Natural Language Processing and Machine Learning Classification

Hippolyte GISSEROT-BOUKHLEF

In this project, we will prepare text from the RCV1 corpus for text analytics research, and use it for training, evaluation, and selection of supervised machine learning algorithms.

The set of objects that our classifiers need to handle consists of over 800,000 news articles in English that have been conventionally pre-processed: stemming, stop word removal, capitalization, punctuation, etc. Every article has been manually coded and mapped to a hierarchical category of subject topics.

Part I: Model Training

Before starting our analysis regarding top categories, we need to complete several data cleaning tasks on our dataset:

- First, we create a sub-data frame only comprising of the h1 categories (top categories: ECAT, MCAT, GCAT, CCAT) and their id.
- Then, we remove all the NAs and the potential duplicates.

Now that the dataset is clean, we can create the relevant document-term matrix and container. Below are the parameters we chose to use in our analysis:

- Size of train/test sets: We decided to divide the whole dataset into two parts of equal sizes. The model was trained on 50% of the sample, which leaves a sufficient amount of exploitable data and limits the processing time (which would have been significantly longer by choosing an 80/20 split instead).
- Sparse threshold: Removing sparse terms is critical since it allows to drastically shorten processing time by reducing the size of the document-term matrix. 1% appeared to be a good compromise between keeping enough terms to train the model while removing enough to significantly lighten the model.
- Weighting method: "weightTF" is the most intuitive method regarding term weighting. It simply accounts for the frequency of observation of each term in the sample.
- Virgin: The "virgin" parameter is set to FALSE because we are still in the evaluation stage and not yet ready to classify virgin documents. For instance, when the virgin flag is set to FALSE, indicating that all data in the training and testing sets have corresponding labels, create_analytics() will check the results of the learning algorithms against the true values to determine the accuracy of the However, if the virgin flag is set to TRUE, indicating that the testing set is unclassified data with no known true values, create_analytics() will return as much information as possible without comparing each predicted value to its true label.

Let's then train and test the different models.

Support Vector Machine:

A support-vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space, which is commonly used for classification purposes. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called

functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

The vectors defining the hyperplanes can be chosen to be linear combinations with parameters α_i of images of feature vectors x_i that occur in the data base. With this choice of a hyperplane, the points x in the feature space that are mapped into the hyperplane are defined by the relation:

$$\sum_i \alpha_i k(x_i, x) = constant.$$

Note that if $k(x, y)$ becomes small as y grows further away from x, each term in the sum measures the degree of closeness of the test point x to the corresponding data base point x_i. In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated.

Note the fact that the set of points x mapped into any hyperplane can be quite convoluted as a result, allowing much more complex discrimination between sets that are not convex at all in the original space.

Decision Tree:

A decision tree is a simple representation for classifying examples. For this section, assume that all the input features have finite discrete domains, and there is a single target feature called the "classification". Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into either a specific class, or into a particular probability distribution (which, if the decision tree is well-constructed, is skewed towards certain subsets of classes).

A tree is built by splitting the source set, constituting the root node of the tree, into subsets—which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions.

Data comes in records of the form:

$$(x, Y) = (x_1, x_2, \dots, x_k, Y)$$

The dependent variable, Y, is the target variable that we are trying to understand, classify or generalize. The vector x is composed of the features, $x_1, x_2, x_3\ etc.$, that are used for that task.

Random Forests:

Random forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

Decision trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e., have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, x_2, \dots, x_n$ with responses $Y = y_1, y_2, \dots y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples. For $b = 1, \dots, B$, sample, with replacement, n training examples from X, Y; call theseX_b, Y_b, and then train a classification tree f_b on X_b, Y_b. After training, predictions for unseen samples x' can be made by taking the majority vote in the case of classification trees.

Neural Networks:

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Each individual node is composed of input data, weights, a bias (or threshold), and an output. The formula would be of the form:

$$\sum_i w_i x_i + bias$$

$$output = f(x) = \begin{cases} 1 \; if \; \sum_i w_i x_i + bias \geq 0 \\ 0 \; if \; \sum_i w_i x_i + bias < 0 \end{cases}$$

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it "fires" (or activates) the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

Let's now try to assign the articles to the most specific topic levels we can. The ideal model would be able to predict the most specific category, that is the one that has the lower rank in the hierarchy (h5). However, such an approach would raise several issues:

- More specific categories would result in a lowest number of articles belonging to each topic level, which would hinder the training phase of the model (an extreme case would be to observe a different category per article).
- Not all articles are assigned to highly specific categories. Therefore, training the model on too specific categories would reduce the amount of data available.

Sample Sizes for Different Hierarchy Levels

	h2	h3	h4	h5
Sample Size	8243	23638	14330	399
Number of Categories	21	33	42	1

It appears that h3 is a good trade-off between the size of the sample (large enough) and the number of topic levels (not too high to avoid overfitting).

We also have to take care of the following issue: some of the models we are going to use are limited in the number of topic levels they can support (TREE is for instance limited to 32). To solve this, one solution could be to remove low frequency categories (let's say those that represent less than 0.5% of the whole sample).

Let's now use this new dataset to train the four relevant models.

Part II: Model Comparison, Evaluation and Selection

Now that the four models have been trained with the goal to predict the four top categories each article belongs to, let's have a look at how they perform out of sample.

The metrics we used are the following:

- Accuracy, which gives the percentage of right predictions:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- Precision, which is the fraction of relevant instances among the retrieved instances:

$$Precision = \frac{TP}{TP + FP}$$

- Recall, which is the fraction of relevant instances that were retrieved:

$$Recall = \frac{TP}{TP + FN}$$

- F-Score, which is computed as the geometric mean of precision and recall:

$$FScore = \frac{2 * Precision * Recall}{Precision + Recall}$$

- We also had a look at the confusion matrices in order to assess more precisely the precision and recall specific to each of the four categories.

General Summary:

General Performance Summary

	SVM	TREE	FORESTS	NNETWORK
ACCURACY	0.80	0.58	0.80	0.53
PRECISION	0.78	0.46	0.80	0.30
RECALL	0.75	0.48	0.75	0.42
FSCORE	0.76	0.46	0.76	0.33

Out of the four models used, SVM and RF appear to stand out significantly.

e.g.: the SVM model made 80% of good predictions and has a F-Score of 0.76 while the decision tree made only 58% of good predictions and has a F-Score of 0.46.

Now let's take a closer look at the performances of each model specifically.

Support Vector Machine:

SVM Confusion Matrix

	39-ACTUAL	72-ACTUAL	84-ACTUAL	117-ACTUAL
39-PREDICTED	4626	391	519	278
72-PREDICTED	76	875	174	91
84-PREDICTED	389	312	2729	63
117-PREDICTED	279	146	53	2542

SVM Performance Summary

	SVM_PRECISION	SVM_RECALL	SVM_FSCORE
39	0.80	0.86	0.83
72	0.72	0.51	0.60
84	0.78	0.79	0.78
117	0.84	0.85	0.84

Despite the good overall performances of SVM, the model still appears to struggle at accurately predicting category "72", with a recall of 0.51. It means that among all the articles belonging to the "72" category, only 51% were correctly classified.

Decision Tree:

TREE Confusion Matrix

	39-ACTUAL	72-ACTUAL	84-ACTUAL	117-ACTUAL
39-PREDICTED	4379	1044	1426	1088
72-PREDICTED	0	0	0	0
84-PREDICTED	271	226	1718	74
117-PREDICTED	720	454	331	1812

TREE Performance Summary

	TREE_PRECISION	TREE_RECALL	TREE_FSCORE
39	0.55	0.82	0.66
72	NA	0.00	NA
84	0.75	0.49	0.59
117	0.55	0.61	0.58

The overall performances of the decision tree are much weaker than those of the SVM model. The algorithm did not even manage to assign the category "72" to any of the articles in the test sample.

Random Forests:

RF Confusion Matrix

	39-ACTUAL	72-ACTUAL	84-ACTUAL	117-ACTUAL
39-PREDICTED	4638	407	448	267
72-PREDICTED	64	770	98	52
84-PREDICTED	413	385	2871	65
117-PREDICTED	255	162	58	2590

RF Performance Summary

	FORESTS_PRECISION	FORESTS_RECALL	FORESTS_FSCORE
39	0.81	0.86	0.83
72	0.78	0.45	0.57
84	0.77	0.83	0.80
117	0.85	0.87	0.86

Random forests (sort of upgraded version of decision trees) unsurprisingly performed better than the decision tree and can be compared to the SVM model. Let's note that even though the precision and recall for categories "39", "84" and "117" are slightly better, SVM significantly outperforms random forests for category "72", which makes it difficult to choose between the two.

Neural Networks:

NNET Confusion Matrix

	39-ACTUAL	72-ACTUAL	84-ACTUAL	117-ACTUAL
39-PREDICTED	4059	171	345	175
72-PREDICTED	0	0	0	0
84-PREDICTED	1311	1553	3130	2799
117-PREDICTED	0	0	0	0

NNET Performance Summary

	NNETWORK_PRECISION	NNETWORK_RECALL	NNETWORK_FSCORE
39	0.85	0.76	0.80
72	NA	0.00	NA
84	0.36	0.90	0.51
117	NA	0.00	NA

Neural networks performed even more poorly than the decision tree since the model was not only unable to predict category "72", but also failed to predict category "117". This unsurprisingly resulted in low overall performance metrics (see general summary above).

Selection:

Regarding the different metrics, SVM and RF appear to be the most relevant models to use for this classification task. However, even though RF seems to perform slightly better overall, it could be wise to wait for the results of the second classification task to see if the trend is confirmed.

Let's now repeat the same selection process for the second classification task.

General Performance Summary

	SVM	TREE	FORESTS	NNETWORK
ACCURACY	0.65	0.34	0.60	0.27
PRECISION	0.56	0.08	0.56	0.04
RECALL	0.52	0.13	0.43	0.08
FSCORE	0.52	0.09	0.45	0.05

Since we increased the number of predictable topic levels and therefore increased the models' complexity, we logically observe a general drop in the performance metrics. Nevertheless, the raking remains unchanged, with SVM and random forests still significantly outperforming neural networks

and decision tree. We can however observe that this time, SVM performed better than random forests on 3 out of the 4 relevant metrics (accuracy, recall and F-score).

Let's therefore use SVM for further tuning and classification. There is a good chance that this model will still outperform the others (and specifically decision tree and neural networks) in similar situations (articles of the same nature, same methods for text cleaning, use of word frequency for text interpretation etc.). However, changing these parameters may result in different outcomes.

Note: For the sake of clarity, we have chosen not to display the confusion matrix and the specific statistics for each category, given the high number of topic levels.

Part III: Model Tuning

Now that we have selected the best performing model, it may be interesting to optimize its main parameters. The methodology we implemented is the following:

- First, we considered two key parameters regarding data preparation:
  - We made the sparse threshold take the values 0.01 (default value), 0.05 and 0.1 and then chose the best option regarding the performances on the test sample.
  - We then did the same with the weighting option: Tf (default value) and Tf-Idf.
- Secondly, we selected two of the main parameters of the model and varied them:
  - We tested the SVM model using linear $(x^T y)$, radial $(e^{\gamma|x-y|^2},\ \gamma = \frac{1}{data\ dimension})$ and polynomial $((\gamma x^T y + C_0)^d,\ \gamma = \frac{1}{data\ dimension},\ C_0 = 0,\ d = 3)$ kernels.
  - We finally chose to vary the cost option (the cost of constraints violation, which corresponds to the C-constant of the regularization term in the Lagrange formulation) and test the model on the following values: 0.1, 1 (default value), 10, 100 and 1000.

In the end, we came up with an optimized model on these four parameters.

Sparse Threshold:

SVM Performance Depending on Sparse Threshold

	ACCURACY	PRECISION	RECALL	FSCORE
ST = 0.01	0.80	0.78	0.75	0.76
ST = 0.05	0.78	0.77	0.74	0.74
ST = 0.1	0.75	0.74	0.69	0.70

Selected option: Sparse Threshold = 0.01.

Weighting:

SVM Performance Depending on Weighting Method

	ACCURACY	PRECISION	RECALL	FSCORE
Weighting = weightTf	0.80	0.78	0.75	0.76
Weighting = weightTfIdf	0.73	0.70	0.68	0.69

Selected option: Weighting = Tf.

Kernel:

Performance Depending on Kernel

	ACCURACY	PRECISION	RECALL	FSCORE
Kernel = linear	0.76	0.75	0.70	0.72
Kernel = radial	0.80	0.78	0.75	0.76
Kernel = polynomial	0.58	0.69	0.47	0.50

Selected option: Kernel = radial.

Cost:

Performance Depending on Cost

	ACCURACY	PRECISION	RECALL	FSCORE
Cost = 0.1	0.74	0.72	0.69	0.70
Cost = 1	0.80	0.78	0.75	0.76
Cost = 10	0.80	0.80	0.76	0.78
Cost = 100	0.80	0.78	0.75	0.76
Cost = 1000	0.79	0.78	0.74	0.76

Selected option: Cost = 10.

Default Model vs. Tuned Model:

See below the comparison between the default and tuned model's performances for the first classification task, regarding accuracy, precision, recall and F-score.

First Classification Task: Default Model vs. Tuned Model

	ACCURACY	PRECISION	RECALL	FSCORE
DEFAULT	0.8	0.78	0.75	0.76
TUNED	0.8	0.80	0.76	0.78

As we can see in the table above, we managed to slightly increase the performance of our model for the first classification task.

Sparse Threshold:

SVM Performance Depending on Sparse Threshold

	ACCURACY	PRECISION	RECALL	FSCORE
ST = 0.01	0.65	0.56	0.52	0.52
ST = 0.05	0.61	0.53	0.46	0.47
ST = 0.1	0.55	0.44	0.37	0.38

Selected option: Sparse Threshold = 0.01.

Weighting:

SVM Performance Depending on Weighting Method

	ACCURACY	PRECISION	RECALL	FSCORE
Weighting = weightTf	0.80	0.78	0.75	0.76
Weighting = weightTfIdf	0.73	0.70	0.68	0.69

Selected option: Weighting = Tf.

Kernel:

SVM Performance Depending on Kernel

	ACCURACY	PRECISION	RECALL	FSCORE
Kernel = linear	0.64	0.55	0.51	0.51
Kernel = radial	0.65	0.56	0.52	0.52
Kernel = polynomial	0.36	0.48	0.18	0.21

Selected option: Kernel = radial.

Cost:

SVM Performance Depending on Cost

	ACCURACY	PRECISION	RECALL	FSCORE
Cost = 0.1	0.58	0.47	0.47	0.45
Cost = 1	0.65	0.56	0.52	0.52
Cost = 10	0.66	0.57	0.53	0.53
Cost = 100	0.65	0.56	0.52	0.52
Cost = 1000	0.65	0.56	0.52	0.52

Selected option: Cost = 10.

Default Model vs. Tuned Model:

See below the comparison between the default and tuned model's performances for the first classification task, regarding accuracy, precision, recall and F-score.

Second Classification Task: Default Model vs. Tuned Model

	ACCURACY	PRECISION	RECALL	FSCORE
DEFAULT	0.65	0.56	0.52	0.52
TUNED	0.66	0.57	0.53	0.53

For the second classification task as well, we managed to slightly increase the performance of our model (see summary table above). As we can see in the table above, we managed to slightly increase the performance of our model for the first classification task.

Parameters Summary

Model Parameters Summary

	4-Category Classification Task	Specific-Category Classification Task
Algorithm	SVM	SVM
Sparse Threshold	0.01	0.01
Weighting	Tf	Tf
Kernel	Radial	Radial
Cost	10	10

There is a good chance that these models will perform approximately the same in similar situations (articles of the same nature, similar topics, same methods for text cleaning, use of word frequency for text interpretation etc.). However, radically changing the nature of the input data may produce significantly different results.