MASTER IN BUSINESS ANALYTICS 2019/2020

Summative assessment for the module

Natural Language Analysis

Z0151576

Durham 2020

Table of Contents

Exploring the Data

The news articles dataset organized in two separate files – with 40,000 instances of training and 4000 items of test datasets. Both datasets structured in three columns, where first contains labels of four different news topics (World, Business, Sports, Sci/Tech), second and third columns represent article title and text respectively. Generally, news headlines contain information for an intuitive understanding of the topic it was written about. However, to avoid ambiguity and vagueness, the combination of article title and text has been used in this work.

Datasets contain no zero and missing values, and the class attribute equally represented within each category. Additionally, the average number of words (38.43) and average text length (236,63) are approximately equally distributed within each news topic. A total number of words in the training dataset is 1,537,305, the number of unique words is 104,855.

Data Cleaning

The data cleaning process includes lowercasing each word and removing punctuations, non-alphabetical symbols, stopwords and one-character words. After defining and applying a special cleaning function, the total number of words decreased significantly from 1,537,305 to 982,083, and the number of unique words in the corpus fell from 104,855 to 53,202.

TF- IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a bag-of-words technique, which converts word sequences into numerical representation – vectors. The vectorization of the news articles dataset on the training dataset performed with TfidfVectorizer () function initially with default arguments (ngram_range = (1,1), min_df =1, max_df = 1, max_features = None):

```
print('Vocabulary size:', len(vectorizer.vocabulary_))
print('Vocabulary content:\n {}'.format(vectorizer.vocabulary_))
```

```
Vocabulary size: 53202
Vocabulary content:
 {'wizards': 52271, 'guard': 19561, 'blake': 5070, 'week': 51560, 'ap': 2285, 'washington': 51305, 'point': 34934,
'steve': 44895, 'miss': 29130, 'first': 16574, 'month': 29483, 'season': 41504, 'injuring': 22655, 'ankle': 1999, '
pickup': 34433, 'game': 18129, 'brief': 5978, 'hp': 21454, 'acquires': 363, 'synstar': 46188, 'move': 29762, 'desig
ned': 11915, 'help': 20552, 'better': 4684, 'compete': 9171, 'europe': 14977, 'hewlettpackard': 20694, 'said': 4065
0, 'acquired': 359, 'plc': 34787, 'ukbased': 49290, 'technology': 46668, 'services': 41979, 'firm': 16561, 'million
': 28900, 'us': 50089, 'airways': 1030, 'pilots': 34508, 'vote': 50991, 'salary': 40710, 'reduction': 38549, 'voted
': 50993, 'approve': 2423, 'new': 30794, 'labor': 25207, 'agreement': 887, 'yesterday': 52880, 'reduce': 38544, 'sa
laries': 40709, 'percent': 34019, 'save': 41021, 'airline': 1003, 'year': 52818, 'radiation': 37776, 'risks': 39880
, 'need': 30596, 'updating': 49976, 'uk': 49288, 'panel': 33355, 'examining': 15149, 'says': 41058, 'official': 319
32, 'estimates': 14892, 'dangers': 11007, 'health': 20380, 'may': 27977, 'wide': 51926, 'mark': 27645, 'substantial
ly': 45474, 'presidents': 35643, 'fate': 15982, 'line': 26195, 'venezuela': 50497, 'caracas': 7068, 'partisans': 33
545, 'sides': 42724, 'calling': 6815, 'polarized': 34970, 'important': 22096, 'election': 14022, 'venezuelas': 5050
0, 'history': 20972, 'presidential': 35638, 'recall': 38307, 'referendum': 38597, 'today': 47935, 'determine': 1200
1, 'course': 10193, 'democracy': 11688, 'could': 10105, 'buffet': 6309, 'world': 52427, 'oil': 32085, 'prices': 357
38, 'campaigns': 6879, 'also': 1462, 'utterly': 50267, 'convinced': 9825, 'win': 52070, 'australia': 3280, 'virgin'
: 50803, 'blue': 5248, 'increases': 22275, 'fuel': 17864, 'surcharge': 45820, 'hikes': 20873, 'sydney': 46123, 'aus
tralian': 3285, 'announced': 2028, 'sharply': 42261, 'increasing': 22276, 'imposed': 22106, 'compensate': 9166, 'ri
```

Figure 1. TF-IDF vocabulary content

Following Figure 1, vocabulary size is equal to 53,202 and matches the number of unique words in the corpus. Then, changing min_df and max_df parameters revealed that only the four most frequent terms (new, reuters, said, us) appear in 10% of the corpus, and therefore these parameters can remain the default values. The next step is to train machine learning algorithms such as Linear SVM, Naïve Bayes, Logistic Regression and Random Forest with different numbers of ngram_range (unigram, bigram, trigram), vocabulary size (7000, 15000) and hyperparameters with 3-fold grid and random search techniques and “accuracy” performance measure. Figure 2 (a,b) illustrates results sorted by the average validation score (column 2). The highest score of about 90,8% obtained with Linear SVM algorithm, unigram and bigram sequence of

words, size of vocabulary equal to 15,000 and regularization hyperparameter (C) equal to 0.5.

	params	mean_test_score	mean_train_score	rank_test_score
16	{'clf': LinearSVC(C=0.5, 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), 'clf__C': 0.5, 'vec__max_features': 15000, 'vec__ngram_range': (1, 2)}	0.907825	0.967550	1
17	{'clf': LinearSVC(C=0.5, 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), 'clf__C': 0.5, 'vec__max_features': 15000, 'vec__ngram_range': (1, 3)}	0.907200	0.967150	2
15	{'clf': LinearSVC(C=0.5, 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), 'clf__C': 0.5, 'vec__max_features': 15000, 'vec__ngram_range': (1, 1)}	0.906625	0.967262	3
82	{'clf': LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=500, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False), 'clf__C': 1.0, 'clf__penalty': 'l2', 'clf__solver': 'newton-cg', 'vec__max_features': 15000, 'vec__ngram_range': (1, 2)}	0.906200	0.946075	4
88	{'clf': LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=500, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False), 'clf__C': 1.0, 'clf__penalty': 'l2', 'clf__solver': 'lbfgs', 'vec__max_features': 15000, 'vec__ngram_range': (1, 2)}	0.906125	0.946075	5

a

	params	mean_test_score	mean_train_score	rank_test_score
6	{'vec__ngram_range': (1, 2), 'vec__max_features': 15000, 'r_f__n_estimators': 400, 'r_f__min_samples_split': 10, 'r_f__min_samples_leaf': 1, 'r_f__max_features': 'auto', 'r_f__max_depth': None, 'r_f__criterion': 'gini'}	0.863800	0.997763	1
3	{'vec__ngram_range': (1, 1), 'vec__max_features': 15000, 'r_f__n_estimators': 100, 'r_f__min_samples_split': 10, 'r_f__min_samples_leaf': 2, 'r_f__max_features': 'auto', 'r_f__max_depth': None, 'r_f__criterion': 'gini'}	0.859325	0.931350	2
7	{'vec__ngram_range': (1, 3), 'vec__max_features': 7000, 'r_f__n_estimators': 200, 'r_f__min_samples_split': 10, 'r_f__min_samples_leaf': 2, 'r_f__max_features': 'auto', 'r_f__max_depth': None, 'r_f__criterion': 'entropy'}	0.857075	0.936613	3
9	{'vec__ngram_range': (1, 2), 'vec__max_features': 7000, 'r_f__n_estimators': 200, 'r_f__min_samples_split': 2, 'r_f__min_samples_leaf': 1, 'r_f__max_features': 'auto', 'r_f__max_depth': None, 'r_f__criterion': 'entropy'}	0.856975	0.999913	4
5	{'vec__ngram_range': (1, 1), 'vec__max_features': 7000, 'r_f__n_estimators': 200, 'r_f__min_samples_split': 10, 'r_f__min_samples_leaf': 1, 'r_f__max_features': 'sqrt', 'r_f__max_depth': 60, 'r_f__criterion': 'gini'}	0.837325	0.934212	5

b

Figure 2. Top five results for Linear SVM, Naïve Bayes, Logistic Regression (a) and Random Forest (b) algorithms

Evaluation on the test dataset

Transforming the test dataset with the same TF-IDF vectorizer and applying trained earlier Linear SVM model, the accuracy score 91,2% has been achieved.

	precision	recall	f1-score	support
world	0.93	0.91	0.92	1000
sports	0.95	0.99	0.97	1000
business	0.88	0.87	0.88	1000
sci/tech	0.88	0.88	0.88	1000
accuracy			0.91	4000
macro avg	0.91	0.91	0.91	4000
weighted avg	0.91	0.91	0.91	4000

Figure 4. Classification report for the test dataset

According to Figure 4 and Figure 5, the sports news topic is the most accurately classified with f1-score equal to 0.97. More classification errors of the model occurred on the business and sci/tech topics, both performing with 0.88 f1-score.

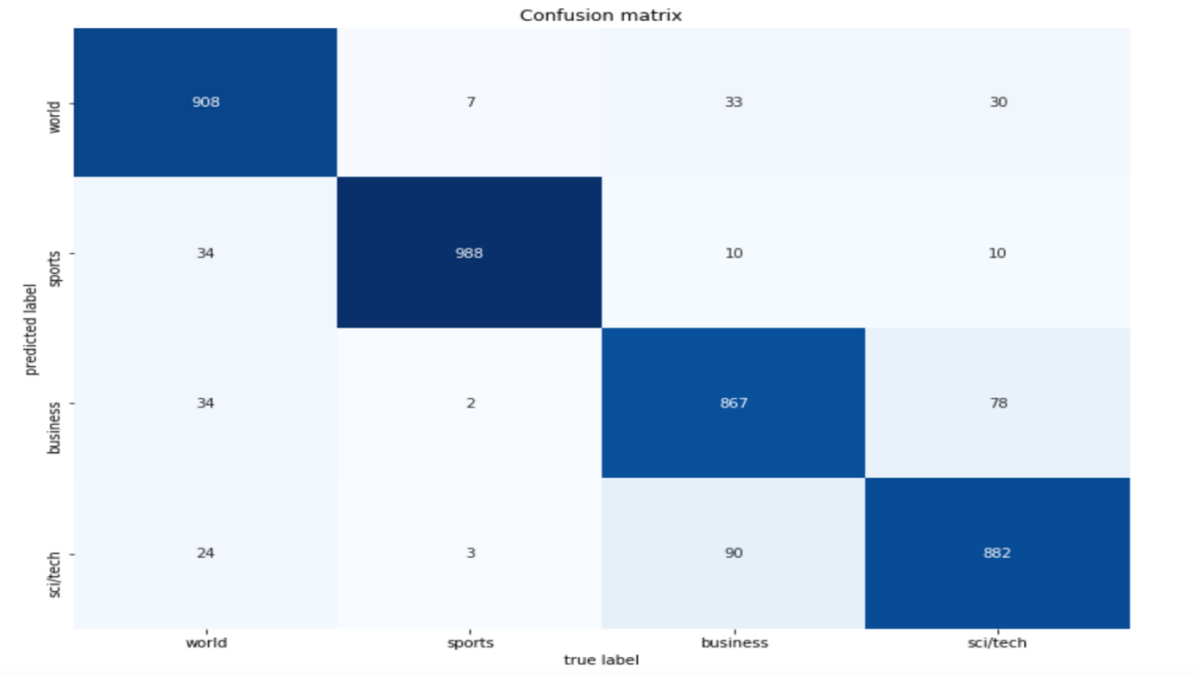

Figure 5. Confusion matrix for the test dataset

Word2vec

Word2vec is a word embedding technique with the general assumption that words in the corpus share similar contexts and meaning and therefore, might have similar vector representation (Mikolov et al., 2013).

In this work, Word2vec technique was implemented with the Word2Vec () module of the Gensim library. Since the overall size of the corpus is relatively small (53202 unique words) and the dataset was previously cleaned from stopwords, parameters such as size = 100, window = 5, min_count = 1 and sg =1 (Skip-gram architecture) were selected for text transformation.

Figure 6 shows the 100-dimensional vector representation and the most associated with ‘computer’ words.

```
w_model_sg.wv['computer']
```

```
array([ 0.06388441, -0.08133382, -0.13676958,  0.10582004,  0.06654062,
       -0.04945404,  0.07931393,  0.08044662, -0.19894224, -0.04958534,
        0.08718207, -0.20376472, -0.02995504, -0.11479849, -0.09902271,
        0.08149272, -0.02656364, -0.00617725, -0.01187406, -0.13356061,
        0.05705734, -0.0504782 ,  0.21819203,  0.14567892,  0.11150423,
        0.22995125, -0.13097145, -0.04231865, -0.05635202,  0.01697813,
        0.09151268,  0.08142684,  0.09050979, -0.07395874,  0.02765094,
       -0.10545062, -0.07931811,  0.00223664, -0.00463745, -0.06310347,
        0.019389  ,  0.17977041,  0.17574738,  0.04723195,  0.0760856 ,
       -0.00848501,  0.12051502,  0.15135323, -0.02821335, -0.07651005,
       -0.05400637, -0.06249378, -0.03090185, -0.1311977 ,  0.13637047,
       -0.01506626, -0.06128463,  0.07527307,  0.03063625, -0.02353359,
       -0.135863  ,  0.11806294,  0.00530435, -0.06856249,  0.13208763,
        0.02626646, -0.05626931, -0.24905108, -0.03653201, -0.05046824,
       -0.01047773, -0.00566028,  0.03844219, -0.16854328, -0.18983673,
       -0.00351789, -0.03189205, -0.00683244, -0.07032195, -0.11395986,
        0.15294251, -0.10210698,  0.05696347,  0.02076365, -0.00205236,
       -0.05633365,  0.00996392,  0.03992171,  0.09938776,  0.10060339,
        0.06184255,  0.03764948,  0.06169024, -0.07659867,  0.02831339,
       -0.15962256, -0.02482823, -0.03540504, -0.31427738,  0.04544675],
      dtype=float32)
```

```
w_model_sg.wv.most_similar('computer')
```

```
[('computers', 0.7709169387817383),
 ('powerbook', 0.7192236185073853),
 ('hacker', 0.7096250057220459),
 ('personal', 0.7085227966308594),
 ('ibook', 0.6991468667984009),
 ('macintosh', 0.6917257308959961),
 ('aapl', 0.6909997463226318),
 ('pc', 0.6901720762252808),
 ('cupertino', 0.6838955879211426),
 ('accessed', 0.6793504953384399)]
```

Figure 6. Vectoral representation of the ‘computer’ word

One common approach to performing classification with ML algorithms is to calculate the average feature vector for each list of words in the training dataset (Mikolov et al., 2013). As for the TF-IDF method, Linear SVM, Logistic Regression and Random

Forest algorithms were estimated and tuned through a 3-fold grid and random search tools with ‘accuracy’ scoring. Naïve Bayes was excluded due to negative values in the training dataset. Figure 7 illustrates results, ranked by the average test score. Although the highest score of about 88,8% achieved with the Random Forest algorithm, this model overfitted with accuracy 99,99% on the training dataset. Thus, Logistic Regression algorithm with accuracy scores 88,6% on validation, and 88,8% on training datasets selected as the best model.

	params	mean_test_score	mean_train_score	rank_test_score
9	{'clf': LogisticRegression(C=500, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=9000, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False), 'clf__C': 500, 'clf__solver': 'liblinear'}	0.886075	0.888675	1
3	{'clf': LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=7000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0), 'clf__C': 50}	0.886000	0.888600	2
4	{'clf': LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=7000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0), 'clf__C': 100}	0.885950	0.888625	3
2	{'clf': LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=7000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0), 'clf__C': 10}	0.885750	0.888575	4
5	{'clf': LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=7000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0), 'clf__C': 500}	0.885600	0.888475	5

a

	params	mean_test_score	mean_train_score	rank_test_score
2	{'n_estimators': 400, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 30, 'criterion': 'gini'}	0.890225	0.999400	1
7	{'n_estimators': 250, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 90, 'criterion': 'entropy'}	0.889525	0.998237	2
5	{'n_estimators': 400, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 30, 'criterion': 'gini'}	0.889250	0.995250	3
3	{'n_estimators': 250, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 60, 'criterion': 'entropy'}	0.889025	0.977188	4
0	{'n_estimators': 250, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': None, 'criterion': 'gini'}	0.888850	0.990925	5

b

Figure 7. Top five results for Linear SVM, Logistic Regression (a) and Random Forest (b) algorithms

Evaluation on the test dataset

Applying the Logistic Regression algorithm with best-tuned hyperparameters on the test dataset, the accuracy score 88.7% was achieved. According to the classification report and confusion matrix in Figure 8, the best result was achieved by sports news topic with 967 from 1000 correctly classified articles. The most classification errors (160) were generated by Business topic.

	precision	recall	f1-score	support
world	0.91	0.89	0.90	1000
sports	0.94	0.97	0.95	1000
business	0.85	0.84	0.84	1000
sci/tech	0.85	0.85	0.85	1000
accuracy			0.89	4000
macro avg	0.89	0.89	0.89	4000
weighted avg	0.89	0.89	0.89	4000

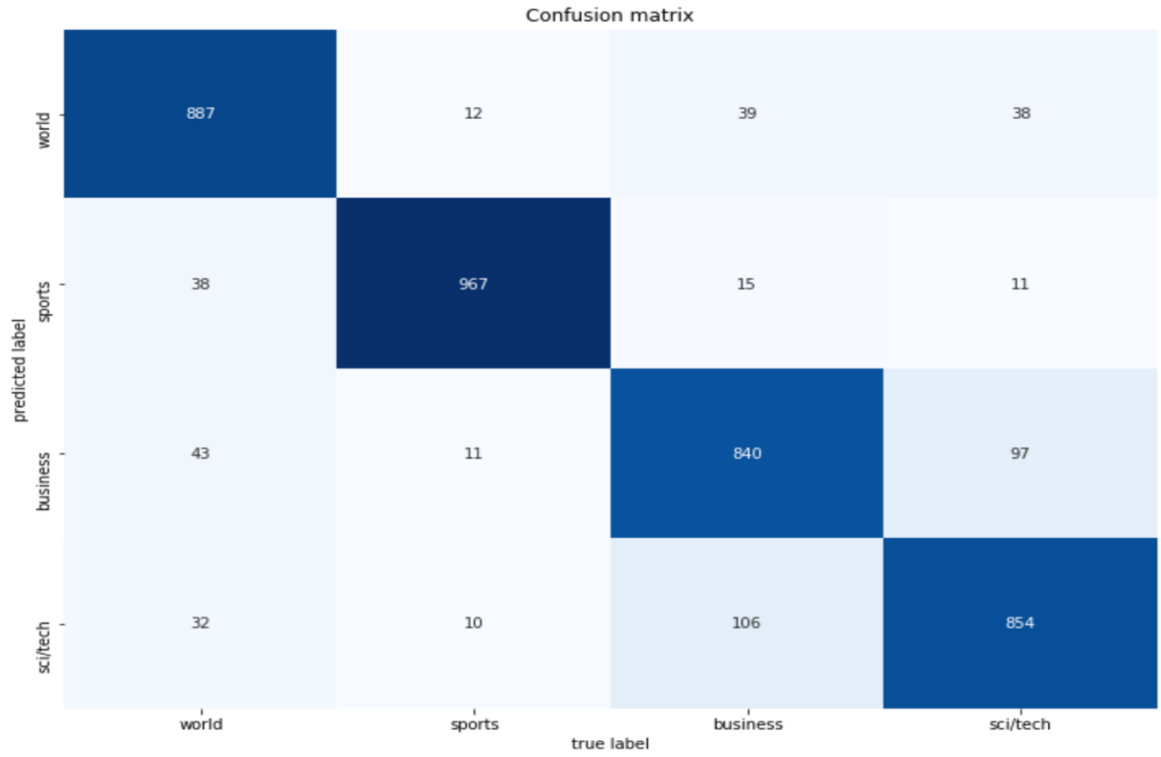

Figure 8. Performance on the test dataset

Deep Learning Model

Keras embedding layer was built with a pre-trained from the Word2vec model weight matrix, which capable of mapping encoded with integers input data to the pre-trained weights.

Model architecture

In the present work, news articles classification task was performed with C-LSTM Neural Network model, which consists of both CNN and LSTM layers (Figure 9).

Layer (type)	Output Shape	Param #
embedding_16 (Embedding)	(None, 90, 100)	5320300
dropout_42 (Dropout)	(None, 90, 100)	0
conv1d_15 (Conv1D)	(None, 86, 256)	128256
max_pooling1d_14 (MaxPooling	(None, 43, 256)	0
dropout_43 (Dropout)	(None, 43, 256)	0
lstm_32 (LSTM)	(None, 64)	82176
dropout_44 (Dropout)	(None, 64)	0
dense_19 (Dense)	(None, 16)	1040
dense_20 (Dense)	(None, 4)	68

Total params: 5,531,840
Trainable params: 211,540
Non-trainable params: 5,320,300

Figure 9. C-LSMT model architecture

A convolutional layer followed by a max-pooling layer capable of extracting sentences from a high-dimensional embedding matrix and capture short-term patterns in the text

(Yoon, 2014, Kalchbrenner et al., 2014). An LSTM layer with special operating as memory elements gating units learns long-range dependencies between words and avoids the vanishing gradient problem (Sundermeyer et al., 2012). Thus, capturing both local and global text features, combined C-LSTM architecture might outperform individual CNN and LSTM models (Zhou et al., 2015). Furthermore, dropout was applied before and after CNN and LSTM layers to prevent co-adaptation of hidden units and model overfitting. Finally, a fully connected Dense layer with 16 units interprets features before feeding to the Dense output layer.

Hyperparameters

The following parameters were applied to train C-LSTM model:

- "categorical_crossentropy" loss function – because of the multiclass nature of the classification task.
- "softmax" output activation function, as it accommodates multiple classes into the normalized probabilistic vector.
- "accuracy" performance measure, since classes in the dataset well balanced.
- "batch_size" equal to 128, to speed the model training up.

Other hyperparameters such as filters number, kernel size and optimizer were tuned with Talos library (Autonomio Talos [Computer Software], 2019). Furthermore, the model fitted with 10 training epochs and 20% of the validation data. Figure 10 illustrates grid-search results provided with Talos tool, sorted descending by validation accuracy.

	round_epochs	val_loss	val_accuracy	loss	accuracy	filters	kernel_size	optimizer
17	10	0.286478	0.899000	0.269526	0.905781	256	8	Nadam
16	10	0.298538	0.895625	0.296623	0.900719	256	4	Nadam
2	10	0.290606	0.895500	0.280066	0.903188	128	8	Nadam
10	10	0.300019	0.895500	0.288677	0.902344	256	8	Adam
14	10	0.293536	0.894750	0.284947	0.902594	256	5	Nadam
8	10	0.308003	0.893750	0.311851	0.894781	128	8	Adam
4	10	0.315393	0.893375	0.309617	0.895094	128	5	Nadam
7	10	0.304281	0.893000	0.310762	0.895063	256	4	Adam
12	10	0.318829	0.891125	0.326161	0.890063	128	5	Adam
3	10	0.307992	0.891125	0.310003	0.896031	256	5	Adam
15	10	0.321082	0.891000	0.334669	0.886594	64	8	Adam
13	10	0.317294	0.889625	0.322137	0.891688	128	4	Nadam
9	10	0.323350	0.888375	0.320613	0.890000	64	5	Nadam
5	10	0.322759	0.888250	0.328162	0.890250	64	4	Nadam
6	10	0.319459	0.887500	0.325117	0.890719	128	4	Adam
1	10	0.350608	0.886250	0.371882	0.877281	64	4	Adam
0	10	0.336040	0.886125	0.344987	0.884875	64	5	Adam
11	10	0.328732	0.883125	0.297502	0.898406	64	8	Nadam

Figure 10. Talos results table

According to Figure 10, the best combinations of hyperparameters represented with Nadam optimizer, 256 filters and 8 kernel size with validation accuracy 89,90 %. Next, C-LSTM model was trained with selected best hyperparameters, 26 epochs and batch size equal to 128 (Figure 11). Finally, applying the trained model on the test dataset, the accuracy score 91.47% has been achieved.

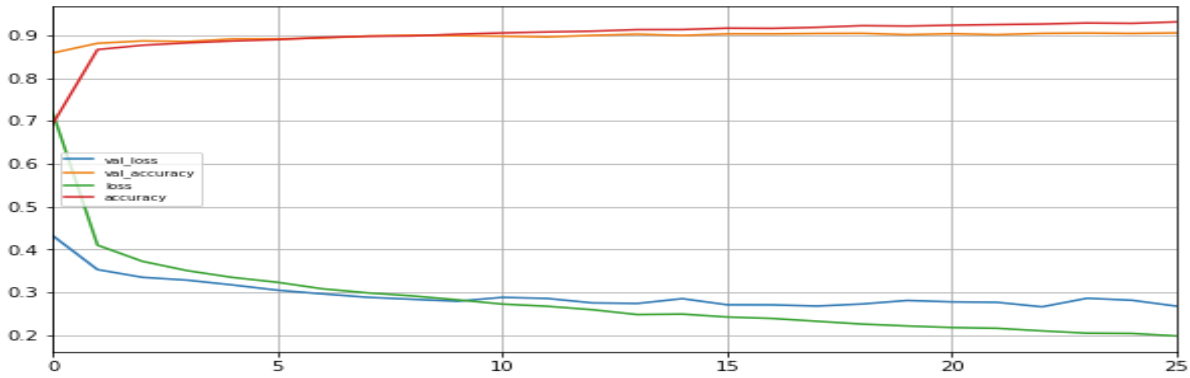

Figure 11. C-LSTM model training results

Results analysis and comparison

Table 1 below compares performance and training time of different word vectorizer techniques trained with three algorithms.

Performance on the test dataset (accuracy score)	Training time	Algorithm	Word Vectorizer
91.27%	507 ms	Linear SVM	TF- IDF
88.70%	8.13 sec	Logistic Regression	Word2vec
91.47%	2316.27 sec	C-LSTM	Word2vec

Table 1. Performance and training time results

According to Table 1, Linear SVM algorithm with TF-IDF vectorizer and C-LSTM neural network model achieved almost equal accuracy score, but with a considerable difference of training time (Linear SVM more than 4K faster to train). Such results were obtained, probably due to the relatively small size of the input data or the data is linearly separable and for accurate classification a simple ML algorithm might be sufficient.

Text generation

The second part of this work was to train a text generation language model, where a model trained on the news article titles of “sports” topic. Uploaded data first cleaned, tokenised and organised into sequences with 51 words each, where first 50 words processed as predictors and the last token as a label. Then obtained 60,601 sequences encoded to integer numbers and labels to one-hot vectors.

After preprocessing steps, input and label vectors were processed through the text generation language model with Bidirectional LSTM with 256 neurons and fully connected Dense layer with 128 neurons (Figure 12). Regular Unidirectional LSTM neural networks generate outputs from past inputs, while Bidirectional LSTM designed for process the input data in both directions and therefore capture information from past and future contexts. (Arisoy et al., 2015, Schmidhuber, 2015, Sak, et al., 2014, Schuster and Paliwal, 1997). Next, Dropout regularisation layer was added to prevent the model from overfitting. The output layer determined with the 9,581 nodes (vocabulary size) and softmax activation function, which produce normalised probability values.

After training with 60 epochs and batch size equal 256, the model achieved an accuracy score of 68.6% on the sports training dataset. Finally, the trained language model with randomly selected input text was applied to generate 100 text samples (Figure 13).

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 50)            479000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               183296
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 9580)              1235820
=================================================================
Total params: 1,931,012
Trainable params: 1,931,012
Non-trainable params: 0
```

Figure 12. Text generation language model architecture

```
# generate new text
generated = generate_seq(model_gen4, tokenizer, seq_length, seed_text, 100)
print(generated)
```

```
show their power with homers against eagles totti rout soccer bc men game in testing gamecocks gunners doping lions
sign australia close through in shaky gold open semifinal fifa khan lb clinches gold chiefs in four oklahoma state
former black signs journey from sven of bomb or nhl miller takes sweep of year as vikings hold their club expos ear
ly better of the capriati innings gets vote to reverse the trophy win colts brees grabs mvp award ap college angels
swap mother joins test at hawaii ap pacers misses a winsa team notebook coolhead would oneyear do lead to talk
```

Figure 13. Generated 100 samples

According to Figure 13, a vast majority of generated text represents a set of distinguishable words (power, soccer), word formation (mvp award, oklahoma state) and named entities (totti, capriati, fifa). Moreover, despite samples contain unrecognisable (lb, winsa) words, the overall meaning of the generated sequences associates with sports. More deep data cleaning, executing named entity recognition, increasing complexity of the model (adding more hidden LSTM layers) and training for more time might probably produce better text.

Testing with ML and DL algorithms

Generated samples were organised to 4 sentences with 25 tokens each and then tested with developed earlier ML and DL models (Table 2).

Performance (accuracy score)	Algorithm	Word Vectorizer
100%	Linear SVM	TF- IDF
100%	Logistic Regression	Word2vec
100%	C-LSTM	

Table 2. ML and DL models performance on the generated samples

As shown in Table 2, both ML and DL algorithms have been performed with the accuracy score of 100%, correctly predicting 4/4 sentences. Such high accuracy is explained by the fact that the language modelling model and ML and DL algorithms were trained on the same training dataset. In addition, such results may be obtained due to small size of the data (100 tokens).

Reference

Autonomio Talos [Computer software] (2019). [online] *GitHub.* Available at: https://github.com/autonomio/talos [Accessed 01 Apr. 2020].

Arisoy, E., Sethy, A., Ramabhadran, B. and Chen, S. (2015). *Bidirectional recurrent neural network language models for automatic speech recognition*.

Kalchbrenner, N., Grefenstette, E. and Blunsom, P. (2014). "A Convolutional Neural Network for Modelling Sentences".

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). "Efficient Estimation of Word Representations in Vector Space"*, arXiv.org,* .

Sak, H., Senior, A.W. and Beaufays, F. (2014). "Long short-term memory recurrent neural network architectures for large scale acoustic modeling".

Schmidhuber, J. (2015). "Deep learning in neural networks: An overview"*, Neural Networks,* Vol. 61, pp. 85-117.

Schuster, M. and Paliwal, K. (1997), *Bidirectional recurrent neural networks*.

Sundermeyer, M., Schlüter, R. and Ney, H. (2012). "LSTM neural networks for language modeling", "LSTM neural networks for language modeling", *Thirteenth annual conference of the international speech communication association*.

Yoon, K. (2014). "Convolutional Neural Networks for Sentence Classification"*, arXiv.org,* .

Zhou, C., Sun, C., Liu, Z. and Lau, F.C.M. (2015), "A C-LSTM Neural Network for Text Classification".