)

A convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the cat's visual cortex.

The visual cortex contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a *receptive field*. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.

A CNN is comprised of convolutional layer(s) --> MaxPooling layer(s) --> DropOut layer(s) --> fully connected layer(s).

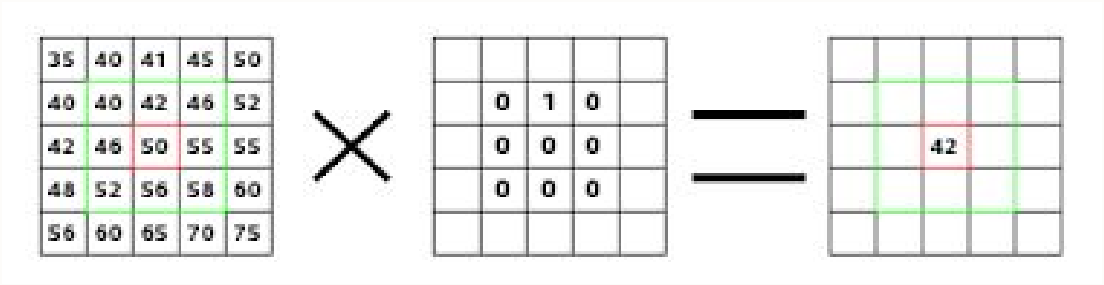


A. Convolution Layer

Natural images(here word vectors) have the property of being "'stationary"', meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image. Specifically, we can take the learned 8x8 features and "'convolve"' them with the larger image, thus obtaining a different feature activation value at each location in the image.

Convolution is performed as follows



Pass the $kernal_{(mxn)}$ over the $image_{(MxN)}$ and calculate $K^T I_s$. The resulting image size is $Res_{(M-m+1xN-n+1).}$

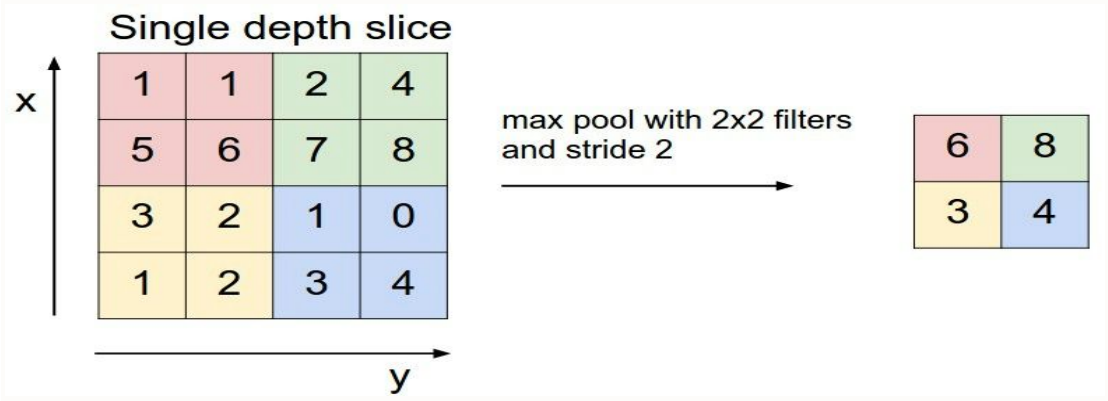
B. MaxPooling

Another important concept of CNNs is *max-pooling,* which is a form of non-linear down-sampling. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value.

Max-pooling is useful in vision for two reasons:

1. By eliminating non-maximal values, it reduces computation for upper layers.
2. It provides a form of translation invariance. Imagine cascading a max-pooling layer with a convolutional layer. There are 8 directions in which one can translate the input image by a single pixel. If max-pooling is done over a 2x2 region, 3 out of these 8 possible configurations will produce exactly the same output at the convolutional layer. For max-pooling over a 3x3 window, this jumps to 5/8.

Since it provides additional robustness to position, max-pooling is a "smart" way of reducing the dimensionality of intermediate representations.



Fig

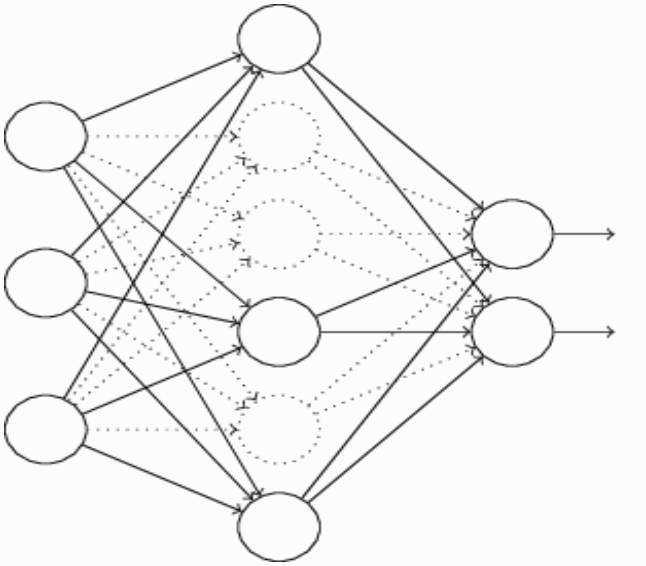
On a input image$_{(MxN)}$ and a stride S the resulting image dimensions are Res$_{(M-SxN-S).}$

C. DropOut Layer

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time.

Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much.

During training, dropout samples from an exponential number of different thinned networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods.



A simple way to prevent Neural Networks from overfitting. Neurons of i^{th} layer can be connected to some neurons in the $i\text{-}1^{th}$ layer.

D. Fully connected layer

Neurons of i^{th} layer are connected to all the neurons in the $i\text{-}1^{th}$ layer. (Refer figure)

Experiment 1: CNN-static using Word2Vec

CNN-static using Word2Vec - 80.15%

Experiment 2: CNN-non-static using Word2Vec

CNN-non-static using Word2Vec - 80.73%

Experiment 3: CNN non-static using rand:

CNN-non-static using rand - 78.31%

Experiment 4: CNN-static using random

Accuracy score: 75.4%

Analysis: We can observe that when a CNN is fed with word vectors which is knowledge or relations among the words of a sentence it gave better results than the CNN with random initial values.

CONCLUSIONS:

We began by constructing both deep and shallow models over bag of words and mean word vector representations – we note that shallow methods perform better over bag of words based features, while deep models are more performant on the mean word vector representation of phrases. We hypothesize that this is due to the fact that the expressive deep networks can use the semantic information contained in the word vectors.

After further analysis we found out that neural networks takes input in continuous format and we have word vectors which does the same unlike Bag of words which are discrete. So, this can be architectural behaviour of each algorithms.

Despite all this, the results of our experiments were inconclusive as to whether deep learning methods can actually capture the structure of a humorous text and thus, assist in the task of humour detection where required.

Links:

Project Webpage: http://srishti-1795.github.io/Humour-Detection/

Github repository: https://github.com/srishti-1795/Humour-Detection

Youtube Video: https://youtu.be/Y2VHCbGppMs

Presentation: http://www.slideshare.net/SrishtiAggarwal5/humour-detection

References:

https://cs224d.stanford.edu/reports/OliveiraLuke.pdf

http://emnlp2014.org/papers/pdf/EMNLP2014181.pdf