Quora Insincere Questions Challenge

12 December 2018

Outline of Report

Market overview

Quora

Ask

Answers.com

reddit

Quora competitors

- Ask.com outperforms both Quora and Answers.com when looking at the overall desktop and mobile visits, despite its losses in traffic in 2015 (it still managed to concluding the year with more than 1.1 billion total visits).
- However, in the last few years, Quora has experienced a constant growth thanks to organic search traffic. The Google Phantom update (aimed to weed out poor how-to websites) allowed it to significantly improve it's search ranking, compared to its competitors. In fact, in 2015 only, Quora was capable of increasing its organic search visits by 120%.
- Reddit is stickier than Quora, probably since it has been around for longer (2005 against 2009), but rather than focusing on the ability of writers to satisfy users' requests, its format is more similar to a forum where anyone can join the discussion. This is the reason why people only looking for a concise answer prefer Quora, since they can just get in, read the answer and leave.

Quora business model

- The main source of growth for Quora has been represented by the founding coming from several rounds of investments made by venture capital firms, overall amounting to 452 million dollars.
- The main peculiarity, that constitutes one of the pillars on which the platform is built on, are question answered by real people, whereas search engines as Google simply match user queries with the most relevant content already available on the web. Alongside this, Quora is an engaged community, whose members are the Quorans, where expertise can be accessed freely and easily.
- An additional source of revenue is generated through advertisements.

Quora distinctive traits

- Instead of being based on algorithms that rank web pages according to their relevance with respect to a certain query, Quora starts from human creation.
- A growing number of businesses is now adopting Chatbots and softwares capable of creating contents automatically, still, people exhibit the tendency to prefer interactions with other human beings and this is what Quora does: the majority of questions is about life and career advice. Users are not only looking for answers; they also want to find someone that can sympathize with them or that has already gone through what they are experiencing.
- Quora aims to provide value to users and to writers as well.

Goal of the Analysis

- Nowadays websites are increasingly looking for a way to handle toxic and discordant content. Among them there is Quora, which aims to create a productive environment, where users will feel free to ask questions that matter to them and share their knowledge meaningfully.
- The platform was created with the intent to empower people to learn from one another: the questions they make are mapped to good quality answers. Issues arise when it comes to insincere questions, built upon false premises, that can destroy the sentiment of trust pervading the community, destroying the underlying policy of Quora: “Be Nice, Be Respectful”.
- Their mission is “to share and grow the world’s knowledge, building a world-class team to help them achieve this mission”

Goal of the Analysis: Insincere questions, a brief contextualization

- Insincere questions are not meant to ask for useful answers on a topic of interest, they can in fact be rhetorical or statements. Given this peculiarity, people can distinguish them from usual and sincere questions by looking at some factors:
  - The tone, which rather than being non-neutral is exaggerated to draw attention to a point or a position
  - Their ultimate intent is to inflame, suggesting a discriminatory idea, seeking confirmation of a stereotype, insulting a person or a group, often basing upon characteristics that are not measurable
  - They are generally not based on evidence, nor have a solid connection with reality
  - They sometimes have sexual content to be more provocative and generate a wider response (whether disdain or shock) from the community of users

Goal of the Analysis

- Kaggle set up a competition to encourage contestants to develop an algorithm capable of predicting whether a question asked on Quora is sincere or not, as defined by Quora’s “Be Nice, Be Respectful” policy guidelines.
- To evaluate the performance of our algorithm, we were instructed to optimize for the F1 score instead of just the accuracy. This is because solely focusing on the accuracy would give a misleading impression of the performance of the model in this task. Case in point: it is possible to obtain an accuracy score of 94%, by classifying every question as sincere, given the fact that this is the ratio of sincere to insincere questions in the dataset. Hence the F1 score captures a more holistic picture of the performance of the model.

The F1 score is computed as follows, thus giving a completely biased algorithm a score of 0.

$$F_\beta = \frac{(1+\beta^2)\cdot \text{true positive}}{(1+\beta^2)\cdot \text{true positive} + \beta^2 \cdot \text{false negative} + \text{false positive}}$$

Dataset Used

The training data provides the question that was posed and how it was labeled (if insincere, *target = 1*)

File descriptions
* train.csv - the training set
* test.csv - the test set
* sample_submission.csv - A sample submission in the correct format
* Word embeddings to be used in the model

Data fields
* qid - unique question identifier
* question_text - Quora question text
* target - a question labeled "insincere" has a value of 1, otherwise 0

Dataset Used: Embeddings

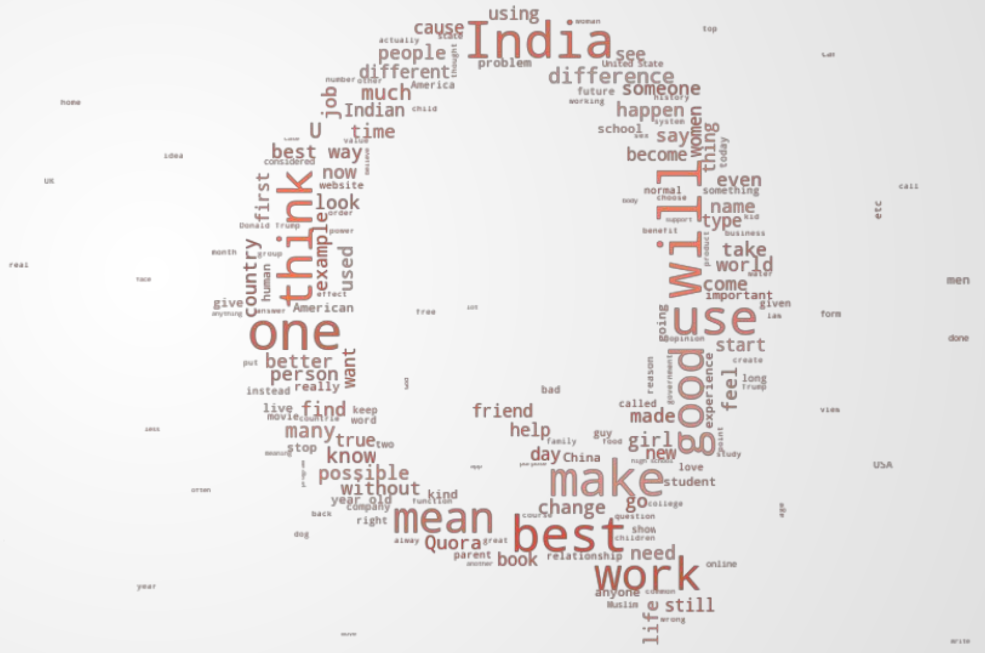
Additionally, we utilized four different publicly-available embeddings (complying to the competition rules). Each of these pre-trained word embeddings, were trained on English text and have a dimensionality of 300.

These embeddings are, in no particular order:

1. **Word2Vec (trained on the Google News Corpus)**
2. **GloVe (Global Vectors for Word Representations, maintained by Stanford NLP Group)**
3. **Paragram (tuned embeddings from compositional paraphrased model of Wieting et al.)**
4. **fastText (trained on a large English Wikipedia dataset)**

Exploratory Data Analysis: Word Cloud

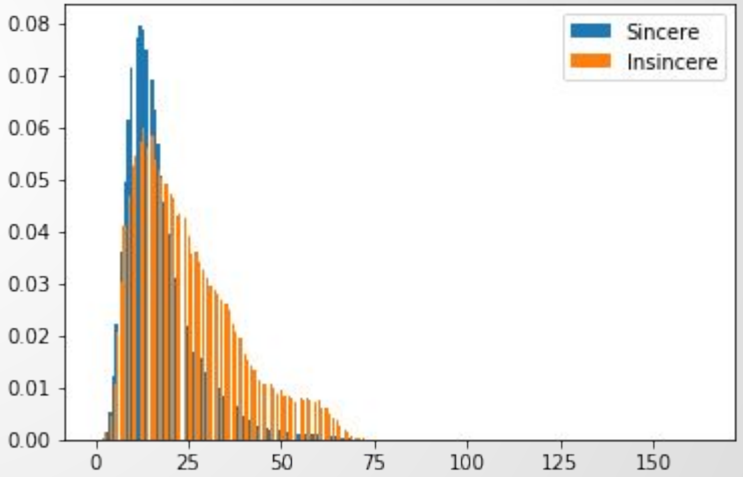
- We created this word cloud using Python’s equally named module (see Quora_Insincere_Questions_3.ipynb).
- It takes a vector graphic, in our case shaped like the Quora ‘Q’, the desired color (shades of red for us) and a string containing all our sentences. The library then creates a word cloud, emphasising the most used words.
- One can, for example, see that India is mentioned fairly often, given its size and intensity of color



Exploratory Data Analysis: Syllables

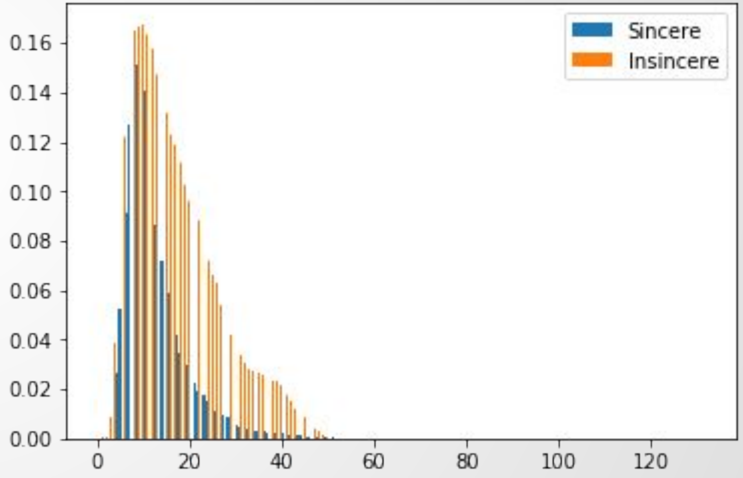
To start with, we decided to get an idea on the composition of the training set, so we counted the number of insincere and sincere questions, which are 80810 and 1225312 respectively.

- Next, we deemed it necessary to explore the data a bit further, first by looking at the syllables composing the words in each category.
- From the graph, it is evident that insincere questions tend to have more syllables (this might be due to the use of longer words or of more structured sentences).



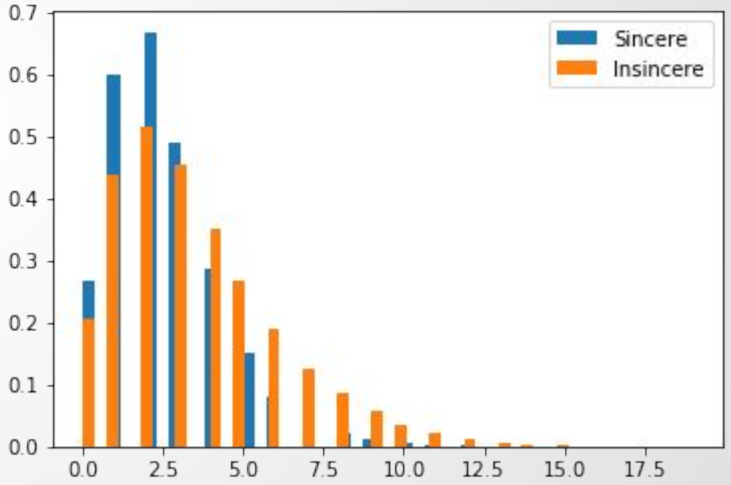
Exploratory Data Analysis:Lexicon

- From the distributions of the words used when posing both “sincere” and “insincere” questions, we can see that the word choice does not vary that much: their shape is very similar, what does change somewhat is the frequency of the words.



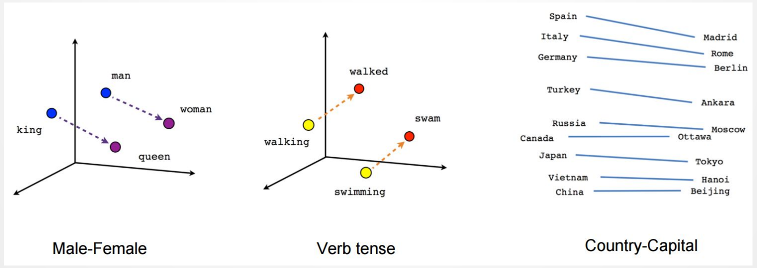
Exploratory Data Analysis: Difficult words

- Clearly, the distribution related to insincere questions is slightly more skewed to the right, which can be recollected to the fact that they are not actually looking for an answer, but they are just meant to induce reactions in users (make a statement).
- This might be due to the fact that complex, or difficult words, are typically used when sophistry–clever use of arguments to convey a false statement as true– is committed (which is directly linked to “insincere” questions).



Data Pre-Processing: Word Embeddings Overview

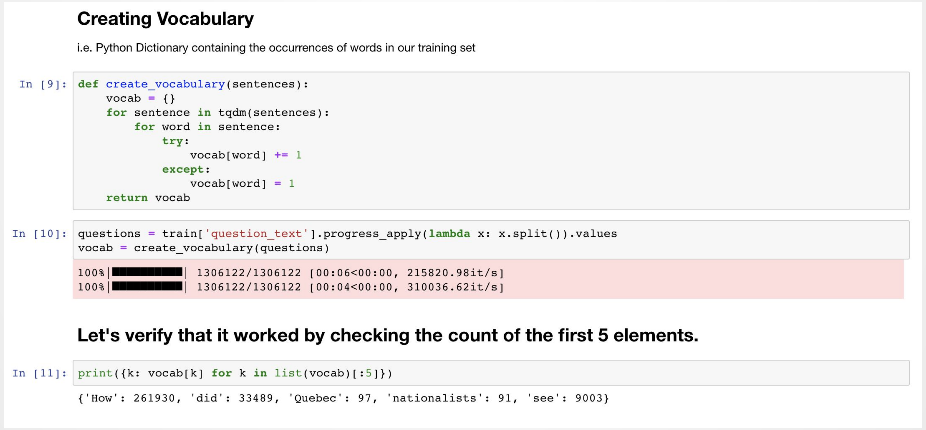
- Word Embeddings are the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.
- Conceptually the technique involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension.



- The technique has been shown to drastically improve the performance of algorithms (especially Deep Learning methods) in NLP tasks.
- Given the combination of well-documented resources and the performance boosts obtained by using Word Embeddings, we opted to apply this tool in our analysis.

Data Pre-Processing: Word Embeddings Vocabulary Creation

- In order to optimize our Embedding procedure, we created a so called ‘Vocabulary’: a simple hash-table containing the occurrences of the ‘words’ in our Training Set.



Creating Vocabulary

i.e. Python Dictionary containing the occurrences of words in our training set

```
In [9]: def create_vocabulary(sentences):
            vocab = {}
            for sentence in tqdm(sentences):
                for word in sentence:
                    try:
                        vocab[word] += 1
                    except:
                        vocab[word] = 1
            return vocab
```

```
In [10]: questions = train['question_text'].progress_apply(lambda x: x.split()).values
         vocab = create_vocabulary(questions)

100%|██████████| 1306122/1306122 [00:06<00:00, 215820.98it/s]
100%|██████████| 1306122/1306122 [00:04<00:00, 310036.62it/s]
```

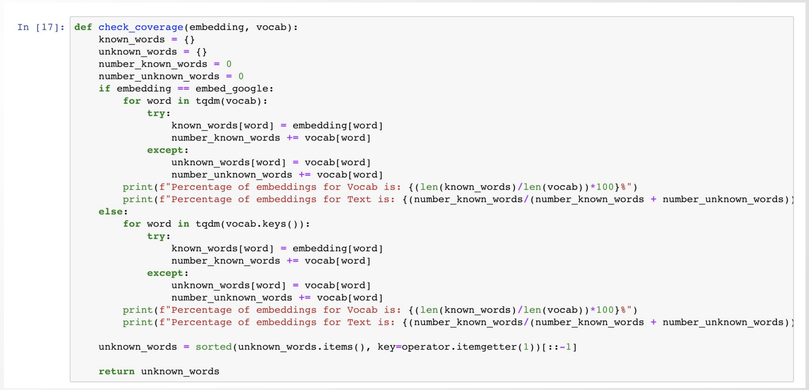
Let's verify that it worked by checking the count of the first 5 elements.

```
In [11]: print({k: vocab[k] for k in list(vocab)[:5]})

{'How': 261930, 'did': 33489, 'Quebec': 97, 'nationalists': 91, 'see': 9003}
```

- More specifically, this will allow us to compare the performance of the different Word Embeddings at our disposal (Word2Vec, GloVe, Paragram, and FastText) on our particular dataset.

Data Pre-Processing: Word Embeddings Performance Evaluation

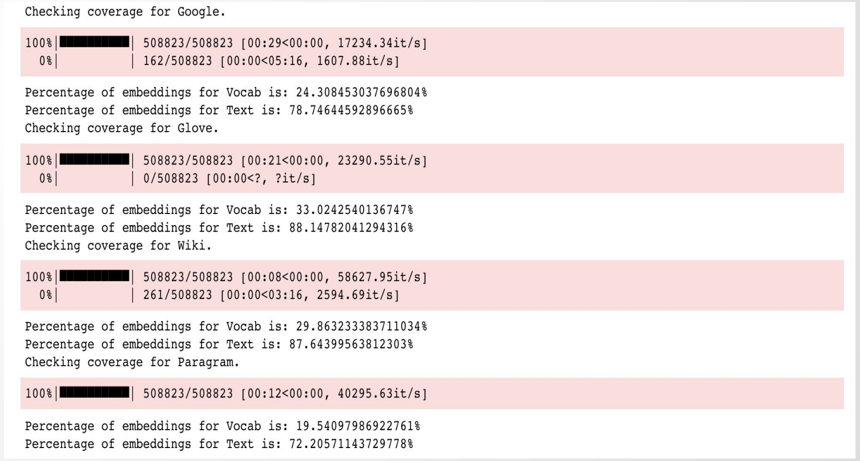
- We proceed to evaluate the performance (coverage) of the Word Embeddings by computing both the proportion of ‘words’ in our ‘Vocabulary’ also contained in each of the Embeddings, as well as the proportion of the occurences of the ‘words’ in our ‘Vocabulary’ that have an Embedding.
- This was done using the Python Script below (technical note: we had to treat the ‘Word2Vec Embedding trained on Google News’ differently from the other Embeddings)



```
In [17]: def check_coverage(embedding, vocab):
             known_words = {}
             unknown_words = {}
             number_known_words = 0
             number_unknown_words = 0
             if embedding == embed_google:
                 for word in tqdm(vocab):
                     try:
                         known_words[word] = embedding[word]
                         number_known_words += vocab[word]
                     except:
                         unknown_words[word] = vocab[word]
                         number_unknown_words += vocab[word]
                 print(f"Percentage of embeddings for Vocab is: {(len(known_words)/len(vocab))*100}%")
                 print(f"Percentage of embeddings for Text is: {(number_known_words/(number_known_words + number_unknown_words))
             else:
                 for word in tqdm(vocab.keys()):
                     try:
                         known_words[word] = embedding[word]
                         number_known_words += vocab[word]
                     except:
                         unknown_words[word] = vocab[word]
                         number_unknown_words += vocab[word]
                 print(f"Percentage of embeddings for Vocab is: {(len(known_words)/len(vocab))*100}%")
                 print(f"Percentage of embeddings for Text is: {(number_known_words/(number_known_words + number_unknown_words))

             unknown_words = sorted(unknown_words.items(), key=operator.itemgetter(1))[::-1]

             return unknown_words
```

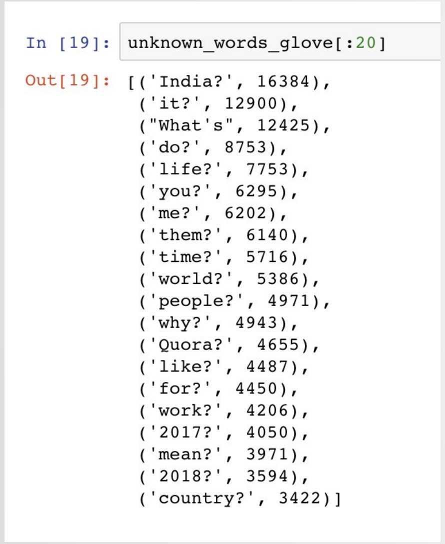
Data Pre-Processing: Word Embeddings Initial Performance Evaluation

- The initial results obtained by utilizing the **check_coverage** Python function are displayed on the right.
- We can observe that the best results are obtained by using the GloVe embedding, hower notice that approximately only 30% of our Vocabulary in this case has an embedding. In other words, around 10% (i.e. 100% - 88.147%) of our data *in the current state* is useless!



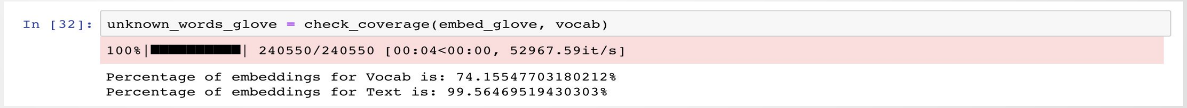
```
Checking coverage for Google.

100%|██████████| 508823/508823 [00:29<00:00, 17234.34it/s]
  0%|          | 162/508823 [00:00<05:16, 1607.88it/s]

Percentage of embeddings for Vocab is: 24.308453037696804%
Percentage of embeddings for Text is: 78.74644592896665%
Checking coverage for Glove.

100%|██████████| 508823/508823 [00:21<00:00, 23290.55it/s]
  0%|          | 0/508823 [00:00<?, ?it/s]

Percentage of embeddings for Vocab is: 33.0242540136747%
Percentage of embeddings for Text is: 88.14782041294316%
Checking coverage for Wiki.

100%|██████████| 508823/508823 [00:08<00:00, 58627.95it/s]
  0%|          | 261/508823 [00:00<03:16, 2594.69it/s]

Percentage of embeddings for Vocab is: 29.863233383711034%
Percentage of embeddings for Text is: 87.64399563812303%
Checking coverage for Paragram.

100%|██████████| 508823/508823 [00:12<00:00, 40295.63it/s]

Percentage of embeddings for Vocab is: 19.54097986922761%
Percentage of embeddings for Text is: 72.20571143729778%
```

Data Pre-Processing: Word Embeddings Improvements



```
In [19]: unknown_words_glove[:20]

Out[19]: [('India?', 16384),
          ('it?', 12900),
          ("What's", 12425),
          ('do?', 8753),
          ('life?', 7753),
          ('you?', 6295),
          ('me?', 6202),
          ('them?', 6140),
          ('time?', 5716),
          ('world?', 5386),
          ('people?', 4971),
          ('why?', 4943),
          ('Quora?', 4655),
          ('like?', 4487),
          ('for?', 4450),
          ('work?', 4206),
          ('2017?', 4050),
          ('mean?', 3971),
          ('2018?', 3594),
          ('country?', 3422)]
```

- We opted to proceed with the GloVe embedding given the initial better performance on our dataset. However, as mentioned, we could certainly process our textual data in order to achieve an improved coverage. Therefore, we proceeded to investigate the instances of words not covered by GloVe in order to see if there's any obvious fixes.
- As can be noticed by the output to the left, the concatenation of words and punctuation marks, as well as contractions (e.g. "What's") posed a problem for the GloVe embedding. Hence, we proceeded to take care of this problem by both separating our 'words' from 'punctuation marks', and mapping contractions to their expanded forms (e.g. "What's" -> "What is")
- (Note that punctuation marks **are** supported by GloVe, which means that no data loss was incurred during this procedure)

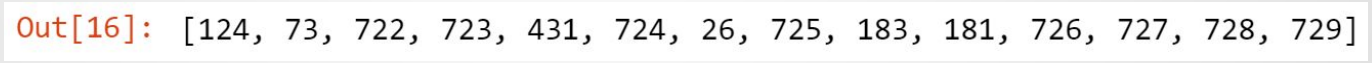


```
In [32]: unknown_words_glove = check_coverage(embed_glove, vocab)

         100%|██████████| 240550/240550 [00:04<00:00, 52967.59it/s]

         Percentage of embeddings for Vocab is: 74.15547703180212%
         Percentage of embeddings for Text is: 99.56469519430303%
```

- As we can see, major improvements were obtained by this preprocessing. From a coverage of 33% of embeddings for our Vocab to 74%, and from a coverage of 88% of embeddings for all of our Text to 99.5%. Which means that only less than 0.5% of our textual data is unusable, say because it is random gibberish.

Data Modeling - Naive Approach

- The first approach we tried was a naive Linear Classifier without a specialized word dictionary. We started by building a dictionary simply by mapping every word present in the dataset to an integer. For example, the first sentence is 'How did Quebec nationalists see their province as a nation in the 1960s?' and gets transformed to [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13].
- The sentence "What do physicists, mathematicians, computer scientists and philosophers think of David Deutsch's 'Constructor Theory'?" becomes [124, 73, 722, 723, 431, 724, 26, 725, 183, 181, 726, 727, 728, 729].

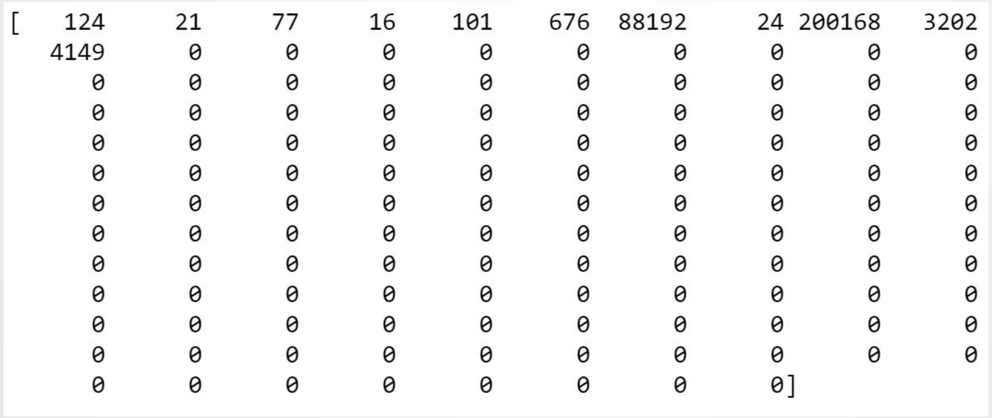


```
Out[16]:  [124, 73, 722, 723, 431, 724, 26, 725, 183, 181, 726, 727, 728, 729]
```

- We also didn’t remove any stopword/punctuation nor transformed everything in lowercase as those features could help determine a insincere question.

Data Modeling - Naive Approach

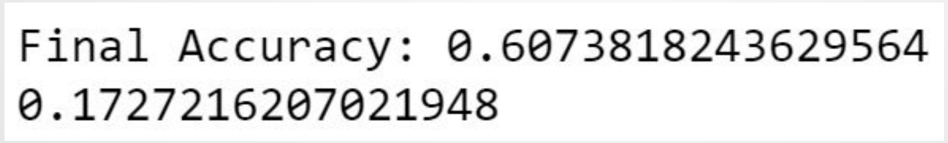
- Using the Keras API, we then unified the length of the arrays, turning each sentence into this format, where 0 represents no word on that position. The maximum length we imposed is 128, while the longest title contained 134 words before the resizing.



```
[    124      21      77      16     101     676   88192      24  200168    3202
    4149       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0       0       0
       0       0       0       0       0       0       0       0]
```

Data Modeling - Naive Approach

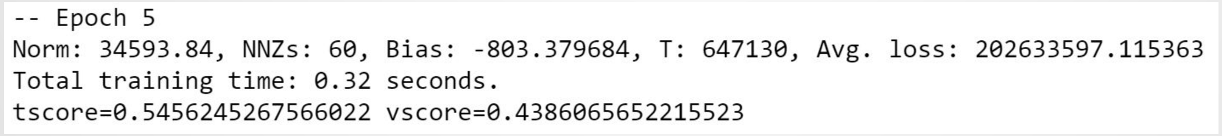
- On this set of arrays we first tried to execute Linear Regression, we implemented it via Scikit Learn. The first results were basically useless as the dataset itself is highly biased with a ratio of over 94% of questions classified as sincere. Following this, we resampled the data using RandomUnderSampler, which we installed into Python.
- As instructed by the task description, we used the F1 score to distinguish how successful our algorithms are. That said, this first try achieved a score of 0.17. Looking at the confusion matrix, we see 183340 “True Negatives”, but also 8735“False Negatives”, meaning we have a high amount of sincere questions classified as insincere.



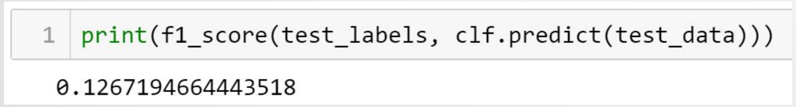
```
Final Accuracy: 0.6073818243629564
0.1727216207021948
```

Data Modeling - Naive Approach

- Moving on from this method we decided to go for a little more advanced classifier, specifically a Support Vector Machine. While the accuracy is higher at 94%, this was achieved by simply classifying all questions as sincere, even though we trained it with a balanced dataset. This is also represented in or F1 score, which was 0.
  Using a Stochastic Gradient Descent Algorithm with a Support Vector Machine as a loss function, the F1 score just slightly increases to 0.13
- We then modified the dataset slightly and moved temporarily to KNIME.



```
-- Epoch 5
Norm: 34593.84, NNZs: 60, Bias: -803.379684, T: 647130, Avg. loss: 202633597.115363
Total training time: 0.32 seconds.
tscore=0.5456245267566022 vscore=0.4386065652215523
```

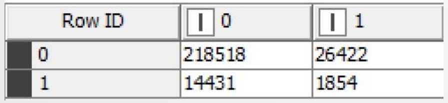


```
print(f1_score(test_labels, clf.predict(test_data)))
```

```
0.1267194664443518
```

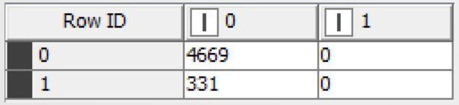
Data Modeling - Naive Approach

- For the last “naive approach”, we used the dataset, already mapped using the negative words database by Google, and imported it into KNIME. There we used the Logistic Regression and, with undersampled and reduced data, a Support Vector Machine.
- The resulting confusion matrix were the following:
    - Logistic Regression:

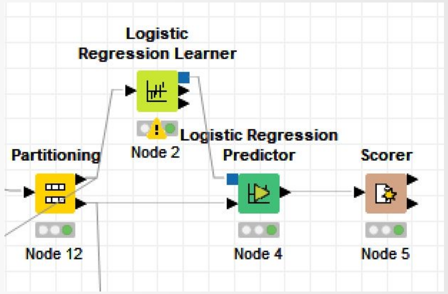


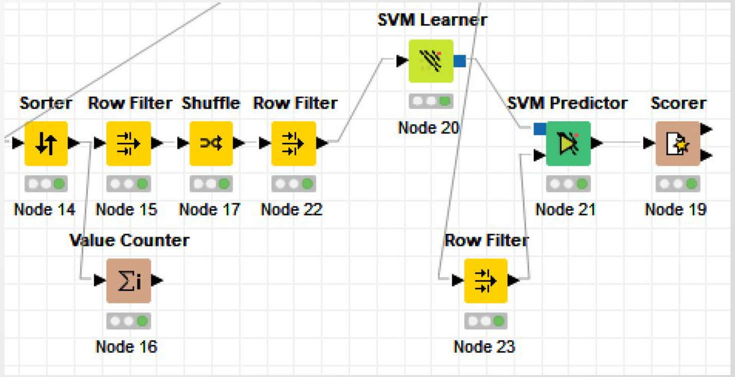
Row ID	0	1
0	218518	26422
1	14431	1854

    - Support Vector Machine:



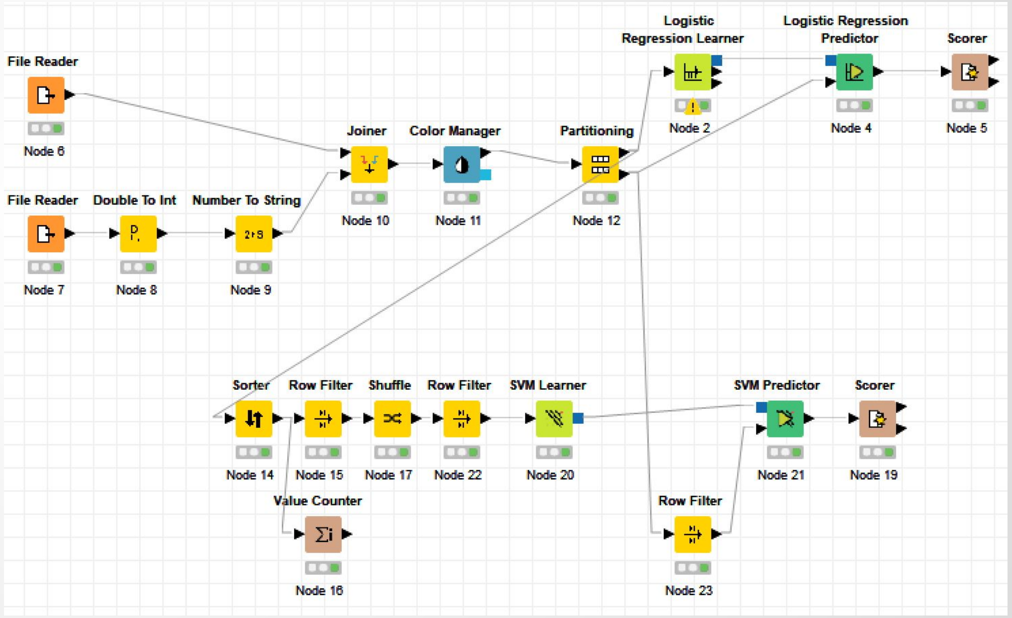
Row ID	0	1
0	4669	0
1	331	0





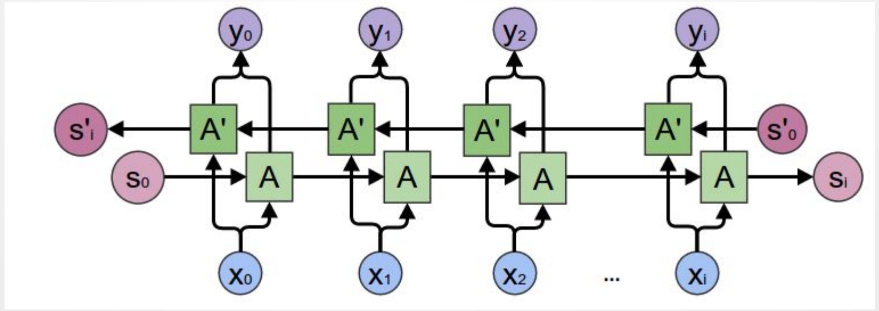
Data Modeling - Naive Approach

- As we saw, with the transformed dataset, the scores for Logistic Regression were not bad but a lot of false negative. Support Vector Machine just classified everything as True, even though it used a balanced dataset.
- After these trials we can say for sure that we need a better approach to solve this fairly complex problem.



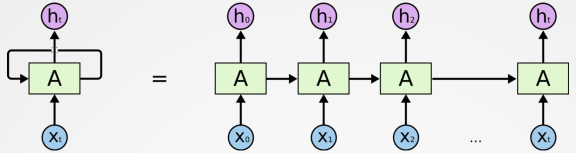
Data Modeling: Recurrent Neural Network/LSTM

- Taking inspiration from the latest research in the Natural Language Processing and Deep Learning field, we implemented a bi-directional Recurrent Neural Network model with LSTM (Long Short Term Memory) modules with Attention Layers.



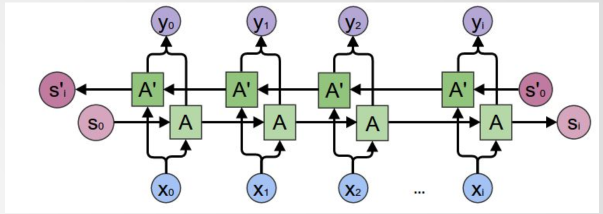
- This complex model has achieved state-of-the-art performance in other NLP tasks, and our idea was to attempt to emulate the acclaimed success in the Quora Challenge.

Data Modeling: Recurrent Neural Network/LSTM Motivation



- The Recurrent Neural Network has been applied across a gamut of sequence tasks, from time-series financial forecasting to –particularly relevant for the Challenge– Natural Language Processing. This is because at each time-step **t** the Recurrent Neural Network takes as input not only the current input **X_t** , but also the output computed at the previous node/time-step **A_t**.
- As alluded to above, this model is especially useful for the processing of textual data, since it not only predicts the target variable based on the input, but also on the relationship between the components of the input.
- Furthemore, the use of an LSTM guarantees that the RNN is able to utilize representations from time-steps far-away in the past when training/gradient-descent/backpropagation. (Essentially helps combat the "vanishing gradient" problem) Hochreiter & Schmidhuber's paper originally introduced the LSTM.
- Additionally, an Attention Layer, based on the work of Raffel & Ellis (https://arxiv.org/abs/1512.08756) is implemented in our model to further solve the long-term dependency problem mentioned above.

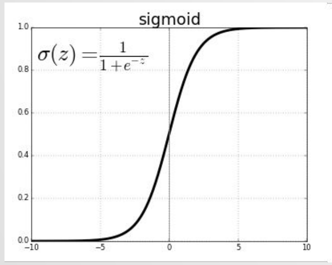
Data Modeling: Recurrent Neural Network/LSTM Motivation

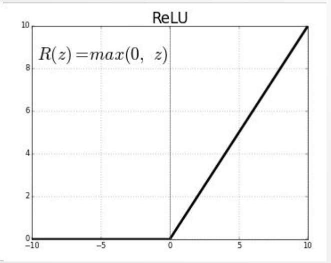


- The ***bi-directional*** nature of the implemented model is appropriate for this particular task since we want our RNN not only to learn representations from previous time-steps, but also to consider future time steps in the learning procedure.
- As a simple example, consider the question: “Why is the Italian football team terrible?” vs. “Why is the Italian capital Rome?” The first question would certainly be flagged as “Insincere” according to Quora’s guidelines, as it is attempting to make a statement. The second question on the other hand is certainly valid. However, a traditional RNN even with an LSTM implementation, would have trouble accurately labeling these questions, given the fact that both of them begin with the sequence of text “Why is the Italian...”
- In other words a traditional RNN/LSTM would neglect future word sequences, which encapsulate important semantic data for the Quora Challenge. A bi-directional network, on the other hand would propagate back instances of words seen further down the line, and would solve this aforementioned issue.

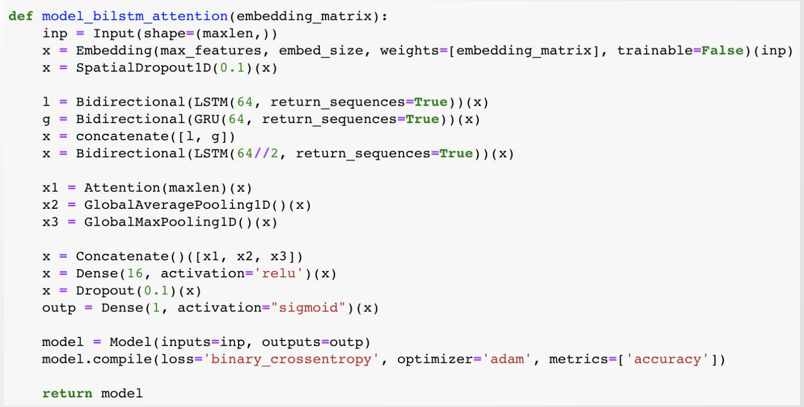
Data Modeling: Recurrent Neural Network/LSTM Implementation

- The final architecture implementation of our RNN model is carried out in Python using the Keras library, and can be seen below.
- The model is fed the fine-tuned Word Embedding Matrix for our Data, uses a ReLu activation function within the hidden layers, and a Sigmoid Function for the output.
- The optimization (training) algorithm utilized is Adam (an extension of Stochastic Gradient Descent).
- The Loss Function is the traditional Binary Cross Entropy, which simply ensures that our model maximizes the probability of our training data. (Seen in Prof. Tonini's lecture)
- Dropout is implemented in order to help prevent the overfitting of our model.





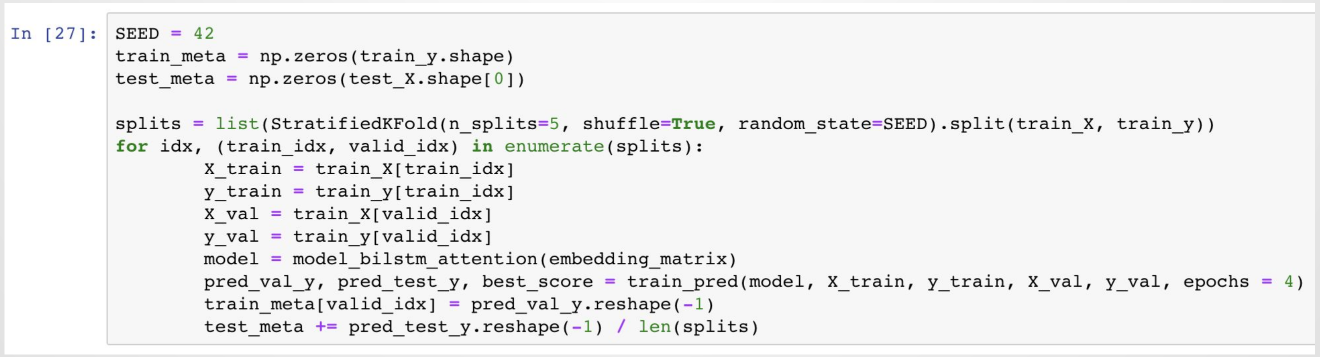
$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$



```python
def model_bilstm_attention(embedding_matrix):
    inp = Input(shape=(maxlen,))
    x = Embedding(max_features, embed_size, weights=[embedding_matrix], trainable=False)(inp)
    x = SpatialDropout1D(0.1)(x)

    l = Bidirectional(LSTM(64, return_sequences=True))(x)
    g = Bidirectional(GRU(64, return_sequences=True))(x)
    x = concatenate([l, g])
    x = Bidirectional(LSTM(64//2, return_sequences=True))(x)

    x1 = Attention(maxlen)(x)
    x2 = GlobalAveragePooling1D()(x)
    x3 = GlobalMaxPooling1D()(x)

    x = Concatenate()([x1, x2, x3])
    x = Dense(16, activation='relu')(x)
    x = Dropout(0.1)(x)
    outp = Dense(1, activation="sigmoid")(x)

    model = Model(inputs=inp, outputs=outp)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

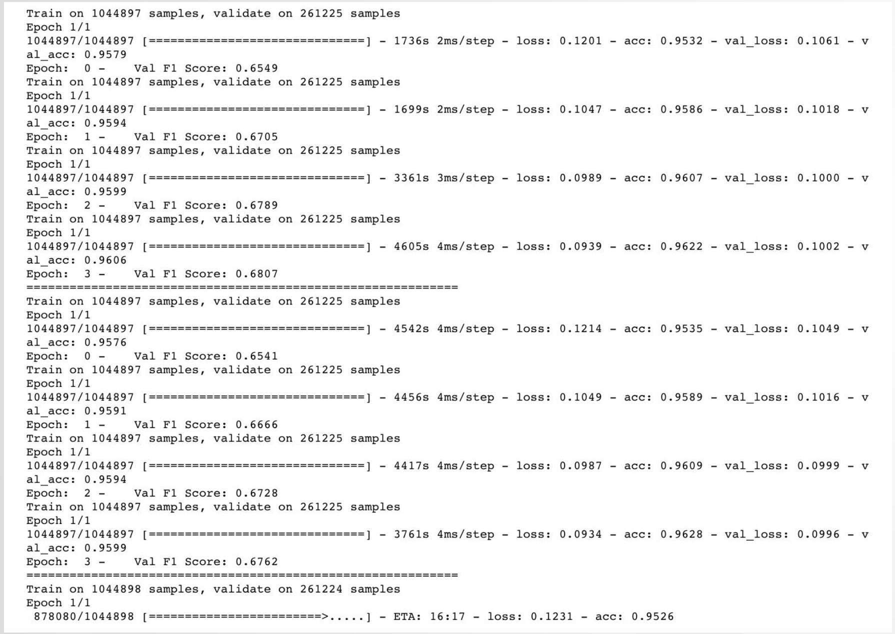
Data Modeling: Recurrent Neural Network/LSTM Implementation

- We perform a 5-Fold Cross-Validation assessment of our model, instead of a single train-test split. In other words, we:
    1. Shuffle the dataset randomly.
    2. Split the dataset into k groups
    3. For each unique group:
        a. Take the group as a hold out or test data set.
        b. Take the remaining groups as a training data set.
        c. Fit a model on the training set and evaluate it on the test set
        d. Retain the evaluation score and discard the model
    4. Finally summarize the skill of the model using the sample of model evaluation scores.



```
In [27]: SEED = 42
         train_meta = np.zeros(train_y.shape)
         test_meta = np.zeros(test_X.shape[0])

         splits = list(StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED).split(train_X, train_y))
         for idx, (train_idx, valid_idx) in enumerate(splits):
                 X_train = train_X[train_idx]
                 y_train = train_y[train_idx]
                 X_val = train_X[valid_idx]
                 y_val = train_y[valid_idx]
                 model = model_bilstm_attention(embedding_matrix)
                 pred_val_y, pred_test_y, best_score = train_pred(model, X_train, y_train, X_val, y_val, epochs = 4)
                 train_meta[valid_idx] = pred_val_y.reshape(-1)
                 test_meta += pred_test_y.reshape(-1) / len(splits)
```

Model Evaluation: Recurrent Neural Network/LSTM Results



```
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 1736s 2ms/step - loss: 0.1201 - acc: 0.9532 - val_loss: 0.1061 - v
al_acc: 0.9579
Epoch:  0 -    Val F1 Score: 0.6549
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 1699s 2ms/step - loss: 0.1047 - acc: 0.9586 - val_loss: 0.1018 - v
al_acc: 0.9594
Epoch:  1 -    Val F1 Score: 0.6705
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 3361s 3ms/step - loss: 0.0989 - acc: 0.9607 - val_loss: 0.1000 - v
al_acc: 0.9599
Epoch:  2 -    Val F1 Score: 0.6789
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 4605s 4ms/step - loss: 0.0939 - acc: 0.9622 - val_loss: 0.1002 - v
al_acc: 0.9606
Epoch:  3 -    Val F1 Score: 0.6807
============================================================
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 4542s 4ms/step - loss: 0.1214 - acc: 0.9535 - val_loss: 0.1049 - v
al_acc: 0.9576
Epoch:  0 -    Val F1 Score: 0.6541
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 4456s 4ms/step - loss: 0.1049 - acc: 0.9589 - val_loss: 0.1016 - v
al_acc: 0.9591
Epoch:  1 -    Val F1 Score: 0.6666
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 4417s 4ms/step - loss: 0.0987 - acc: 0.9609 - val_loss: 0.0999 - v
al_acc: 0.9594
Epoch:  2 -    Val F1 Score: 0.6728
Train on 1044897 samples, validate on 261225 samples
Epoch 1/1
1044897/1044897 [==============================] - 3761s 4ms/step - loss: 0.0934 - acc: 0.9628 - val_loss: 0.0996 - v
al_acc: 0.9599
Epoch:  3 -    Val F1 Score: 0.6762
============================================================
Train on 1044898 samples, validate on 261224 samples
Epoch 1/1
 878080/1044898 [========================>.....] - ETA: 16:17 - loss: 0.1231 - acc: 0.9526
```

- As we can see our model obtains an F1 Score of ~0.67 across the four different Epochs that is trained for, with an associated accuracy of ~96%.
- We can safely say that the additional complexity, relative to our previous modelling approaches, was certainly worth it. Since the model attains a much better F1 score, while slightly improving the overall accuracy.
- Given that Quora is ***not*** specifically interested in the inference, or interpretation, of what constitutes a "sincere" or "insincere" question, since the organization itself decides this a priori, we can recommend our "black box" approach as an ideal instrument to improve Quora's platform and business at large.

Managerial Implications

- As previously mentioned in the problem definition section, Quora was built as a platform where the free flow of knowledge could take place.
  The premises for this to happen rely on the sentiment of trust that arises among people, who know that their questions (which matter to them or are at least of interest) will be answered sincerely and competently. At the same time, those replying will be sure that their effort was worth, since their response will be of use for someone.
- However, insincere questions can erode the ideal relationship among members of the community, who can start to doubt on the legitimacy of the enquirers, deciding not to spend time on answering to sentences that have no ground, are often offensive and potentially shocking.
- This behavior might eventually generate a negative feedback, according to which, the less the responses, the fewer the questions and so on, until the platform shuts down, because nobody will see the point in using it any further.

Appendix Guide

- ❖ The computations, and technical implementations, of the algorithms referenced throughout the report can be found in the following set of complementary files:
  - ➢ **Quora_Insincere_Questions.ipynb:** Jupyter Notebook Python implementation of the Word Embedding Pre-Processing procedure mentioned in slides 16-20.
  - ➢ **Quora_Insincere_Questions_2.ipynb:** Jupyter Notebook Python (+ Keras) implementation and evaluation of the bi-linear Recurrent Neural Network/LSTM model mentioned in slides 27-32.
  - ➢ **Quora_Insincere_Questions_3.ipynb:** Jupyter Notebook Python (+Sci-Kit Learn) implementation and evaluation of the Linear Classifier and Support Vector Machine mentioned in slides 21-24.
  - ➢ **Quora_Insincere_Questions_Project.knwf:** KNIME Workflow containing implementation and evaluation of Logistic Regression and Support Vector Machine models mentioned in slides 25 & 26.
  - ➢ **Quora_Insincere_Questions_EDA.ipynb:** Jupyter Notebook Python implementation of the Exploratory Data Analysis of the Quora dataset as detailed in slides 13-15.