University of Copenhagen

M.Sc. IT & Cognition

Language Processing 2

"Did you have a pleasant flight?"

Sentiment analysis of flight review tweets

Author:

Cristian Mitroi

15/06/2018

Contents

Introduction

The goal of the project is to develop a Sentiment Analysis system based on the Twitter airline review dataset (crowdflower user at Kaggle n.d.). This will start with an exploratory phase of visualizing the data available. I will then apply different combinations of preprocessing tasks and machine learning algorithms. These will all be measured by objective measures in order to decide which is the most effective. I will also provide a qualitative assessment of the model.

I have also decided that I will provide quantitative assessments not just in their respective section, but also throughout the 'Methods' section. This allows me to filter out algorithms as I progress through the options I explore. Exhausting all the possible combinations is outside the alloted time frame for this project.

Throughout the project, the main focus will be arguing and explaining the different decisions made. The learning goal is to become acquainted with the different possibilities in the field, and to apply them in practice.

Background

The field of sentiment analysis is a burgeoning one. There is a sea of literature in the field, each piece specializing in different aspects of it.[1] The field has been growing at a steady pace since the late 1990s (Google NGrams n.d.). The main reasons for this, according to (Pang & Lee 2008, p.4), are the "the rise of machine learning", "the availability of datasets [on] the blossoming [...] World Wide Web", and the focus on "commercial and intelligence applications". In the end, it is all fundamentally about the age-old question of "what other people think" (Pang & Lee 2008, p.1).

(Pang & Lee 2008) provides an excellent overview of the different techniques prevalent in the field of sentiment analysis (also referred to and overlapping with *opinion mining* or *subjectivity analysis*). Firstly, they list the various possibilities for transforming the raw text data into a design matrix that can be interpreted by machine learning algorithms (Pang & Lee 2008, p.20). They argue that this is a crucial choice to be made in any such enterprise. The book mentions approaches involving term presence, frequency, parts of speech etc.

(Pang & Lee 2008) also discusses the actual models employed in this area, emphasizing the split between the two categories of supervised and unsupervised. In this project we are dealing with labeled data, thus it will be in the supervised category.

[1] One search on the term 'sentiment analysis' on Google Scholar results in about 1,700,000 results (Google Scholar n.d.).

Methods

In this section I will discuss the sentiment analysis methods that I applied on the twitter dataset. The main purpose here was to experiment with different combinations of approaches within the preprocessing (data cleaning), text feature extraction and algorithm.

I first tested which approaches in preprocessing and feature extraction work best for this task. I then employed `GridSearchCV` in order to optimize the hyperparameters of several algorithms I selected for the task.

3.1 Preprocessing

In the first batch of model testing I wanted to ascertain what feature extraction methods produce the best results.

Firstly, I tested two different approaches to cleaning the tweets themselves. The 'basic' one employed the removal of all digits and stopwords (of the English language). The 'stemmer' employed the above plus the removal of twitter mentions, urls and then the stemming of each word (using the `PorterStemmer`).

We can see that the stemmer produces better results when all the other elements of the model chain (feature extractor and algorithm) stay the same:

Table 3.1: Cleaner comparison

Feature Extractor	**Algorithm**	**f1**	**Cleaner**
CV	LogisticRegression	0.781471	stemmer
CV	LogisticRegression	0.779978	basic
TFIDF	LogisticRegression	0.763283	stemmer
TFIDF	LogisticRegression	0.753713	basic

Thus, I opted for the 'stemmer' approach on the dataset. I believe it performs better because it compounds all the variations of a word to a single root. Also, it removes all the meaningless noise specific to twitter (references, hashtags, urls). These do not add any semantic meaning to the text itself.

3.2 Feature extractors

The second important element in the pipeline was the feature extractor. For this, the main options were either the `CountVectorizer` or the `TfidfVectorizer`. While the former method simply extracts the counts for the words in the dataset, the latter also adjusts the number for each of the entries depending on its frequency (*idf* = inverse document frequency).[1]

3.3 Algorithms

I then chose six algorithms that perform classification tasks. I limited myself to only six mainly because of the time aspect. The six were:

- Logistic Regression
- Linear SVC
- Perceptron
- MultinomialNB
- Decision Tree
- Random Forest

3.4 Parameter tuning

Finally, I used the `GridSearchCV` method from `sklearn` in order to obtain the optimal hyperparameters for the algorithm. In this step I joined the two feature extractors (`CountVectorizer` and `TfidfVectorizer`) with each of the six algorithms from above. I then selected a set of different parameter possibilities for the algorithms. The `GridSearchCV` tests all the possible fusions. In this case, the performance measure was the default one provided by the `GridSearchCV` method - the mean of the accuracy score across $k = 5$ stratified k-fold.

The results were as follows:[2][3]:

[1] Because of the time limit, I opted not to experiment with different parameters for the feature extractors themselves.

[2] For the full results (including the optimized parameters) consult the Appendix 'Full results'.

[3] 'LogisticRegression-l2' because it uses the 'l2' penalization norm.

Table 3.2: GridSearchCV Results

transformer_name	model_name	score
CountVectorizerDefault	LogisticRegression-l2	0.783607
CountVectorizerDefault	LinearSVC	0.781694
TfidfVectorizerDefault	LogisticRegression-l2	0.771995
TfidfVectorizerDefault	LinearSVC	0.769262
TfidfVectorizerDefault	RandomForestClassifier	0.767623
CountVectorizerDefault	RandomForestClassifier	0.767008
CountVectorizerDefault	MultinomialNB	0.753210
CountVectorizerDefault	Perceptron	0.745355
TfidfVectorizerDefault	Perceptron	0.726844
TfidfVectorizerDefault	MultinomialNB	0.726434
TfidfVectorizerDefault	DecisionTreeClassifier	0.694399
CountVectorizerDefault	DecisionTreeClassifier	0.692828

We can see that Logistic Regression and LinearSVC are clearly in the lead, with the Decision Tree performing the worst.

It also interesting to note that, in general, the `CountVectorizer` outperforms the `TfidfVectorizer` in feature extraction. `TfidfVectorizer` is better suited for the tree-based algorithms (`RandomForestClassifier` and `DecisionTreeClassifier`).

3.5 Word embeddings

In this part I will talk about the steps I undertook in experimenting with word embeddings. I used the `Doc2Vec` model from the `gensim` package in order to generate word embeddings on the `stemmer` dataset. The results were not nearly as good as the ones obtained via `CountVectorizer` and `TfidfVectorizer`:[4][5]:

[4]Results obtained via `GridSearchCV` (as in first method). Parameters for `Doc2Vec` were: `(min_count=1, window=10, size=100, sample=1e-4, negative=5, workers=7)`. When training the model, I specified `epochs=20`.

[5]MultinomialNB does not support negative values in the dataset. I did not have time to apply a Naive Bayes classifier that did.

Table 3.3: Doc2Vec Results

transformer name	model name	score
doc2vec	LinearSVC	0.628279
doc2vec	RandomForestClassifier	0.628142
doc2vec	LogisticRegression-l2	0.627937
doc2vec	Perceptron	0.627117
doc2vec	DecisionTreeClassifier	0.626913

I think that this performs so poorly because of the nature of the twitter dataset: a lot of colloquialisms, typos, emojis etc. render the model useless.

Quantitative evaluation

In this section I will write about the procedures used for measuring the performance of the models.

In order to maintain consistency across the trial I opted to simply use the default scoring method of the `GridSearchCV` method from `sklearn`. This metric is calculated as the average of the model's `score` method across the k ($k = 5$, in my case) folds. As all of the algorithms I employ are classification algorithms, this method will be the accuracy. Accuracy is defined as

$$\frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

There are of course other metrics one could use to compare the different methods. In this section I will present some other performance measures of the models I have developed. I will simply use the optimal hyperparameters obtain from the previous method to train a new instance of the model. This time I simply used a split of the dataset as the training part, and the other as the test test (4:1 ratio).

4.1 Stratified K-fold

Throughout the trials mentioned in the 'Methods' section I opted to use `GridSearchCV` with the stratified k-fold option ($k = 5$). This is because we are dealing with a multi-class (three classes, specifically: negative, neutral, positive) dataset, and we want to make sure we keep the same ratios of classes per each train/test split. In order to do this I passed the parameter `cv=5` when calling the method.

4.2 Precision. Recall. F1

Precision is defined as

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall is defined as

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 is defined in terms of precision and recall as

$$\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Below we can see the metrics obtained from this experiment. I have sorted the table by the *f1* value[1]:

Table 4.1: Precision. Recall. F1

Transformer name	**Algorithm**	**Score**	**f1**	**recall**	**precision**
CountVectorizerDefault	LogisticRegression-l2	0.783607	0.790633	0.795082	0.789830
CountVectorizerDefault	LinearSVC	0.781694	0.788430	0.792691	0.787160
TfidfVectorizerDefault	LogisticRegression-l2	0.771995	0.782844	0.788934	0.782689
TfidfVectorizerDefault	LinearSVC	0.769262	0.777532	0.783470	0.776689
TfidfVectorizerDefault	RandomForestClassifier	0.767623	0.767329	0.780738	0.773190
CountVectorizerDefault	RandomForestClassifier	0.767008	0.763568	0.772883	0.763608
CountVectorizerDefault	MultinomialNB	0.753210	0.762075	0.769809	0.759313
TfidfVectorizerDefault	Perceptron	0.726844	0.753620	0.755464	0.752151
doc2vec	LogisticRegression-l2	0.627937	0.751548	0.751366	0.751914
CountVectorizerDefault	Perceptron	0.745355	0.750380	0.751366	0.750385
doc2vec	Perceptron	0.627117	0.747575	0.753074	0.744554
doc2vec	LinearSVC	0.628279	0.742561	0.741462	0.743775
doc2vec	RandomForestClassifier	0.628142	0.737115	0.761270	0.754671
TfidfVectorizerDefault	MultinomialNB	0.726434	0.719443	0.743511	0.728980
TfidfVectorizerDefault	DecisionTreeClassifier	0.694399	0.635314	0.700137	0.682900
CountVectorizerDefault	DecisionTreeClassifier	0.692828	0.625290	0.699454	0.678784
doc2vec	DecisionTreeClassifier	0.626913	0.582839	0.673497	0.530215

[1]The 'score' is the score obtained in the 'Methods' section (average accuracy across the k folds).

4.3 Confusion matrix

Another method of evaluating a model is the confusion matrix. According to the `sklearn` documentation, "it is a matrix C [...] such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j" (Pedregosa et al. 2011).

I will provide only one example here, with the rest included in the Appendix.

This is the confusion matrix for the CountVectorizerDefault - LogisticRegression combination:

	negative	neutral	positive
negative	1659	133	44
neutral	205	369	46
positive	99	73	300

This can be read as follows:

- first row, first column: the nr of negative tweets identified as negative
- second row, third column: the nr of neutral tweets identified as positive
- third row, first column: the nr of positive tweets identified as negative

They are useful because they provide an overview of how the models have performed with respect to each of the classes.

4.4 Classification report

Another method for performing a quantitative assessment of the prediction is the classification report. This provides the precision, recall, and f1 for each of the classes in the dataset.

An example of this, for the CountVectorizerDefault-LogisticRegression combination:[2]

```
             precision    recall  f1-score   support

         -1       0.85      0.90      0.87      1836
          0       0.64      0.60      0.62       620
          1       0.77      0.64      0.70       472

avg / total       0.79      0.80      0.79      2928
```

[2]Again, you can find all the classification reports in the Appendix.

In the above, -1 is the 'negative' class, 0 is the 'neutral', and 1 is the 'positive'. I provide a discussion of this in the qualitative evaluation section.

Qualitative evaluation

In this section I will provide an overview of how the models perform from the point of view of a 'real world' application. I will attempt to discuss whether it can actually be deployed in a production environment and what sort of results we might expect. Such an evaluation is important because we need to describe a model's efficiency in more than just a number.

5.1 Class-level performance

Overall, we can see that the top combination of feature extraction and algorithm (`CountVectorizer` and `LogisticRegression`) performs quite well, with all of the metrics at approximately 78%.

However, if we examine the predictions at a class-level, we notice that it does not achieve such good results. Looking at the classification report for this combination, we observe the low numbers for the 0 class ('neutral'). We also see that the numbers are lower for the 1 ('positive') class. This is most likely because of the imbalance in number of samples in the twitter dataset. This can be read in the 'support' column and in the figure below. The 'neutral' class is even more problematic and delicate because it lies between the two other classes, in terms of subjective perspective. Thus, it can get confused with either of the two.

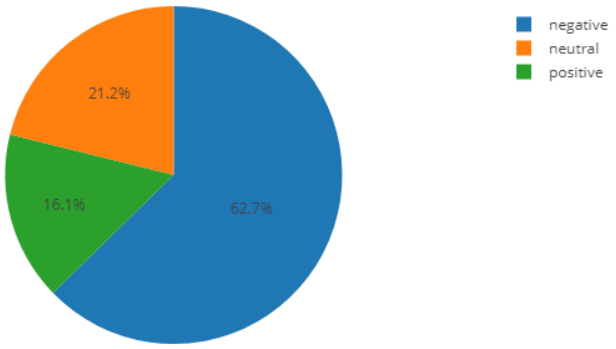

Figure 5.1: class distribution

5.2 New data

I have decided to also test the model with some new tweet-like data that I wrote myself:

```
t = pd.Series(np.array([
    "I loved my flight",
    "I hated my flight",
    "I flew today"
]))
t_clean = StemmerCleaner().fit(t).transform(t)
t_trans = trans.transform(t_clean)
model.predict(t_trans)
array([ 1, -1,  0], dtype=int64)
```

It is clear that for such simple data it can detect the sentiment easily enough.

I then searched twitter for some tweets targeted at some of the airlines mentioned in the dataset:

```
-1 When @VirginAmerica didn't have flights for Vegas. So you decide to give
   @united a try. #fail
-1 I didn't realize that if the smoke alarm goes off in an airplane bathroom,
    @VirginAmerica plane staff protocol is to bang down the door mid
```

```
   pee-stream and berate you loudly so the entire rear cabin can hear, while
   you're still buttoning up your pants and asking what's going on.
3 -1 Please do not fly @AmericanAir. When their flights arrive over 1hr 40min
   late causing me to miss my connection they REFUSED to comp a hotel. Told
   me to pay myself or sleep in the airport.
4 1 The @VirginAmerica lounge at London Heathrow beats all 'Ive ever seen.
   Table service. Full spa. Amazed!
5 1 Thank you, @AlaskaAir, for adopting and integrating @VirginAmerica humour,
   referencing the "onboard spacious, luxurious lavatories".
```

As we can see from the numbers on the sides, it seems to predict these accordingly too.

Conclusion

Throughout this project I have attempted to build a sentiment analysis model to predict whether a given line is either negative, neutral or positive in tone. This is one of the essential tasks of sentiment analysis nowadays, as companies seek to understand the public's opinions.

Starting with a relatively small dataset, I was able to construct what I think is a decently performing system for this task. While the class-level efficiency seems to be uneven, the model seems to comprehend the new tweets that I have selected for testing. This makes me optimistic of its deployment success.

Overall, the usage of `GridSearchCV` allowed me to find the optimum combination in terms of parameters for each of the algorithms. While this took a long time to compute, it yielded an application that is generally capable of determining the sentiment of a tweet.

One major drawback of this project was the dataset itself. It was uneven in terms of the class balance, and twitter lingo is not reliable for building vocabulary-centric models (colloquialisms, contractions, typos, emojis, hashtags etc.).

6.1 Further research

For further research, I think it would be interesting to develop a model from a better, bigger dataset. This could help alleviate the differences in metrics across the three labels.

Another idea would be to experiment with different features, not just those extracted by the two transformers. This could be the amount of exclamation marks, question marks, emojis or other interesting statistics per tweet.

Appendix

Full results

```
CountVectorizerDefault
LogisticRegression-l2
0.7836065573770492
{'C': 1, 'class_weight': None, 'dual': False, 'fit_intercept': True,
    'intercept_scaling': 1, 'max_iter': 2000, 'multi_class': 'ovr', 'n_jobs':
    1, 'penalty': 'l2', 'random_state': None, 'solver': 'sag', 'tol': 0.0001,
    'verbose': 0, 'warm_start': False}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'preprocessor': None, 'stop_words': None,
    'strip_accents': None, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer':
    None, 'vocabulary': None}
==============================
CountVectorizerDefault
LinearSVC
0.7816939890710383
{'C': 0.1, 'class_weight': None, 'dual': True, 'fit_intercept': True,
    'intercept_scaling': 1, 'loss': 'squared_hinge', 'max_iter': 1000,
    'multi_class': 'ovr', 'penalty': 'l2', 'random_state': None, 'tol':
    0.0001, 'verbose': 0}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'preprocessor': None, 'stop_words': None,
    'strip_accents': None, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer':
    None, 'vocabulary': None}
==============================
TfidfVectorizerDefault
```

```
LogisticRegression-l2
0.7719945355191257
{'C': 10, 'class_weight': None, 'dual': False, 'fit_intercept': True,
    'intercept_scaling': 1, 'max_iter': 2000, 'multi_class': 'ovr', 'n_jobs':
    1, 'penalty': 'l2', 'random_state': None, 'solver': 'lbfgs', 'tol':
    0.0001, 'verbose': 0, 'warm_start': False}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None, 'smooth_idf':
    True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
    'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True,
    'vocabulary': None}
==============================
TfidfVectorizerDefault
LinearSVC
0.7692622950819672
{'C': 1, 'class_weight': None, 'dual': True, 'fit_intercept': True,
    'intercept_scaling': 1, 'loss': 'squared_hinge', 'max_iter': 1000,
    'multi_class': 'ovr', 'penalty': 'l2', 'random_state': None, 'tol':
    0.0001, 'verbose': 0}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None, 'smooth_idf':
    True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
    'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True,
    'vocabulary': None}
==============================
TfidfVectorizerDefault
RandomForestClassifier
0.7676229508196721
{'bootstrap': True, 'class_weight': None, 'criterion': 'gini', 'max_depth':
    None, 'max_features': 'auto', 'max_leaf_nodes': None,
```

```
    'min_impurity_decrease': 0.0, 'min_impurity_split': None,
    'min_samples_leaf': 1, 'min_samples_split': 2,
    'min_weight_fraction_leaf': 0.0, 'n_estimators': 500, 'n_jobs': 1,
    'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start':
    False}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None, 'smooth_idf':
    True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
    'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True,
    'vocabulary': None}
==============================
CountVectorizerDefault
RandomForestClassifier
0.7670081967213115
{'bootstrap': True, 'class_weight': None, 'criterion': 'gini', 'max_depth':
    None, 'max_features': 'auto', 'max_leaf_nodes': None,
    'min_impurity_decrease': 0.0, 'min_impurity_split': None,
    'min_samples_leaf': 1, 'min_samples_split': 2,
    'min_weight_fraction_leaf': 0.0, 'n_estimators': 500, 'n_jobs': 1,
    'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start':
    False}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'preprocessor': None, 'stop_words': None,
    'strip_accents': None, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer':
    None, 'vocabulary': None}
==============================
CountVectorizerDefault
MultinomialNB
0.7532103825136612
{'alpha': 0.5, 'class_prior': None, 'fit_prior': True}
```

```
41 {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
      <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
      'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
      'ngram_range': (1, 1), 'preprocessor': None, 'stop_words': None,
      'strip_accents': None, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer':
      None, 'vocabulary': None}
42 ==============================
43 CountVectorizerDefault
44 Perceptron
45 0.7453551912568306
46 {'alpha': 0.0001, 'class_weight': None, 'eta0': 1.0, 'fit_intercept': True,
      'n_iter': 5, 'n_jobs': 1, 'penalty': None, 'random_state': 0, 'shuffle':
      True, 'verbose': 0, 'warm_start': False}
47 {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
      <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
      'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
      'ngram_range': (1, 1), 'preprocessor': None, 'stop_words': None,
      'strip_accents': None, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer':
      None, 'vocabulary': None}
48 ==============================
49 TfidfVectorizerDefault
50 Perceptron
51 0.7268442622950819
52 {'alpha': 0.0001, 'class_weight': None, 'eta0': 1.0, 'fit_intercept': True,
      'n_iter': 5, 'n_jobs': 1, 'penalty': None, 'random_state': 0, 'shuffle':
      True, 'verbose': 0, 'warm_start': False}
53 {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
      <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
      'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
      'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None, 'smooth_idf':
      True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
      'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True,
      'vocabulary': None}
54 ==============================
```

```
55 TfidfVectorizerDefault
56 MultinomialNB
57 0.7264344262295082
58 {'alpha': 0.1, 'class_prior': None, 'fit_prior': True}
59 {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None, 'smooth_idf':
    True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
    'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True,
    'vocabulary': None}
60 ==============================
61 TfidfVectorizerDefault
62 DecisionTreeClassifier
63 0.6943989071038251
64 {'class_weight': None, 'criterion': 'gini', 'max_depth': 19, 'max_features':
    None, 'max_leaf_nodes': None, 'min_impurity_split': None,
    'min_samples_leaf': 1, 'min_samples_split': 490,
    'min_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': None,
    'splitter': 'best'}
65 {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
    <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
    'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
    'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None, 'smooth_idf':
    True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
    'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True,
    'vocabulary': None}
66 ==============================
67 CountVectorizerDefault
68 DecisionTreeClassifier
69 0.6928278688524591
70 {'class_weight': None, 'criterion': 'gini', 'max_depth': 19, 'max_features':
    None, 'max_leaf_nodes': None, 'min_impurity_split': 1e-07,
    'min_samples_leaf': 1, 'min_samples_split': 450,
```

```
   'min_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': None,
   'splitter': 'best'}
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype':
   <class 'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content',
   'lowercase': True, 'max_df': 1.0, 'max_features': None, 'min_df': 1,
   'ngram_range': (1, 1), 'preprocessor': None, 'stop_words': None,
   'strip_accents': None, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer':
   None, 'vocabulary': None}
==============================
```

Confusion matrices

CountVectorizerDefault-DecisionTreeClassifier

	negative	neutral	positive
negative	1767	13	56
neutral	548	30	42
positive	213	8	251

CountVectorizerDefault-LinearSVC

	negative	neutral	positive
negative	1652	135	49
neutral	205	366	49
positive	98	71	303

CountVectorizerDefault-LogisticRegression-l2

	negative	neutral	positive
negative	1659	133	44
neutral	205	369	46
positive	99	73	300

CountVectorizerDefault-MultinomialNB

	negative	neutral	positive
negative	1647	130	59
neutral	250	305	65
positive	101	69	302

CountVectorizerDefault-Perceptron

	negative	neutral	positive
negative	1557	165	114
neutral	199	329	92
positive	87	71	314

CountVectorizerDefault-RandomForestClassifier

	negative	neutral	positive
negative	1672	120	44
neutral	249	324	47
positive	132	73	267

doc2vec-DecisionTreeClassifier

	negative	neutral	positive
negative	1761	0	75
neutral	589	0	31
positive	261	0	211

doc2vec-LinearSVC

	negative	neutral	positive
negative	1523	213	100
neutral	207	345	68
positive	84	85	303

doc2vec-LogisticRegression-l2

	negative	neutral	positive
negative	1547	206	83
neutral	211	348	61
positive	83	84	305

doc2vec-Perceptron

	negative	neutral	positive
negative	1601	150	85
neutral	236	309	75
positive	109	68	295

doc2vec-RandomForestClassifier

	negative	neutral	positive
negative	1749	56	31
neutral	365	223	32
positive	167	48	257

TfidfVectorizerDefault-DecisionTreeClassifier

	negative	neutral	positive
negative	1745	23	68
neutral	529	50	41
positive	208	9	255

TfidfVectorizerDefault-LinearSVC

	negative	neutral	positive
negative	1662	132	42
neutral	224	347	49
positive	102	85	285

TfidfVectorizerDefault-LogisticRegression-l2

	negative	neutral	positive
negative	1673	125	38
neutral	221	354	45
positive	101	88	283

TfidfVectorizerDefault-MultinomialNB

	negative	neutral	positive
negative	1726	77	33
neutral	340	230	50
positive	188	63	221

TfidfVectorizerDefault-Perceptron

	negative	neutral	positive
negative	1576	185	75
neutral	214	335	71
positive	91	80	301

TfidfVectorizerDefault-RandomForestClassifier

	negative	neutral	positive
negative	1718	84	34
neutral	277	308	35
positive	149	63	260

Classification reports

CountVectorizerDefault-DecisionTreeClassifier

```
             precision    recall  f1-score   support

         -1       0.70      0.96      0.81      1836
          0       0.59      0.05      0.09       620
          1       0.72      0.53      0.61       472

avg / total       0.68      0.70      0.63      2928
```

CountVectorizerDefault-LinearSVC

```
             precision    recall  f1-score   support

         -1       0.85      0.90      0.87      1836
          0       0.64      0.59      0.61       620
          1       0.76      0.64      0.69       472

avg / total       0.79      0.79      0.79      2928
```

CountVectorizerDefault-LogisticRegression-l2

```
             precision    recall  f1-score   support

         -1       0.85      0.90      0.87      1836
          0       0.64      0.60      0.62       620
          1       0.77      0.64      0.70       472

avg / total       0.79      0.80      0.79      2928
```

CountVectorizerDefault-MultinomialNB

```
             precision    recall  f1-score   support

         -1       0.82      0.90      0.86      1836
          0       0.61      0.49      0.54       620
          1       0.71      0.64      0.67       472

avg / total       0.76      0.77      0.76      2928
```

CountVectorizerDefault-Perceptron

```
             precision    recall  f1-score   support

         -1       0.84      0.85      0.85      1836
          0       0.58      0.53      0.56       620
          1       0.60      0.67      0.63       472

avg / total       0.75      0.75      0.75      2928
```

CountVectorizerDefault-RandomForestClassifier

```
             precision    recall  f1-score   support

         -1       0.81      0.91      0.86      1836
          0       0.63      0.52      0.57       620
          1       0.75      0.57      0.64       472

avg / total       0.76      0.77      0.76      2928
```

doc2vec-DecisionTreeClassifier

```
             precision    recall  f1-score   support

         -1       0.67      0.96      0.79      1836
          0       0.00      0.00      0.00       620
          1       0.67      0.45      0.53       472

avg / total       0.53      0.67      0.58      2928
```

doc2vec-LinearSVC

```
             precision    recall  f1-score   support

         -1       0.84      0.83      0.83      1836
          0       0.54      0.56      0.55       620
          1       0.64      0.64      0.64       472

avg / total       0.74      0.74      0.74      2928
```

doc2vec-LogisticRegression-l2

```
             precision    recall  f1-score   support

         -1       0.84      0.84      0.84      1836
          0       0.55      0.56      0.55       620
          1       0.68      0.65      0.66       472

avg / total       0.75      0.75      0.75      2928
```

doc2vec-Perceptron

```
             precision    recall  f1-score   support

         -1       0.82      0.87      0.85      1836
          0       0.59      0.50      0.54       620
          1       0.65      0.62      0.64       472

avg / total       0.74      0.75      0.75      2928
```

doc2vec-RandomForestClassifier

```
             precision    recall  f1-score   support

         -1       0.77      0.95      0.85      1836
          0       0.68      0.36      0.47       620
          1       0.80      0.54      0.65       472

avg / total       0.75      0.76      0.74      2928
```

TfidfVectorizerDefault-DecisionTreeClassifier

```
             precision    recall  f1-score   support

         -1       0.70      0.95      0.81      1836
          0       0.61      0.08      0.14       620
          1       0.70      0.54      0.61       472

avg / total       0.68      0.70      0.64      2928
```

TfidfVectorizerDefault-LinearSVC

```
             precision    recall  f1-score   support

         -1       0.84      0.91      0.87      1836
          0       0.62      0.56      0.59       620
          1       0.76      0.60      0.67       472

avg / total       0.78      0.78      0.78      2928
```

TfidfVectorizerDefault-LogisticRegression-l2

```
             precision    recall  f1-score   support

         -1       0.84      0.91      0.87      1836
          0       0.62      0.57      0.60       620
          1       0.77      0.60      0.68       472

avg / total       0.78      0.79      0.78      2928
```

TfidfVectorizerDefault-MultinomialNB

```
             precision    recall  f1-score   support

         -1       0.77      0.94      0.84      1836
          0       0.62      0.37      0.46       620
          1       0.73      0.47      0.57       472

avg / total       0.73      0.74      0.72      2928
```

TfidfVectorizerDefault-Perceptron

```
             precision    recall  f1-score   support

         -1       0.84      0.86      0.85      1836
          0       0.56      0.54      0.55       620
          1       0.67      0.64      0.66       472

avg / total       0.75      0.76      0.75      2928
```

TfidfVectorizerDefault-RandomForestClassifier

```
             precision    recall  f1-score   support

         -1       0.80      0.94      0.86      1836
          0       0.68      0.50      0.57       620
          1       0.79      0.55      0.65       472

avg / total       0.77      0.78      0.77      2928
```

Bibliography

crowdflower user at Kaggle, Twitter us airline sentiment. Available at: https://www.kaggle.com/crowdflower/twitter-airline-sentiment [Accessed June 5, 2018].

Google NGrams, Google ngrams search for 'sentiment analysis' and 'opinion mining'. Available at: https://books.google.com/ngrams/graph?content=sentiment+analysis%2Copinion+mining&year_start=1990&year_end=2008&corpus=15&smoothing=3&share=&direct_url=t1%3B%2Csentiment%20analysis%3B%2Cc0%3B.t1%3B%2Copinion%20mining%3B%2Cc0 [Accessed June 5, 2018].

Google Scholar, Google scholar search for 'sentiment analysis'. Available at: https://scholar.google.dk/scholar?hl=en&as_sdt=0%2C5&q=sentiment+analysis&btnG= [Accessed June 5, 2018].

Pang, B. & Lee, L., 2008. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2), pp.1–135. Available at: http://dx.doi.org/10.1561/1500000011.

Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, pp.2825–2830.