DNB Data Science Nanodegree

Capstone Project

Thomas Schøyen
December 8th, 2018

Definition

Project Overview

This project is based on the Kaggle competition "Quora Insincere Questions Classification" [1], where the goal is to identify and flag insincere questions. Quora[1] is a platform for questions and answers, and the usefulness of the site relies on quality content. To classify questions using machine learning techniques, one must use a form of Natural Language Processing (NLP). NLP is a subfield of computer science and has been an area of research for many years. It can be described as the study of how computers can be used to understand and manipulate natural language [2]. This includes extracting topics and context from text, speech recognition, sentiment analysis, translation between languages and much more. Perhaps the most well-known (or at least common) example is the speech recognition software available on almost any smartphone these days. Being able to automatically (by machine) interpret and extract context and intent from text is arguably increasingly important, given the amount of content being produced on various platforms on the web today, and the impact it may have in the future (e.g. misinformation and "fake news" [3]).

There have been considerable contributions to the NLP field in recent years, especially by big technology companies. A notable example is "word embeddings" - dense vector representations of words suitable for large data sets, such as the *skip-gram with negative sampling model* suggested by [4] and implemented in the word2vec software[2]. Several other embeddings have been developed since, some of which were supplied together with the data for the Kaggle competition in question. The project leverages these models in combination as well as a selection of machine learning classification algorithms to solve the problem.

This capstone project is motivated by enormous amount of unstructured text available to businesses, both internally and externally (such as web and news sources), and the insights that can be extracted from them with the use of NLP and cutting-edge algorithms and technology. This is also my personal motivation – as a business intelligence architect and consultant for over ten years, I have personally observed the vast amount of unstructured data left untouched and un-analyzed. I also

[1] https://www.quora.com

[2] https://code.google.com/archive/p/word2vec/

wanted to take this opportunity to go deeper into the neural network/deep learning field than I was able to do through the Data Science Nanodegree content.

Problem Statement

Problem Definition

The challenge of this project is to accurately predict if a given question posted on the Quora platform is sincere or not. An insincere question is in this case defined as a question intended to make a statement rather than look for helpful answers [1]. Generally, handling ill-intended, fake or simply bad content has become a major challenge for any site today, and it is especially true for businesses with a platform business model, connecting producers and consumers of some value unit [5]. Quora is an example of a platform where the value creation relies on quality content being produced by question and answers provided by the users of the platform. Quora currently use both machine learning and manual reviews to address this challenge [1], in addition to the inherent curation that comes from the social aspect of "upvoting" (liking) a question or answer.

From a technical viewpoint, the Kaggle competition in question (and hence this project) is a supervised learning problem, more specifically a binary classification problem. Given a list of free-text questions asked on the Quora platform, the challenge is to predict whether a question is sincere or not. A sincere question will be given the label 0 and an insincere question will be given the label 1. The goal is to maximize the F1-score (harmonic mean) of classification precision and recall (see next section for details). A labeled training data set of questions is provided as part of the Kaggle competition.

Project Design

The project solution will require a combination of NLP-tasks and machine learning tasks. Since different machine learning algorithms require different pre-processing for good results, multiple NLP pipelines must be built for different purposes. The project work will follow the general process of data science projects (or any data/analysis project) described in the Cross-Industry Process for Data Mining (CRISP-DM). The phases described in this standard (c.f. [6]) are:

1. Business Understanding
2. Data Understanding
3. Prepare Data
4. Data modeling
5. Evaluate the Results

(6. Deploy)

Given the specificity of the project and the nature of the datasets, most of the time will be spent on preparing data and creating models. Nevertheless, exploratory data analysis will be performed to understand the data and hopefully create better predictive models. Examples include looking at the distribution of the target variable, exploring languages, lengths of texts, common words etc. Deploying a solution is not applicable except submitting test results to Kaggle for scoring.

While the overall process is the same, in contrast to working with tabular data, NLP projects have some differences related to the steps that must be performed. As NLP is an entire field of study with many scientific articles, books and blog-posts published on the subject, a thorough discussion of these is not in scope for this project. There are several very common tasks however, that is usually performed when working with text for a machine learning / classification problem. Referring to the CRISP-DM phases mentioned above, phase 3 and 4 especially involves some common NLP tasks:

Text pre-processing, such as:

- **Cleaning text**, e.g. removing punctuation and fixing common misspellings etc. Simple match-replace or more advanced regular expression techniques are commonly used.
- **Tokenization** – splitting text into sentences and words.
- **Text lemmatization** – replacing words with their basic form or lemma[3], e.g. cars->car.
- **Text stemming**[4] – reducing words to their stem, i.e. the part that remains after removing affixes, e.g. removing->remov.
- **Removing stop-words** – these are words that are common in any text and have little or no significance to the meaning of the text. Examples are "the", "a", "I" etc.

Mapping text to a representation suitable for machine learning algorithms, such as:

- **Bag-of-words representation** - occurrences of words within text/document, see [7] for a good explanation.
- **Term Frequency-Inverse Document Frequency (TF-IDF)** - considers how important a word is to a document; importance increases proportionally to the number of times a word appears in the text but is scaled by how frequent the word is in the corpus[5].
- **Word embeddings** - This is method for representing data with many classes more efficiently (such as words in a document). Details will follow in subsequent sections.

[3] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[4] https://en.wikipedia.org/wiki/Stemming

[5] https://en.wikipedia.org/wiki/Text_corpus

In summary, the following figure depicts the project outline:

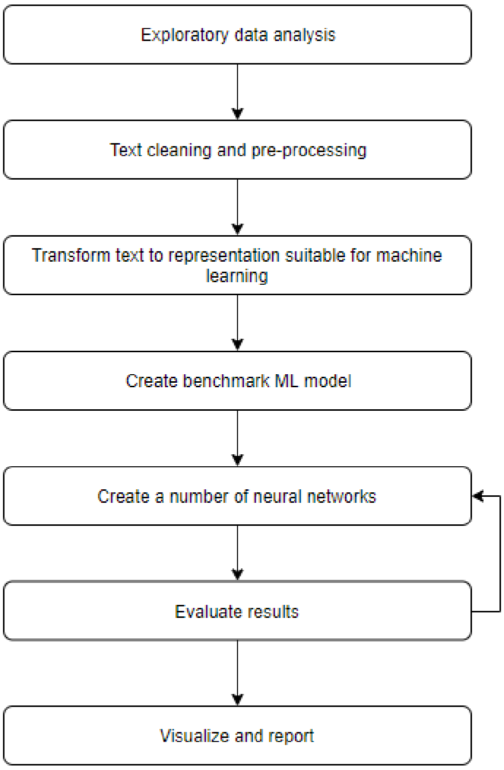

Exploratory data analysis
Text cleaning and pre-processing
Transform text to representation suitable for machine learning
Create benchmark ML model
Create a number of neural networks
Evaluate results
Visualize and report

Figure 1: Project outline

Solution Overview

The solution to this project will be programmed in a set of Jupyter notebooks, using Python 3[6]. The notebook-way of working with machine learning allows for rich content to be mixed with code so that notes and explanations can be described together with the solution. This will ease the evaluation of the project and enable the solution to be easily reproduced. Notebooks is a good way to do prototyping and sharing but is not suitable for a production scenario. Since the focus of this project is on the machine learning / data science part, the result will be kept in notebooks. The goal is nevertheless to strive for modularity and clean, efficient code.

The solution will use two well-known libraries for machine learning in python, as well as supporting libraries for data wrangling, feature extraction and exploratory analysis. The most important libraries used:

Library	Description
scikit-learn[7]	Open source library for machine learning, built on Numpy, SciPy and Matplotlib (see below). Contains tools for classification, regression, clustering, dimensionality reduction, model selection and pre-processing. This project will use scikit-learn for classification (logistic regression and random forest), model

[6] https://www.python.org/download/releases/3.0/

[7] https://scikit-learn.org/stable/

	selection (grid search and metrics), feature extraction and pre-processing (e.g. splitting training data in training and validation).
Keras[8]	Open source high-level (i.e. sacrificing some control for simplicity) neural networks API, written in Python and capable of running on top of several underlying neural network frameworks. Its high-level API makes it very suitable to prototype different neural network architectures. This project will use Keras with a Tensorflow[9] backend for developing neural networks and NLP tasks.
NLTK[10]	Natural Language Toolkit. A suite of libraries for classification, tokenization, stemming, tagging, parsing and semantic reasoning. This project will use NLTK for NLP tasks such as tokenization, word-to-vector conversion, stop word-removal and lemmatization.
Spacy	Library for NLP in Python, with much overlap with NLTK. This project will use NLTK for NLP tasks such as tokenization, word-to-vector conversion, stop word-removal and lemmatization.
Pandas[11]	Pandas is a Python data analysis library, providing fast, flexible data structures for working with data. Pandas will be used to hold data structures as well as reading/writing/parsing files and cleaning data.
Numpy[12]	Python library for scientific programming, adding support for large multi-dimensional arrays and matrices and many high-level mathematical functions. Numpy is required for several of the other libraries used and will be used by itself for vector/matrix operations.
Matplotlib[13]	Python plotting library for programmatic visualization. This will be used for all visualizations.

Table 1: Python libraries

The solution to this project will be programmed in a set of Jupyter notebooks, using Python 3[14]. The notebook-way of working with machine learning allows for rich content to be mixed with code so that notes and explanations can be described together with the solution. This will ease the distribution and evaluation of the project and enable the solution to be easily reproduced. Notebooks is a good way to do prototyping and sharing but is not suitable for a production scenario. Since the focus of this project is on the machine learning / data science part, the result will be kept in a Notebook., but code used by many notebooks will be modularized and kept in utility files.

Different machine learning algorithms will be explored, including logistic regression as a benchmark. Most likely, a form of recurrent neural network (RNN) architecture capable of learning from

[8] https://keras.io/

[9] https://www.tensorflow.org/

[10] https://www.nltk.org/

[11] https://pandas.pydata.org/

[12] http://www.numpy.org/

[13] https://matplotlib.org/

[14] https://www.python.org/download/releases/3.0/

sequences of text will be part of the final solution, since this family of neural networks have been shown to yield state-of-the-art results for similar problems [8]. Other architectures will also be explored, such as convolutional neural networks (CNNs, commonly used for image classification) and standard fully connected feed forward networks.

The Kaggle competition supplies a set of pre-trained word embeddings for use in algorithm training. Word embedding is a neural network approach in which words are "embedded" into a lower dimensional space [9]. According to [9], vectors that are close to each other are shown to be semantically related. E.g. it captures relationships such as "queen is like woman" and "king is like man", and "man/woman is like person". In practice, pre-trained word embeddings trained on a large corpus are often used, and words are simply looked in a word-to-vector matrix before feeding the vector-representation through a neural network. Using pretrained weights with RNNs have been shown to generalize well [10], and word embeddings with pretrained weights will be a part of the final solution. The performance of the models will be evaluated and compared with a benchmark model.

Metrics

The measure of performance in this project is given by the Kaggle competition which states that "submissions are evaluated on F1 Score between the predicted and the observed targets" [1]. The F_1 score is the harmonic mean[15] of precision and recall, and the formula is given by:

$$F_1 = 2\frac{precision * recall}{precision + recall}$$

Precision captures the positive predictive value and is calculated by dividing true positives (TP) by the sum of true and false positives (FP). Recall captures how well the model can identify actual positives and is calculated by dividing true positives by the sum of true positives and false negatives (FN) (i.e. all actual positives).

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

There is often a tradeoff between precision and recall – it is often observed that an increase in recall is followed by a decrease in precision (i.e. by increasing its ability to identify actual positives, a model will also predict more false positives). F_1 score gives a single number considering both precision and recall. In addition to the F_1 metric required by the Kaggle competition, I will use precision-recall curves to summarize the precision-recall tradeoff at different probability thresholds as well as

[15] https://en.wikipedia.org/wiki/Harmonic_mean#Harmonic_mean_of_two_numbers

confusion matrices to visually inspect absolute numbers of true/false positives and negatives. The Quora questions dataset is imbalanced (see next section), and both F_1 score and precision-recall curves are appropriate to measure the accuracy of imbalanced datasets. Alternatively, if recall was of very high importance, a receiver operating characteristic curve (ROC curve) and the area under the ROC curve (AUC) could have been used to show the relationship between true positive rate and false positive rate and subsequently used to decide on a level of false positives to accept.

Analysis

Data Exploration and Exploratory Visualization

Dataset

The dataset for this project is given by Quora through the Kaggle platform. At the time of writing, the capstone problem is an ongoing Kaggle competition, with given rules and restrictions. The dataset consists of two files, downloadable from the competition's web site[16]: train.csv and test.csv, containing the train and test set respectively. The training set contains 1,306,122 examples while the test set contains 56,370 examples (~4%). The raw data files (no transformations applied) contains the following columns:

Column	Description	Datatype
qid	Question ID – unique identifier.	String
question_text	Quora question text.	String
target	Target variable – a question labeled "insincere" has a value of 1, otherwise 0. This column is only available in the test.csv file.	Integer

Table 2: Project datasets meta-data

The test file only contains the first two columns – the target (1/0 – insincere or not) is not given, and the file must be submitted to Kaggle for scoring. Given that this is a text classification problem, the actual features used for machine learning will be extracted from the question text. No other data is allowed in the contest. The target categories in the training set are quite imbalanced – the number of questions labeled sincere is much higher than the ones labeled insincere, as can be seen in the figure below.

[16] https://www.kaggle.com/c/quora-insincere-questions-classification/data

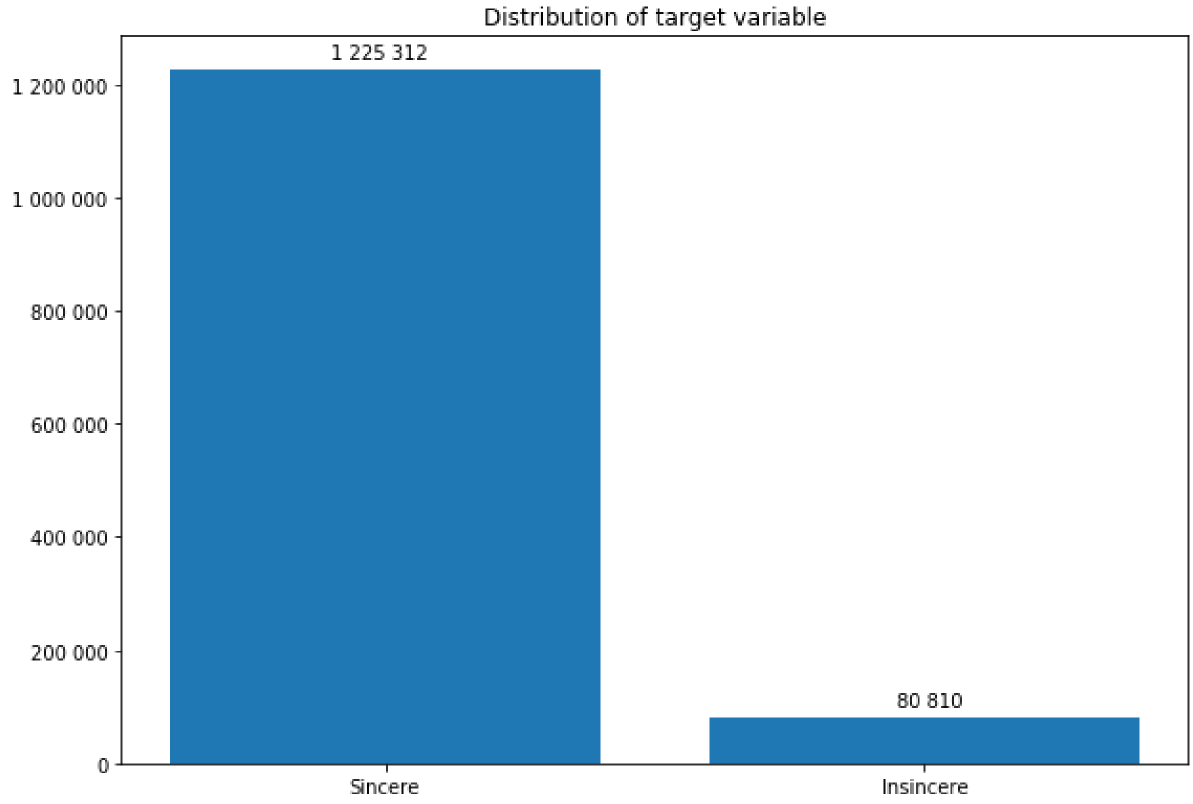


Figure 2: Distribution of target variable

Around 94 percent of the training data is labeled sincere. This must be considered in the classification implementation.

Table 2 below shows basic statistics about two derived features (question length and number of words in question) before pre-processing (removing stop words[17] etc.):

Feature	Dataset	Mean	Median	Minimum	Maximum
Number of words in question	train.csv	12.8	11	1	134
	test.csv	12.8	11	2	65
Question length (number of characters)	train.csv	70.7	60	1	1017
	test.csv	70.4	60	11	294

Table 3: Data statistics

Note that the statistics is before pre-processing, and "words" are simply the tokens resulting from splitting the question_text by white-space. I.e. it will include for example contradictions and URLs. From the statistics, it seems like the training and test set properties are quite similar with both mean and medians almost equal. However, the minimum and maximum values of the training set is more extreme, which suggests some outliers. Looking at the distribution of word counts (number of questions by number of words in question), the test and train datasets look similar:

[17] https://en.wikipedia.org/wiki/Stop_words

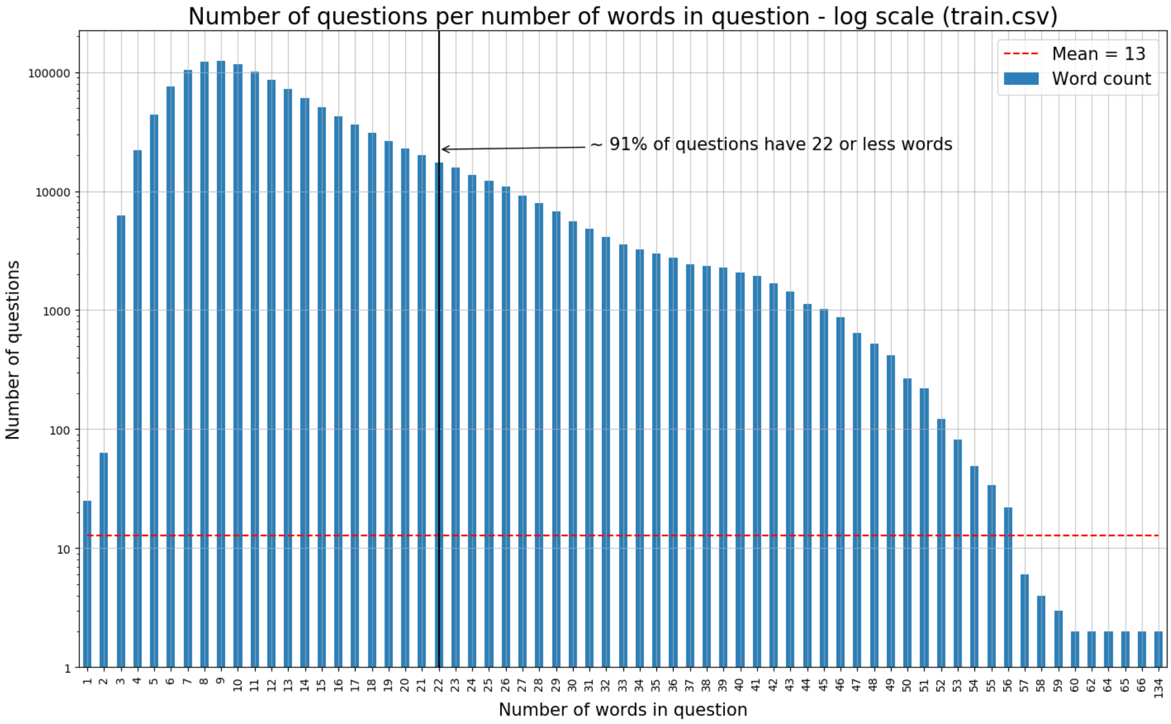


Figure 3: Distribution of word-counts -train

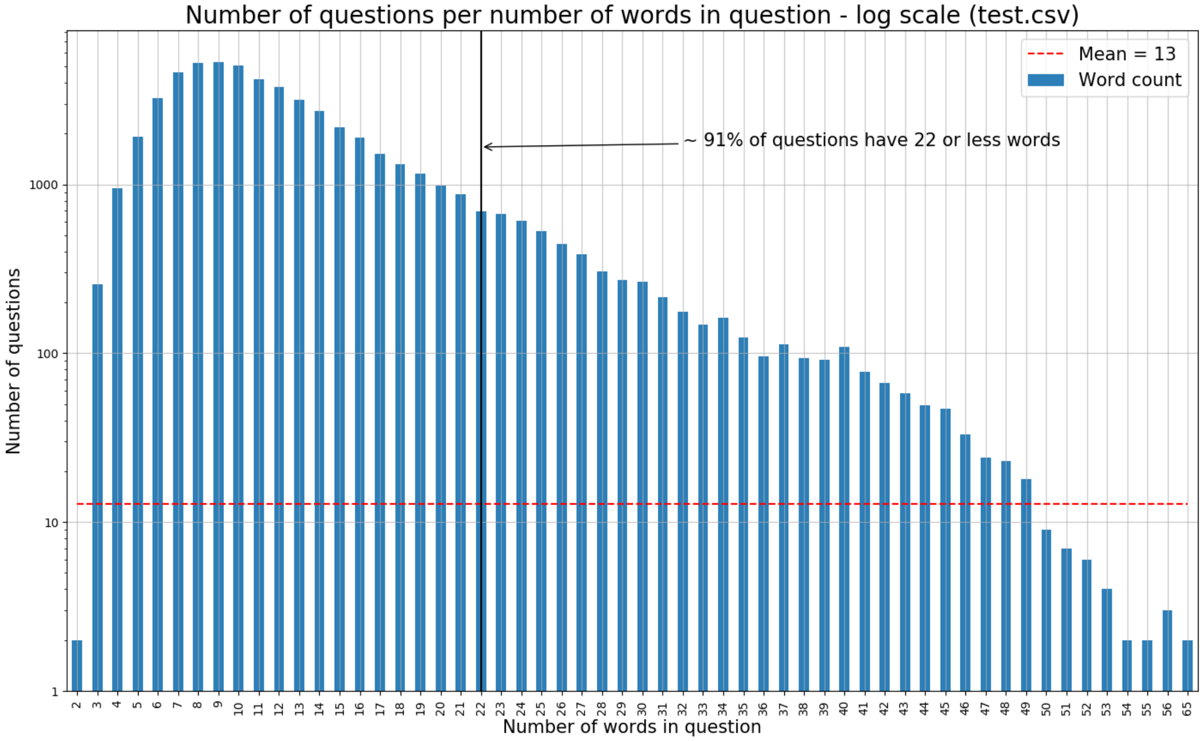


Figure 4: Distribution of word-counts -test

Note that zero-values have been omitted from the plots. For both train and test, over 90 percent of questions have 22 words or less. 99.9 percent of questions in the train set have 50 or fewer words; the figure below shows a normalized histogram of number of words for sincere and insincere questions in the training data. (The area under each distribution sums to 1, ~ 0.023% of questions have been added to the last bin in the histogram below because the max range is set to 50).

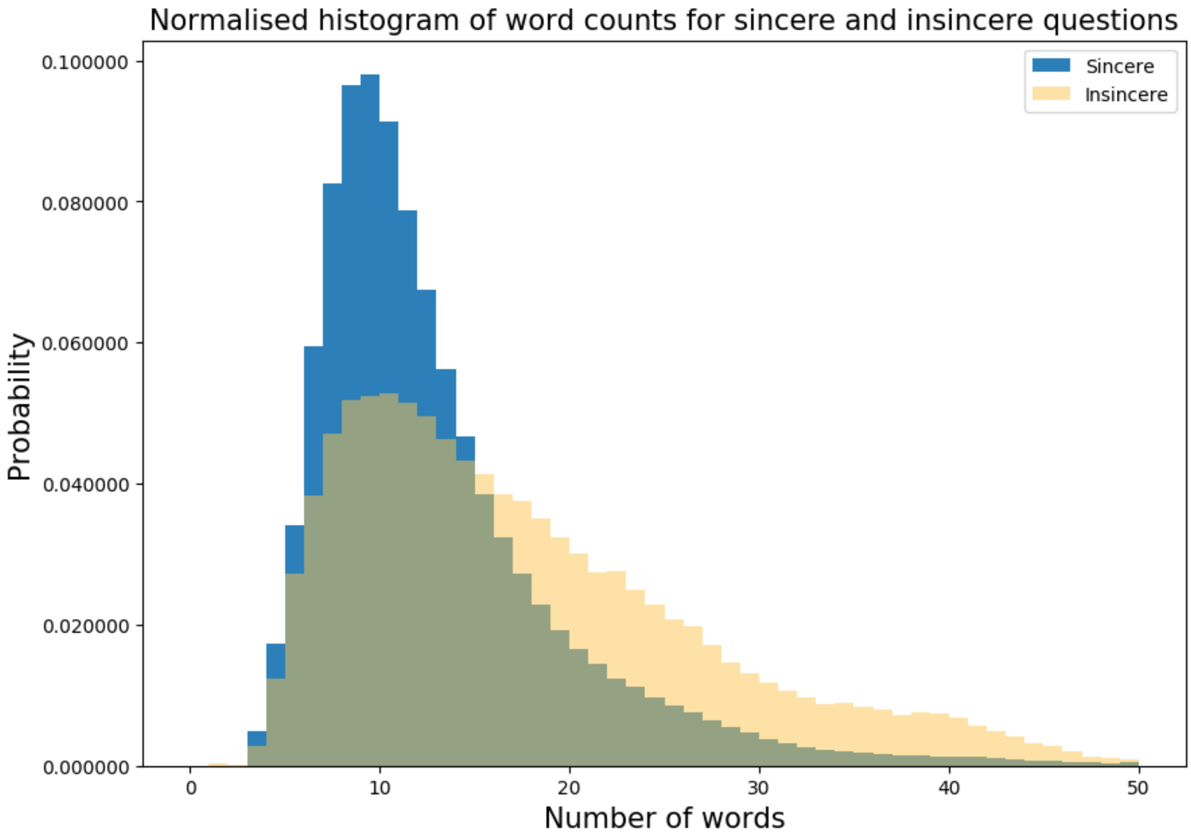


Figure 5: Word-count distribution for sincere and insincere questions

The distributions are quite different, the insincere questions seem to have a more even distribution of word counts, while sincere questions are more concentrated around the mean. This is also clear from the statistics shown in the table below.

Statistic	**Sincere**	**Insincere**
Mean	12.51	17.28
Standard deviation	6.75	9.57

Table 4: Word-count statistics for sincere and insincere questions

The same pattern can be seen by plotting the distribution of character-counts (which seem logical – more words means more characters), but an additional interesting pattern emerges – there seem to be a sharp drop around 150 characters. The reason being that the question length on Quora is limited to 150 characters, while the details can be up to 300 characters [11]. The figure below shows a normalized histogram of the distribution of character-count, cut off at 200 characters (leaves around 1.5% of sincere and 6.4% of sincere questions in the last bin in the histogram, explaining the spike at 200 characters).

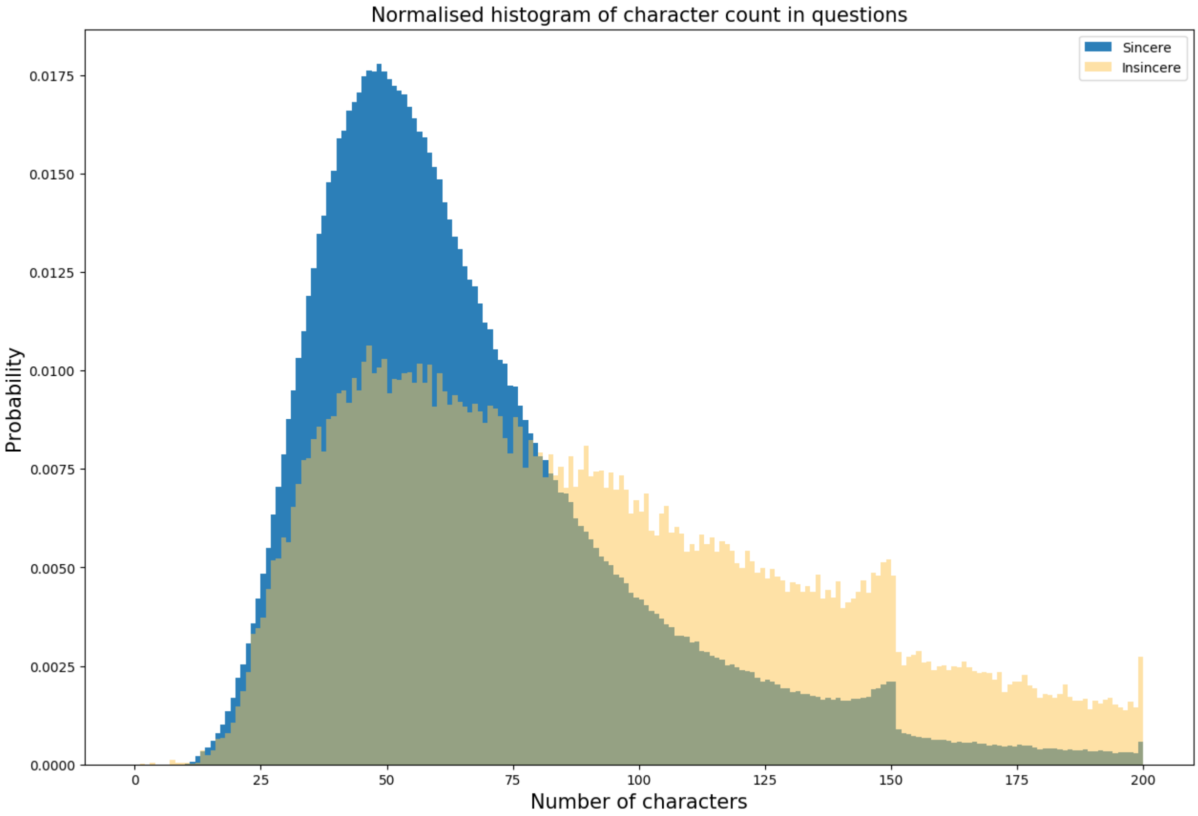


Figure 6: Distribution of character-counts for sincere and insincere questions.

Furthermore, looking at individual words, the statistics is shown in the table below for test and training data.

Statistic	**Train**	**Test**
Mean	4.6	4.6
Standard deviation	2.64	2.66
Max	230	169
Min	1	1
Median	4	4

Is the distribution of long words for sincere questions different than the insincere questions in the training set (e.g. does insincere questions have long non-real words or urls?)? The normalized histogram below suggest that the distribution is similar, but insincere questions seem have a higher relative proportion of long words.

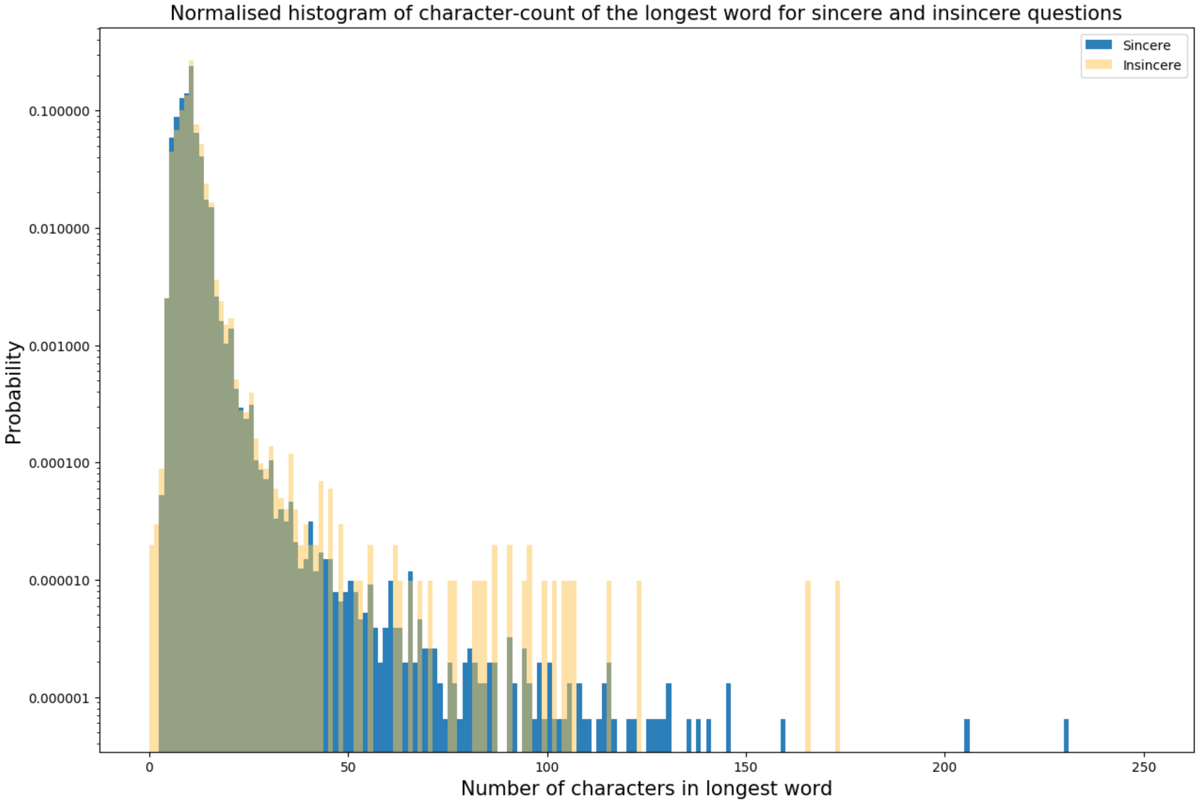


Figure 7: Longest word distribution for sincere and insincere questions

The two longest words in the training set (the two blue outliers in the figure above) were found in the below questions (both labeled sincere):

> "Is 84303978245329874519123841798151507540237416095743623894460139273409764703461 24809844383429043832189438712948136120984309312643809602349843496043429781442 3790438809481927423097438412377777409324590447848903148903219487309487320194 a prime number?"
>
> and
>
> "What does 10110011101000101110010001100001110000100001000001000110000001000001000010011 00101001001010010110101100011101000101111010011001101001010100000000000000000 0000000000011111111111111111111111111111110101010100101 mean in binary?"

Finally, it is interesting to compare the most frequently occurring words and n-grams[18] (n words) in the target groups. Since a few words are very common in the English language, this comparison does not make sense before doing some preprocessing, such as lemmatization and removing stop words. Figure 8 and 9 below shows the most frequently occurring words and trigrams for insincere and sincere questions. While there is clearly a difference between single words, the trigrams provides a bit

[18] https://en.wikipedia.org/wiki/N-gram

more context - e.g. why does "people" have a high occurrence among insincere questions? It seems like the definition of insincere includes questions that is considered offensive, such as "hate black/white people". It is also clear that some phrases are common in both classes, such as "xx year old". Note that the sincere questions have many questions that includes the same long sequence of words, the trigrams "term business traveler", "short term business", "hotel short term" and "good hotel short" all appears in 519 questions, suggesting that the phrase (after lemmatization and stop word removal) is actually similar to "good hotel short term business traveler".

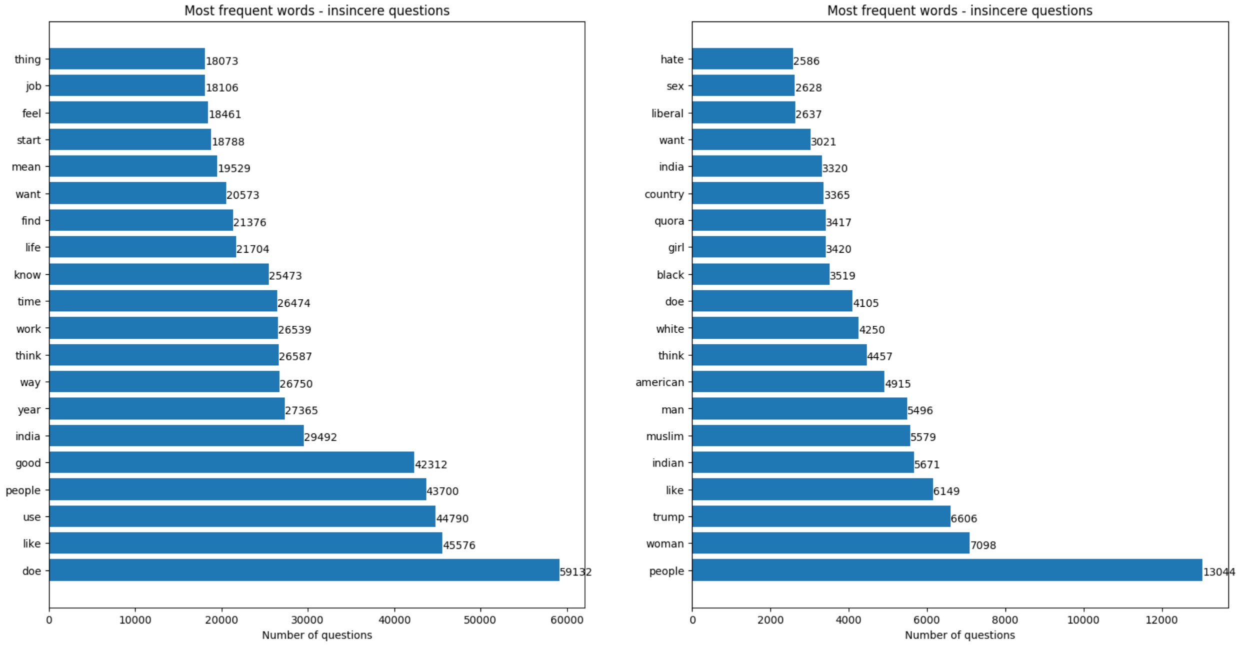


Figure 8: Frequent words in sincere/insincere questions

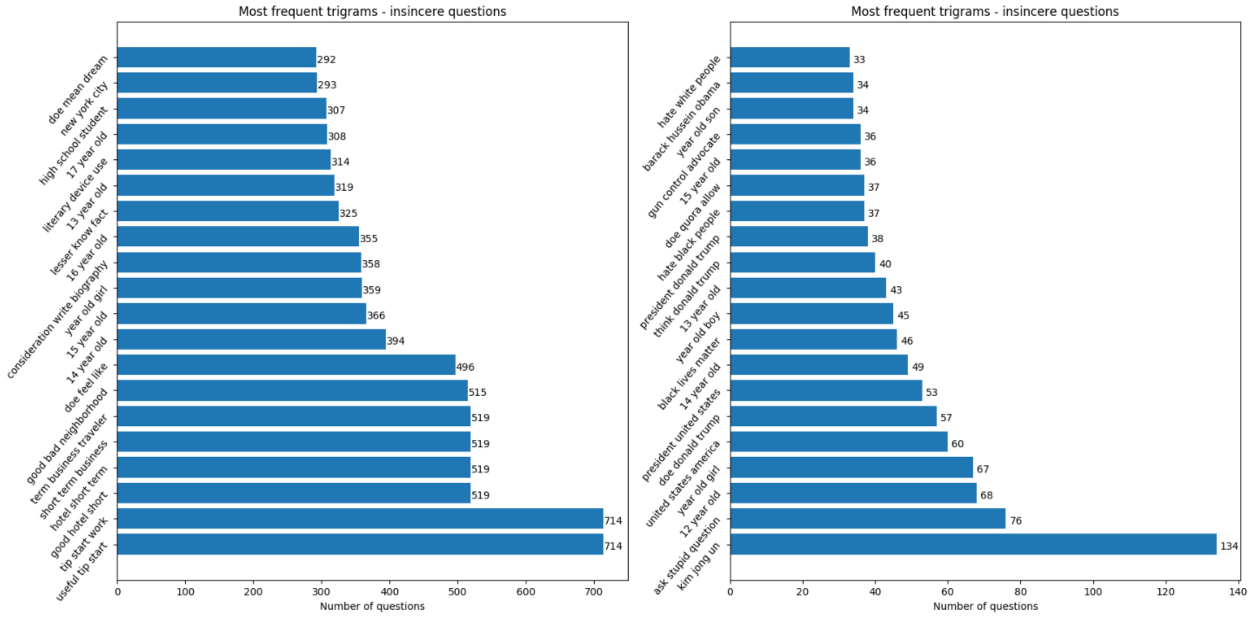


Figure 9:Frequent trigrams in sincere/insincere questions

Additional inputs

Additional inputs in the project are a set of pre-trained word vectors (embeddings), also downloadable from the competition's website. These are:

File	Description
GoogleNews-vectors-negative300[19]	Implementation of the word2vec algorithm from [4]more specifically the skip-gram architecture for computing vector representations of words [4]. This file contains pre-trained vectors based on the Google News dataset
glove.840B.300d[20]	GloVe is an unsupervised learning algorithm for obtaining vector representations for words [12]. This file contains vectors pre-trained on the 840 billion token Common Crawl corpus.
paragram_300_sl999[21]	These are 300-dimensional vectors from [13], pre-trained on the Paraphrase Database [14].
wiki-news-300d-1M[22]	FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. This file contains vectors pre-trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset [15] [4].

The Quora-competition is a so-called Kernels-only competition, which means that actual submissions to Kaggle must be done through a "kernel" on the platform (i.e. a script or Jupyter notebook) [23]. The competition is segmented into two stages 1 and 2; in stage 2, the train.csv and test.csv files will be replaced with new files, which will **not be downloadable from the competition site**. For the purpose of this project however, using the files in Stage 1 will be enough, and it should not make any difference to the solution.

Algorithms and Techniques

The problem which this project attempts to solve is supervised learning problem, more specifically a binary classification problem. As such, any classification algorithm could be a suitable solution. A big part of the challenge, however, is finding a good representation of the data, which in this case is questions in "free form" English posted to the Quora website. Examples of word representations are bag-of-words and word embeddings, see details below. While the same representation is possible to use for any classification algorithm, different algorithms possibly perform better on different

[19] https://code.google.com/archive/p/word2vec/

[20] https://nlp.stanford.edu/projects/glove/

[21] https://cogcomp.org/page/resource_view/106

[22] https://fasttext.cc/docs/en/english-vectors.html

[23] https://www.kaggle.com/c/quora-insincere-questions-classification#Kernels-FAQ

representations. Testing all permutations of representations/algorithms is not feasible; the following algorithms and techniques were chosen for this project:

Text representations

Bag-of-words (BOW): A simple and very common way for extracting features from text. The name comes from the fact that the order of words is not considered in the learning process, the model is only concerned with whether words from a known vocabulary occur in a document or not (and how many times it occurs) – i.e. BOW (w, d) = number of times word w appears in document d. Document in this case is a Quora question. This representation will result in a D × V matrix, where D is number of documents and V is the size of the vocabulary. In our case, the test dataset contains 1,306,122 questions consisting of 13 words on average, and a vocabulary size (after stop word removal and lemmatizing) of more than 3,400,000 (clearly, there are many made-up words, misspellings etc.). I.e. the matrix will be extremely **sparse**. Luckily, there are Python libraries handling the complexity and utilizing data structures suitable for such matrices. For BOW representation, we will use scikit-learn's CountVectorizer[24]. As explained in the exploratory analysis section above, n-grams seem to better distinguish sincere and insincere questions, and thus we will include bigrams in the BOW representation.

TF-IDF: A challenge with "plain" BOW is that the number of times a word appears does not necessarily speak to its importance. TF-IDF is a way to normalize the word frequency by considering how many times the word appears in all documents. I.e. it will consider common words less important. More specifically, TF-IDF transformation will result in rare words occurring often in few examples to have high importance, and rare words or words occurring in many documents to have less importance. There are several variations on TF-IDF, but in the simplest form, term frequency (TF) is simply the number of times (n) a term occurs in a document. The document frequency of a term (df_t) is the fraction of documents that contain the term t. IDF is the logarithmically scaled inverse of this, so that IDF is given by $IDF = \log(\frac{\mathrm{n}}{df_t})$. We will extend the BOW representation resulting from CountVectorizer to TF-IDF (including bigrams) using scikit-learn's TfidfTransformer[25]. Like BOW, TF-IDF result in a D × V matrix. For both BOW and TF-IDF, it is common to limit the vocabulary size, so that V = vocabulary limit.

Word embeddings: A significant drawback of both BOW and TF-IDF, including their n-gram extensions, is that they do not capture distances (or similarities) between individual words. Word embeddings is a set of language modeling and feature learning techniques that basically are functions for mapping words to dense high-dimensional vectors of real numbers. According to [16], the distributional hypothesis states that words in similar contexts have similar meanings, and word embeddings is a way to capture context. Many different types of models have been proposed to learn the function for mapping words to vectors, such as Latent Dirichlet Allocation (LDA) [17]. The word embeddings supplied with the Kaggle competition that is the basis for this project are mentioned above. We will utilize GloVe and word2vec. GloVe is a "... log-bilinear model with a

[24] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[25] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning." [12]. Word2vec is a neural network approach for learning the word to vector function proposed by [4]. The pre-trained representation of word2vec supplied in the Kaggle competition is trained on google-news dataset using the skip-gram approach, where the training objective is to learn word vector representations that are good at predicting neighboring words given a input word [18]. Word embedding exhibit the remarkable property that words with similar meanings will result in similar vectors, so that analogies between words are encoded in the difference vector between them. Mikolov et al.'s work in [4] describes how subtracting the vector representation of "man" from the vector representation of "king" and adding the vector for "woman" results in a vector that is closes to the representation of the word "queen". As such, it can generalize to new/unseen combinations of words and hopefully increase the performance of our models as measured by the defined metric F_1. Since we are utilizing pre-trained embeddings in this project, **the vectors are simply read in from a file, and used as a lookup-table for looking up words in our dataset.**

Algorithms

Logistic regression: Logistic regression is considered a linear model (linear combination[26] of vectors), like linear regression. It computes a weighted sum of the input features plus a bias term, but instead of outputting the result directly it uses a logistic function (the sigmoid function), which is a cumulative distribution function that takes a s-shaped curve and maps the output to values between 0 and 1 that can be interpreted as a probability. See Wikipedia[27] for a simple explanation. Logistic regression is suitable for problems with a dichotomous dependent variable and will be used as a base rate/benchmark solution as a basis for comparison to the other models. The logistic regression models will be tested on BOW and TF-IDF representations text and implemented using scikit-learn.

Feed forward neural network (FFNN): As a next step, building on logistic regression, we will try a solution using a fully connected neural network, also called a feed forward neural network. These work as a nonlinear function approximator, built by stacking layers of neurons, where each neuron is a linear combination of weights and inputs passed through an "activation function". The objective is to learn the best set of weights W_n ($n = n^{th}$ layer) that makes the function best fit the data. The process of training FFNNs involves two main processes: feed forward and backpropagation[28]. Each neuron can be viewed as a "logistic regression unit", although the activation function is not necessarily the sigmoid function (and probably should not be, since it has been shown to perform poorly on deep neural networks [19]). By adding layers and using a non-linear activation function the network can learn complex non-linear relationships, where each layer takes the previous layers output as input. FFNN will be used in this project on the word embedding representation of Quora questions and implemented using Keras. Being fully connected, every input is connected to every

[26] https://en.wikipedia.org/wiki/Linear_combination

[27] https://en.wikipedia.org/wiki/Logit

[28] https://en.wikipedia.org/wiki/Backpropagation

output, which is very computationally heavy, and we will test other types of network architectures as well.

Convolutional neural network (CNN): Most commonly used in image classification tasks, CNNs are neural networks for processing data that a grid-like topology [20]. In our case, text can be thought of as a 1-dimensinal grid. A thorough discussion of CNNs is not feasible to include here, and readers are referred to [20]. Convolution is a mathematical operation, and convolutional networks are "… simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers" [20, p. 326]. The GIF on Stanford's deep learning site[29] illustrates the process well. Convolutional neural networks are normally built up by a sequence of convolutional and pooling layer and ending with one or more fully connected layers for classification. A pooling layer replaces the output of the net at a certain location with a summary statistic of the nearby outputs [20]. In practical terms, pooling is used as a form of downsampling which reduces the dimensionality of a layer but retains the information, e.g. by keeping the highest values in a filter (max pooling) or averaging (average pooling). By stacking convolutional and pooling layers, a network can learn increasingly complex patterns. Below is an illustration of a CNN, taken from [21]:

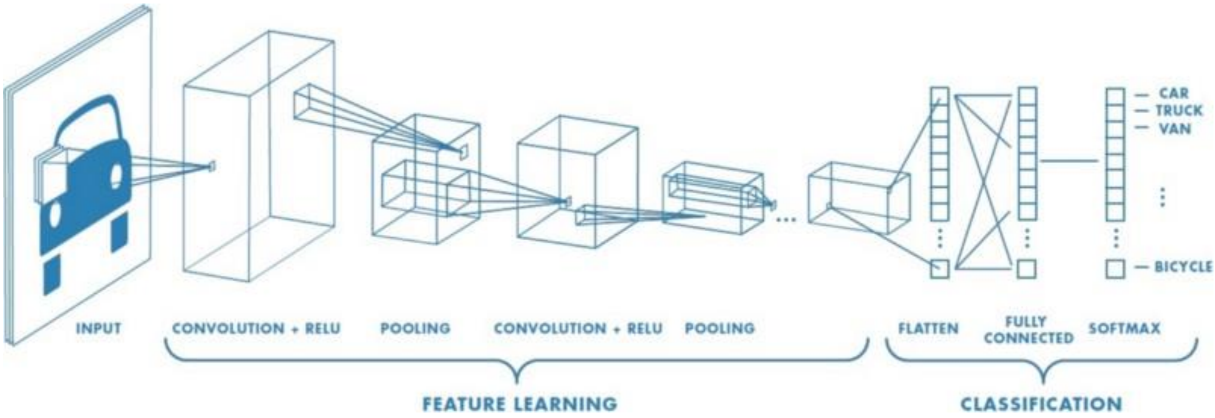


Figure 10: CNN architecture [21].

Recurrent Neural Networks (RNN): Many applications involve temporal dependencies, i.e. the current output depends on past inputs in addition to the current input. RNNs are neural networks that includes a form of memory and is especially suitable for processing sequential data. Natural language is an example of such data. Take for example the sentence "and should as such be a perfect fit for the classification task at hand. In very simple terms, RNNs, in contrast to CNNs and ANNs, does not treat each observation as independent, but is able to "remember" previous observations by allowing cyclical connections (adding the output of one observation to the input of the next). That is, while an ANN calculates the state of a hidden layer as

$h = \sigma(xW + b)$, a simple RNN includes the previous timestep: $h = \sigma(x_t W_x + s_{t-1} W_s + b)$. σ denotes the activation function, x is the input vector, W are weight matrices and s_{t-1} is the output of the previous time step. Figure 11 below, taken from [22], illustrates the architecture. In this picture,

[29] http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

$\overline{x}$ represents the input vector (a sequence of word embeddings in our case), $\overline{y}$ represents the output vector, and $\overline{s}$ represents the state vector. W_x, W_y and W_s are the weight matrixes connecting inputs to hidden/state, state to output and previous state to current state respectively. This simple RNN structure is sometimes called an Elman network.

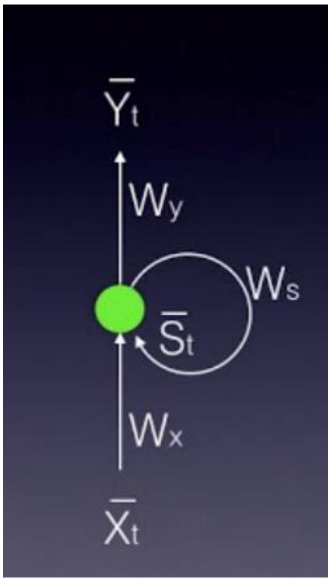


Figure 11: Folded RNN model [22]

Simple RNNs suffer from the vanishing gradients problem[30], hence this project we will use more complex versions of RNN networks which contains multiple state variables or "gates" designed to remember or forget previous inputs. This way, arbitrary time intervals or sequences can be represented. In simple terms, these gates are just more linear combinations and activation functions, each with their own learnable parameters. The architectures that will be tested are Long short term memory networks (LSTM) [23] and Gated recurrent unit networks (GRU) [24] – Keras provides a simple API for both of these. [25] has a very good introduction to LSTMs. GRUs and LSTMs are similar, but GRUs have fewer gates, and hence requires less matrix operations which makes it faster to train.

Benchmark Model

Being an active Kaggle competition, one benchmark is the Kaggle Leaderboard (LB) scores where the leader at the time of writing has a score of 0.7. Competing against the top contenders is a high bar however, and we will **use logistic regression with a simple bag-of-words approach (output of scikit-learn's CountVectorizer) as the primary benchmark for this project**. Since the supplied test dataset does not include a target variable, we will split the training data into a training and validation set, and use this for our benchmark, and for testing and improving models. Submitting every attempt to Kaggle would be too time-consuming. The benchmark model is built using scikit-

[30] https://en.wikipedia.org/wiki/Vanishing_gradient_problem

learn for producing the bag-of-words and fitting a logistic regression classifier. Removal of stop-words as well as tokenization and lemmatization were done as part of the pre-processing steps before fitting the model. The **full vocabulary** was included in the model (in contrast to the neural networks, for which this would have been too computationally heavy). The benchmark logistic regression model, with the class_weight parameter set to "balanced" to account for imbalance in the dependent variable, gives a F_1 score of **0.52** without searching and **0.58** by tuning the probability cutoff value.

Methodology

Data Preprocessing

As mentioned above, different classification models will use different word representations, and will potentially be tuned in different ways. For example – for "traditional" machine learning models to perform well, lemmatization and punctuation should be removed. Some of the word-embeddings (such as GloVe) however, includes punctuation, and removing stop words might degrade the performance of our models when using word embeddings. Testing and tuning will be done to see what works best for the different models. To prepare the text for machine learning, the following preprocessing steps will be implemented for at least one machine learning model:

Text cleaning: removing punctuation, very long numbers, multiple consecutive characters, replacing URLs. This will be done with Python and the Pandas library. This will remove most of the outliers described in the exploration section - since the outliers are just rare occurrences of very long or very short questions, they will not have much predictive value (nor would they have affected the results much in a negative direction, since they are rare).

Tokenization: Splitting the question text into lists of words and number-representation of words. This will be done in Python/Pandas for logistic regression, and by NLP utilities in the Keras library for neural networks.

Lemmatization: Replacing words with their basic form or lemma[31], e.g. cars->car. Spacy and NLTK will be used for this.

Removing stop-words: These are words that are common in any text and have little or no significance to the meaning of the text. Examples are "the", "a", "I" etc. Spacy and NLTK will be used for this.

Transforming text to numerical data that can be processed by the machine learning algorithms

Bag of words: Creating document word vectors with CountVectorizer, including bigram extraction.

TF-IDF: Using TfidfTransformer to create TF-IDF representations from CountVectorizer output.

[31] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Word embedding lookup: Transforming individual words to their pre-learned vector representations. This will be handled by pandas (reading vectors) and Keras (looking up embeddings)

Normalization: For improvement of logistic regression, normalization of the CountVectorizer will be tested. Scikit-learn will be used for this.

Implementation

The project work follows a typical NLP/machine learning workflow. The process is mostly linear but involves many iterations of steps 3-5-6 as shown in the figure below. Iterations between steps 5 and 6 involved tuning neural network architectures and parameters, and some iterations going back to step 3 to try different text representations, with and without stop-word removal and so on.

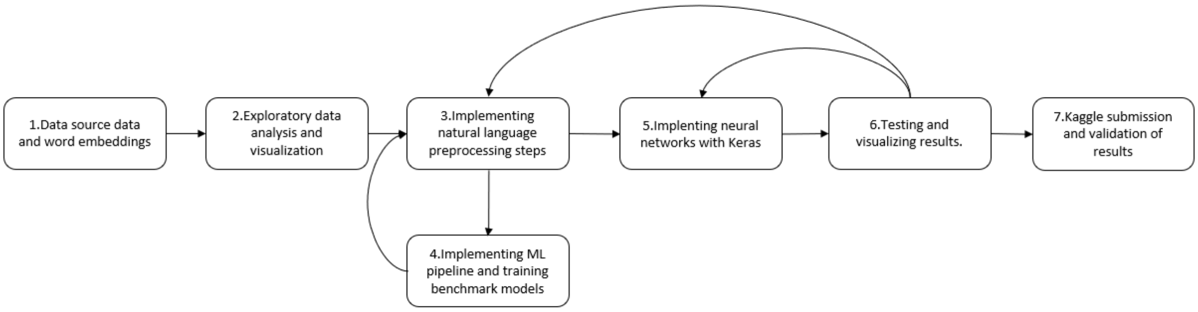

1.Data source data and word embeddings
2.Exploratory data analysis and visualization
3.Implementing natural language preprocessing steps
4.Implementing ML pipeline and training benchmark models
5.Implenting neural networks with Keras
6.Testing and visualizing results.
7.Kaggle submission and validation of results

Figure 12: Implementation workflow

As a first step, input parsing and specifically reading of embeddings is implemented as a function, since this requires some parsing/transformation effort, and it will be re-used in multiple Jupyter notebooks. In addition, I implemented utility functions for searching through threshold values (for finding best cut-off point for predicted probabilities) and visualization of predicted results. I chose to implement the following steps in **multiple** Jupyter notebooks, to avoid the clutter that often results from having too much functionality implemented in a single notebook. Since the neural network computations are time-consuming, the project was implemented on an AWS EC2 Deep Learning AMI with GPU access. Before NLP and machine learning tasks, I performed exploratory analysis and visualization of the raw datasets to gain insights about train/test data differences, properties related to the target variable etc. These tasks were all implemented using standard Python and the Pandas and Matplotlib libraries. As part of the exploratory analysis I also started the NLP implementation, since I wanted to compare commonly occurring words and n-grams in each of the target classes in the training data. For this I used the Spacy library to perform tokenization, stop-word removal and lemmatization.

Following basic NLP tasks, I implemented multiple versions of NLP-processing and tested the results in the benchmark logistic regression model as well as different neural network architectures. Before running the models, the training data were split into a training and validation dataset with a 70:30 ratio. There is a separate testing data set, but it does not include the target variable, and running iterations involving Kaggle-submissions would be too time-consuming. The benchmark model was implemented using scikit-learn's Pipeline, CountVectorizer, TfidfVectorizer and LogisticRegression classes. As the point of this exercise was to have a baseline for comparing more advance methods, I did not spend much time tuning the model, but set the "class_weight" parameter to "balanced", which makes the model penalize errors in the minority class more (and thereby handling imbalance better). Next, before implementing various versions of neural network architectures, I used Keras to prepare the text. This involved creating an embedding matrix to store vector representations of the words in our vocabulary. Since the vocabulary of the test data contained millions of tokens, the number of tokens to keep (sorted by number of occurrences) was a parameter to tune. Moreover, I used the Keras Tokenizer class to tokenize data for neural networks. This has the advantage of including some text cleaning (removing punctuations) and transforming the words to integers. The result is sequences of integers. Furthermore, Keras requires fixed size input, so all questions were padded or truncated to a fixed sequence length. Following neural network preparation, several neural network architectures were explored, and I spent a lot of time iterating on steps 5 and 6, testing variations in:

- Number of layers
- Number of output units (and hidden layer dimension in case of FFNN)
- Length of input sequence (text length)
- Size of vocabulary.
- RNN type (LSTM and GRU)
- Bidirectionality (doubles the number of outputs).
- Batch size
- Dropouts (regularization technique)
- Number of epochs

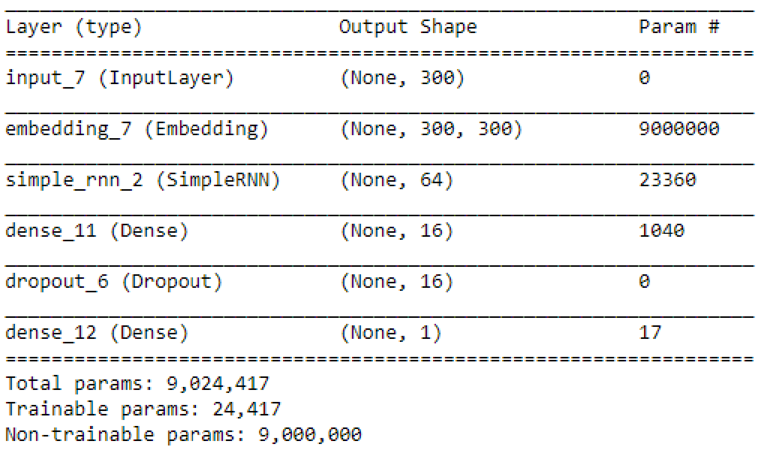
See refinement section. For all variations tested I performed a threshold search (predicted probability cut-off) and tested on the specified metric – F_1 score. All neural networks were optimized with the Adam optimizer[32] on a binary cross entropy loss function. During and after training, promising neural network models were saved to disk with the model.save() Keras method. The parameter and architecture tuning are very time consuming, and the project deadlines as well as price of GPU instance training set a natural limit to the amount of testing that could be performed.

Finally, the best performing models were run in a Kaggle kernel and submitted to the competition.

Refinement

As mentioned above, many attempts to refine the initial approach was performed. The figure below, produced by Keras model.summary() summarizes the first RNN model. This model resulted in a F_1 score of **0.60** on the validation set, which is a slight increase from the benchmark model.

[32] https://keras.io/optimizers/



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 300)               0
_________________________________________________________________
embedding_7 (Embedding)      (None, 300, 300)          9000000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 64)                23360
_________________________________________________________________
dense_11 (Dense)             (None, 16)                1040
_________________________________________________________________
dropout_6 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 17
=================================================================
Total params: 9,024,417
Trainable params: 24,417
Non-trainable params: 9,000,000
```

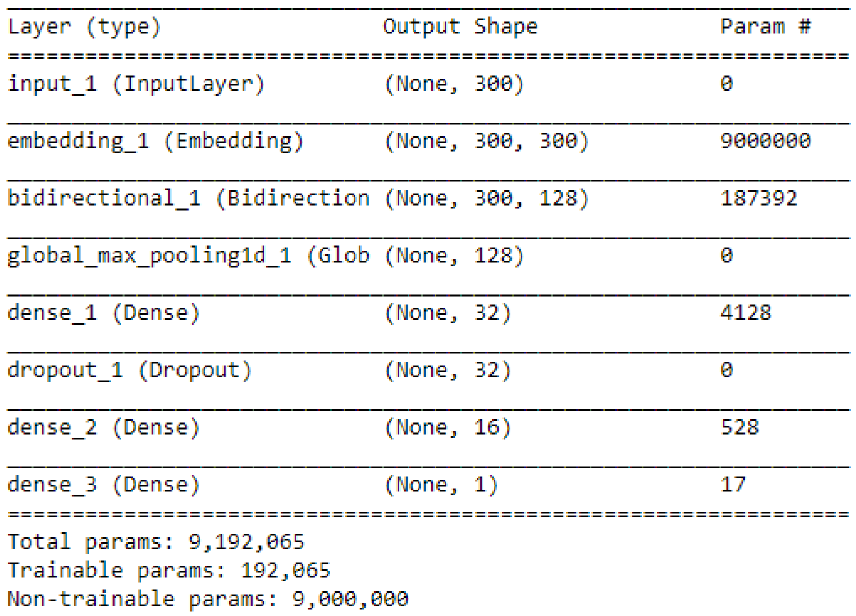
Figure 13: Initial simple RNN

Given the time it takes to train a neural network on text data, I did all parameter and architecture searching manually, i.e. no grid search wrapping. The table below summarizes the most important testing that was recorded (some are actual parameters to the APIs, some are names given by the author to the tests that were done). A number of other variations were also tested – a 3-layer LSTM network would take many hours to train, a 3-layer FFNN finish in minutes.

Parameter	**Description**	**Values**	**Chosen value**
RNN type	As described in algorithms and techniques section.	SimpleRNN, LSTM, GRU	LSTM
Bidirectionality	Bidirectional wrapper for RNNs (producing 2x outputs one for each direction of the input sequence)	Yes/No	Yes
Number of layers	Number of stacked LSTM cells	1,2	2
Length of input sequence	How long a sequence to pass through the networks.	100, 150, 300	150
Size of vocabulary	How many of the words from the training data to include.	20000, 25000, 30000	30000
Batch size	How big a batch to run before updating weights	256, 512	512
Dropouts	Regularization technique	1 and 2 layers, rate 0.1	1 layer, rate 0.1
Epochs	Number of passes through the training data on which to train the network.	1,3,4,10 Depending on architecture	4

Table 5: Model refinement parameters

The figure below summarizes the final model. Since pretrained embeddings are used, the number of trainable parameters is (relatively) few.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 300)               0         
_________________________________________________________________
embedding_1 (Embedding)      (None, 300, 300)          9000000   
_________________________________________________________________
bidirectional_1 (Bidirection (None, 300, 128)          187392    
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                4128      
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 16)                528       
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17        
=================================================================
Total params: 9,192,065
Trainable params: 192,065
Non-trainable params: 9,000,000
```

Figure 14: Final LSTM model

Results

Model Evaluation and Validation

Model evaluation and validation was done mostly by using the training and validation split and some but not all versions were submitted to Kaggle for LB scoring. F_1 was consistently used to evaluate models. The table below summarizes the results, and the precision-recall curve of the final model is supplied in figure 15. Note that only the final versions for each model has been kept in the notebooks as well as in the table below – keeping everything would have resulted in a lot of code and notebooks. All the results below (except logistic regression) used GloVe pretrained word embeddings. As expected, RNN types of networks performed well. Even though CNNs have showed some promise (c.f. [8]) it did not perform well in this case, but I did not have time to test different architectures and hyperparameters. It is interesting though, that while it only scored the same as logistic regression on the validation set, it generalized better to the test dataset submitted to Kaggle.

Model	**F_1-score at best cutoff**	**LB score**
Logistic regression	0.58	0.515
CNN	0.58	0.573
FFNN	0.58	n/a
Simple RNN	0.6	n/a
GRU	0.66	n/a
Bidirectional GRU	0.65	0.635

LSTM	0.66	n/a
Bidirectional LSTM	**0.67**	**0.658**

Table 6: Comparison of models

Bidirectional LSTM ended up producing the highest F_1 score, but only resulted in a place among the top 60 % on the Kaggle public LB at the time of writing, which is not very good. On the other hand, an improvement of only 0.036 (to 0.694) would have taken us to the top 3%. I am sure the results would have been better by copy-pasting some existing code, but that would certainly have affected the learning value of the project.

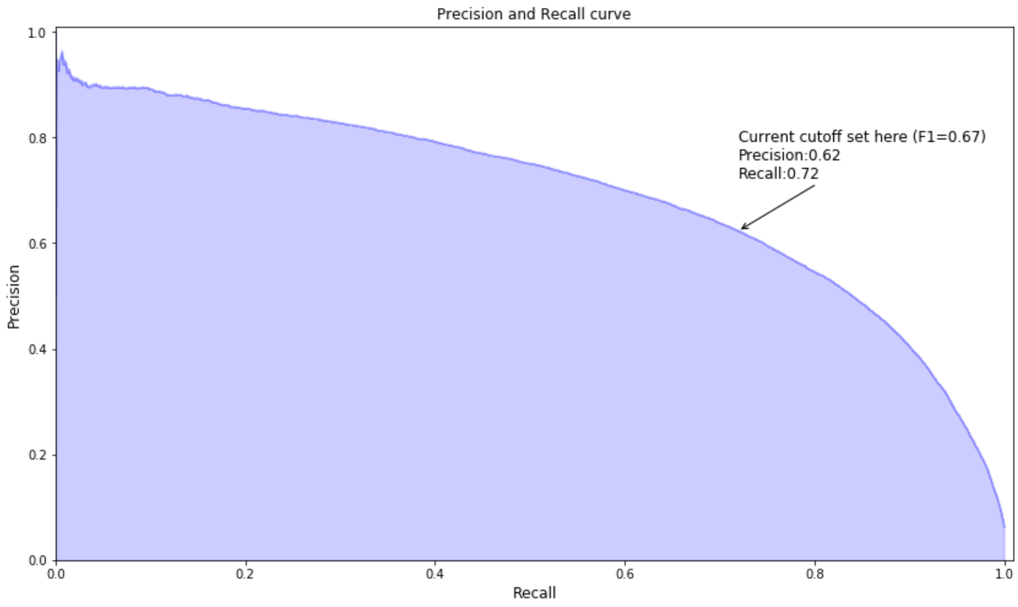


Figure 15: Precision-recall of final LSTM model

The precision recall curve shows in a visual way what the F_1 score reflects: precision falls quite steeply when the recall increases.

Justification

I applied multiple neural network models that have been found to work well on text classification problems. The goal was to beat the benchmark, and the final LSTM model produced an F_1 score that is a significant improvement of the initial logistic regression model. The final model ended up being a RNN variant as expected. Since it produces an OK F_1 result on the validation set and on the Kaggle LB, the model seems robust enough for the problem. Nevertheless, given more time, many improvements could have been implemented, as discussed in the improvements section below.

Conclusion

Free-Form Visualization

The first figure below shows how the loss of the final LSTM model decreased with each epoch. As both validation loss and training loss is decreasing at each step, including the last, adding more epochs could potentially have improved the results. With more time, I would have liked to do better hyperparameter tuning, such as testing different batch sizes, different optimizers etc. The second figure shows the confusion matrix at the optimal cutoff value (maximizing F_1 score). Clearly, if the current model were to be used, a lot of false positives would have had to be accepted.

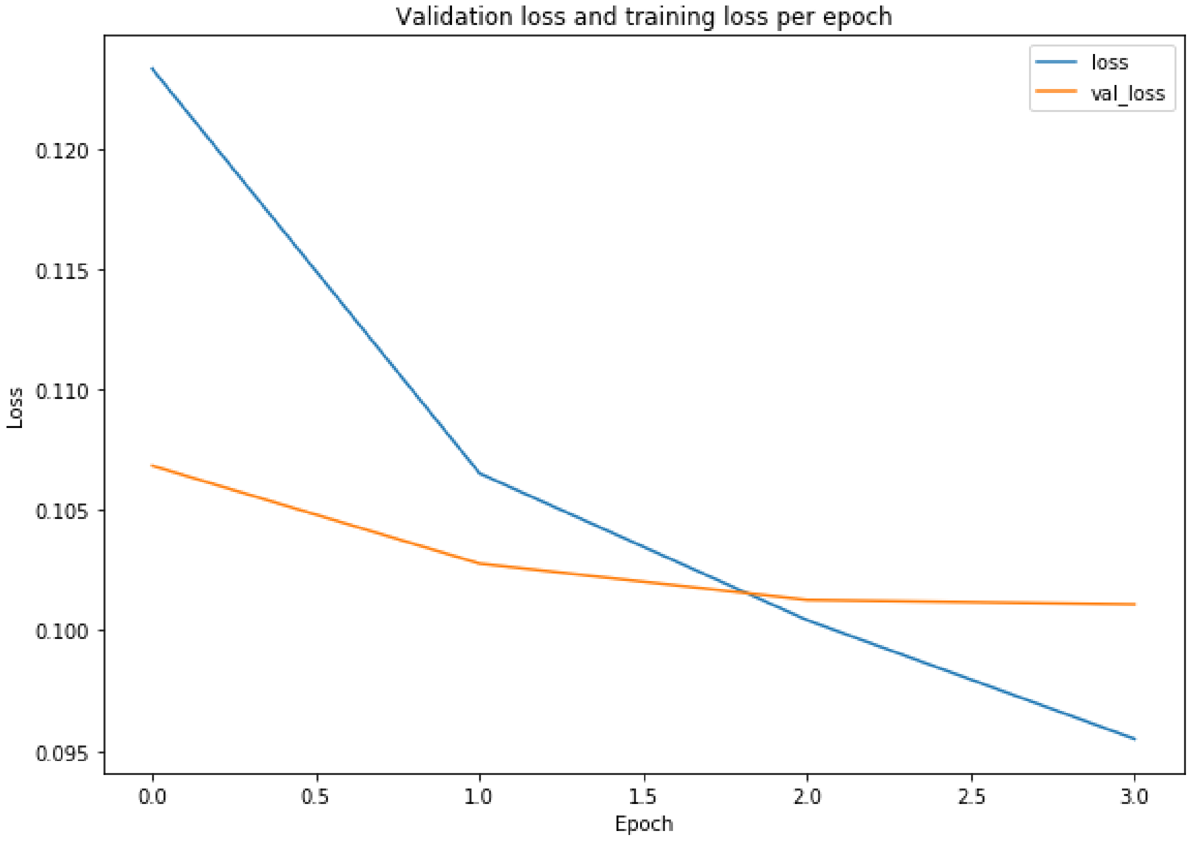


Figure 16: Training and validation loss per epoch for final model

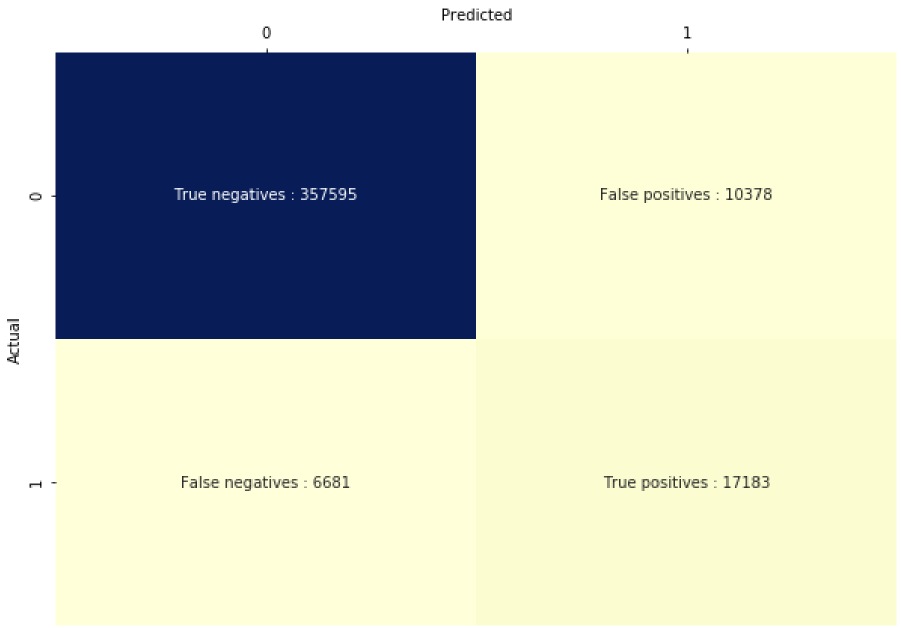


Figure 17: Confusion matrix for final model at optimal cutoff value

Reflection

This capstone project turned out to be much more time consuming than initially planned. I did not have much experience with any of the techniques and algorithms used: NLP and deep learning. The project was very challenging, and I had to read up a lot on the theory to have a good enough understanding for implementation, and what architectures could fit a text classification problem. I followed a common advanced analytics workflow, with data wrangling, exploratory data analysis and visualization, NLP-specific transformation and machine learning tuning. I think the final RNN proved to be an acceptable solution. The most time-consuming part was testing different neural network architectures, and I was not able to try anywhere near as much as I would have liked. The number of permutations that can be tried are simply staggering, and I think only experience can help the intuition and speed up the iterations (not considering automatic machine learning products). As for computational power, I used a GPU instance on AWS, but it still took well over an hour to train the final bidirectional LSTM network with 4 epochs. I had more ideas on how to improve the models, and I would have liked to try other machine learning algorithms as well, such as XGBoost, which often performs well on classification tasks. In addition, I was only able to test one of the pretrained word embeddings – GloVe. I initially planned to test word2vec as well. Nevertheless, the experience was educational and fun, and I have learned a lot. NLP is an area of active research, and new papers and blog posts are produced every day – the models produced in this project is a starting point that I will continue to build on to learn more about NLP and deep learning.

Improvement

There are many improvements that could be implemented on the final model. For instance, I could have tried combining word embeddings with other features. I could also have spent more time tuning hyperparameters, e.g. learning rate. There is also a technique called "attention" in RNNs that is getting some attention(!) currently and have been getting better results than my model on the Kaggle leaderboard. Typically, ensembles of models will perform better, so I could also have tried combining multiple saved models, both neural networks and other types of models. I ended up doing very little feature engineering and no cross-validation on the final model, which is also a point of improvement. I would also have liked to try:

- Testing different levels of data preprocessing.
- Different types of regularization.
- Techniques for balancing training data, such as up- or down-sampling.

Time and computing power have been a limiting factor in this project, but the work has been very rewarding.

References

[1] Quora, "Kaggle," September 2018. [Online]. Available: https://www.kaggle.com/c/quora-insincere-questions-classification. [Accessed 23 November 2018].

[2] G. G. Chowdhury, "Natural Language Processing," *Annual Review of Information Science and Technology,* pp. 51-89, 2005.

[3] H. Allcott and M. Gentzkow, "Social Media and Fake News in the 2016 Election," *Journal of Economic Perspectives,* pp. 211-236, 2017.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781,* 2013.

[5] G. G. Parker, M. W. Van Alstyne and S. P. Choudary, Platform Revolution - How network markets are transforming the economy and how to make them work for you, New York: W. W. Norton & Company, Inc., 2016.

[6] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer and R. Wirth, *CRISP-DM 1.0 Step-by-step data mining guide,* 2000.

[7] J. Brownlee, "A Gentle Introduction to the Bag-of-Words Model," 9 October 2017. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/. [Accessed 32 November 2018].

[8] S. Lai, L. Xu, L. Kang and J. Zhao, "Recurrent Convolutional Neural Networks for Text Classification," in *AAAI*, Austin, Texas, 2016.

[9] O. Levy, Y. Goldberg and I. Dagan, "Improving Distributional Similarity with Lessons Learned from Word Embeddings," in *Transactions of the Association for Computational Linguistics*, 2015.

[10] A. M. Dai and Q. V. Le, "Semi-supervised Sequence Learning," in *Advances in Neural Information Processing Systems*, 2015.

[11] S. M. Lyon, "Quora," 9 May 2017. [Online]. Available: https://www.quora.com/What-is-the-maximum-length-of-a-question-on-Quora. [Accessed 3 December 2018].

[12] J. Pennington, R. Socher and C. D. Mannin, "GloVe: Global Vectors for Word Representation," *Empirical Methods in Natural Language Processing (EMNLP),* pp. 1532-1543, 2014.

[13] J. Wieting, M. Bansal, K. Gimpel, K. Livescu and D. Roth, "From Paraphrase Database to Compositional Paraphrase Model and Back," in *Transactions of the Association for Computational Linguistics*, 2015.

[14] J. Ganitkevitch, B. Van Durme and C. Callison-Burch, "PPDB: The paraphrase database," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013.

[15] Facebook, "FastText," [Online]. Available: https://fasttext.cc/. [Accessed 23 November 2018].

[16] Y. Goldberg and O. Levy, "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722,* 2014.

[17] M. J. Kusner, Y. Sun, N. I. Kolkin and K. Q. Weinberger, "From word embeddings to document distances," in *International Conference on Machine Learning*, 2015.

[18] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems,* pp. 3111-3119, 2013.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010.

[20] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[21] R. Prabhu, "Understanding of Convolutional Neural Network (CNN)—Deep Learning," 4 March 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Accessed 5 December 2018].

[22] Udacity, *Deep Learning Nanodegree - Course offered by Udacity,* 2018.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* pp. 1735-1780, 1997.

[24] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Gated feedback recurrent neural networks," in *International Conference on Machine Learning*, 2015.

[25] C. Olah, "colah's blog," 27 August 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 5 December 2018].

[26] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *arXiv preprint arXiv:1408.5882,* 2014.