**Authors: Kovalev Evgeny, Glazov Vsevolod, Chesakov Daniil**

In this project we wanted to identify sarcasm in online comments. It is an interesting and non-trivial NLP task, with which we wanted to get acquainted in our project. In addition to building the classification model, we also wanted to analyze what distinguishes sacrasm from normal text.

The main questions of our project: **Can sarcasm can be identified with a good accuracy? What differs sarcasm from usual language?**

Dataset: [https://www.kaggle.com/danofer/sarcasm (https://www.kaggle.com/danofer/sarcasm)](https://www.kaggle.com/danofer/sarcasm)

This is the main dataset that we used. It contains ~1.3M comments from Reddit, and an important thing is that the data is balanced.

The metric used to measure model quality on validation data was AUC-ROC in order to work with probabilities of classes, not with just predictions of classes.

Repository with all the results: [https://github.com/blacKitten13/Sarcasm-Detection (https://github.com/blacKitten13/Sarcasm-Detection)](https://github.com/blacKitten13/Sarcasm-Detection)

Final notebook with all the results: [https://github.com/blacKitten13/Sarcasm-Detection/blob/master/final_notebook.ipynb (https://github.com/blacKitten13/Sarcasm-Detection/blob/master/final_notebook.ipynb)](https://github.com/blacKitten13/Sarcasm-Detection/blob/master/final_notebook.ipynb)

Link to all the files: [https://yadi.sk/d/21u3tcl5mhZwDA (https://yadi.sk/d/21u3tcl5mhZwDA)](https://yadi.sk/d/21u3tcl5mhZwDA)

Model results:

| Model | AUC-ROC |
|---:|---:|
| BiLSTM + GloVe | 0.809782 |
| BiLSTM + ELMo | 0.808948 |
| LR + TF-IDF | 0.794278 |
| BiLSTM + Word2Vec | 0.794272 |
| LR + CntVectorizer | 0.787733 |
| LR + Doc2Vec | 0.677191 |
| LR + GloVe | 0.674855 |
| LR + Word2Vec | 0.673213 |

In [ ]:
```
# glove embeddings
!wget "http://nlp.stanford.edu/data/glove.6B.zip"
!unzip -j "glove.6B.zip" "glove.6B.300d.txt"
# word2vec embeddings
!wget -c "https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"
!gunzip GoogleNews-vectors-negative300.bin.gz
# bilstm + glove checkpoint
!wget "https://s189vla.storage.yandex.net/rdisk/423031d98d5e8152dbcee5a0066531cb2e9848a3b59455273f5458195879b9f3/5dfbef1e/ehujOxZo5-meNvCV7YBHPzq-ozMAW_bR5rElcV72PW8OnxF1I7JYZF-MSeJ_eJesgUImKmxuwStl_AG-a4zsew==?uid=0&filename=bilstm_glove.pt&disposition=attachment&hash=iGBn3Lkkam%2BCpH7WwL26JS0d0KnJOcPzobP4Ue417oa/O25HJC9wZrP5ASIchRJkq/J6bpmRyOJonT3VoXnDag%3D%3D%3A/bilstm_glove.pt&limit=0&content_type=application%2Foctet-stream&owner_uid=34782215&fsize=190609017&hid=104aaf7c48f1630550f7ff1709d621ca&media_type=data&tknv=v2&rtoken=u5uVDKLaqBl6&force_default=no&ycrid=na-4034740f5f4bd5adbca3ad00b43f32e1-downloader6h&ts=59a1575638380&s=4fb7f401186a613a45ef6f41d05addd26e70d39c531d816d32f6bc56ee2af450&pb=U2FsdGVkX19uYXwSeP1jIhcToAw0bji4o5J8XlnEd9rl4fvCc-eEd1CoFnIMXr0uiSBLQPd1SRoLQWmGXoFcIhoadSumw9gJ9nXIfi_SFEo" -O "bilstm_glove.pt"
# bilstm + elmo checkpoint
!wget "https://s579sas.storage.yandex.net/rdisk/2aedcdb98a5d527613a85fcb36d2fa6606a5445b6a3c68d15c454130fc4f04f7/5dfc0611/ehujOxZo5-meNvCV7YBHPzP2kD72dIvEUwSMZrkE8zGVd3CkaNQERYGkSOjepMqoRI_OK-NuJnPYlKeaOOy70Q==?uid=0&filename=bilstm_elmo.pt&disposition=attachment&hash=iGBn3Lkkam%2BCpH7WwL26JS0d0KnJOcPzobP4Ue417oa/O25HJC9wZrP5ASIchRJkq/J6bpmRyOJonT3VoXnDag%3D%3D%3A/bilstm_elmo.pt&limit=0&content_type=application%2Foctet-stream&owner_uid=34782215&fsize=55045100&hid=403aa2c332e438282bb2ae792a025ffd&media_type=data&tknv=v2&rtoken=fDV2ojd2lKJE&force_default=no&ycrid=na-bc324cac3ed0d512e91f826924fbdea1-downloader7f&ts=59a16d390e640&s=378ad72de31e17ff49c4e82058b15b8faf266e10da3bff5a1e68d03969675bee&pb=U2FsdGVkX1_Lk2I3TytkFfyaKTgD3Vl-pGF-7rdqjkYMi387HaquT_TpRYNT7AsQcyyVcTME-zVFeOVnFt0JZtk66vun7z8gOFurXQwA5Ho" -O "bilstm_elmo.pt"
# bilstm + w2v checkpoint
!wget "https://s223vla.storage.yandex.net/rdisk/aba8603898d1a430b7e1857ff7225f0f528c894694be6395b0ac15777e773991/5dfc0737/ehujOxZo5-meNvCV7YBHP2RGiJnbECMrX77hktXs-zYCfCdW57p81Bo2T3v1IOJte7FMDSVkkb4uTXzdFeF0OQ==?uid=0&filename=bilstm_w2v.pt&disposition=attachment&hash=iGBn3Lkkam%2BCpH7WwL26JS0d0KnJOcPzobP4Ue417oa/O25HJC9wZrP5ASIchRJkq/J6bpmRyOJonT3VoXnDag%3D%3D%3A/bilstm_w2v.pt&limit=0&content_type=application%2Foctet-stream&owner_uid=34782215&fsize=151462615&hid=d74a31e6722f4610a85205ddb0535367&media_type=data&tknv=v2&rtoken=FLDTZK5pab2q&force_default=no&ycrid=na-8d7b3bbc029e1c399208933bd207b643-downloader20e&ts=59a16e516fbc0&s=016837589c887b27465c2c561637d44554ab1f2e2af19c4833feb5bafa8659b7&pb=U2FsdGVkX19UPKaSd7urigTc_dYHoyHkJNs74LdVbhJBl-6f7zkpk0hYAvldePOOfI3jPTdPLVHioUzer_egro5zL5yfeWks2Cq6kjajTag" -O "bilstm_w2v.pt"
# elmo files
!wget "https://s120vla.storage.yandex.net/rdisk/e6e111f144116c5cfa807e0a85f6bd86ea2a1c37d3d1ed04e069b9e92291f0b5/5dfc084d/ehujOxZo5-meNvCV7YBHP4ItFhUmJ7vPryQPBvY5mc5HzskD8P4BjHMEiJIpZ18kwsiRxhkzXLuK6AfaheLUAA==?uid=0&filename=elmo_2x1024_128_2048cnn_1xhighway_weights.hdf5&disposition=attachment&hash=iGBn3Lkkam%2BCpH7WwL26JS0d0KnJOcPzobP4Ue417oa/O25HJC9wZrP5ASIchRJkq/J6bpmRyOJonT3VoXnDag%3D%3D%3A/elmo_2x1024_128_2048cnn_1xhighway_weights.hdf5&limit=0&content_type=application%2Fx-hdf&owner_uid=34782215&fsize=54402456&hid=0287794cb448c0ed4f6e16cc64e58613&media_type=data&tknv=v2&rtoken=FEt2YM9BJRh4&force_default=no&ycrid=na-5b7405c9ced34f67e046ece5ac5ee405-downloader24h&ts=59a16f5a8ed40&s=114001520cf251585d5dc6bc650a77c09ee1b08d86f6224f614cc60bed9242dd&pb=U2FsdGVkX1_6RQhPCAVMmj62iUepUOBRqcEAepj_swxrSNUmIMJdG1dU4bInyukh
```

```
K4RN7B39SRKl0ehGy3tE0WVcynjQebatDQQ88RHIftA" -O "elmo_2x1024_128_2048cnn_1xhighway_weights.hdf5"
!wget "https://s69vla.storage.yandex.net/rdisk/3f68c56d540daf7b199b9907652850349216dddf51b0ebc33fb1a4776e95f2c0/5dfc0889/ehujOxZo5-meNvCV7YBHP4Su1Hbt0IMCey5hebYCJDsp1vYNtZWmrcdKUQicr8M0lyGtw29fZ-88-c964N0Xig==?uid=0&filename=elmo_2x1024_128_2048cnn_1xhighway_options.json&disposition=attachment&hash=iGBn3Lkkam%2BCpH7WwL26JS0d0KnJOcPzobP4Ue417oa/O25HJC9wZrP5ASIchRJkq/J6bpmRyOJonT3VoXnDag%3D%3D%3A/elmo_2x1024_128_2048cnn_1xhighway_options.json&limit=0&content_type=text%2Fplain&owner_uid=34782215&fsize=336&hid=1cd0bcd50b0e106adeadcb934f51d4c9&media_type=text&tknv=v2&rtoken=eAdnvFdgRwPW&force_default=no&ycrid=na-2e2fce506fd1459222132ee0459a354d-downloader24h&ts=59a16f93c7440&s=05edcf692895ce801449b587337974360d68d214a08baf2033de86548ad27b5c&pb=U2FsdGVkX19beWPBv2f3hrS1sgJWWIH2WydD8irIYd2MpVFPIWQnSXhVHtubJzO0KSI3Vo2D7sPZ6avAghyJgiDlUl8m8C4jK_asH5W_Aq4" -O "elmo_2x1024_128_2048cnn_1xhighway_options.json"
# dataset
!wget "https://s198myt.storage.yandex.net/rdisk/494150156dfefd6ff2d1d133e6a6214d388fb0fd7e00c933def83df9360fee7a/5dfc092d/9ku0RfoqgvhHN2N0OIFZz-7zg_TkzwYFtqtmdq3gmtsOudcv4x-7wSgWoTbgZQROoNGa30r11_U0IRbqh2_JeA==?uid=0&filename=train-balanced-sarcasm.csv.zip&disposition=attachment&hash=iGBn3Lkkam%2BCpH7WwL26JS0d0KnJOcPzobP4Ue417oa%2FO25HJC9wZrP5ASIchRJkq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2Ftrain-balanced-sarcasm.csv.zip&limit=0&content_type=application%2Fzip&owner_uid=34782215&fsize=110984528&hid=4312f30be0b4111dc79ee4600936bfb1&media_type=compressed&tknv=v2&rtoken=e9JgAcJ0HuBr&force_default=no&ycrid=na-77dc8f072ca22f34baee00c8a7fe6e69-downloader22h&ts=59a170302e540&s=e61a98856f91d092fa6da9d52c8196548e5752dbd4ce16ad991500cad842c531&pb=U2FsdGVkX1_6rKgWWXAbIaxmMKKhES9YmEZFvSY3eBqHGo-Mue4O1YEClJiD1JJ3IXBBp8PlWwBHJC6NY4j6GEao0f8B1MHL_NlkotgDjAQ" -O "train-balanced-sarcasm.csv.zip"
!unzip "train-balanced-sarcasm.csv.zip"
```

In [104]:
```
# !conda install -c pytorch pytorch
!pip install allennlp
!pip install gensim
!pip install lime
!pip install seaborn
!pip install wordcloud
```

```
Requirement already satisfied: allennlp in c:\users\asus\anaconda3\lib\site-packages (0.9.0)
Requirement already satisfied: pytorch-pretrained-bert>=0.6.0 in c:\users\asus\anaconda3\lib\site-packages (from alle
nnlp) (0.6.2)
Requirement already satisfied: scikit-learn in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (0.21.2)
Requirement already satisfied: jsonpickle in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.2)
Requirement already satisfied: spacy<2.2,>=2.1.0 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (2.1.9)
Requirement already satisfied: responses>=0.7 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (0.10.8)
Requirement already satisfied: unidecode in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.1.1)
Requirement already satisfied: editdistance in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (0.5.3)
Requirement already satisfied: tensorboardX>=1.2 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.9)
Requirement already satisfied: flask-cors>=3.0.7 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (3.0.8)
Requirement already satisfied: h5py in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (2.9.0)
Requirement already satisfied: numpydoc>=0.8.0 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (0.9.1)
Requirement already satisfied: torch>=1.2.0 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.3.0)
Requirement already satisfied: tqdm>=4.19 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (4.32.1)
Requirement already satisfied: flaky in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (3.6.1)
Requirement already satisfied: sqlparse>=0.2.4 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (0.3.0)
Requirement already satisfied: flask>=1.0.2 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.1.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (2019.1)
Requirement already satisfied: overrides in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (2.6)
Requirement already satisfied: conllu==1.3.1 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.3.1)
Requirement already satisfied: requests>=2.18 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (2.22.0)
Requirement already satisfied: pytest in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (5.0.1)
Requirement already satisfied: pytorch-transformers==1.1.0 in c:\users\asus\anaconda3\lib\site-packages (from allennl
p) (1.1.0)
Requirement already satisfied: numpy in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.16.4)
Requirement already satisfied: nltk in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (3.4.4)
Requirement already satisfied: scipy in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.2.1)
Requirement already satisfied: word2number>=1.1 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.1)
Requirement already satisfied: parsimonious>=0.8.0 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (0.8.
1)
Requirement already satisfied: gevent>=1.3.6 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.4.0)
Requirement already satisfied: ftfy in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (5.6)
Requirement already satisfied: boto3 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (1.10.42)
Requirement already satisfied: matplotlib>=2.2.3 in c:\users\asus\anaconda3\lib\site-packages (from allennlp) (3.1.0)
Requirement already satisfied: regex in c:\users\asus\anaconda3\lib\site-packages (from pytorch-pretrained-bert>=0.6.
0->allennlp) (2019.12.19)
Requirement already satisfied: joblib>=0.11 in c:\users\asus\anaconda3\lib\site-packages (from scikit-learn->allennl
p) (0.13.2)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.
2,>=2.1.0->allennlp) (1.0.2)
```

```
Requirement already satisfied: wasabi<1.1.0,>=0.2.0 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=2.
1.0->allennlp) (0.4.2)
Requirement already satisfied: plac<1.0.0,>=0.9.6 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=2.1.
0->allennlp) (0.9.6)
Requirement already satisfied: preshed<2.1.0,>=2.0.1 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=
2.1.0->allennlp) (2.0.1)
Requirement already satisfied: blis<0.3.0,>=0.2.2 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=2.1.
0->allennlp) (0.2.4)
Requirement already satisfied: thinc<7.1.0,>=7.0.8 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=2.
1.0->allennlp) (7.0.8)
Requirement already satisfied: srsly<1.1.0,>=0.0.6 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=2.
1.0->allennlp) (0.2.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\asus\anaconda3\lib\site-packages (from spacy<2.2,>=2.
1.0->allennlp) (2.0.3)
Requirement already satisfied: six in c:\users\asus\anaconda3\lib\site-packages (from responses>=0.7->allennlp) (1.1
2.0)
Requirement already satisfied: protobuf>=3.8.0 in c:\users\asus\anaconda3\lib\site-packages (from tensorboardX>=1.2->
allennlp) (3.11.1)
Requirement already satisfied: sphinx>=1.6.5 in c:\users\asus\anaconda3\lib\site-packages (from numpydoc>=0.8.0->alle
nnlp) (2.1.2)
Requirement already satisfied: Jinja2>=2.3 in c:\users\asus\anaconda3\lib\site-packages (from numpydoc>=0.8.0->allenn
lp) (2.10.1)
Requirement already satisfied: click>=5.1 in c:\users\asus\anaconda3\lib\site-packages (from flask>=1.0.2->allennlp)
(7.0)
Requirement already satisfied: Werkzeug>=0.15 in c:\users\asus\anaconda3\lib\site-packages (from flask>=1.0.2->allenn
lp) (0.15.4)
Requirement already satisfied: itsdangerous>=0.24 in c:\users\asus\anaconda3\lib\site-packages (from flask>=1.0.2->al
lennlp) (1.1.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\asus\anaconda3\lib\site-packages
(from requests>=2.18->allennlp) (1.24.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\users\asus\anaconda3\lib\site-packages (from requests>=2.1
8->allennlp) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in c:\users\asus\anaconda3\lib\site-packages (from requests>=2.18->alle
nnlp) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\anaconda3\lib\site-packages (from requests>=2.18->
allennlp) (2019.6.16)
Requirement already satisfied: py>=1.5.0 in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (1.8.0)
Requirement already satisfied: packaging in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (19.0)
Requirement already satisfied: attrs>=17.4.0 in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (1
9.1.0)
Requirement already satisfied: more-itertools>=4.0.0 in c:\users\asus\anaconda3\lib\site-packages (from pytest->allen
nlp) (7.0.0)
```

```
Requirement already satisfied: atomicwrites>=1.0 in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (1.3.0)
Requirement already satisfied: pluggy<1.0,>=0.12 in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (0.12.0)
Requirement already satisfied: importlib-metadata>=0.12 in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (0.17)
Requirement already satisfied: wcwidth in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (0.1.7)
Requirement already satisfied: colorama in c:\users\asus\anaconda3\lib\site-packages (from pytest->allennlp) (0.4.1)
Requirement already satisfied: sentencepiece in c:\users\asus\anaconda3\lib\site-packages (from pytorch-transformers==1.1.0->allennlp) (0.1.85)
Requirement already satisfied: greenlet>=0.4.14 in c:\users\asus\anaconda3\lib\site-packages (from gevent>=1.3.6->allennlp) (0.4.15)
Requirement already satisfied: cffi>=1.11.5 in c:\users\asus\anaconda3\lib\site-packages (from gevent>=1.3.6->allennlp) (1.12.3)
Requirement already satisfied: botocore<1.14.0,>=1.13.42 in c:\users\asus\anaconda3\lib\site-packages (from boto3->allennlp) (1.13.42)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in c:\users\asus\anaconda3\lib\site-packages (from boto3->allennlp) (0.9.4)
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in c:\users\asus\anaconda3\lib\site-packages (from boto3->allennlp) (0.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2.3->allennlp) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2.3->allennlp) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2.3->allennlp) (2.4.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2.3->allennlp) (2.8.0)
Requirement already satisfied: setuptools in c:\users\asus\anaconda3\lib\site-packages (from protobuf>=3.8.0->tensorboardX>=1.2->allennlp) (41.0.1)
Requirement already satisfied: sphinxcontrib-jsmath in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=0.8.0->allennlp) (1.0.1)
Requirement already satisfied: alabaster<0.8,>=0.7 in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=0.8.0->allennlp) (0.7.12)
Requirement already satisfied: sphinxcontrib-applehelp in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=0.8.0->allennlp) (1.0.1)
Requirement already satisfied: sphinxcontrib-devhelp in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=0.8.0->allennlp) (1.0.1)
Requirement already satisfied: babel!=2.0,>=1.3 in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=0.8.0->allennlp) (2.7.0)
Requirement already satisfied: Pygments>=2.0 in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=0.8.0->allennlp) (2.4.2)
```

```
Requirement already satisfied: sphinxcontrib-htmlhelp in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.
5->numpydoc>=0.8.0->allennlp) (1.0.2)
Requirement already satisfied: sphinxcontrib-serializinghtml in c:\users\asus\anaconda3\lib\site-packages (from sphin
x>=1.6.5->numpydoc>=0.8.0->allennlp) (1.1.3)
Requirement already satisfied: docutils>=0.12 in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpy
doc>=0.8.0->allennlp) (0.14)
Requirement already satisfied: imagesize in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5->numpydoc>=
0.8.0->allennlp) (1.1.0)
Requirement already satisfied: sphinxcontrib-qthelp in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5-
>numpydoc>=0.8.0->allennlp) (1.0.2)
Requirement already satisfied: snowballstemmer>=1.1 in c:\users\asus\anaconda3\lib\site-packages (from sphinx>=1.6.5-
>numpydoc>=0.8.0->allennlp) (1.9.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\asus\anaconda3\lib\site-packages (from Jinja2>=2.3->numpy
doc>=0.8.0->allennlp) (1.1.1)
Requirement already satisfied: zipp>=0.5 in c:\users\asus\anaconda3\lib\site-packages (from importlib-metadata>=0.12-
>pytest->allennlp) (0.5.1)
Requirement already satisfied: pycparser in c:\users\asus\anaconda3\lib\site-packages (from cffi>=1.11.5->gevent>=1.
3.6->allennlp) (2.19)
Requirement already satisfied: gensim in c:\users\asus\anaconda3\lib\site-packages (3.8.1)
Requirement already satisfied: numpy>=1.11.3 in c:\users\asus\anaconda3\lib\site-packages (from gensim) (1.16.4)
Requirement already satisfied: six>=1.5.0 in c:\users\asus\anaconda3\lib\site-packages (from gensim) (1.12.0)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\asus\anaconda3\lib\site-packages (from gensim) (1.9.0)
Requirement already satisfied: scipy>=0.18.1 in c:\users\asus\anaconda3\lib\site-packages (from gensim) (1.2.1)
Requirement already satisfied: boto>=2.32 in c:\users\asus\anaconda3\lib\site-packages (from smart-open>=1.8.1->gensi
m) (2.49.0)
Requirement already satisfied: boto3 in c:\users\asus\anaconda3\lib\site-packages (from smart-open>=1.8.1->gensim)
(1.10.42)
Requirement already satisfied: requests in c:\users\asus\anaconda3\lib\site-packages (from smart-open>=1.8.1->gensim)
(2.22.0)
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in c:\users\asus\anaconda3\lib\site-packages (from boto3->sma
rt-open>=1.8.1->gensim) (0.2.1)
Requirement already satisfied: botocore<1.14.0,>=1.13.42 in c:\users\asus\anaconda3\lib\site-packages (from boto3->sm
art-open>=1.8.1->gensim) (1.13.42)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in c:\users\asus\anaconda3\lib\site-packages (from boto3->smart
-open>=1.8.1->gensim) (0.9.4)
Requirement already satisfied: idna<2.9,>=2.5 in c:\users\asus\anaconda3\lib\site-packages (from requests->smart-open
>=1.8.1->gensim) (2.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\asus\anaconda3\lib\site-packages
(from requests->smart-open>=1.8.1->gensim) (1.24.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\asus\anaconda3\lib\site-packages (from requests->smart-
open>=1.8.1->gensim) (2019.6.16)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\users\asus\anaconda3\lib\site-packages (from requests->sma
```

```
rt-open>=1.8.1->gensim) (3.0.4)
Requirement already satisfied: docutils<0.16,>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from botocore<1.14.
0,>=1.13.42->boto3->smart-open>=1.8.1->gensim) (0.14)
Requirement already satisfied: python-dateutil<2.8.1,>=2.1; python_version >= "2.7" in c:\users\asus\anaconda3\lib\si
te-packages (from botocore<1.14.0,>=1.13.42->boto3->smart-open>=1.8.1->gensim) (2.8.0)
Collecting lime
  Downloading https://files.pythonhosted.org/packages/e5/72/4be533df5151fcb48942515e95e88281ec439396c48d67d3ae41f2758
6f0/lime-0.1.1.36.tar.gz (275kB)
Requirement already satisfied: numpy in c:\users\asus\anaconda3\lib\site-packages (from lime) (1.16.4)
Requirement already satisfied: scipy in c:\users\asus\anaconda3\lib\site-packages (from lime) (1.2.1)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\asus\anaconda3\lib\site-packages (from lime) (0.21.2)
Requirement already satisfied: matplotlib in c:\users\asus\anaconda3\lib\site-packages (from lime) (3.1.0)
Requirement already satisfied: scikit-image>=0.12 in c:\users\asus\anaconda3\lib\site-packages (from lime) (0.15.0)
Requirement already satisfied: joblib>=0.11 in c:\users\asus\anaconda3\lib\site-packages (from scikit-learn>=0.18->li
me) (0.13.2)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib->lime) (0.1
0.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib->lime)
(1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\asus\anaconda3\lib\site-packages
(from matplotlib->lime) (2.4.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib->li
me) (2.8.0)
Requirement already satisfied: networkx>=2.0 in c:\users\asus\anaconda3\lib\site-packages (from scikit-image>=0.12->l
ime) (2.3)
Requirement already satisfied: imageio>=2.0.1 in c:\users\asus\anaconda3\lib\site-packages (from scikit-image>=0.12->
lime) (2.5.0)
Requirement already satisfied: pillow>=4.3.0 in c:\users\asus\appdata\roaming\python\python37\site-packages (from sci
kit-image>=0.12->lime) (6.2.1)
Requirement already satisfied: PyWavelets>=0.4.0 in c:\users\asus\anaconda3\lib\site-packages (from scikit-image>=0.1
2->lime) (1.0.3)
Requirement already satisfied: six in c:\users\asus\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib->lime)
(1.12.0)
Requirement already satisfied: setuptools in c:\users\asus\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matpl
otlib->lime) (41.0.1)
Requirement already satisfied: decorator>=4.3.0 in c:\users\asus\anaconda3\lib\site-packages (from networkx>=2.0->sci
kit-image>=0.12->lime) (4.4.0)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py): started
  Building wheel for lime (setup.py): finished with status 'done'
  Created wheel for lime: filename=lime-0.1.1.36-cp37-none-any.whl size=284196 sha256=464fc34b74d7bb47624235b3e2f7282
949a136d80e92b863e5fcea4224786e62
```

```
  Stored in directory: C:\Users\ASUS\AppData\Local\pip\Cache\wheels\a9\2f\25\4b2127822af5761dab9a27be52e175105772aebb
cbc484fb95
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.1.1.36
Requirement already satisfied: seaborn in c:\users\asus\anaconda3\lib\site-packages (0.9.0)
Requirement already satisfied: scipy>=0.14.0 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (1.2.1)
Requirement already satisfied: matplotlib>=1.4.3 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (3.1.0)
Requirement already satisfied: pandas>=0.15.2 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (0.24.2)
Requirement already satisfied: numpy>=1.9.3 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (1.16.4)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=1.4.3->sea
born) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=1.4.3
->seaborn) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\asus\anaconda3\lib\site-packages
(from matplotlib>=1.4.3->seaborn) (2.4.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=1.
4.3->seaborn) (2.8.0)
Requirement already satisfied: pytz>=2011k in c:\users\asus\anaconda3\lib\site-packages (from pandas>=0.15.2->seabor
n) (2019.1)
Requirement already satisfied: six in c:\users\asus\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib>=1.4.3
->seaborn) (1.12.0)
Requirement already satisfied: setuptools in c:\users\asus\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matpl
otlib>=1.4.3->seaborn) (41.0.1)
Requirement already satisfied: wordcloud in c:\users\asus\anaconda3\lib\site-packages (1.6.0)
Requirement already satisfied: matplotlib in c:\users\asus\anaconda3\lib\site-packages (from wordcloud) (3.1.0)
Requirement already satisfied: pillow in c:\users\asus\appdata\roaming\python\python37\site-packages (from wordcloud)
(6.2.1)
Requirement already satisfied: numpy>=1.6.1 in c:\users\asus\anaconda3\lib\site-packages (from wordcloud) (1.16.4)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib->wordcloud)
(0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib->wordc
loud) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\asus\anaconda3\lib\site-packages
(from matplotlib->wordcloud) (2.4.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib->wo
rdcloud) (2.8.0)
Requirement already satisfied: six in c:\users\asus\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib->wordc
loud) (1.12.0)
Requirement already satisfied: setuptools in c:\users\asus\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matpl
otlib->wordcloud) (41.0.1)
```

In [105]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import random
import re
import seaborn as sns
import torch
import torch.nn as nn
import torch.optim as optim
from allennlp.modules.elmo import Elmo, batch_to_ids
from collections import OrderedDict
from gensim.models import KeyedVectors
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from lime.lime_text import LimeTextExplainer
from IPython import display
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, auc, confusion_matrix, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from wordcloud import STOPWORDS

%matplotlib inline

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

# Data Analysis

In [3]:
```
TRAIN_FILE = 'train-balanced-sarcasm.csv'
```

In [4]:
```
train_df = pd.read_csv(TRAIN_FILE)
train_df.head()
```

Out[4]:

| | label | comment | author | subreddit | score | ups | downs | date | created_utc | parent_comment |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | NC and NH. | Trumpbart | politics | 2 | -1 | -1 | 2016-10 | 2016-10-16 23:55:23 | Yeah, I get that argument. At this point, I'd ... |
| **1** | 0 | You do know west teams play against west teams... | Shbshb906 | nba | -4 | -1 | -1 | 2016-11 | 2016-11-01 00:24:10 | The blazers and Mavericks (The wests 5 and 6 s... |
| **2** | 0 | They were underdogs earlier today, but since G... | Creepeth | nfl | 3 | 3 | 0 | 2016-09 | 2016-09-22 21:45:37 | They're favored to win. |
| **3** | 0 | This meme isn't funny none of the "new york ni... | icebrotha | BlackPeopleTwitter | -8 | -1 | -1 | 2016-10 | 2016-10-18 21:03:47 | deadass don't kill my buzz |
| **4** | 0 | I could use one of those tools. | cush2push | MaddenUltimateTeam | 6 | -1 | -1 | 2016-12 | 2016-12-30 17:00:13 | Yep can confirm I saw the tool they use for th... |

In [5]:
```
train_df.shape
```

Out[5]:
```
(1010826, 10)
```

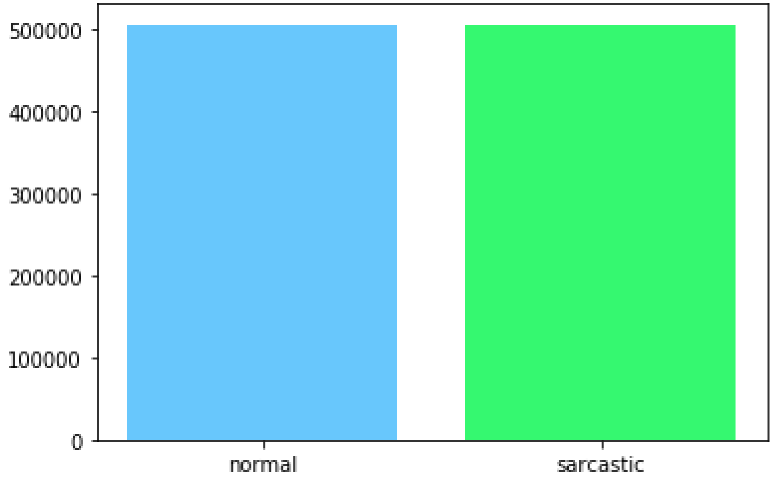
In [6]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1010826 entries, 0 to 1010825
Data columns (total 10 columns):
label             1010826 non-null int64
comment           1010773 non-null object
author            1010826 non-null object
subreddit         1010826 non-null object
score             1010826 non-null int64
ups               1010826 non-null int64
downs             1010826 non-null int64
date              1010826 non-null object
created_utc       1010826 non-null object
parent_comment    1010826 non-null object
dtypes: int64(4), object(6)
memory usage: 77.1+ MB
```

In [7]:
```
train_df.dropna(subset=['comment'], inplace=True)
```

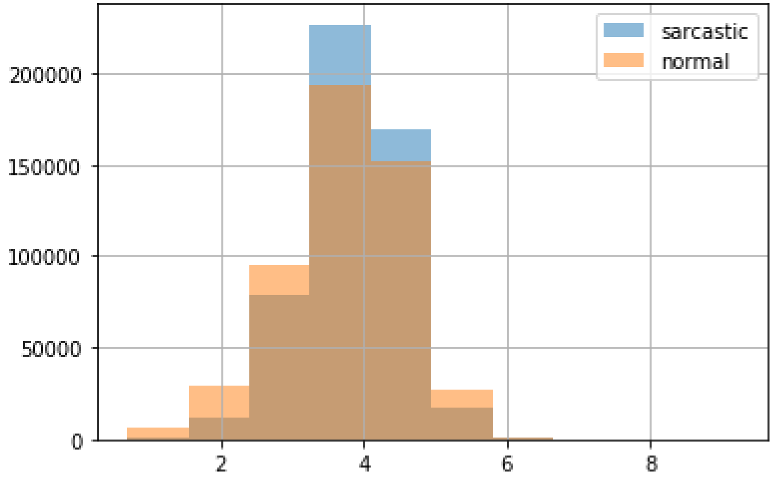
***Distribution of the target value***

In [91]:
```
color_rectangle = np.random.rand(2, 3)
plt.bar([0,1], train_df['label'].value_counts().values, color=color_rectangle)
plt.xticks([0,1], ['normal', 'sarcastic']);
```




***Distribution of lengths for sarcastic and normal comments is almost the same.***

In [92]:
```
train_df.loc[train_df['label'] == 1, 'comment'].str.len().apply(np.log1p).hist(label='sarcastic', alpha=.5)
train_df.loc[train_df['label'] == 0, 'comment'].str.len().apply(np.log1p).hist(label='normal', alpha=.5)
plt.legend();
```




***Let's see which subreddits are the most sarcastic ones***

In [93]:
```
sub_df = train_df.groupby('subreddit')['label'].agg([np.size, np.mean, np.sum])
sub_df[sub_df['size'] > 1000].sort_values(by='mean', ascending=False).head(10)
```

Out[93]:

| subreddit | size | mean | sum |
|---|---|---|---|
| creepyPMs | 5466 | 0.784303 | 4287 |
| MensRights | 3355 | 0.680775 | 2284 |
| ShitRedditSays | 1284 | 0.661994 | 850 |
| worldnews | 26376 | 0.642516 | 16947 |
| Libertarian | 2562 | 0.640125 | 1640 |
| atheism | 7377 | 0.639555 | 4718 |
| Conservative | 1881 | 0.639553 | 1203 |
| TwoXChromosomes | 1560 | 0.632692 | 987 |
| fatlogic | 2356 | 0.623090 | 1468 |
| facepalm | 1268 | 0.617508 | 783 |

***And the most sarcastic authors***

In [94]:
```
sub_df = train_df.groupby('author')['label'].agg([np.size, np.mean, np.sum])
sub_df[sub_df['size'] > 300].sort_values(by='mean', ascending=False).head(10)
```

Out[94]:

| | size | mean | sum |
|---|---|---|---|
| **author** | | | |
| **NeonDisease** | 422 | 0.500000 | 211 |
| **ShyBiDude89** | 404 | 0.500000 | 202 |
| **ivsciguy** | 342 | 0.500000 | 171 |
| **mad-n-fla** | 318 | 0.500000 | 159 |
| **mindlessrabble** | 302 | 0.500000 | 151 |
| **pokemon_fetish** | 432 | 0.500000 | 216 |
| **Biffingston** | 845 | 0.499408 | 422 |

### *Lets look at the most (frequent) "sarcastic" and "normal" words*

In [95]:
```
def preprocessing(texts):
    return [re.sub(r"([^ \w])", r" \1 ", str.lower(text)) for text in texts]

def smart_tokenization(texts):
    return [x for text in texts for x in text.split() if x not in STOPWORDS]
```

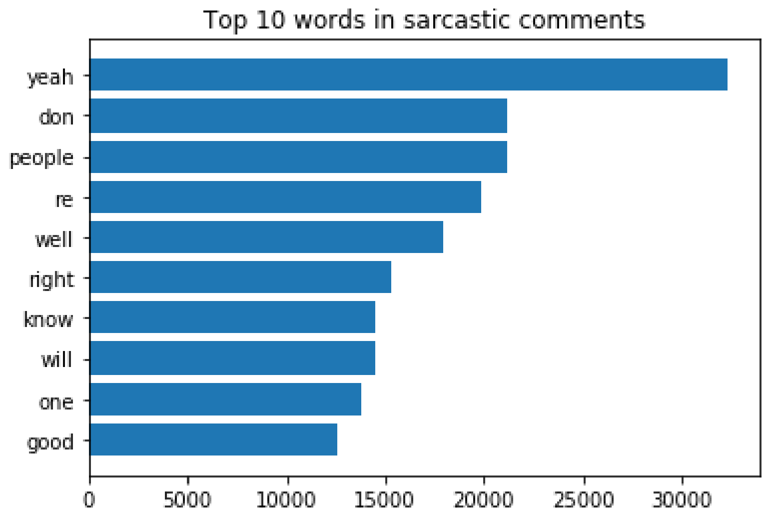
In [98]:
```
sarcastic_tokens = smart_tokenization(preprocessing(train_df.loc[train_df['label'] == 1, 'comment'].values))
normal_tokens = smart_tokenization(preprocessing(train_df.loc[train_df['label'] == 0, 'comment'].values))
```

In [99]:
```
def build_freq_vocab(tokens):
    freq_dict = {}
    for token in tokens:
        if (len(token) > 1):
            try:
                freq_dict[token] += 1
            except:
                freq_dict[token] = 1
    return freq_dict
```

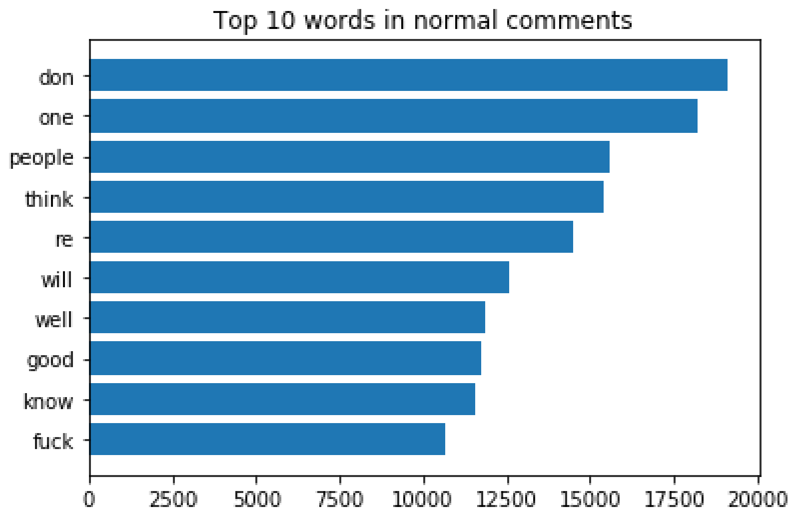
In [100]:
```
sarcastic_dict = build_freq_vocab(sarcastic_tokens)
normal_dict = build_freq_vocab(normal_tokens)
```

In [101]:
```
top_sarcastic = sorted(sarcastic_dict.items(), key=lambda x: x[1], reverse=True)[:10]
top_normal = sorted(normal_dict.items(), key=lambda x: x[1], reverse=True)[:10]
```

In [102]:
```
plt.barh([(10-i) for i in range(10)], [i[1] for i in top_sarcastic])
plt.yticks([(10-i) for i in range(10)], [i[0] for i in top_sarcastic]);
plt.title("Top 10 words in sarcastic comments");
```



In [103]:
```
plt.barh([(10-i) for i in range(10)], [i[1] for i in top_normal])
plt.yticks([(10-i) for i in range(10)], [i[0] for i in top_normal]);
plt.title("Top 10 words in normal comments");
```




In [9]:
```
train_texts, valid_texts, y_train, y_valid = train_test_split(train_df['comment'].values, train_df['label'].values, ra
ndom_state=17)
```

# Models

In [10]:
```
def preprocessing(texts):
    return [re.sub(r"([^ \w])", r" \1 ", str.lower(text)) for text in texts]

def tokenization(texts):
    return [text.split() for text in texts]

def build_vocabulary(data):
    vocab = dict()
    for d in data:
        for w in d:
            try:
                vocab[w]
            except:
                vocab[w] = len(vocab)
    return vocab

def build_embeddings_glove(file_path, vocab, d=300):
    emb_dict = dict()
    unk_array = np.zeros(d)
    with open(file_path, 'r', encoding="utf8") as f:
        for line in f:
            values = line.split()
            word = values[0]
            try:
                vocab[word]
                vector = np.asarray(values[1:], "float32")
                emb_dict[word] = vector
                unk_array += vector
            except:
                continue
    emb_dict['UNK'] = unk_array / len(emb_dict)
    return emb_dict

def build_w2v_dict(file_path, vocab, d=300):
    emb_dict = dict()
    unk_array = np.zeros(d)
    w2v_model = KeyedVectors.load_word2vec_format(file_path, binary=True)
    for word in vocab.keys():
        try:
            vector = w2v_model.get_vector(word)
            emb_dict[word] = vector
```

```
                unk_array += vector
            except:
                continue
        emb_dict['UNK'] = unk_array / len(emb_dict)
        return emb_dict

def build_emb_matrix_lr(data, emb_dict):
    X = []
    cnt_unk = 0
    cnt_total = 0
    for d in data:
        sentence_emb = np.zeros(len(emb_dict['UNK']))
        for w in d:
            cnt_total += 1
            try:
                sentence_emb += emb_dict[w]
            except:
                cnt_unk += 1
                sentence_emb += emb_dict['UNK']
        X.append(sentence_emb / len(d))
    return np.array(X), cnt_unk / cnt_total

def build_emb_dict_nn(file_path, vocab, d=300):
    emb_dict = dict()
    unk_array = np.zeros(d)
    with open(file_path, 'r', encoding="utf8") as f:
        for line in f:
            values = line.split()
            word = values[0]
            try:
                vocab[word]
                vector = np.asarray(values[1:], "float32")
                emb_dict[word] = vector
                unk_array += vector
            except:
                continue
    emb_dict['UNK'] = unk_array / len(emb_dict)
    emb_dict['PAD'] = np.zeros(d)
    return emb_dict

def build_emb_matrix_nn(file_path, vocab, d=300):
    emb_dict = build_emb_dict_nn(file_path, vocab, d=d)
```

```
    emb_matrix = np.zeros((len(emb_dict), d))
    word2idx = {'UNK': 0, 'PAD': 1}
    for word in sorted(list(set(emb_dict.keys()) - set(['UNK', 'PAD']))):
        word2idx[word] = len(word2idx)
    for w, i in word2idx.items():
        emb_matrix[i] = emb_dict[w]
    emb_matrix = torch.tensor(emb_matrix)
    return emb_matrix, word2idx

def build_w2v_dict_nn(file_path, vocab, d=300):
    emb_dict = dict()
    unk_array = np.zeros(d)
    w2v_model = KeyedVectors.load_word2vec_format(file_path, binary=True)
    for word in vocab.keys():
        try:
            vector = w2v_model.get_vector(word)
            emb_dict[word] = vector
            unk_array += vector
        except:
            continue
    emb_dict['UNK'] = unk_array / len(emb_dict)
    emb_dict['PAD'] = np.zeros(d)
    return emb_dict

def build_emb_matrix_nn_w2v(file_path, vocab, d=300):
    emb_dict = build_w2v_dict_nn(file_path, vocab, d=d)
    emb_matrix = np.zeros((len(emb_dict), d))
    word2idx = {'UNK': 0, 'PAD': 1}
    for word in sorted(list(set(emb_dict.keys()) - set(['UNK', 'PAD']))):
        word2idx[word] = len(word2idx)
    for w, i in word2idx.items():
        emb_matrix[i] = emb_dict[w]
    emb_matrix = torch.tensor(emb_matrix)
    return emb_matrix, word2idx

class LR_Doc2Vec:
    def __init__(self, doc2vec_model, C=1.0):
        super(LR_Doc2Vec, self).__init__()
        self.doc2vec_model = doc2vec_model
        self.C = C
        self.lr = LogisticRegression(C=C, random_state=13)
```

```
    def load_embeddings(self, X):
        X_emb = []
        for x in X:
            X_emb.append(self.doc2vec_model.infer_vector(x))
        X_emb = np.array(X_emb)
        return X_emb

    def fit(self, X_train, y_train):
        X_train_emb = self.load_embeddings(X_train)
        self.lr.fit(X_train_emb, y_train)
        del X_train_emb
        return self

    def predict(self, X_test):
        X_test_emb = self.load_embeddings(X_test)
        y_pred = self.lr.predict(X_test_emb)
        del X_test_emb
        return y_pred

    def predict_proba(self, X_test):
        X_test_emb = self.load_embeddings(X_test)
        y_pred = self.lr.predict_proba(X_test_emb)
        del X_test_emb
        return y_pred

class BiLSTM(nn.Module):
    def __init__(self, emb_matrix, hidden_size=64, output_size=2, freeze_emb=True):
        super(BiLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding.from_pretrained(emb_matrix)
        if freeze_emb:
            self.embedding.weight.requires_grad = False
        self.lstm = nn.LSTM(
            input_size=self.embedding.embedding_dim,
            hidden_size=hidden_size,
            bidirectional=True,
            batch_first=True
        )
        self.fc = nn.Linear(2 * hidden_size, output_size)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
```

```
        x_emb = self.embedding(x)
        # (batch, seq_len, num_directions * hidden_size)
        lstm_out, _ = self.lstm(x_emb.float())
        # (batch, seq_len, num_directions, hidden_size)
        lstm_out = lstm_out.view(lstm_out.shape[0], lstm_out.shape[1], -1, self.hidden_size)
        # lstm_out[:, :, 0, :] -- output of the forward LSTM
        # lstm_out[:, :, 1, :] -- output of the backward LSTM
        # we take the last hidden state of the forward LSTM and the first hidden state of the backward LSTM
        x_fc = torch.cat((lstm_out[:, -1, 0, :], lstm_out[:, 0, 1, :]), dim=1)
        fc_out = self.fc(x_fc)
        out = self.softmax(fc_out)
        return out

def as_matrix(documents, word2idx, max_len=None):
    max_doc_len = max(map(len, documents))
    if max_len is None:
        max_len = max_doc_len
    else:
        max_len = min(max_doc_len, max_len)
    matrix = np.ones((len(documents), max_len), dtype=np.int64)
    for i, doc in enumerate(documents):
        row_ix = [word2idx.get(word, 0) for word in doc[:max_len]]
        matrix[i, :len(row_ix)] = row_ix
    return matrix

def predict_bilstm(model, dev_data, word2idx, max_len=300, device=device, batch_size=16):
    with torch.no_grad():
        val_size = len(dev_data)
        y_pred = np.zeros(val_size, dtype=float)
        for i in range(0, val_size, batch_size):
            x = as_matrix(dev_data[i:(i + batch_size)], word2idx, max_len)
            x = torch.tensor(x).long()
            x = x.to(device)
            prediction = model(x)[:, 1]
            y_pred[i:(i + batch_size)] = prediction.cpu().detach().numpy()
    return y_pred

class BiLSTM_ELMo(nn.Module):
    def __init__(self, elmo, input_size=256, hidden_size=64, output_size=2):
        super(BiLSTM_ELMo, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = elmo
```

```
        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            bidirectional=True,
            batch_first=True
        )
        self.fc = nn.Linear(2 * hidden_size, output_size)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        x_emb = self.embedding(x)['elmo_representations'][0]
        # (batch, seq_len, num_directions * hidden_size)
        lstm_out, _ = self.lstm(x_emb.float())
        # (batch, seq_len, num_directions, hidden_size)
        lstm_out = lstm_out.view(lstm_out.shape[0], lstm_out.shape[1], -1, self.hidden_size)
        # lstm_out[:, :, 0, :] -- output of the forward LSTM
        # lstm_out[:, :, 1, :] -- output of the backward LSTM
        # we take the last hidden state of the forward LSTM and the first hidden state of the backward LSTM
        x_fc = torch.cat((lstm_out[:, -1, 0, :], lstm_out[:, 0, 1, :]), dim=1)
        fc_out = self.fc(x_fc)
        out = self.softmax(fc_out)
        return out

def predict_bilstm_elmo(model, dev_data, max_len=300, device=device, batch_size=4):
    with torch.no_grad():
        val_size = len(dev_data)
        y_pred = np.zeros(val_size, dtype=float)
        for i in range(0, val_size, batch_size):
            x = batch_to_ids(dev_data[i:(i + batch_size)])
            if max_len is not None:
                x = x[:, :max_len]
            x = x.to(device)
            prediction = model(x)[:, 1]
            y_pred[i:(i + batch_size)] = prediction.cpu().detach().numpy()
    return y_pred

def set_random_seeds(seed_value=13, device='cpu'):
    '''source https://forums.fast.ai/t/solved-reproducibility-where-is-the-randomness-coming-in/31628/5'''
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    random.seed(seed_value)
    if device != 'cpu':
```

```
        torch.cuda.manual_seed(seed_value)
        torch.cuda.manual_seed_all(seed_value)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False
```

In [11]:
```
%%time

train_tokens = tokenization(preprocessing(train_texts))
valid_tokens = tokenization(preprocessing(valid_texts))

vocab = build_vocabulary(train_tokens)
print("Vocabulary size:", len(vocab))
```

```
Vocabulary size: 143374
Wall time: 10.6 s
```

First of all, we tried Logistic Regression with various feature representations for texts. We tried to use it based on TF-IDF matrix, matrix of counts, word embeddings (GloVe and word2vec) and document embeddings (Doc2Vec).

## LR + TF-IDF

In [24]:
```
%%time

tfidf_vec = TfidfVectorizer(ngram_range=(1, 3), max_features=50000, min_df=5)
train_texts_tfidf = tfidf_vec.fit_transform(train_texts)
valid_texts_tfidf = tfidf_vec.transform(valid_texts)
```

```
Wall time: 1min 1s
```

In [25]:
```
%%time

lr = LogisticRegression(solver='lbfgs', max_iter=500, random_state=13)
lr.fit(train_texts_tfidf, y_train)
y_pred_lr_tfidf = lr.predict_proba(valid_texts_tfidf)[:, 1]
print(roc_auc_score(y_valid, y_pred_lr_tfidf))
```

```
0.7942782786624244
Wall time: 31.8 s
```

## LR + CountVectorizer

In [12]:
```
%%time

cnt_vec = CountVectorizer(ngram_range=(1, 3), max_features=40000, min_df=1)
train_texts_cnt = cnt_vec.fit_transform(train_texts)
valid_texts_cnt = cnt_vec.transform(valid_texts)
```

```
Wall time: 50.4 s
```

In [13]:
```
%%time

lr = LogisticRegression(solver='lbfgs', max_iter=500, random_state=13)
lr.fit(train_texts_cnt, y_train)
y_pred_lr_cnt = lr.predict_proba(valid_texts_cnt)[:, 1]
print(roc_auc_score(y_valid, y_pred_lr_cnt))
```

```
0.7877326249300394
Wall time: 55.2 s
```

## LR + GloVe

In [15]:
```
emb_dict = build_embeddings_glove('glove.6B.300d.txt', vocab)
print('Unique vectors in embeddings dictionary:', len(emb_dict))

train_emb_matrix, train_unk = build_emb_matrix_lr(train_tokens, emb_dict)
valid_emb_matrix, valid_unk = build_emb_matrix_lr(valid_tokens, emb_dict)
print('Train embedding matrix shape:', train_emb_matrix.shape)
print('Train: {:.2f}% unknown words'.format(train_unk * 100))
print('Valid embedding matrix shape:', valid_emb_matrix.shape)
print('Valid: {:.2f}% unknown words'.format(valid_unk * 100))
```

```
Unique vectors in embeddings dictionary: 78684
Train embedding matrix shape: (758079, 300)
Train: 1.34% unknown words
Valid embedding matrix shape: (252694, 300)
Valid: 1.59% unknown words
```

In [16]:
```
%%time

lr = LogisticRegression(C=5, solver='sag', max_iter=500, random_state=13)
lr.fit(train_emb_matrix, y_train)
y_pred_lr_glove = lr.predict_proba(valid_emb_matrix)[:, 1]
print(roc_auc_score(y_valid, y_pred_lr_glove))
```

```
0.6748553399754051
Wall time: 46.7 s
```

## LR + Word2Vec

In [17]:
```
emb_dict = build_w2v_dict('GoogleNews-vectors-negative300.bin', vocab)
print('Unique vectors in embeddings dictionary:', len(emb_dict))

train_emb_matrix, train_unk = build_emb_matrix_lr(train_tokens, emb_dict)
valid_emb_matrix, valid_unk = build_emb_matrix_lr(valid_tokens, emb_dict)
print('Train embedding matrix shape:', train_emb_matrix.shape)
print('Train: {:.2f}% unknown words'.format(train_unk * 100))
print('Valid embedding matrix shape:', valid_emb_matrix.shape)
print('Valid: {:.2f}% unknown words'.format(valid_unk * 100))
```

```
Unique vectors in embeddings dictionary: 62373
Train embedding matrix shape: (758079, 300)
Train: 23.67% unknown words
Valid embedding matrix shape: (252694, 300)
Valid: 23.71% unknown words
```

In [18]:
```
%%time

lr = LogisticRegression(solver='lbfgs', max_iter=500, random_state=13)
lr.fit(train_emb_matrix, y_train)
y_pred_lr_w2v = lr.predict_proba(valid_emb_matrix)[:, 1]
print(roc_auc_score(y_valid, y_pred_lr_w2v))
```

```
0.673213417887968
Wall time: 24.7 s
```

## LR + Doc2Vec

In [19]:
```
%%time

train_docs = [TaggedDocument(d, [i]) for (i, d) in enumerate(train_tokens)]
doc2vec_model = Doc2Vec(vector_size=300, min_count=1, epochs=5)
doc2vec_model.build_vocab(train_docs)
doc2vec_model.train(train_docs, total_examples=doc2vec_model.corpus_count, epochs=doc2vec_model.epochs)
```

```
Wall time: 3min 8s
```

In [20]:
```
%%time

lr_doc2vec = LR_Doc2Vec(doc2vec_model)
lr_doc2vec.fit(train_tokens, y_train)
```

```
Wall time: 3min 51s
```

In [21]:
```
%%time

y_pred_lr_doc2vec = lr_doc2vec.predict_proba(valid_tokens)[:, 1]
print(roc_auc_score(y_valid, y_pred_lr_doc2vec))
```

```
0.6774603171532291
Wall time: 49.2 s
```

Then we tried NN models. We used BiLSTM with GloVe, word2vec and ELMo embeddings.

## BiLSTM + GloVe

In [106]:
```
%%time

emb_matrix_bilstm_glove, word2idx_bilstm_glove = build_emb_matrix_nn('glove.6B.300d.txt', vocab)
print('Unique vectors in embedding matrix:', len(emb_matrix_bilstm_glove))
```

```
Unique vectors in embedding matrix: 78685
Wall time: 24.8 s
```

def train_bilstm(model, optimizer, train_data, train_labels_tensor, dev_data, dev_labels_tensor, word2idx, max_len=300, device=device, n_epochs=50, batch_size=256): model.to(device) train_loss_curve = [np.nan] * n_epochs val_loss_curve = [np.nan] * n_epochs train_accuracy_curve = [np.nan] * n_epochs val_accuracy_curve = [np.nan] * n_epochs max_val_accuracy = 0 max_val_accuracy_epoch = 0 best_state_dict = None train_size = len(train_data) val_size = len(dev_data) n_batches_train = (train_size - 1) // batch_size + 1 n_batches_val = (val_size - 1) // batch_size + 1 for epoch in range(n_epochs): model.train() train_loss_curve[epoch] = 0 train_accuracy_curve[epoch] = 0 for i in range(0, train_size, batch_size): x = as_matrix(train_data[i:(i + batch_size)], word2idx, max_len) x = torch.tensor(x).long() y = train_labels_tensor[i:(i + batch_size)].float() x = x.to(device) y = y.to(device) optimizer.zero_grad() prediction = model(x) labels_pred = prediction.argmax(dim=-1, keepdim=False).view(-1) labels_true = y.argmax(dim=-1, keepdim=False).view(-1) train_accuracy_curve[epoch] += labels_pred.eq(labels_true).sum().item() / len(y) loss = nn.BCELoss()(prediction, y) train_loss_curve[epoch] += loss.item() loss.backward() optimizer.step()

train_accuracy_curve[epoch] /= n_batches_train display.clear_output(wait=True) f, axes = plt.subplots(1, 2, figsize=(15, 5)) train_loss_curve[epoch] /= n_batches_train axes[0].plot(train_loss_curve, label='train') model.eval() val_loss_curve[epoch] = 0 val_accuracy_curve[epoch] = 0 with torch.no_grad(): for i in range(0, val_size, batch_size): x = as_matrix(dev_data[i:(i + batch_size)], word2idx, max_len) x = torch.tensor(x).long() y = dev_labels_tensor[i:(i + batch_size)].float() x = x.to(device) y = y.to(device) prediction = model(x) labels_pred = prediction.argmax(dim=-1, keepdim=False).view(-1) labels_true = y.argmax(dim=-1, keepdim=False).view(-1) val_accuracy_curve[epoch] += labels_pred.eq(labels_true).sum().item() / len(y) loss = nn.BCELoss()(prediction, y) val_loss_curve[epoch] += loss.item() val_accuracy_curve[epoch] /= n_batches_val val_loss_curve[epoch] /= n_batches_val axes[0].plot(val_loss_curve, label='val') axes[0].set_title('Loss: train {:.4f}, val {:.4f}'.format(train_loss_curve[epoch], val_loss_curve[epoch])) axes[0].legend() axes[0].set_xlabel('epochs') axes[0].set_ylabel('loss') val_accuracy = val_accuracy_curve[epoch] if val_accuracy > max_val_accuracy: max_val_accuracy = val_accuracy max_val_accuracy_epoch = epoch best_state_dict = deepcopy(model.state_dict()) axes[1].set_title('Accuracy: train {:.4f}, val {:.4f} (best: {:.4f}, epoch {})'.format( train_accuracy_curve[epoch], val_accuracy, max_val_accuracy, max_val_accuracy_epoch )) axes[1].plot(train_accuracy_curve, label='train') axes[1].plot(val_accuracy_curve, label='val') axes[1].legend() axes[1].set_xlabel('epochs') axes[1].set_ylabel('accuracy') plt.tight_layout() plt.show() return best_state_dict train_labels_tensor = torch.tensor(y_train) train_labels_tensor = torch.cat([1 - train_labels_tensor.view(-1, 1), train_labels_tensor.view(-1, 1)], dim=1) dev_labels_tensor = torch.tensor(y_valid) dev_labels_tensor = torch.cat([1 - dev_labels_tensor.view(-1, 1), dev_labels_tensor.view(-1, 1)], dim=1) hidden_size = 128 lr = 0.1 bilstm_glove = BiLSTM(emb_matrix_bilstm_glove, hidden_size) optimizer = optim.SGD(bilstm_glove.parameters(), lr=lr) set_random_seeds(13, device) bilstm_glove_state_dict = train_bilstm(bilstm_glove, optimizer, train_tokens, train_labels_tensor, valid_tokens, dev_labels_tensor, word2idx_bilstm_glove, n_epochs=20) STATE_DICT_PATH = 'bilstm_glove.pt' torch.save(bilstm_glove_state_dict, STATE_DICT_PATH)

In [107]:

```
%%time

BILSTM_STATE_DICT = 'bilstm_glove.pt'
hidden_size = 128

bilstm_glove = BiLSTM(emb_matrix_bilstm_glove, hidden_size).to(device)
bilstm_glove.load_state_dict(torch.load(BILSTM_STATE_DICT, map_location=device))
bilstm_glove.eval()

y_pred_bilstm_glove = predict_bilstm(bilstm_glove, valid_tokens, word2idx_bilstm_glove)
print(roc_auc_score(y_valid, y_pred_bilstm_glove))
```

```
0.8097815263231354
Wall time: 44.5 s
```

## BiLSTM + Word2Vec

In [28]:
```
%%time

emb_matrix_bilstm_w2v, word2idx_bilstm_w2v = build_emb_matrix_nn_w2v('GoogleNews-vectors-negative300.bin', vocab)
print('Unique vectors in embedding matrix:', len(emb_matrix_bilstm_w2v))
```

```
Unique vectors in embedding matrix: 62374
Wall time: 1min 4s
```

train_labels_tensor = torch.tensor(y_train) train_labels_tensor = torch.cat([1 - train_labels_tensor.view(-1, 1), train_labels_tensor.view(-1, 1)], dim=1) dev_labels_tensor = torch.tensor(y_valid) dev_labels_tensor = torch.cat([1 - dev_labels_tensor.view(-1, 1), dev_labels_tensor.view(-1, 1)], dim=1) hidden_size = 128 lr = 0.1 bilstm_w2v = BiLSTM(emb_matrix_bilstm_w2v, hidden_size) optimizer = optim.SGD(bilstm_w2v.parameters(), lr=lr) set_random_seeds(13, device) bilstm_w2v_state_dict = train_bilstm(bilstm_w2v, optimizer, train_tokens, train_labels_tensor, valid_tokens, dev_labels_tensor, word2idx_bilstm_w2v, n_epochs=20) STATE_DICT_PATH = 'bilstm_w2v.pt' torch.save(bilstm_w2v_state_dict, STATE_DICT_PATH)

In [29]:
```
%%time

BILSTM_STATE_DICT = 'bilstm_w2v.pt'
hidden_size = 128

bilstm_w2v = BiLSTM(emb_matrix_bilstm_w2v, hidden_size).to(device)
bilstm_w2v.load_state_dict(torch.load(BILSTM_STATE_DICT, map_location=device))
bilstm_w2v.eval()

y_pred_bilstm_w2v = predict_bilstm(bilstm_w2v, valid_tokens, word2idx_bilstm_w2v)
print(roc_auc_score(y_valid, y_pred_bilstm_w2v))
```

```
0.7942717703722182
Wall time: 33.4 s
```

## BiLSTM + ELMo

def train_bilstm_elmo(model, optimizer, train_data, train_labels_tensor, dev_data, dev_labels_tensor, max_len=None, device=device, n_epochs=50, batch_size=256): model.to(device) train_loss_curve = [np.nan] * n_epochs val_loss_curve = [np.nan] * n_epochs train_accuracy_curve = [np.nan] * n_epochs val_accuracy_curve = [np.nan] * n_epochs max_val_accuracy = 0 max_val_accuracy_epoch = 0 best_state_dict = None train_size = len(train_data) val_size = len(dev_data) n_batches_train = (train_size - 1) // batch_size + 1 n_batches_val = (val_size - 1) // batch_size + 1 for epoch in range(n_epochs): model.train() train_loss_curve[epoch] = 0 train_accuracy_curve[epoch] = 0 for i in tqdm(range(0, train_size, batch_size)): x = batch_to_ids(train_data[i:(i + batch_size)]) if max_len is not None: x = x[:, :max_len] y = train_labels_tensor[i:(i + batch_size)].float() x = x.to(device) y = y.to(device) optimizer.zero_grad() prediction = model(x) labels_pred

= prediction.argmax(dim=-1, keepdim=False).view(-1) labels_true = y.argmax(dim=-1, keepdim=False).view(-1) train_accuracy_curve[epoch] += labels_pred.eq(labels_true).sum().item() / len(y) loss = nn.BCELoss()(prediction, y) train_loss_curve[epoch] += loss.item() loss.backward() optimizer.step() train_accuracy_curve[epoch] /= n_batches_train display.clear_output(wait=True) f, axes = plt.subplots(1, 2, figsize=(15, 5)) train_loss_curve[epoch] /= n_batches_train axes[0].plot(train_loss_curve, label='train') model.eval() val_loss_curve[epoch] = 0 val_accuracy_curve[epoch] = 0 with torch.no_grad(): for i in tqdm(range(0, val_size, batch_size)): x = batch_to_ids(dev_data[i:(i + batch_size)]) if max_len is not None: x = x[:, :max_len] y = dev_labels_tensor[i:(i + batch_size)].float() x = x.to(device) y = y.to(device) prediction = model(x) labels_pred = prediction.argmax(dim=-1, keepdim=False).view(-1) labels_true = y.argmax(dim=-1, keepdim=False).view(-1) val_accuracy_curve[epoch] += labels_pred.eq(labels_true).sum().item() / len(y) loss = nn.BCELoss()(prediction, y) val_loss_curve[epoch] += loss.item() val_accuracy_curve[epoch] /= n_batches_val val_loss_curve[epoch] /= n_batches_val axes[0].plot(val_loss_curve, label='val') axes[0].set_title('Loss: train {:.4f}, val {:.4f}'.format(train_loss_curve[epoch], val_loss_curve[epoch])) axes[0].legend() axes[0].set_xlabel('epochs') axes[0].set_ylabel('loss') val_accuracy = val_accuracy_curve[epoch] if val_accuracy > max_val_accuracy: max_val_accuracy = val_accuracy max_val_accuracy_epoch = epoch best_state_dict = deepcopy(model.state_dict()) axes[1].set_title('Accuracy: train {:.4f}, val {:.4f} (best: {:.4f}, epoch {})'.format( train_accuracy_curve[epoch], val_accuracy, max_val_accuracy, max_val_accuracy_epoch )) axes[1].plot(train_accuracy_curve, label='train') axes[1].plot(val_accuracy_curve, label='val') axes[1].legend() axes[1].set_xlabel('epochs') axes[1].set_ylabel('accuracy') plt.tight_layout() plt.show() return best_state_dict train_labels_tensor = torch.tensor(y_train) train_labels_tensor = torch.cat([1 - train_labels_tensor.view(-1, 1), train_labels_tensor.view(-1, 1)], dim=1) dev_labels_tensor = torch.tensor(y_valid) dev_labels_tensor = torch.cat([1 - dev_labels_tensor.view(-1, 1), dev_labels_tensor.view(-1, 1)], dim=1) hidden_size = 64 lr = 0.1 options_file = "elmo_2x1024_128_2048cnn_1xhighway_options.json" weight_file = "elmo_2x1024_128_2048cnn_1xhighway_weights.hdf5" elmo = Elmo(options_file, weight_file, 1, dropout=0) bilstm_elmo = BiLSTM_ELMo(elmo, hidden_size=hidden_size) optimizer = optim.SGD(bilstm_elmo.parameters(), lr=lr) set_random_seeds(13, device) bilstm_elmo_state_dict = train_bilstm_elmo(bilstm_elmo, optimizer, train_tokens, train_labels_tensor, valid_tokens, dev_labels_tensor, max_len=50, n_epochs=6, batch_size=256) STATE_DICT_PATH = 'bilstm_elmo.pt' torch.save(bilstm_elmo_state_dict, STATE_DICT_PATH)

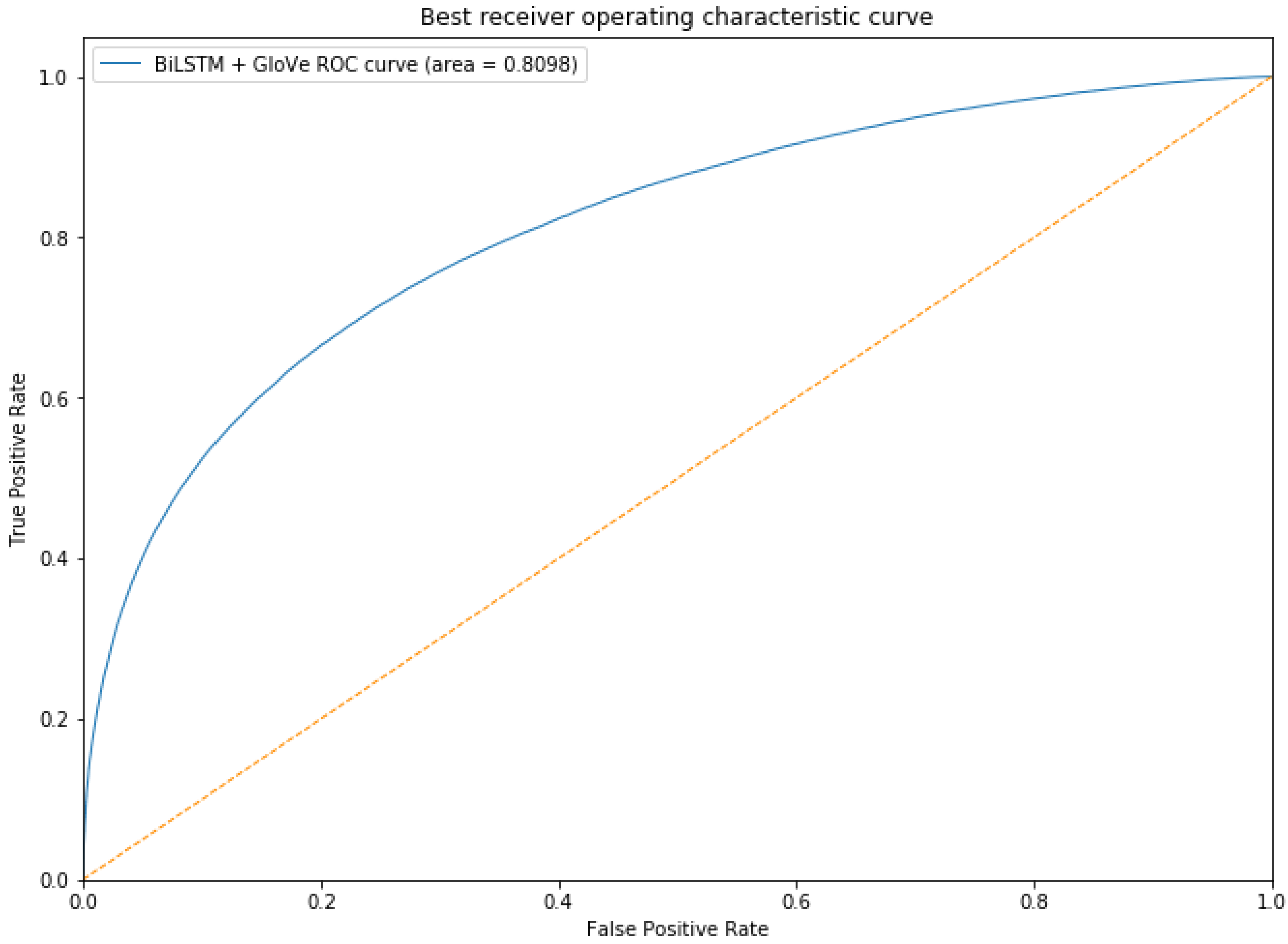
In [11]:
```
%%time

BILSTM_STATE_DICT = 'bilstm_elmo.pt'

hidden_size = 64

options_file = "elmo_2x1024_128_2048cnn_1xhighway_options.json"
weight_file = "elmo_2x1024_128_2048cnn_1xhighway_weights.hdf5"

elmo = Elmo(options_file, weight_file, 1, dropout=0)
bilstm_elmo = BiLSTM_ELMo(elmo, hidden_size=hidden_size).to(device)
bilstm_elmo.load_state_dict(torch.load(BILSTM_STATE_DICT, map_location=device))
bilstm_elmo.eval()

y_pred_bilstm_elmo = predict_bilstm_elmo(bilstm_elmo, valid_tokens, max_len=50, batch_size=256)
print(roc_auc_score(y_valid, y_pred_bilstm_elmo))
```

```
0.8089476675477861
Wall time: 17min 48s
```

# Results Analysis

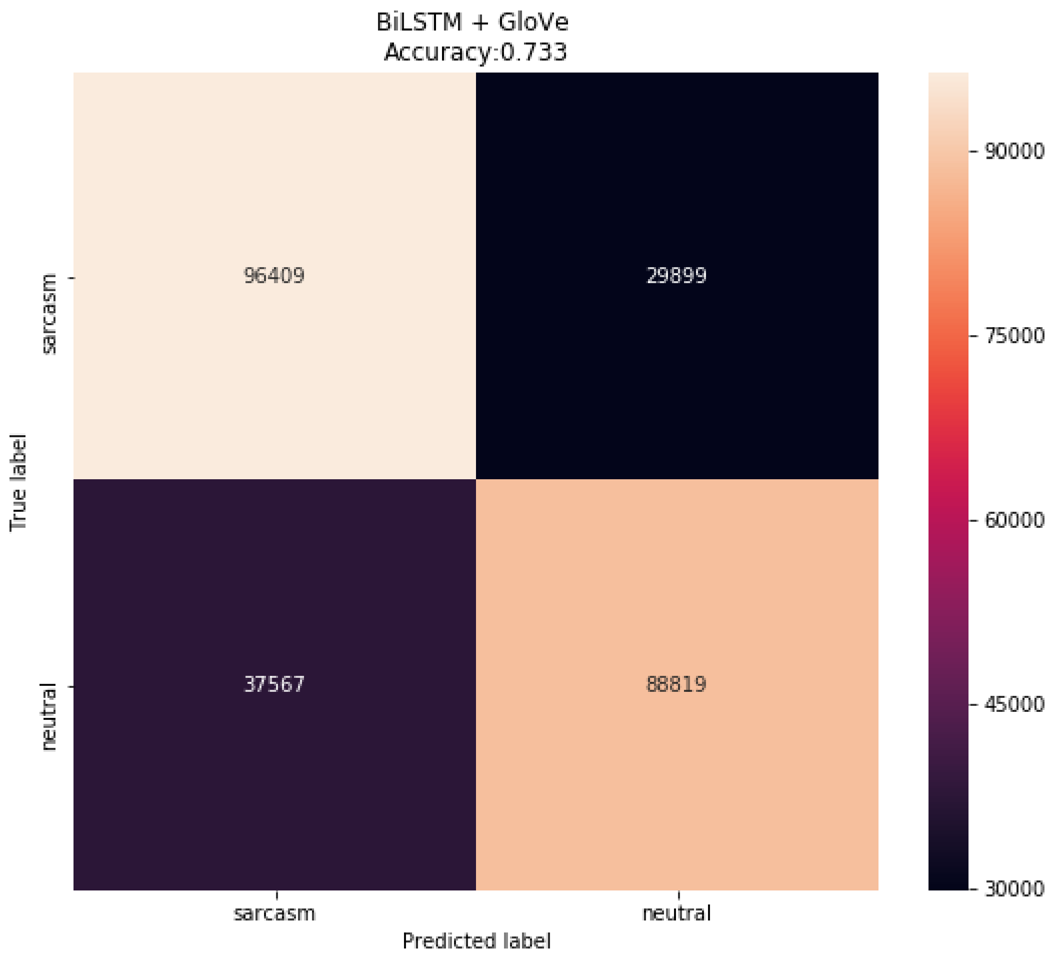
BiLSTM model with GloVe embeddings showed the best results. Let's take a look at the ROC-curve.

In [23]:
```
def generate_metrics(prediction, y_true):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    fpr[0], tpr[0], _ = roc_curve(y_true, 1 - prediction)
    roc_auc[0] = auc(fpr[0], tpr[0])
    fpr[1], tpr[1], _ = roc_curve(y_true, prediction)
    roc_auc[1] = auc(fpr[1], tpr[1])
    return fpr, tpr, roc_auc

bilstm_glove_fpr, bilstm_glove_tpr, bilstm_glove_roc_auc = generate_metrics(y_pred_bilstm_glove, y_valid)

plt.figure(figsize=(11, 8))
plt.plot(bilstm_glove_fpr[1], bilstm_glove_tpr[1], lw=1, label='BiLSTM + GloVe ROC curve (area = %0.4f)' % bilstm_glov
e_roc_auc[1])
plt.plot([0, 1], [0, 1], lw=1, color='darkorange', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Best receiver operating characteristic curve')
plt.legend(loc='best')
plt.show()
```




Transform probabilities to classes to plot confusion matrix and compute accuracy.

In [37]:
```
y_pred = (y_pred_bilstm_glove >= 0.5).astype(int)
cm = confusion_matrix(y_valid, y_pred)
cm_df = pd.DataFrame(cm, index=['sarcasm', 'neutral'], columns = ['sarcasm', 'neutral'])

plt.figure(figsize=(9, 7.5))
sns.heatmap(cm_df, fmt='d', annot=True)
plt.title('BiLSTM + GloVe \nAccuracy:{0:.3f}'.format(accuracy_score(y_valid, y_pred)))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



As we can see, model makes more mistakes in neutral class, more often considering it as sarcasm then considering sarcasm class neutral. Let's check where it makes worst mistakes.

In [84]:
```
def select_mistakes(y_true, y_pred_proba, texts, model_prediction, n=10):
    assert model_prediction in ['sarcasm', 'neutral']
    false_idx = np.where(y_true != (y_pred_proba >= 0.5).astype(int))
    mistakes_idx = y_pred_proba[false_idx].argsort()
    if model_prediction == 'sarcasm':
        mistakes_idx = mistakes_idx[::-1][:n]
    else:
        mistakes_idx = mistakes_idx[:n]
    mistakes = []
    for i in range(n):
        text = texts[false_idx][mistakes_idx[i]]
        model_pred = y_pred_proba[false_idx][mistakes_idx[i]]
        true_label = y_true[false_idx][mistakes_idx[i]]
        mistakes.append((text, model_pred, true_label))
    return mistakes
```

In [85]:
```
mistakes_sarcasm = select_mistakes(y_valid, y_pred_bilstm_glove, valid_texts, 'sarcasm')
for text, model_pred, true_label in mistakes_sarcasm:
    print('True label: {} | Model prediction: {:.4f}'.format(true_label, model_pred))
    print(text)
```

```
True label: 0 | Model prediction: 0.9987
Yeah, objective evidence refuting a belief is totally the same as a political ideology.
True label: 0 | Model prediction: 0.9982
Yeah, everyone insulting everyone for everything gives it an air of slight acceptance of everything.
True label: 0 | Model prediction: 0.9982
Because those poor black men are being oppressed!
True label: 0 | Model prediction: 0.9980
Oh ok, glad we cleared that up
True label: 0 | Model prediction: 0.9979
Yeah, sound quality is arguably the most important aspect of a piece of media.
True label: 0 | Model prediction: 0.9979
Oh yes, I'm sure women who have been raped will tell you the same thing.
True label: 0 | Model prediction: 0.9977
Yeah because we needed another rant video
True label: 0 | Model prediction: 0.9968
But unreliable gun has such better prices!
True label: 0 | Model prediction: 0.9967
Yeah, because we all know how well Kespa promotes their players' personality to foreign fans.
True label: 0 | Model prediction: 0.9966
Because America is the only country that matters duh!
```

Seems like the model overfits to the words "yeah" and "oh ok" in the beginning of the sentences.

In [86]:
```
mistakes_neutral = select_mistakes(y_valid, y_pred_bilstm_glove, valid_texts, 'neutral')
for text, model_pred, true_label in mistakes_neutral:
    print('True label: {} | Model prediction: {:.4f}'.format(true_label, model_pred))
    print(text)
```

```
True label: 1 | Model prediction: 0.0135
There actual field manuals for running an insurrection, iirc at one time the CIA had one, there was also a US army ma
nual on something similar plus the usual suspects I'm not naming just on the off chance there is a list I'm not alrea
dy on
True label: 1 | Model prediction: 0.0160
I didn't think this was necessary given the vein of the comment I was replying to, but:
True label: 1 | Model prediction: 0.0165
I want to upvote this because I like EXO, but I can't because I have no idea what is the reason of this performance a
nd I feel uncomfortable watching this because of the weird shots here and there EDIT : And also I noticed Lay is miss
ing ,ggwp EXO OT8 scandal inc
True label: 1 | Model prediction: 0.0186
From the number of times I have read about fines and settlements reached - monetary punitive measures really seem to
be fucking working.
True label: 1 | Model prediction: 0.0191
You have indeed, and it's interesting as FUCK :) Sounded like it, but not
True label: 1 | Model prediction: 0.0206
I did not like it when they made fun of Winnipeg :( it was funny but harsh.
True label: 1 | Model prediction: 0.0216
I know some that are decently priced and some that are reasonably priced, but none that are reasonably *and* decently
priced.
True label: 1 | Model prediction: 0.0216
I wouldn't go that far, but they're not wrong.
True label: 1 | Model prediction: 0.0231
Reminds me of some modern campers, may as well have stayed at home.
True label: 1 | Model prediction: 0.0232
that was a while ago, since then the IMF reserves have been put on a negative interest forcing huge investment, the B
ritish economy has recovered way above schedule, Germany took a little hit due to cheaper labour being found out east
but is still in control....Luxemburg, Switzerland, France (ish) and Scandinavia aren't really at much risk now, it wi
ll balance out and with a few political pledges and sanctions from those countries we will get back on track....Germa
ny has Greece by the balls anyway, if all else fails, borrow more from China
```

The model mistakenly takes these sarcastic comments as neutral, but actually it is pretty hard to see sarcasm in some of them, so these are probably really hard cases.

Now let's select best model guesses.

In [87]:
```
def select_guesses(y_true, y_pred_proba, texts, model_prediction, n=10):
    assert model_prediction in ['sarcasm', 'neutral']
    correct_idx = np.where(y_true == (y_pred_proba >= 0.5).astype(int))
    guesses_idx = y_pred_proba[correct_idx].argsort()
    if model_prediction == 'sarcasm':
        guesses_idx = guesses_idx[::-1][:n]
    else:
        guesses_idx = guesses_idx[:n]
    guesses = []
    for i in range(n):
        text = texts[correct_idx][guesses_idx[i]]
        model_pred = y_pred_proba[correct_idx][guesses_idx[i]]
        true_label = y_true[correct_idx][guesses_idx[i]]
        guesses.append((text, model_pred, true_label))
    return guesses
```

In [88]:
```
guesses_sarcasm = select_guesses(y_valid, y_pred_bilstm_glove, valid_texts, 'sarcasm')
for text, model_pred, true_label in guesses_sarcasm:
    print('True label: {} | Model prediction: {:.4f}'.format(true_label, model_pred))
    print(text)
```

```
True label: 1 | Model prediction: 0.9996
yeah, because filibusters are totally a recent invention
True label: 1 | Model prediction: 0.9996
Yeah, because that is so constructive and totally not falling to her level.
True label: 1 | Model prediction: 0.9996
Yes, because we totally need another MOBA
True label: 1 | Model prediction: 0.9996
Yeah, clearly that developer knew nothing about computers and only uses Facebook
True label: 1 | Model prediction: 0.9995
Yeah, clearly everyone should use 3200DPI with a tiny mousepad
True label: 1 | Model prediction: 0.9995
Yeah, we totally see mass shootings every day over here in europe
True label: 1 | Model prediction: 0.9995
Yep, because Asian women are totally incapable of giving consent
True label: 1 | Model prediction: 0.9995
Yeah, it's totally illuminating and welcoming to a multitude of world views.
True label: 1 | Model prediction: 0.9995
Yeah totally seems like a reliable source
True label: 1 | Model prediction: 0.9995
Yeah, because women can't do additions
```

Wow, ten out of ten most correctly identified sarcastic comments start with the words "yeah", "yes" or "yep"!

In [89]:
```
guesses_neutral = select_guesses(y_valid, y_pred_bilstm_glove, valid_texts, 'neutral')
for text, model_pred, true_label in guesses_neutral:
    print('True label: {} | Model prediction: {:.4f}'.format(true_label, model_pred))
    print(text)
```

```
True label: 0 | Model prediction: 0.0061
I did the same but i lost the button i removed so there goes any chance of trading it in for a slim :/ Sometimes i ge
t lazy and have a straightened paper clip and push it through the hole bellow where the eject button used to be and y
ou can press the contact in there to eject it.
True label: 0 | Model prediction: 0.0066
Fair enough but still for me it directly comes out of his hands propelled forwards, what you say is true with the for
ward momentum in most cases but not this one.
True label: 0 | Model prediction: 0.0072
I tried it a few weeks ago, it just didn't seem to do too much for me, it was pretty bland IMO, i was sad :/
True label: 0 | Model prediction: 0.0073
I've usually been inside, but found that temp has still been negative inside... I'm not home right now but when I get
home I can check, it might just have been me misreading the temperature :P
True label: 0 | Model prediction: 0.0077
I honestly didn't check to see the costs of higher ones, but typically this Xbox branded one is more expensive, and p
eople here recommend against Seagate because they have a higher failure rate.
True label: 0 | Model prediction: 0.0083
not saying it's EVERY time... but if a person is around one person/ a few people who use it that way all or even most
of the time, it's pretty easy to develop a bias to believing that.
True label: 0 | Model prediction: 0.0085
I feel like the Seal meme used here is from the OPs perspective, but could have been an Insanity Wolf meme just as ea
sily (grandpa's perspective).
True label: 0 | Model prediction: 0.0085
I dont have a source but according to some linguists, had the Normans never conquered England, English and Dutch woul
d still be mutually intelligible to a decent extent.
True label: 0 | Model prediction: 0.0086
I didn't up-vote it, but it is pretty funny.
True label: 0 | Model prediction: 0.0086
ok thanks:)
```

Nice, clearly these are neutral comments.

# Model interpretation

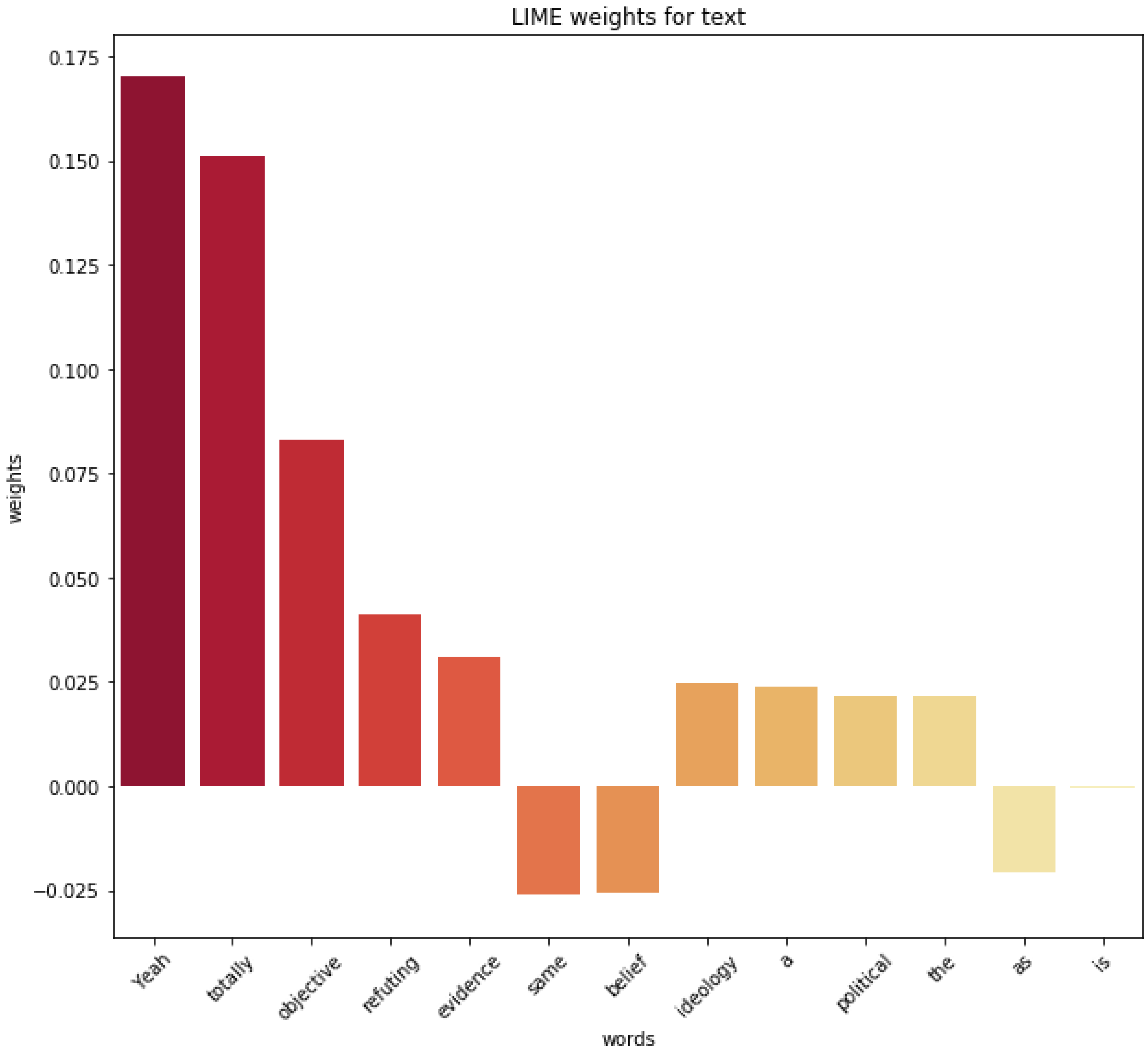
In order to make more clear model interpretation, we used LIME (https://arxiv.org/abs/1602.04938).

In [113]:
```
def predict_for_lime(list_texts):
    x = tokenization(preprocessing(list_texts))
    y_pred = predict_bilstm(bilstm_glove, x, word2idx_bilstm_glove)
    return np.stack([1 - y_pred, y_pred], axis=1)
```

Let's see what triggers model on the example with the highest mistake.

In [115]:
```
text = mistakes_sarcasm[0][0]
explainer = LimeTextExplainer()
explanation = explainer.explain_instance(text, predict_for_lime, num_features=20)

weights = OrderedDict(explanation.as_list())
lime_weights = pd.DataFrame({'words': list(weights.keys()), 'weights': list(weights.values())})
print(text)
print()
print('Sarcasm:', predict_bilstm(bilstm_glove, tokenization(preprocessing([text])), word2idx_bilstm_glove)[0])

plt.figure(figsize=(10, 9))
sns.barplot(x="words", y="weights", data=lime_weights, palette='YlOrRd_r');
plt.xticks(rotation=45)
plt.title('LIME weights for text'.format(0))
plt.show()
```

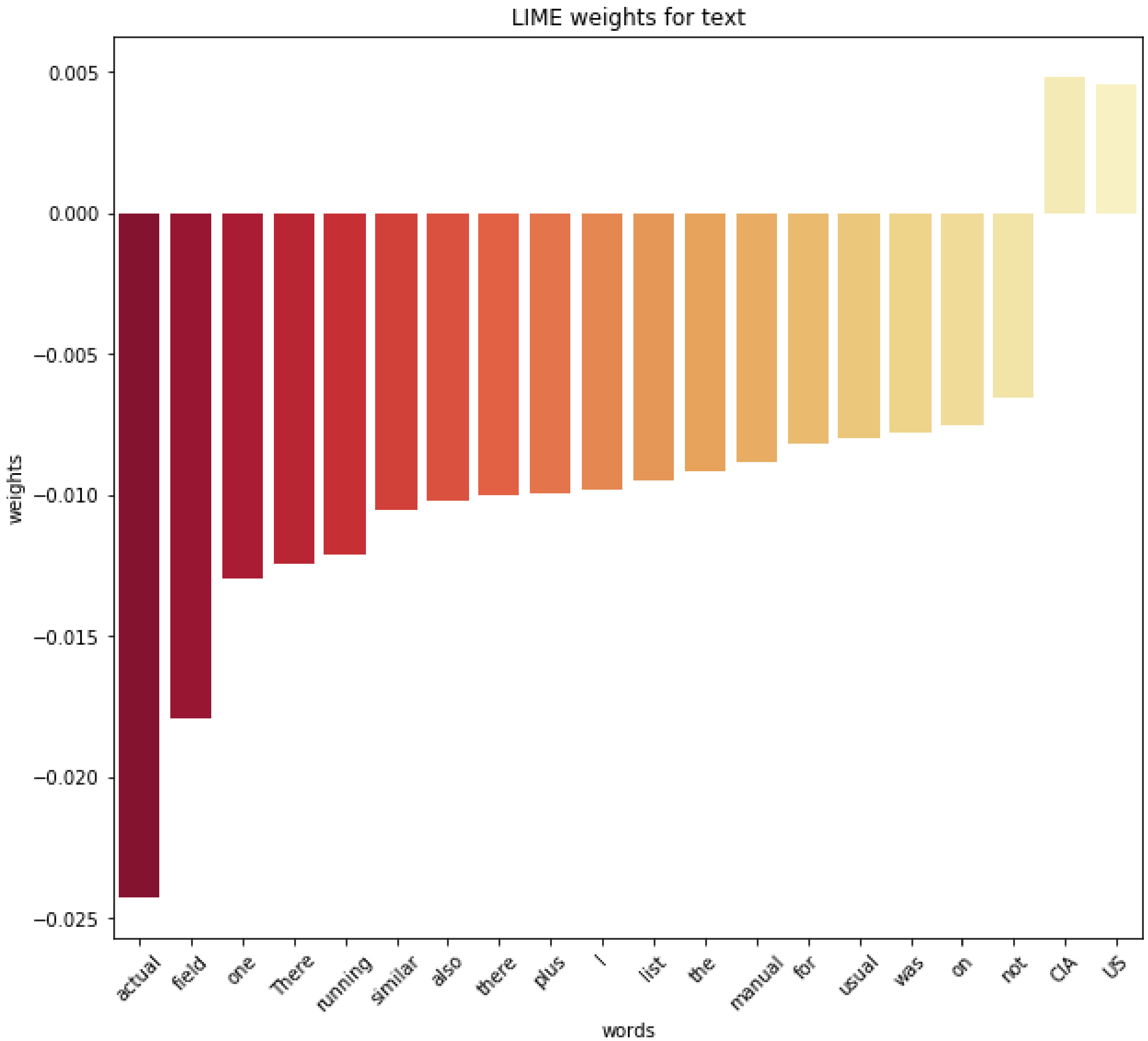
```
Yeah, objective evidence refuting a belief is totally the same as a political ideology.

Sarcasm: 0.9964801669120789
```



It reacts too much to the words "yeah" and "totally". Actually, from a human point of view, these are really hard trigger words, but according to the sentence it seems like they really are used in a neutral context. So the model is really tricked here.

In [116]:
```
text = mistakes_neutral[0][0]
explainer = LimeTextExplainer()
explanation = explainer.explain_instance(text, predict_for_lime, num_features=20)

weights = OrderedDict(explanation.as_list())
lime_weights = pd.DataFrame({'words': list(weights.keys()), 'weights': list(weights.values())})
print(text)
print()
print('Sarcasm:', predict_bilstm(bilstm_glove, tokenization(preprocessing([text])), word2idx_bilstm_glove)[0])

plt.figure(figsize=(10, 9))
sns.barplot(x="words", y="weights", data=lime_weights, palette='YlOrRd_r');
plt.xticks(rotation=45)
plt.title('LIME weights for text'.format(0))
plt.show()
```

```
There actual field manuals for running an insurrection, iirc at one time the CIA had one, there was also a US army ma
nual on something similar plus the usual suspects I'm not naming just on the off chance there is a list I'm not alrea
dy on

Sarcasm: 0.013517528772354126
```



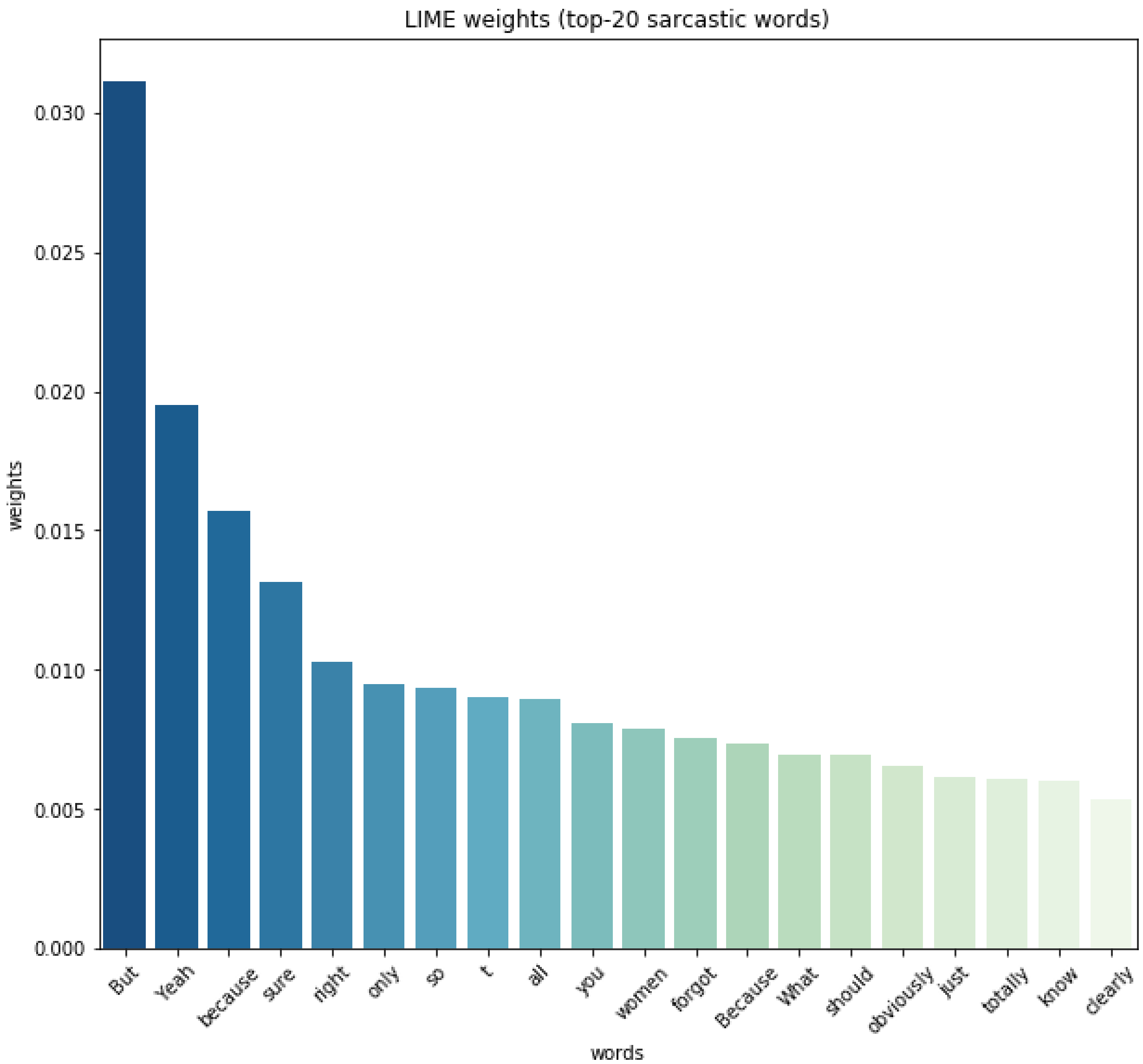
Absolutely no triggers => model thinks this is a neutral case.

In [131]:
```
%%time

n = 500

np.random.seed(13)
valid_global = np.random.choice(valid_texts, size=n)

lime_weights = {}
for i in range(n):
    text = valid_global[i]
    if (len(text.split()) > 1):
        explanation = explainer.explain_instance(text, predict_for_lime, num_features=20)
        weights = OrderedDict(explanation.as_list())
        for key in weights.keys():
            try:
                lime_weights[key] += weights[key]
            except:
                lime_weights[key] = weights[key]

print()
```

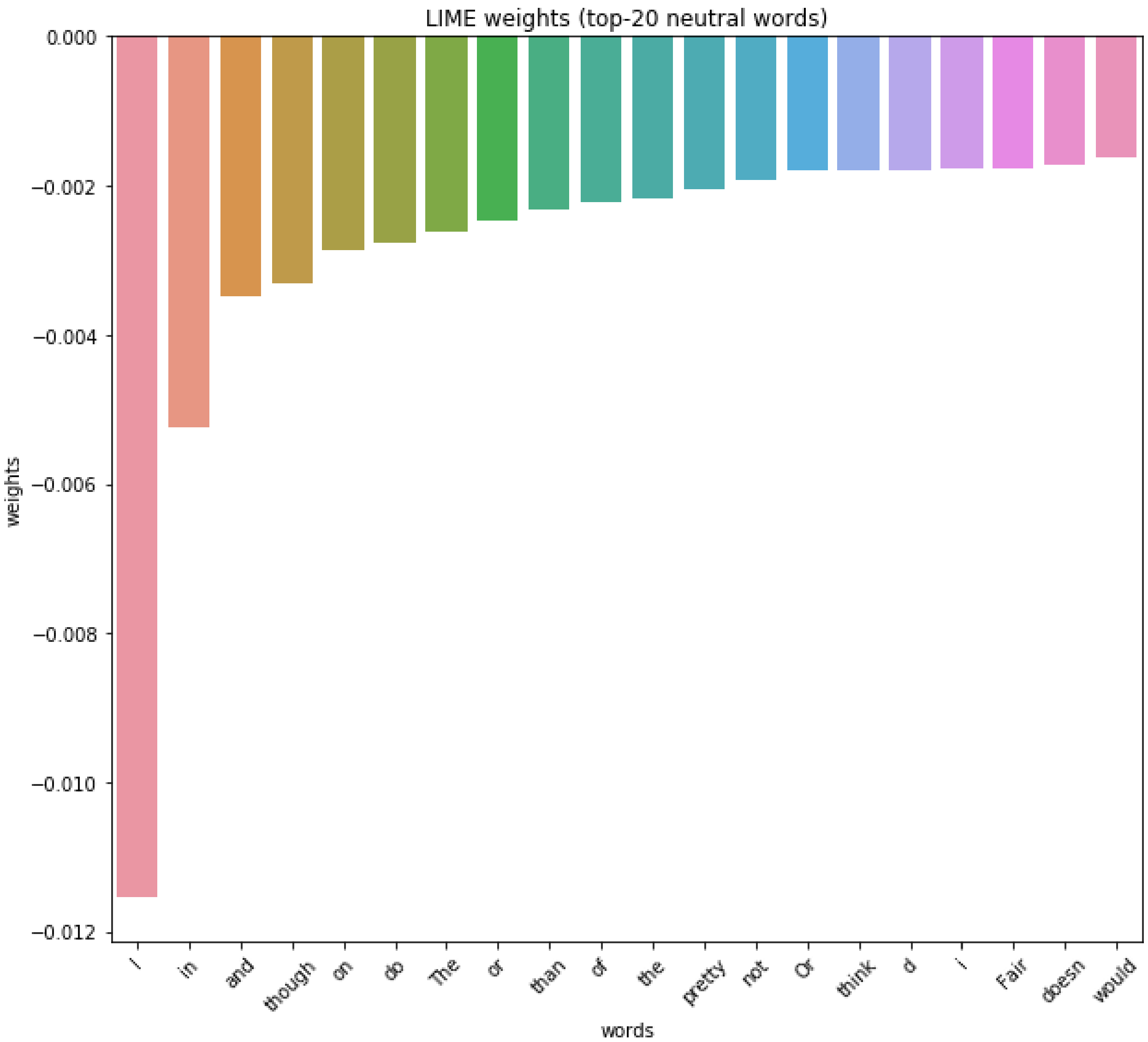
```
Wall time: 11min 46s
```

In [133]:
```
df_weights = pd.DataFrame({'words': list(lime_weights.keys()), 'weights': list(lime_weights.values())})
df_weights['weights_scaled'] = df_weights['weights'] / abs(df_weights['weights']).sum()
df_weights_scaled = df_weights[['words', 'weights_scaled']]

w_neg = df_weights_scaled.sort_values(by=['weights_scaled'], ascending=True)[:20]
w_pos = df_weights_scaled.sort_values(by=['weights_scaled'], ascending=False)[:20]
```

In [134]:
```
plt.figure(figsize=(10, 9))
sns.barplot(x="words", y="weights_scaled", data=w_pos, palette='GnBu_r');
plt.xticks(rotation=45)
plt.title('LIME weights (top-20 sarcastic words)'.format(0))
plt.ylabel('weights')
plt.show()
```



In [135]:
```
plt.figure(figsize=(10, 9))
sns.barplot(x="words", y="weights_scaled", data=w_neg);
plt.xticks(rotation=45)
plt.title('LIME weights (top-20 neutral words)'.format(0))
plt.ylabel('weights')
plt.show()
```



We can see that the results are pretty interpretable. Neutral words are really neutral. An interesting result is that a word "I" is the most neutral - probably because if people use "I", they naturally express their own opinion, and this imply a serious comment (non-sarcasm). Many sarcastic words are really used in sarcastic contexts: "Yeah", "sure", "right", "only".

So, we achieved a quality of approximately 0.8 AUC-ROC. It seems pretty OK, but it shows that the task is really tough which is seen from the interpreted examples (where there seems to be absolutely no sarcastic words, but there is a sarcasm). So the models might be more complex to understand sarcasm, but from the tested options BiLSTM + GloVe embeddings were the best.

According to the analysis of the results, the sarcasm mainly can be identified by trigger (sarcastic) words. This seems applicable to the real life - there, sarcastic words also can often show that the whole phrase is sarcastic.