In this classification task, we are given 2 sets of news articles and a label for each article based on its category: natural disasters, attacks, health, etc.... We trained three different machine learning algorithms on a test set (920 examples), compared their accuracy on a validation set (200 examples), and measure our best result with a test set of 680 examples.

Feature Extraction:

The first task was to extract a feature vector for each sample text. To do so we used the feature extraction module from Scikit Learn. We used the CountVectorizer class that is based on occurrence counting. This returned the term-document matrix based on the training corpus of all articles (*scikit-learn documentation*).

A few parameters needed to be tested for this step. First, we consider to lowercase all the words in an article. In order to generalize better, all the tokens were lowercased.

Second, we decided to remove punctuation markers from our data since we safely assume that they don't affect the type of news article.

Moreover, the list of english stop words form the nltk library was used to remove stop words from our data, making it less “noisy”. We tested the algorithms with those stop words in our corpus and the results were very similar. This showcase the fact that stop words are useless for the classification task and are slightly increasing the running time since more parameters are considered.

After fixing all that, we also considered to lemmatize our tokens or not. To do so the nltk WordNet lemmatizer along with the nltk Treebank tag set were used. The part of speech (POS) tag was used as a parameter to the lemmatizer for each word. Indeed, the default behavior of this lemmatizer considers all words as nouns, so “has” wouldn't have changed (to “have” in this case) if no POS tag indicated that this is a verb. Results with and without lemmatization can be found below.

Eventually, the scikit learn feature extraction module is based on N-gram features, so we considered both unigram training, and unigram + bigram training. Such results can be found below.

Algorithms:

The next step of this task is to train different algorithms to learn how to classify this data. We compared three different algorithms using the scikit learn library again: Logistic regression, SVM, and Multinomial Naïve Bayes. Here again a few parameters were tested.

In the case of logistic regression and SVM, a penalty parameter C is used for the error term. It represents the inverse of regularization strength. This basically tells the algorithm how tolerent we are with missclassified training points. Smaller values specify stronger regularization. Being too strict about errors may lead in overfitting, while being too flexible may cause our algorithm to generalize too much and not learn anything. We considered a wide range of values: from 10^{-5} to 10^{5} by jumps of 10^{i}. Similarly, the Naïve Bayes algorithm has an α parameter that accounts for features not present in the learning samples and prevent zero probabilities in further computations (*scikit-learn documentation*). Having $\alpha=1$ is the same as Laplace smoothing for example. Here again we considered a wide range of values: from 10^{-5} to 10^{5} by jumps of 10^{i}.

Results:

We trained the three different algorithms on a training set of 920 samples and measured the validation error based on a validation set of 200 samples. This development set was used to decide which set of parameters yields the best accuracy for each algorithm. The tables below represent the validation accurary in percentage for each algorithm in four cases: Lemmatization = True or False and n-gram = 1 or 2. Note that in all cases, lowecase = true, remove_punctuation = true, and remove_stop_words = true.

(1) LEMMATIZE=False | N_GRAM = 1

C | Alpha	Logistic Regression	SVM	Naïve Bayes
0.000010	0.545000	0.300000	0.665000
0.000100	0.735000	0.300000	0.695000
0.001000	0.825000	0.300000	0.710000
0.010000	0.900000	0.300000	0.740000
0.100000	0.895000	0.300000	0.775000
1.000000	**0.905000**	0.300000	0.780000
10.00000	0.890000	0.600000	**0.805000**
100.0000	0.880000	**0.925000**	0.665000
1,000.000	0.870000	0.900000	0.520000
10,000.00	0.875000	0.900000	0.360000
100,000.0	0.875000	0.900000	0.300000

Table 01: Validation score (%) for Logistic Regression, SVM, and Naïve Bayes without lemmatization and unigram features for different C and alpha values

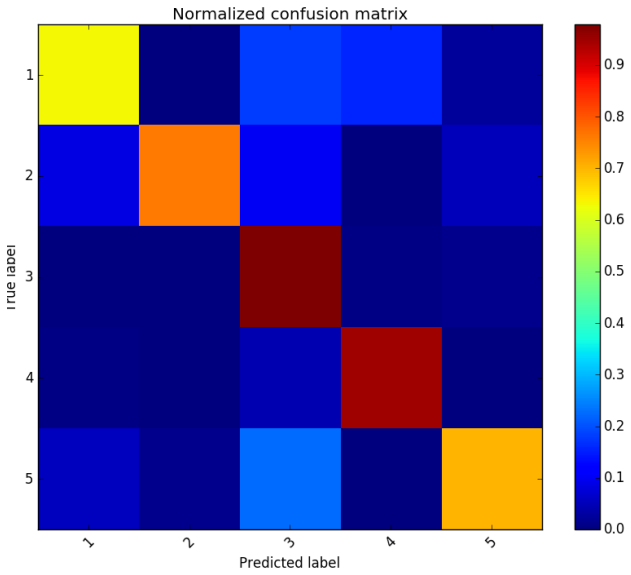

Figure 01: Normalized confusion matrix for the test set accuracy using SVM(C=100), lemmatizatize=false and unigram features.

(2) LEMMATIZE=True | N_GRAM = 1

C | Alpha	Logistic Regression	SVM	Naïve Bayes
0.000010	0.490000	0.300000	0.680000
0.000100	0.760000	0.300000	0.685000
0.001000	0.825000	0.300000	0.700000
0.010000	0.865000	0.300000	0.715000
0.100000	0.865000	0.300000	0.765000
1.000000	**0.875000**	0.320000	0.775000
10.00000	0.865000	0.680000	**0.795000**
100.0000	0.865000	**0.895000**	0.630000
1,000.000	0.850000	0.880000	0.490000
10,000.00	0.850000	0.880000	0.350000
100,000.0	0.850000	0.880000	0.300000

Table 02: Validation score (%) for Logistic Regression, SVM, and Naïve Bayes with lemmatization and unigram features for different C and alpha values.

(3) LEMMATIZE=True | N_GRAM = 2

Logistic Regression	SVM	Naïve Bayes
0.505000	0.300000	0.710000
0.760000	0.300000	0.720000
0.830000	0.300000	0.735000
0.860000	0.300000	0.750000
0.875000	0.300000	0.755000
0.885000	0.300000	**0.795000**
0.880000	0.320000	0.730000
0.860000	0.685000	0.615000
0.860000	**0.885000**	0.490000
0.860000	0.870000	0.350000
0.860000	0.870000	0.300000

Table 03: Validation score (%) for Logistic Regression, SVM, and Naïve Bayes with lemmatization and unigram + bigram features for different C and alpha values.

(4) LEMMATIZE=False | N_GRAM = 2

C | Alpha	Logistic Regression	SVM	Naïve Bayes
0.000010	0.550000	0.300000	0.705000
0.000100	0.745000	0.300000	0.705000
0.001000	0.820000	0.300000	0.725000
0.010000	0.880000	0.300000	0.750000
0.100000	0.895000	0.300000	0.760000
1.000000	0.900000	0.300000	**0.775000**
10.00000	**0.905000**	0.300000	0.760000
100.0000	0.900000	0.625000	0.655000
1,000.000	0.885000	**0.910000**	0.515000
10,000.00	0.880000	0.895000	0.355000
100,000.0	0.880000	0.895000	0.300000

Table 04: Validation score (%) for Logistic Regression, SVM, and Naïve Bayes without lemmatization and unigram+bigram features for different C and alpha values.

Note that in all cases the Naïve Bayes algorithm is the worst predictor. This is essentially due to the very wrong independance assumption made in this model.
For optimal C values the Logisitc Regression (LR) scores are not far behind SVM. However we note that the learning curve is more drastic in the SVM case, which showcase how this regularization variable C is important in SVM. On the other hand, the LR scores are more gradually increasing towards an optimal value, which makes the LR algorithm better for less optimal C values.
Overall, we also found that having lemmatization turned on is slightly worst than no lemmatization at all (tables 2&3 vs tables 1&4). This may be due to the fact that the tense used in the news article sentences could be an indicator of its class.
Similarly, having bigram features reduces very slightly our prediction accuracy (tables 1&2 vs tables 3&4). Since the difference is very small, this could be specific to this data, or it could be the start of overfitting since we only train on 920 samples.

Eventually, we can see that the best validation result is achieved with the SVM algorithm (with C=100) when lemmatization is turned off and we use unigram features (table 01).
The test score for this specific set of parameters is **0.811764705882** and the confusion matrix can be found in Figure 01. We can conclude that class 3 is very well classified (around 90% accuracy) while classes 1 and 5 are slightly harder to properly classify (around 65% accuracy). This may be due to the distribution of each class in the training data. Indeed, the training data could very well have a lot of class 3 examples, and fewer class 1 and 5 examples. A way to make sure that this doen't happen would be to draw an equal number of samples from each class as our training data, and put the rest in our validation or test set. We would then expect our confusion matrix to be more “balanced”.